MSc-IT

GAUHATI UNIVERSITY
Institute of Distance and Open Learning

ADVANCED COMPUTER ORGANIZATION AND ARCHITECTURE

Semester- I

MSc-IT
Paper: INF 1026

Advanced Computer Organization and Architecture

www.idolgu.in

M.Sc.-IT-19-I-1026

GAUHATI UNIVERSITY
Institute of Distance and Open Learning

Semester- I

M.Sc.IT

Paper: INF 1026
ADVANCED COMPUTER ORGANIZATION AND ARCHITECTURE

CONTENTS:

Contributors:

Mr. Kalyanbrat Medhi (Block I : Unit- 1)
Faculty, Dept. of Computer Science
Bhattadev University, Bajali, Assam
Dr. Manash Protim Bhuyan (Block I : Unit- 2)
Asstt. Prof., Dept. of Computer Science and Engineering
Golaghat Engineering College, Golaghat, Assam
Mrs. Manjula Kalita (Block I : Unit- 3)
Asstt. Prof., Dept. of Computer Science and Engineering
GIMT, Guwahati, Assam
Mr. Rahul Lahkar (Block I : Unit- 4)
Asstt. Prof., Dept. of Computer Science
Pub Kamrup College, Assam
Mr. Dipankar Dutta (Block II : Units- 1 & 2)
Asstt. Prof., Dept. of Computer Science
NERIM, Guwahati, Assam
Dr. Pranab Das (Block II : Unit- 3, Block III: Unit 1)
Asstt. Prof.(Sr.), Dept. of Computer Applications
Assam Don Bosco University, Guwahati, Assam
Mrs. Manasi Hazarika (Block II : Units- 4 & 5)
Asstt. Prof.(Sr.), Dept. of of Computer Science and Engineering
Assam Don Bosco University, Guwahati, Assam
Dr. Kshirod Sarmah (Block III : Unit- 2)
Asstt. Prof., Dept. of Computer Science
PDUAM, Bajali, Assam
Mr. Deepjyoti Saikia (Block III : Unit- 3)
Asstt. Prof., Dept. of Computer Science
Mangaldai College, Darrang, Assam
Mrs. Epsita Medhi (Block III : Unit- 4)
Research Assistant, Dept. of Information Technology
Gauhati University, Assam
Mr. Subhomoy Dey (Block III : Unit- 5)
Asstt. Prof., Dept. of Computer Science
PDUAM, Goalpara, Assam

Content Editor:

Prof. Kandarpa Kumar Sarma
Dept. of Electronics and Communication Engineering,
Gauhati University

Course Coordination:

Prof. Dandadhar Sarma Director, IDOL, Gauhati University
Prof. Anjana Kakoti Mahanta Prof., Dept. Computer Science, G.U.

Cover Page Designing:

Bhaskar Jyoti Goswami IDOL, Gauhati University

May, 2022

Published on behalf of Institute of Distance and Open Learning, Gauhati University by the Director, and printed at Gauhati University Press, Guwahati-781014.

BLOCK I:

INSTRUCTION SET ARCHITECTURE AND PROCESSOR DESIGN

UNIT 1: INSTRUCTION SET DESIGN AND ARCHITECTURE

Unit Structure:

Space for learners:

Space for learners:

1.1 INTRODUCTION

In this unit, we will discuss addressing types, addressing modes and representation of characters. The organization of computer processor as well various registers is explained in brief. Here, machine languages program using different addressing type is elaborated. We will also know about the instruction cycle. At the end of the chapter integer, fixed point representation, floating point representation and character representation inside computer are discussed.

1.2 UNIT OBJECTIVES

The objective of the unit is:

- To know the addressing type
- To know the addressing mode
- Overview of processor
- Overview of registers
- To know about instruction cycle
- Data representation in computer

1. 3 INSTRUCTION SET DESIGN

An instruction set is collection of machine language or assembly language instructions that are understood by central processing unit (CPU). The following issues are considered in instruction set design:

- Whether operands are to be stored in registers, memory, stack or accumulator

- How many operands are present in instructions 0, 1, 2, or 3
- Whether access mode of operand are register, immediate, indirect and so on.
- What are the operations that are supported in instruction add, sub, mul etc.

Space for learners:

1.3.1 How many addresses

Let us assume the statement in a high level programming language given bellow

a = a + b + a * c

It is clear that the value of *a* multiply with *c* is added with *a, b* and the final result is stored in the variable *a*. You know the precedence and associativity rules of high level languages. However, you cannot expect the computer hardware to directly understand these rules.

It requires operations to be performed in small steps. The desired result will be produced after going through **sequence of simple steps**. Hence, it eliminates the necessity for the machine to understand about these rules. In most of the cases operands name i.e. **address** is used rather than value. The machine may be following types depending on addresses:

- 3-address machines
- 2-address machines
- 1-address machines
- 0-address machines

Here number 0, 1, 2, 3 indicates maximum number of address/operand the machine can have.

Here we will use the convention that 'first operand is destination' in an instruction. This means we will consider that the result of operation will be stored in first operand of the instruction.

STEP TO CONSIDER

The address may be either memory or computer registers. In a particular machine final result of operation may be stored in first, or last operand. Here, we consider that the first operand will hold the result of the operation.

1.3.1.1 *3-address machines*

Space for learners:

The general format of a 3-address machine instruction is:

operation dst, op1, op2

Here, *operation* indicates opcode of the operation to be performed, the first operand *dst* represent destination operand i.e. where the result of operation will be stored, op1 and op2 indicates two source operands between which operation is to be performed. Thus the following instruction means:

ADD R2, R1, R0

Add the values stored in register *R1* and *R0*, and store result in the register *R2*.

When all operands of instructions are only in register then we call it a **register-register** machine or a **load-store** machine. Instead of that if all operands of instructions are only in memory then we call it a **memory-memory** machine. The following is such an example:

ADD X, Y, Z

Add the value of variable *y* to the value of variable *z* and then store the result in the memory location *x*. In a memory-memory machine the CPU has to get the operands from memory prior to execution of the operation. After that it has to store the result back in memory. There are several ways to specify the address of an operand. We will discuss this topic in addressing mode section.

Let us now see how to implement a3-address instruction for the statement

$a = a + b + a * c$

Answer:

```
MUL R4, a, c          # store a*c in R4
ADD R1, a, b          # store a + b in R1
ADD R1, R1, R4        # Store result in R1
```

The final result of the expression can be found in register R1.

1.3.1.2 2-address machines

The general format of 2-address machine instruction is:

operation dst, op

where, *operation* is opcode of the operation, *dst* represent the source operand as well as destination, *op* represent the second source operand. Let us see the following instruction

ADD R1, R2

The meaning of this instruction is to add the values stored in registers R1 and R2, and then store the result back in register R1.

The advantage of 2-address instructions over three-address instructions is that it helps in preserving memory, since they are shorter. Moreover shorter instructions take less time for fetching. The drawback having two-address instructions is that one of the source operands is destroyed. It requires extra moves *to retain the operand* as sometimes operand may be needed later.

Let us now see how to implement a 2-address instruction for the statement

$a = a + b + a * c$

Answer:

MUL c, a	# multiply *a, b* and store in *c*
MOV R1,c	# move content of c to R1
ADD b,a	# add a, b and store in b
MOV R2,b	# move content of b to R2
ADD R1, R2	#add R1, R2 and store in R1

The final result of the expression can be found in register R1.

1.3.1.3 1-address machines

In a 1-address machine accumulator has a source operand and result of operation is put back implicitly in the accumulator. The instruction needs to indicate the second source operand. The format of a 1-address instruction is as follows:

Space for learners:

operation op

The opcode '*operation*' is the name of the operation to be done, *op* indicates either source or destination operand. Here the instruction:

ADD a

It means addition of value of variable *a* with the content of accumulator. The result of addition is put in the accumulator. The accumulator is a special purpose register.

Let us now see how to implement a 2-address instruction for the statement.

$a = a + b + a * c$

Answer:

```
LOAD a          # load content of a in accumulator
MUL c           # multiply accumulator i.e. a and c
ADD b           # add b to previous contents of the
                    accumulator i.e. a * c + b
ADD a           # a * c + b + a
STO a           # store the final result in location a
```

The final result of the expression can be found in the memory location a.

STOP TO CONSIDER

As the number of address reduced the number of instruction increases to do the same task.

1.3.1.4 Zero-address machines

The zero-address machines are implemented using stack. A stack is last in first out (LIFO) data structure that is operated by using PUSH and POP. PUSH moves an operand from computer memory into top of stack, on the other hand POP gets out the last item from top of the stack. Only PUSH and POP indicates an operand. No other opcode specify any operand. This is the reason why it is called a zero address machine. The question is how then operands are handled by the machine for the operation. It is done by

Space for learners:

extraction top two elements of stack and putting the result back into stack.

Let us see how to implement a zero-address instruction for the statement

*a = a + b + a * c*

Answer:

```
PUSH a          # push the value of a
PUSH c          # push the value of c
MUL             # multiply top two value a * c
PUSH b          # push the value of b;
ADD             # add top two value b + a * c
PUSH a          # push the value of a
ADD             # add top two value a + b + a * c
POP  a          # store in top of stack in a
```

The final result of the expression can be found in the memory location a.

Space for learners:

1.3.2 Types of Instructions

The computer supports the following types of instructions:

- Data Transfer Instructions
- Arithmetic Instructions
- Bit Manipulation Instructions
- Program Execution Transfer Instructions
- Processor Control Instructions
- Iteration Control Instructions
- Interrupt Instructions

1.3.2.1 Data Transfer Instructions

These instructions transfer data from the source to the destination location inside the computer. The common data transfers are among registers or between registers and memory or between the register (s) and the input/output devices. Different computer uses various mnemonics for the same instruction. The following are some of the data transfer mnemonics with their meaning.

- **MOV**: Transfer data from register to resister or resister to memory.
- **ST:** Store from register (accumulator) to memory
- **LD:** Load data from memory to register
- **PUSH:** Transfer data from CPU register to top of the stack.
- **POP:** Transfer data from top of stack to CPU register
- **XCHG:** Exchange data between two given locations.
- **IN:** Read data from an input port to accumulator.
- **OUT:** Transfer data from accumulator to particular output port.

Space for learners:

1.3.2.2 Arithmetic Instructions

The basic arithmetic operations are addition, subtraction, multiplication and division between two numbers. These arithmetic operations are performed between two operands. Some of the arithmetic operations may be performed on a single operand too.

Following a few arithmetic instructions:

- **ADD**: Add the contents of two source locations.
- **MUL**: Multiply the contents of two source locations.
- **DIV**: Divide content of one source locations with the other.
- **SUB**: Subtract content of one source locations from the other.
- **ADC**: Add the contents of two source locations with carry.
- **INC**: Increment the content of source location by 1.

1.3.2.3 Bit Manipulation Instructions

These instructions manipulates data in bit level i.e. operations like shift or logical. Below is a few instructions of this group with meaning are given:

- **NOT**: This invert each bit of source bit pattern.

- **AND**: Logical AND operation between each corresponding bit of both source operand.
- **OR:** Logical OR operation between each corresponding bit of both source operand.
- **XOR** – Perform logical Exclusive-OR operation between each corresponding bit of both source operand.
- **SHL:** Perform bits shift towards left and fill zero in LSBs.
- **SHR**: Perform bits shift towards left and fill zero in MSBs.

Space for learners:

1.3.2.4 Program Execution Transfer Instructions

These instructions transfer the control during an execution of instructions. The transfer of control during execution of instruction may be conditional or unconditional. A few such examples are listed below:

- **CALL**: It calls a subprogram and saves the return address on top stack.
- **RET:** Returns from subprogram/function to the main program.
- **JMP:** Jumps to the given address and process the next instruction.
- **JC:** Jumps when value of carry flag is 1
- **JNC:** Jumps when value of carry flag is 0

1.3.2.5 Processor Control Instructions

These instructions set or reset the flag values and thus control the actions of the processor. Following are the instructions under this group:

- **STC:** Set the carry flag (CF) to 1
- **CLC:** Reset the carry flag i.e. CF = 0
- **CMC:** Complement state of carry flag.
- **STI:** Set the interrupt flag to 1.
- **CLI:** Reset the interrupt flag to 0.

1.3.2.6 Iteration Control Instructions

These instructions can execute a group of instructions repeatedly. A few list of iteration control instructions are:

- **LOOP**: Execute a group of instructions repeatedly until the condition is true.
- **JCXZ:** Jump to a given address if CX = 0
-

1.3.2.7 Interrupt Instructions

These instructions call an interrupt during execution of instructions.

- **INT:** Interrupt the process and call service routine.
- **INTO:** Interrupt the process if OF = 1
- **IRET:** Return to main program from interrupt service.

Check Your Progress-1

1. When all operands of instructions are only in register then we call it a _____machine**.**

2. If all operands of instructions are only in memory then we call it a _____ machine.

3. The drawback having two-address instructions is that one of the source operands is _____.

4. In a one-address machine the result of operation is put back implicitly in the _____.

5. The zero-address machines are implemented using _____.

State TRUE or FALSE:

6.The processor has three types of organization.

7. The advantage of two-address instructions over three-address instructions is that it helps in preserving memory.

8. The accumulator is a special purpose register.

9. MOV is control transfer instruction.

10. POP insert an operand from computer memory into top of stack.

Space for learners:

1.4 ADDRESSING MODES

In a typical instruction, we see the address fields are relatively small. The purpose of addressing mode is to reference main memory locations as large as possible. This is the reason why a variety of addressing modes have been implemented. The most commonly used addressing modes are:

- Immediate
- Direct
- Indirect
- Register
- Register indirect
- Displacement
- Stack

Mode	Algorithm	Advantage	Disadvantage
Immediate	Operand=A	No memory reference	Limited operand magnitude
Direct	EA=A	Simple	Limited address space
Indirect	EA=(A)	Large address place	Multiple memory reference
Register	EA=R	No memory reference	Limited address space
Register indirect	EA=(R)	Large address place	Extra memory reference
Displacement	EA=A+(R)	Flexibility	Complexity
Displacement	EA= top of stack	No memory reference	Limited applicability

Table 1.1Basic Addressing Modes

The Table 1.1depicts the address calculation procedure for each addressing mode. Each of the addressing modes will be represented with different opcodes. The opcode may be one or more bits in the instruction format.

STOP TO CONSIDER

The effective address of operand is calculated after decoding the opcode.

Space for learners:

1.4.1 Immediate Addressing

The immediate addressing holds the operand value in the instruction.

Operand = A

This addressing mode is generally used to set initial values of variables or constants. The primary advantage is that there is no need of memory reference. Thus it saves one memory or cache cycle in the instruction cycle. The disadvantage of immediate addressing mode is that size of the number is limited to size of the address field.

1.4.2 Direct Addressing

In direct addressing mode the address field holds the effective address of the operand:

EA = A

The advantage of direct addressing mode is that it needs only one memory reference. The disadvantage this addressing mode is limited address space accessibility.

1.4.3 Indirect Addressing

In direct addressing mode usually length of the address field is less than word length. It causes limitation in address range. If the address field refers to address of a word in memory, it can access a full-length address of the operand. This way of accessing memory word is known as indirect addressing. In indirect addressing mode the address field contains address of another memory location where the value of actual operand remains.

EA = (A)

The parenthesis interpreted as contents of '*A*' is another address. The disadvantage of indirect addressing is that it requires two

Space for learners:

memory references to fetch actual operand value, first to get its address and next to get its value.

Space for learners:

1.4.4 Register Addressing

The register addressing mode has similarity to direct addressing. The difference here is that address field indicates a register instead of main memory address:

EA = R

The register *R* specifies the address where the operand value contains. The advantages of this mode are that a small address field is needed and no memory references needed means less time required for fetching instruction. The disadvantage of this mode is that the available address space is limited to registers only.

1.4.5 Register Indirect Addressing

The register indirect addressing mode is similar to indirect addressing mode. The only difference is that address field refers to a register instead of memory location. Let us see the register indirect address.

EA = (R)

The advantages and disadvantages of register indirect addressing mode are similar to indirect addressing mode. But, register indirect addressing mode has one more advantage since it uses one less memory reference it save one cycle time when it is executed.

1.4.6 Displacement Addressing

The displacement addressing mode combines the direct addressing with register indirect addressing. The effective address in this mode looks like as:

EA = A + (R)

This addressing mode the instruction contains two address fields, out of which at least one of it is explicit. The value stored in one of the addresses field (i.e. *A*) is used directly. The contents of second address field i.e. register is added to *A* to obtain the effective address. We will discuss three most commonly used displacement addressing:

- Relative addressing
- Base-register addressing
- Indexing

RELATIVE ADDRESSING: The relative addressing is also known as PC-relative addressing. In this mode of addressing the register that implicitly referenced is program counter (PC). As we know PC contains the address next instruction to be executed. Hence, it is added to the address field in order to produce the *EA*. This is how the effective address in this addressing mode is a displacement relative to the address of the instruction.

EA= PC + address field value

BASE-REGISTER ADDRESSING: In the base-register addressing mode, the referenced register contains a main memory address. The address field indicates a displacement from that address, which is usually an unsigned integer.

EA=base register + address field value

INDEXING: In this addressing mode, the effective address of the operand is calculated by adding content of index register with address field value.

EA= IR+ address field value

The indexing mechanism is extensively used for implementing iterative operations. Suppose a list of numbers present in memory location starting from A and we want to add 1 to each number on that list. Here, we have to fetch each number and after adding 1 to it, store it back to that location. The effective addresses that requires are *A, A + 1, A + 2, . . .,* and so on to last location. It can be done easily with indexing. The value of A is

Space for learners:

stored in the instruction's address field value, and the index register is initialized to 0. At the end of each operation, index register is incremented by 1.

1.4.7 Stack Addressing

The stack addressing is also referred to as a *last-in-first-out* or *queue pushdown list.* In this addressing mode items are placed to the top of the stack so that. Hence, the stack is partially filled at any given time.

The stack is associated with a pointer called stack pointer (SP) whose value refers to the top address of the stack. If the top two item of the stack is in processor registers, the SP references the third item of the stack. The stack pointer is a dedicated special purpose register. It is a form of implied addressing. The instructions do not require a memory reference; it always implicitly indicates the top of the stack.

Check Your Progress-2

11. The purpose of addressing mode is to reference _____ as large as possible**.**

12. The immediate addressing mode generally used to set initial values of _____ or _____.

13. In direct addressing mode address field holds _____address of the operand.

14. In indirect addressing mode the address field contains _____ of another memory location.

15. The displacement addressing mode combines the direct addressing with _____ addressing.

State TRUE or FALSE:

16.The advantage of direct addressing mode is that it needs only one memory reference.

17. In register addressing mode one memory references needed.

18. The stack is associated with a pointer called stack pointer.

19. Effective address is calculated after decoding an instruction.

20. In stack addressing two memory references needed.

Space for learners:

1.5 PROCESSOR ORGANISATION

Space for learners:

The computer processor needs to do the following things to execute an instruction:

• **Fetch instruction:** The processor *has* to read instructions from memory i.e. from register, cache or main memory.

• **Interpret instruction:** After reading an instruction it is decoded to know what action to be performed.

• **Fetch data:** During the execution of an instruction it may need to read data from computer memory or input/output (I/O) module.

• **Process data:** In execution time of an instruction, it may have to perform either arithmetic or logical operation on data.

• **Write data:** At the end of an instruction execution, the results may need to write data to main memory or an I/O module. In order to do these, it clears that the processors sometimes have to store intermediate data. Hence, the processor requires a small internal memory.

Figure 1.1 is a block diagram of a processor depicting its connection to the rest of the system through system bus. The vital components of the central processing unit are

- Arithmetic and logic unit (ALU)
- Control unit (CU).
- Registers

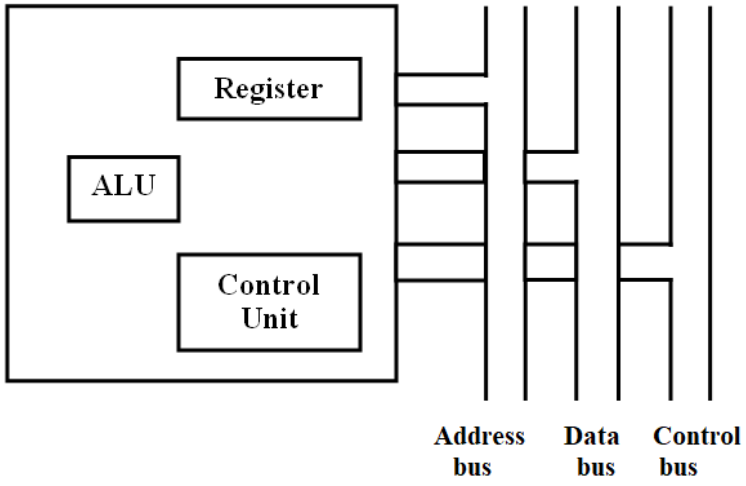


Figure1.1: The block diagram of CPU

The ALU performs the actual processing of data. The CU controls the data and instructions movement in the processor. It also controls the operations of the ALU. The figure also depicts internal memory of processor, called registers.

In general, CPU or processor organization has three categories depending on the number of address fields:

- Single Accumulator organization
- General register organization
- Stack organization

In accumulator based organization, a special purpose register called accumulator is used for performing the operations. In general, register organization involves different registers in the computation tasks. In the stack organization the calculations performed on top of the stack. The instruction of stack organization does not contain any address field. In general, a combination of different organizations is mostly used.

Space for learners:

1.6 REGISTER ORGANISATION

The computer system consists of memory in different level called hierarchy. At top levels of the hierarchy means memory is faster than the bellow level. In this level it is smaller as well as more expensive. The register inside the processor is top level memory followed by cache memory and main memory respectively. The registers have two categories:

• User-visible registers
• Control and status registers

STOP TO CONSIDER

The address bus, data bus and control bus are together called system bus. Operand address bits can travel through address bus, data bits travel trough data bus and CPU generated signal travel through control bus. The processor interaction with main memory is done through these buses.

1.6.1 User-Visible Registers

Space for learners:

The user-visible registers are used by assembly language programmer in order to minimize main memory references. It can be in the following types:

- General purpose register
- Data
- Address
- Condition codes

General-purpose registers are used to store temporary data during execution of instruction. For a given *opcode* the general-purpose register can holds the operand. This is true use of general purpose registers. The general-purpose registers sometimes can be used for addressing purpose (e.g., register indirect, displacement).

Data registers can be used to hold data only. It cannot be used for calculating of operand address.

Address registers may either general purpose or devoted to an individual addressing mode. The following are examples of it:

- **Segment pointers:** The segment register is used to hold the address of the base of the segment.

- **Index registers:** These registers are used for auto indexing in indexed addressing.

- **Stack pointer:** In stack addressing a dedicated register is used called stack pointer.

Condition codes (*flags*): These are bits set by the processor depending on result of an operation. As we know, result of arithmetic operation may be positive, negative, zero, or overflow. In this case a condition code *(flag)* is set and result is stored in memory or register. Subsequently the code may be tested during execution of conditional branch operation.

1.6.2 Control and Status Registers

Space for learners:

The operations of processor are controlled by variety of internal registers. In general, these registers are not visible to programmer or user. Here, we will discuss four such essential registers.

• **Program counter (PC):** It holds the address of the next instruction to be executed.

• **Instruction register (IR):** It holds the address of currently executed instruction.

• **Memory address registers (MAR):** It holds the address of an instruction to be fetched.

• **Memory buffer registers (MBR):** Holds data that needs the current instruction or result produced by the instruction. Another of register that includes in a processor is called the *program status word* (PSW). It contains condition code and other status information. The followings are status flags:
• **Sign:** It holds sign bit of the recent arithmetic operation.
• **Zero:** It is set when the result of operation is 0.
• **Carry:** It is set if addition operation produce a carry or borrow (for subtraction) from lower order bit.

• **Equal:** Set if a logical comparison of two operands is equal.

• **Overflow:** When arithmetic operation produces overflow it is set.

• **Interrupt Enable/Disable:** This flag is used to enable or disable the interrupts.

• **Supervisor:** It indicates the execution mode of processor (supervisor or user). Some of the privileged instructions are executed only in supervisor mode. Similarly, certain memory location can be accessed through supervisor mode only.

1.7 INSTRUCTION CYCLE

An instruction cycle goes through the following stages:
•**Fetch:** The processor reads the next instruction from PC

•**Execute:** Decode the *opcode* and perform the required operation.
• **Interrupt:** If interrupt occurs, pause the current process, save status of it and go to the interrupt.

Before elaborating instruction cycle it's important to know one additional stage called indirect cycle.

Check Your Progress-3

21. The _____ performs the actual processing of data.

22. The CPU organization has _____ categories.

23. The computer system consists of memory in different level called _____.

24. _____registers are used to store temporary data during execution of instruction.

25. The execution mode of processor either _____or _____.

State TRUE or FALSE:

26.The CU controls the data and instructions movement in the processor.

27. The segment register is used to hold the address of the base of the segment.

28. PC holds the address of current instruction executing.

29. MBR holds the address of an instruction to be fetched.

30. Carry flag is set if addition operation produce a carry.

1.7.1 The Indirect Cycle

During instruction execution it may have one or more operands that need memory access. In case of indirect addressing additional memory accesses are needed. The Figure 1.2 depicts instruction cycle.

After fetching the instruction it is checked to see if it involves any indirect addressing. If indirect addressing involves the operands are fetched according to indirect addressing. After execution, an interrupt will process if occurs before fetching the next instruction.

Space for learners:

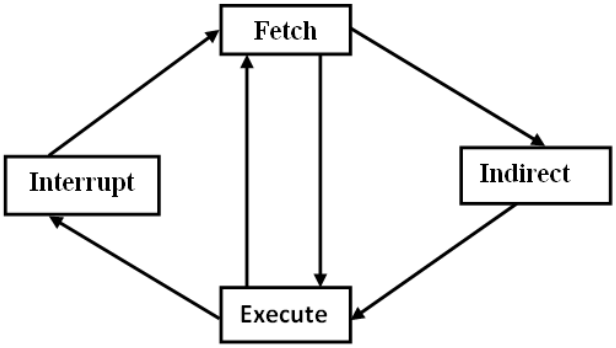


Figure 1.2 Instruction Cycle

After fetching the instruction the operand are fetched from memory. If the operand are in register then fetching is not required. Once execution of instruction is completed the result may be needed to store in main memory.

1.7.2 Data Flow

In an instruction cycle sequence of events occurs according to the design of processor. Suppose, a processor consist of a program counter (PC), a memory address register (MAR), a memory buffer register (MBR), and an instruction register (IR).

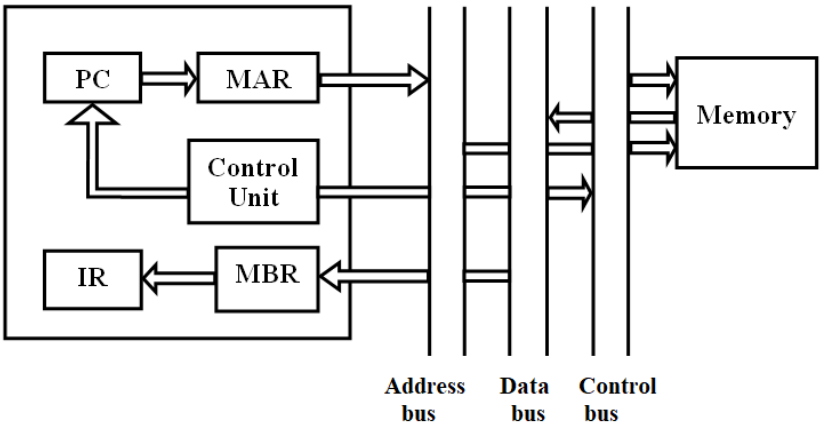


Figure 1.3 Data Flow, Fetch cycle

Figure 1.3 depicts the data flow during fetch cycle. The PC holds the address of the next instruction to be fetched. This address is placed on the address bus through the MAR.

Space for learners:

The CU requests a main memory read to fetch the required for the instruction. The requested result is placed on the data bus and goes to the MBR and finally reached the IR. In the mean time, the PC is incremented for fetching the next instruction. At the end of fetch cycle, the CU checks the IR to know whether it's holding an operand specifier using indirect addressing. If indirect addressing is found, indirect cycle is performed after fetch cycle. Figure 1.4 depicts this simple cycle. The address reference bits of the MBR are transferred to the MAR. After that the CU place a memory read request. Then desired address of the operand is placed in MBR through address bus.

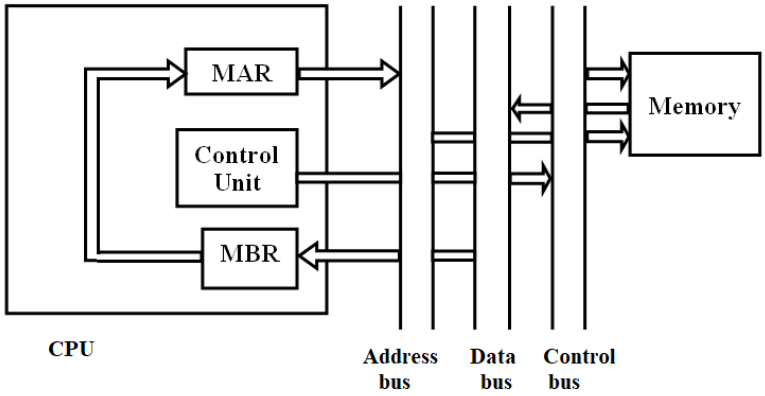


Figure 1.4 Data Flow, Indirect cycle

The fetch and indirect cycles are very simple. The *execute cycle* may have various stages. It many involve ALU operation, register transfer of data, read/write operation from memory or I/O. On the other hand the *interrupt cycle* is as simple as fetch and indirect cycle. It is depicted in figure 1.5. Before going to interrupt current status of the PC must be in order to resume normal activity after the interrupt. So, the contents of the PC is transferred to the MBR and written to memory. For this purpose special memory location is reserved and it is loaded into the MAR from the CU. The special memory may be a stack pointer. The PC is filled with the address of interrupt routine. Henceforth, the next instruction cycle will fetch the desired instruction.

Space for learners:

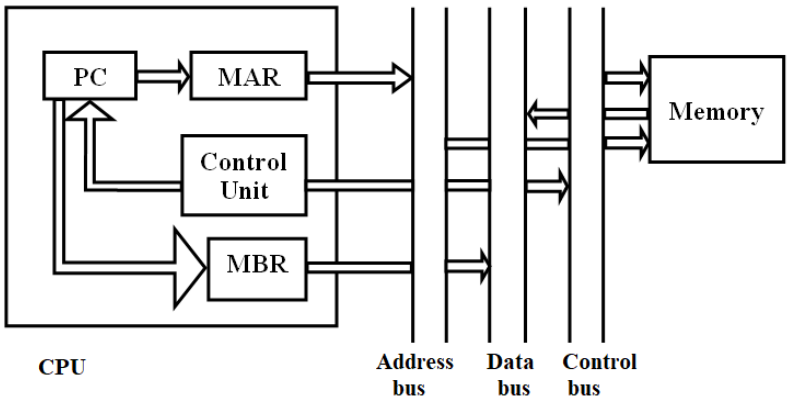


Figure 1.5 Data Flow, Interrupt Cycle

STOP TO CONSIDER

The instruction cycle has different stages fetching, decoding opcode, effective address calculation, execution of operation on data and writing data in memory that are executed in sequence.

1.8 DATA REPRESENTATION

A digital computer represents all types of information in binary number system due to following reasons:

• In digital computers all electronic components operate in binary mode.
• Computers use binary system where only two digits present.
• Whatever can be done using decimal number system can also be done using a binary number system.

1.8.1 Number representation

The numbers in computer are represented using binary number system. An r base number system uses r distinct digits. The decimal number has 10 digits. So, decimal numbers are 10-base number system. The binary numbers system has two digits '0' and '1'. It is called base 2 number system. The octal numbers system has eight digits 0, 1, 2, 3, 4, 5, 6 and 7. The decimal number 831.6 can be written as follows with power of base 10.

Space for learners:

$8 \times 10^2 + 3 \times 10^1 + 1 \times 10^0 + 6 \times 10^{-1}$

When a binary number 101101 is written in this way with power of base 2, it provides decimal equivalent.

STOP TO CONSIDER
The hexadecimal numbers system has 16 digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.

$1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 45$

The decimal number can be converted to r base number system by using the steps:

- At first the number is separated into its integer and fraction parts and then each part converted separately.
- The integer part is converted to r base by dividing it successively with r until it becomes zero.
- The remainders in reverse order give the r base equivalent.
- The fraction part is converted to r base by multiply it repeatedly by r until its fraction part becomes zero.

Suppose, decimal number 112.8125 has to convert into binary. Here integer part is 112 and fraction part is 0.8125. At first, we will convert integer part 112 into binary then fraction part according to above rules. Since binary number system is 2 base we will divide 112 by 2 until it become zero. The following table depicts the process.

Division	Remainder
112 / 2 = 56	0
56 / 2 = 28	0
28 / 2 = 14	0
14 / 2 = 7	0
7 / 2 = 3	1
3 / 2 = 1	1
1 / 2 = 0	1

Now write down the remainder in reverse order i.e. 1110000 which is binary equivalent number of decimal integer 112. Next, the fraction part 0.81252 is multiplied by 2. The fraction of that result is again multiplied by 2until fraction part become zero.

Space for learners:

Multiplication	Resultant integer part (R)
0.81252 x 2= 1.625	1
0.6252 x 2= 1.25	1
0.252 x 2= 0.50	0
0.50 x 2= 1.0	1
0 x 2 = 0	0

The binary equivalent of fraction will be 0.11010. Using the same rules we can convert a decimal number to any base system.

1.8.1.1 Complements

Complements simplify the subtraction and logical manipulation in digital computer. There are two types of complements present in *r* base system namely *r's* and *(r – 1)'s* complement. If a number *N* in *r* base contains n digits, the *(r – 1)'s* complement of *N* is calculated as $(r^n - 1) - N$. For a decimal number, the 9's complement of *N* is $(10^n - 1) - N$. Thus, 9's complement of 545700 is 999999 – 545700 = 454299. In case of binary number, the 1's complement of *N* is calculated as $(2^n - 1) - N$. Thus, 1's complement of 1011000 is 1111111 – 1011000 = 0100111. Simply, 1's complement is obtained by just toggling all bits. The *r's* complement of a number *N* with n-digit is calculated as $n - N$. This is like adding 1 to the *(r – 1)'s* complement of the number. Thus, 10's complement of 2389 is 7610 + 1 = 7611. Similarly, 2's complement of 101100 is 010011 + 1 = 010100.

Check Your Progress-4

31. Computers use _____ system where only two digits present.

32. The octal numbers system has _____ digits.

33. The hexadecimal numbers system has _____ digits.

34. Complements simplify the _____ operation.

State TRUE or FALSE:

35. The decimal integer part is converted to r base by dividing it successively with r until it becomes zero.

36. There are two types of complements present in r base system namely r's and (r + 1)'s complement

37. 1's complement is obtained by just toggling all bits.

38. The binary number system has base 2.

Space for learners:

1.8.2 Fixed-Point Representation

Space for learners:

All positive integer numbers and zero can be considered as unsigned number. In order to represent negative numbers in computer signed numbers must be used. Because + and – signs are not present, rather these sign are represented by either '0' or '1'. The most significant bit of signed number is 0 for positive and 1 for negative. The fixed-point number representation has three parts as depicts in figure 1.6.

- Sign field
- Integer field
- Fractional field.

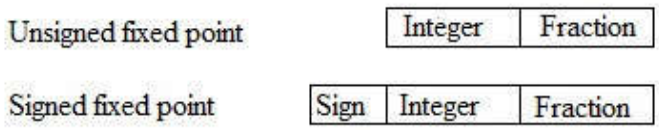


Figure 1.6: Fixed-point number representation

The 2's complementation representation is common in computer due to easier for arithmetic operations.

In a 32 bit register 1 bit reserved for the sign. Assume 15 bits are reserved for the integer part and 16 bits for the fractional part. The number -43.625 can be represented in register as depicted in figure 1.7.

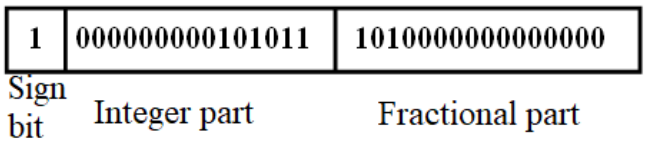


Figure 1.7: Representation of -43.625

The sign bit 1 represent - and 000000000101011 is 15 bit binary equivalent for decimal 43 and 1010000000000000 represent 16 bit binary equivalent for fraction 0.625.

1.8.3 Floating-Point Representation

The floating number consists of two parts. The first part is a signed fixed point number that is called mantissa. The second part exponent

represents the position of the decimal (or binary) point. The fixed point mantissa is either fraction or integer. The floating point number always represent in the form $M \times r^e$.

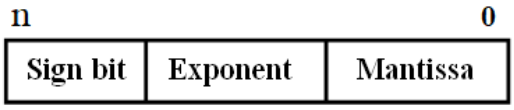


Figure 1.8: Floating point representation in register

The mantissa *M* and the exponent *e* present in the register with their sign as depicted in figure 1.8. A floating-point decimal number use base 10 for the exponent and binary number use base 2 for the exponent. A floating-point number is called normalized if the most significant bit (MSB) of the mantissa is 1. For positive integer, the MSB, or sign bit, is 0 and the remaining bits represent the magnitude. On the other hand for negative number, the MSB, or sign bit, is 1. The rest of the number can be represented in one of three ways

Signed-magnitude representation

Signed-1's complement representation

Signed-2's complement representation

Using floating point representation any non-zero number can be represented in the normalized form. Suppose, in 32-bit register 1 bit use as sign bit, 8 bits use for signed exponent, and remaining 23 bits represents fractional part. Now the decimal number −53.5 can represented as depicted in figure 1.9. The binary equivalent of -53.5 is $(-110101.1)_2$ and normalized representation is $(-1.101011)x2^5$

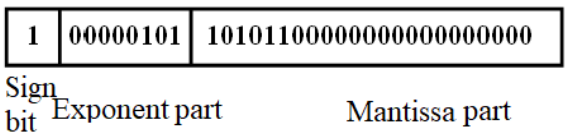


Figure 1.9: Floating point representation of -53.5

1.8.4 Character Representation

Different character codes are used to represent alphanumeric characters in bits 0 and 1. The most commonly used character code

Space for learners:

is American standard Code for Information Interchange (ASCII). **ASCII** uses 7-bits that provides 128 bit-patterns. In ASCII there are 26 lowercase and uppercase letters, 10 digits, and 32 punctuation marks. The remaining represents whitespace characters and special *control characters*. The uppercase A-Z, lowercase a-z and the digits 0-9 are in continuous series.

Space for learners:

Bit positions 654								Bit positions
000	**001**	**010**	**011**	**100**	**101**	**110**	**111**	**3210**
NUL	*DLE*	*SP*	0	@	P	‘	p	0000
SOH	*DC1*	!	1	A	Q	a	q	0001
STX	*DC2*	“	2	B	R	b	r	0010
ETX	*DC3*	#	3	C	S	c	s	0011
EOT	*DC4*	$	4	D	T	d	t	0100
ENQ	*NAK*	%	5	E	U	e	u	0101
ACK	*SYN*	&	6	F	V	f	v	0110
BEL	*ETB*	‘	7	G	W	g	w	0111
BS	*CAN*	(	8	H	X	h	x	1000
HT	*EM*	)	9	I	Y	i	y	1001
LF	*SUB*	*	:	J	Z	j	z	1010
VT	*ESC*	+	;	K	[	k	{	1011
FF	*FS*	,	<	L	\	l	|	1100
CR	*GS*	-	=	M	]	m	}	1101
SO	*RS*	.	>	N	^	n	~	1110
SI	*US*	/	?	O	_	o	*DEL*	1111

1.9 SUMMING UP

- An instruction set is collection of machine language or assembly language instructions that are understood by central processing unit (CPU).

- The machine may be 3-address machines, 2-address machines, 1-address machines and 0-address machines
- The computer supported instructions types are Data Transfer Instructions, Arithmetic, Bit Manipulation, Program Execution Transfer, Processor Control, Iteration Control and Interrupt Instructions.
- The most commonly used addressing modes are Immediate, Direct, Indirect, Register, Register indirect, Displacement and Stack addressing.
- CPU or processor organization has three categories: Single Accumulator organization, General register organization and Stack organization.
- The register inside the processor is in top level memory hierarchy followed by cache memory and main memory respectively.
- The registers have two categories: user-visible registers and control and status registers
- **General-purpose registers** are used to store temporary data during execution of instruction.
- **Data registers** can be used to hold data only. It cannot be used for calculating of operand address.
- **Address registers** may either general purpose or devoted to an individual addressing mode.
- **PC** holds the address of the next instruction to be executed.
- **IR** holds the address of currently executed instruction.
- **MAR** holds the address of an instruction to be fetched.
- **MBR** holds data that needs the current instruction or the result produced by the instruction.
- The use of status flags:
  Sign: It holds sign bit of the recent arithmetic operation. **Zero:** It is set when the result of operation is 0. **Carry:** It is set if addition operation produce a carry or borrow (for subtraction) from lower order bit.
- The numbers in computer are represented using binary number system.
- The floating number consists of two parts. The first part is a signed fixed point number that is called mantissa. The second part exponent represents the position of the decimal (or binary) point.

Space for learners:

- In ASCII there are 26 lowercase and uppercase letters, 10 digits, and 32 punctuation marks. The remaining represents whitespace characters and special *control characters*.

Space for learners:

1.10 ANSWERS TO CHECK YOUR PROGRESS

1. Register-register
2. Memory-memory
3. Destroyed
4. Accumulator
5. Stack
6. True
7. True
8. True
9. False
10. False
11. Memory location
12. Variable, constant
13. Effective
14. Address
15. Register indirect
16. True
17. False
18. True
19. True
20. False
21. ALU
22. Three
23. hierarchy
24. General
25. Supervisor, user
26. True
27. True
28. False
29. False
30. True
31. Binary
32. Eight
33. Sixteen
34. Subtraction
35. True
36. False
37. True
38. True

1.11 POSSIBLE QUESTIONS

Short answer type questions:

1. What is an instruction set?
2. Write the type of instruction for the following:
   JUMP, ADD
3. What are the types of CPU organization?
4. Arrange the followings in ascending order of access time: Secondary memory, Register, Main Memory, Cache Memory
5. What type of buses the system bus has?
6. What is the use of immediate addressing?
7. What is the Indirect Addressing? Give examples.

8. What is an accumulator?
9. Write assembly language code to evaluate X = (A-B) + (C-D) for stack based CPU
10. What are the categories of registers?
11. What happens to PC when interrupt occurs?
12. What is floating point representation?
13. What is 1's complement of 10011010?
14. What is 2's complement of 11000111?
15. Convert the decimal number 26.578 into binary number.

Long answer type questions:

1. Briefly explain the various addressing modes.
2. Briefly explain the instruction cycle.
3. List any five instruction types with adequate examples.
4. Convert decimal number 56.789 into binary, octal and hexadecimal number.
5. Briefly explain the data flow process with block diagram.

1.12 REFERENCES AND SUGGESTED READINGS

- Computer Architecture and Organization by B. Govindarajalu.; TMH publication.
- Advanced Computer Architecture A systems Design Approach by Richard Y. Kain; PHI Publication
- Computer Organization and Architecture Designing for Performance by William Stallings; Pearson Education
- Computer System Architecture by M. Morris Mano, PHI Publication.

---x---

Space for learners:

UNIT 2: COMBINATIONAL CIRCUITS AND ITS APPLICATIONS

Space for learners:

Unit Structure

Space for learners:

2.1 INTRODUCTION

This chapter describes the combinational circuits and the applications of combinational circuits. Sum of Product (SOP) and Product of Sum (POS) forms are the basic building blocks of the combinational circuits. When the logic gates are connected together to produce some specific output the resulting electronic circuit is known as combinational circuits, the combinational circuits don'tpossess any memory capacity. The output of the circuit always depends on the combination of the input variables.

2.2 UNIT OBJECTIVES

The unit is describing the designing and applications of combinational logic circuits. After completing the unit students′ will able to:

- Analyze and apply different combinations of the logic gates.
- Design combinational circuits from the Boolean expressions.
- Design combinational circuits from the truth table.
- Describe the universal behaviour of NAND and NOR logic gates.
- Explain and describe adder circuits.
- Analyze the comparator circuits.
- Describe decoders and encoders
- Describe multiplexers and demultiplexers
-

2.3 AND-OR LOGIC COMBINATIONAL CIRCUIT:

The Figure 2.1 shows an AND-OR circuit consisting of two input AND gates and one two input OR gate. The Boolean expression for

the AND gate outputs and the resulting SOP expression for the output Y are shown on the circuit diagram. The AND-OR circuit can have any number of AND and OR with any number of inputs.

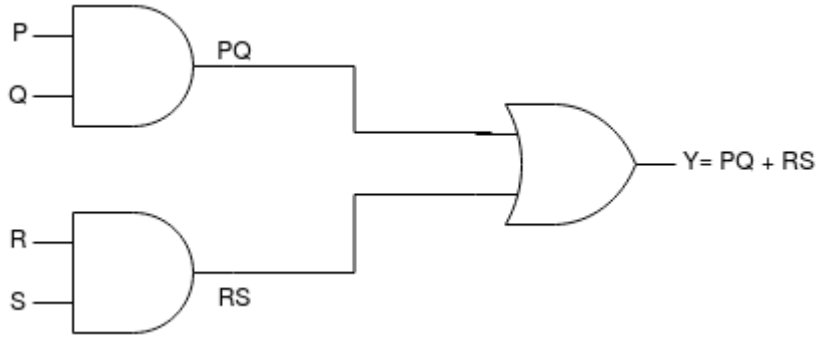


Figure 2.1 AND-OR logic diagram

The truth table for the above combinational circuit is shown in Table-2.1. The outputs of the AND gates are also shown in the table.

INPUTS				PQ	RS	OUTPUT Y
P	Q	R	S			
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	1	1
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	1	1
1	1	0	0	1	0	1
1	1	0	1	1	0	1
1	1	1	0	1	0	1
1	1	1	1	1	1	1

Table 2.1 Truth table for the logic circuit of Figure 2.1

2.4 AND-OR-INVERT LOGIC COMBINATIONAL CIRCUIT

If the output of the AND-OR circuit is complemented i.e. inverted, the resultant circuit is called AND-OR-Inverted circuit. The AND-

Space for learners:

OR logic implements the SOP expression and the corresponding POS expressions can be obtained using the AND-OR-Inverted logic. The logic circuit diagram Figure 2.2 shows an AND-OR-Inverted circuit and development of the POS output expression.

Y = (P′+Q′)(R′+S′) = (PQ)′(RS)′ = (((PQ)′(RS)′)′)′ = (((PQ)′)′ +((RS)′))′)′ = (PQ + RS)′

Space for learners:

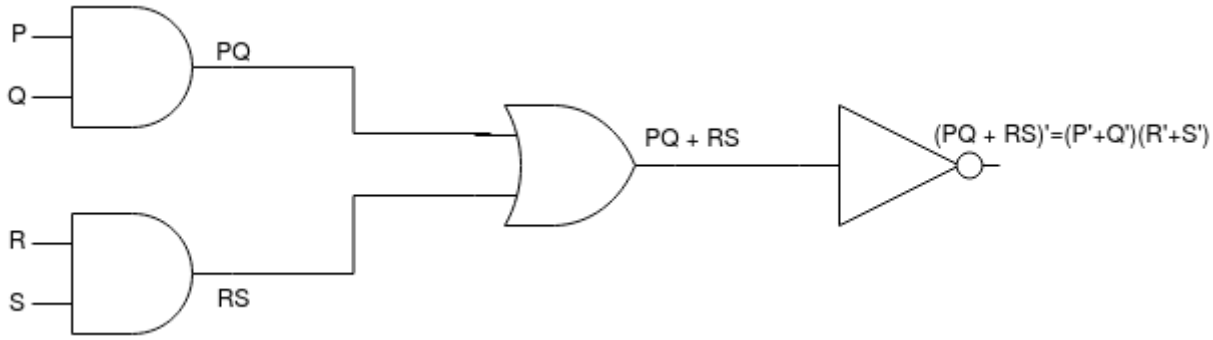


Figure 2.2 AND-OR Invert logic

In general, an AND-OR-Invert circuit can have any number of AND gates each with any number of inputs. A truth table can be developed from the AND-OR truth table in Table 2.1 by simply changing all 1s to 0s and all 0s to 1s in the output column.

STOP TO CONSIDER

- The AND-OR logic implements the SOP expressions, in other words, the SOP expressions are implemented using AND-OR logic.
- The AND-OR-Inverted logic implements POS expressions, in other words, the POS expressions are implemented using AND-OR-Inverted logic

2.5 EXCLUSIVE-OR LOGIC:

The exclusive-OR gate is considered a type of logic gate with its own unique symbol; it is actually a combination of two AND gates, one OR gate, and two inverters (NOT gate) as shown in Figure 2.3. The output is 1 only if the two inputs are at opposite levels.

Space for learners:

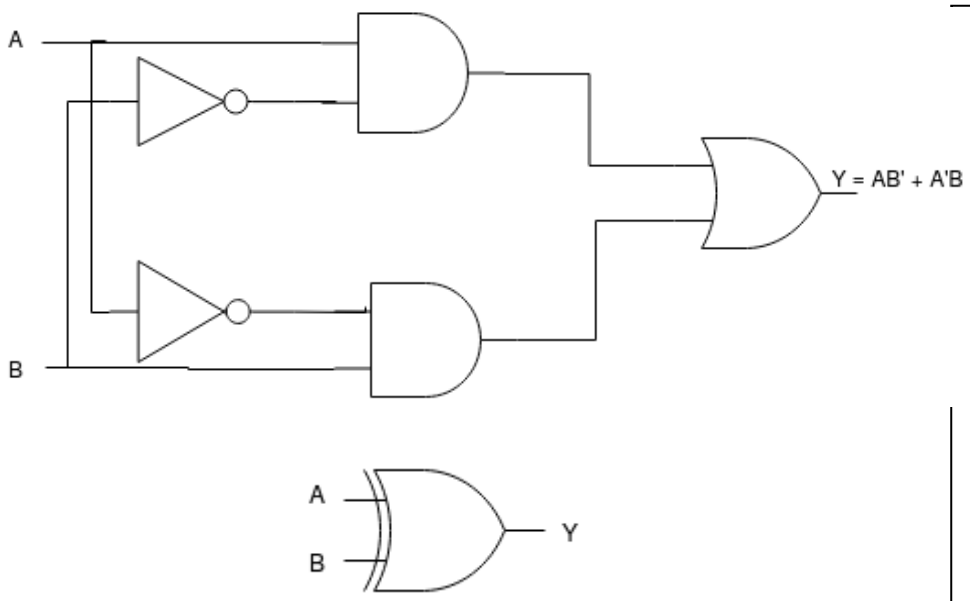


Figure 2.3 Exclusive-OR logic

The output expression for the circuit in Figure 2.3 is Y = AB′ + A′B i.e. Y = A ⊕ B

The truth table for exclusive-OR is shown in Table 2.2.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.2: Truth table of exclusive-OR

2.6 EXCLUSIVE-NOR LOGIC

The complement of exclusive-OR is known as exclusive-NOR, which is derived as follows:

Y = (AB′+A′B)′ = (AB′)′(A′B)′ = (A′ + B)(A+B′) = A′B′ + AB

The output Y is 1 only if the two inputs A and B are at the same level. The exclusive-NOR can be implemented by simply inverting the output of an exclusive-OR. The following Figure 2.4 (a) shows

the exclusive-NOR and Figure 2.4(b) shows the direct implementation of the expression A′B′+AB.

Space for learners:

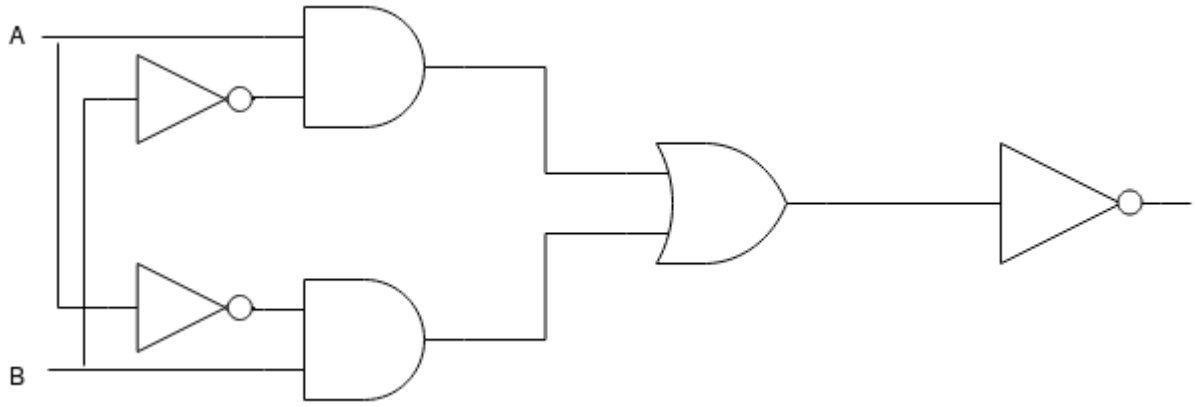


Figure 2.4(a) Y = (AB′+A′B)′

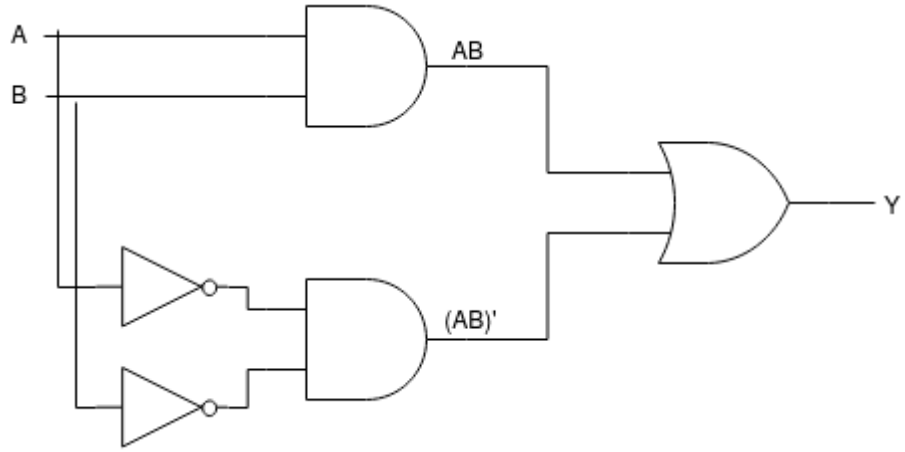


Figure 2.4(b) Y = A′B′ + AB

STOP TO CONSIDER

- Exclusive-OR (XOR) logic is a combination of two AND gates, one OR gate, and two inverters (NOT gate)
- Exclusive-NOR (XNOR) logic is a combination of two AND gates, one OR gate, and three inverters (NOT gate) or XNOR is obtained by applying an inverter at the output of XOR.

2.7 IMPLEMENTING COMBINATIONAL LOGIC

This section will describe the methods of implementing the logic circuits. The first method describes the implementation from the

Boolean expression and the second method describes the implementation from the truth table.

Space for learners:

2.7.1 Logic circuit design from Boolean expression

Let us consider the following Boolean expression:

$$Y = (A+B)(C+D+E)$$

A closer observation shows that the above expression ′Y′ consists of two terms.

The first term is formed by doing OR operation between A and B, and the second term is formed by doing OR operation among C, D, and E. The two terms are then AND together to produce the final output Y. The OR operations must be performed before the AND operation.

To design the combinational circuit, a 2-input OR gate is required to form the term A+B and a 3-input OR gate is required to form the term C+D+E. A 2-input AND gate is then required to combine the two OR terms. The resulting logic circuit is shown in Figure 2.5.

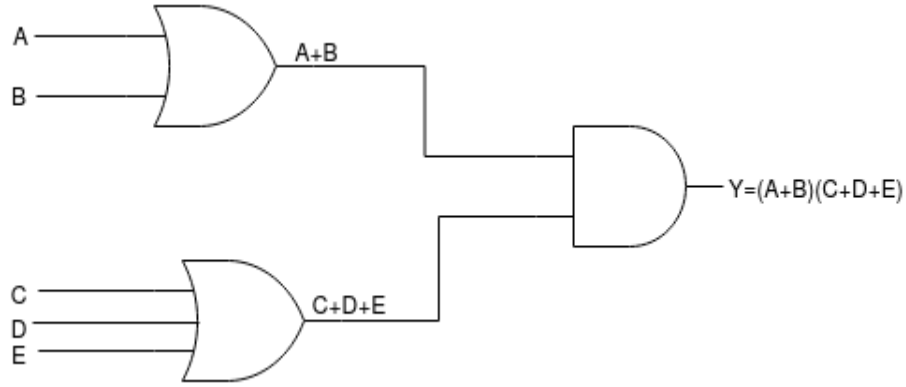


Figure 2.5. Logic circuit for the expression Y =(A+B)(C+D+E)

Let us implement the following expression as another example.

$$Y = (A+B)(C′D′+EF)$$

Like the previous example, let′s have a closer look at the expression. The terms A+B and (C′D′+EF) are AND together to form Y. The term C′D′+EF is first formed by doing AND between C′ and D′, E and F and then performs OR operation between these two terms. Before getting the expression C′D′ +EF, you must have the C′D′ and EF, before these two terms you must have C′ and D′. So, the logic operation must be performed in proper order. The logic gates

required to implement the expression Y = (A+B)(C′D′+EF) are as follows:

a. Two NOT gates to get C′ and D′
b. Two 2-input AND gates to form C′D′ and EF
c. Two 2-input OR gates to form A+B and C′D′+EF
d. One 2-input AND gate to form Y.

The logic circuit of this expression is shown in Figure 2.6

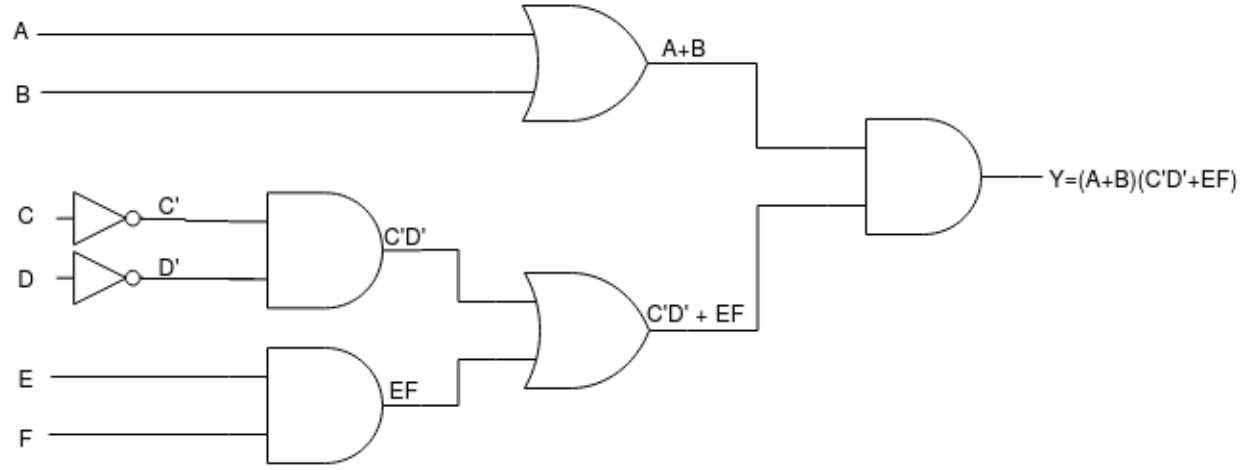


Figure 2.6. Logic diagram for the expression Y = (A+B)(C′D′+EF)

Space for learners:

2.7.2. Logic circuit design from truth table

Instead of using the SOP expression to design the combinational circuit you can use the truth table and from the truth table thatyou can deriveusing the SOP expression. Table 2.3 shows one example of such an implementation.

Inputs			Output Y	Product Terms
A	B	C		
0	0	0	0	
0	0	1	0	
0	1	0	1	A′BC′
0	1	1	0	
1	0	0	1	AB′C′
1	0	1	0	
1	1	0	0	
1	1	1	1	ABC

Table 2.3: Truth table for logic function

The Boolean expression obtained for the Table 2.3 is given below:

$$Y = A'BC'+AB'C'+ABC$$

The expression Y is obtained by doing OR operations among the product terms for which the output is 1. The first, second, and third are formed by doing AND operations among (A′, B, C′), (A, B′, C′), and (A, B, C) respectively. The logic gates required to implement the circuit are as follows:

a. Three NOT gates.
b. Three 3-input AND gates.
c. One 2-input OR gate.

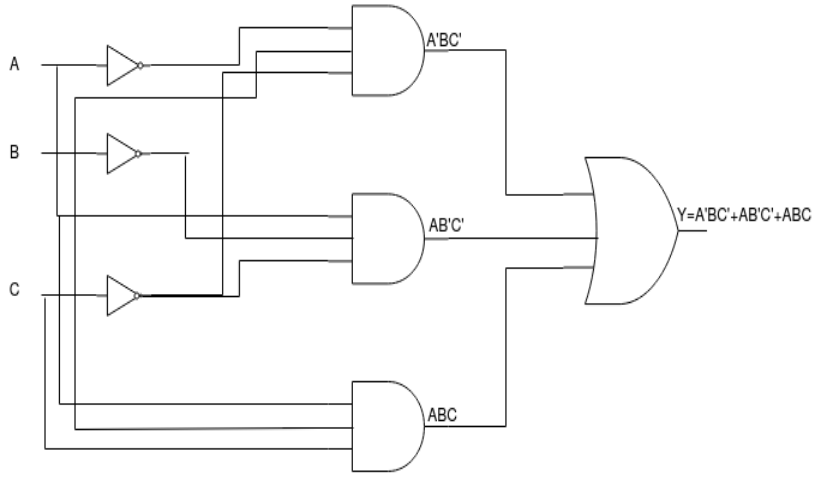


Figure 2.7 Logic diagram for the expression Y = A′BC′+AB′C′+ABC

Reduce the combinational logic circuit shown in Figure 2.8 to a minimum form.

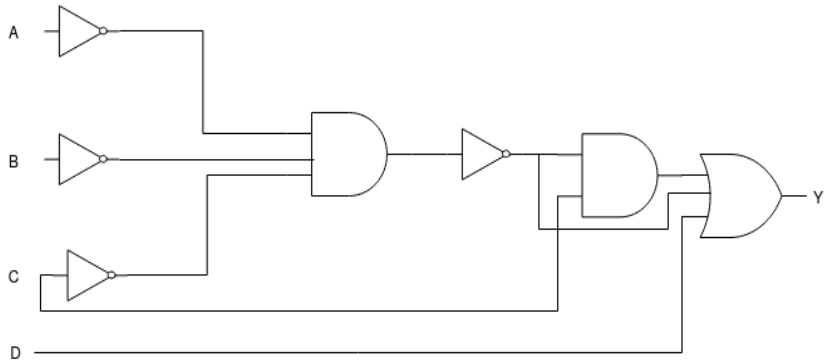


Figure 2.8 Combinational logic circuit to be reduced

The expression for the output of the circuit is Y = (A′B′C′)′C+(A′B′C′)′+D

Space for learners:

Applying D′ Morgan′s theorem and Boolean algebra,

$Y = ((A')'+(B')'+(C')')C+(A')'+(B')'+(C')'+D$

$= AC+BC+CC+A+B+C+D$

$=C(A+B+1)+A+B+C+D$

$Y=A+B+C+D$

The simplified circuit is a 4-input OR gate as shown in the Figure 2.9

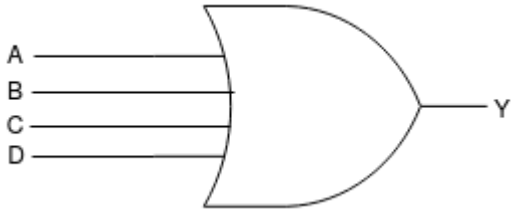


Figure 2.9 Reduced form the logic circuit of Figure 2.7

Note: Before implementing the logic circuit directly it is better to reduce the algebraic expressions to its minimized form so that the number of gates required to implement the circuit is minimum. This leads to reduction of propagation delay among the gates. More the number of gates, the more the propagation delay and also the heat produced by the circuit will increase.

STOP TO CONSIDER

- The implementation of combinational logic circuits is either from the Boolean expression or truth table.
- The expression should be carefully observed and has to identify the number of AND, OR, inverters required.
- Before implementing the logic circuit, it is advisable to reduce the expression by applying the boolean algebra
- If the number of gates are less in the final combinational circuit then the propagation delay will also be minimal.

CHECK YOUR PROGRESS

1. POS stands for _____
2. SOP stands for _____
3. An Exclusive-OR can be represented as _____
4. The number of AND gates required to implement the boolean expression ABC is ____

Space for learners:

2.8 THE UNIVERSAL PROPERTY OF NAND AND NOR GATES

Till now, you have studied combinational circuits designing with AND and OR, and NOT gates. This section will describe the universal property of NAND and NOR gates. The universality of NAND means it can be used as an inverter and the combinations of NAND gates can be used to implement the AND, OR, and NOR operations. Similarly, the NOR gate can be used to implement the inverter, AND, OR, and NAND operations.

2.8.1 The NAND gate as a universal logic element

The NAND gate is a universal gate because it can be used to produce the NOT, the AND, the OR, and the NOR functions. An inverter can be made from a NAND gate by connecting all of the inputs together and creating, in effect, a single input as shown in the Figure 2.10(a) for a 2-input gate. An AND function can be generated by the use of NAND gates alone as shown in Figure 2.10(b). An OR function can be implemented with only NAND gates, as shown in Figure 2.10(c). Similarly, the NOR function can also be produced as shown in Figure 2.10(d).

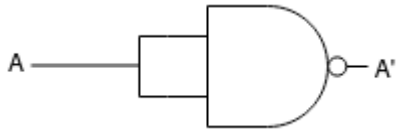


Figure 2.10(a) NAND gate as inverter or NOT

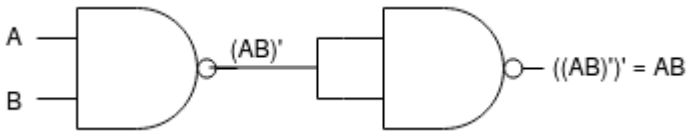


Figure 2.10(b) Two NAND gates are combined to produce AND operation

Space for learners:

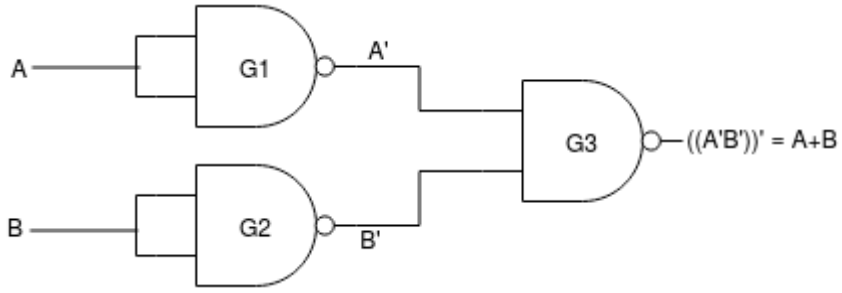


Figure 2.10(c) Three NAND gates are combined to produce OR operation

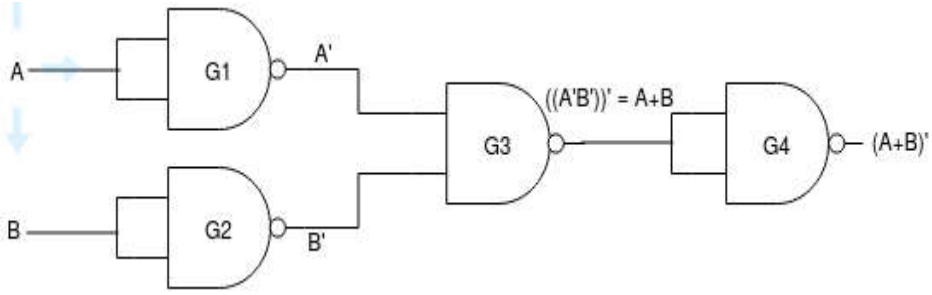


Figure 2.10(d) Four NAND gates are combine to produce NOR operation

In Figure 2.10(b), a NAND gate is used to invert a NAND output to form the AND function which is given below:

$$Y = ((AB)')' = AB$$

In Figure 2.10(c), NAND gates G1 and G2 are combined to invert the two input variables before they are applied to NAND gate G3. The OR gate output is derived as follows by applying DeMorgans′s theorem:

$$Y = ((A'B')' = A + B$$

In Figure 2.10(d), NAND gate G4 is used as an inverter connected to the circuit of part (c) to produce the NOR operation (A+B)′.

Finally, we can conclude that using the NAND gate it is possible to implement any logic function.

2.8.2 The NOR gate as a Universal logic element

The NOR gate can also be used to produce the NOT, AND, OR, and NAND functions. A NOT circuit, or inverter, can be made from NOR gate by connecting all of the inputs together to effectively create a single input, as shown in Figure 2.11(a) with a 2-input example. Also, an OR gate can be produced from NOR gates as shown in Figure 2.11(b). An AND gate can be produced using the

Space for learners:

NOR gates as shown in the Figure 2.11(c), the NOR gates G1 and G2 are used as inverters and the final output is derived by the use of DeMorgan′s theorem as follows:

$$Y = (A'+B')' = AB$$

Figure 2.11(d) shows the implementation of NAND function using NOR gates. Hence we can conclude that the NOR gate can also work as a universal gate like the NAND gate.

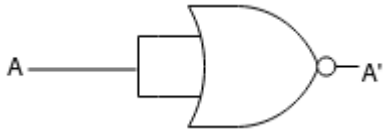


Figure 2.11(a) NOR gate used as inverter

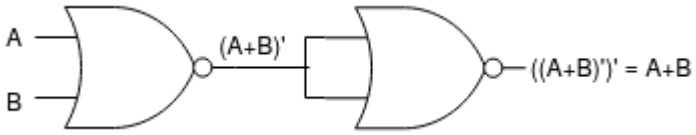


Figure 2.11(b) NOR gates are combined to produce OR operation

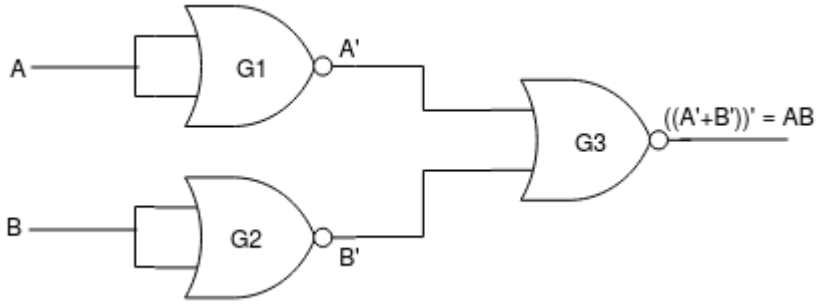


Figure 2.11(c) NOR gates are combined to produce AND operation

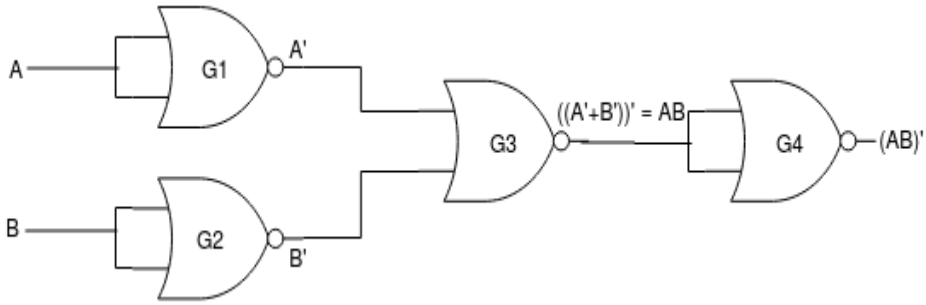


Figure 2.11(d) NOR gates are combined to produce NAND operation

2.8.3 Combinational circuit using NAND gate

NAND gates can work as either NAND or negative OR by applying DeMorgan′s theorem.

$$(AB)' = A' + B'$$

Space for learners:

Consider the NAND logic as shown in Figure 2.12. The output expression is developed in the following steps:

$$Y = ((AB')(CD'))'$$
$$= ((A'+B')(C'+D'))'$$
$$= (A'+B')'+(C'+D')'$$
$$= (A')'(B')' + (C')'(D')'$$
$$= AB + CD$$

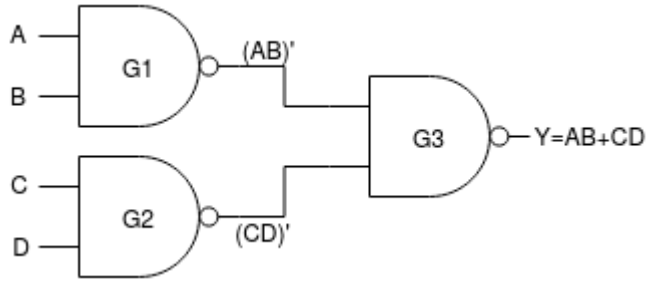


Figure 2.12 Implementation of the Boolean expression Y= AB+CD using NAND gate

2.8.4 Combinational circuit using NOR gate

The NOR gate can work as either a NOR or negative AND, as shown by DeMorgan's theorem.

$$(A+B)' = A'B'$$

Consider the NOR logic in Figure 2.13. The output expression is developed as follows:

$$Y = ((A+B)'+(C+D)')' = ((A+B)')'((C+D)')' = (A+B)(C+D)$$

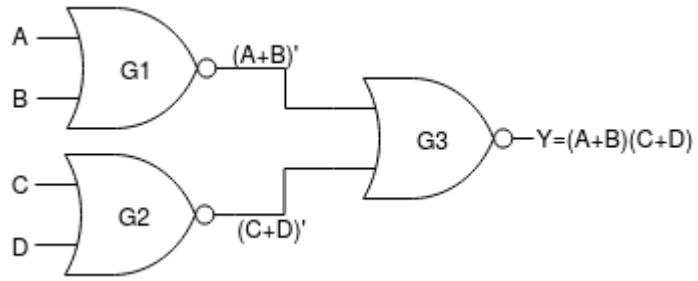


Figure 2.13 Implementation of the boolean expression Y= (A+B)(C+D) using NOR gate

STOP TO CONSIDER

- NAND and NOR gates are called universal gates.
- NAND and NOR can be used to implement all the primary logic like AND, OR, NOT(invert).
- NAND can produce NOR and, similarly, NOR can produce NAND.

Space for learners:

Space for learners:

CHECK YOUR PROGRESS

5. The number of NOR gate(s) required to implement OR is/are ____
6. The number of NAND gates(s) required to implement AND is/are ____
7. The number of NAND gates(s) required to implement Y = A′+B is/are _____
8. The number of NOR gates(s) required to implement Y = A′+B is/are _____

2.9 COMBINATIONAL LOGIC CIRCUIT FUNCTIONALITIES

In this section, many types of fixed functions of combinational circuits are introduced, viz. Adders, comparator, decoders, encoders, code converters, multiplexer, demultiplexer etc.

2.9.1 The comparison function

The magnitude of comparison performed by a logic circuit is called a comparator. A comparison compares two quantities and indices whether or not they are equal. Figure 2.14 represents a comparison function, one number in binary form is applied to input A, and the other number in binary form is applied to input B. The outputs indicate the relationship of the two numbers by producing 1 on the proper output line. Suppose that the binary representation of 3 is applied to input A and a binary representation of the number 6 is applied to input B. A ′1′ (HIGH) will appear on the A<B output, indicating the relationship between the two numbers.

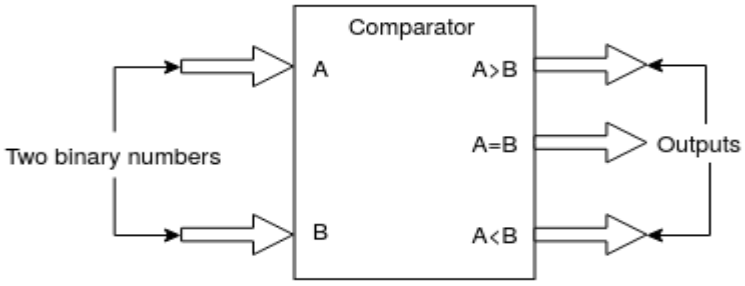


Figure 2.14 Basic magnitude comparator

2.9.2 The Arithmetic functions

Addition is performed by a logic circuit called adder. An adder adds two binary numbers on inputs A and B with a carry input (C_{in}) and generates a sum (Σ) and a carry output (C_{out}) as shown in Figure 2.15.

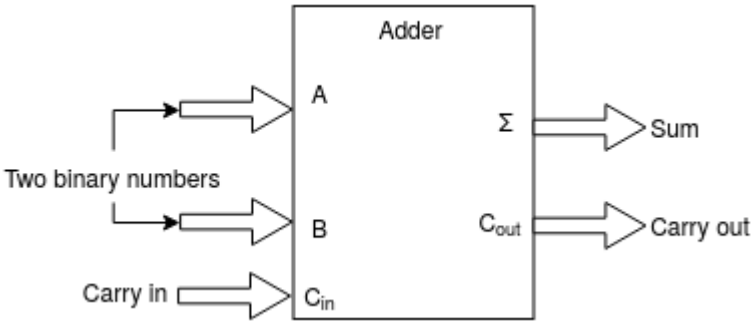


Figure 2.15 Basic Adder

Subtraction is also performed by a logic circuit. A subtractor requires three inputs, viz.,the two numbers that are to be subtracted and a borrow input. The two outputs are the difference and the borrow output. The subtraction operation is a special case of addition operation.

Multiplication: A multiplier is a logic circuit that performs multiplication. Because numbers are always multiplied in twos, two inputs are necessary. The product is the multiplier′s output. Multiplication can be achieved using an adder in conjunction with other circuits since it is merely a series of additions with shifts in the positions of the partial products.

Division: Division can be achieved with a series of subtraction, comparisons, and shifts, therefore an adder can be used in conjunction with other circuits. The division requires two inputs, and the quotient and remainder are generated as outputs.

Code conversion: The logic circuits can also be used for code conversion. A code is a collection of bits arranged in a specific pattern and used to represent data. A code converter converts one type of coded data into another type of coded data. For example, Conversion from binary to Binary Coded Decimal (BCD) or Gray code.

Encode: The **encoder** is a logic circuit that performs the encoding function. The encoder turns data into a coded representation, such as

Space for learners:

a decimal number or an alphanumeric letter. One form of encoder, for example, turns all of the decimal digits, 0 through 9, to binary code.

Decoder: A logic circuit called a *decoder* performs the decoding operation. The decoder translates coded data, such as binary numbers, to uncoded data, such as decimal numbers. One form of decoder, for example, translates a 4-bit binary code into the appropriate decimal digits.

Data selection function: The multiplexer and the demultiplexer are two types of circuits that select data in the data selection function. A multiplexer, often known as a MUX, is a logic circuit that transfers digital data from many input lines to a single output line in a predetermined time sequence. A multiplexer can be thought of as an electronic switch that links each of the input lines to the output line in a sequential manner. A demultiplexer (DEMUX) is a logic circuit that converts digital data from one input line to multiple output lines in a predetermined order. The demux is a reverse mux. When data from numerous sources needs to be sent across one line to a distant place and then redistributed to multiple recipients, multiplexing and demultiplexing are utilized.

STOP TO CONSIDER

- The AND, OR, and NOT can be used to design the complex logic circuits to perform specific operations.

2.9.3 Basic Adders

Adders are essential not only in computers, but also in a wide range of digital systems that process numerical data. The study of digital systems requires a basic understanding of the adder action. The half-adder and full-adder are described in this section.

2.9.3.1 The Half-Adder

Recall the basic rules for binary addition
0+0 = 0
0+1 = 0

Space for learners:

1+0 = 1
1+1 = 10
A logic circuit known as a half-adder performs the operations.
The half-adder takes two binary digits as inputs and produces two binary digits, a sum bit and a carry bit, as outputs. Figure 2.16 shows a half-adder represented by the logic symbol.

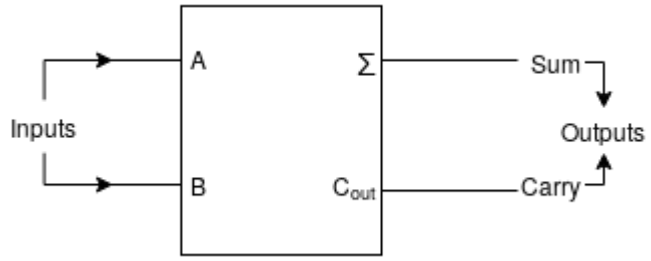


Figure 2.16 Logic symbol for a half-adder

Half-Adder Logic from the operation of the half-adder as stated in Table 2.3, expressions can be derived for the sum and the output carry as functions of the inputs. Note that the output carry (C_{out}) is a 1 only when both A and B are 1s, therefore, C_{out} can be expressed as the AND of the input variables. $C_{out} = AB$.

Table 2.3 Half-adder truth table

A	**B**	**C_{out}**	**Σ**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Now, observe that the sum (Σ) is a 1 only if the input variables, A and B, are not equal. The sum can therefore be expressed as the exclusive-OR of the input variables. $\Sigma = A \oplus B$. The logic implementation required for the half-adder function can be developed using Σ and C_{out}. The output carry is produced with AND gate with A and B on the inputs, and the sum output is generated with an exclusive-OR (XOR) gate, as shown in Figure 2.17. Remember, the XOR is implemented with AND gates, an OR gate, and inverters.

Space for learners:

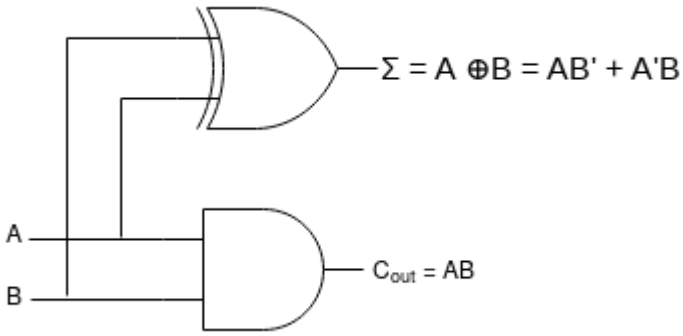


Figure 2.17 Half-adder logic diagram

2.9.3.2 The Full Adder

The second category of adder is the full-adder. The full-adder accepts two input bits and an input carry and generates a sum output and an output carry. The basic difference between full-adder and a half-adder is that the full-adder accepts an input carry. A logic symbol for a full-adder is shown in Figure 2.18, and the truth table in Table 2.4 shows the operation of a full-adder.

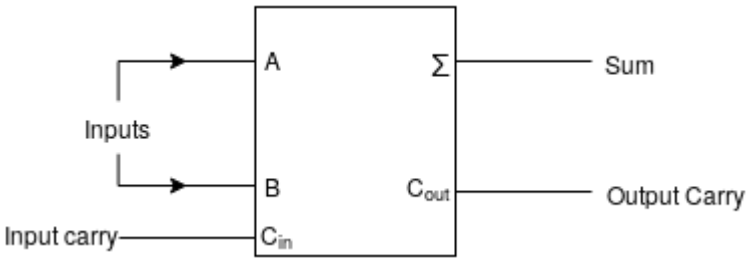


Figure 2.18 Logic symbol for a full-adder

Table 2.4 Full-adder truth table

A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Space for learners:

Full-Adder Logic The full-adder must add the two input bits and input carry. From the half-adder you know that the sum of the input bits A and B is exclusive-OR of those variables. A⊕B. For the input carry (C_{in}) to be added to the input bits, it must be exclusive-ORed with A⊕B, yielding the equation for the sum output of the full-adder.

$\Sigma = (A \oplus B) \oplus C_{in}$. This means that to implement the full-adder sum function, two 2-input exclusive-OR gates can be used. The first must generate the term A⊕B, and the second has as its inputs the output of the first XOR gate and the input carry, as shown in Figure 2.19(a).

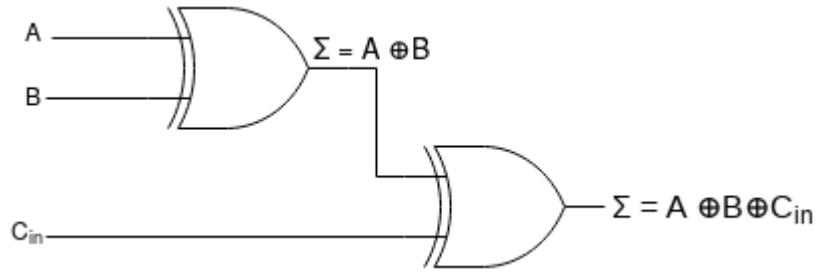


Figure 2.19(a) Logic required to form the sum of three bits

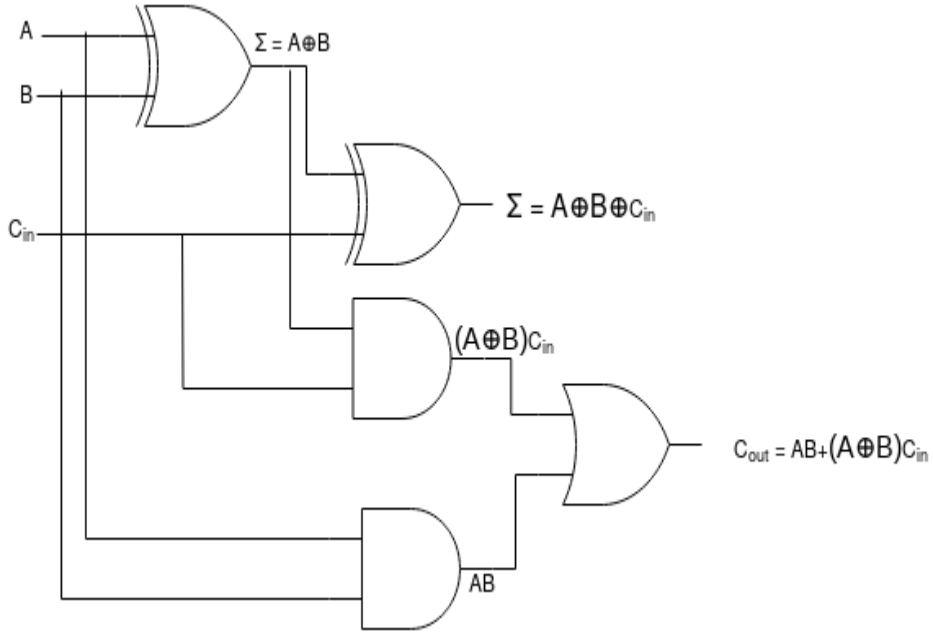


Figure 2.19(b) Complete logic circuit for a full-adder

The output carry is a 1 when both inputs to the first XOR gate are 1s or when both inputs to the second XOR gate are 1s. You can verify this fact by studying Table 2.4. The output carry of full-adder is therefore produced by the inputs A ANDed with B and A⊕B

Space for learners:

ANDed with C_{in}. These two terms are ORed, as expressed in the expression of C_{out}. This function is implemented and combined with the sum logic to form a complete full-adder circuit, as shown in Figure 2.19(b). Notice that in Figure 2.19(b) there are two half-adders, connected as shown in the block diagram of Figure 2.20(a), with their output carries ORed. The logic symbol shown in Figure 2.20(b) will normally be used to represent the full-adder.

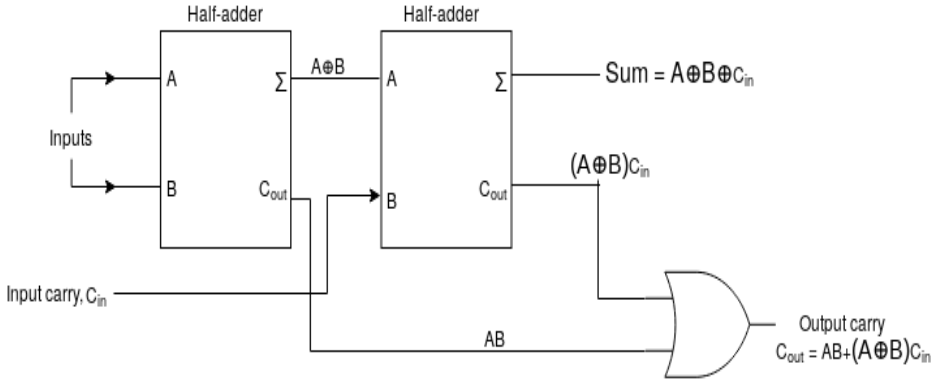


Figure 2.20(a) Arrangement of two half-adders to form a full-adder

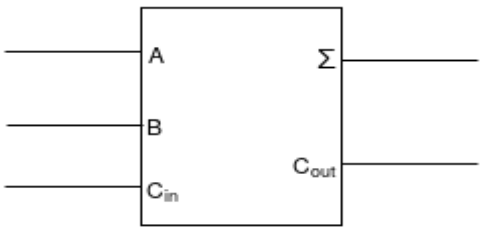


Figure 2.20(b) Full-adder logic symbol

2.9.3.3 Parallel Binary Adders

Parallel binary adders are formed by connecting two or more full-adders. The basic operations of such adders, as well as their associated input and output functions, are described in this section. A single full-adder can add two one-bit numbers as well as an input carry. Additional full-adders must be used to add binary numbers with more than one bit. As shown above with 2-bit integers, when one binary number is added to another, each column creates a sum bit and a 1 or 0 carry bit to the next column to the left.

10
+10

100

Space for learners:

Space for learners:

In this case, the second column′s carry bit becomes the third column′s sum bit. A full adder is required for each bit in two binary numbers to be added. So two adders are required for 2-bit numbers, four adders are required for 4-bit values, and so on. Each adder′s carry output is connected to the next higher-order adder′s carry input, as shown in Figure 2.21 for a 2-bit adder. Because there is no carry input to the least significant bit location, either a half-adder or the carry input of a full-adder can be set to 0 (grounded). In Figure 2.21 the least significant bits (LSB) of the two numbers are represented by A_1 and B_1. The next higher-order bits are represented by A_2 and B_2. The three sum bits are Σ_1, Σ_2, and Σ_3. Notice that the output carry from the left-most full-adder becomes the most significant bit (MSB) in the sum Σ_3.

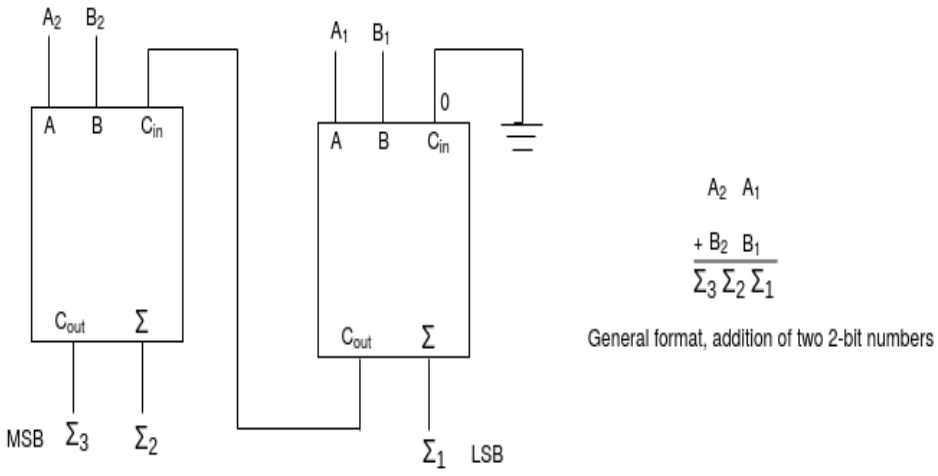


Fig 2.21. A 2-bit adder

Four-bit Parallel Adders

A nibble is a collection of four bits. As demonstrated in Figure 2.22, a basic 4-bit parallel adder is developed with four full-adder stages. The LSBs (A_1 and B_1) of each number being added are applied to the right-most full-adder; the higher-order bits are applied to the gradually higher-adders as illustrated; and the MSBs (A_4 and B_4) of each number are applied to the left-most full-adder. The carry output of each adder is connected to the carry input of the next higher-order adder as indicated. These are called internal carries.

In terms of the method used to handle carries in a parallel adder, there are two types: the *ripple carry* adder and *carry look-ahead* adder. A *ripple carry* adder is one in which the carry output of each full-adder is connected to the carry input of the next higher-order stage (a stage is one full-adder). The sum and the output carry of any stage cannotbe produced until the input carry occurs. This causes a time delay in the addition process. The carry propagation delay for each full-adder is the time from the application of the input carry until the output carry occurs, assuming that the A and B inputs are already present.

Space for learners:

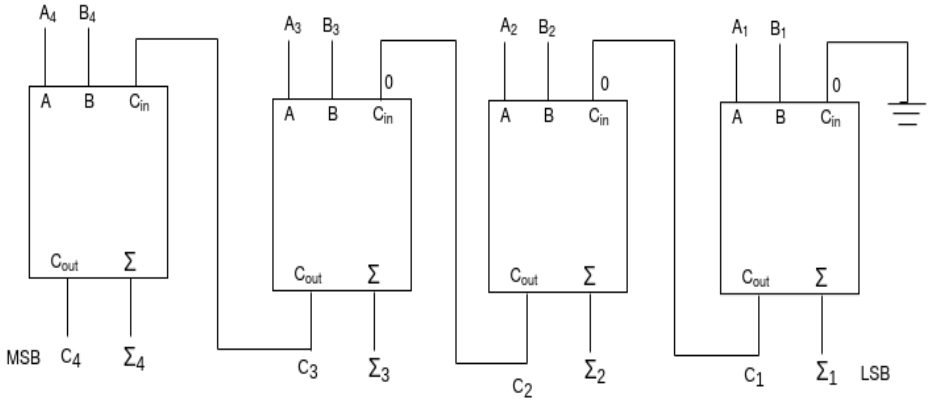


Figure 2.22 A 4-bit Adders

Look-ahead carry addition is a technique for speeding up the addition process by eliminating the *ripple carry* delay. The *look-ahead carry* adder predicts each stage′s output carry and produces it using either *carry generation* or *carry propagation* based on the input bits of each stage.

Carry generation occurs when an output carry is produced (generated) internally by the full-adder. A carry is generated only when both input bits are 1s. The generated carry, C_g, is expressed as the AND function of the two input bits, A and B. C_g=AB.

Carry Propagation occurs when the input carry is rippled to become the output carry. An input carry may be propagated by the full-adder when either or both of the input bits are 1s. The propagated carry, Cp, is expressed as the OR function of the input bits. C_p = A + B.

2.9.3.4 Truth table for 4-bit parallel adder

Table 2.5 is the truth table for a 4-bit adder. On Some data sheets, truth tables may be called *function tables or functional truth tables*. The subscripts *n* represent the adder bits and can be 1, 2, 3, or 4 for

the 4-bit adder. C_{n-1} is the carry from the previous adder. Carries C_1, C_2, and C_3 are generated internally. C_0 is an external carry input and C_4 is an output.

Table 2.5 Truth table for 4-bit parallel adder

C_{n-1}	A_n	B_n	Σ_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

STOP TO CONSIDER

- A half-adder has two inputs and two outputs.
- A full-adder has three inputs and two outputs.
- A 4-bit parallel adder can add two 4-bit binary numbers.
- Two half-adders can be used to design a full adder.

CHECK YOUR PROGRESS

State whether true or false

9. The sum expression for a half adder is A+B
10. The carry out C_{out} expression for a full adder is $AB+C_{in}$
11. A 4-bit parallel adder has four full adders.
12. There are two types of carry, they are ripple carry and look ahead carry
13. Carry generation occurs when an output carry is produced.

2.9.4 Binary Subtractor

Binary subtractors are special circuits which subtract two binary numbers from each other. Binary subtractor produced a difference and borrow output after the completion of the subtraction operation. Binary subtraction has two digits, subtracting a "0" from a "0" or a "1" leaves the result unchanged as 0-0 = 0 and 1-0 = 1. Subtracting

Space for learners:

a “1” from a “1” results in a “0”, but subtracting a “1” from a “0” requires a borrow. In other words, 0-1 requires a borrow and if you borrow 1 then the minuend 0 becomes 10 and the operation 0-1 becomes 10-1 which will give the output 1, this also leads to set the borrow bit 1. The half-subtractor and full-subtractor are discussed below.

Space for learners:

2.9.4.1 The Half-Subtractor

A half subtractor is a logical circuit that performs a subtraction operation on two binary digits. The half subtractor produces a difference (D) and a borrow out (B_{out}) bit for the next stage. The Figure 2.23 shows the logic symbol of a half-subtractor circuit.

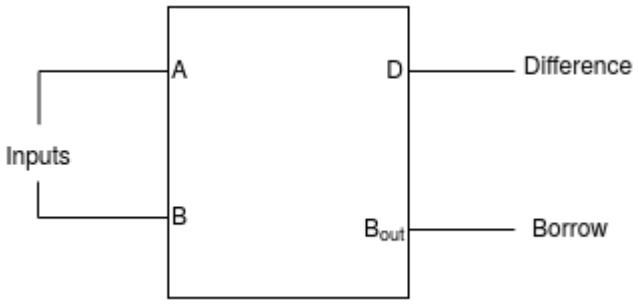


Figure 2.23 Logic symbol of Half-Subtractor

Table 2.6: Truth Table of a Half-Subtractor

Inputs		Outputs	
A	**B**	**Difference(D=A-B)**	**Borrow (B_{out})**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

From the Table 2.6 of the half subtractor, the difference (D) output can be obtained by doing exclusive-OR between A and B and the Borrow (B_{out}) can be obtained by doing AND operation between A′ and B. The Boolean expression for a half subtractor is as follows.

$$D = A \oplus B$$
$$B_{out} = A'B$$

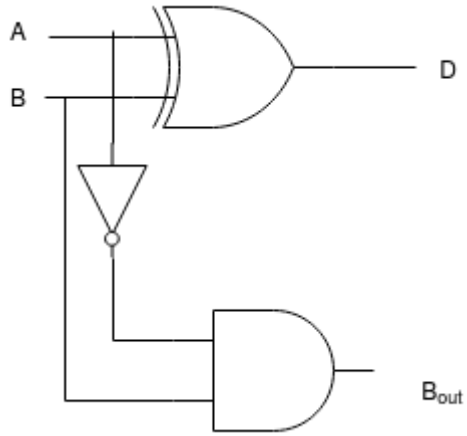


Figure 2.24 Logic circuit for Half-subtractor

The Boolean expressions for 'sum' in half-adder and 'difference' in half-subtractor are exactly the same. The only difference is the output carry of the half-adder and the borrow out of the subtractor circuit, difference between these two quantities is the inversion of the minuend input A.

The disadvantage of the half-subtractor is that if you subtract multiple bits there is no option for 'borrow-in' from its earlier stages. So, we need a full subtractor circuit to take into account this borrow-in input from the earlier stages.

2.9.4.2 The Full-Subtractor

The full-subtractor has three inputs. The two single bit data inputs A (minuend) and B (subtrahend) are the same as before plus an additional *Borrow-in* (B_{in}) input to receive the borrow generated by the subtraction process from a previous stage as shown in Figure 2.25.

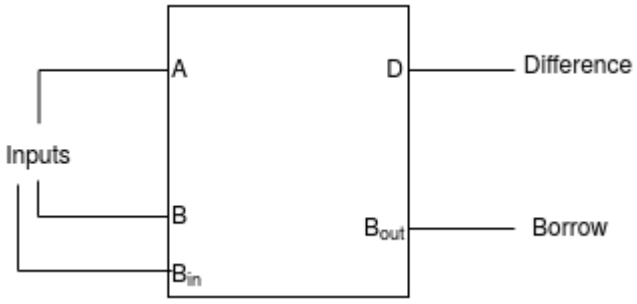


Figure 2.25 : Logic symbol of a Full-Subtractor

Space for learners:

Space for learners:

The "full subtractor" combinational circuit performs the subtraction operation on three binary bits resulting in outputs for the difference D and borrow B_{out}. Like the adder circuit, the full subtractor can also be thought of as two half subtractors connected together, with the first half subtractor passing its borrow to the second half subtractor as shown in the Figure 2.26 and the complete logic circuit of the full-subtractor is shown in the Figure 2.27.

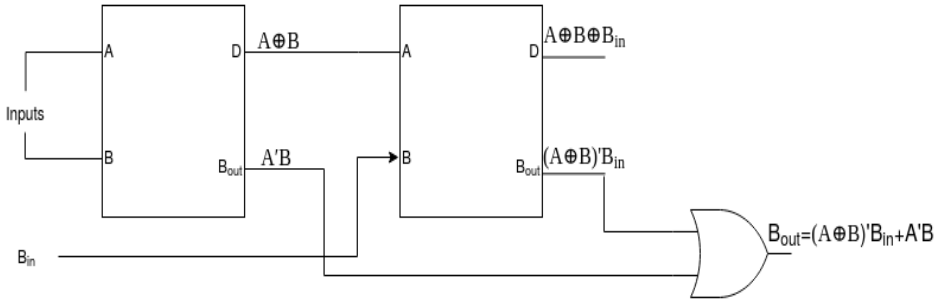


Figure 2.26: Arrangement of two half-subtractors to form a full-subtractor

Table 2.7: Truth for the Full-Subtractor

Inputs			**Outputs**	
A	**B**	**B_{in}**	**D**	**B_{out}**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

The truth table Table 2.7 shows the subtraction operation between A and B, the truth table operations are explained below:

a. If A = 0, B = 0, and B_{in}= 0, then the output D and B_{out} both are 0.

b. In the second set of inputs the A = 0, B = 0, but the B_{in} =1, so before performing the subtraction operation, first, you have to increment the B by 1 unit then the B will change to 1

(B_{in}=1 indicates there was a borrow in the *previous step* of the series of operations here *previous step* is not referring the first set of input operation of the Table 2.7), now if you perform the A-B i.e. 0-1 you need a borrow then only the operation will be possible, so, if you borrow 1 then the A will change to 10 and the subtraction operation 10-1 will give 1, i.e. D=1, since the operation was performed using borrow, the B_{out}=1.

c. In the third set of inputs A=0, B=1, B_{in}= 0, since B_{in}= 0, so, it is not necessary to increment the B, but A-B i.e. 0-1 in this step needs a borrow, so you have to borrow 1 to A, the A will change to 10 and the operation will change to 10-1=1, so, D=1, B_{out}=1.

d. In the fourth set of inputs A=0, B=1, B_{in}= 1, this time B_{in}= 1, so, you have to increment the B by one unit, then B will change to 10, now, if you perform the subtraction operation A-B, i.e. 0-10 then you need a borrow, if you borrow 1 the A will change to 10 and the operation becomes 10-10=0, therefore D=0, since the operation was completed using a borrow so, B_{out}= 1.

e. In the fifth set of inputs A=1, B=0, B_{in}= 0, since B_{in}= 0, so the B will not change and the A-B i.e. 1-0 does not need any borrow therefore D=1 and B_{out}= 0.

f. In the sixth set of inputs A=1, B=0, B_{in}= 1, since B_{in}= 1, so, B will be incremented by 1 unit and B becomes 1 i.e. B=1 and the operation A-B will be 1-1=0, hence D=0 and B_{out}= 0.

g. In the seventh set of inputs A=1, B=1, B_{in}= 0, since B_{in}= 0, so, B will not change and the A-B i.e. 1-1 does not need any borrow therefore D=0 and B_{in}= 0.

h. In the eighth set of inputs A=1, B=1, B_{in}= 1, since B_{in}= 1, so, B will be incremented by 1 unit and the value of B becomes 10, now, the operation A-B becomes 1-10 which is not possible therefore A needs borrow to complete the operation, after 1 borrow to A the A becomes 11 and A-B = 11-10. Hence D=1 and B_{out}= 1.

Space for learners:

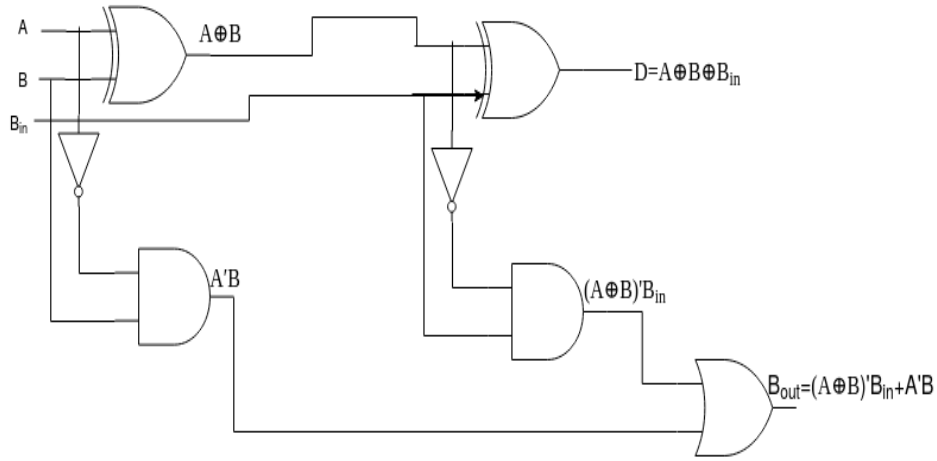


Figure 2.27: Complete logic circuit for a full-subtractor

STOP TO CONSIDER

- Subtractor circuits are similar to the adder circuits.
- Subtraction operation in binary works in the same pattern that works in normal mathematics.
- The difference expression of the subtractor circuit is the same as the sum expression of the adder circuit.
- The Full-subtractor supports borrow in from the previous stages.
- B_{in}= 1 indicates there is a borrow in the previous step.
- If A<B+B_{in}then A needs borrow from its

2.9.5 Comparators

A comparator′s primary role is to compare the magnitudes of two binary quantities in order to identify their relationship. A comparator circuit, in its most basic form, determines if two integers are equal.

2.9.5.1 Equality

Because its output is 1 when the two input bits are not equal and 0 when they are equal, the exclusive-OR gate can be used as a basic comparator. As a 2-bit comparator, Figure 2.28 depicts the exclusive-OR gate.

Space for learners:

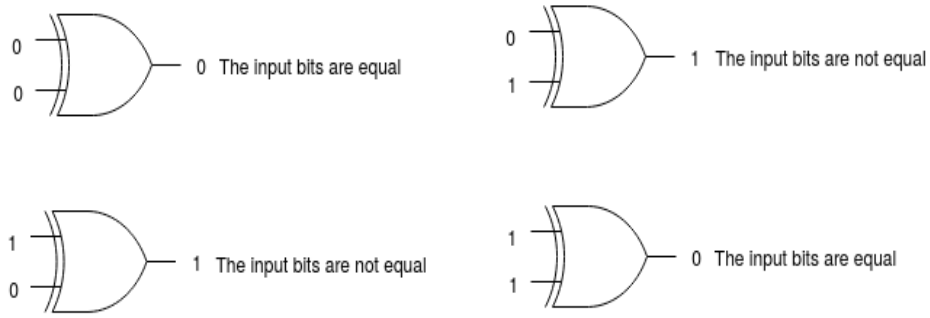


Figure 2.28 Basic comparator operation

An additional exclusive-OR gate is required to compare binary values comprising two bits each. Gate G_1 compares the two numbers′ least significant bits (LSBs), while gate G_2 compares the two most significant bits (MSBs), as seen in Figure 2.29. If the two numbers are equal, their corresponding bits are also equal, and each exclusive-OR gate′s output is a 0. If the corresponding sets of bits are not equal, the exclusive-OR gate output is set to 1.

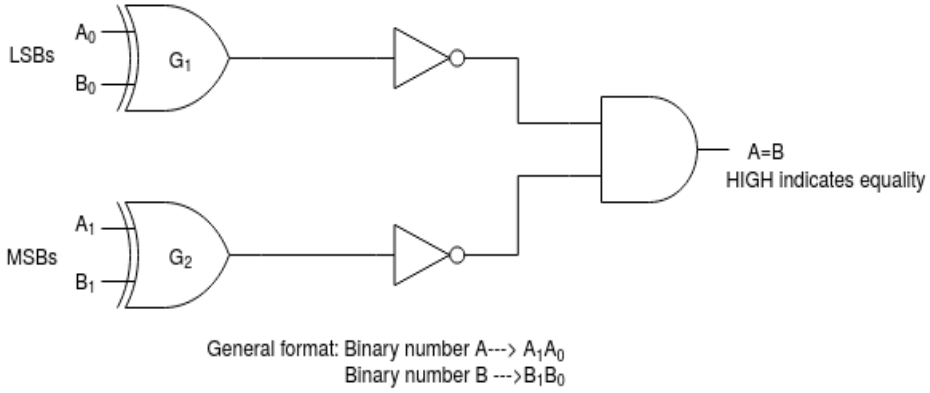


Figure 2.29 2-bits binary number comparison

Two inverters and an AND gate can be used to produce a single output representing the equality or inequality of two values, as shown in Figure 2.29. Each exclusive-OR gate′s output is inverted and applied to the AND gate′s input. When each exclusive-two OR′s input bits are identical. The numbers′ corresponding bits are equal, resulting in a 1 on both AND gate inputs and a 1 on the output. When the two numbers are not equal, one or both sets of corresponding bits are different, and a 0 appears on at least one AND gate input, resulting in a 0 on the AND gate′s output. As a result, the AND gate′s output indicates whether the two numbers are equal (1) or unequal (0).

Space for learners:

2.9.5.2 Inequality

Many IC comparators have additional outputs in addition to the equality output that show which of the two binary integers being compared is greater. As indicated in the logic symbol for a 4-bit comparator in Figure 2.30, there is an output that indicates when number A is larger than number B (A>B) and an output that indicates when number A is smaller than number B (A<B).

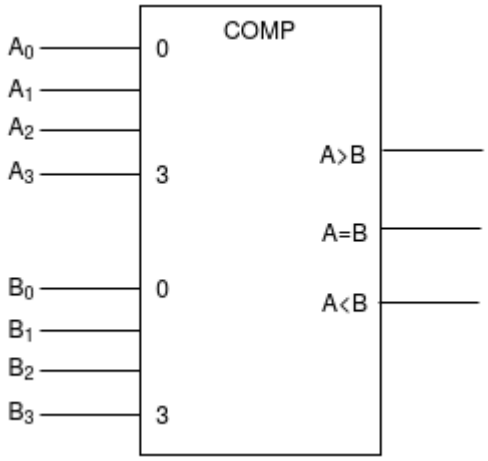


Figure 2.30 Logic symbol for a 4-bit comparator

To determine an inequality of binary numbers A and B, you first examine the highest-order bit in each number. The following conditions are possible:

1. If A_3=1 and B_3=0, number A is greater than number B.
2. If A_3=0 and B_3=1, number A is less than number B.
3. If A_3= B_3, then you must examine the next lower bit position for an inequality.

These three operations are valid for every bit position in the number. The general procedure used in the comparator is to check the inequality of the bit position, starting from the mostsignificant bit (MSB). When such an inequality is found, the relationship of the two numbers is established, and any other inequalities in lower-order bit positions must be ignored because it is possible for an opposite indication to occur; the highest-order indication must take precedence.

CHECK YOUR PROGRESS

State whether true or false

14. The exclusive-OR gate is a basic comparator.
15. The HIGH output will appear if we compare 11_2 and 11_2.
16. LSB stands for Least Significant Bit.

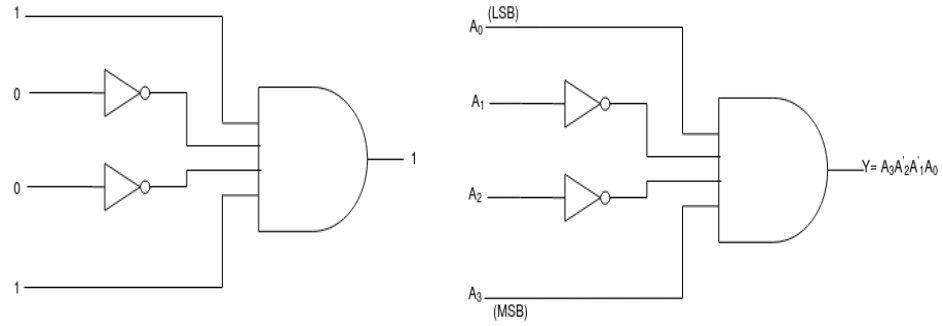
Space for learners:

2.9.6 Decoders

A decoder′s basic task is to identify the presence of a specific combination of bits (code) on its inputs and to signify that presence with a specific output level. A decoder contains n input lines to handle *n* bits and 1 to 2^n output lines to signal the presence of one or more *n*-bit combinations in its most basic form.

2.9.6.1 The Basic Binary Decoder

Assume you need to figure out when a binary 1001 appears on a digital circuit′s inputs. Because it provides a HIGH output only when all of its inputs are HIGH, an AND gate can be utilized as the basic decoding element. As a result, when the binary number 1001 occurs, you must ensure that all of the AND gate′s inputs are HIGH; this may be done by inverting the two middle bits (the 0s) as shown in Figure 2.31.



(a) (b)

Figure 2.31 Binary decoder

The logic equation for the decoder of Figure 2.31(a) is developed as illustrated in Figure 2.31(b). You should verify that the output is 0 except when A_0=1, A_1=0, A_2=0, and A_3=1 are applied to the inputs. A_0 is the LSB and A_3 is the MSB. In the representation of a binary number, the LSB is the right-most bit in a horizontal arrangement and the top-most bit in a vertical arrangement, unless specified otherwise. If the NAND gate is used in place of the AND gate in Figure 2.31, a LOW output will indicate the presence of the proper binary code, which is 1001 in this case.

Space for learners:

2.9.6.2 3 to 8 line Decoder

Figure 2.32 shows a decoder circuit with three inputs and $2^3 = 8$ outputs. It uses all AND gates, so the output is active high. Please note that for a given input code, the only valid output (HIGH) is the output corresponding to the decimal equivalent of the binary input code (for example, only when CBA = $101_2 = 5_{10}$, the O_5 output will become HIGH). The decoder can be referenced in various ways. It can be called a 3-8 line decoder because it has 3 input lines and 8 output lines. It can also be called a binary octal decoder or converter because it takes a 3-digit binary input code and activates one of the eight (octal) outputs corresponding to that code. It is also called a 1 of 8 decoder because only 1 of the 8 outputs is activated at a time.

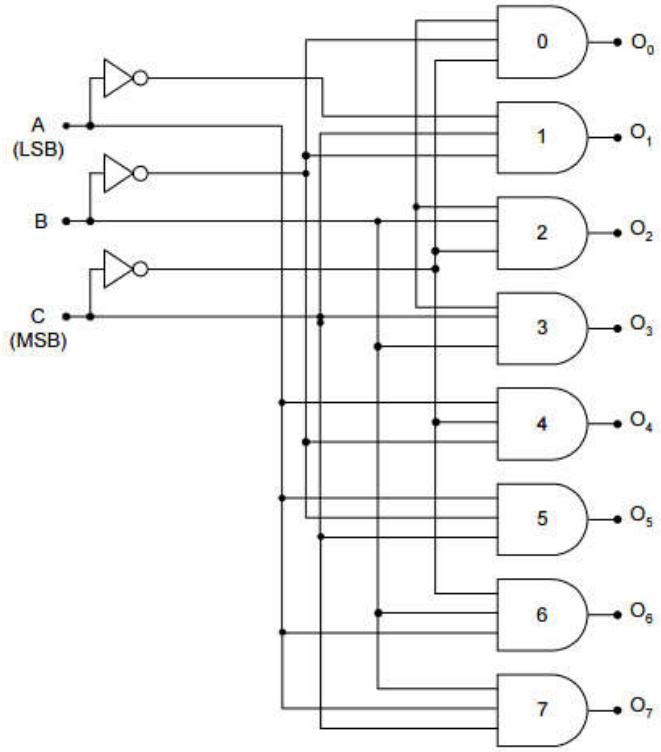


Figure 2.32 3-to-8-line decoder

Table 2.8: 3-to-8-line Decoder truth table with decoding function

Inputs			Decoding Function	Outputs							
C	B	A		O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7
0	0	0	C′B′A′	1	0	0	0	0	0	0	0
0	0	1	C′B′A	0	1	0	0	0	0	0	0
0	1	0	C′BA′	0	0	1	0	0	0	0	0
0	1	1	C′BA	0	0	0	1	0	0	0	0

Space for learners:

1	0	0	CB′A′	0	0	0	0	1	0	0	0
1	0	1	CB′A	0	0	0	0	0	1	0	0
1	1	0	CBA′	0	0	0	0	0	0	1	0
1	1	1	CBA	0	0	0	0	0	0	0	1

CHECK YOUR PROGRESS

17. An n-bit decoder can have ____ output lines
18. Determine the logic expression for the input 0111 by producing HIGH level on the output

2.9.7 Encoders

An encoder is essentially a combinatorial logic circuit that does the opposite of a decoder. An encoder accepts an active level from one of its inputs representing a number such as decimal or octal and converts it to a coded output such as BCD and binary. You can devise an encoder for encoding various symbols or characters. The process of converting the familiar symbols and numbers into a coded form is called encoding.

2.9.7.1 Decimal to BCD Encoder

As shown in Figure 2.33, this form of encoder has 10 inputs, one for each decimal digit, and four outputs that correspond to the BCD code. This is a simple ten-to-four line encoder.

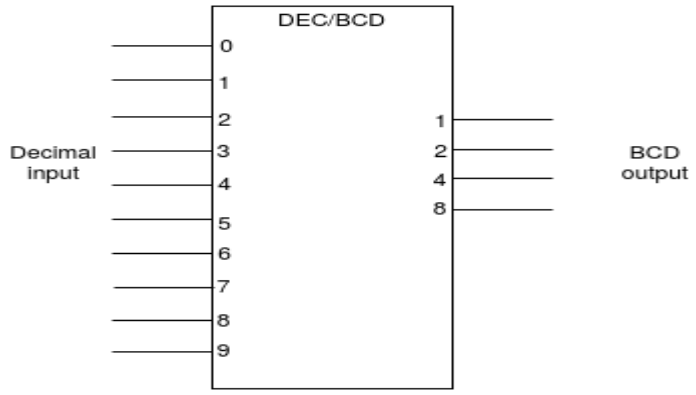


Figure 2.33 Logic symbol for a decimal to BCD encoder

Space for learners:

Table 2.7 lists the BCD (8421) code. To evaluate the logic, you can use this table to explore the relationship between each BCD bit and the decimal digits. For instance, the most significant bit of the BCD code, A_3, is always a 1 for decimal digit 8 or 9. An OR expression for bit A_3 in terms of the decimal digits can therefore be written as $A_3 = 8+9$. Bit A_2 is always a 1 for decimal digit 4, 5, 6 or 7 can be expressed as an OR function as follows:

$A_2 = 4+5+6+7$

Bit A_1 is always 1 for decimal digit 2, 3, 6, or 7 and can be expressed as

$A_1 = 2+3+6+7$

Finally, A_0 is always a 1 for decimal digits 1, 3, 5, 7, or 9. The expression for A_0 is

$A_0 = 1+3+5+7+9$

Table 2.9: Decimal to BCD encoder truth table

Decimal Digit	BCD code			
	A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Let us now use the logic expression we just generated to implement the logic circuitry required for encoding each decimal digit to a BCD code. Each BCD output is easily formed by ORing the relevant decimal digit input lines. Figure 2.34 depicts the basic encoder logic that results from these expressions. The circuit in Figure 2.34 has the following fundamental operation: The appropriate levels are displayed on the four BCD output lines when HIGH occurs on one of the decimal digit input lines. If input line 9 is HIGH (and all other input lines are LOW), for instance, this

Space for learners:

condition will result in HIGH on outputs A_0 and A_3 and LOW on outputs A_1 and A_2, which is the BCD code (1001) for decimal 9.

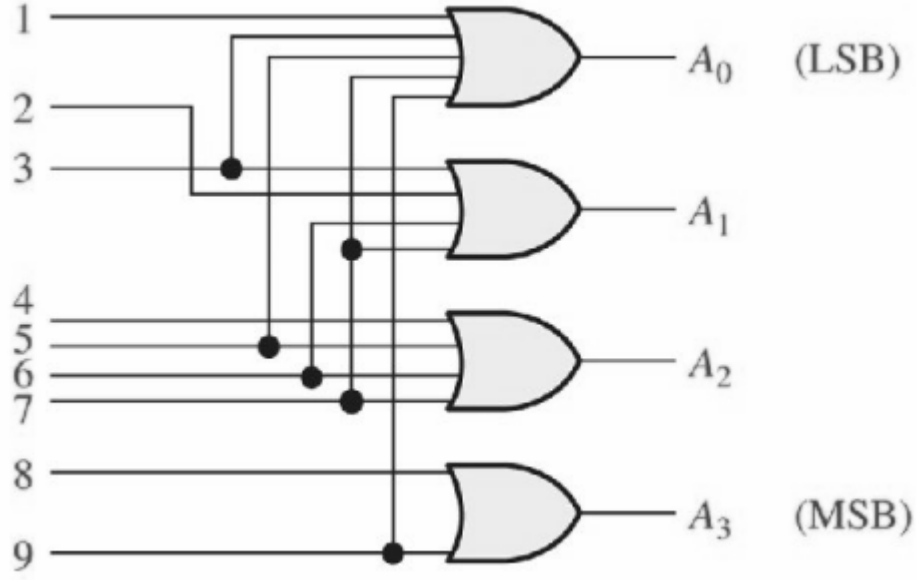


Figure 2.34 BCD encoder logic circuit

STOP TO CONSIDER

- Encoders perform the reverse operation of the decoders
- Encoders convert familiar symbols or numbers to coded forms.
- An encoder having 2^n input lines in the input will have n output lines in the output.

2.9.8 Multiplexers

The multiplexer (MUX) is a device that allows digital information from multiple sources to be routed to a single line for transmission over that line to a common destination. The basic multiplexer has several data input lines and one output line. It also has a data select input, allowing you to change digital data from any input to the line out. The multiplexer is also called a data selector.

Figure 2.35 shows a logic symbol for a 4-input multiplexer (MUX). Because there are two data select lines, any one of the four data input lines can be picked with two select bits.

Space for learners:

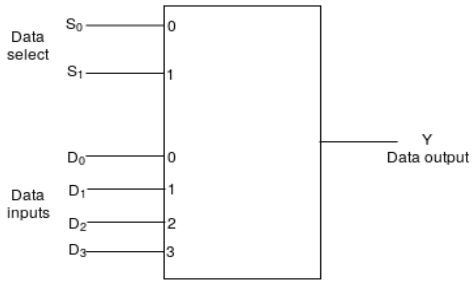


Figure 2.35 Logic symbol of a 4-input multiplexer

In Figure 2.35, a 2-bit code on the data-select (S) inputs will allow the data on the selected data input to pass through to the data output. If a binary 0 (S_1=0 and S_0=0) is applied to the data-select lines, the data on input D_0 appear on the data-output line. If a binary 1 (S_1=0 and S_0=1) is applied to the data-select lines, the data on input D_1 appear on the data output. If a binary 2 (S_1=1 and S_0=0) is applied, the data on D_2 appear on the output. If a binary 3 (S_1=1 and S_0=1) is applied, the data on D_3 are switched to the output line. A summary of this operation is shown in Table 2.8.

Table 2.10: 4 to 1 line multiplexer truth table

Data select inputs		**Input selected**
S_1	**S_0**	
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Let′s have a look at the logic circuits that this multiplexing process requires. The status of the selected data input is replicated in the data output. As a result, you can construct a logic expression for the output from the data input and the inputs you choose.

The data output is equal to D_0 only if S_1=0 and S_0=0; $Y= D_0S'_1S'_0$
The data output is equal to D_1 only if S_1=0 and S_0=1; $Y= D_1S'_1S_0$
The data output is equal to D_2 only if S_1=1 and S_0=0; $Y= D_2S_1S'_0$
The data output is equal to D_3 only if S_1=1 and S_0=1; $Y= D_3S_1S_0$

When these terms are ORed, the total expression for the data output is

Space for learners:

$Y = D_0S'_1S'_0 + D_1S'_1S_0 + D_2S_1S'_0 + D_3S_1S_0$

The implementation of this equation requires four 3-input AND gates, a 4-input OR gate, and two inverters to generate the complements of S_1 and S_0 as shown in Figure 2.36. Because data can be selected from any one of the input lines, this circuit is also referred to as a data*selector*.

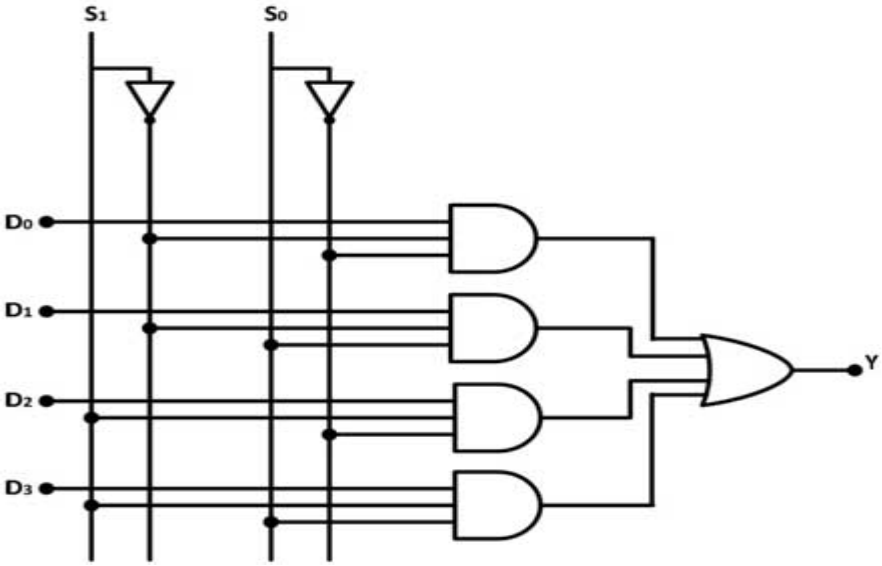


Figure 2.36 Circuit diagram of 4-to-1 multiplexer

STOP TO CONSIDER

- Multiplexers are also known as data selectors.
- A 4-input data lines multiplexer has two select lines.
- A 8-input data lines multiplexer has three select lines.
- A 2^n input data lines multiplexer has n select lines.
- The output depends on the input data and select lines bits.

2.9.9 Demultiplexer

The demultiplexer (DEMUX) basically reverses the multiplexing function. It takes digital information from one line and distributes it to a specified number of output lines. Therefore, the demultiplexer is also called a data distributor. As you will learn, the decoder can also be used as a demultiplexer.

Space for learners:

A 1-to-4-line demultiplexer (DEMUX) circuit is shown in Figure 2.37. The data-input line is connected to all AND gates. Only one gate is enabled at a time by the two data-select lines, and data on the data-input line passes via the selected gate to the associated data output line.

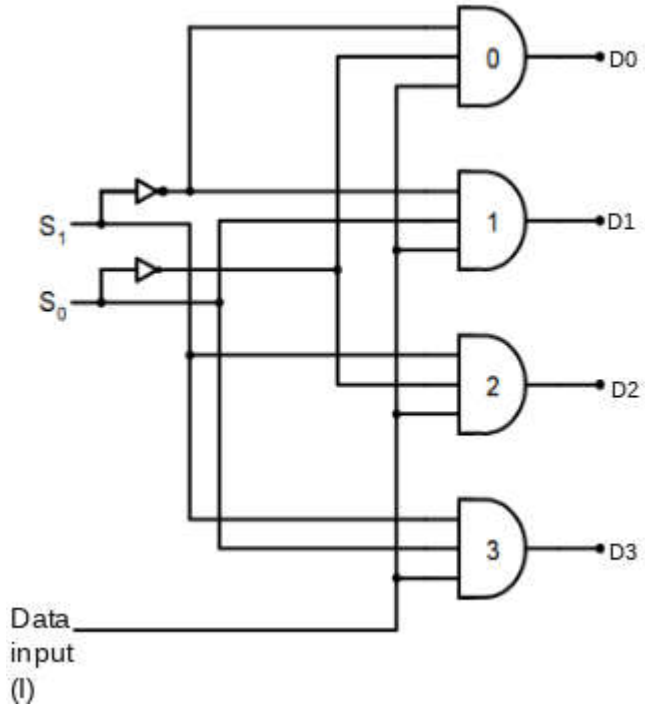


Figure 2.37 Circuit diagram of a 1-to-4 line demultiplexer

Table 2.11: 1 to 4 line demultiplexer truth table

Select code		Outputs			
S_1	S_0	D_3	D_2	D_1	D_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

The algebraic expressions for the functions shown in Table 2.9 are:

$D_0 = IS'_1S'_0$

$D_1 = IS'_1 S_0$

$D_2 = IS_1S'_0$

$D_3 = I S_1 S_0$

Space for learners:

CHECK YOUR PROGRESS

19. Demultiplexer basically _____ the multiplexing function.
20. In demultiplexer only _____ gate is enabled at a time by the data-select lines.
21. Data on the data-input line passes via the selected gate to the associated data _____ line.

2.10 SUMMING UP

- AND-OR logic produces an output expression in SOP form
- AND-OR-Invert logic produces a complemented SOP form, which is actually a POS form.
- Combinational circuits are designed either using Boolean expression or truth tables.
- The operational symbol for exclusive-OR is $\oplus$. An exclusive-OR expression can be stated in two equivalent ways: $AB'+A'B=A\oplus B$
- To do an analysis of a logic circuit, start with the logic circuit, and develop the Boolean output expression or the truth table or both.
- Implementation of a logic circuit is the process in which you start with the Boolean output expressions or the truth table develop a logic circuit that produces the output function.
- Minimization of Boolean expressions should be tried before implementing a logic circuit.
- NAND and NOR gates are called universal logic gates.
- All NAND or NOR logic diagrams should be drawn using appropriate dual symbols so that bubble outputs are connected to bubble inputs and non-bubbleoutputs are connected to non-bubbleinputs.
- When two negation indicators (bubbles) are connected, they effectively cancel each other.
- The basic logic functions are comparison, arithmetic, code conversion, decoding, encoding, data selection, storage, and counting.
- Addition, subtraction, multiplication, division, encoding, decoding, multiplexing, demultiplexing, etc. are the functionalities of the combinational logic circuits.

Space for learners:

- To perform the addition operation half-adder, full-adders, parallel adders are used.
- Half-adders can be combined to design the full-adders.
- Ripple carry and look-ahead carry are the examples of carries seen in the adder circuits.
- Comparator circuits are used to compare any two binary numbers and MSB are given more precedence while comparing two binary numbers.
- Subtractor circuits also have a similar design like the adder circuits.
- Encoder and decoder are the code converter circuits, they perform reverse operation with each other.
- Multiplexer and demultiplexer are data selector and data distributor, they also perform reverse operation with each other.

Space for learners:

2.11 KEY TERMS

- **SOP:** Sum of Product expressions
- **POS**: Product of Sum expressions
- **Half-adders:** add two binary numbers and produce sum and carry in the output.
- **Full-adders:** add two binary numbers with input carry and produce sum and carry in the output.
- **Half-subtractor**: subtract binary numbers and produce difference and borrow out.
- **Full-subtractor**: subtract two binary numbers with borrow in and produce difference and borrow out.
- **Comparators:** Compare two binary numbers
- **Decoders:** Detect the specific combination of bits in the input.
- **Encoders:** An encoder converts understandable alphabet to numbers into the coded forms.
- **Multiplexers:** Multiplexers are the data selectors. Multiplexers transmit data coming from different sources over a single line.
- **Demultiplexers:** Demultiplexers show the reverse operation of multiplexers. It takes digital data from a single line and distributes them in several lines.

2.12 ANSWERS TO CHECK YOUR PROGRESS

1. Product of Sum
2. Sum of Product
3. A′B+AB′
4. One (one 3-input AND gate)
5. Two
6. Two
7. Four
8. Three
9. False
10. False
11. True
12. True
13. True
14. True
15. True
16. True
17. 2^n
18. $I'_3I_2I_1I_0$
19. Reverses
20. One
21. Output

2.13 POSSIBLE QUESTIONS

Short answers type questions

1. Determine the output (1or 0) of a 4-variable AND-OR-Invert circuit for each of the following input conditions:
    a. A=1, B=0, C=1, D=0
    b. A=1, B=1, C=0, D=1
    c. A=0, B=1, C=1, D=1
2. Draw the logic diagram for an exclusive-NOR circuit.
3. Determine the output of an exclusive-OR gate for each of the following input conditions:
    a. A=1, B=0
    b. A=1, B=1
    c. A=0, B=1
    d. A=0, B=0
4. Implement the following boolean expressions:
    a. Y = ABC+AB+AC
    b. Y = AB(C+DE)
5. Reduce *Question 4* to minimum SOP form.

Space for learners:

6. Use NAND and NOR gates or combination of both to implement the following logic expressions:
    a. Y = A′B+CD+(A+B)′(ACD+(BE)′)
    b. Y = ABC′D′+DE′F+(AF)′
7. Find out the full adders input for the following set of outputs:
    a. $\Sigma = 0$, $C_{out}=0$
    b. $\Sigma = 1$, $C_{out}=0$
    c. $\Sigma = 1$, $C_{out}=1$
8. Implement the expression Y = ((A′+B′+C′)DE)′ using NAND logic.
9. Implement the expression Y = ((A′B′C′)+(D+E))′ using NOR logic.
10. Develop a logic circuit that produces a 1 on its output only when all three inputs are 1s or when all three inputs are 0s.

Long answers type questions:

1. Define combinational logic circuits. Explain the various components used to design the combinational logic circuits.
2. Design combinational logic circuit for the following expression

    Y = AB(C+DEF)+CE(A+B+F)
3. Develop the truth table for a certain 3-input logic circuit with the output expression

    Y = AB′C+A′BC+A′B′C′+ABC′+ABC
4. Develop truth for the following expressions and draw the circuits:
    a. A′B+ABC′+(AC)′+AB′C
    b. P′+QR′+ SR+PQ′R
5. From the following truth table draw the logic circuit in minimized form

Inputs			Output Y
A	B	C	
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1

Space for learners:

1	1	0	0
1	1	1	1

6. Design a 6-bit parallel adder.
7. Design a 4-to-2 line encoder using logic gates.
8. Design a 8-to-1 line multiplexer using logic gates
9. Design a 4-to-16 line decoder using logic gates
10. Design a 1-to-8 line demultiplexer using logic gates
11. Design an adder-subtractor circuit.

2.14 REFERENCES AND SUGGESTED READINGS

1. Mano, M. Morris, *Digital Logic and Computer Design,* Pearson Education.
2. Mano, M. Morris, *Computer System Architecture,* Pearson Education.
3. Jain, R. P., *Modern Digital Electronics,* Mc Graw Hill India.

---×---

Space for learners:

UNIT-3: COMPUTER ARITHMETIC

Space for learners:

Unit Structure:

3.1INTRODUCTION

A separate section in central processing unit used to execute arithmetic operations is called arithmetic processing unit. The arithmetic instructions are performed generally on binary or decimal data. Fixed-point numbers are used to represent integers or fractions. We can have signed or unsigned negative numbers. Fixed-point addition is the simplest arithmetic operation. In digital computers data is manipulated by using arithmetic instructions. Data is manipulated to produce results necessary to give solution for the computation problems. The addition, subtraction, multiplication and division are the four basic arithmetic operations. We can derive some other operations by using these four operations.

3.2 UNIT OBJECTIVES

This unit is an attempt to give an idea of multiplication and division of numbers in digital computer. After going through this unit you will be able to-

- understand the multiplication operation
- understand the division operation
- explain the floating-point arithmetic operation

3.3 MULTIPLICATION OF NUMBERS

Multiplication of two fixed point unsigned binary numbers can be done by a process of successive shift and add operations. But the multiplication of two fixed point signed binary numbers in 2's complement representation requires special consideration.

3.3.1 Multiplication of Unsigned Numbers

Multiplying unsigned numbers in binary is quite easy. We already know that with 4 bit numbers we can represent numbers from 0 to 15.

For Multiplication of binary numbers only we have to remember the number facts: 0*1=0 and 1*1=1 (this is the same as a logical "and").

Multiplication is different than addition. Multiplication of an *n* bit number by an *m* bit number results in an *n*+*m* bit number. Let's discuss with an example where n=m=4 and the result of multiplication is 8 bits:

Space for learners:

Example 1:

Decimal	Binary
10 x 6 60	1010 (Multiplicand) x 0110(Multiplier) 0000 1010 Partial Product 1010 +0000 0111100 (Product)

In this case of binary multiplication the result is 7 bit, which can be extended to 8 bits by adding a 0 at the left.

Example 2:

Decimal	Binary
7 x 6 42	0111 x 0110 0000 0111 0111 +0000 0101010

3.3.2 Multiplication of Signed Numbers

For multiplying binary integers in signed 2's complement representation requires special consideration.

Example 3:

Decimal	Binary
7 x -6 -42	0111 x 1010 (2's complement) 0000 0111 0000 +0111 1000110 (The result is incorrect) So, there is an error

Space for learners:

Solution: We must sign extend to the product bit width. The additional values out to the MSB position are called sign extension.

Decimal Number	3 bits	4 bits	8 bits	16 bits
6	110	0110	0000 0110	0000 0000 0110
-6	110	1110	1111 1110	1111 1111 1110
7	111	0111	0000 0111	0000 0000 0111
-7	111	1111	1111 1111	1111 1111 1111

As we know that multiplication of two 4 bit numbers results in8 bits. So for signed multiplication of two 4 bit numbers we must sign extent the numbers to the product bit width i.e, 8 bits.

Example 4:

Decimal	Binary
7 x -6 -42	0111 x 1010 After Sign extension 00000111 x 11111010 00000000 00000111 00000000 00000111 10000111 00000111 00000111 +00000111 |11010110 (2's complement of 42) Stop after 8 bits So the result is correct

Space for learners:

CHECK YOUR PROGRESS

1. Do the binary multiplication of (-7) and (-6)
2. Do the binary multiplication of (-7) and (-6)

Space for learners:

3.3.3 Hardware Implementation of Multiplication Operation

The multiplier and multiplicand are stored in two registers Q and M. A third register A is initially set to 0. A 1-bit register C is used to store the carry bit resulting from addition. Control logic reads the bit of the multiplier one at a time. The multiplicand is added to the register A if Q_0 is 1 and then stored the result back to register A with C bit is used to store carry. Then all the bits of CAQ are shifted one position right. No addition is performed if Q_0 is 0. The process is repeated for each bit of the multiplier. The resulting 2n bit product is the contain of QA register

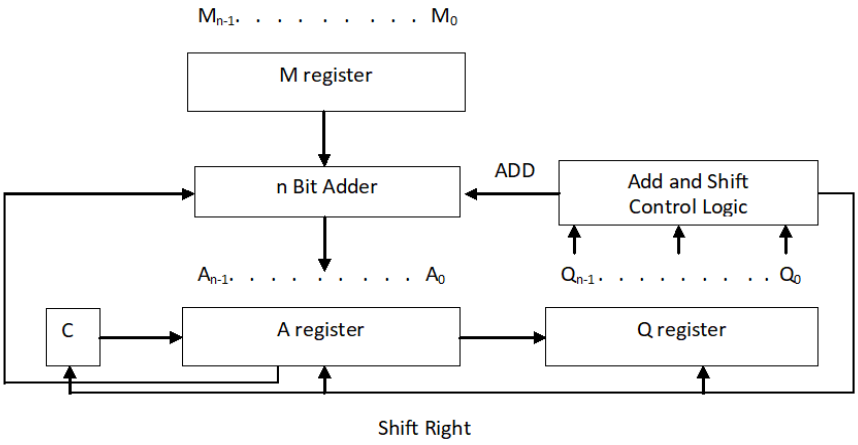
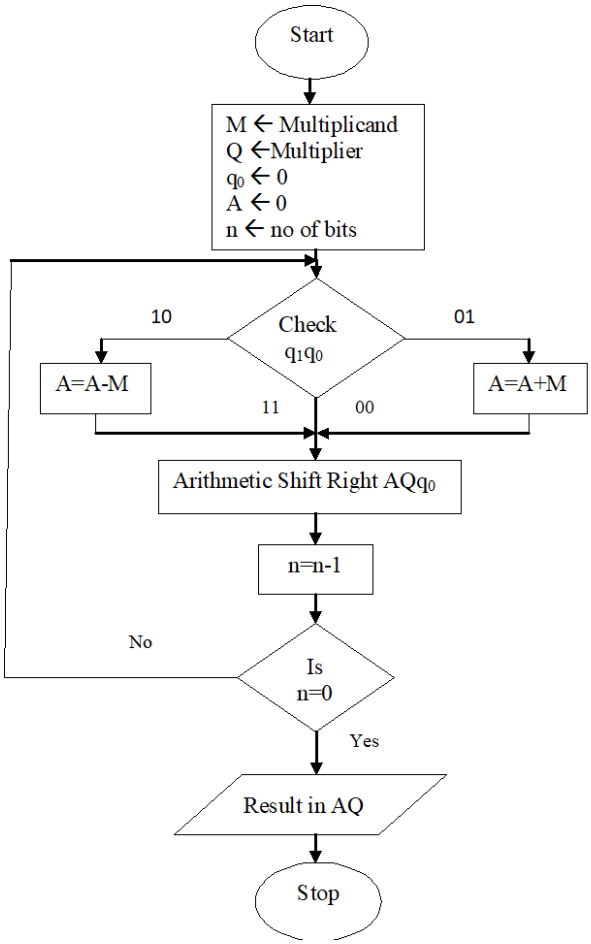


Figure 1: Hardware for Multiplication Operation

3.3.4 Booth's Multiplication Algorithm

Booth's algorithm gives a procedure for multiplying binary integers in signed 2's complement representation.

Figure 2: Flow chart of Booth's Algorithm for multiplication of signed numbers



Example 5:

Decimal
-6 (Multiplicand) x 3 (Multiplier) -18 (Product)

Here, number of bits (n) required for this calculation is 4 bits (1 bit to represent the sign and 3 bits to represent the numbers). Since 6 can be represented using 3 bits and negative sign is represented using 1 bit.

Space for learners:

So, the size of registers M, Q, A(Accumulator) is 4 bits and register q_0 is 1 bit.

$M=(-6)_{10}$

= 2's complement of $(0110)_2$

$= (1010)_2$

$-M = (0110)_2$

$Q = (3)_{10} = (0110)_2 = Q\ (q_4q_3q_2\ q_1)$

Operations:

(i) If q_1q_0 bits are 1 0 then then do A = A – M = A + (-M)
(ii) If $q_1\ q_0$ bits are 0 1 then then do A = A + M
(iii) Otherwise Arithmetic Shift Right of (A Q q_0) is done.

Suppose (A Q q_0) = (0 1 1 0 0 0 1 1 0)

Then ASR will yield the result = (0 0 1 1 0 0 0 1 1). Here sign bit(MSB) is restored and all bits (including the sign bit) is shifted one position right.

TABLE 1: Multiplication of Example 5 using Booth's Algorithm

n	A	Q (q_4 q_3 q_2 q_1)	q_0	Action/Comment
4	0 0 0 0	0 0 1 1	0	Initialization
	0 1 1 0	0 0 1 1	0	A = A – M
3	0 0 1 1	0 0 0 1	1	ASR of (A Q q_0)
2	0 0 0 1	1 0 0 0	1	ASR of (A Q q_0)
	1 0 1 1	1 0 0 0	1	A = A + M
1	1 1 0 1	1 1 0 0	0	ASR of (A Q q_0)
0	1 1 1 0	1 1 1 0	0	ASR of (A Q q_0)

Result is content of AQ i.e, 1 1 1 0 1 1 1 0

Here, LSB (Sign bit) is 1 so the result is –ve.

Space for learners:

Therefore result: - (2's complement of 1 1 1 0 1 1 1 0) = - (0 0 0 0 1

SAQ

1. Do the binary multiplication of -7 and 3 using Booth's Algorithm.

3.4 DIVISION OPERATION

The division operation involves repetitive shifting and addition or subtraction.

First, the bits of the dividend are examined from left to right to search a number greater than or equal to the divisor. Until this event occurs, 0s are placed in the quotient from left to right. When such a number is found and the divisor divides the number, a 1 is placed in the quotient and the divisor is subtracted from the partial dividend. The result is referred to as a partial remainder. In the subsequent cycle, additional bits from the dividend are appended to the partial remainder until the result is greater than or equal to the divisor. The divisor is subtracted from this number to produce a new partial remainder. The process continues until all the bits of the dividend are exhausted.

CHECK YOUR PROGRESS

1. Divide 1001 by 11
2. Divide 111000 by 111

3.5 FLOATING-POINT ARITHMETIC OPERATIONS

Arithmetic operations on floating point numbers are addition, subtraction, multiplication and division.

A floating point number can be represented as m x r^e, where m is called mantissa, r is called radix and e is called exponent part. In computer memory two registers: mantisa and exponent is used to represent a floating point number.

Space for learners:

For example, the decimal number 423.75 can be represented in a register with m=42375 and e=3 and is interpreted to represent the floating point number

. 42375 x 10^3

Space for learners:

3.5.1 Addition/Subtraction of two Floating point numbers:

Steps to add/subtract two floating point numbers are as discussed below:

(i) Alignment: Compare the magnitudes of two exponents and align the number with smaller magnitude of exponents

(ii) Addition/Subtraction: Addition or subtraction is done following the addition or subtraction rules.

(i) Normalize the result: If MSB of mantissa part of the product is 1, the product is already normalized. If it is 0 underflow occurs and the mantissa of the product is shifted left and decrement the exponent value. If overflow occurs, mantissa is shifted right and exponent is incremented

Example 6: Add 1.1010 x $2^4$and 1.101x 2^2

Solution:

Step (i), Here 1.101 x 2^2 is aligned to 0.01101 x 2^4.

Step (ii), Add the two numbers 1.1010 x 2^4 and 0.01101 x 2^4 to get 10.00001 x 2^4

Step (iii), Result = 10.00001 x 2^4. So, overflow in the result.

After normalization the result is 0.1000001 x 2^6

SAQ

1. Add 1.1100 x 2^4and1.100 x 2^2.

Space for learners:

3.5.2 Multiplication of two Floating point numbers:

(ii) **Add the exponents:** Exponents of two numbers are added to get the exponent of the product.

(iii) **Multiply the mantissas:** Multiplication of mantissas are done following multiplication rule.

(iv) **Normalize and round the result:** Overflow cannot occur during multiplication. If MSB of mantissa part of the product is 1, the product is already normalized. If it is 0, then the mantissa of the product is shifted left and decrement the exponent value.

Example 7: Multiply 1.000×2^{-2} and 1.010×2^{-1}

Solution:

Step (i), (-2) + (-1)=-3, this is the exponent value of the product.

Step (ii), Multiply the mantissas: 1.000 x (1.010) = 1.010000

Step (iii), Result = 1.010000×2^{-3}

Under flow in the result. So after normalization result is = 0.10000×2^{-2}

STOP TO CONSIDER

In floating point multiplication if either operand is equal to zero, the product is set to zero and operation is terminated. Procedure for arithmetic operations on floating point numbers is different than integers.

3.6 SUMMING UP

- Procedure to do multiplication of signed and unsigned number is different.
- Multiplication of two fixed point unsigned binary numbers can be done by a process of successive shift and add operations.
- Multiplication of signed numbers can be done using Booth's Algorithm.
- The multiplier and multiplicand are stored in two registers Q and M.
- The division operation involves repetitive shifting and addition or subtraction.
- Arithmetic operations on floating point numbers is done in a different way.

Space for learners:

3.7 ANSWER TO CHECK YOUR PROGRESS

1. State the Booth's algorithm for multiplication of two numbers, Draw a block diagram for the implementation of the Booth's algorithm for determining the product of two 8−bit signed numbers.
2. Multiply 1.1100×2^{-3} and 1.01×2^{2}

3.8 POSSIBLE QUESTIONS

1. Perform binary multiplication of -8 and -3 using sign extension method
2. Perform binary multiplication of 9 and -4 using Booth's Algorithm.
3. Write the steps of Booth's Algorithm.
4. Discuss the hardware implementation of multiplication operation.
5. Write the steps of division operation.
6. How the alignment and normalization is done in addition of two floating point numbers?

3.9 REFERENCES AND SUGGESTED READINGS

1. Computer System Architecture, M. Morries Mano

---×---

UNIT4: REGISTERTRANSFER LANGUAGE AND PROCESSOR LOGIC DESIGN

Unit Structure:

Space for learners:

4.1 INTRODUCTION

As you know, all the operations or instructions in a digital computer are carried out by a processor with the help of various other interconnected modules. The elementary operations are also called as micro operations that are performed on the data stored on the processor registers. This unit contents the fundamentals of micro operations and the language used to represent various micro operations which is known as Register Transfer Language (RTL),

the concept of Data paths and other significant parts of the Central Processing Unit (CPU) such as the Arithmetic and Logic Unit (ALU) where you will be able to understand the functioning of the Arithmetic Circuit and Logic Circuit and lastly in the Control Unit (CU) part you will be able learn the design of CU and its functionalities and also about alternative designs of the CU - hardwired and micro programmed control unit.

Space for learners:

4.2 UNIT OBJECTIVES

After completing this unit, you will be able to learn:

- The concepts Micro operation and representation of Register and uses of Register Transfer Language (RTL)
- Concepts of one-bus, two-bus and three-bus organization
- Concepts related to ALU (basic design of Arithmetic Circuit and Logic Circuit)
- Concept of the Control Unit (Micro programmed and Hardwired Control Unit)

4.3 REGISTER TRANSFER LANGUAGE

The internal hardware organization of a digital computer exhibits an interconnection of digital modules such as registers, decoders, arithmetic logic, and control logic etc. The complete digital system is interconnected with data and control paths commonly known as bus. The elementary operations performed by the CPU on the data stored in one or more registers are termed as micro operations. clear, count, load and shift are some examples of such micro operations.

Various categories of micro operations:

The most commonly used micro operations in a digital computer are listed below-

- **Register transfer micro operations:** The micro operations that are used to transfer binary information among various registers.

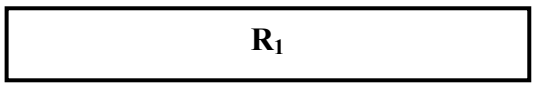
- **Arithmetic micro operations:** The micro operations that are used to perform various arithmetic operations (add, subtract, increment, decrement) on arithmetic data stored in the registers.
- **Logic micro operations:** The micro operations that are used to perform logical operations (AND, OR, NOT) on the data stored in the registers.
- **Shift micro operations:** The micro operations that are used to perform either left or right shift operations (logical, arithmetic, circular) on the data stored in the registers.

The **Register Transfer Language (RTL)** is the representation system used to describe the sequence of micro operations in a symbolic form. The term register transfer refers to the transfer of binary information among the registers via a common path or bus.

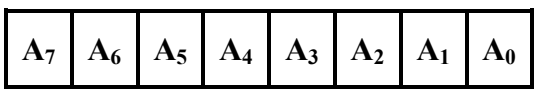
Space for learners:

4.3.1 Representation of Registers

In a digital computer system the registers are represented using upper case letters (and followed by a numeral sometimes). PC- Program Counter, IR- Instruction Register etc. are examples of special purpose registers and R_1, R_2, R_3… etc. are examples of general purpose registers. A register is represented by a rectangular box containing the name inside and the bit numbers can be marked at the top of the box starting from left to right as shown in figure 4.1 (a) & (b). Each bit of the data stored in the register can be represented as shown in the figure where each individual bit is assigned a letter along with a numeral subscript that indicates the position of the bit. Considering a 16-bit register the bits numbered from 0 to7 are termed as low byte (L) while the bits from 8 to 15 are termed as high byte (H) of the register as shown in figure 4.1 (c) & (d).



(a)



(b)

A_{15} A_0

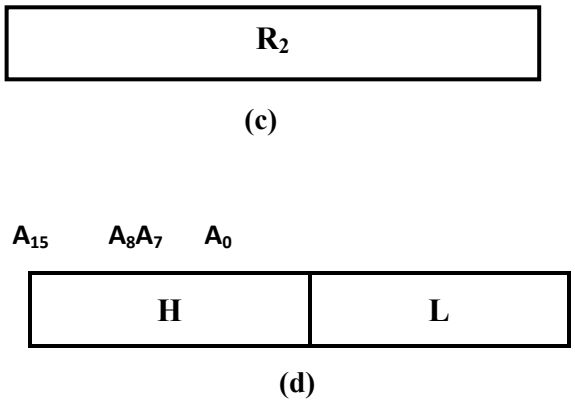


Fig4.1: Block diagram of Register

4.3.2 Register Transfer Representation

The transfer of contents from one register to another register can be shown with the help of replacement operator (←). For example, transfer of data from register R_1 to register R_2can be symbolically expressed using the following statement in RTL.

$R_2 \leftarrow R_1$

When this statement gets executed contents of R_2 will be replaced by contents of R_1, but the contents of R_1 remains unchanged. The execution of this statement can be controlled by putting some control condition also. That means when the control condition satisfies then only the transfer takes place, otherwise not. This is shown in the following expression:

`If (P = 1)  then (`$R_2 \leftarrow R_1$`)`

Here, `P=1` is the control statement or control function which is a Boolean variable.

The statement can also be written as

`P:` $R_2 \leftarrow R_1$

The colon (:) separates the control condition from the rest.

The hardware implementation of the statement `P:` $R_2 \leftarrow R_1$ is shown below:

Space for learners:

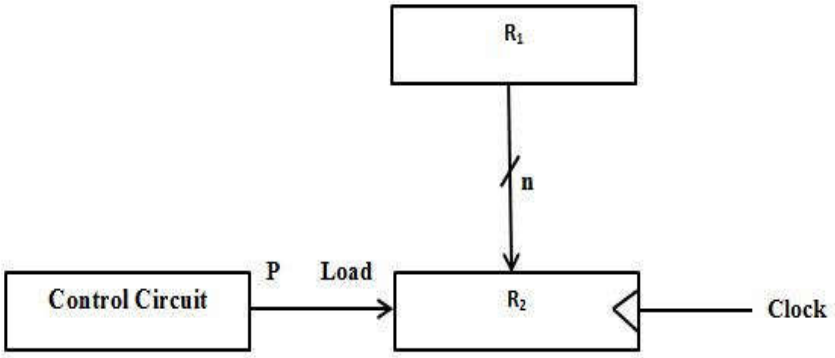


Fig4.2: Hardware implementation of $P: R_2 \leftarrow R_1$

4.3.3 RTL Representation of Memory Transfers

Data flow from memory to external environment is known as *memory read* operation while data from external environment is stored in memory is referred to as *memory write* operation. In RTL, the memory word is symbolized by the letter *M* followed by square brackets [ ] having the address of the memory word. For example, transferring a data word *M* from memory whose address is stored in Address Register (AR) to Data Register (DR) can be symbolized as:

$Read: DR \leftarrow M[AR]$

Similarly, the write operation can be symbolized as:

$Write: M[AR] \leftarrow R_1$

which means transfer of data from register R_1 to the memory word *M* whose address is stored in the Address Register (AR).

Check Your Progress

1. The operations executed on data stored on registers are called _____.
   a) macro operations b) micro operations
   c) Byte Operations d) Bit Operations
2. Which of the following register transfer statements is correct?
   a) P, $R_1 \leftarrow R_2$ b) P : $R_1 \leftarrow R_2$: $R_3 \leftarrow R_4$
   c) P : $R_1 \leftarrow R_2$, $R_3 \leftarrow R_4$ d) None of the these
3. What does the following transfer statement indicate?
   $R_2 \leftarrow M[R_1]$
   a) Read a memory word at the address stored in R_1
   b) Read a memory word at the address stored in R_2
   c) Write a memory word at the address stored in R_1
   d) Write a memory word at the address stored in R_2

Space for learners:

4. *State TRUE or FALSE:*
   a) Considering a 16-bit register the bits numbered from 0 to7 are termed as low byte (L).
   b) Shift micro operations are used to perform various logical (AND, OR, NOT) operartions.
   c) A register transfer can't occur unless the specified control condition becomes true.

Space for learners:

4.4 DATAPATH

The Central Processing Unit of a digital computer can be divided into a data section and a control section. The data section, also called as data path, contains the registers and the Arithmatic Logic Unit (ALU) and the Buses. There are three types of buses- Address bus, Data bus and Control bus. The data path is accomplished for performing certain operations on data items stored in various memory units. The control section is basically comprised of the control unit, which issues various control signals to the data path. Internal data (which may be data, instructions or addresses) transfers are carried out via local buses. Externally, data transfer from registers to memory and Input-Output devices, often carried out by a system bus. The local bus organization to perform internal data transfer among registers and the ALU may be of different organizations like one-bus, two-bus, or three-bus organization.

4.4.1 One-Bus Data path

In this organization, the CPU registers and the ALU use a single bus to transfer data. This bus organization is least expensive and simplest in design, but it restricts the amount of data transfer that can be done in the same clock cycle, which results in decrease of overall performance of the system. Figure 4.3 shows a one-bus data path organization comprising of a set of general-purpose registers, a memory address register (MAR), a memory data register (MDR), an instruction register (IR), a program counter (PC), and an ALU, all are interconnected via a single data path.

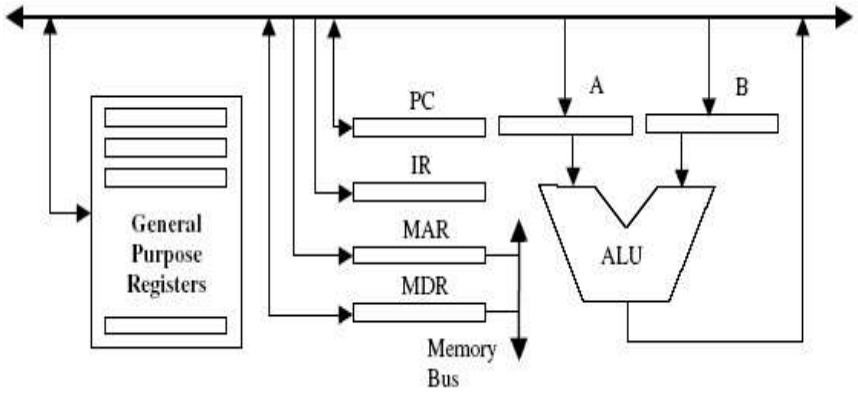
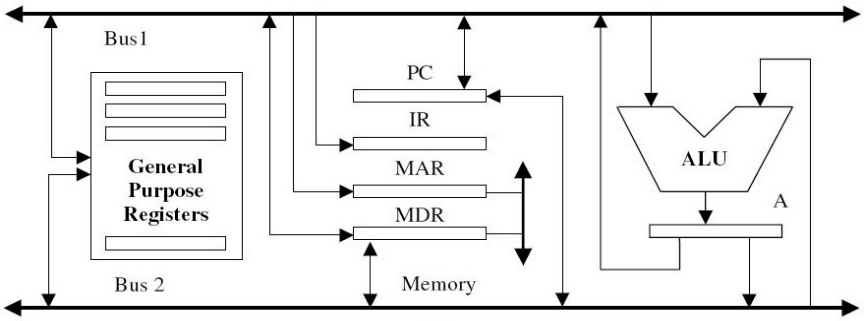


Fig4.3: One-bus data path

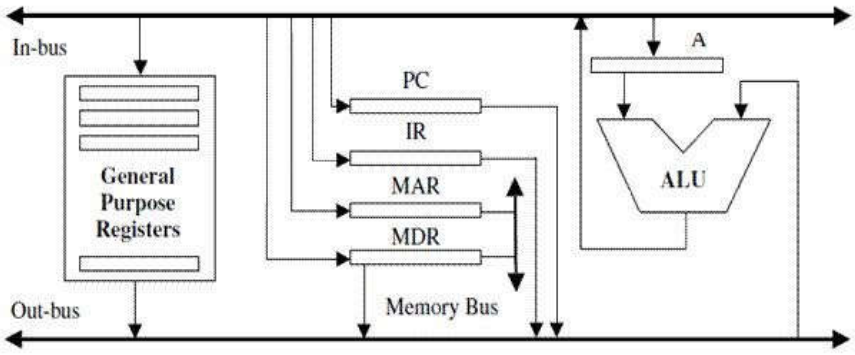
4.4.2 Two-Bus Data path

In two-bus organization, two buses are used which results a faster performance than the one-bus organization. In this case, the general-purpose registers are connected to both the buses. Data can be transferred from two different registers to the input point of the ALU at the same time. Therefore, an operation having two operands can fetch both operands in the same clock cycle. There may be a need of an additional buffer to hold the output of the ALU when the two buses remain busy in carrying the two operands. Figure (4.4-a) shows a two-bus organization. There may be another implementation of two bus organization where one of the buses is dedicatedly used for moving data into registers (in-bus), while the other bus is dedicatedly used for transferring data out of the registers (out-bus). For this purpose, the buffer register may be used additionally, as one of the ALU inputs, to hold one of the operands. The ALU output can be connected directly to the in-bus, which transfers the result to one of the registers. A two-bus organization with in-bus and out-bus is shown in Figure (4.4-b).



(a)

Space for learners:



(b)

Fig4.4 :(a) Two bus datapath(b) Two bus datapath with in-busand out-bus

4.4.3 Three-Bus Datapath

In case of three-bus organization, two buses may be used as source buses whereas the third bus is used as destination. The source buses are used to transfer data out from registers (out-bus), and the destination bus may be used to transfer data into a register (in-bus). Each of the two out-buses is connected to an ALU input point and the output of the ALU is connected directly to the in-bus. As we have more buses in this organization, more data can be transferred within a single clock cycle. However, increasing the number of buses also increases the complexity as well as cost of the hardware. Figure (4.5) shows the organization of a three-bus data path.

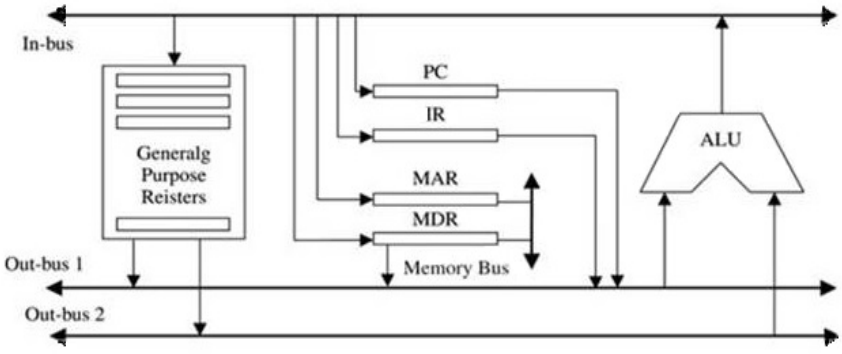


Fig 4.5 : Three-bus data path

Space for learners:

Check Your Progress

5. The data section of the CPU is also known as _____.
6. In-bus, out-bus organization is related to _____.
   a) one-bus data path b) two-bus data path
   c) three-bus data path d) None of these.
7. In _____ all the General Purpose Registers (GPR), Special Purpose Registers (SPR) and the ALU are connected via a single data path.
   a) one-bus data path b) two-bus data path
   c) three-bus datapath d) None of these.
8. Two-bus datapath is more efficient than Three-bus datapath. (*State TRUE or FALSE)*

Space for learners:

4.5 ALU DESIGN

The arithmetic and logic unit (ALU) is a combinational circuit in a digital computer which performs the following operations-

- *Arithmetic operations* such as add, subtract, increment and decrement etc.
- *Logic operations* such as AND, OR, XOR and compliment etc.
- *Bit Shifting operations* such as logical left and right shift used for multiplication purpose.

Therefore, we can say ALU is the combination of arithmetic unit, logic unit and shift unit all the three circuits together. It is usually a part of the central processing unit (CPU). Many CPUs have separate units for arithmetic operations (Arithmetic Unit-AU) and for logic operations (Logic Unit-LU).

4.5.1 Arithmetic Circuit

The 4-bit arithmetic circuit which is shown in Figure 4.6 is able to perform different basic arithmetic operations such as add, subtract, increment and decrement. It employs parallel full adders (FA) to perform these operations depending on the inputs. The select inputs S_0 and S_1 are used to provide different inputs to the multiplexers (MUX) present in the circuit in order to obtain different arithmetic operations as outputs.

The output of the arithmetic circuit is calculated from the following arithmetic expression-

$$D = A + y + C_{in}$$

Where A is the 4-bit number (A_0, A_1, A_2, A_3) to the x input (X_0, X_1, X_2, X_3) of the full adders, y is the 4-bit number (the outputs from the multiplexers) to the y inputs (Y_0, Y_1, Y_2, Y_3) to the full adders and C_{in} is the input carry which is either 0 or 1.

Depending on the values of S_0, $S_1$and C_{in}, the arithmetic circuit performs eight different micro operations as listed in the function table shown in Table 4.1.

Let's consider a few cases for better understanding the functioning of the arithmetic circuit.

CASE I: $S_1 = 0$ and $S_0 = 0$

In this situation, the input pins of multiplexers I_0(i.e. the bits of B) are chosen as the output and as a result B directly goes to the y inputs of the full adders (FA). i.e. $y = B$. Now, if $C_{in} = 0$ then the output becomes $D = A + B$ and if $C_{in} = 1$ then $D = A + B + 1$. This is how add micro operation is performed.

CASE II: $S_1 = 0$ and $S_0 = 1$

In this situation, the input pins of multiplexers I_1 (i.e. the complimented bits of B) are chosen as the output and as a result $\overline{B}$ goes to the *y* inputs of the full adders (FA). i.e. $y = \overline{B}$. Now, if $C_{in} = 0$ then the output becomes $D = A + \overline{B}$ which is equivalent to $D = A - B - 1$ and if $C_{in} = 1$ then $D = A + \overline{B} + 1$ which is equivalent to $D = A - B$. This is how subtract micro operation is performed.

CASE III: $S_1 = 1$ and $S_0 = 0$

In this situation, the input pins of multiplexers I_2 (connected to logic 0) are chosen as the output and as a result 0 goes to the *y* inputs of the full adders (FA). i.e. $y = 0$. Now, if $C_{in} = 0$ then the output becomes $D = A + 0$ i.e. $D = A$ which means transfer operation is done from A to Dand if $C_{in} = 1$ then $D = A + 1$ which means increment operation is performed.

Space for learners:

CASE IV: $S_1 = 1$ and $S_0 = 1$

In this situation, the input pins of multiplexers I_3 (connected to logic 1) are chosen as the output and as a result `1` goes to the all *y* inputs of the full adders (FA) and we know that if all bits of a number are 1 then it's equivalent to `-1`. So, y =`-1` here. Now, if $C_{in} = 0$ then the output becomes `D = A-1` which means decrement operation is done and if $C_{in} = 1$ then `D = A-1+1` i.e. `D = A` which means which means transfer operation is done from `A` to `D`.

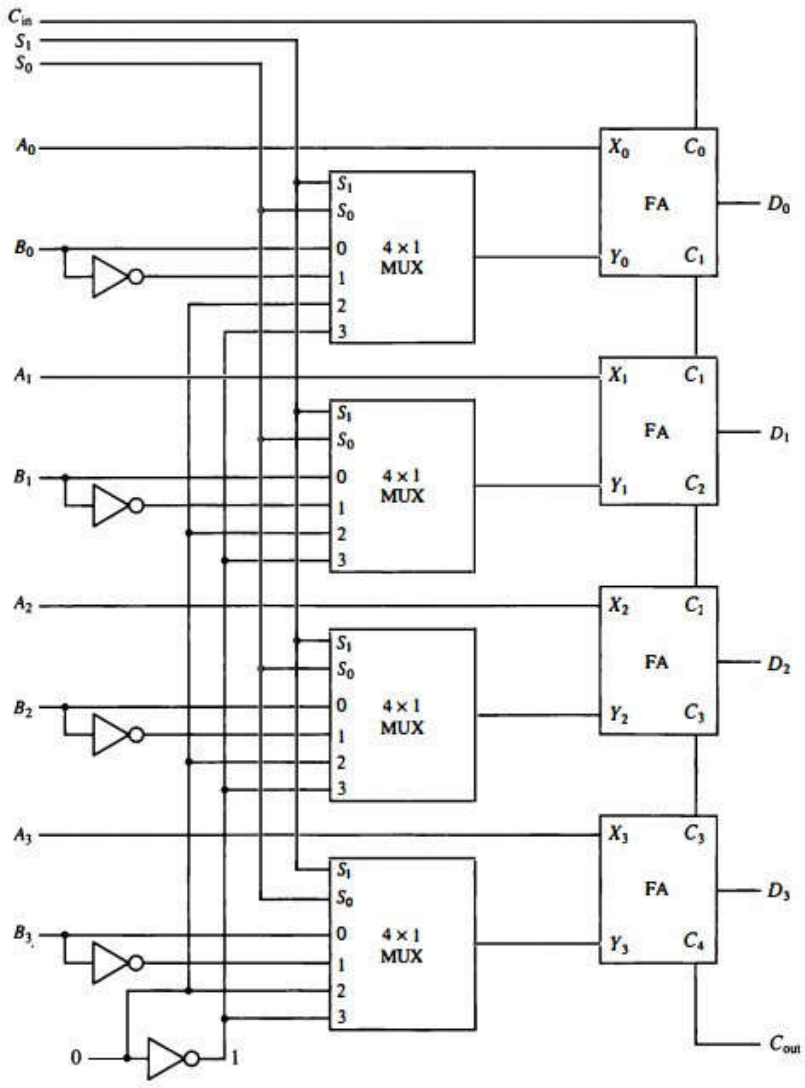


Fig 4.6 : A 4-bit Arithmetic Circuit

Space for learners:

Table 4.1 Function Table of Arithmetic Circuit

Inputs				Outputs	Micro operations
S_1	S_0	C_{in}	Y	D = A + y + C_{in}	
0	0	0	B	A + B	Add
0	0	1	B	A + B + 1	Add with Carry
0	1	0	$\overline{B}$	A + $\overline{B}$ or A − B − 1	Subtract with Borrow
0	1	1	$\overline{B}$	A + $\overline{B}$ + 1 or A − B	Subtract
1	0	0	0	A	Transfer A
1	0	1	0	A + 1	Increment A
1	1	0	1	A − 1	Decrement A
1	1	1	1	A	Transfer A

4.5.2 Various Arithmetic Micro operations

Add, subtract, increment and decrement are the basic set of arithmetic micro operations which are described below-

- **Add**: To add the contents of two or more registers and store the resultant sum in either one of the registers or in a third register, this micro operation is used. For example, to add the contents of two registers R_1 and R_2 and store the result in a third register R_3, the micro operation can be symbolized as:

  $$R_3 = R_1 + R_2$$

- **Subtract:** When the contents of one register needs to be subtracted from another register and store the result in either one of the registers or in a third register, then this micro operation is used. The subtraction operation is implemented through complement and addition operation. For example, to subtract the contents of register R_2 from register $R_1$and store the result in a third register R_3, the micro operation can be symbolized as:

  $$R_3 = R_1 + \overline{R_2} + 1$$ [Equivalent to $R_3=R_1-R_2$]

  Here, first we take the complement of R_2, add 1 to it and then the content of $R_1$is added to it. In other words, the 2's complement of$R_2$is added with R_1in order to carry out R_1-R_2operation.

Space for learners:

- **Increment:** This type of micro operation is used to increase the contents of a register by 1. For example, to increase the contents of register R_1 by one the symbolic micro operation will be:

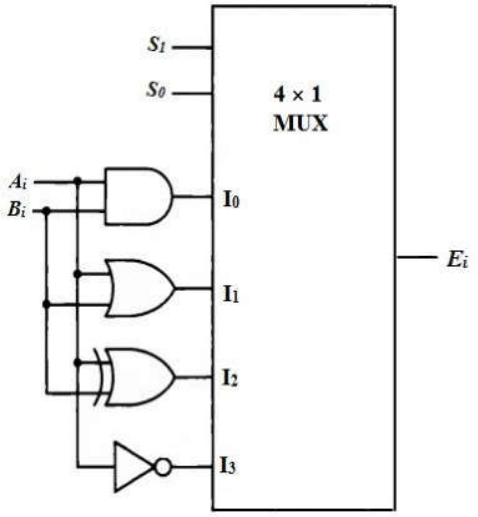
  $R_1 = R_1 + 1$

- **Decrement:** This type of micro operation is used to decrease the contents of a register by 1. For example, to decrease the contents of register R_1 by one the symbolic micro operation will be:

  $R_1 = R_1 - 1$

4.5.3 Logic Circuit

The basic logic circuit of the ALU performs various logic micro operations such as AND, OR, XOR and Compliment at bit-level.



S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	*AND*
0	1	$E = A \vee B$	*OR*
1	0	$E = A \oplus B$	*XOR*
1	1	$E = \overline{A}$	*Complement*

Fig 4.7: Single stage of Logic Circuit with Function Table

Fig 4.7 shows the hardware implementation for four common logic micro operations. The circuit is consisting of a 4×1 multiplexer with four inputs (I_0, I_1, $I_2$and I_3) and two select pins ($S_0$and S_1) to perform one of the four logic micro operations and direct it as the output E_i as shown in the function table.

Space for learners:

4.5.4 Some Applications of Logic Micro operations

The basic logic operations (AND, OR, NOT, XOR) can be applied to achieve various operations like set, clear, masking and inserting new bits etc. Let's discuss such common applications here-

- **Selective-set:** To set selected bits in register R_1 to 1 where there are corresponding 1's in register R_2. The 0's in are not considered. For example, before operation if R_1=0011 and R_2=0101 then after selective-set operation the contents of R_1 will be 0111. This operation is achieved by the OR logic micro operation, for above example this will be symbolized as: $R_1 \leftarrow R_1 \vee R_2$

- **Selective-clear:** This operation clears those bits in register R_1 to 0 where there are corresponding 1's in register R_2. For example, before operation if R_1=0011 and R_2=0101 then after selective-clear operation the contents of R_1 will be 0010. This operation is achieved by the AND logic micro operation with R_1 and complement of R_2, for above example this will be symbolized as: $\overline{R_1} \leftarrow R_1 \wedge R_2$

- **Selective-complement:** This operation complements those bits in register R_1 where there are corresponding 1's in register R_2. For example, before operation if R_1=0011 and R_2=0101 then after selective-complement operation the contents of R_1 will be 0110. This operation is achieved by the XOR logic micro operation with R_1 and R_2, for above example this will be symbolized as: $R_1 \leftarrow R_1 \oplus R_2$

- **Mask:** This operation clears those bits to 0 in R_1 where there are corresponding 0's in R_2. For example, before operation if R_1=0011 and R_2=0101 then after mask operation the contents of R_1 will be 0001. This operation is achieved by the AND logic micro operation with R_1 and R_2, for above example this will be symbolized as: $R_1 \leftarrow R_1 \wedge R_2$

- **Clear:** To compare the contents of two registers and results in all 0's if the contents of both the registers are same. For example, before operation if R_1=0011 and R_2=0011 then after clear operation the contents of R_1 will be 0000. This operation is achieved by the XOR logic micro operation

Space for learners:

with R_1andR_2, for above example this will be symbolized as: $R_1 \leftarrow R_1 \oplus R_2$.Thus, XOR operation can be implemented to determine whether two binary numbers are equal.

- **Insert:** This operation is used to insert new group of bits in a register. To perform this operation, first the unwanted bits of the register are masked and then OR operation is performed with the desired value. For example, if R_1=00111100 and we want to insert 0110 in the rightmost four bits. For this, first we mask the rightmost four bits which is done by ANDing the contents of R_1 with the value 11110000. After this mask operation we get R_1=00110000. Now the contents of R1 are ORed with the desired value (00000110) and after this operation we get R1 = 00110110. Thus, the new bits (0110) are inserted at the rightmost four bits.

4.5.5 Shift Micro operations

The shift micro operations move the contents (bits) of a register either to the left or to the right. There are three types of shift micro operations: *arithmetic, logical* and *circular* shifts. Let's discuss them one by one here.

- **Arithmetic Shift**: It shifts a signed binary number left or right without changing the sign of the number. To understand the arithmetic shift operation, let's consider a n-bit signed binary number$b_{n-1}, b_{n-2}, \ldots\ldots, b_1, b_0$ where b_{n-1} denotes the sign bit and b_{n-2} denotes most significant bit (MSB) and $b_0$denotes the least significant bit (LSB).

  The arithmetic shift operations can be symbolized as:

  $R_1 \leftarrow$ ashr R_1[1-bit arithmetic shift right R_1]

Space for learners:

$R_1 \leftarrow$ ashl R_1[1-bit arithmetic shift left R_1]

In arithmetic shift right operation, as the sign bit must be kept unchanged; all the bits including the sign bit are shifted to the right. So, the rightmost bit is lost. The b_{n-1} remains the same, while $b_{n-2}$is replaced by b_{n-1}, b_{n-3} is replaced by b_{n-2}, and so on and at last b_0 is lost.

In arithmetic shift left operation, all the bits are shifted to the left and a 0 is inserted in the previous b_0 bit position. The original value of b_{n-1} is lost as it is replaced by b_{n-2};$b_{n-2}$is replaced by b_{n-3} and so on. After this operation if the value of b_{n-1} changes then a sign reversal occurs which happens because of *overflow* which occurs if $b_{n-1} \neq b_{n-2}$ before the shift operation. The overflow condition can be checked by XO Ring the bit b_{n-1} withbitb_{n-2}. If the XOR operation results in 1 then there is overflow; otherwise not.

- **Logical Shift:** This micro operation moves the contents (bits) of a register either to the left or to the right. After left or right shift the empty/lost (the leftmost or the rightmost) bit is replaced by a 0. Symbolically they are represented as:

  $R_1 \leftarrow$ shr R_1[1-bit shift right R_1]

  $R_1 \leftarrow$ shl R_1[1-bit shift left R_1]

- **Circular Shift:** This micro operation is almost same as the logical shift, except there is no bit lost occurs here as the leftmost or rightmost bit which is shifted out at one end is circulated back to the other end. Symbolically they are represented as:

  $R_1 \leftarrow$ cir R_1[1-bit circular shift right R_1]

  $R_1 \leftarrow$ cil R_1[1-bit circular shift left R_1]

Space for learners:

Check Your Progress

*9.*The full form of ALU is _____

*10.*The ALU gives the output of the operations and the output is stored in the _____.

a) Memory Devices
b) Registers
c) Flags
d) Output Unit

11. The content of an 8-bit register is initially is 10011100. The content of the register after an arithmetic right shift opearation will be _____.

a) 11001110 b) 11001111

c) 11001101 d) 01001110

12. In arithmetic left shift, overflow occurs when _____.

a) $b_{n-1}=b_{n-2}$ b) $b_0=b_{n-1}$

c) $b_{n-1}\neq b_{n-2}$ d) $b_0 \neq b_n$

13. A digital computer performs which one of the following micro operations to subtract R2 from R1?

a) $R_3 = R_1+R_2+1$ b) $R_3 = R_1-R_2$

c) $R_3 = R_1+\overline{R2}+1$ d) $R_2= R_1-1$

14. State TRUE or FALSE:

a) The ALU performs logic operations only.

b) XOR operation can be used to check whether the contents of two registers are same.

c) In Circular Shift the bit in either end is circulated back to the other end.

Space for learners:

4.6 CONTROL UNIT

The main unit of the CPU is the control unit (CU) which generates control signals to the data path to direct the entire system operations. Data inside the CPU, memory unit and I/O devices are controlled by

these control signals issued by the control unit. The CU generally uses the control bus to carry the control signals to control the data flow between the CPU and other external units. The two basic tasks performed by the CU are:

- **Sequencing:** The CU is responsible for generating a proper sequence for the micro operations depending on the program currently being executed by the CPU.
- **Execution:** The CU causes the execution of micro-operations by generating control signals for opening and closing of gates to let the data pass through, while performing ALU operations.

Space for learners:

4.6.1 General Model of the CU

The Control Unit (CU) has inputs that empower it to identify the state of the system and outputs that empower it to control the function of the whole system. In addition to input and outputs it must include the logic required to perform the sequencing and execution which are the main functions of the CU. Fig 4.8 illustrates a general model of a control unit consisting of four major inputs: clock, Instruction Register (IR), flags, and the control signals from the control bus. The outputs from the CU are: control signals within CPU and control signals to control bus.

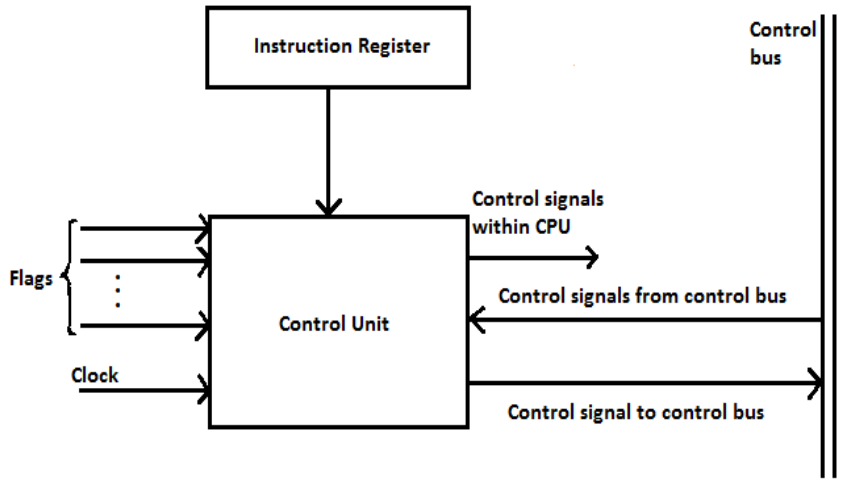


Fig 4.8: General Model of the CU

The inputs of the control unit are described below:

- **Clock:** The control unit uses a clock to keep track the time and sequence of micro operation execution. For each clock

pulse the control unit executes one micro operation or a set of concurrent micro operations which can be referred to as clock cycle time or processor cycle time.

- **Instruction Register (IR):** Micro operations that are fetched from the memory are stored in the IR. The opcode part of the instruction used for decoding the type of instruction to be executed.
- **Flags:** These are certain memory units capable of holding just 1 bit of information that are used to indicate the CU the current state of the processor and the results of recent ALU operations.
- **Control signals form control bus:** Interrupt signals, acknowledgement signals are such signals that are received by the CU from the control bus.

The outputs from the CU are described below:

- **Control signals within CPU:** Two types of control signals are generated by the CU within the CPU; one of these causes the register transfers and the other one is used to activate specific ALU functions.
- **Control signals to control bus:** Two types of control signals are generated by the CU to the control bus; one goes to the I/O modules and another goes to the memory.

The control unit of a digital computer can be implemented in two alternate ways: *hardwired* and *micro programmed* implementation. In hardwired implementation the control unit is comprised of logic gates, decoders, flip-flops and other control signal generating digital circuits etc. In micro programmed implementation, the control unit is comprised of a *control memory* where the control information is stored, which is programmed in such a way to initiate proper sequence of micro operations as required. Let's discuss both the implementations one by one.

4.6.2 Hardwired Control Unit

If the control signals are generated by the hardware using conventional logic design then control unit is said to be hardwired controlled. Fig 4.9 depicts the block diagram of a hardwired control unit consisting of a sequence counter (SC) and a number of logic

Space for learners:

circuits which may include decoders, flip-flops and other control logic gates.

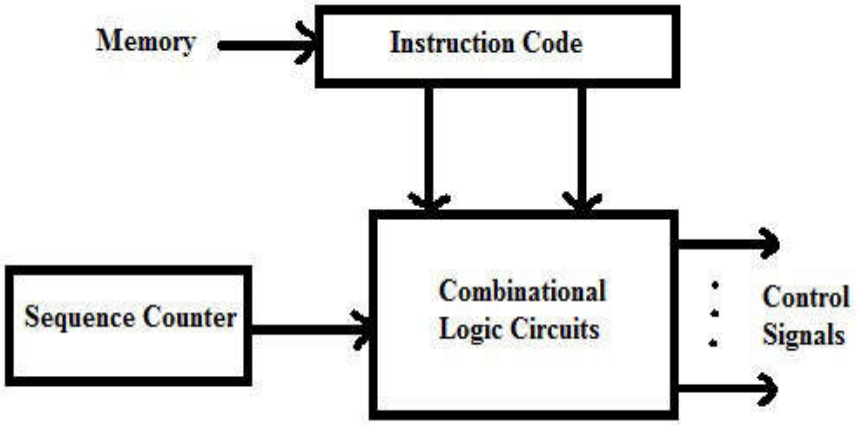


Fig 4.9 : Hardwired Control Unit

The hardwired organization is very complicated if we have a large control unit. In this organization, if the design has to be modified or changed then it requires changes in the wiring among various components. Thus the modification of all the combinational circuit may be very difficult.

Advantages:

- It works fast because of the use of combinational circuits to generate control signals.
- It can be optimized to operate in fast mode.
- It is faster than micro programmed control unit.

Disadvantages:

- Hardwired control unit is expensive.
- If the design has to be modified or changed then it demands changes in the wiring among various components.
- The design becomes complex if the number of control points in the CPU is large.

4.6.3 Micro programmed Control Unit

A micro programmed control unit's design is based on the concepts of microprogramming. Unlike the hardwired CU where the control signals are generated via combinational circuits, here control signals are generated using a sequence of microinstructions that specify the internal control signals for executing the micro operations. *Fig 4.10* depicts the organization of a micro programmed control unit.

Space for learners:

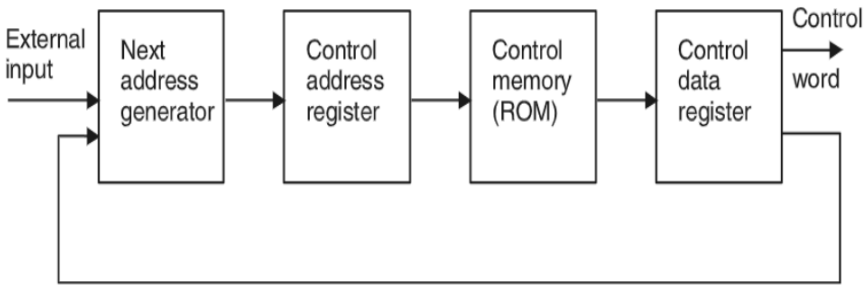


Fig 4.10 :Micro programmed Control Unit

The micro programmed control unit comprises of four components, which are described below:

- **Control Address Register (CAR):**The CAR specifies the address of the microinstruction that is read from the control memory. It can also be termed as the micro program counter ($\boldsymbol{\mu PC}$).
- **Control Data Register:** Depending upon the address specified in the CAR the control data register holds the microinstruction fetched from the control memory. It can also be termed as microinstruction register ($\boldsymbol{\mu IR}$)
- **Control Memory:** In addition to the main memory, a miro programmed CU has a separate memory called the control memory to hold the microinstructions and fixed micro programs that cannot be changed by a general user. The control memory can be read only memory (ROM) as changes in the micro programs are not required once the CU is under operation.
- **Next-address generator:** After having executed all the micro operations generated by a microinstruction, it is required to find the address of the next microinstruction. The next-address generator is responsible for computing the address of the next microinstruction to be executed. This is why it can be termed as micro program sequencer.

Advantages:

- The design of a micro programmed control unit is less complex.
- It is cheaper as number of hardware units is lesser and less error prone to implement.
- It can efficiently compute complex functions such as floating-point arithmetic etc.

Space for learners:

- It offers more flexibility to modification or change; as the modification can be brought just by changing the micro-program residing in the control memory to specify a different control sequence.

Disadvantages:

- It is slower than the hardwired control unit that means it requires more time to execute an instruction.
- In case of limited hardware resources it costs more than the hardwired control unit.
- For smaller CPU, the design duration of micro programmed control unit is more than the hardwired control unit.

Space for learners:

Check Your Progress

15. Control Memory is associated with _____.

a) Hardwired CU b) Micro programmed CU

c) Both a) & b) d) None of these

16. Which one is **not** a function of a Control Unit?

a) Control Signal b) Execution
c) Sequencing d) Programming

17. Which one of the followings is also known as Microprogram Counter?

a) Address Register b) Program Counter (PC)

c) Control Address Register d) Data Register (DR)

18. State TRUE or FALSE:

a) Control Data Register is also known as Microinstruction Register (μIR).

b) Micro programmed CU is faster than Hardwired CU.

c) Control Signals are carried by Control Bus.

4.7SUMMING UP

- In a digital computer, the elementary operations are also termed as micro operations which are performed on the data stored on the processor registers.

- The language used to specify the sequence of micro operations is known as Register Transfer Language.
- Various types of micro operations are register transfer micro operations, arithmetic micro operations, logical micro operations and shift micro operations.
- The data section of the CPU is data path. There are three types of Buses: address bus, data bus, control bus. One-bus, two-bus and three-bus are the various types of data path organization.
- The arithmetic logic unit (ALU) performs arithmetic, logical and bit-shifting operations using various circuits. Arithmetic circuit for various arithmetic operations and logic circuit for logical operations. The shifting operation can be done with the help of Arithmetic Logic Shift Unit.
- Another major part of the CPU which is the control unit (CU) responsible for generating control and timing signals to maintain the proper sequence of micro operation executions.
- The CU can be differentiated based on its design approach as hardwired CU and mirco programmed CU.

Space for learners:

4.8 ANSWERS TO CHECK YOUR PROGRESS

1. (a), 2 (c), 3 (a), 4.a True, 4.b False, 4.c True, 5. Data path, 6. (b), 7. (a), 8. False, 9.Arithmetic Logic Unit, 10. (b), 11. (a), 12. (c), 13. (c), 14.a False, 14.b True, 14.c True, 15. (b), 16. (d), 17. (c), 18.a True, 18.b False, 18.c True.

4.9 POSSIBLE QUESTIONS

Short answer type questions:

- What do you mean by RTL? Explain.
- What is control function of an RTL?
- How memory transfers are represented by RTL?
- What is datapath? What are its types?
- What are the operations performed by the ALU?

- How subtraction operation is performed by the arithmetic circuit?
- How overflow occurs in arithmetic shift left operation?
- What operations can be performed by the logic circuit of the ALU?
- What are the functions performed by the CU?
- What are various types of CU available?
- What are the components of a hardwired CU?
- What are the components of a micro programmed CU?

Long answer type questions:

- Explain the arithmetic circuit with its function table.
- Explain the logic circuit with its function table.
- Explain the general model of the Control Unit? What are the various types of CU available?
- What is hardwired CU? Discuss its advantages and disadvantages.
- What is micro programmed CU? Discuss its advantages and disadvantages.
- List out various differences between Hardwired and micro programmed Control Unit.

4.10 REFERENCES AND SUGGESTED READINGS

- M. Morris Mano, *Computer System Architecture,* Pearson Education, Latest edition.
- Express Learning Series- *Computer Organization and Architecture,* ITL Education Solutions Limited.

---×---

Space for learners:

BLOCK II:

MEMORY AND INPUT OUTPUT ORGANIZATIONS

UNIT 1: MEMORY ORGANIZATION

Space for learners:

Unit Structure

1.1 INTRODUCTION

A computer consists of three primary building blocks as input /output unit, control unit, and memory unit. It is used as a storage device in a system to store programs or a set of instructions, data,

and the intermediate results of arithmetical and logical computations. Depending on storing strategy the memories are classified into two prime categories – main memory or primary memory and auxiliary or secondary memory. Memory can be classified into different categories based on some key characteristics such as:

a. Depending on location memory are classify as CPU-based, internal memory, and external memory.
b. Depending on media used for manufacturing memory i.e. physical type memory is classified as semi-conductor based and magnetic surface-based.
c. Depending on physical characteristics volatile / non-volatile and erasable / non-erasable.
d. Depending on the access method memories are classified as sequential access, direct access, and random access memory.

Space for learners:

1.2 UNIT OBJECTIVES

After completing this unit, you will be able to learn:

- Learn about the functions of the memory unit.
- Memory operations
- Representation memory location in terms of byte
- Big-endian and little-endian assignment
- Composition of a memory chip.
- Learn about the memory hierarchy.
- Learn about the key factors that affect memory performance.
- Know about the different types of RAM and ROM
- Mapping of a memory chip and required amount of memory.

- Learn about the memory access methods.
- Learn about the concepts of cache and virtual memory.
- Learn about the hardware used in memory management.
- Concepts of secondary memory
- Functions of MMU.

Space for learners:

1.3 MEMORY OPERATIONS

Computers memory is used to store both program instructions and data operands. To execute an instruction the processor should load or transfer the instruction or set of instructions into the processor from the primary memory. During the processing of instructions, the operands or the results are also transferred between the memory and the CPU. Thus the basic operations involving in the memory can be classified into two categories such as Load or Read / Fetch and store or Write.

The READ operation transfers a copy of the contents from the memory location specified by the CPU. During the transfer, the memory contents remain unchanged. The READ operation is initiated by the CPU sending a request to the memory with a specific memory location in the address bus. The memory unit will read the contents from the specified address by the CPU and send them to the processor by loading the data into the databus.

The WRITE operation transfers data from the processor to a specific memory location specified by the instruction. This operation will overwrite the contents in that memory location. The processor sends the data along with the memory location where it has to be store or write. In a single operation, one word or 1 byte of data can transfer between the memory and the processor.

STOP TO CONSIDER

- The two main memory operations are READ and WRITE
- READ operation perform to fetch data from memory to processor
- WRITE operation perform to store data from processor to memory

Space for learners:

1.4 MEMORY CHIP

An integrated circuit (IC) consists of several capacitors and transistors with the capacity of storing information can be defined as a memory chip. Memory chips can be used for process code also. Memory chips can hold data either temporarily or permanently through RAM and ROM respectively. The size or shape and storage capacity of the memory chip can vary.

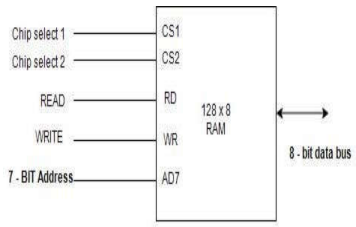


Figure: 1.1 Block diagram of a RAM Chip

A RAM chip is used to communicate with the CPU through control lines. Through a bidirectional data bus, RAM chips are allowed to communicate either from memory to CPU during a read operation or from CPU to RAM during a write operation. Following figure Fig.1.1 shows a typical block diagram of a RAM chip. The chip capacity is 128 words of 8 bits per word. This 128 x 8 chip required a 7-bit address and an 8-bit bidirectional data bus. The signal RD and WR are used to specify memory operations Read/Write respectively during communication. The chip select (CS) line is a control line through which the microprocessor can select and enable a particular chip. The functions of a RAM chip can be depicted as shown in Table 1.

Table 1: Functions table for RAM Chip

CS1	$\overline{CS2}$	RD	WR	Memory function	State of data bus
0	0	x	x	Inhibit	High impedance
0	1	x	x	Inhibit	High impedance
1	0	0	0	Inhibit	High impedance
1	0	0	1	Write	Input data to RAM
1	0	1	x	Read	Output data from RAM
1	1	x	x	Inhibit	High impedance

The RAM chip is in operation only when the value of CS1 = 1 and $\overline{CS2}$= 0. The barin $\overline{CS2}$ indicates that the input is enabled for its complements i.e. for the value 0. If the select controls are not enabled or if it is enabled, but Read and Write lines are not enabled, then the memory is inhibited and the data bus is in high impedance. The high impedance is a state where it behaves like an open circuit i.e. the output does not carry any signal.It leads to very high resistance and hence no current flows. When the CS1 = 1 and $\overline{CS2}$ = 0 the memory can be in reading or write mode. When the WR input line is enabled, then a byte of information will be transferred from the data bus into the memory location specified by the address lines. When the read input line is enabled, a byte of information from the memory specified by the address line is transferred into the data bus.

The ROM chip is also organized the same as that of the RAM chip. In the ROM chip there is no need for reading and write input control because the unit can only read. Thus if the chip is selected the bytes as specified by the address line will be appeared in the data bus.

Space for learners:

STOP TO CONSIDER

- Memory chips hold data temporarily or permanently through RAM and ROM.
- A RAM chip is used to communicate with CPU through control lines
- Depending on the requirement of memory the number of memory chip may vary.

Generally, the size of RAM and ROM varies from machine to machine. If a system required more memory storage than the capacity of a chip then many chips are required to get the necessary memory size. If the required size of memory is M x N and the chip capacity is m x n then the required number of chips can be calculated as

$$\mathbf{k = \frac{M * N}{m * n}}$$

CHECK YOUR PROGRESS

1. A. Calculate the number of memory chips of capacity 128 x 8 RAM needed to provide a memory capacity of 2048 bytes?
   B. How many lines of the address bus must be used to access 2048 bytes of memory? How many of these will be common to all chips?
   C. How many lines must be decoded for chip select? Specify the size of the decoder.

1.5 MEMORY LOCATIONS AND ADDRESSES

Program instructions, operands, and results of arithmetical logical operations are stored in computer memory. The computer memory is composed of millions of storage cells. Each storage cell can store

Space for learners:

one bit of information in the form of 0 or 1. To perform basic memory operations the cells are grouped into a fixed number of cells. Each group with n-bit is referred to as a "**word**" of information and the "**n**" will be known as "**word length**". It can be depictedin figure 1.2. Now a day's modern computers are typically ranging between 16 – 64 bits of word length.

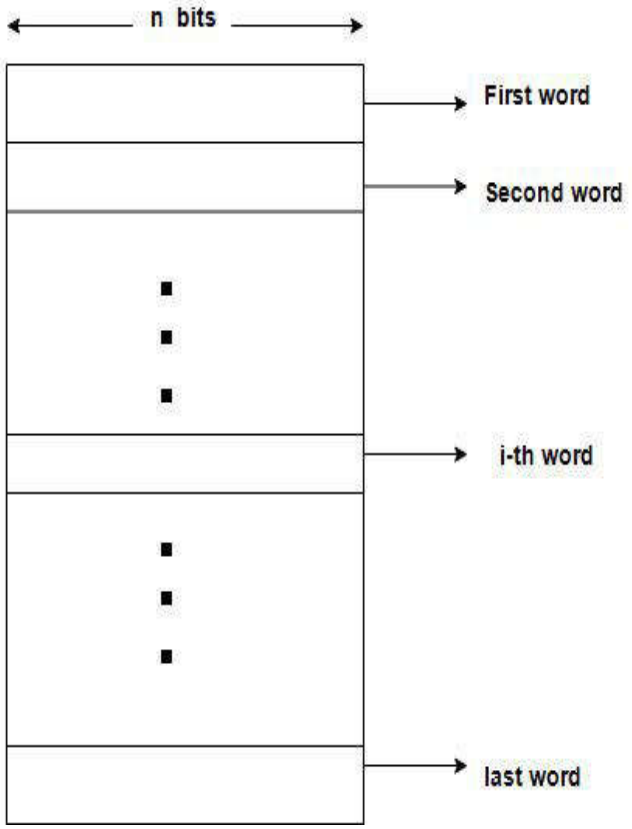


Figure 1.2 Memory words

A computer with 64 bit means that the address bus can carry an address of 64 bit for a specified memory location in computer memory. The address of memory locations is represented by 64-bit numbers. A computer with 32 bit can represent 2^{32} = 4294967296

Space for learners:

Thus the addressing scheme of a system determined the maximum size of computer memory or address space. For example, a system with a 16-bit computer i.e. with an addressing scheme of 16-bit addresses can address up to 2^{16} = 64 K number of memory locations. Similarly, a machine with 32-bit addresses can generate 2^{32} = 4GB memory locations. The memory location of a system determines the address space. Thus the addresses of each memory location are represented with k bits and using k address bit, 2^k nos. of locations or addresses can be represented. The address bit and number of locations is depictedin Table 2. Most computer systems are byte-addressable and memory is usually designed to store or access data in word-length quantities. For a computer, the word length can be defined as the number of bits that store or are retrieved in one access. The processor reads the memory data by loading the address of the required memory location into the Memory Address Register (MAR). Similarly, during a write operation, the processor writes data into a memory location by loading the address of that location into MAR. To perform the read/write operation on a set of consecutive memory locations in the main memory, then a block transfer operation may perform by sending the first address of the memory locations.

Table 2: Address bit and number of locations

K	Number of Locations
10	2^{10} = 1024 = 1 K
16	2^{16} = 65,536 = 64 K
20	2^{20} = 1,048,576 = 1 M
24	2^{24} = 16,777,216 = 16 M

Space for learners:

1.5.1 Byte addressability

A nibble is always 4 bits and a byte is 8 bits. The word length of a computer system can range between 16 – 64 bits. To assign an individual address for each of the bit locations in memory will increase the complexity of memory organization. In modern practices, each successive address refers to successive byte locations in memory. For a computer system with 32 bits, successive words will be located at addresses 0000, 0004, 0008,……. with each word of four bytes.

Space for learners:

1.5.2 Big – Endian and Little – Endian assignments

To assign the addresses across the words, there are two ways known as big-endian and little-endian assignments. The big-endian is used when the lower order byte addresses are used for the more significant bytes (MSB) or the leftmost bytes as shown in Figure 1.3. The little-endian is used when the lower order byte addresses are used for the less significant bytes (LSB) or rightmost bytes of the word as shown in Figure 1.3. Commercial machines are used both ways of assignment. To specify the address ordering of bytes within a word it is mandatory to specify the labelling of bits within a byte or a word as shown in Figure 1.4.

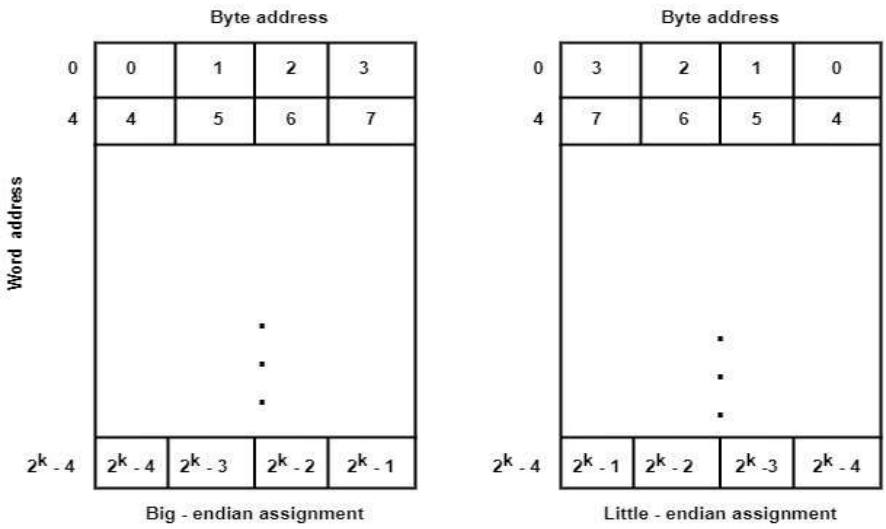


Figure 1.3 Big-endian and little-endian assignment

STOP TO CONSIDER

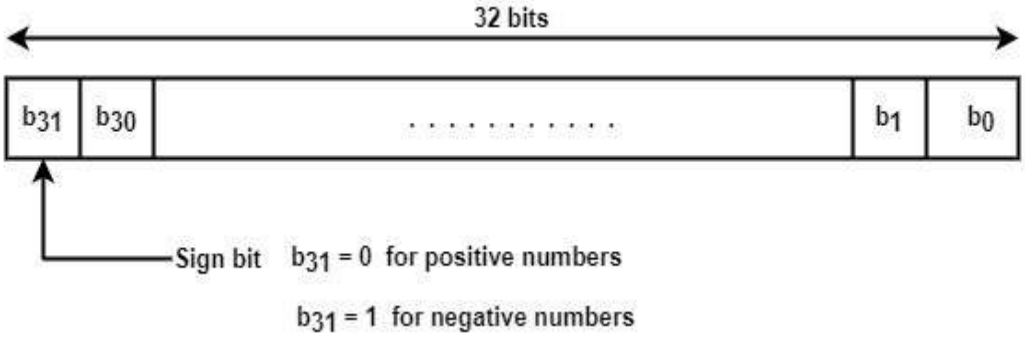
- The processor reads the memory data by loading the address of the required memory location into the Memory Address Register (MAR).
- A computer with 64 bit means that the address bus can carry an address of 64 bit for a specified memory location in computer memory.
- To assign the addresses across the words, there are two ways known as Big – endian and little – endian assignments.
- In modern practices each successive addresses refers to successive byte locations in memory.

CHECK YOUR PROGRESS

1. An address space is specified by 24 bits and the corresponding memory space by 16 bits.
    a. How many words are there in the word space?
    b. How many words are there in the memory space?
2. If a page consists of 2K words, how many pages and blocks are there in the system?

Space for learners:

Figure 1.4 labelling of bits within a byte or a word



Space for learners:

1.6 MEMORY HIERARCHY

Memory performance mainly depends on some key parameters –

a. **Memory access time** –It is defined as the total time requirement from submission of a request for the required piece of information by the CPU for getting or availability of the information in the CPU. CPU registers are local memory for the CPU and the access time is few nanoseconds. Cache memory takes small multiple access times of CPU registers. Cache memory is portions of memory made up of very high-speed static RAM (SRAM). Primary memory access time is few tens of nano seconds. For secondary memory, the access time is at least 10 msec. and it may measure in seconds if the data is to be fetched/write from or to a drive.

b. Storage capacity: The storage capacity of memory has a greater role in performance. As the capacity increase, the access time of the memory is also increased. CPU registers are good for almost 128bytes. Cache memory can be range as for L1 cache – 8 KB to 64 KB, for L2 cache – 256 KB to 512 KB, and for L3 cache 8 MB to 32 MB. Primary memory storage capacity ranges from 512 MB to 32 GB. The storage capacity of Secondary memory can vary from few gigabytes to terabytes or more than that.

Space for learners:

The memory hierarchy shows the organization of different types of memories depending on their performance. It can explain with a block diagram as shown in the following figure Fig.1.5. At the top of the memory hierarchy, CPU registers are located which are compact and accessible at full CPU speed. The next high-speed and high-cost memory is cache memory. CPU collects the required piece of information from cache memory. From the peak to the bottom of the hierarchy diagram, memory access time and size of the memory are gradually increase and the costs of

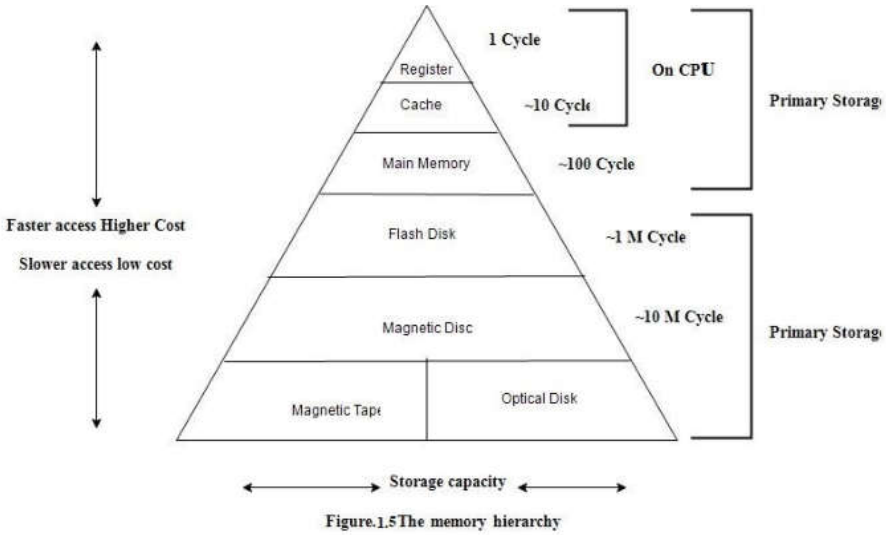


Figure.1.5The memory hierarchy

memory decrease. The memory hierarch primarily depends on some key parameters such as access time and storage capacity of the memory.

STOP TO CONSIDER

- Depending on the key factors such as storage capacity, accessibility, average access time memory can be organized in a pyramid structure known as memory hierarchy.
- Fastest and smallest memory lies on top or peak of the pyramid structure.
- Slower and bigger storage capacity memories are lies in bottom side of the pyramid.

1.7 SECONDARY MEMORY

Memory devices where data are kept permanently for a long time can call secondary memory. Secondary memories are non-volatile i.e. can store data permanently during a power cut or off mode. Some of the secondary memory devices are computer hard drive disk, pen drive, floppy disk, CD, etc. In comparison to the main memory size of the secondary or auxiliary memory is very large. The memory access rate of auxiliary memory is comparatively very less than main memory. Hence the cost is also relatively inexpensive. Thus we can say that cost is directly proportional to the storage capacity of the memory.

Space for learners:

1.8 MAIN MEMORY

The CPU directly communicated with a memory unit known as the main memory. The storage capacity of this type of memory is very large in comparison to cache memory and very small in comparison to secondary memory. The main memory can be classified into two different categories such as RAM and ROM.

1.8.1 Random Access Memory (RAM)

RAM can be defined as a read/write memory. Users can read the memory contents from RAM and also can write into RAM. It is volatile, i.e. it loses all the data when the power goes down. As the power supply goes down the memory contents or stored information in RAM are lost. In RAM any memory location can be accessed randomly without going through any other memory location. The access time for each location is the same. RAM can be classified

into two categories as static or SRAM and Dynamic memory or DRAM.

Space for learners:

1.8.1.1 Static RAM (SRAM):

SRAM consists of CMOS technology and uses transistors. For storing binary data it is used two cross-coupled inverters which is similar to flip-flops along with two other transistors for access control. The binary information exits in SRAM as long as the power supply is on. Figure 1.6illustrates how an SRAM cell is implemented. Two inverters are cross-coupled to form a latch. Two transistors T1 and T2 are used to connect the latch with the two-bit lines. Using the word line the transistors can be open or closed. When the word line is at ground level, the transistors are turned off and the latch retains its state.

To read the state of the SRAM cell, the word line is activated to close the switches T1 and T2.The signal on bit line b will be high and $b^{/}$will be low for cell state 1. Similarly, for cell state 0, the signal on bit line b is low and the signal in $b^{/}$ is high. Thus b and $b^{/}$are complements of each other. The state of the cell is set by activating the word line and putting the appropriate value for the bit line b and its complement $b^{/}$. Required signals on the bit line are generated by the sense/write circuit.

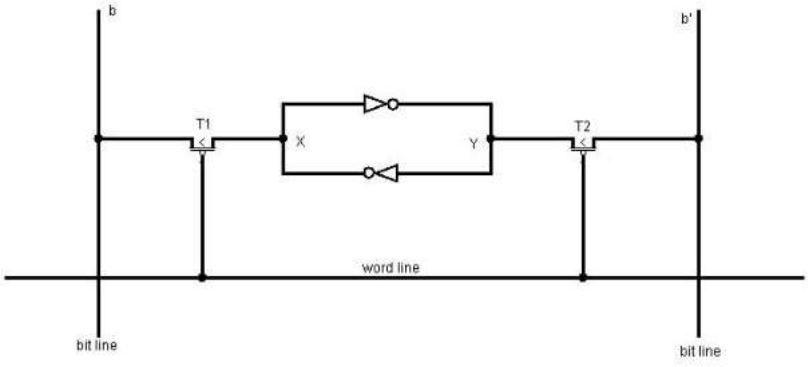


Figure 1.6 a static RAM cell

Space for learners:

1.8.1.2 Dynamic RAM (DRAM):

DRAM is constructed using capacitors and few transistors. The term dynamic in DRAM indicates that the charges are continuously discharging even in presence of an uninterrupted power supply and hence the capacitors must refresh periodically through refreshing the DRAM. DRAM is available in the market as it is less expensive. SRAM is an on-chip memory with very little access time whereas DRAM is off-chip memory with a large access time in comparison to SRAM. So SRAM is faster than DRAM. The storage capacity of SRAM is less than DRAM. Cache memories are comprised of SRAM whereas the main memory is comprised of DRAM. Power consumption in DRAM is more in comparison to SRAM.

STOP TO CONSIDER

- Memory can be volatile or non-volatile in nature.
- RAM is volatile and ROM is non-volatile memory.
- SRAM and DRAM are the two categories of random access memory (RAM).

1.8.2 Read Only Memory (ROM)

ROM is a non-volatile memory, i.e. contents of this type of memory remain the same or permanently in the memory and not erased due to power cut. Contents of ROM can be read or access during operation and nothing can be writing into it by the user or programmer. The manufacturing company decides and writes permanently into the ROM during manufacture. Different types of ROMs are PROM, EPROM, and EEPROM. Programs or sets of instructions that are required for starting a computer i.e. bootstrap programs are stores in ROM. ROM is used in some electronic gadgets such as fridges, refrigerators, washing machines, microwaves, etc.

Space for learners:

1.8.2.1 Programmable read-only memory (PROM)

PROM was first developed in 1956 by Wen Tsing Chow. It is a memory chip that can be programmed once after is created. Once the memory chip is programmed, the information written on it becomes permanent and cannot be erased or deleted. PROM was used in computer BIOS in early day's computers and now it is replaced by EEPROM.

1.8.2.2Erasable Programmable Read-Only Memory (EPROM)

EPROM is a memory chip that can store data even after a power cut also. Data from EPROM can be erased using ultraviolet light and makes it re-writable or programmable. It was first developed by Dov Frohman in 1971 at Intel. The contents of EPROM can be erased limitedly. Too much deletion can make the memory unit unreliable by destroying the silicon dioxide layer. It is not possible to erase the

EPROM contents partially. The whole data from EPROM is erased. For erasing and reprogramming the EPROM, the chip has to remove from the computer system and it consumes lots of time to erase data. The process of programming on EPROM is known as Burning and it is not a reversible process. It was developed to overcome the drawbacks of ROM and PROM. EPROM is successfully used in some microcontrollers such as Intel 8048, bootstrap loader, video-game, personal computers, etc.

Space for learners:

1.8.2.3Electrically Erasable Programmable Read-Only Memory (EEPROM)

It is a memory chip that can be erased by exposing electrical charge. Like other ROM, EEPROM retains its stored data even power is turned off. In the year 1978, George Perlegos developed the concept of EEPROM at INTEL. To erase the contents of EEPROM consumes approximately 5 milliseconds. In EEPROM erasing and reprogramming can be done without switching off the electrical circuit of the system.

1.9 CACHE MEMORY

The processing speed of the CPU in comparison to the access time of primary memory is very high. Due to this bottlenecking CPU cannot be utilized at its utmost level and remains idle. To remove this barrier a smaller memory is used in the system such that the average access time got increases and makes the computer memory more efficient. This chip-based smaller and faster memory is known as cache memory. It is a temporary storage area from which the CPU can retrieve data easily during processing. Sometimes it is

called the CPU Memory as it is typically integrated directly with the CPU chip or placed on a separate chip that has a direct connection with the CPU through a separate bus. As the cache memory is smaller in size and faster access time in comparison to primary memory, it increases the average access time and efficiency of the processor. The access time of cache memory is 10 to 100 times faster than the primary memory of a system. Cache consumes only a few nanoseconds to respondto a CPU request. Cache memory built with high-speed SRAM. It can be categorized into three different levels such as L1, L2, and L3 cache. L1 cache is extremely fast and usually embedded in the processor chip. L2 or secondary cache can be implanted on the CPU with a system bus. L3 cache is used to increase the performance of L1 and L2. The speed of L3 is comparatively slower than L1 and L2 but two times faster than DRAM.

Space for learners:

1.10 VIRTUAL MEMORY

A computer has a limited amount of memory space in primary memory or RAM. During programming, a user or developer has to concern about the limited amount of free memory address in RAM which increases the complexity of programming. To overcome this difficulty a technique called virtual memory has arisen. Virtual memory allows using more addresses than that the amount physically exists in the system. The main advantage of this memory is that the program may be larger than the size of the primary memory that physically exists. Using the concept of virtual memory the logical and the physical memory can be separated. This separation allows using of a large virtual memory for the developers over the actual physical memory in the system. It gives an illusion to

the programmer that large memory locations are available at their end even though the system has a smaller main memory.

The address generated by the user program is called a virtual address. A set of virtual addresses makes the *virtual address space*. The set of main memory addresses or locations are called *memory space* or *physical address space*. Usually, the virtual address space is larger than the physical address space. To map between virtual addresses with physical addresses, memory mapping techniques are used by the memory controller of the system.

Consider a computer system with RAM of a storage capacity of 32K words. To specify a location in RAM with 32K physical address space ($32K = 2^{15}$) 15 bit is required. Consider that the system has 2^{20}= 1024K storage capacity auxiliary memory. Assume the memory space is M and the P is the address space. Hence M=32K and P=1024K. Thus the address bit of the instruction code will have 20bits whereas the memory address must be specified by 15bit only. CPU will ask for reference instructions with the address of 20bit, but at this point, the reference address must be taken from the primary memory of the address with 15 bit rather than auxiliary memory. Thus there is a requirement of mapping of virtual addresses of 20 bit to physical 15 bit.

STOP TO CONSIDER

- Cache memory is typically integrated into CPU chip or placed on a separate chip that has direct connection with the CPU through a separate bus.
- To specify a location in RAM with 32K physical address space ($32K = 2^{15}$) 15 bit is required.
- Speed of CPU is very high then the average access rate of primary memory. Cache is used to remove this bottleneck between CPU and RAM.
- This bottlenecking problem can decrease the performance of the computer system.

Space for learners:

1.11CLASSIFICATION OF MEMORY BASED ON THE ACCESS METHOD

There are three types of memory access methods as Sequential access, Random access, and direct access.

1.11.1 Sequential access:

In this system, the stored data are accessed in afixed ordered manner i.e. in a specific linear specific manner. Here the access time depends on the location where the data exist. To go from memory location 1001 to 1010 in sequential access, it has to pass through all intervening memory locations. No one can jump from 1001 to 1010 directly as shown in figure 1.7. Examples of sequential media access memory devices are magnetic tape, magnetic disk, optical memories, DVDs, CDs, hard drives, etc.

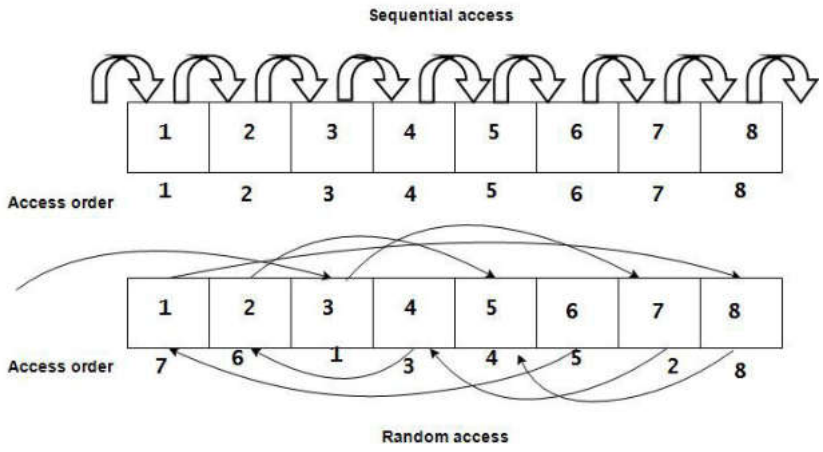


Figure 1.7 Sequential and random access method

Space for learners:

1.11.2 Random access:

It refers to access data randomly from the storage device. In the random access method, one can jump from memory location 1001 to 1010 directly without passing through all the intervening locations i.e. 1002, 1003,…. etc. Examples of random access memory devices are disk, RAM, ROM.

1.11.3 Direct access:

In this method, a unique address has been assigned for each block or record based on physical location. It can be seen as a hybrid of random and sequential access methods. The direct access method is used in magnetic hard disks as it contains a huge number of rotating storage tracks. Each track is associated with its own read/write head. Magnetic tracks are accessed randomly, but within the track, the data are accessed sequentially. Magnetic hard disk is a good example of using a direct access method for accessing memory contents.

STOP TO CONSIDER

- Data stored in memory can be access three different ways.
- Sequential access READ or Write data sequentially where as in Random access READ or WRITE operation are performed randomly on memory locations.
- Magnetic hard disk is good example of using direct access method for accessing memory contents.

Space for learners:

1.12 MEMORY MANAGEMENT HARDWARE

Space for learners:

To manage the operations performed by memory dedicated hardware is placed in between the processor and main memory called Memory Management Unit (MMU). If the processor does not have an on-chip MMU, then use an external MMU. The operations done by MMU are performed by the operating system. But to reduce the load on the operating system MMU is used. The logical address can be defined as the memory address which is being used by a program. A logical address represents or specifies the location of an instruction or data in a program relative to the starting address of the program. During the compilation of a program statement, the logical addresses are represented by a memory pointer consisting of two parts namely segment selector and offset. For a page-oriented system, the memory pointer has a page address and page offset. The physical address will be represented in terms of page number and page offset. The virtual memory concept is also performed by the MMU to provide a very large memory space to users. The concept of virtual memory allows the users to use more memory than that a system has in reality. A computer processor can access the data from the main memory during the execution of an instruction. For the execution of a program statement, it has to store or load into the main memory. The MMU allows users to store the program instructions into the secondary memory and it transfers a block of instructions to the main memory which is currently required by the processor. Similarly, the parts of the program statements are sending back to the secondary memories which are not being used by the processor currently. This to and fro data transferring process between main memory and secondary memory is known as swapping.

When a request for data or instruction sends by the processor to the MMU by specifying a logical address, the MMU checks the segment containing that logical address in the main memory. If it is available in the physical memory then the MMU calculates the physical address corresponding to the logical address specified by the processor. If the required segment is not available in the physical memory then MMU interrupts the CPU. On receiving an interrupt signal from the MMU, the CPU access or read the desired segment from the secondary memory.

Space for learners:

1.13 SOLVED EXAMPLES

1. 16K x 8 RAM chips are used to construct 64K x 16 RAM. Calculate the required number of chips for construction.

   Solution: Number of chips required $=\frac{64\,K*16}{16\,K*8}$ = 8 chips

2. 1K x 4 RAM chips are used to construct 1K x 8 RAM. Calculate the required number of chips for construction.

   Solution: Number of chips required $=\frac{1\,K*8}{1\,K*4}$ = 2 chips

3. **Direct Mapping Question:** Assume a computer has 32-bit addresses. Each block stores 16 words. A direct-mapped cache has 256 blocks. In which block (line) of the cache would we look for each of the following addresses? Addresses are given in hexadecimal for convenience.

   a. 1A2BC012

   b. FFFF00FF

   c. 12345678

   d. C109D532

   Solution: Of the 32-bit address, the last four bits denote the word on the line. Since four bits are used for one hex digit, the

last digit of the address is the word on the line. With 256 blocks in the cache, we need 8 bits to denote the block number. This would be the third to last and second to last hex digit.

a. this would be blocking 01, which is block 1

b. this would be 0F which is block 15

c. this would be 67 which is block 103 (remember, 67 is a hex value)

d. this would be 53 which is block 83.

Space for learners:

CHECK YOUR PROGRESS:

1. Choose the correct options from the following: (Multiple choice questions)

   i. What is true about the memory unit?
      A. Memory is the collection of storage units or devices together.
      B. The memory unit stores the binary information in the form of bits.
      C. Both A and B
      D. None of the above

   ii. When the power of a computer system shuts down, then which type of memory loses its data?
      A. Non-volatile memory
      B. Volatile memory
      C. Both A and B
      D. None of the above

   iii. The fastest data access is provided using_____
      A. Cache memory
      B. DRAM
      C. SRAM
      D. Registers

   iv. The minimum time delay between two successive memory read operations is_____.
      A. Cycle time
      B. Latency
      C. Delay
      D. None of the above

v. The effectiveness of the cache memory is based on the property of_____.
   A. Locality of reference
   B. Memory localization
   C. Memory size
   D. None of the above

vi. The drawback of building a large memory with DRAM is
   A. Large cost factor
   B. Inefficient memory organization
   C. Slow speed of operation
   D. All of the above

vii. The memory which is used to store the copy of data or instructions stored in larger memories, inside the CPU is called _____.
   A. L1 cache
   B. L2 cache
   C. Registers
   D. TLB

viii. Four memory chips of 16 x 4 sizes have their address bases connected. The system will be of size
   A. 64 x 64
   B. 16 x 16
   C. 32 x 16
   D. 256 x 1

ix. In a virtual memory system, the address space specified by the address lines of the CPU must be _____ than the physical memory size and _____ then the secondary storage size.
   A. Smaller, smaller
   B. Smaller, larger
   C. Larger, smaller
   D. Larger, larger

x. For the synchronisation of the read head we make use of a _____.
   A. Framing bit
   B. Synchronization bit
   C. Clock
   D. Dirty bit

xi. The BOOT sector files of the system are stored in _____.
   A. RAM

Space for learners:

B. ROM
C. Hard disk
D. Fast solid-state chip in the motherboard.

xii. The technique where the controller is given complete access to the main memory is
A. Cycle stealing
B. Memory stealing
C. Memory con
D. Burst mode

xiii. How many address bits are required to represent a 32K memory?
A. 10bits
B. 12 bits
C. 14 bits
D. 16 bits

xiv. Which of the following memories stores the most number of bits?
A. 64 K x 8 memory
B. 1 M x 8 memory
C. 32 M x 8 memory
D. 64 x 6 memory

xv. In a virtual memory system, the addresses used by the programmer belongs to
A. Memory space
B. Physical address
C. Address space
D. Main memory address

Space for learners:

1.14 SUMMING UP

1. The memory unit is a key part of a computer system.
2. Computer memory can be divided into two categories namely primary and secondary memory.
3. The central processing unit of a system directly communicates with the primary memory.
4. The processor can access the secondary memory through primary memory.

5. Primary memory can be categorized as RAM and ROM.
6. To increase the throughput of a system cache memory is used in between the primary memory and CPU.
7. Different types of ROMS are available such as ROM, PROM, EPROM, and EEPROM.
8. If the required chip size is M x N and if the chip capacity is m x n, then the number of the required chip is k = $\frac{M*N}{m*n}$
9. A set of physical addresses is known as memory space.
10. The address space can be divided into groups of equal size known as a page.
11. The memory space is broken into groups of equal size called blocks.
12. A computer processing speed can represent using the address bit.
13. Big-endian and Little-endian assignments are used in byte addressing.
14. MMU is used to control the communication between the CPU and memory.
15. The concept of virtual memory allows users to use memory space during programming which does not exist physically.
16. Lower order bytes are used to represent MSB in big-endian assignments.
17. Lower order bytes are used to represent LSB in little-endian assignments.
18. In modern practices, each successive address refers to successive byte locations in memory
19. Cache memory can be categorized into three different levels such as L1, L2, and L3 cache.
20. SRAM is an on-chip memory with very little access time whereas DRAM is off-chip memory with a large access time in comparison to SRAM.

Space for learners:

1.15 ANSWERS TO CHECK YOUR PROGRESS

Answers to question no. 1:

i. C ii. B iii. D iv. A v. A
vi. C vii. A
viii. B ix. C x. C xi.B
xii.D xiii.D xiv.C xv. C

1.16 POSSIBLE QUESTIONS

1) How many 128 x 8 RAM chips are needed to provide a memory capacity of 2048 bytes?
2) How many 128 x 8 RAM chips are needed to provide a memory capacity of 4096 x 16?
3) What is the bit storage capacity of a ROM with a 1024 x 8 organization?
4) Find the total number of cells in a 64k x 8 memory chip.
5) What is virtual memory?
6) Differentiate between ROM and EEPROM.
7) What are the key factors for memory efficiency?
8) What is memory access time?
9) What is clock cycle and CPU burst time?
10) Differentiate between sequential and random access?
11) What are SRAM and DRAM?
12) What is a memory chip? How the number of chips is calculated for the required number of memory?
13) What is memory hierarchy?
14) Why DRAM is slower than SRAM?
15) What is cache memory?
16) Explain the memory hierarchy with a block diagram.
17) What are the classifications of memory depending on access method, Explain?

Space for learners:

18) Explain the organization of a RAM chip.
19) What are the different types of ROM? Explain the differences between them.
20) Explain some data structures which are using sequential access and random access.
21) How direct memory is different from random access memory?
22) Discuss static and dynamic RAM.
23) Explain the functions of the memory management unit (MMU) of a computer system.

Space for learners:

1.17 REFERENCES AND SUGGESTED READINGS

- William Stallings, Computer Organization and Architecture Designing for Performance, Pearson Education India.
- Carl Hamacher, ZvonkoVranesic, SafwatZaky, Computer Organization, McGraw Hill Education.
- M. Morris Mano, Computer System Architecture, Pearson Education India.

---×---

UNIT 2 : CACHE MEMORY

Space for learners:

Unit Structure:

2.1 INTRODUCTION

To compensate for the speed of primary memory access time and CPU, a high speed memory is used called cache memory. Cache memory increases the processing speed of the CPU by making the required data available to it. Thus the cache memory has a great role in increasing the throughput of a system. It is placed in between the processor and main memory. The memory access time of cache memory is very high in comparison to main memory and compatible with the speed of the processor. The cache is used to store the program segment currently executed by the CPU and the data used by the CPU frequently. Since the memory space of the cache is much smaller than the main memory, mapping is required to identify the location in the main memory as specified by the CPU. Using cache replacements algorithms the contents of the cache can be changed by new program segments as required by the processor.

Space for learners:

2.2 UNIT OBJECTIVES

The primary objectives of the chapter are

- to know about cache memory and its use
- to understand how to measure the performance of cache memory
- to explore what are the operations of cache memory?
- to learn about the mapping process of cache memory.
- to find the different mapping processes
- to visualize cache replacement policies.
- to understand the cache optimization techniques.
- to know how to write into the cache.
- to discuss the different levels of cache memory.
- to learn about cache coherency

2.3 BASIC OPERATIONS

When the processor needs a particular data during its execution, at first it searches the data in cache memory. If it is found then the content is extracted from the memory location of cache as specified by the processor. If the word addressed by the CPU is not available in the cache memory, the main memory is accessed to read the word. The program segment or a block of the word containing the desired one will be transferred from the main memory to the cache memory. In multilevel cache, it can be categorized into two; internal cache, typically located inside the CPU chip and external cache, normally placed in the system board. Internal caches are known as primary or L1 cache and the range may have within 1– 32 KB. External caches are known as secondary or L2 cache and the range may vary between the range 64 KB – 1 MB.

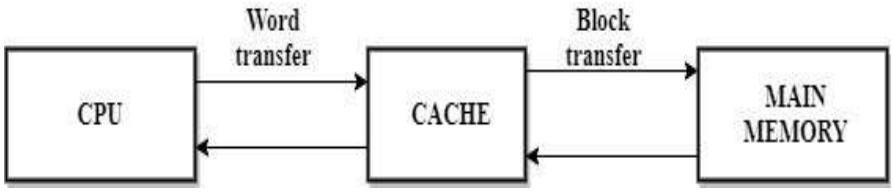


Figure 2.1 Cache memory

Space for learners:

STOP TO CONSIDER

- In multilevel cache, it can be categorized into two; internal cache, typically located inside the CPU chip and external cache, normally placed in the system board.
- Internal caches are known as primary or L1 cache and the range may have within 1kb – 32kb.
- External caches are known as secondary or L2 cache and the range may vary between the range 64kb – 1mb.

Space for learners:

2.4 PERFORMANCE

The performance of cache memory can determine by the ratio of finding the required data by the processor. If the required data is available in cache memory then it can be defined as a HIT, if it is not available in the cache then it is called a MISS. Three different types of cache miss may exist namely – compulsory miss, conflict miss and capacity miss. Compulsory miss may occur when a memory location is accessed for the first time. Conflict miss can occur due to insufficient space when two blocks are mapped on the same location. Capacity miss may take place due to smaller space in cache memory. The time taken to check the presence of data in the cache is called hit latency. For every hit, the CPU accesses the data from the cache directly but for a miss, the CPU has to wait for responding from the main memory. The block of data will be transferred from the main memory to cache memory and then the required word will transfer from the cache to the CPU. The ratio between the hit and the total number of references by the CPU (the summation of hit and miss) can be defined as the hit ratio. The hit ratio "h" always lies between the range 0 and 1. Let us consider that the

h is the hit ratio,$\mathbf{t_m}$ is the memory access time, $\mathbf{t_c}$ is the cache access time

$\bar{t}$is the average access time.

Then the average access time $\bar{t}$ can be calculated by the relation:

$\bar{t}$= $ht_c + (1 - h)(t_c + t_m)$ …….. (1)

The relation (1) is derived using the fact that for a cache hit, the main memory will not access by the processor. For a miss, both the cache and main memory will be accessed by the CPU. Consider that the ratio between cache and main memory access time is $\gamma = \frac{t_m}{t_c}$ then the efficiency (Λ) of a system using cache memory can be derived as:

$$\Lambda = \frac{t_c}{\bar{t}}$$

$$= \frac{t_c}{ht_c + (1-h)(t_c + t_m)}$$

$$= \frac{t_c}{t_c\left[h + (1-h)\left(1 + \frac{t_m}{t_c}\right)\right]}$$

$$= \frac{1}{h + (1-h)(1+\gamma)}$$

$$= \frac{1}{h + 1 - h + \gamma(1-h)}$$

$$= \frac{1}{1 + \gamma(1-h)}$$

For the value of h = 1, the efficiency Λ = 1, i.e. efficiency will be maximum for h = 1 or all the CPU references are confined to the cache.

Space for learners:

STOP TO CONSIDER

- An INTEL motherboard of 100MHz consume 180ns to retrieve information from main memory, whereas 45ns from the cache memory.
- Static RAM (SRAM) is typically used to build cache memory.
- Systems with Multi-core CPUs are generally used a separate L1 and L2 cache for each core and L3 is shared by each core.

Space for learners:

Example 2.1: Calculate average access time ($\bar{t}$), the ratio between main memory access time and cache access time (γ),and efficiency (Λ) of a memory system whose parameters are indicated as: t_c=150 ns, t_m=950 ns, and h=0.90.

Solution: Since the average access time $\bar{t}=ht_c+(1-h)(t_c+t_m)$

$=0.90*150+(1-0.90)(150+950)$

$=135+0.1(1100)$

$=245ns$

And since the $\gamma = \frac{t_m}{t_c} = \frac{\mathbf{950}}{\mathbf{150}} = \mathbf{6.33}$

And efficiency $\Lambda = \frac{\mathbf{1}}{\mathbf{1+\gamma(1-h)}}$

$$= \frac{\mathbf{1}}{\mathbf{1+6.33(1-0.9)}} = \frac{\mathbf{1}}{\mathbf{1+0.633}} = \frac{\mathbf{1}}{\mathbf{1+0.633}} = \mathbf{0.612}$$

Example 2.2: The access time of cache memory is 50 ns. And the access time for the main memory is 500 ns. It is estimated that 80% of the main memory requests are for reading operation and the remaining are for the write operation. The hit ratio for reading operation is 0.09 and a write-through policy is used.

a. Compute the average access time for the memory read cycles only?
b. Calculate the average access time for both read and write requests?
c. What is the hit ratio regarding the write cycle?

Solution:

a. Since the average access time $\bar{t}=ht_c+(1-h)(t_c+t_m)$

$=0.90*50+(1-0.90)(50+500)$
$=45+55$
$=100ns$

b. For both read and write cycle

Average access time = P_r * average access time for read + (1 - $P_{r)}$ * t_m

$= 0.8 * 100 + 0.2 * 500$
$= 80 + 100$
$= 180$ ns.

c. Hit ratio when write cycle is also considered is

$h = P_r * h_r + (1 - P_{r)} * h_w$ [h_w is the hit ratio for write cycle]

$= 0.8 * 0.9 + 0.2 * 0$
$= 0.72$

2.5 MAPPING PROCESS

There are three different types of mapping techniques in cache organization such as

a. Associative mapping
b. Direct mapping
c. Set – associative mapping

Space for learners:

2.5.1 Associative mapping

Space for learners:

In the case of the associative mapping procedure, each of the cache memory contents is associated with an address. But during the execution of a program statement, the data is not read or fetch by referring to or specifying any memory address. Instead of it, the data are searched by matching with the contents. In associative mapping, the cache memory contains the data along with the main memory address of the corresponding data as shown in Figure 2.2. The address bit sent by the CPU for searching the required data in the cache memory is matched with the stored addresses in the cache. If any address is matched, the corresponding word

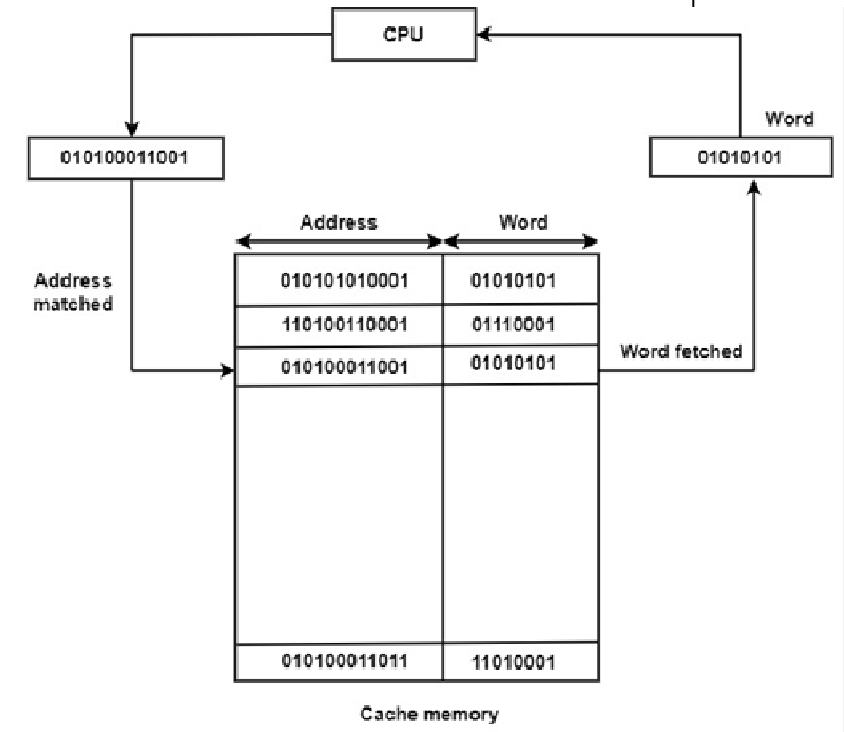


Figure 2.2 Associative mapping in cache memory

from that memory location will be fetched by the CPU. For a miss or if no match is found for the required word, then it will be searched in the main memory. Then the word from the main memory along with the address will be transferred into the cache memory. If the cache is full, using any replacement technique must make room for the new word.

Associative mapping is a very fast access method. But the manufacturing difficulties and cost are more in comparison to other mapping methods.

Space for learners:

2.5.2 Direct mapping

Consider a computer system with the main memory storage capacity is 4K, i.e. 4 x 1024 = 2^{12} bytes. Then the required number of bits to address the main memory location will be 12. Consider a cache memory of 1K = 2^{10} bytes, i.e. 10 bits are required to address a cache memory location. Thus the main memory required a 12-bit address line whereas the cache memory required only 10 bits of the address. In the direct mapping method, the address sent by the CPU is divided into two parts namely tag field and index field. The index field contains an equal number of bits that are required to address a word in cache

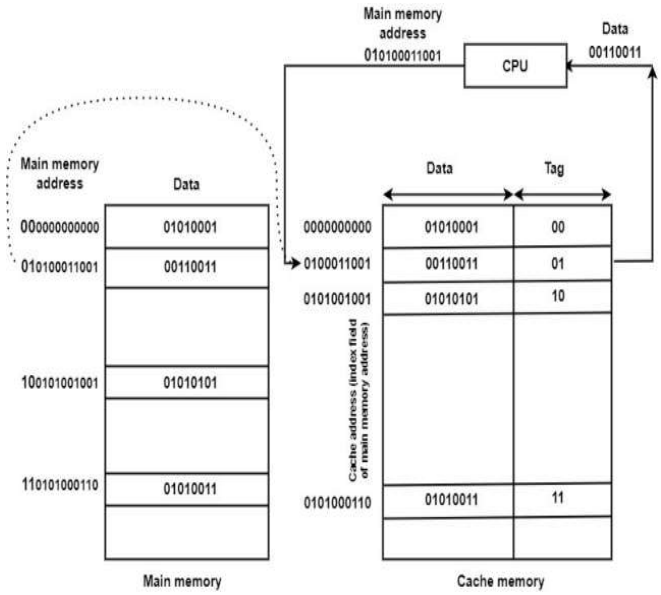


Figure 2.3 Block diagram

memory. The remaining bits are used in the tag field. If a system contains the main memory of capacity 2^m and cache of capacity 2^n, then the bits in the index field will be n bits and in the tag field is an **(m-n)** bit. In the example cited above, the index field and the tag field are consist of 10 bits and 2 bits respectively.

In direct mapping, the cache memory stores the data as well as the tag field as shown in Figure 2.3. In the cache, the words are stored in a memory location as the index field defined. When an address is requested by the CPU, the index part of the address is used to get the location in the cache memory. If the tag of the cache is matched with the tag of the requested address, the word will be fetched by the CPU. Else there will be a miss and the data will be searched in the main memory. For a miss, the block of data from the main memory has to be transferred into the cache memory by dividing the main memory address into index and tag fields. The main disadvantage of direct mapping is that, if the index field is the same for more than one word in cache memory with a different tag value, the hit ratio may drop considerably.

CHECK YOUR PROGRESS

Question: A digital computer has a memory unit of 64k x 16 and a cache memory of 1K words. The cache uses direct mapping with a block size of four words.

A. how many tags are there in the tag, index, block and words fields of the address format.

B. how many bits are there in each word of cache, and how are they divided into functions? Include a valid bit.

C. How many blocks can the cache accommodate?

Space for learners:

2.5.3 Set associative mapping

In direct mapping, two words or data of a similar index field cannot be store at the same time. To overcome this drawback of direct mapping, the third method of mapping is used which is known as set-associative mapping. In this method, cache memories are allowed to store more than one word with a similar index in the same word location along with a different tag. The number of tags–data pair in the one-word location of the cache is said to be as a set. An example of set-associative mapping has been depicted in Figure 2.4. As shown in the figure, the word stored in the memory addresses 001010011001 and 011010011001 of main memory is stored in cache memory at index address 1010011001. Similarly, the word stored at address 101010000111and 111010000111 of main memory is stored in cache memory index address 1010000111.

INDEX	TAG	DATA	TAG	DATA
1010011001	00	10000111	01	00111010
1010000111	10	10100110	11	11100001

Figure 2.5 Block diagram of set associative

STOP TO CONSIDER

- If a system contains main memory of capacity 2^m and cache of capacity 2^n, then the bits in index field will be n bits and in tag field is an **(m-n)** bit.
- In direct mapping when an address is request by the CPU, the index part of the address are used to get the location in the cache memory.

Space for learners:

CHECK YOUR PROGRESS

Question: A block set associative cache consists of a total of 64 blocks divided into 4 blocks sets. The main memory contains 4096 blocks each consisting of 128 words.

1. How many bits are there in main memory address?
2. How many bits are there in each of the TAG, SET and WORD fields?

Space for learners:

2.6 CACHE REPLACEMENT POLICIES

When the cache memory of a system is full, then cache replacement policies are used to make a decision about which page or data has to be replaced from the cache to make room for new data. The main problem in cache is that how to identify the page or data to be removed from cache memory. Lots of algorithms for cache replacement are being developed. The efficiency of those algorithms depends on the factors such as time, number of misses and balancing cost, etc. An efficient algorithm takes less time, lower number of miss rate and balancing cost. Some of the cache replacement algorithms are discussed as follows.

2.6.1 Least recently used (LRU) algorithm

This algorithm discards the least recently used data from the cache to make space for new data. A variable known as the **aging bit** is used to keep the record of all data items such as which data is used when or kept at what time in the cache. It is one of the most popular algorithms among all others as it provides better performance. The implementation policy of the LRU algorithm is also very simple and time and space overhead are constant.

Example 2.3: Let us consider a set of data items 7 0 1 2 0 3 0 4 2 3 0 3 2, which have to load into the cache memory of size 4 word. How many misses will occur if the LRU technique is being used as a cache replacement policy?

Solution: Initially the cache was empty, so for the first four data items namely 7, 0, 1, and 2, there will be **4MISS** as shown in figure 2.6.

For the 5th element 0, does already exist in the cache so **0 MISS**.

For the 6th element 3, which does not exist in the cache, it will replace the least recently used 7 from the cache – **1 MISS**

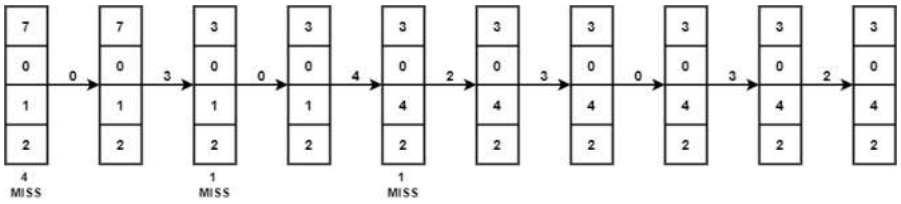


Figure 2.6 Example of Cache least recently used replacements

For the 7th element 0, does already exist in the cache so **0 MISS**.

For the 8th element 4, which does not exist in the cache, it will replace the least recently used 1 from the cache – **1 MISS**

For further referencing all the data exist in the cache, so no more replacement is required for any one of them. The total number of MISS is 6 and the number of hits is 7.

2.6.2 Least frequently used (LFU) algorithm

The LFU counts the number of uses of a particular data item or it counts how frequently the data item has been used. The data which

Space for learners:

is used very few will be identified and removed from the cache first. If all the data items in the cache have the same count then randomly any one of the items has been chosen and deleted. The min-heap data structure is a suitable one to implement this algorithm.

Space for learners:

2.6.3 First in first out (FIFO) algorithm

FIFO algorithm removes the data which has been come first into the cache and has not been used for a long time. It is the simplest algorithm to implement. Here the system keeps track of all the blocks or words in memory in a queue. The oldest page is in the front of the queue. When a replacement is required, the data from the front of the queue will be selected for removal.

Belady's anomaly – proves that it is possible to have more MISS for an increasing number of frames while using the FIFO replacement algorithm. For example consider a set of data items such as 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4, and 3 and 3 slots frame. The number of a miss for 3 slots frame will be 9, whereas the MISS is 10 for 4 numbers of slots in a frame.

Example 2.4: Let us consider a block referencing strings 1, 3, 0, 3, 5, 6, 3 with 4 block frames. Find the number of misses.

Solution: Initially the frame is empty, so for the first three elements 1, 3, and 0, there will be three miss consecutively. Further referencing has been shown in the following figure 2.7.

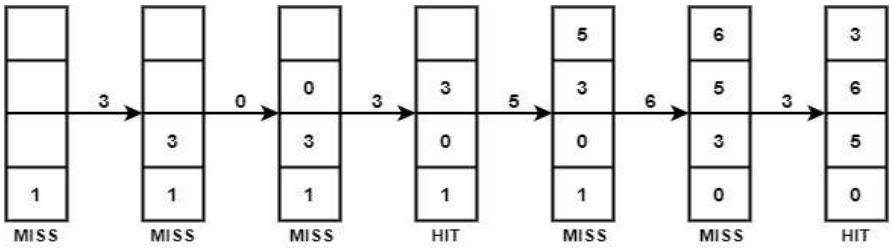


Figure 2.7 Example of FIFO replacement algorithms

A third iteration when 3 comes, is already in the queue, so one hit occurred. Again at the 7th iteration when 3 come, one hit occurred. At steps 5th when 6 come, the data do not exist in the queue. The element entered first into the queue i.e. 1 will be replaced by 6 as shown in figure 2.7.

Space for learners:

2.6.4 Segmented LRU (SLRU) algorithm

SLRU algorithm divided the cache memory into two parts as protected and unprotected. The protected part is reserved for the most used objects. One the first request for an object is done by the CPU; it has been transferred into the unprotected section.The least recently used technique is used to manage both the portion. A count variable

2.6.5 Optimal block replacement

In this method that block will be replaced from the cache which would not be used for a longer period in the future. Optimal page replacement is theoretically perfect, but the operating systems could not know or guess the future request.

Example 2.5 Consider a set of cache block references as 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 slots frame. Find the number of MISS that occurred during cache access.

Solution: Initially there will be four miss for the first four data items 7, 0, 1, and 2 as the slots were empty.

0 is already exit, **0 – MISS or HIT**,

When 3 came, the algorithms identified 7 as the not-used item for the longest period in the future and replace it by 3. – **1 MISS.**

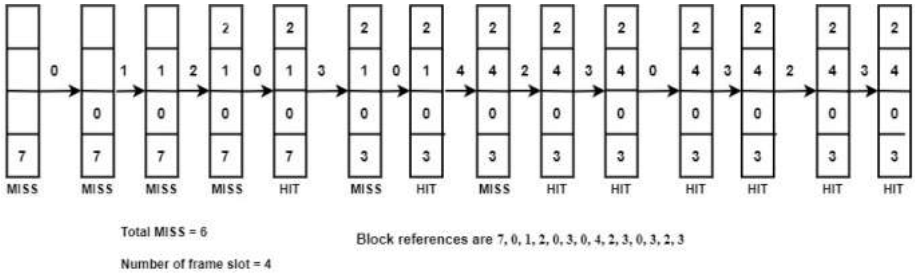


Figure 2.8 Example of Cache optimal block replacements

0 is already exit, **0 – MISS**,

When 4 came, the algorithms identified 1 as the not-used item for the longest period in the future and replace it by 4. – **1 MISS**. Thus there will be 6 misses.

2.6.6 Random replacement (RR) algorithm

The RR algorithm randomly selects any of the data items from the cache memory and replaces it with the required one. It never keeps track of the history of removable data items and does not follow any data structure.

2.6.7 Pseudo – least recently used (PLRU) algorithm

It is one of the most popular and common block replacement policies for the current generation's cache memory. It is widely used by the industry and a common policy for AMD and INTEL products. It uses the data structure binary tree for saving the status of cache memory and hence it is also known as Tree – LRU. The tree structure is used to identify the block position which is to be replaced in case of a miss.

Space for learners:

2.6.8 Lowest latency first (LLF)

LLF algorithm keeps the average and minimum latency by removing the objects with the lowest latency. It shows the best performance during the execution of database queries in the relational database.

All the algorithms discussed above can be classified into several classes such as –

a. Recency-based algorithms,
b. Frequency-based algorithms
c. Function-based algorithms
d. Randomized algorithm

2.7 CACHE OPTIMIZATION TECHNIQUE

Cache optimization can be achieved by reducing the miss penalty, miss rate, and hit time and increasing the cache bandwidth. These can be obtained using different optimization techniques.

To decrease the gap between CPU cycle and memory latency, a multilevel cache can be used. Generally, the cache memory can be categorized into three levels such as L1, L2, and L3 cache. L1 is comparatively the smallest and fastest cache memory in comparison to L2 and L3 levels. It is located within the CPU itself and hence it is called on-chip memory. L2 is faster than L3 cache. L3 is larger and slower in comparison to other levels of cache memory. In a multiprocessor system, each processor hasownL1 and L2 cache memories and the L3 cache is shared by all processor. Miss rate of L1 cache can be reduced by introducing L2 cache.

User-level cache-control (ULCC) is another technique through which space allocation in the cache can be controlled by the user.

Space for learners:

Less hit rate and minimal cache pollution can be produced using this technique. The implementation is very complex.

Cache memory optimization can be achieved by optimizing the loop in compiler-level implementation. A set of compiler algorithms are being used to predict the data to be reuse in near future. This will help to achieve a better hit ratio in cache access.

The performance of the cache can be improved by producing the next data to be used by the cache. To produce the next data a process called data perfecting can be used in advance.

Space for learners:

2.8 WRITING INTO THE CACHE

The cache is a technique to keep a copy of one or more blocks of data from the main memory into the fastest memory storage such that the processor can access it easily. Cache mostly works as a buffer in between the processor and RAM and increases the speed of the processor by making the data available. Whenever the CPU wants to write data or a word, at first it checks the address where the word to be written is available in cache or not. If the address is available in cache memory then it is known as a **write-hit**. During the write operation, if the main memory is not updated simultaneously, it may lead to inconsistency of data. That is the content in cache and main memory may be different for the same reference address. If the system memory is sharing with more than one device then problems may arise due to this inconsistent data. Hence the two methods write through and write back is come into the picture to perform the write operation in cache effectively and efficiently. If the referred address is not available in the cache during a written request a **write-miss** will occur. For a write-miss, two other processes are being used to maintain the data consistency

in cache and main memory namely **write allocation** and **write around**.

2.8.1 Write through

When the number of the write operation in cache is less than this process is used. It is comparatively simpler and reliable to perform the write operation. In write-through, both the memory cache and main memory are updated simultaneously. During a write operation, a data or word has to write in both the memory locations and due to this, the write-through process experienced delays in the write operation. This process has been solved the problem of inconsistent data but raisesa question about the use of cache memory during the write operation. Because the access of main memory along with the cache during writes operation increases the cost of the write operation and decreases the CPU performance.

2.8.2 Write back

In this process, the cache is updated during the write operation and the main memory is updated later.

2.8.3 Dirty bit

A status bit is used to indicate whether the data present in the cache memory is modified or not during a write operation. It is known as dirty bit. If the status bit is set to clean-bit, no need to update the main memory later as the data is not modified in the cache. For a dirty bit, the main memory has to be updated as it represents that the cache has been updated during the write operation. But if power fails due to any cause the modified data will be lost in the cache. Lost data from the cache cannot be restored.

Space for learners:

2.8.4 Write allocation

In this process, data has to be loaded from the main memory into cache memory and then updated. It works with both write-through and write-back processes.

2.8.5 Write around

Write around process allows for writing or updating the main memory without interrupting the cache memory.

2.9 CACHE COHERENCE

During the write operation of cache memory, data inconsistency may occur among adjacent or within the same level of the memory hierarchy. It is possible to have many copies of one instruction operand in a shared memory multiprocessor system. If any operand value is changed in one memory, then it should reflect in the main memory as well as all the levels of cache memories simultaneously.

Consider a system with three processors P1, P2, and P3. The P1 reads the data X with the value 25 from the main memory and stores it into the cache. The P2 also reads the same data X = 25 from the main memory and stores it into the cache. In the next instruction cycle, the P1 writes the X as 55 locally into cache but not updated into the RAM. If the P3 reads the data from RAM, what value it will get against X? For the main memory and P2, it will be 25, whereas for P1 it will be 55. Thus in caches write operation can create multiple copies of data in different levels of cache and main memory which may lead to the cache coherence problem.

Cache coherence can be defined as a protocol or discipline which ensures that the values of shared operands are propagated throughout the system in a timely fashion.

Space for learners:

Generally, cache coherence may occur from three different sources of inconsistency problem –

i. Sharing of writable data
ii. Process migration
iii. Input / Output (I/O) activity.

Space for learners:

2.9.1 Sharing of writable data

When two processors P1 and P2 read the same word X from shared memory into their local cache and P1 writes to the word as X1 using the write-through method, then the shared memory will be updated from X to X1. Now when the P2 will read the data from its local cache it will be X which is become outdated as shown in Figure 2.9.

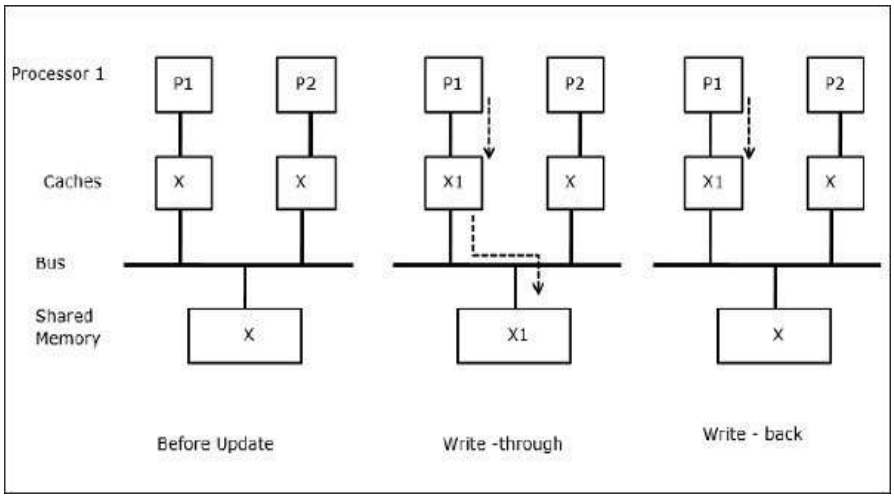


Figure 2.9Sharing of writable data

2.9.2 Process migration

Consider that the process P1 has a data operand X and P2 does not hold any data in its cache. The process P2 first writes on data operand X as X1 and then migrated to P1. Now the process P1 starts reading outdated data X. So P1 writes the data operand X onto the main memory and migrated to P2 as shown in figure 2.10. After migration P2 will start reading the data element and found X in the main memory which is outdated for P2.

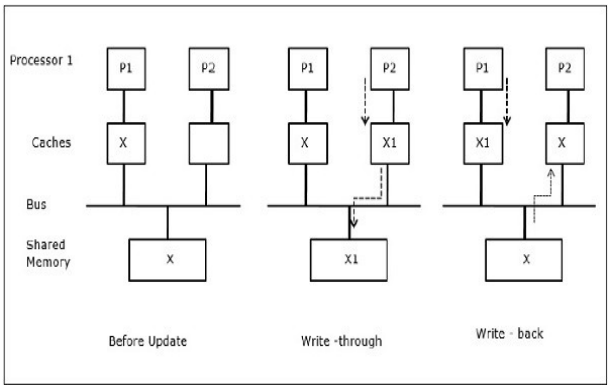


Figure 2.10Process migration

2.9.3 I/O activity

As shown in Figure 2.11, an input/output device is added to the bus in a multi-processor system. As shown in the figure initially both the processor P1 and P2 holds the data operand X. If the I/O peripheral write the data operand as X1 into the main memory, then the process P1 and P2 will get outdated data in the successive read operation. Then the process P1 will modify the operand as X into the main memory as well in the local cache. Now if the I/O device wants to transfer the data, it will get a copy of outdated data.

Space for learners:

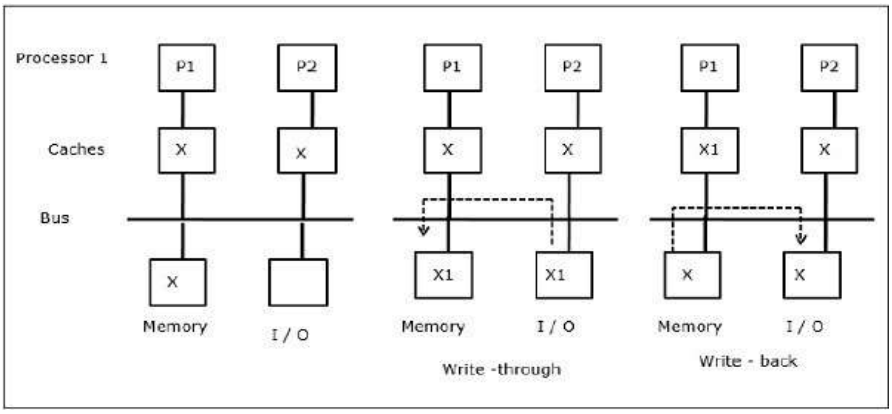


Figure 2.11I/O activity

2.10 COHERENCY MECHANISM

Coherency mechanisms are categorized into four categories –

2.10.1 Directory-based

In this method, the data which is to be shared is placed into a common directory, which helps to maintain the cache coherency. Before each read / writes operation by the processor from the main memory into the local cache, the common directory have to be checked once. Once the directory is changed by any processor, immediately invalidates or updates the other cache with that entry.

2.10.2 Snooping

It is a process where the individual cache monitors the address lines for checking the memory location where the cache is mapped. When a write operation is observed at that location in the main memory, the cache controller invalidates its copy of the snooped memory location. It is known as the write invalidate protocol.

Space for learners:

2.10.3 Snarfing

It is quite similar to snooping. This method is used to monitor both the memory location that has been cached as well as the actual information that is store in the main memory. During a memory write operation, the cache can be updated by new data.

Space for learners:

Check Your Progress:

1. **Choose the correct options from the following for each question:**

a. Assume that there are 3-page frames that are initially empty. If the page reference string is 1, 2, 3, 4, 2, 1, 5, 3, 2, 4, 6, the number of page faults using the optimal replacement policy is_____.

(A) 5
(B) 6
(C) 7
(D) 8

b. Consider the virtual page reference string 1, 2, 3, 2, 4, 1, 3, 2, 4, 1 on a demand paged virtual memory system running on a computer that main memory size of 3 pages frames which are initially empty. Let LRU, FIFO, and OPTIMAL denote the number of page faults under the corresponding page replacements policy. Then

(A) OPTIMAL < LRU < FIFO
(B) OPTIMAL < FIFO < LRU
(C) OPTIMAL = LRU
(D) OPTIMAL = FIFO

c. A virtual memory system uses First in First out (FIFO) block replacement policy and allocates a fixed number of frames to a process. Consider the following statements:

P: Increasing the number of page frames allocated to a process sometimes increases the page fault rate.
Q: Some programs do not exhibit locality of reference.

Which one of the following is TRUE?
(A) Both P and Q are true, and Q is the reason for P
(B) Both P and Q are true, but Q is not the reason for P.
(C) P is false, but Q is true
(D) Both P and Q are false

d. A process has been allocated 3-page frames. Assume that none of the pages of the process are available in the memory initially. The process makes the following sequence of page references (reference string): 1, 2, 1, 3, 7, 4, 5, 6, 3, and 1
If an optimal page replacement policy is used, how many page faults occur for the above reference string?
(A) 7
(B) 8
(C) 9
(D) 10

e. A system uses 3-page frames for storing process pages in the main memory. It uses the Least Recently Used (LRU) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below?
4, 7, 6, 1, 7, 6, 1, 2, 7, 2
(A) 4
(B) 5
(C) 6
(D) 7

f. The optimal page replacement algorithm will select the page that
(A) Has not been used for the longest time in the past.
(B) Will not be used for the longest time in the future.
(C) Has been used least number of times.
(D) Has been used most number of times.

g. Consider a virtual memory system with a FIFO page replacement policy. For an arbitrary page access pattern, increasing the number of page frames in main memory will

Space for learners:

(A) always decrease the number of page faults
(B) always increase the number of page faults
(C) sometimes increase the number of page faults
(D) never affect the number of page faults

h. A system uses a FIFO policy for page replacement. It has 4-page frames with no pages loaded, to begin with. The system first accesses 100 distinct pages in some order and then accesses the same 100 pages but now in the reverse order. How many page faults will occur?

(A) 196
(B) 192
(C) 197
(D) 195

i. Which of the following is not a written policy to avoid cache coherence?
(A) Write through
(B) Write within
(C) Write back
(D) Buffered write

j. The transfer between CPU and cache is _____.
(A) Block transfer
(B) Word transfer
(C) Set transfer
(D) Associative transfer

k. Which of the following is a common cache?
(A) DIMM
(B) SIMM
(C) TLB
(D) Cache

l. How many possibilities of mapping does a direct-mapped cache have?
(A) 1
(B) 2
(C) 3
(D) 4

Space for learners:

m. In which writing scheme does all the data writes go through to the main memory and update the system and cache?
(A) Write-through
(B) Write-back
(C) Write-buffering
(D) No caching of writing cycle

n. In which writing scheme does the cache is updated but the main memory is not updated?
(A) Write-through
(B) Write-back
(C) Write-buffering
(D) None of these

o. What is the main idea of the writing scheme in the cache memory?
(A) Debugging
(B) Accessing data
(C) Bus snooping
(D) Write allocate

2. Answer the following questions and fill up the bllanks:
(A) Which cache has a separate comparator for each entry?
(B) What is the disadvantage of a fully associative cache?
(C) Which mechanism splits the external memory storage into memory pages?
(D) Which of the following cache mapping can prevent bus thrashing?
(E) Which cache mapping has a sequential execution?
(F) Which address is used for a tag?
(G) What do you mean by locality of reference?
(H) The number of failed attempts to access memory, stated in the form of a fraction is called as _____.
(I) The extra time needed to bring the data into cache memory in case of a miss is called as _____.
(J) The counter that keeps track of how many times a block is most likely used is _____.

Space for learners:

2.11 SUMMING UP

1. Cache memory is smaller in size and one of the faster memory used in a computer system.
2. The cache is used to place in between CPU and RAM.
3. The memory access time of cache memory is very high in comparison to the main memory
4. L1 cache and L2 cache may embed on the CPU chip, hence it is known as an on-chip cache.
5. The cache is a very high-speed memory and is used to increase the processing speed by making the data available to the CPU at a rapid rate.
6. Cache works as a buffer between the CPU and the RAM.
7. Performance of cache memory is measured in terms of hit-ratio.
8. If the CPU finds the referred address in the cache then it can be defined as a hit.
9. If the CPU does not find the referred address in the cache then it can be defined as a miss.
10. The ratio between the hit and the total amount of address referred by the CPU can be defined as hit-ratio.
11. Through the mapping process, data can be transfer from main memory to cache memory.
12. There are three different types of mapping processes in cache memory such as – associative mapping, direct mapping, and set-associative mapping.
13. In associative mapping, the cache memory contains the data along with the address references of that data in the main memory.
14. Direct mapping divides the main memory reference done by the CPU into two fields – index and tag field.

Space for learners:

15. If a system contains the main memory of capacity $\mathbf{2^m}$ and cache of capacity $\mathbf{2^n}$, then the bits in the index field will be n bits and in the tag field is an **(m-n)** bit in the direct mapping.
16. Cache replacement policies are used to make room for the new data in cache memory if it is full.
17. Some of the cache replacement policies are LRU, LFU, FIFO, RR, etc.
18. Belady's anomaly – proves that it is possible to have more MISS for an increasing number of frames while using the FIFO replacement algorithm.
19. User-level cache-control (ULCC) is one technique for block replacement, through which space allocation in the cache can be controlled by the user.
20. Cache optimization can be achieved by reducing the miss penalty, miss rate, and hit time and increasing the cache bandwidth.
21. Generally, the cache memory can be categorized into three levels such as L1, L2, and L3 cache.
22. When the CPU wants to write into cache and the CPU referred address is available in cache memory then it is known as a **write-hit**.
23. When the CPU wants to write into cache and the CPU referred address is not available in cache memory then it is known as a **write-miss**.
24. To write into cache two methods are used – write through and write back.
25. In the write-through process, both the memory cache and RAM are writing simultaneously.
26. In the write-back process, the cache is used to write first and the main memory is updated later with the help of dirty-bit.

Space for learners:

27. Two other processes write-allocation and write around are used, when a write miss has occurred.

Space for learners:

2.12 ANSWERS TO CHECK YOUR PROGRESS

Answers to the question number 1:

a) C
b) C
c) B
d) A
e) C
f) B
g) C
h) A
i) B
j) B
k) C
l) A
m) A
n) B
o) C

Answers to the question number 2:

(A) Fully associative cache.
(B) Hardware
(C) Index mechanism
(D) N-way set associative
(E) Burst fill
(F) Logical address
(G) The surroundings of the recently accessed block are called the locality of reference.

(H) MISS rate

(I) MISS penalty

(J) Reference counter

Space for learners:

2.13 POSSIBLE QUESTIONS

a. Consider block reference strings 1, 3, 0, 3, 5, 6, and a block frame size 3 is used. Count the cache block miss when the FIFO replacement algorithm is used.

b. Consider the reference string: 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1. Count the number of miss using FIFO page replacement algorithm.

c. Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 block frame. Find number of miss using optimal block replacement technique.

d. Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 block frames. Find the number of misses using the Least recently used method.

e. Consider page reference strings 1, 3, 0, 3, 5, 6 with the frame size of 3. Find the number of misses using the FIFO replacement technique.

f. What is cache memory? Explain the role of cache memory in program statements execution.

g. Explain different cache mapping processes for example.

h. Why block replacement is necessary for cache memory? What are the replacement policies; explain the pros and cons of each.

i. Why cache optimization is required? Discuss any two cache optimization techniques.

j. What types of problems may arise during cache write and how it can be solved? Explain.

k. What is Belady's anomaly? Explain with an example.

l. How the multilevel cache is implemented?

m. Discuss the factors on which the cache optimization techniques are dependent.

n. How in compiler level cache memory can be optimized? Explain.

o. Define the terms: cache access time, efficiency, average access time, hit-ratio, miss, memory access time.

Space for learners:

2.14 REFERENCES AND SUGGESTED READINGS

- William Stallings, Computer Organization and Architecture Designing for Performance, Pearson Education India.
- Carl Hamacher, Zvonko Vranesic, Safwat Zaky, Computer Organization, McGraw Hill Education.
- M. Morris Mano, Computer System Architecture, Pearson Education India.

---×---

UNIT 3: VIRTUAL MEMORY AND PAGING

Unit Structure:

Space for learners:

3.1 INTRODUCTION

Even though the focus of the subject is computer hardware, there is one area of software that needs to be addressed and that is the operating system. An operating system is a software that acts as an interface between a computer hardware and computer user. The operating system manages computer hardware, software resources and allocates resources and services, such as memory, processors and devices. One of the most important function of operating system is memory management that includes the hardware support in processor for paging, virtual memory and segmentation. Virtual memory allows a program with memory space larger than the size of the main memory to be available in the system. This is possible by allowing only that section of the code that is active at that point of time without the need of having all instructions and data of the process being present in main memory at the same time. The

concept of paging and segmentation eliminates the need of allocating main memory to the process in contiguous manner.

Space for learners:

3.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- Explain the mapping of logical address to physical address using paging memory management scheme.
- Analyze and solve problems on paging.
- Explain the working of paging hardware.
- Explain the mapping of logical address to physical address using segmentation.
- Explain the working of segmentation hardware.
- Explain Virtual memory management scheme.

3.3 PAGING

To understand the concept of paging we have to go through the following concepts:

- Process: It is a program in execution or a program placed in main memory for execution.
- Logical Address: It is the address that is generated by the CPU for a program while it is running. As the address does not exist physically it is also called virtual address. The hardware unit of memory known as memory management unit (MMU) maps logical address to physical address.
- Physical Address: A physical address is the actual address in the main memory.

Paging is a memory management scheme that is used to map CPU generated logical address of a process to physical address in main

memory. A process consists of fixed size blocks; Figure 3.1 shows an example of a process with 4 blocks each of size 1 kilobyte. Size of a block depend upon architecture of the computer and varies between 512 bytes to 16 megabytes.

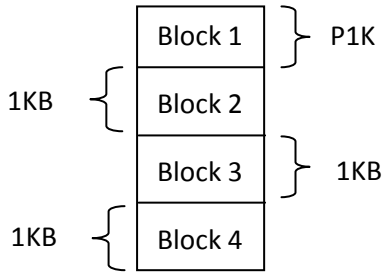


Figure 3.1: A Process with 4 blocks each of size 1 kilobyte.

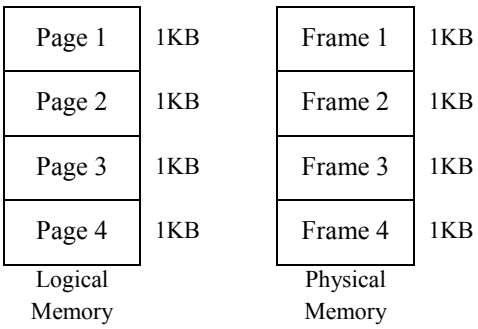


Figure 3.2: A Process with 1KB block size in logical and physical memory.

The paging technique divides the logical memory to blocks of the fixed size known as pagesand divides physical memory into blocks of fixed-size known as Frames. Figure 3.2 shows an example of pages and frames in logical and physical memory respectively.

Paging scheme allows a process to be stored in the main memory in noncontiguous manner. It also solves the problem of searching and fitting blocks of different sizes in main memory by having all block of same size. One more advantage of the paging scheme is that it prevents from external fragmentation that is if the main memory

Space for learners:

blocks are of varying sizes and the size of the free blocks are smaller than the size of the pages, then the operating will be required to merge two or more blocks into a single block large enough to fit a page. By keeping block of equal sizes for both pages and frames, such problems are resolved.Figure 3.3shows paging model of physical and logical memory. A page table is used for mapping between logical addresses and physical addresses. A page table resides in the main memory.Figure 3.3 shows noncontiguous allocation of a process in main memory. The mapping of logical address to physical address is achieved using the page table.

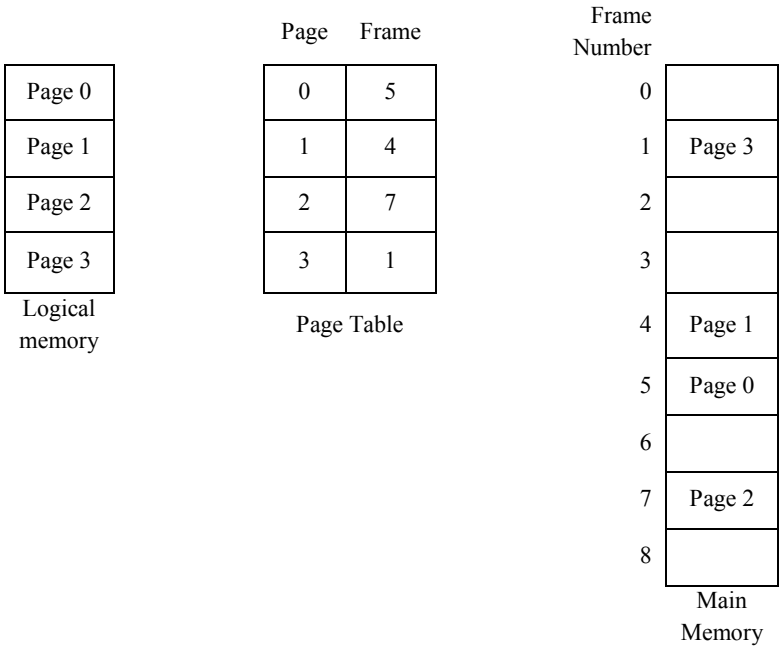


Figure 3.3: Paging model of physical and logical memory.

The hardware support for paging is demonstrated using an example in Figure 3.4. The logical address generated by the CPU is divided into two parts namely *page number* and *displacement* within the page. The page number is used as an index in the page table to search for the corresponding frame number. The displacement is combined with frame number to get the physical address. In the Figure 3.4, the logical address

Space for learners:

having *page number 3* is searched for the corresponding frame number in the page table which is *frame number 15*. The *frame number 15* is combined with the *displacement 7* to form the physical address.

Space for learners:

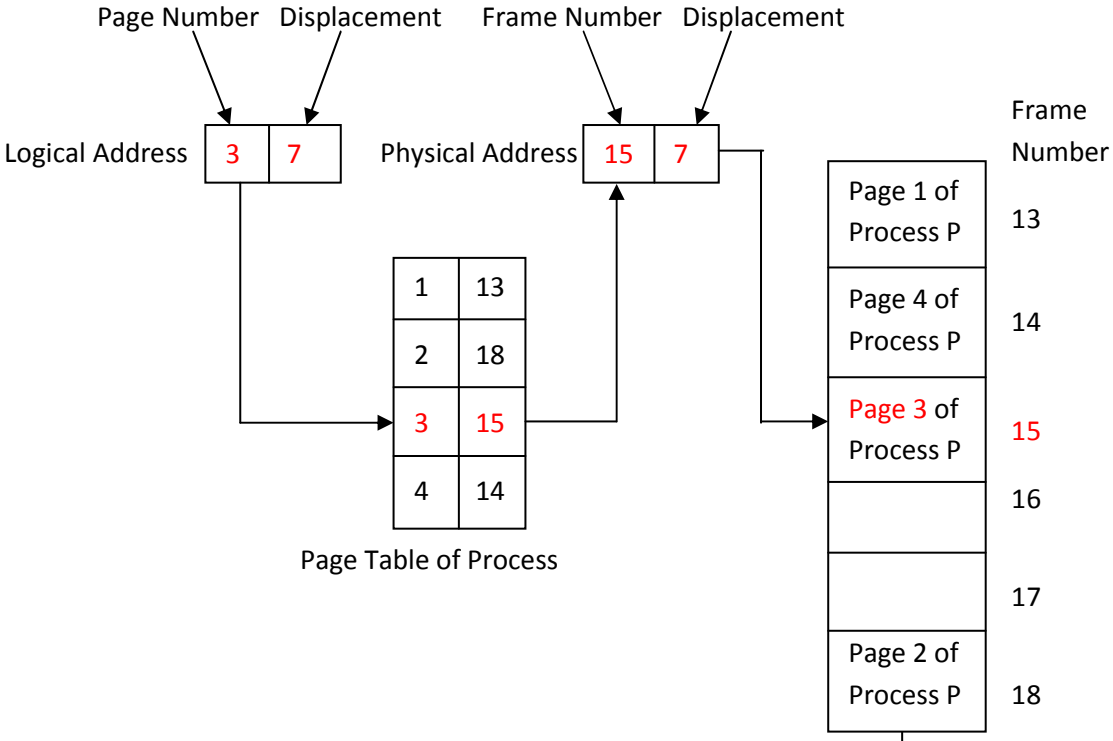


Figure 3.4: Paging hardware support

If the size of the logical address space is 2^m and size of a page is 2^n bytes/words, then "*m-n*" bits of a logical address designate the page number the "*n*" bits designate the displacement or offset. Therefor the logical address is:

Page Number	**Displacement**
p	d
m - n	n

Paging Example -1:
Assume a page size of 1K and a 15-bit logical address space. How many pages are in the system?

Solution:
Page size = 1K = 2^{10} i.e. displacement, n=10 bits
No. of bits in logical address = 15, i.e. m=15 bits.
Therefore, no. of bits used for page number is, m - n = 5 bits
Total no. of pages in the system is 2^5 =32.

Paging Example -2:
Assume that a CPU has a 15-bit logical address space with 8 logical pages. How large are the pages?
Solution:
There are 8 logical pages, that means3 bits are required to address 8 logical pages (2^3 = 8).
Therefore, m - n=3 bits
Logical address is 15 bits, m=15 bits
Displacement = 15 -3 = 12 bits.
So, the pages are of size 2^{12} = 4096 = 4K bytes

3.3.1 Paging Hardware Support

Operating system provides support for storing page table of a process. Generally, a page table can be stored in following ways:

- Set of dedicated registers
- In main memory
- Translation lookaside buffer (TLB)

The feasibility of the first approach using a set of dedicated registers is that the page table should be reasonably smaller in size like 256 entries. With the second approach page table can be very large like millions of entries can be stored in the main memory with a pointer to the starting address of the page table for referencing. However, in this case the time required to access the page table is slower by a factor of two as it involves first accessing memory for the page table to locate the frame number which is combined with the displacement to get the physical address and then a second memory access to read the byte.

Space for learners:

The solution to the disadvantages of the first two approaches is resolved using a fast lookup hardware support called Translation Look aside Buffer (TLB). TLB is a small, expensive but very fast associative memory.

It can store entries in the range of 64 to 1024. Associative memory has two parts: a tag and a value. When a page/key needs to be searched the key is compared simultaneously with all the tags of the in the associative memory.

There are possibly two cases for a page search in TLB. Figure 3.5 illustrates the paging hardware with TLBfor these two cases:

- If the search key/page is found it is called as a TLBhit and corresponding value/frame is returned from the TLB. Displacement is combined with frame number and the physical address is accessed.
- If the search key/page is not found it is called as a TLB miss and the page is searched in the page table stored in main memory. The frame number corresponding to the search page is combined with the displacement to access the address in the physical memory. Also the page number and frame number is added to the TLB so that if the same page is referred next time it is found quickly. In case the TLB is full, operating system selects a page replacement algorithm to replace an existing page with the new entry.

The percentage of times that a particular page number is found in the TLB is called the *hit ratio*. If the hit ratio is 60% that means 60 times out of 100 references,the page will be found in TLB and remaining 40 times the page is found in the page table.

Space for learners:

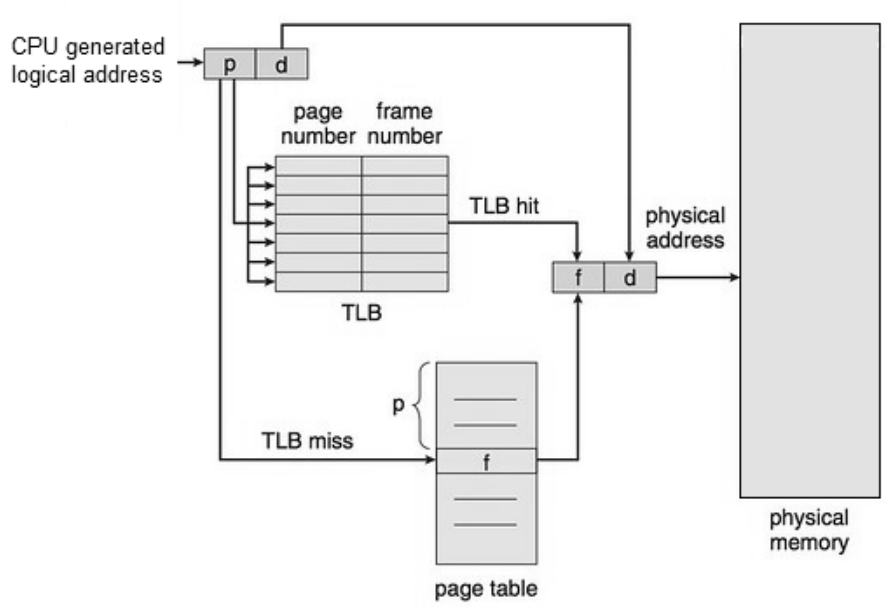


Figure 3.5: Paging hardware with Translation Look aside Buffer [1].

Paging Example -3:
If it takes 25 nanoseconds to search the TLB and 75 nanoseconds to access memory. If the hit ratio is 70%, calculate effective memory access time.

Solution:

If the page is in the TLB, time taken to access the physical address
= Time taken to search the TLB + Time taken to access memory
= 25 +75 =100 nanoseconds

If the page is not in the TLB, time taken to access the physical address
= Time taken to search the TLB + Time taken to access page table
+ Time taken to access memory
= 25 +75 +75
= 175 nanoseconds

Hit ratio is 70%, therefore

Space for learners:

Effective access time = 0.70 X 100 + 0.30 X 175 =122.5 nanoseconds.

Space for learners:

3.4 SEGMENTATION

Segmentation is a memory management scheme similar to paging that allows a process to be stored in the main memory in noncontiguous manner. Unlike paging where all the pages or frames are of fixed size, segmentation allows blocks or segments of variable size. Segmentation maps the user's view of a program onto the physical memory. Looking at the user's view in Figure 3.6, a program contains several variable size segments, such as the main program, subroutine, symbol table, methods etc. It also includes data structures like arrays, objects, variables, stacks etc. These segments and data structures are referred by their name without concerning about the address these segments are stored in memory. Users are not concerned about the order in which the segments are stored in the memory.

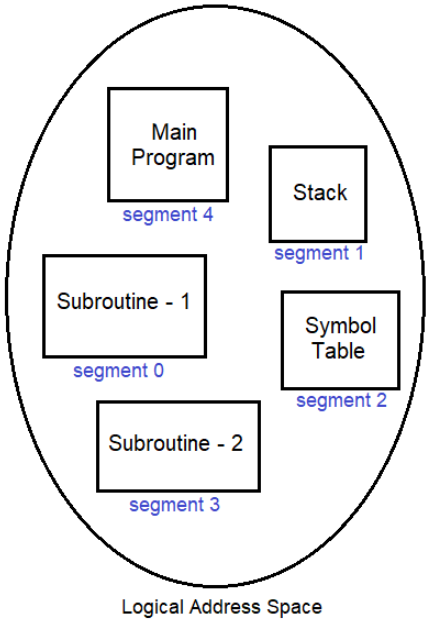


Figure 3.6: User's view of a program

The logical address space is a group of segments. Each segment has a name and a length.From the implementation point of view, segments are numbered instead of using name and the logical address is represented using the *two tuple:*

Segment-number	Displacement

3.4.1 Segmentation Hardware

The mapping of the logical address <segment-number, displacement>to the physical address is achieved with the help of segment table and the segmentation hardware as shown in Figure 3.7. Each entry of the segment table has a segment limit and segment base. The base represents the starting address of the segment in the main memory and the limit specifies the length of the segment. The segment table is indexed on the segment number.

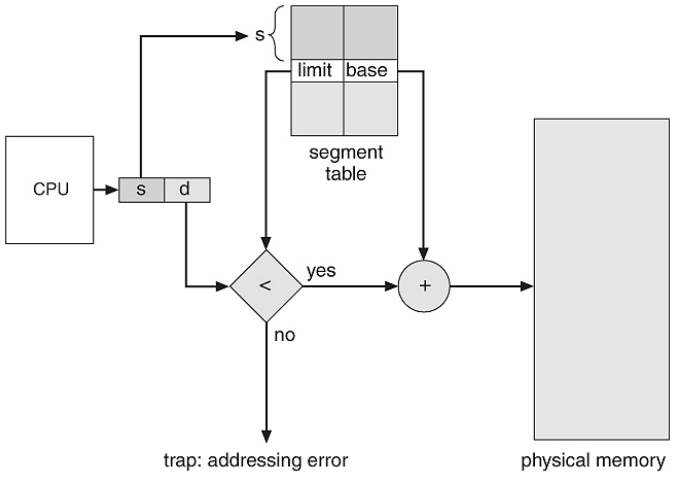


Figure 3.7: Segmentation Hardware[1].

The working of segmentation hardware starts by first identifying the segment number, *s* and the displacement, *d*of the logical address. The segment number is used to search the segment table, which is indexed on the segment number. The displacement, *d*of the logical address should be between 0 and limit. If the condition is not satisfied, it means that the logical address is going beyond the segment limit and a trap interrupt is initiated which is handled by the operating system.

Space for learners:

A segmentation example is shown in Figure 3.8. There are 5 segments numbered from 0 through 4. The segments are stored in physical memory in noncontiguous manner. Also, no specific ordering is followed for storing the segments as can be observed in the example. The segment table has an entry for each of the segment, the starting address of the segment mentioned as *base*and the length of the segment mentioned as *limit*. For example, segment 0 begins at address 5100 and length of the segment is limited to 500 bytes. Therefore, a reference to byte 17 of segment 0 is mapped to 5100 (base of segment 0) + 17 = 5117. Similarly, a reference to byte 88 of segment 4 is mapped to 7300 + 88 = 7388.A trap interrupt will be called if byte 1700 of segment 4 is referenced as the limit is 1500.

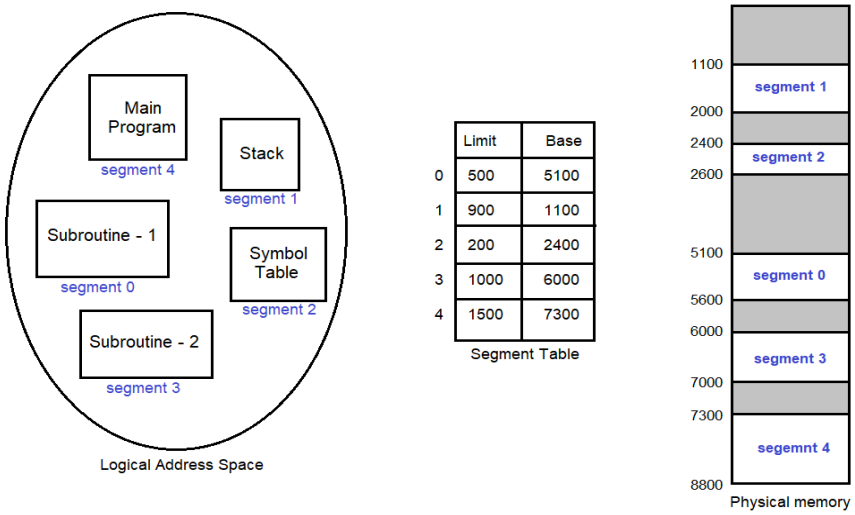


Figure 3.8: Segmentation Example

3.5 VIRTUAL MEMORY

The memory management scheme discussed in previous section requires the entire process to be in the main memory for execution. Most of the times there can be a requirement of many processes to be in the memory simultaneously for execution. This situation can

Space for learners:

prevent simultaneous execution of multiple processes due to the size of the main memory, which may not be large enough to hold all the processes. So the concept of virtual memory was introduced.

A virtual memory management scheme allows execution of a process even if it is not completely in memory. That is, it requires only that section of the code of the process to be in the memory that will be executed. Generally, a process contains several functions or procedures and not all the functions are required to be in the memory at the same time. So the function or the procedure that will be executed needs to be in the main memory while the other functions or procedures can be placed in the secondary memory and wait for their turn of execution. So whenever a function is not available in the main memory, it is brought from the secondary memory to main memory for execution. The main advantage of this scheme is that a program larger than main memory can still run on a smaller physical memory. This is how a games like *Need for speed* or *Call of Duty*which require respectively 30 GB and 90 GB of memory can still run on a system having 6 GB RAM with sufficient hard disk space. Also, as only a section of the process's code needs to be in memory so many process can be there in memory simultaneously. Thereby increasing CPU utilization and throughput.

Space for learners:

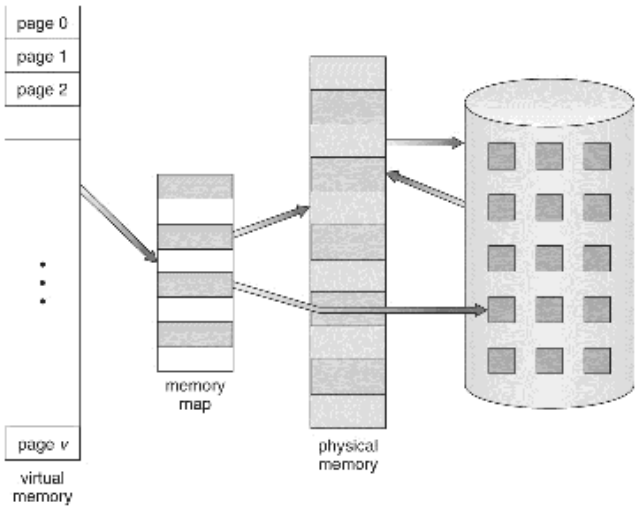


Figure 3.9: Example showing virtual memory larger than physical memory[1].

Figure 3.9 shows an example of a larger virtual memory than physical memory. The programmer thus need not have to worry about the size of the main memory available, thus can concentrate on the problem to be programmed. As can be seen in the Figure 3.9, pages from the large virtual memory address space is stored in the secondary memory and the pages are brought back to main memory whenever a call to those pages are required. If the main memory does not have any free slot for the pages, then some page replacement algorithms are used to replace the pages in main memory with the pages from secondary memory.

3.5.1 Demand Paging

Suppose a user wants to run a program, so the entire program is loaded to main memory from the secondary memory. However, if the program runs only one option/case out of the several cases based on the user input, it is impractical to load the code for all the cases, other cases my never be called for execution. So, a virtual memory technique known as demand paging is used to load only those pages

Space for learners:

of the process when they are required or whenever there is a demand for the page occurs during the program execution.

In Figure 3.10 shows and example of demand paging where pages 4, 5, 6 and 7 of Program A is swapped out of memory and pages 17, 18 and 19 of Program B is moved in to the memory because of the demand for the pages 17, 18 and 19. The method is implemented by a *pager* program responsible for demand paging.

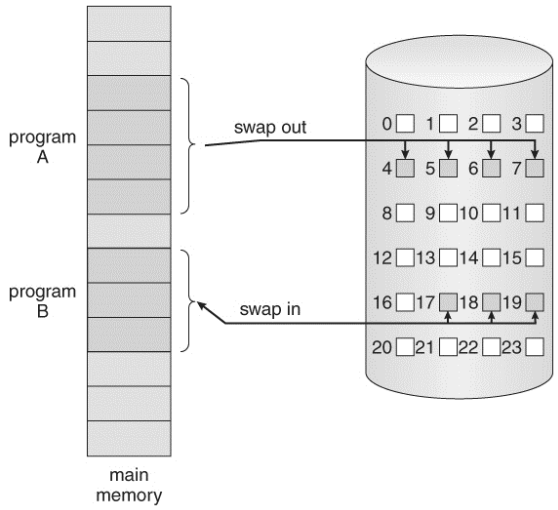


Figure 3.10: Example showing Demand Paging [1].

CHECK YOUR PROGRESS

i. Fixed-sized blocks in physical memory is called _____

a) Block
b) Frame
c) Pages
d) Segment

ii. In paging CPU generated logical address has two parts _____and _____.

a) Page offset & Frame bit
b) Page number & Page offset
c) Frame offset&Displacement
d) Frame number& page offset

Space for learners:

iii. Fixed-sized blocks in logical memory is called _____
   a) Block
   b) Frame
   c) Pages
   d) Segment

iv. Paging does not suffer from _____.
   a) Internal Fragmentation
   b) External Fragmentation
   c) Both a) and b)
   d) None of the above

v. If it takes 10 milliseconds to search the TLB and 80 milliseconds to access the physical memory. If the TLB hit ratio is 0.6, the effective memory access time (in milliseconds) is _____.
   a) 120
   b) 122
   c) 134
   d) 124

vi. The displacement 'd' in a logical address must be _____
   a) Greater than segment limit
   b) Greater than the segment number
   c) Between 0 and the segment number
   d) Between 0 and segment limit

vii. In segmentation, each address is specified by _____
   a) A key and value
   b) A displacement and value
   c) A segment number & displacement
   d) A value and segment number

viii. A CPU generated memory larger than main memory is called as
   a) Logical Memory
   b) Secondary Memory
   c) Virtual Memory

Space for learners:

d) All of the above

ix. The virtual memory manager loads only those component of a program during execution as a when required is known as _____.
   a) Segmentation
   b) Swapping
   c) Virtual memory
   d) Demand Paging

x. Virtual memory can be implemented with
   a) Swapping
   b) Paging
   c) Segmentation
   d) Both b) and c)

Space for learners:

3.6 SUMMING UP

- Logical address is the address that is generated by the CPU for a running program.
- A physical address is the actual address in the main memory.
- Paging is a memory management scheme that is used to map CPU generated logical address of a process to physical address in main memory.
- The logical address generated by the CPU is divided into two parts namely page number and displacement with the page.
- Translation Lookaside Buffer is a small, expensive but very fast associative memory.
- In a TLB, if the search page is found it is called as a TLB hit if the page is not found it called as TLB miss.
- The percentage of times that a particular page number is found in the TLB is called the hit ratio.

- Segmentation is a memory management scheme similar to paging that allows a process to be stored in the main memory in noncontiguous manner.
- The mapping of the logical address <segment-number, displacement> to the physical address is achieved with the help of segment table and the segmentation hardware.
- A virtual memory management scheme allows execution of a process even if it is not completely in memory.
- A virtual memory technique known as demand paging is used to load only those pages of the process when they are required or whenever there is a demand for the page occurs during the program execution.

Space for learners:

3.7 ANSWERS TO CHECK YOUR PROGRESS

i, b	ii, b	iii, c	iv, b	v, b
vi, d	vii, c	viii, c	ix, d	x, d

3.8 POSSIBLE QUESTIONS

Q1 Differentiate betweenphysical and logical address space.
Q2 Explain paging memory management scheme.
Q3 Define a page table. Why it is needed in paging?
Q4 What is hit ratio? Why page should be replaced in the memory?
Q5 Explain the working of a paging memory management scheme.
Q6 Consider a logical address space of 16 pages of 512 words each, mapped on to a physical memory of 64 frames. How many bits are there in the logical address? How many bits are there in the physical address?

Q7 If it takes 125 nanoseconds to search the TLB and 500 nanoseconds to access memory. If the hit ratio is 90%, calculate effective memory access time.

Q8 Assume a page size of 4K and an 18-bit logical address space. How many pages are in the system?

Q9 Assume that a CPU has a 16-bit logical address space with 4 logical pages. How large are the pages?

Q10 What is segmentation? Explain.

Q11 Define a virtual memory. With a neat diagram, explain the working of a virtual memory. What are the benefits of a virtual memory?

Q12 What is demand paging? Explain.

Q13 What is the benefit of demand paging?

Q14 Consider logical address 1025 and the followingpage table for some process P0. Assume a 15-bit address space with a page size of 1K. What is the physical address to which logical address 1025 will be mapped?

6
2
3

Q15 Consider the following segment table:

Segment	Base	Length
34	100	100
21	2500	200
0	1200	50
90	1700	300
7	500	500
2	600	50
99	650	200

What are the physical address for the following logical address?

i. 0,25
ii. 2,89
iii. 90,345

Space for learners:

iv. 34,50
v. 99,201

3.9 REFERENCES AND SUGGESTED READINGS

- Computer Organization and Architecture, 10th edition, William Stallings, Pearson.
- Computer System Architecture Third Edition, M. Morris Mano, Rajib Mall, Pearson
- Computer Organization, 5th Edition, Carl Hamacher, McGraw Hill
- Operating System Principles 8th edition by Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin, Willey

---×---

Space for learners:

UNIT 4: BASIC I/O SYSTEM-I

Space for learners:

Unit Structure:

4.1 INTRODUCTION

Input and Output (I/O) devices are integral parts of computer systems. I/O devices and I/O modules are the functional units of a computer along with the Central Processing Unit (CPU) and the memory units. There exist a wide variety of I/O devices having different characteristics. Thus I/O devices are not directly connected to the CPU; rather they are connected via I/O modules. I/O modules take the responsibility of establishing the communication between the CPU and I/O devices by bridging the gap between an I/O device and the CPU. Each I/O module connects with the system bus or to the central switch. An I/O module can control more than one device.

This unit will provide an understanding of basics of I/O interfacing. We begin this chapter with an overview of bus interconnection and bus arbitration, and then we illustrate the functioning of I/O operations via I/O module. At the end of the unit we present the basics of interrupts and direct memory access (DMA).

Space for learners:

4.2 UNIT OBJECTIVES

On completion of this unit students will be able to:

- Explain the basics ofbus structures, bus arbitration and roles of different buses.
- Get familiarized with various input output devices.
- Comprehend various aspects of input outputinterfacing.
- Learn the functioning of input output modules
- Understand the significance of interrupts in communicationwith input output.
- Learn the concept of data transfer using direct memory access

4.3 BUS INTERCONNECTION

A bus is a pathway via which two or more devices can perform data transfer. Buses are shared transmission media; multiple units can use the same bus for the data transfer but at a time only one unit can send data.A bus can be used to connect either the major components of a computer or the internal components of a CPU or two different computers.

Typically a bus is comprised of multiple lines. Each line can transmit a single bit (0 or 1); thus it can transfer a group of bits in parallel in a single transfer.The number of bits that can be

transferred in parallel is called as the bus width. For example, an 8-bit wide bus can transmit 8 bits at a time.

Computer systems have different types of buses for different levels of communications. The internal components of a CPU are connected via internal CPU bus. The major components of a computer system, i.e., the CPU, memory modules and I/O are connected via a special type of bus called as *system bus*.

STOP TO CONSIDER

Buses are used by different modules of a computer to transfer data to other modules. Via buses various forms of data are transferred.

4.3.1 Structure of Bus

As mentioned earlier, a system bus is a common bus shared by the CPU, memory and the I/O. A typical system bus comprises of about 50 to hundreds of separate lines. The connected modules can send different types of information such as data, address and control signals over these lines. Thus lines are usually divided into three groups: data, address and control lines. The schematic diagram of a typical system bus structure is shown in Fig 4.1.

The **data lines** or **data bus** is used to transfer the data among the components attached to it. The width of data bus of a contemporary machine can be 32, 64 or even more. This width determines the amount of data that can be transferred at a time. The width of the data bus is a key parameter to determine the performance of the system. For example, if the length of an instruction is of 64 bit, then the processor would need to access memory only once if the data bus if 64-bit wide; on the other hand if the data bus is of 16-bit, then

Space for learners:

the processor would need to access the memory 4-times to fetch the 64-bit instruction. Thus wider the bus faster will be the data transfer.

The **address lines**, also known as **address bus**identifies the location of the source or the destination of the data available on the data bus. For example, if the processor has to read the data from memory location X, then it places the address X onto the address bus. The width of the address bus determines the system's memory capacity. For an instance, a system with 16-bit address bus can support a memory of 2^{16} blocks. Moreover, the address bus is also used to locate an I/O port. The higher order bits usually identify a particular I/O module and the lower order bits identify the particular port of the selected module.

The **control lines** or **control bus** are used to carry control signals and timing information to the various computer components. Control signals help in enabling a system to understand what has to be done and timing information indicates the validity of the information available in the data and the address bus. Typical control signals are Memory Read, Memory Write, I/O Read, I/O Write, Bus Request, Bus Grant, Interrupt etc.

Space for learners:

Space for learners:

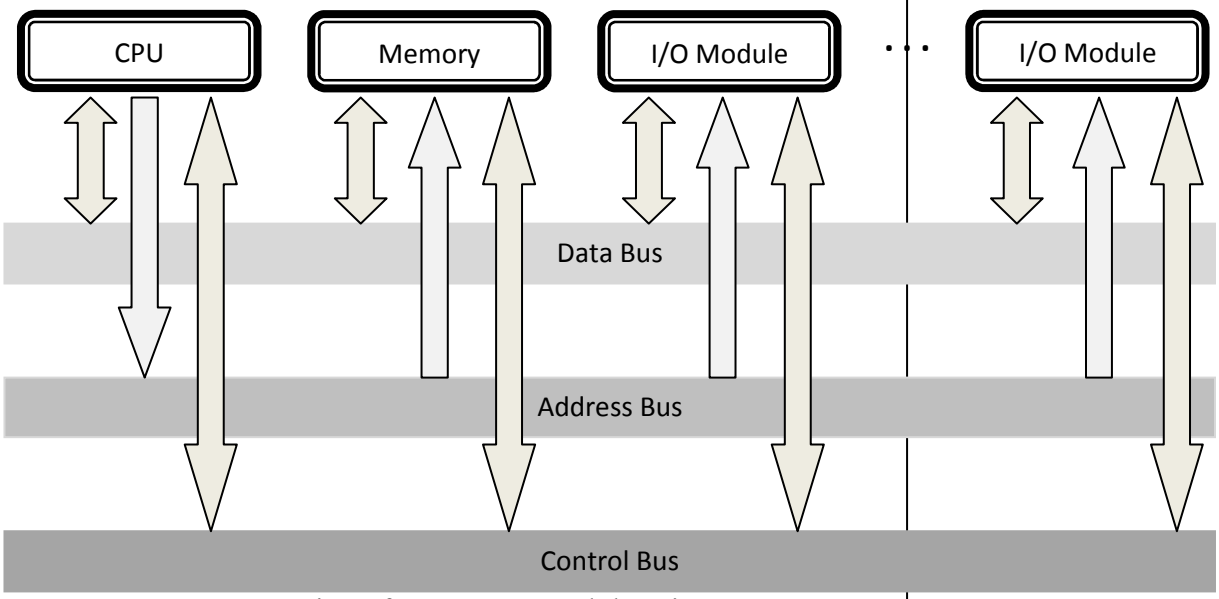


Fig. 4.1 Interconnection of Computer Modules via System Bus

STOP TO CONSIDER

A common bus structure is used to connect the major components of a computer. Such a structure is called as system bus. System bus allows a computer module to transmit data, address and control signals to another module. Thus the bus lines are grouped into data, address and control lines.

4.3.2 Aspects of Bus Design

There are a few aspects whichareneeded to be considered while designing a bus structure. The key aspects are *bus type*, *bus width*, *method of arbitration*, *timing* and *data transfer type*.

Bus Types:

Buses can be categorized into two broad types: dedicated and multiplexed. Dedicated buses are used either for a specific function (e.g., for data or address) or to connect specific physical modules.The advantage of dedicated bus is higher throughput. However, it increases the size as well as the cost of the system.

On the other hand, multiplexed buses are ones either for used multiple functionalities or to be shared amongst multiple physical modules. For example, a common bus can be used to share both data and address information. The main advantage of having multiplexed bus is that it uses of fewer lines which helps in making the system compact as well as cost effective. A major disadvantage of it is that it needed a more complex circuitry for each connecting module.

Bus Width:

We have already addressed the role of the bus width while discussing different bus types. It determines the amount of data that can be transferred at a time. Higher is the width of the data bus higher is the transfer rate. Thus the width of the data bus has an impact on the performance of a system while width of the address bus determines the system's capacity to address memory blocks.

Bus Arbitration:

In case of a shared bus system more than one module may require to have the control of the buses. Typically, the CPU has the main control of the buses; however when a module wishes to perform the data transfer without CPU's intervention then the device which controls the data transfer may need to have the control of the buses. In such a scenario, the CPU has to transfer the control of the buses to the device managing the data transfer. The process of transferring the control of the buses from one device to another is called as bus arbitration. There are basically two types of bus arbitration methods: *centralized* and *distributed*. In case of centralized arbitration, a special hardware called as *bus arbiter* performs the allocation of the buses to the module requiring the buses. This device can be a part of the CPU or can be a standalone module.In distributed method, the modules mutually share the control of the buses without relying on any centralized arbiter.

Space for learners:

Timing:

Space for learners:

Timing is a very important criterion of bus design. It defines a way to coordinate the events occurred on the bus. It can be *synchronous* or *asynchronous*.

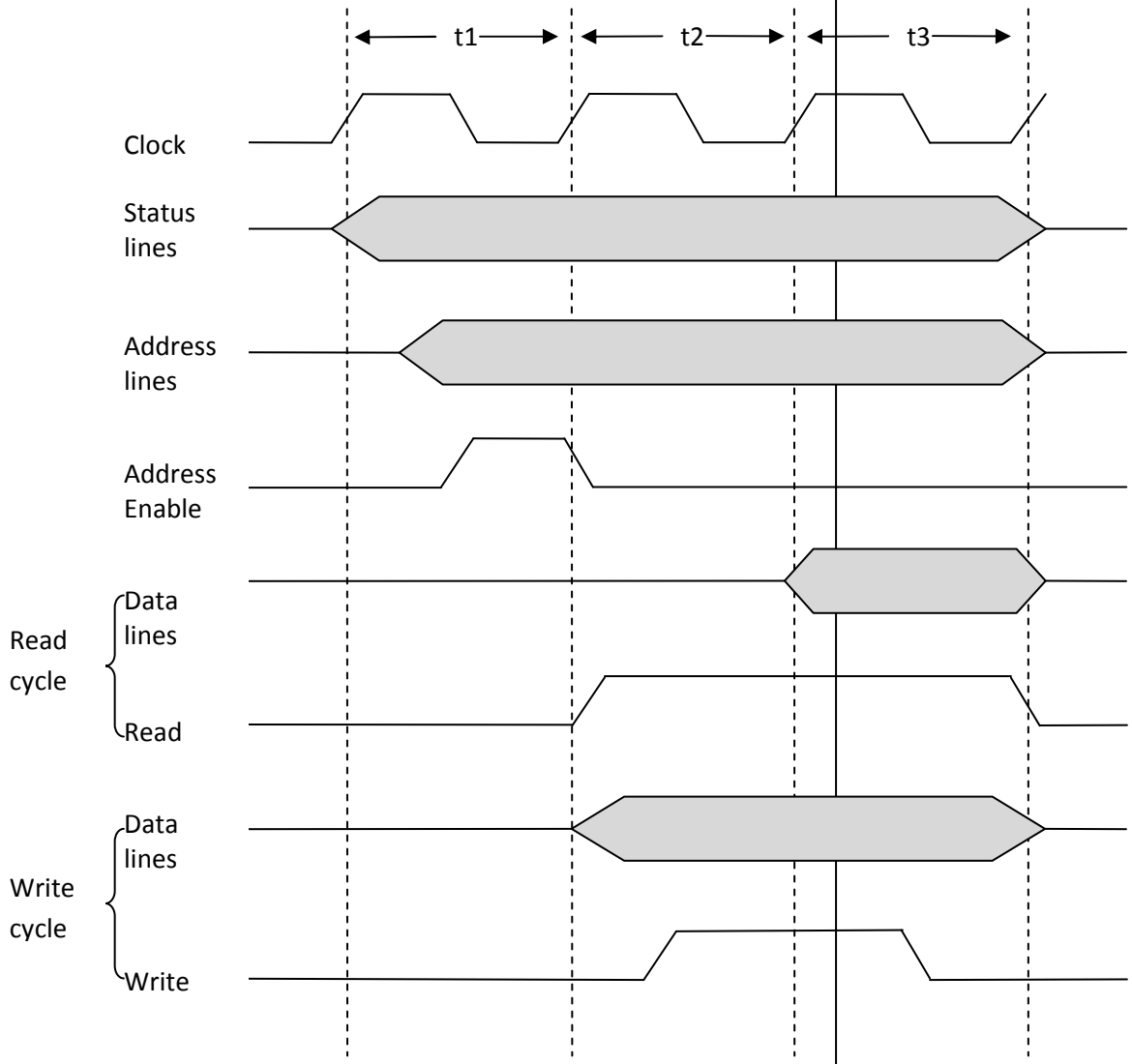
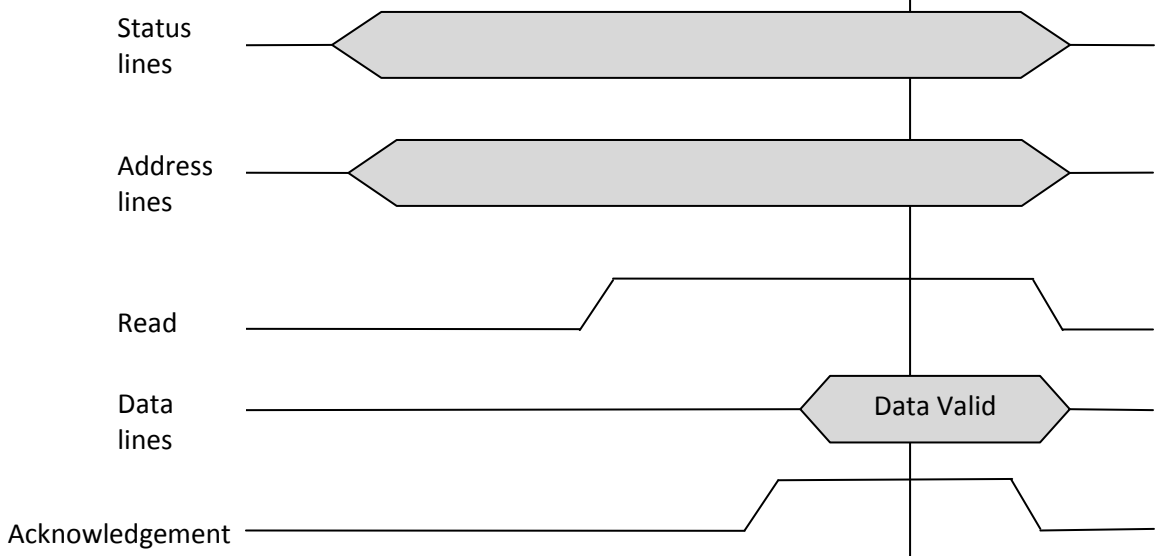


Fig. 4.2 Timing Diagram of Memory Read and Write Cycle

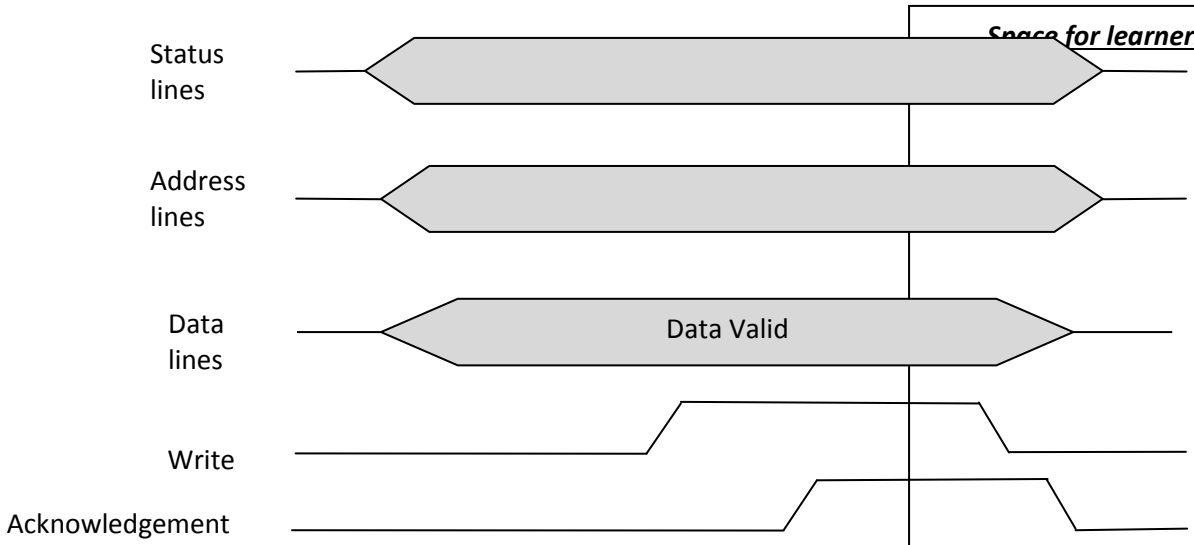
The occurrences of the events in synchronous timing are controlled by the clock. A clock line is attached to the bus that transmits an alternating sequence of 0s and 1s repetitively. One single transition

of 1-0 is termed as one *clock cycle*. A clock cycle defines a slot. All the modules attached to the bus can access the clock line and triggers all events at the beginning of a clock cycle. The Fig. 4.2 presents a sample the timing diagram of both memory read and write cycles. In this example, a memory address is placed onto the address bus at the beginning of a clock cycle. Once the entire address is placed onto the bus, the processor asserts the address enable signal. During read cycle, the processor enables the read signal at the beginning of the second clock cycle; the system identifies the address and places the data from the designated memory address

Space for learners:



(a) Memory Read Cycle



(b) Memory Write Cycle

Fig. 4.3 Timing Diagram of Asynchronous Bus Operations

onto the data bus at the start of the third cycle. The processor reads the data from the bus and disables the read signal on completion of the read operation. During the write cycle, the processor places the data onto the data bus followed by activating the write command. The memory reads the data from the bus during the third cycle.

In asynchronous timing no clock is used to coordinate the occurrence of the events, rather the occurrence of one event depends on a previous event. To coordinate the events, the processor asserts special status signals. During read cycle, the processor first places the address onto the address bus and asserts the status signals. The read command is issued once the address is stabilized to indicate the validity of the address. The memory module recognizes the address and copies the data from the corresponding memory address onto the data bus. The memory module confirms the accomplishment of the transfer of data to the bus by asserting the *acknowledgement* signal. The read signal is disabled once the data is read by the processor. The memory module then drops the acknowledgement

Space for learners:

signal and the processor desserts the read signal. Fig. 4.3(a) demonstrates the sequence of events of the read cycle with asynchronous bus.

During write cycle, the processor places the address, status and data onto the respective buses at the same time. The write signal is asserted by the processor to indicate data valid. The address is recognized by the memory module and fetches the data from the data bus to copy it to the address given. Once write is accomplished, the memory module sends the acknowledgement signal. The write signal is then dropped by the processor or the bus master after on receiving the acknowledgement. The write cycle events with asynchronous bus are shown in Fig. 4.3(b).

STOP TO CONSIDER

To design a bus structure, various criteria like bus type, bus width, type of arbitration and timing are needed to be considered. Based on different parameters chosen for different criteria, the bus has to be designed.

4.4 I/O Devices

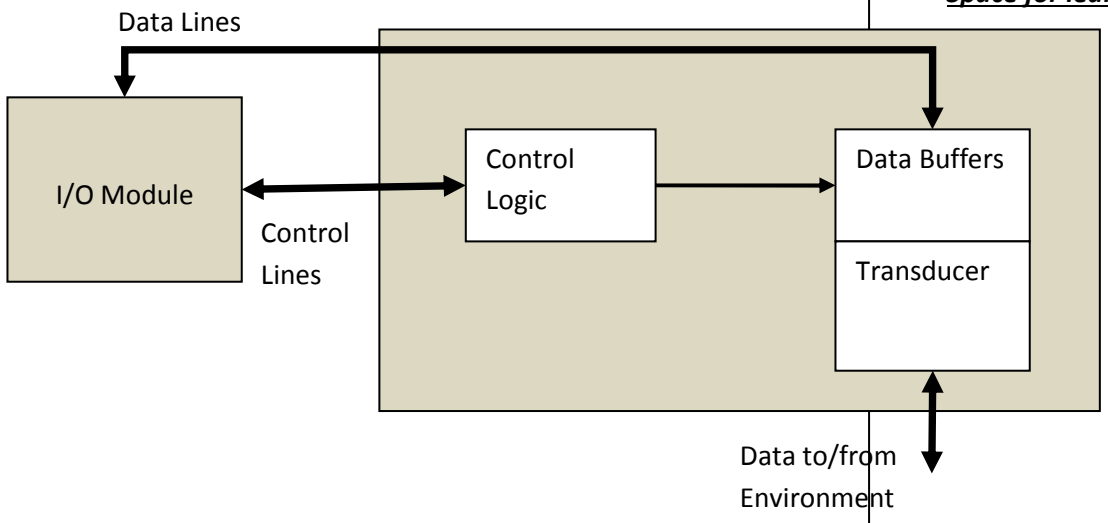
I/O devices are external devices which facilitate the exchange of data between the processor and the external environment. Such devices are also known as *peripheral devices* or simply *peripherals*. An I/O device is connected with the processor via an I/O module port. An I/O device can be used either for input or output or both. Some of the input devices are keyboard, mouse, mic, scanner etc while the output devices include monitor, speaker, printer etc.

I/O devices can be classified broadly into *human readable*, *machine readable* and *communication*. Human readable devices used to allow the users to interact with the computer. These enable the user

Space for learners:

either to give input or to see the output. Keyboard, monitor and printer are some examples of human readable I/O devices. The machine readable I/O devices are used to establish the communications between various devices or components of the computer. The magnetic disks, tapes, sensors and actuators are some examples of machine readable I/O devices. The communication devices are used to transmit data to a remote device. Examples of communication devices include modems, Infrared, Bluetooth and network interface card (NIC). The remote devices can be a human readable device like a terminal or can be a machine readable device or can even be another computer. Fig. 4.4 demonstrates the generic block diagram of I/O device.The control logic performs the controlling of overall operations of the I/O device. It decodes the task to be performed by the device based on the received control signal. It is also responsible for error detection and status reporting to the I/O module. The transducer's job is to convert the data received from the external environment to the format understandable by the device during input operation and converts the data from device understandable to the format which the external environment understands.The data buffers stores data temporarily to be exchanged between the external environment and the I/O module.

Space for learners:



Space for learners:

Fig. 4.4 Generic model of an I/O device

The most common I/O devices that almost every computer possesses are keyboard, mouse, monitor and Disk drives. A brief discussion on these four is presented below.

Keyboard

This is the universal input device for all computers. The keyboard layout is identical to that of a standard QWERTY typewriter. It also has several additional command and function keys. It has between 101 and 104 keys in total. Through this, a user can enter alphabets, numbers and symbols called as characters. Each character is associated with a unique 7 or 8 bit code. One of such code representation is *American Standard Code for Information Interchange* (ASCII). To enter data, you must press the precise combination of keys. The transducer in the keyboard interprets the electrical impulses generated by a keystroke and converts it into its corresponding 7 or 8 bit binary code.

Mouse

Another input device which is used most commonly is the mouse. It has two or three buttons on the top and rolls on a little ball. Different buttons are used to perform different actions. The screen cursors of the mouse move in the direction of mouse movement when you roll it across a flat surface. With the mouse, the cursor moves quite

quickly, providing you more freedom to operate in any direction. Moving using a mouse is easier and faster.

Monitor

It is the most common output device that is common in all the computers. It is a unit that displays the characters entered through the keyboard and to display any message. The message can be in the form of text, image or video. So monitors are also called as video display devices. In market various types of video display devices are available. In earlier time Cathode Ray Tubes (CRT) were used to design the monitors. Such monitors were either monochromatic or colored. Although, CRT monitors are still present, however Liquid Crystal Display (LCD) based monitors are more common in recent time. These monitors have a flat panel display and consume less power than the CRT monitors.

Disk Drive

It is a device used for data storage in the computer. It has mechanisms to exchange data and control signals with an I/O module. An I/O module can perform both read and write operations on the disk drive. The transducer on a fixed-head disk can transform magnetic patterns on the moving disk surface to bits in the device's buffer. The disk arm of a moving-head disk must be able to move radially in and out across the disk's surface.

STOP TO CONSIDER

I/O devices are external devices which enable exchange of data between external environment and the computer. There exists a variety of I/O devices for performing various tasks. Keyboard, mouse, monitor and magnetic disks are the most common I/O devices.

4.5 I/O Interfacing using I/O Modules

A computer is connected with a diverse set of I/O devices. The devices differ largely in terms of data rates, data representations, data formats, word lengths and error conditions. The data rates of the devices differ from the main memory and the processor. Often

Space for learners:

peripheral devices are slower than the processor and the memory. But there are some devices faster than the memory and the processor. So there is a big gap between the processor and any I/O. In such a scenario direct communication between an I/O device and the processor is not easy. To solve this, I/O modules are used as a mediator between an I/O device and the processor. I/O modules interface to the memory and the processor through the system bus, which interface one or more I/O devices by the ports.

Space for learners:

4.5.1 Functions of an I/O Module

As mentioned earlier I/O devices are connected with the processor via the I/O modules. For this, an I/O module needs to interact with both the processor and the I/O devices. Theprocessor initiates the I/O operations and selects the I/O module that connects the target peripheral. The I/O devicessend or receive the data to the I/O module to besent to the processor. The major tasks performed by the I/O module are as follows:

- Control and timing
- Communication between the device and the processor
- Data buffering
- Error Checking

Control and Timing

The processor may need to interact with multiple peripheral devices, memory and buses as per the requirement of the program leading to multiple data transfer among various units.So there must be a proper coordination andsequencing of events in order to avoid any conflict. The events generated by a peripheral device are monitored and synchronized by the connected I/O module. The I/O module controls the activities of the peripheral based on the signals received from the processor.

Communication between the device and the processor

During an I/O transfer, the I/O module performs four major tasks, namely *command decoding*, *status reporting*, *data exchange* and *address recognition*.

***Command decoding*:**The processor sends commands to the I/O module in the form of control signal. The I/O module decodes the command and instructsthe I/O device to perform the necessary task.

***Status reporting*:**As there is a speed mismatch between an I/O device and the processor, it is necessary for the processor toknow the current status of the I/O device before and during any data transfer. The processor requests the I/O module to checkstatus of the I/O device. Typical status signals include ready and busy. The I/O module reports back the status of the I/O device to the processor.

***Data Exchange*:**When the I/O device is ready to send or receive the data, the processor requests the I/O module to initiate the transfer. In case of input operation, the I/O module gets the data from the I/O device and forwards the same to the processor. And for output operation, the I/O module gets the data from the processor and then forwards them to the I/O device.

***Address Recognition*:** To uniquely identify the I/O devices, each device is assigned a unique address. During I/O transfer, the processor refers the I/O devices using their unique address or identifier. The I/O module recognizes the specific I/O device it controlling based on the address received from the processor.

Data Buffering

The data buffering is an essential task that the I/O module has to perform as the data rates of processor or memory is much higher than most of the peripherals. The I/O devices cannot receive the data at the speed of memory or processor. The I/O module buffers data received from memory or processor till the I/O device gets ready to receive the data. Similarly, if the data rates of I/O devices are faster than the memory or the processor, the I/O module buffers data received from I/O device to match the speed of processor and memory.

Error Checking

Errors are inevitable while transferring data over any medium. The error may be mechanical or electrical due to technical malfunctions of the devices or may due to transmission. The transmission errors alter the sequence of bit-pattern of the data. The I/O module

Space for learners:

includes error detecting codes to detect any transmission error. The module checks for error for each every data it receives.

Space for learners:

4.5.2 Structure of I/O Module

The general structure of an I/O module is presented Fig. 4.5. It contains has a register set for storing data, status and control information. The data registers are used to store the buffered data. The status registers stores the current status information. The control information received from the processor is stored in the control registers. The register set is connected with the processor via the data bus. The processor uses the address lines and the control lines to send the address information and command to the I/O modules respectively. The control logic unit recognizes an I/O device based on the address information received via the address lines. It decodes the command received via the control lines. It also has logic to interface with the I/O devices.

STOP TO CONSIDER

Direct exchange of data between the CPU and I/O devices are difficult due to the difference in data transfer rates, data representation and unit of transfer. I/O modules are thus used a third party to establish the communication between CPU and I/O.

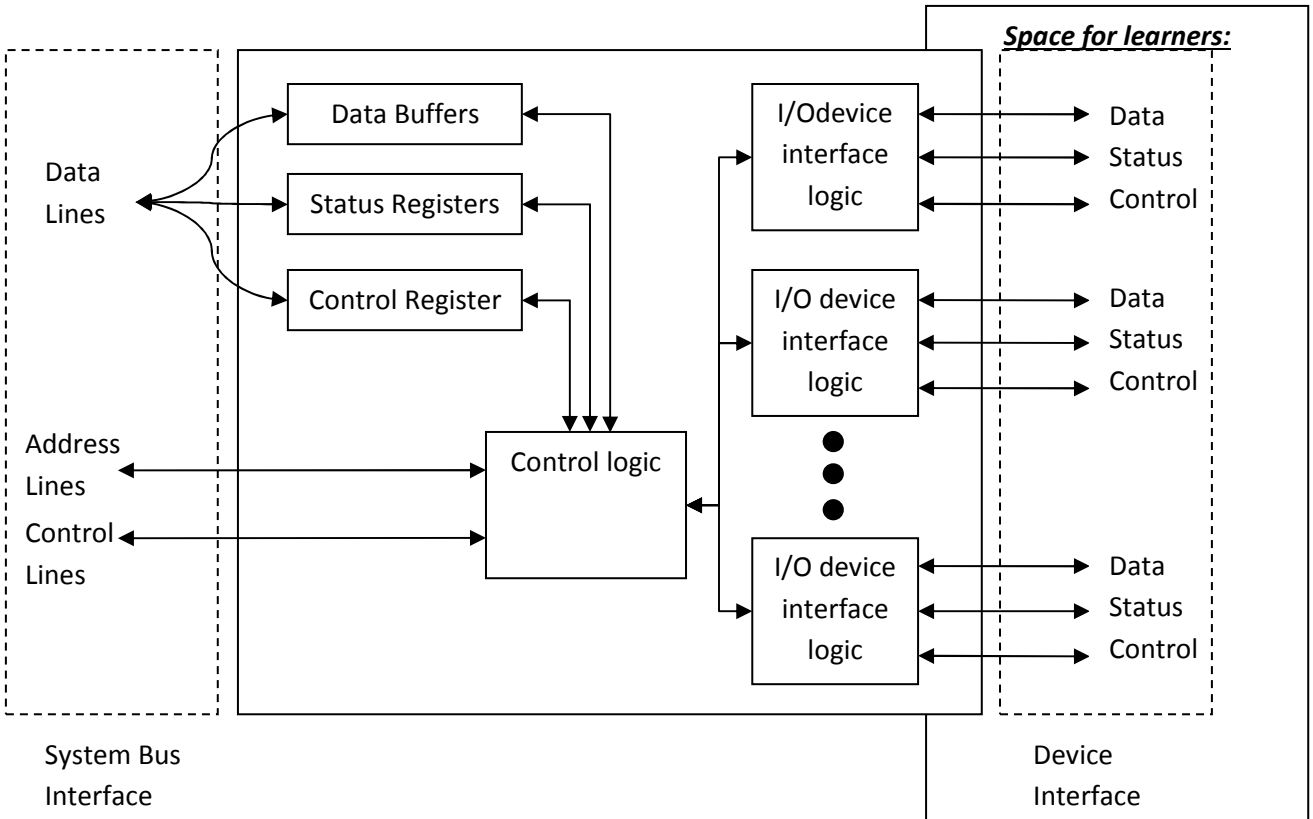


Fig. 4.5 Block Diagram of an I/O Module

4.6 I/O ADDRESSING

The I/O devices are given unique identifiers using any of two addressing modes: *memory mapped* I/O and *isolated* I/O. In *memory mapped* I/O, the I/O devices and memory locations share the same address space. For example, if a system has a 12-bit address bus supporting 4096 unique addresses, then these addresses will be shared among the memory locations and the I/O devices. That means if there is a memory address X, then the address X cannot be assigned to an I/O device. The processor treats I/O transfers exactly same as the memory transfer. Thus, only a single pair of read write lines is required for both memory read/write and I/O read/write. The processor uses the same instructions to access both memory and I/O. The advantage of memory mapped I/O is a large number of instructions are available for I/O operations. However it limits the address space for both memory and I/O.

In *isolated* I/O, memory locations and I/O devices do not share the same address space. If there is a memory address X, then there can be an I/O device with address X as memory locations and I/O have different address space. Thus, a full range of address space is available for both I/O and memory locations. The processor uses different instructions for memory transfer and I/O transfer. It uses separate lines for memory read and I/O read and same holds true for I/O write and memory write. When the memory read/write line is high then the address in the address bus is treated as a memory address and when the I/O read/write line is high then the address in address bus is treated as anI/O address.

Space for learners:

4.7 INTERRUPTS

In computer system, an interrupt is a signal generated by hardware to request the processor to give immediate service suspending the current executions. Hardware interrupts are generally used for handing I/O transfers. As most of the I/O devices are slower than the processor and the memory, the processor does not wait for the I/O to transfer the data. When the I/O is preparing to send or receive data, the processor remains busy with other execution. The I/O device sends interrupt signal to the processor via the I/O module when it gets ready to send or receive data.

For each interrupt, the processor has a routine called as *interrupt service routine* (ISR). This is a special routine that has the code to accomplish the task requested via the interrupt. The processor executes the ISR as a response to the interrupt suspending the current execution. After giving the service to the interrupt, the processor resumes it suspended work.

Apart from hardware interrupts, there are interrupts raised by softwares. These are basically exceptions occurred during the execution of a program. Divide by zero, not a number (NaN), overflow and underflow are some examples of software interrupts.

Space for learners:

4.7.1 Types of Interrupts

A computer system supports a variety of hardware interrupts.These can be broadly classified into two categories: *maskable* and *non-maskable*. Maskable interrupts are the ones that can be ignored. There is a facility to disable such interrupts. These interrupts can be ignored only if they are disabled. The non-maskable interrupts are the highest priority interrupts and cannot be ignored at any cost. Thus, there no option is available to disable such interrupts. TRAP is the example of a non-maskable interrupt.

4.7.2 Interrupt Latency

When the processor suspends the current execution in order to provide the service to interrupt request, it saves the necessary data including the program return address to resume the program execution. The program return address is usually saved onto the processor's stack memory. After saving these data, the program counter is updated by assigning the routine address. This causes a time delay to start the execution of ISR from the time interrupt request has been received. This delay is called as interrupt latency.

STOP TO CONSIDER

When a process or event requires immediate attention, hardware or software emits an interrupt signal.

4.8 DIRECT MEMORY ACCESS

Space for learners:

DMA is a feature of computer systems that allows certain hardware subsystems to access primary system memory (random-access memory) without the intervention of the CPU.

When employing programmed I/O or interrupt driven I/O, without DMA the CPU is often totally engaged for the duration of the read or write operation, leaving it unavailable to execute other tasks. The CPU initiates the transfer via DMA, then does other tasks while the transfer is ongoing, and ultimately receives an interrupt from the DMA controller when the operation is completed.

When the CPU can't keep up with the rate of data transfer, or when the CPU needs to do work while waiting for a relatively slow I/O data transfer, this capability comes in handy. DMA is used by many hardware systems, including disk controllers, graphics cards, network interface cards, and sound devices. In multi-core CPUs, DMA is also employed for intra-chip data transfer. DMA channels allow computers to transport data to and from devices with significantly less CPU overhead than computers without them. A processing element inside a multi-core processor can also transmit data to and from its local memory without consuming processor time, permitting processing and data transfer to happen in parallel.

STOP TO CONSIDER

DMA is technique used to perform data transfer without actively involving the CPU. During the DMA transfer the CPU remains free and can perform some other operations which do not require the system bus.

CHECK YOUR PROGRESS:

i. The key advantage of adopting a single bus structure is that it _____
- **a.** faster transfer
- **b.** ease of access
- **c.** cost effective
- **d.** none of the above

ii. System bus is used to transmit
- **a.** data
- **b.** address
- **c.** control signal
- **d.** all of the above

iii. Width of _____ bus determines the performance of the overall system.
- **a.** data
- **b.** address
- **c.** control signal
- **d.** all of the above

iv. Width of the address bus determines_____
- **a.** the performance of the system
- **b.** system's memory capacity
- **c.** both a and b
- **d.** none of the above

v. Usual bus structure used to connect I/O devices follows
- **a.** single bus structure
- **b.** multiple bus structure
- **c.** star bus structure
- **d.** none of the above

vi. I/O modules are used to overcome difference in _____ between I/O and CPU.
- **a.** speed of data transfer
- **b.** data representation
- **c.** units of data transfer
- **d.** all of the above

vii. Memory mapped I/O has the following advantage over Isolated I/O
- **a.** fewer address lines
- **b.** more instructions for I/O operations
- **c.** bigger buffer space
- **d.** all of the above

Space for learners:

viii. Isolated I/O has the following advantage over Memory mapped I/O
- **a.** fewer address lines
- **b.** more instructions for I/O operations
- **c.** bigger buffer space
- **d.** all of the above

ix. What is the mechanism for synchronizing the CPU with the I/O device in which the device sends a signal when it is ready?
- **a.** DMA
- **b.** interrupt
- **c.** signal handling
- **d.** exception

x. DMA transfer has the following advantage
- **a.** faster data transfer
- **b.** increased CPU throughput
- **c.** both a and b
- **d.** none of the above

Space for learners:

4.9 SUMMING UP

- A bus is a pathway via which two or more devices can perform data transfer. Buses are shared transmission media; multiple units can use the same bus for the data transfer but at a time only one unit can send data.
- The major components of a computer system, i.e., the CPU, memory modules and I/O are connected via a special type of bus called as *system bus*. The system bus has three groups of lines for data, address and control.
- The key aspects of bus design are *bus type*, *bus width*, *method of arbitration*, *timing* and *data transfer type*.
- I/O devices are external devices which facilitate exchange of data between the processor and the external environment. Such devices are also known as a *peripheral device* or simply a *peripheral*. An I/O device is connected with the processor via an I/O module port.
- I/O devices are not directly connected to the CPU; rather they are connected via I/O modules. I/O modules take the

responsibility of establishing the communication between the CPU and I/O devices by bridging the gap between an I/O device and the CPU. Each I/O module connects with the system bus or to the central switch. An I/O module can control more than one device.

- The I/O devices are given unique identifiers using any of two addressing modes: *memory mapped* I/O and *isolated* I/O. In *memory mapped* I/O, the I/O devices and memory locations share the same address space.
- In computer system, an interrupt is a signal generated by hardware to request the processor to give immediate service suspending the current executions.
- DMA is a feature of computer systems that allows certain hardware subsystems to access primary system memory (random-access memory) without the intervention of the CPU.

Space for learners:

4.10 ANSWERS TO CHECK YOUR PROGRESS

i, c	**ii, d**	**iii, a**	**iv, b**	**v, a**
vi, d	**vii, b**	**viii, a**	**ix, b**	**x, c**

4.11 POSSIBLE QUESTIONS

Q1. What is the role of a computer bus?

Q2. Differentiate between multiplexed and dedicated bus.

Q3. What are the various aspects of bus design?

Q4. Why is it not possible to connect an I/O device directly to a computer?

Q5. Explain the tasks performed by an I/O module.

Q6. What are the signals shared by an I/O module?

Q7. What do you mean by an interrupt in terms of a computer system?

Q8. What do you mean by DMA? What are the advantages of using DMA?

Q9. Discuss various types of I/O devices.

Q10. Differentiate between maskable and non-maskable interrupt.

Space for learners:

4.12 REFERENCES AND SUGGESTED READINGS

- William Stallings, Computer Organization and Architecture Designing for Performance, Pearson Education India.
- Carl Hamacher, ZvonkoVranesic, SafwatZaky, Computer Organization, McGraw Hill Education.
- M. Morris Mano, Computer System Architecture, Pearson Education India.

---×---

UNIT 5: BASIC I/O SYSTEM-II

Space for learners

Unit Structure:

5.1 INTRODUCTION

I/O operations are performed through a large variety of I/O devices. These devices provide a way of interchanging data between the external environment and the computer. Different I/O devices have different data transfer rates, different data formats and different word lengths. These variations make the direct interaction between I/O devices and processor (or memory) very complex. Thus, the processor or the memory does not interact with the I/O devices rather I/O modules are used to establish the interactions between I/O devices and the processor (or memory) as a mediator. For an instance, if the processor wishes to send some data to an I/O device, it sends it to the I/O module which forwards the same to the specific I/O device. The I/O operations are performed using three techniques: programmed I/O, interrupt driven I/O and direct memory access.

This unit begins with a discussion on the three mentioned I/O operation techniques. The unit also presents a discussion on various way of handling multiple interrupts.

5.2 UNIT OBJECTIVES

On completion of this unit students will be able to:

- Explain the various aspects of I/O transfer based on Programmed I/O, Interrupt Driven I/O and DMA Transfer
- Compare Programmed I/O, Interrupt Driven I/O and DMA Transfer
- Explain different ways of handling multiple interrupt requests

Space for learners

5.3 PROGRAMMED I/O

In programmed I/O, the processor exchanges data with the I/O module. The processor allows the I/O module to control the I/O operations directly. The I/O module can read the device status, send read or write command and transfer data. The processor sends a command to the I/O module and waits for the I/O module to complete the operation.

When the processor sees an instruction associated with I/O, it issues necessary commands to the concerned I/O module. The I/O module then loads the status register with appropriate values. The processor checks the status of the I/O module periodically until the module is ready for the transfer. The data transfer takes place only when the I/O module is ready. Most of the I/O devices have much slower data rates than the processor and the memory. So, the processor may need to wait for a longer amount of time for the I/O to get ready. This is the major disadvantage of programmed I/O as it reduces the throughput of the processor.

The processor issues some commands to the I/O module along with an address referring an I/O module and an I/O device. There are four types of commands: *control*, *test*, *read* and *write*.

The **control** command is used to specify the operation to be performed by the external device. For example, it may send commands like READ SECTOR, WRITE SECTOR,SCAN record ID to a magnetic disk. The commands are made according to the operations an I/O device performs.

Test commands are used to test various status signals of both the I/O module and the I/O devices. Before any I/O transfer, the processor needs to test the current status of the I/O module or the device to check whether the module or the device is powered on, ready or busy. It may also need to know if the last data transfer is successful or any error has occurred.

The **read** signals are sent to the I/O modules when the processor needs data from any I/O. The I/O module gets the data from the particular I/O device and buffers it in its internal storage (data buffers/ data registers) temporarily before sending back to the processor. The I/O module sends back the data to the processor by placing them onto the data bus on receiving request from the processor.

With **write** signal, the processor requests the I/O module to send the data available on the data bus to a specific I/O device. The I/O module obtains the data from the bus and buffers it until the corresponding I/O device is ready to accept the data.

Fig. 5.1 demonstrates the process of transferring blocks of data from memory to I/O using programmed I/O. The processor first fetches a memory word and tests the status of I/O. If the I/O module is ready it transfers the data immediately otherwise it waits. During the

Space for learners

waiting period, the processor keeps on sensing the I/O status periodically.

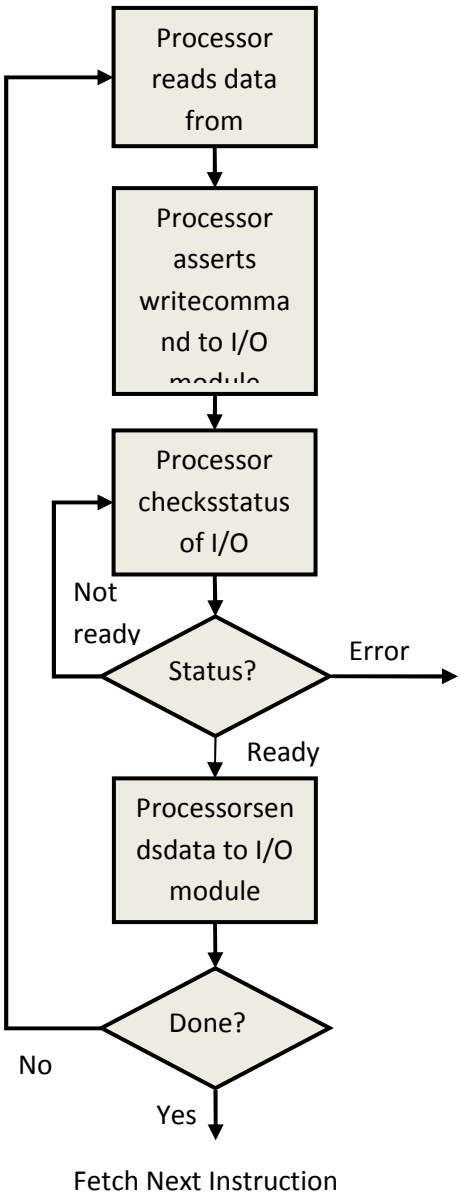


Fig. 5.1 Flowchart showing transfer of data from the processor to I/O using programmed I/O

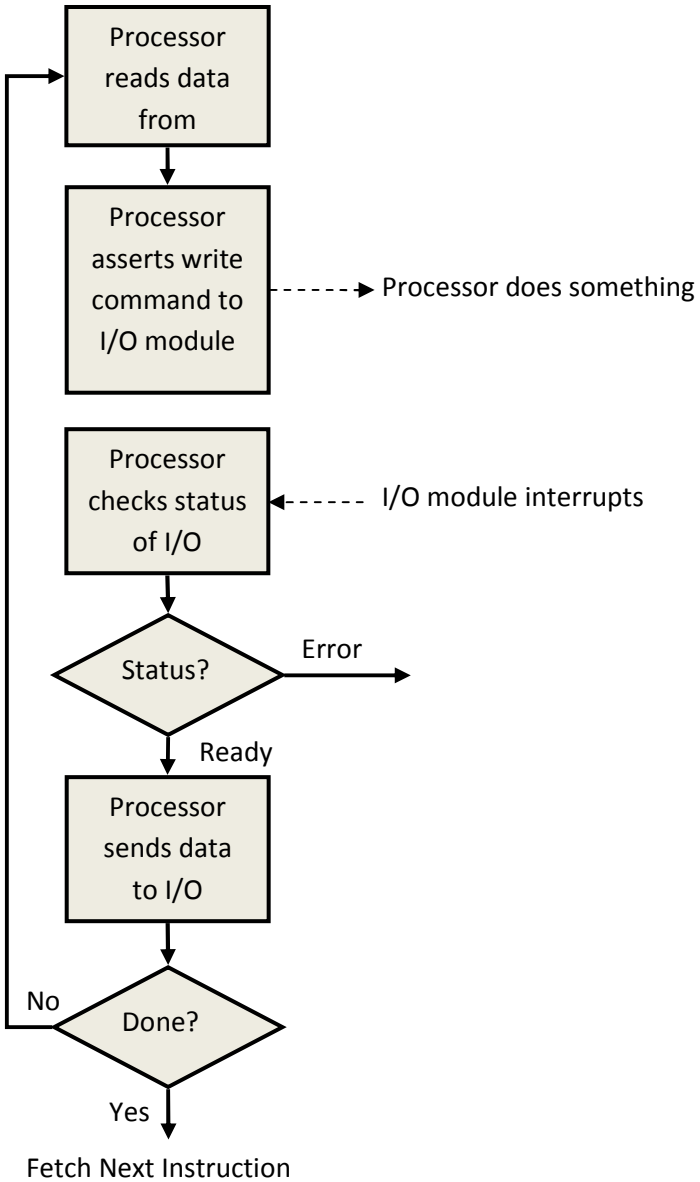
STOP TO CONSIDER

Programmed I/O is an I/O transfer technique wherein the processor continuously senses the status of I/O until the later gets ready for data transfer. This reduces the performance of the processor.

Space for learners

5.4 INTERRUPT DRIVEN I/O

The main problem with programmed I/O is that the processor has to wait for long time for the I/O module to be ready. During the waiting time, the processor must check the status of the I/O continuously. This adversely affects the performance of the overall system. Consequently an alternative solution is required to enhance the performance of the entire system. One best solution is the use of interrupt signals. Instead the processor checking repetitively the status of I/O, the I/O module can send interrupt to the processor when it is ready. Such type of I/O transfer is called as interrupt driven I/O.



Space for learners

Fig. 5.2 Flowchart showing transfer of data from the processor to I/O using interrupt driven I/O

Fig. 5.2 presents the flowchart of transfer of memory words to I/O using interrupt driven I/O. The processor first reads the data from memory and asserts the write signal to the I/O module to which the concerned I/O device is connected. It specifies the I/O device by placing its address on the address bus. The processor does not wait for the I/O to get ready and continues its execution. In interrupt driven I/O, the processor issues an command to an I/O module and then gets busy in doing other processing. The I/O module will send an interrupt request to the processor when it is available to perform the data transfer. Every interrupt has a specific program or routine called as interrupt service routine (ISR) to process the interrupt request. On receiving the interrupt request, the processor finishes its current instruction and then goes on to give the service to the interrupt request by executing the corresponding ISR. The processor stops the current execution temporarily while executing an ISR. It goes back to its previous program immediately after finishing the ISR.

The I/O module identifies the I/O device based on the address available on the address bus. It checks the status of the corresponding I/O device if it is ready. The I/O device sets the status as ready to inform the I/O module when it is ready to send any data. On receiving this information, the I/O module interrupts the processor. The processor then transfers the data and checks if any data is remaining to transfer. If not the processor continues with the data transfer as shown in the diagram.

Space for learners

Space for learners

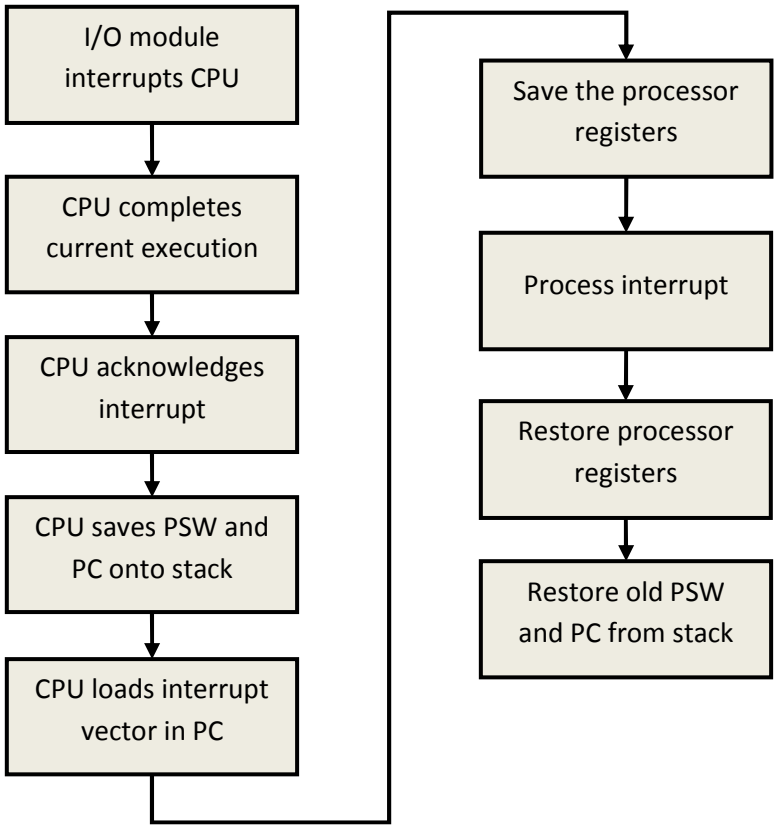


Fig. 5.3 Block Diagram of Interrupt Processing

Fig. 5.3 presents a block diagram of the sequence of events occurred during the processing of a typical interrupt. The following sequence of events occurs during an I/O transfer.

1. The I/O module sends an interrupt signal to the processor.
2. The processor completes current instruction before answering the interrupt.
3. The processor tests for the interrupt at the end of every instruction cycle. When it sees any interrupt, it sends acknowledgement to the I/O module. The I/O module then disables the interrupt signal.

4. The processor prepares to start the execution of the ISR. It saves the program return address (current value of the program counter) and the ALU flags or program status word (PSW) onto the stack.
5. The processor loads the routine address or the interrupt vector onto the program counter (PC).
6. The processor then saves the current status of the executing program, particularly the contents of the ALU registers onto stack. This is very essential as the ISR may need to use these registers.
7. The processor starts processing the interrupt by executing the ISR. At this stage the processor begins its next instruction cycle.
8. After completion of the execution of the ISR, the processor restores ALU registers.
9. Finally it restores PSW and the old value of PC stored from the stack.

STOP TO CONSIDER

Unlike in programmed I/O, in interrupt driven I/O the processor does not continuously check the status of the I/O device. After initiating the I/O transfer the processor gets involved in some important tasks without waiting for the I/O. The I/O module sends an interrupt signal whenever the I/O device is ready for the data transfers.

Design Issues

When it comes to interrupt driven I/O, there are two design challenges to consider. First, how will the processor identify the interrupting device if multiple devices are connected? Second, which interrupt to process if multiple interrupts occur at some time?

To address the first issues, i.e., device identification four techniques are used in common:

Space for learners

- Multiple interrupt lines
- Software Poll
- Hardware Poll (Daisy Chain)
- Bus Arbitration

The simplestand straightforward solution to handle multiple interrupt is the use of **multiple interrupt lines** for multiple I/O devices.However, it is not a practical solution to have too many lines for interrupts. Typically, interrupt lines not assignedtothe I/O devices; instead they are assigned to the I/O modules. This method helps the processor to identify easily the interrupted module. But an I/O module can connect more than one device, so to identify the specific device (the one which triggered the interrupt) from many one of the remaining three methods can be used.

Instead of using multiple interrupt lines **Software polling** can be used alternatively to handle multiple interrupts. In this, a common ISR is executed when the processor sees an interrupt. The job of this ISR is to detect the interrupted module by polling each module. The polling can be done by using a dedicated command line (TESTI/O). The processor sets the TESTI/Oand places the I/O address in the address bus. An I/O module responds to this signal positively if the interrupt is raised by it. Alternatively, each I/O module can possess a status register which will be set when it raises the interrupt signal. The processor will check the status register of each I/O module and will determine the I/O module that caused the interrupt based the status information. After identification of the interrupted module, the processor executes the ISR of the interrupted device. The advantage of software polling is that a single interrupt line is sufficient for implementing interrupt driven I/O. However it is very time consuming.

Space for learners

Hardware polling is a very efficient alternative to software polling for handling multiple interrupts. A technique called **Daisy Chain** can be used to implement this. In this approach, a common interrupt request line is shared among all I/O modules. The I/O modules are connected in a serial order. The interrupt acknowledgement line is shared with the I/O modules through a daisy chain as shown in Fig. 5.4. The processor sets the interrupt acknowledgement signal when it sees any interrupt request. This signal is received by the I/O module which is directly connected with the interrupt acknowledgement line. If the interrupt request is raised by that particular I/O module then it will respond by placing a vector in the data lines; otherwise the module will forward the acknowledgement signal to the next module in the sequence. The next module will react to the signal exactly in the similar manner. Thus the interrupt acknowledgment signal will be propagated through the I/O modules until any response is received from the interrupted module. The vector is usually an address that refers an I/O module. The processor calls the device specific ISR based on the value of the vector.

Another alternative is **bus arbitration**. In this approach, only one module can send interrupt request. To do so, the I/O module has to obtain the control of the bus first. The processor responds to the interrupt by sending an interrupt acknowledgement signal. The I/O module responds to this signal by placing its interrupt vector onto the data bus.

To solve the second issue, different levels of priorities can be assigned to different modules. When more than one module interrupts, the modules are given services according to their priority levels. The module with the highest priority is given the service first. The above mentioned techniques can also be used to handle this priority interrupt. When there are multiple interrupt lines, the

Space for learners

processor simply chooses the one with the highest priority. In software polling, the module polling order is designed according to their priority. In case of hardware polling, the modules in the daisy chain are arranged according to their priority with the highest priority first. In case of bus arbitration, the bus arbiter determines which module should get the control of the bus depending on their priority.

Space for learners

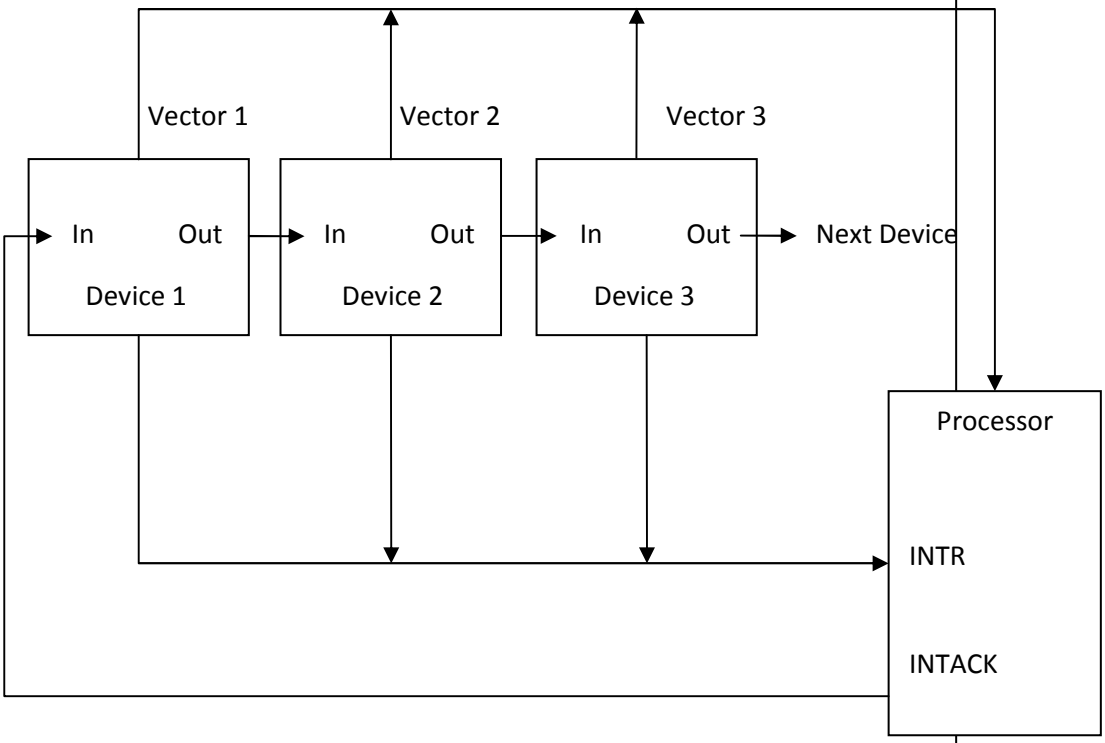


Fig. 5.4 Hardware Polling using Daisy Chaining

5.5 DIRECT MEMORY ACCESS

Both programmed I/O and interrupt driven I/O require the active intervention of the processor to perform the data transfer between memory and I/O. When there is a need to transfer a large amount of data, the processor is often tied up with the I/O transfer. Also, the data transfer speed is affected by lot of testing and condition checking. To avoid these issues a more efficient technique called

direct memory access (DMA)can be used while transferring a large amount of data.

It is a data transfer technique in which transfer of data from memory to I/O takes place without the active involvement of the processor. To accomplish this, an additional module called a DMA controller is required. A DMA controller shares the system bus along with processor, memory and I/O. Its role is to control the entire data transfer. For this, it has to acquire the control of the system bus. The structure of a typical DMA controller is shown in Fig. 5.5.

When the processor needs to perform DMA transfer, it issues a DMA request to the DMA controller and sends the following information to the DMA controller:

- Depending on the operation type, the processor asserts read or write signal to the DMA controller by raising the corresponding control line between the processor and the DMA controller.
- The address of the target I/O device.
- The address in the memory from/to where data transfer to begin through the data lines. The DMA controller saves this address in its address register.
- The number of words to be transferred. This value is then stored in the data count register.

After initiating the transfer, the processor relinquishes the buses and continues with other works while the DMA controller gains the control of the buses and takes over the remaining transfer. The DMA controller transfers the entire blocks of data one by one. Once the DMA transfer is complete, the controller sends an interrupt to the processor.

Space for learners

There are basically two types of DMA transfers: *burst mode* and *cycle stealing*. Burst mode transfers a whole block of data in a single contiguous sequence. When the processor grants the DMA controller the access to the system bus, it transfers entire bytes of data in the data block before returning control of the system buses to the processor; however this leaves the processor inactive for a long time.

In systems where the processor should not be disabled for the length of time required for burst transfer modes, the cycle stealing mode is used. The DMA controller gains control the system bus in cycle stealing mode in the same way as it does in burst mode, by using the BR (Bus Request) and BG (Bus Grant) signals, which control the interface between the processor and the DMA controller. In cycle stealing mode, however the control of the system bus is delegated to the processor via BG after one byte of data transfer. After a cycle, the DMA controller again obtains the buses using BR and BG signal for the next transfer. This switching of the buses between the processor and the DMA controller continues until entire blocks of data are transferred.

CHECK YOUR PROGRESS:

i. _____ is a way of accessing I/O devices by continuously checking the status flags.
- **a.** Programmed I/O
- **b.** Interrupt driven I/O
- **c.** DMA
- **d.** None of the above

ii. The address of an ISR is termed as
- **a.** interrupt location
- **b.** interrupt vector
- **c.** interrupt address
- **d.** none of the above

iii. _____ is used to store the return address of ISR.

Space for learners

a. Registers
b. Cache
c. System heap
d. Stack

iv. In case of interrupt driven I/O, I/O module sends _____ signal to the processor when an I/O device is ready for data transfer.
a. interrupt request
b. interrupt acknowledgement
c. read/write
d. none of the above

v. After receiving an interrupt, the signal delivered from the processor to the device is
a. interrupt request
b. interrupt acknowledgement
c. read/write
d. none of the above

vi. _____ is a technique to handle multiple interrupt.
a. Software polling
b. Daisy Chaining
c. Multiple interrupt line
d. all of the above

vii. DMA transfer is initiated by the
a. DMA controller
b. processor
c. I/O device
d. none of the above

viii. _____ is responsible for controlling the transfer of data during DMA.
a. DMA controller
b. processor
c. I/O device
d. none of the above

ix. During DMA transfer, _____ becomes the master of the system bus.
a. DMA controller
b. processor
c. I/O device
d. none of the above

x. The method by which the DMA controller steals the processor's access cycles is known as

Space for learners

a. bust mode
b. cycle stealing
c. memory stealing
d. bus stealing

Space for learners

5.6 SUMMING UP

- I/O devices provide a way of interchanging data between the external environment and the computer. Different I/O devices have different data transfer rates, different data formats and different word lengths.
- Due to the differences present, the processor or the memory does not interact with the I/O devices rather I/O modules are used to establish the interactions between I/O devices and the processor (or memory) actsas a mediator.
- The I/O operations are performed using three techniques: programmed I/O, interrupt driven I/O and direct memory access.

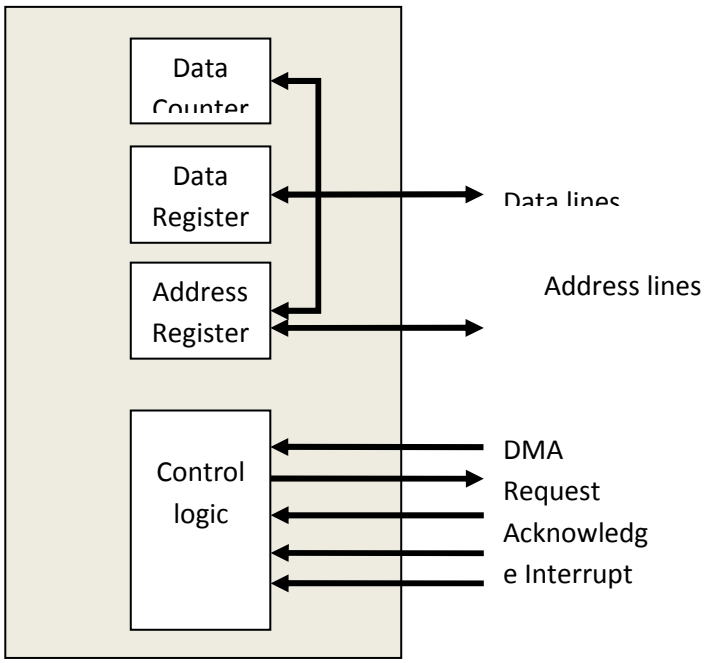


Fig. 5.5 Structure of a DMA controller

- In programmed I/O, the processor exchanges data with the I/O module. The processor allows the I/O module to control the I/O operations directly. The processor senses the status of I/O continuously until the device is ready. It transfers the data only when the I/O is ready.
- In interrupt driven I/O, instead the processor checking repetitively the status of I/O, the I/O module sends interrupts to the processor when it is ready. The processor continues with meaningful tasks after initiating the I/O transfer without waiting for the I/O to get ready.
- DMA is a data transfer technique in which transfer of data from memory to I/O takes place without the active involvement of the processor. To accomplish this, an additional module called a DMA controller is required. A DMA controller shares the system bus along with processor, memory and I/O.

Space for learners

STOP TO CONSIDER

DMA is a technique by virtue of which data transfers take place without involving the processor. To control the entire transfer a special module is attached to the system bus called as DMA controller. During a DMA transfer, the processor remains free to perform other processing.

5.7 ANSWERS TO CHECK YOUR PROGRESS

i, a	**ii, b**	**iii, c**	**iv, a**	**v, b**
vi, d	**vii, b**	**viii, a**	**ix, a**	**x, b**

5.8 POSSIBLE QUESTIONS

Q1. What is meant by interrupt?

Q2. What is the difference between Programmed I/O and Interrupt driven I/O?

Q3. How does a computer handle multiple interrupt?
Q4. What is meant by interrupt priority? What are techniques available to handle priority interrupt?
Q5. What is polling?
Q6. What is the difference between Software and Hardware Polling?
Q7. What is the advantage of DMA transfer?
Q8. What are the different techniques used for DMA transfer?
Q9. Differentiate between Cycle Stealing and Burst Mode.
Q10. What are the major components of a DMA controller?

Space for learners

5.9 REFERENCES AND SUGGESTED READINGS

- William Stallings, Computer Organization and Architecture Designing for Performance, Pearson Education India.
- Carl Hamacher, Zvonko Vranesic, Safwat Zaky, Computer Organization, McGraw Hill Education.
- M. Morris Mano, Computer System Architecture, Pearson Education India.

---×---

BLOCK III:

ADVANCED CONCEPTS OF PARALLEL ARCHITECTURES

UNIT 1: BASIC PARALLEL ARCHITECTURE AND INSTRUCTION PIPELINE

Unit Structure:

Space for learners:

1.1 INTRODUCTION

The chapter reviews architectural evolution of computers starting from uniprocessor systems to multiprocessor system through Flynn's classification of computer architecture. The chapter also compares various processors types like scalar processor, superscalar processor, pipelined processor and vector processor. The basic concept of pipelining and the working of instruction pipeline is

discussed in detail. Finally, the chapter ends with discussion on the types of dependencies that exists in pipelined processors which if not taken care of will affect the overall performance of the system. The three dependencies discussed are resource conflict, branch hazard and data hazard.

Space for learners:

1.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- Classify computer architecture based on the notion of instruction and data stream.
- Compare different types of processors and their characteristics.
- Explain the basic concept of pipelining and the types of pipelining.
- Explainhow pipelining improvesthe performance of a system.
- Explain how multiple instructions are executed in an overlapped fashion in instruction pipelining.
- Identify the types of dependencies in pipelined processors and ways to resolve the dependencies.

1.3 FLYNN'S CLASSIFICATION OF COMPUTER ARCHITECTURE

With the increase in the number of processing units and segmentation of a job/program into multiple segments wherein each of the segment is placed on a different processing unit for concurrent execution has resulted in classification of systems. Flynn's classification or Flynn's taxonomy of computer architectures is proposed by Michael J. Flynn in the year 1972. The classification is based on the notion of instruction stream and data stream. A stream refers to sequence of instruction or data operated by the computer system. The flow of instruction from memory to

processor is called instruction stream and the flow of data between processor and memory is called data stream.

Space for learners:

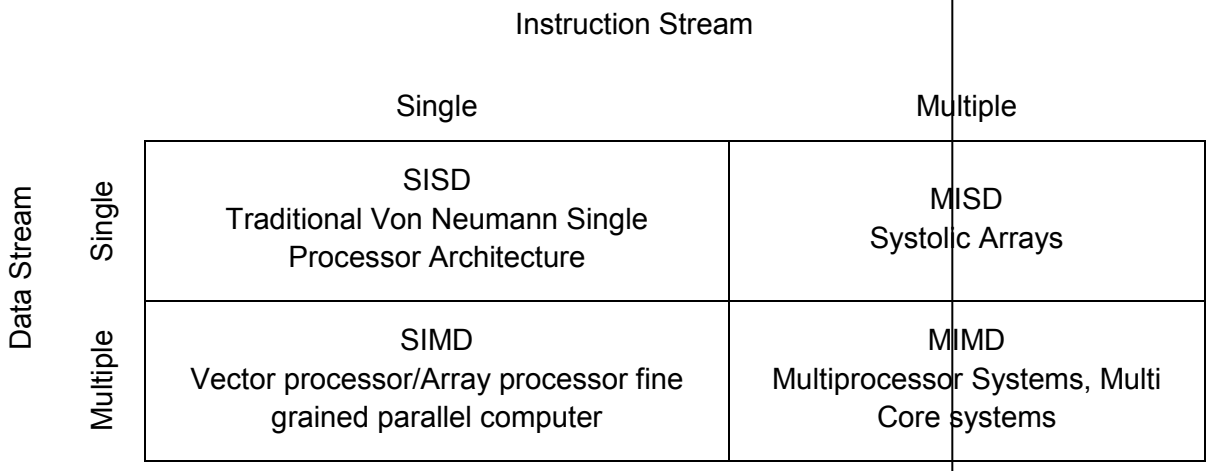


Figure 1.1: Flynn's classification of Computer Architecture

The Figure 1.1 shows four categories in which Flynn has classified computer architecture based on instruction and data stream. A conventional uniprocessor system is called SISD (Single instruction stream Single data stream) computers. A vector /array of processors is called SIMD (Single instruction stream Multiple data stream) computers. In MISD (Multiple instruction stream Single data stream) computers, different instructions work on the same data. Finally, in MIMD (Multiple instruction stream Multiple data stream) computers, multiple processors each working on different data increases the overall performance of the system.

1.3.1 SISD

A Single instruction stream single data stream (SISD) system as shown in Figure 1.2 is a uniprocessor system. Such systems work on

a single instruction stream and single data stream at a time. Instructions in SISD systems are processed in a sequential order and are therefore known as sequential computers. Conventional systems were of SISD architecture. Processing in SISD systems involve storing both instructions and data in primary memory. Processing speed of such systems depends on internal data transfer rate. However, performance of such systems can be improved with the help of multiple functional units or pipelining. Example of SISD systems includes CDC 6600, IBM PC.

Space for learners:

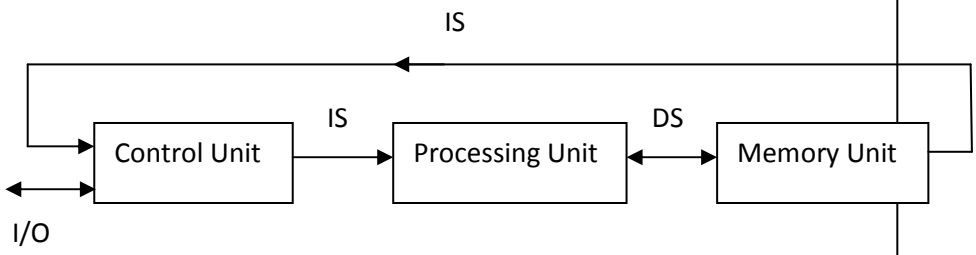


Figure 1.2: SISD Architecture

1.3.2 SIMD

A single instruction stream multiple data stream (SIMD) system as shown in Figure 1.3 are multiprocessor systems capable of working on different data streams through a single instruction stream. SIMD systems are used in scientific computation involving vector operations. They are also known as vector processors or array processors. There are *n* number of processing units each having its own memory and stream of data. All the *n* processing units receive the same instruction from the control unit. Example of SISD systems includes CRAY vector computers.

Space for learners:

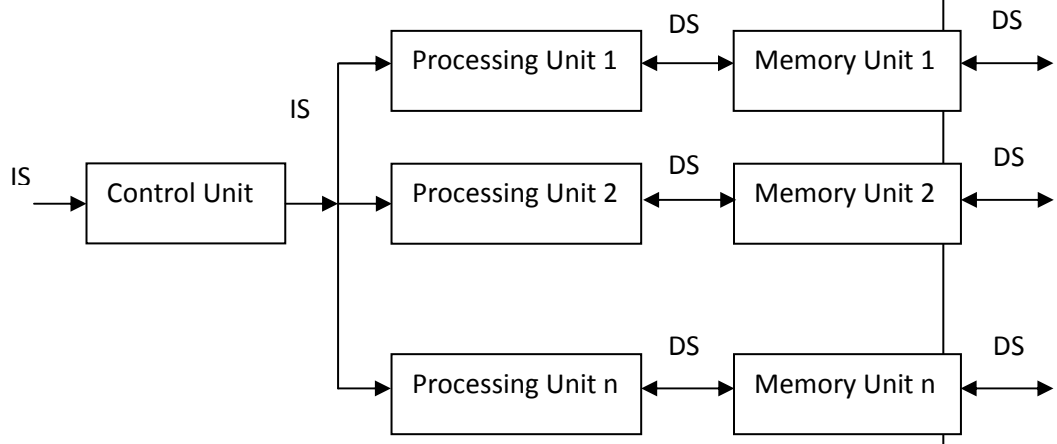


Figure 1.3: SIMD Architecture

1.3.3 MISD

A multiple instruction stream single data stream (MISD) system as shown in Figure 1.4 is a multiprocessor system which executes different instructions on different processing unit but data set is same for all the instructions. MISD systems are not practical in the majority of the application and therefore are not available commercially. One such example of MISD system is systolic array.

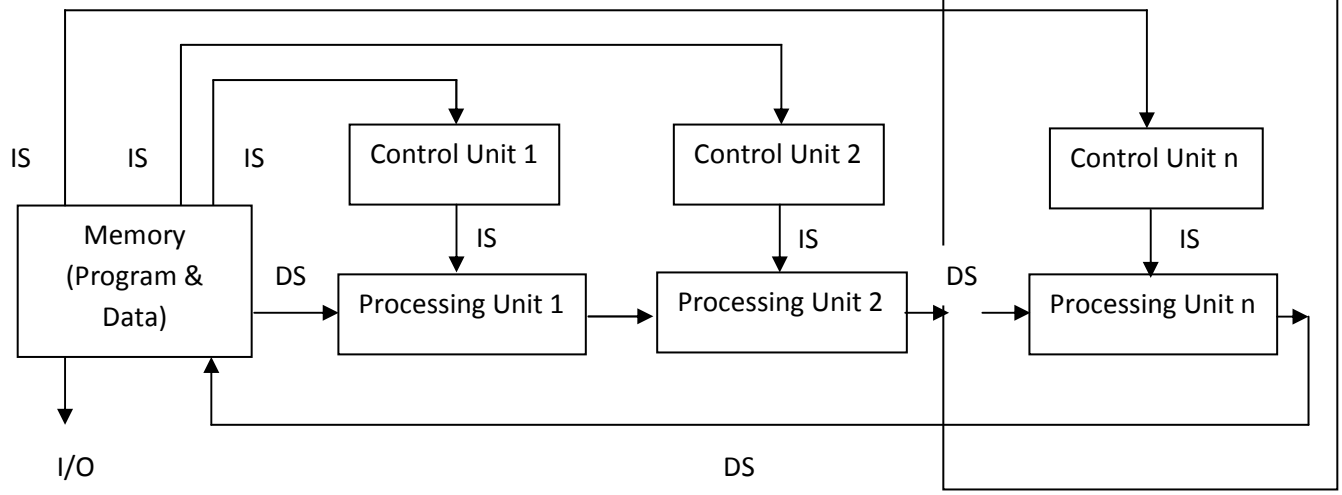


Figure 1.4: MISD Architecture

1.3.4 MIMD

Space for learners:

A multiple instruction stream multiple data stream (MIMD) system is a multiprocessor system capable of executing different sets of instructions each working on a different set of data simultaneously. Figure 1.5 shows MIMD architecture. Multiple SIMD systems connected together can be viewed as a MIMD system. MIMD systems can be classified into shared-memory MIMD and distributed-memory MIMD based on processing unit-main memory connections.

The shared memory MIMD system also known as tightly coupled multiprocessor system, are the one where all the processing units are connected to a single shared memory. Any form of communication between processing unit takes place with the help of shared memory. Changes done to data in shared memory by one processing unit is visible to all other processing units. In distributed memory MIMD systems or loosely coupled multiprocessor systems, all processing units have their own local memory. The communication between processing units takes place through the interconnection.

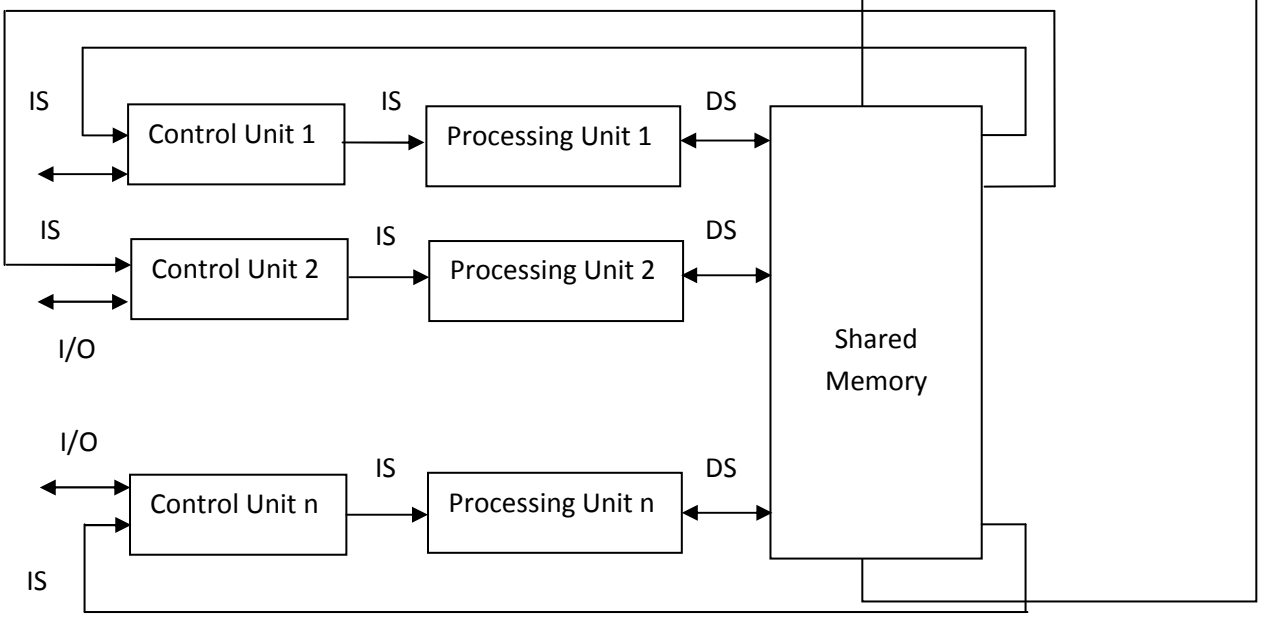


Figure 1.5: MIMD Architecture

1.4 TYPE OF PROCESSORS

Space for learners:

There are several types of processors. A brief description of each of these are provided below.

1.4.1 Scalar processor

A scalar processor also known as Single Instruction Stream Single Data Stream (SISD) can process a single data item at a time. Scalar processor can process either integer or floating point operands. The simplest scalar processor uses floating point unit to process integer operands. However scalar processor may have separate integer and floating point units for handling integer and floating point operands. AMD 2900, Motorola 68040, Intel 386, Intel 486, M88100 are some examples of scalar processor.

1.4.2 Superscalar Processor

Superscalar processors are found in parallel computing architecture to improve the performance of the system by executing multiple instructions in parallel. A superscalar processor manages multiple instruction pipelines to execute multiple instructions concurrently in a clock cycle. The performance of superscalar processor is highly dependent on the instruction dependency quotient. If the instructions to be executed are independent, then high performance is achieved.

Figure 1.6 shows a superscalar pipeline of degree 2 (i.e. Two instructions can be executed in parallel). There are five stages in the pipeline namely fetch, decode, operand fetch, execute and write. It can be observed from the Figure 1.6 that the superscalar pipeline has two units each of fetch, decode, operand fetch, execute and write, therefore two instructions can be simultaneously executed. In the first clock cycle instruction (1, 2) are fetched, in the second clock cycle next two instructions i.e. (3, 4) are fetched and the process

continues. Pentium, DEC Alpha, PowerPC are some of the example of superscalar processor computers.

Space for learners:

	1	2	3	4	5	6	7	8
Instruction 1	F	D	OF	E	W			
Instruction 2	F	D	OF	E	W			
Instruction 3		F	D	OF	E	W		
Instruction 4		F	D	OF	E	W		
Instruction 5			F	D	OF	E	W	
Instruction 6			F	D	OF	E	W	
Instruction 7				F	D	OF	E	W
Instruction 8				F	D	OF	E	W

Figure 1.6: A superscalar pipeline of degree two.

1.4.3 Pipelined Processor

There are four types of pipelined processors namely Scalar Pipeline, Superscalar Pipeline, Super pipeline, Super pipeline Superscalar as shown in Figure 1.7 depending upon the following criterions. It is assumed that all the pipelined processors are of k stages.

- Machine Pipeline Cycle (MPC): Time taken by each stage to process an instruction.
- Instruction Issue Rate (ISR): Number of instruction that can be issued simultaneously.
- Instruction Issue Latency (ISL): Time interval between issue of two instructions.
- Instruction Level Parallelism (ILP): Number of instructions that can be executed simultaneously in the pipeline.

Machine Type	Scalar Pipeline	Superscalar Pipeline	Superpipeline	Superpipeline Superscalar
MPC	1	1	1/n	1/n
ISR	1	m	1	m
ISL	1	1	1/n	1/n
ILP	1	m	n	mn

Figure 1.7: Parameters of Pipelined Processor.

Space for learners:

1.4.4 Vector processor

Vector processors are found mainly in supercomputers combining pipelining and interleaved memory unit. It is used mainly in scientific and multimedia applications involving processing of huge volume of data. It is capable of processing entire vector in single instruction. The operands in the instructions are vectors instead of a single element. One of the advantages of vector processors is less number of fetch and decode instructions.

Vector processor uses many optimization schemes to improve performance of the system such as use of memory banks to reduce load/store latency, use of strip mining technique to adjust the size mismatch between vector operands and vector registers, vector chaining to resolve data dependency between vector instructions etc.

Advantages of Vector processing:

- Programs are smaller in size as the number of instructions is quite less.
- As each data in registers is actually used by the vector processor therefore wastage in memory access is significantly less compared to cache memory.
- Requirement of power is limited to only functional unit and register buses during vector operation.

Based on how operands are fetched in vector processors is categorized into two types:

- Vector register Processor
- Memory-Memory Vector Processor

Vector-Register Processor

It requires that all the operations performed in the vector processor use the source operands and destination operands as vector registers. However, there is a small disadvantage initially that is vector data in memory must be divided into fixed length segments so that can be placed in vector register. But once the pipelining starts this disadvantage is nullified.

Memory-Memory Vector Processor

Such processors allow source operand and destination operand to be routed directly to the arithmetic logic unit (ALU). Once the processing is completed in the ALU, the result is routed back to memory. However due to memory latency the time between initializing the first instruction and the getting the first output from the pipeline is quite large.

Space for learners:

1.5 PIPELINING

A pipeline is similar to an assembly line in a production factory. A product has to go through multiple stages in the assembly line before the final product is manufactured. At a time, all the stages work simultaneously but on different phases of the product. This process is referred to as pipelining. Pipelining is also referred to as execution of multiple jobs/instructions parallelly in an overlapped fashion.

Let us look at a real life example that works on the concept of pipelining. Consider a packaged drinking water plant having the following 3 stages and each stage takes 1 minute to complete its operation.

- Filling (F)--- Stage 1
- Sealing (S) --- Stage 2
- Labeling (L) --- Stage 3

In a non-pipelined operation if we have to do the packaging of 4 bottles, it will take 12 min to complete the operation as shown in Figure 1.8. Each bottle spending 1 min in each of the filling, sealing and labeling stage respectively.

The bottle reaches stage-1 where it is filled and after 1 minute it moves to the stage-2 where it is sealed. At this point stage-1 is in idle state. Now after staying in stage-2 for 1 minute the bottle is moved to stage-3 where it is labeled. At this point stage-1 and stage-2 is in idle state as shown in the figure 1.8. This process of packaging does not utilize the time as the stages remain in idle state during the operation. To overcome the issue and to utilize the stages to its maximum limit, pipelining is used.

	Time in minute→											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Bottle 1	F	S	L									
Bottle 2				F	S	L						
Bottle 3							F	S	L			
Bottle 4										F	S	L

Figure 1.8: Non Pipelined Operation

Space for learners:

Now, in a pipelined operation if we have to do the packaging of 4 bottles, it will take 6 min to complete the operation as shown in Figure 1.9. Compared to 12 minutes taken in non-pipelined operation.

As it can be observed in Figure 1.9, when the first bottle is in stage-2 (Sealing), the second bottle is placed in stage-1(Filling). Similarly, when the first bottle is in stage-3(Labeling), second bottle is placed in stage-2(Sealing) and third bottle is placed in stage-1(Filling). Thus, none of the stages are idle at any moment. All the stages are working on a different bottle at a time. This process of working in an overlapped fashion to utilize the stages of a pipeline to its fullest is called pipelining.

	Time in minute→											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Bottle 1	F	S	L									
Bottle 2		F	S	L								
Bottle 3			F	S	L							
Bottle 4				F	S	L						

Figure 1.9: Pipelined Operation

1.6 INSTRUCTION PIPELINING

In a computer system the technique of executing multiple instructions in an overlapped fashion is known pipelining. A pipeline consists of many stages and these stages are connected to one another in a pipe like structure. An instruction enters one end of the pipeline, goes through several stages before exiting from another end. Pipelining improves the overall throughput of the system.

Space for learners:

In a pipeline system, each stage uses register to hold the output of that stage. Output of one stage is applied as input to the next stage.

Space for learners:

Figure 1.10: Five stage Instruction Pipeline

Figure 1.10 shows an example of five stage instruction pipeline consisting of fetch, decode, operand fetch, execute and write stages. Here streams of instructions are executed in overlapped fashion thereby increasing the throughput of the computer system.

Figure 1.11 shows the timing diagram of an instruction pipeline. While the instruction pipeline reads one instruction from the memory, previous instructions is executed in other stage of the pipeline. Thus, multiple instructions are executed simultaneously. From the Figure 1.11, it can be observed that while the first instruction started at time period one, the second instruction started at time period two and so on. Up to time period four, not all stages were working simultaneously but from time period five onwards all the five stages are working simultaneously. Therefore, from instruction number five onwards each stage is working on a different instruction as:

Instruction 1: Write
Instruction 2: Execute
Instruction 3: Operand Fetch
Instruction 4: Decode
Instruction 5: Fetch

Time →

	1	2	3	4	5	6	7	8	9	10	11
Instruction 1	F	D	OF	E	W						
Instruction 2		F	D	OF	E	W					

Instruction 3			F	D	OF	E	W				
Instruction 4				F	D	OF	E	W			
Instruction 5					F	D	OF	E	W		
Instruction 6						F	D	OF	E	W	
Instruction 7							F	D	OF	E	W

Figure 1.11: Timing diagram for Instruction Pipeline Operation

If there are ***k*** number of stages and ***n*** number of instructions, then total time ***T*** taken to execute n instructions can be given as T = k + (n -1).

Space for learners:

1.7 DEPENDENCY IN PIPELINED PROCESSOR

A pipelined processor may be affected due to the following dependencies, which may also result in the stalls in the pipeline. A stall is a pipeline cycle with no operation or no new input.

- Structural Dependency or Resource Conflict
- Control Dependency or Branch Difficulty
- Data Dependency or Data Hazard

1.7.1 Structural Dependency or Resource Conflict

Structural dependency is the result of resource conflict in the pipeline. When several instructions in the same cycle try to access the same resource, a resource conflict arises.A resource can be a register, memory, or ALU.

Time →

	1	2	3	4	5	6	7	8	9	10	11
Instruction 1	F	D	OF	W							
Instruction 2		F	D	OF	W						
Instruction 3			F	D	OF	W					
Instruction 4				F	D	OF	W				

Figure 1.12: Timing diagram of a 4-Stage Instruction Pipeline

In cycle 4 of the Figure 1.12, instruction1 is trying to do the write operation on memory and instruction 4is trying to fetch from memory. As both the instructions are trying to access same resource i.e. memory at the same time, it introduces a resource conflict between the two instructions. Such situation can be avoided by keeping the instruction 2 in wait state until the required resource becomes available.

Space for learners:

1.7.2 Control Dependency or Branch Hazard

A pipeline achieves its maximum utilization if all the stages of the pipeline take equal amount of time to process and there is no branch instruction in the program. However, if the program contains branch instruction, the pipeline suffers from branch penalty.

The timing diagram of a4 stage instruction pipeline containing branch instruction is shown in Figure 1.13 where instruction 1,2,3 and 4 are non-branch instruction and instruction 5 is a branch instruction.

Time → |←Branch Penalty→|

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	F	D	OF	E										
Instruction 2		F	D	OF	E									
Instruction 3			F	D	OF	E								
Instruction 4				F	D	OF	E							
Instruction 5 (branch to instruction 25)					F	D	OF	E						

Instruction 6						F	D	OF						
Instruction 7							F	D						
Instruction 25								F	D	OF	E			
Instruction 26									F	D	OF	E		

Figure 1.13: Timing diagram for Instruction Pipeline Operation

The pipeline executes instruction 1, 2, 3 and 4 sequentially, followed by instruction 5 (branch instruction). By the time instruction 5 is decoded by the pipeline decode stage, instruction 6 and instruction 7 enters the pipeline. At this point, the pipeline realized that it should have placed instruction 25 after the branch instruction 5 instead of instruction 6.

So the pipeline discards the instructions 6 and 7, that is the pipeline cycle at time period 6 and 7 are wasted. This is known as branch penalty as the processor could not anticipate the branch. So instruction 25is assumed to be the instruction to be executed on the branch and starts at time period eight.

1.7.3 Data Dependency or Data Hazard

In a pipeline, there can be a situation where output of first instruction acts as an input to the second instruction. Such situation exhibits data dependency where the second instruction must wait in the pipeline for the first instruction to complete its execution. Otherwise the second instruction may be working on an invalid data. This dependency between the instructions is known as data dependency or data hazard. So the order of execution of the instructions does matter.

Space for learners:

There are mainly three types of data hazards:

- Read after Write (RAW) Hazard or Flow dependency
- Write after Read(WAR)Hazard or Anti-Data dependency
- Write after Write (WAW) Hazard or Output dependency

Read after Write (RAW) Hazard:
Instruction 1: R3 ← R4 + R5

Instruction 2: R6 ← R3 + R4

Here, the instruction 2 is reading a value in register R3 that is being produced by instruction 1. So instruction 2 should execute after instruction 1 completes its execution.

Write after Read(WAR) Hazard:
Instruction 1: R3← R4 + R5

Instruction 2: R4← R6 + R7

Here the instruction 2 is writing a value in register R4 that is being read before by instruction 1. So instruction 2 should execute after instruction 1 completes its execution.

Write after Write (WAW) Hazard:

Instruction 1: R1← R2 + R3

Instruction 2: R1← R4 + R5

Here the instruction 2 is overwriting the value in register R1 that is being produced by instruction 1. So instruction 2 should execute after instruction 1 completes its execution.

Space for learners:

1.7.4 Pipeline bubbles

A bubble or a pipeline bubble represents a stage in the pipeline that cannot perform any useful operation due to the lack of data from previous stage of the pipeline. It is a method to prevent structural, data and branch hazards. Pipeline control logic analyzes if a hazard

could arise while instructions are fetched. If this is the case, no operations (NOPs) are added to the pipeline by the control logic. As a result, before the next instruction runs, the previous one will have had enough time to complete and avert the hazard.

Space for learners:

CHECK YOUR PROGRESS:

i. CRAY systems are an example of_____.
   a) SISD
   b) SIMD
   c) MISD
   d) MIMD

ii. Pentium, DEC Alpha, PowerPC are some of the example of _____computers.
   a) superscalar processor
   b) Super pipeline
   c) Scalar
   d) Superscalar Super pipeline

iii. To _____data in between the pipeline stages, registers are used.
   a) Write
   b) Process
   c) Read
   d) Hold

iv. Motorola 68040 is an example of _____
   a) Scalar processor
   b) Superscalar processor
   c) Super pipeline processor
   d) Pipelined processor

v. A superscalar pipeline (5 stages) of degree 3 will need _____ cycles to complete 9 instructions.
   a) 6
   b) 7
   c) 8
   d) 9

vi. Instruction Issue Latency (ISL) in a pipelined processor means___
   a) Time interval between issuing of first and last instruction.
   b) Time interval between completion of first and second instruction.
   c) Time taken to complete execution of first instruction.
   d) Time interval between issuing of two instructions.

vii. Instruction Level Parallelism in a pipelined processor means_____
   a) Number of instructions in the pipeline.
   b) Number of instructions that can be completed simultaneously in the pipeline.
   c) Number of instructions that can be executed simultaneously in the pipeline.
   d) None of the above

viii. Vector processors or Array processors are also known as_____ systems
   a) SISD
   b) MISD
   c) SIMD
   d) MIMD

ix. The time period when the pipeline unit remains idle is called as _____
   a) Hazards
   b) Bubbles
   c) Stalls
   d) Both b) and c)

x. In pipelining, memory access speedup is achieved through _____
   a) Cache
   b) Buffers
   c) Memory Registers
   d) Special Registers

xi. In a pipeline branch instructions are handled by _____

Space for learners:

a) Pipeline flush operation
b) Pipeline Freeze operation
c) Pipeline Depth operation
d) Both a) and b)

xii. If second instruction tries to do a write operation before the first instruction can write on the same data, it is called as _____ dependency.
a) Data
b) Anti
c) Flow
d) Output

xiii. If second instruction tries to do a read operation after the first instruction does a write on the same data, it is called as _____ hazard.
a) RAW
b) WAR
c) Data
d) Control

xiv. Time taken by a 7 stage instruction pipeline to complete execution of 10 instructions is_____.
a) 70
b) 32
c) 16
d) 17

xv. Time taken by a 3 stage superscalar pipeline of degree 2 to execute 10 instructions is _____.
a) 10
b) 9
c) 8
d) 7

Space for learners:

1.8 SUMMING UP

- Flynn has classified computer architecture based on instruction and data stream and are SISD, SIMD, MISD, and MIMD.

- SISD systems are processed in a sequential order and are therefore known as sequential computers.
- SIMD systems are multiprocessor systems capable of working on different data streams through a single instruction stream.
- MISD system is a multiprocessor system which executes different instructions on different processing unit but data set is same for all the instructions.
- MIMD system is a multiprocessor system capable of executing different sets of instructions each working on a different set of data simultaneously.
- Superscalar processors are found in parallel computing architecture to improve the performance of the system by executing multiple instructions concurrently in a clock cycle.
- Pipelined processors namely Scalar Pipeline, Superscalar Pipeline, Super pipeline, Super pipeline Superscalar.
- Vector processors are used mainly in scientific and multimedia applications involving processing of huge volume of data. It is capable of processing entire vector in single instruction.
- Pipelining is also referred to as execution of multiple instructions parallelly in an overlapped fashion.
- Structural dependency is the result of resource conflict in the pipeline. When several instructions in the same cycle try to access the same resource, a resource conflict arises.
- If the program contains branch instruction, the pipeline suffers from branch penalty.
- Dependency between the instructions is known as data dependency or data hazard. Order of execution of the instructions does matter.

Space for learners:

- There are mainly three types of data hazards Read after Write (RAW), Write after Read (WAR), and Write after Write (WAW).
- A bubble or a pipeline bubble represents a stage in the pipeline that cannot perform any useful operation due to the lack of data from previous stage of the pipeline.

Space for learners:

1.9 ANSWERS TO CHECK YOUR PROGRESS

i. b	ii. a	iii. d	iv. a	v. b
vi. d	vii. c	viii. c	ix. d	x. a
xi. d	xii. d	xiii. a	xiv.c	xv.d

1.10 POSSIBLE QUESTIONS

Q1 Discuss Flynn's classification of computer architecture.

Q2 According to Flynn's classification, the architecture which is of theoretical interest but no real-world system has been developed on it?

Q3 Differentiate between shared memory and distributed memory MIMD systems.

Q4 Explain how pipelining can increase the performance of a system compared to a single processor system.

Q5 Differentiate between superscalar processor and Super pipeline processor.

Q6 Briefly describe the parameters on which different pipelined processors are measured in terms of their performance.

Q7 Discuss the types of processors that is helpful in parallel processing.

Q8 Discuss the factors that affect the performance of a pipeline.

Q9 Define instruction pipeline with the help of an example.

Q10 Discuss resource conflict in pipelining.

Q11 Discuss Data hazard in pipelining.

Q12 What is a pipeline bubble? In what situation a pipeline bubble is used?

Space for learners:

1.11 REFERENCES AND SUGGESTED READINGS

- Advanced Computer Architecture, 3e, Kai Hwang, Naresh Jotwani; McGraw-Hill Education, 2016
- Computer Organization and Architecture: Designing for Performance 10 Edition, by William Stallings, Pearson.
- Computer System Architecture Third Edition, M. Morris Mano, Rajib Mall, Pearson
- Computer Organization Fifth Edition, Carl Hamacher, McGraw Hill

---×---

UNIT 2: VECTOR PROCESSING

Space for learners:

Unit Structure:

2.1 INTRODUCTION

A normal processor sometimes called scalar processor, which works on simple instruction at a time, which operates on single data items. Standard von Neumann machine is based on the instruction and data are stored in memory, that has one operation at a time, maximum speed of the system is limited by the memory bandwidth(bits/sec or bytes/sec). It means normal processing having limitation on memory bandwidth also memory is shared by CPU and I/O. But in today's world, this technique will prove to be highly inefficient, as the overall processing of instructions will be very slow.

There is a class of computational problems that are beyond the capabilities of a conventional computer. These problems require vast number of computations on multiple data items that will take a conventional computer(with scalar processor) days or even weeks to complete.

Such complex instruction, which operates on multiple data at the same time, requires a better way of instruction execution, which was achieved by vector computing technique using vector processors. Scalar CPUs can manipulate one or two data items at a time, which is not very efficient.

Space for learners:

2.2 UNIT OBJECTIVES

After going through this unit you will be able to:

- Understand the basic concepts of vector computing and working principle of vector processor.
- Know about the pipelining techniques applied in vector computing.

- Understand how arithmetic pipelining works.
- Give the basic concept of array processor and its different categories.
- Know about the vector and scalar registers used in vector processing.
- Define what is a chaining and scatter-gather operation.
- Understand about register-register and memory-memory vector processors.

Space for learners:

2.3 VECTOR COMPUTING

There is a class of computational problems that are beyond the capabilities of a conventional computer. These problems require vast number of computations on multiple data items that will take a conventional computer (with scalar processor) days or even weeks to complete.

Such complex instructions, which operates on multiple data at the same time, requires a better way of instruction execution, which has been achieved by the vector computing technique that done by vector processors.

Vector processor is basically a central processing unit that has the ability to execute the complete vector input in a single instruction. So, we can say vector processing allows operation on multiple data elements by the help of single instruction.

Scalar CPUs can manipulate one or two data items at a time, which is not very efficient. Also, simple instructions like ADD A to B, and store into C are not practically efficient.

Addresses are used to point to the memory location where the data to be operated will be found, which leads to added overhead of data lookup. So until the data is found, the CPU would be sitting ideal, which is a big performance issue.

Hence, the concept of Instruction Pipeline comes into picture, in which the instruction passes through several sub-units in turn. These sub-units perform various independent functions, for example: the first one decodes the instruction, the second sub-unit fetches the data and the third sub-unit performs the math itself. Therefore, while the data is fetched for one instruction, CPU does not sit idle; it rather works on decoding the next instruction set, ending up working like an assembly line.

Vector computing technique, not only use Instruction pipeline, but it also pipelines the data, working on multiple data at the same time.

A normal scalar processor instruction would be *ADD A, B*, which leads to addition of two operands, but what if we can instruct the processor to ADD a group of numbers (from 0 to n memory location) to another group of numbers (let's say, *n* to *k* memory location) then a scalar processor cannot able to add of these set values. This can be achieved by vector processors.

In vector processor a single instruction, can ask for multiple data operations, which saves time, as instruction is decoded once, and then it keeps on operating on different data items.

Space for learners:

2.4 VECTOR PROCESSOR

Vector processor is basically a central processing unit that has the ability to execute the complete vector input in a single instruction. More specifically we can say, it is a complete unit of hardware resources that executes a sequential set of similar data items in the memory using a single instruction.

Vector processors are co-processor to general-purpose microprocessor. Vector processors are generally register-register or memory-memory. A vector instruction is fetched and decoded and then a certain operation is performed for each element of the operand vectors, whereas in a normal processor a vector operation needs a loop structure in the code. To make it more efficient, vector processors chain several vector operations together, i.e., the result from one vector operation are forwarded to another as operand.

We know elements of the vector are ordered properly so as to have successive addressing format of the memory. This is the reason why we have mentioned that it implements the data sequentially. It holds a single control unit but has multiple execution units that perform the same operation on different data elements of the vector.

Unlike scalar processors that operate on only a single pair of data, a vector processor operates on multiple pair of data. However, one can convert a scalar code into vector code. This conversion process is known as vectorization. So, we can say vector processing allows operation on multiple data elements by the help of single instruction.

These instructions are said to be single instruction multiple data (SIMD) or vector instructions**.** The CPU used in recent time makes use of vector processing as it is advantageous than scalar processing.

A vector processor is a processor that can operate on an entire vector in one instruction. The operand to the instructions are complete vectors instead of one element. Vector processors reduce the fetch and decode bandwidth as the number of instructions fetched are less. They also exploit data parallelism in large scientific and multimedia applications. Based on how the operands are fetched, vector processors can be divided into two categories - in memory-memory architecture operands are directly streamed to the

Space for learners:

functional units from the memory and results are written back to memory as the vector operation proceeds. In vector-register architecture, operands are read into vector registers from which they are fed to the functional units and results of operations are written to vector registers. Many performance optimization schemes are used in vector processors. Memory banks are used to reduce load/store latency. Strip mining is used to generate code so that vector operation is possible for vector operands whose size is less than or greater than the size of vector registers. Vector chaining - the equivalent of forwarding in vector processors - is used in case of data dependency among vector instructions. Special scatter and gather instructions are provided to efficiently operate on sparse matrices.

Instruction set has been designed with the property that all vector arithmetic instructions only allow element *N* of one vector register to take part in operations with element *N* from other vector registers. This dramatically simplifies the construction of a highly parallel vector unit, which can be structured as multiple parallel lanes. As with a traffic highway, we can increase the peak throughput of a vector unit by adding more lanes.

Space for learners:

2.4.1 Some Important Facts on a Vector Processor

- A vector processor is an ensemble of hardware resources, including vector registers, functional pipelines, processing elements and register counters for performing register operations.
- Vector processing occurs when arithmetic or logical operations are applied to vectors. It is distinguished from scalar processing which operates on one or one pair of data.

The conversion from scalar code to vector code is called vectorization.

- Both pipelined processors and SIMD computers can perform vector operations.
- Vector processing reduces software overhead incurred in the maintenance of looping control, reduces memory access conflicts and above all matches nicely with pipelining and segmentation concept to generate one result per each clock cycle continuously.

Space for learners:

2.4.2 Advantages of Vector Processor

Some advantages of vector processors are given below:

- Programs size is small as it requires less number of instructions. Vector instructions also hide many branches by executing a loop in one instruction.
- Vector memory access has no wastage like cache access. Every data item requested by the processor is actually used.
- Once a vector instruction starts operating, only the functional unit (FU) and the register buses feeding it need to be powered. Fetch unit, decode unit, ROB etc can be powered off. This reduces the power usage.

2.4.3 Applications of Vector Processors

Computer with vector processing capabilities are in demand in specialized applications. The following are some areas where vector processing is used:

1. Petroleum exploration.
2. Medical diagnosis.

3. Data analysis.
4. Weather forecasting.
5. Aerodynamics and space flight simulations.
6. Image processing.
7. Artificial intelligence.

Space for learners:

2.4.4 Cost of Vector Processor

The reason behind the declining popularity of vector processors are their cost as compared to multiprocessors and superscalar processors. The reasons behind high cost of vector processors are

• Vector processors do not use commodity parts. Since they sell very few copies, design cost dominates overall cost.

• Vector processors need high speed on-chip memory which are expensive.

• It is difficult to package the processors with such high speed. In the past, vector manufactures have employed expensive designs for this.

• There have been few architectural innovations compared to superscalar processors to improve performance keeping the cost low.

2.4.5 Classification of Vector Processor

The classification of vector processor relies on the ability of vector formation as well as the presence of vector instruction for processing. So, depending on these criteria, vector processing is classified as follows:

(i) Register to Register Architecture (Vector register processors)and

(ii) Memory to Memory Architecture(Memory-memory vector processors)

According to from where the operands are retrieved in a vector processor, pipe lined vector computers are classified into two architectural configurations:

Space for learners:

2.4.5.1 Memory to memory architecture

In memory to memory architecture, source operands, intermediate and final results are retrieved (read) directly from the main memory. For memory to memory vector instructions, the information of the base address, the offset, the increment, and the vector length must be specified in order to enable streams of data transfers between the main memory and pipelines. The processors like *TI-ASC, CDC STAR-100, and Cyber-205* have vector instructions in memory to memory formats. The main points about memory to memory architecture are:

- There is no limitation of size
- Speed is comparatively slow in this architecture

2.4.5.2 Register to Register Architecture

This architecture is highly used in vector computers. As in this architecture, the fetching of the operand or previous results indirectly takes place through the main memory by the use of registers. The several vector pipelines present in the vector computer help in retrieving the data from the registers and also storing the results in the desired register. These vector registers are user instruction programmable. In a vector-register processor, all vector operations—except load and store—are among the vector

registers. These architectures are the vector counterpart of a load-store architecture.

All major vector computers shipped since the late 1980s use a vector-register architecture, including the Cray Research processors (Cray-1, Cray-2, X-MP,YMP, C90, T90, SV1, and X1), the Japanese supercomputers (NEC SX/2 through SX/8, Fujitsu VP200 through VPP5000, and the Hitachi S820 and S-8300), and the mini-supercomputers (Convex C-1 through C-4).

In register to register architecture, operands and results are retrieved indirectly from the main memory through the use of large number of vector registers or scalar registers. The processors like *Cray-1 and the Fujitsu VP-200* use vector instructions in register to register formats. The main points about register to register architecture are:

(i) Register to register architecture has limited size.

(ii) Speed is very high as compared to the memory to memory architecture.

(iii) The hardware cost is high in this architecture.

Space for learners:

2.5 SUPERSCALAR PROCESSOR

It was first invented in 1987. It is a machine which is designed to improve the performance of the scalar processor. In most applications, most of the operations are on scalar quantities. Superscalar approach produces the high performance general purpose processors.

A scalar processor works on one or two data items, it is a normal processor, which works on simple instruction at a time, which operates on single data items, while the vector processor works with multiple data items. A superscalar processor is a combination of

both. Each instruction processes one data item, but there are multiple execution units within each CPU thus multiple instructions can be processing separate data items concurrently.

The main principle of superscalar approach is that it executes instructions independently in different pipelines. As we already know, that Instruction pipelining leads to parallel processing thereby speeding up the processing of instructions. In Superscalar processor, multiple such pipelines are introduced for different operations, which further improves parallel processing.

There are multiple functional units each of which is implemented as a pipeline. Each pipeline consists of multiple stages to handle multiple instructions at a time which support parallel execution of instructions.

It increases the throughput because the CPU can execute multiple instructions per clock cycle. Thus, superscalar processors are much faster than scalar processors.

While a superscalar CPU is also pipelined, there are two different performance enhancement techniques. It is possible to have a non-pipelined superscalar CPU or pipelined non-superscalar CPU. The superscalar technique is associated with some characteristics, these are given below:

- Instructions are issued from a sequential instruction stream.
- CPU must dynamically check for data dependencies.
- Should accept multiple instructions per clock cycle.

Space for learners:

2.6 VECTOR COMPUTER

Space for learners:

The functional units of a vector computer are as follows:

(i) IPU or instruction processing unit
(ii) Vector register
(iii)Scalar register
(iv)Scalar processor
(v) Vector instruction controller
(vi)Vector access controller
(vii) Vector processor

Let us now understand the overall operation performed by the vector computer.

As it has several functional pipes thus it can execute the instructions over the operands. We know that both data and instructions are present in the memory at the desired memory location. So, the instruction processing unit i.e., IPU fetches the instruction from the memory.

Once the instruction is fetched then IPU determines either the fetched instruction is scalar or vector in nature. If it is scalar in nature, then the instruction is transferred to the scalar register and then further scalar processing is performed.

While, when the instruction is a vector in nature then it is fed to the vector instruction controller. This vector instruction controller first decodes the vector instruction then accordingly determines the address of the vector operand present in the memory.

Then it gives a signal to the vector access controller about the demand of the respective operand. This vector access controller then fetches the desired operand from the memory. Once the operand is

fetched then it is provided to the instruction register so that it can be processed at the vector processor.

At times when multiple vector instructions are present, then the vector instruction controller provides the multiple vector instructions to the task system. And in case the task system shows that the vector task is very long then the processor divides the task into sub-vectors.

These sub-vectors are fed to the vector processor that makes use of several pipelines in order to execute the instruction over the operand fetched from the memory at the same time. The various vector instructions are scheduled by the vector instruction controller.

A block diagram of a modern multiple pipeline vector computer is shown below:

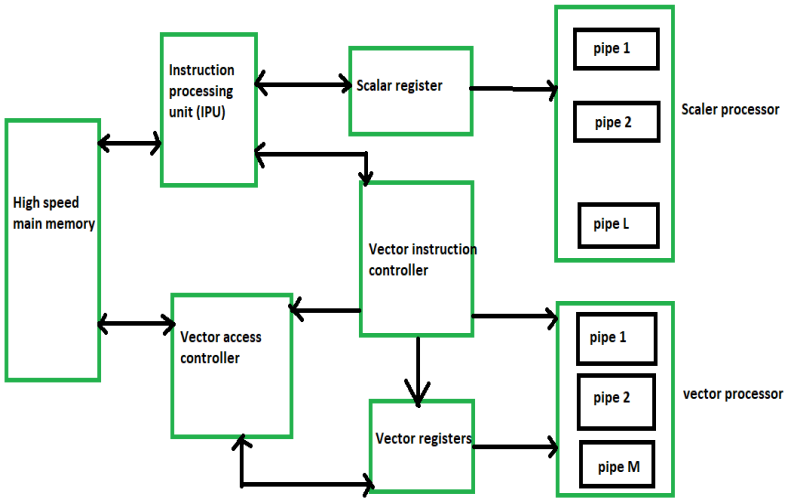


Fig.2.1 A block diagram of a modern multiple pipeline vector computer

2.6.1 Vector registers

Vector registers are the storage areas in a CPU core that contain the operands for vector computations, as well as the results. The size of

Space for learners:

the vector registers determines the level of SIMD instructions that can be supported by a given processor's CPUs.

Each vector register is a fixed-length bank holding a single vector. VMIPS has eight vector registers, and each vector register holds 64elements. Each vector register must have at least two read ports and one write port in VMIPS. This will allow a high degree of overlap among vector operations to different vector registers. The read and write ports, which total at least 16 read ports and 8write ports, are connected to the functional unit inputs or outputs by a pair of crossbars. Real machines make use of the regular access pattern within a vector instruction to reduce the costs of the vector-register file circuitry. For example, the Cray-1 manages to implement the register file with only a single port per register.

Space for learners:

2.6.2 Scalar registers

Scalar processors represent a class of computer processors. A scalar processor processes only one data item at a time, with typical data items being integers or floating point numbers. A scalar processor is classified as a single instruction, single data (SISD) processor in Flynn's taxonomy.

Scalar registers can also provide data as input to the vector functional units, as well as compute addresses to pass to the vector load-store unit. These are the normal 32 general-purpose registers and 32 floating-point registers of MIPS. Scalar values are read out of the scalar register file, then latched at one input of the vector functional units.

2.7 ARRAY PROCESSORS

Array processors are also known as multiprocessors or vector processors. An array processor is a processor that performs computations on large arrays of data. Thus, they are used to improve the performance of the computer.

In other words, an array processor is a CPU which implements an instruction set that are designed to operate efficiently and effectively on large one-dimensional arrays of data called vectors.

Vector and array processing are essentially the same because, with slight and rare differences, a vector processor and an array processor are the same type of processor. A vector processor is in contrast of the simpler scalar processor, which handles only one piece of information at a time.

2.7.1Types of Array Processors

There are basically two types of array processors:

1. Attached Array Processors
2. SIMD Array Processors

2.7.1.1 Attached Array Processors

An attached array processor is a processor which is attached to a general purpose computer and its purpose is to enhance and improve the performance of that computer in numerical computational tasks. It achieves high performance by means of parallel processing with multiple functional units. The objective of the attached array processor is to provide vector manipulation capabilities to a conventional computer at a fraction of the cost of supercomputer.

Space for learners:

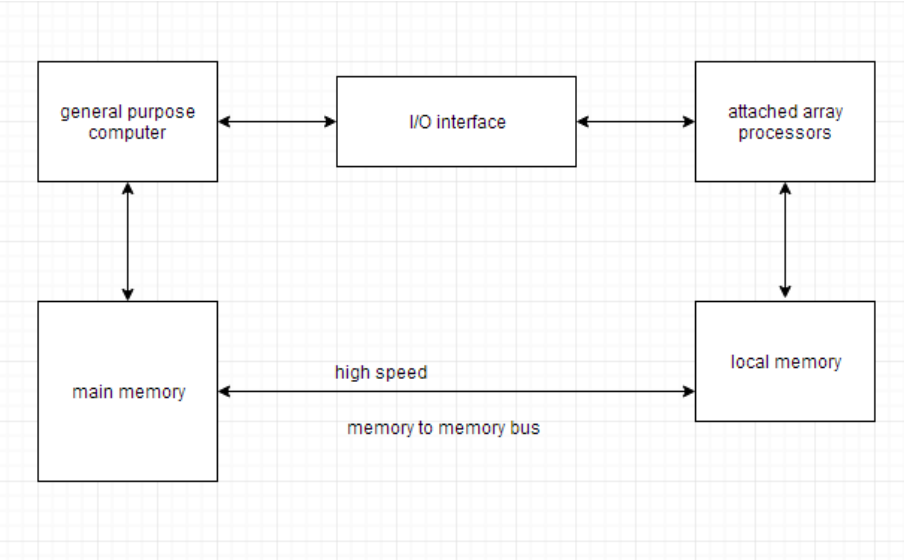


Fig.2.2 Block diagram of Attached Array Processors

Space for learners:

2.7.1.2 SIMD Array Processors

Single-instruction, multiple data (SIMD) is the organization of a single computer containing multiple processors operating in parallel. The processing units are made to operate under the control of a common control unit, thus providing a single instruction stream and multiple data streams.

A general block diagram of an array processor is shown below. It contains a set of identical processing elements (PE's), each of which is having a local memory M. Each processor element includes an ALU and registers. The master control unit controls all the operations of the processor elements. It also decodes the instructions and determines how the instruction is to be executed.

The main memory is used for storing the program. The control unit is responsible for fetching the instructions. Vector instructions are send to all PE's simultaneously and results are returned to the memory.

The best known SIMD array processor is the ILLIAC IV computer developed by the Burroughs corps. SIMD processors are highly specialized computers. They are only suitable for numerical problems that can be expressed in vector or matrix form and they are not suitable for other types of computations.

Space for learners:

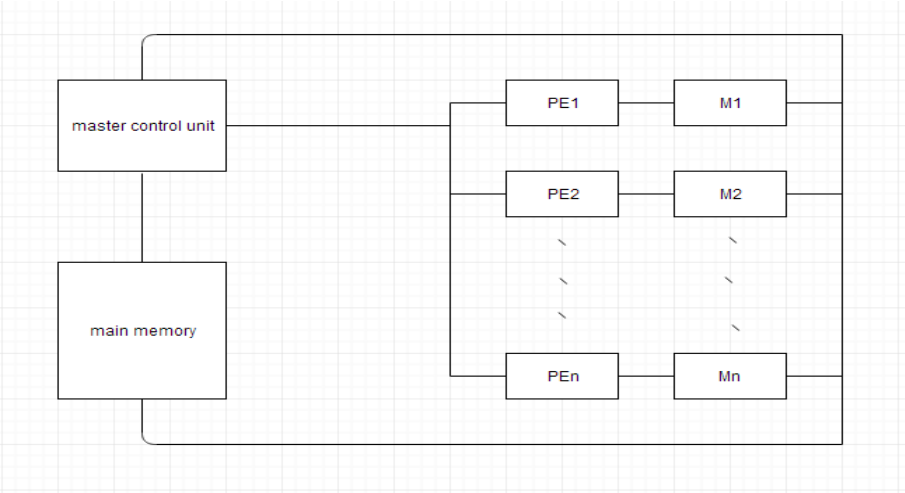


Fig.2.3A general block diagram of an array processor

2.7.2 Advantages of Array Processor

- An array processor increases the overall instruction processing speed.
- As most of the Array processors operate asynchronously from the host CPU, hence it improves the overall capacity of the system.
- Array Processors has its own local memory, hence providing extra memory for systems with low memory.

2.8 PIPELINING

Pipelining is the process of accumulating instruction from the processor through a pipeline. It allows storing and executing instructions in an orderly process. It is also known as pipeline processing.

Pipelining is a technique where multiple instructions are overlapped during execution. Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end.

Pipelining increases the overall instruction throughput.

In pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and combinational circuit performs operations on it. The output of combinational circuit is applied to the input register of the next segment.

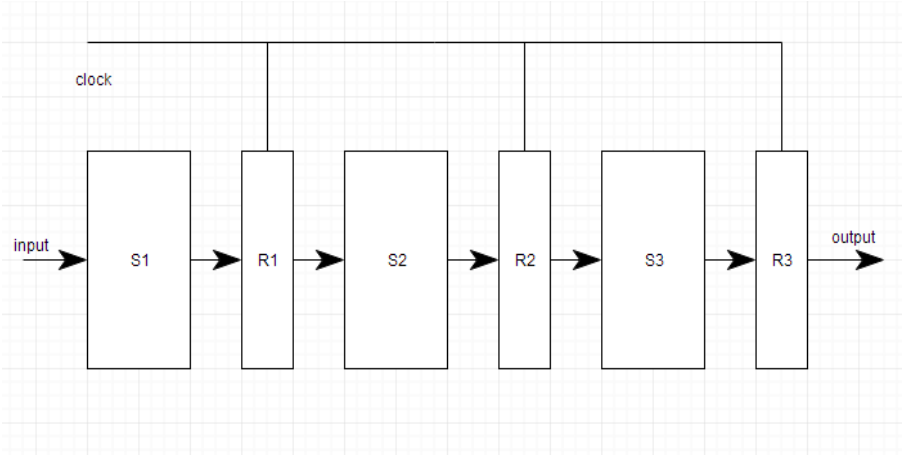


Fig.2.4 A general block diagram of a pipeline system

Pipeline system is like the modern day assembly line setup in factories. For example in a car manufacturing industry, huge assembly lines are setup and at each point, there are robotic arms to perform a certain task, and then the car moves on ahead to the next arm.

In summary, we can say Pipelining is a technique of

- Decomposing a sequential process into sub-operations (segments).

Space for learners:

- Divide the processor into segment processors each one is dedicated to a particular segment.
- Each segment is executed in a dedicated segment processor operates concurrently with all other segments.
- Information flows through these multiple hardware segments.
- The overlapping of computation is made possible by associating a register with each segment in the pipeline.
- The registers provide isolation between each segment so that each can operate on distinct data simultaneously.

Space for learners:

2.8.1 Types of Pipeline

It is divided into two categories:

1. Arithmetic Pipeline
2. Instruction Pipeline

2.8.1.1 Arithmetic Pipeline

Arithmetic pipelines are usually found in most of the computers. They are used for floating point operations, multiplication of fixed point numbers etc. For example: The input to the Floating Point Adder pipeline is:

Suppose X=A*2^a and Y=B*2^b

Here A and B are mantissas (significant digit of floating point numbers), while a and b are exponents.

The floating point addition and subtraction is done in 4 parts:

1. Compare the exponents.
2. Align the mantissas.
3. Add or subtract mantissas
4. Produce the result.

Registers are used for storing the intermediate results between the above operations.

An arithmetic pipeline divides an arithmetic problem into various sub problems for execution in various pipeline segments. It is used for floating point operations, multiplication and various other computations. The process or flowchart arithmetic pipeline for floating point addition is shown in the below diagram.

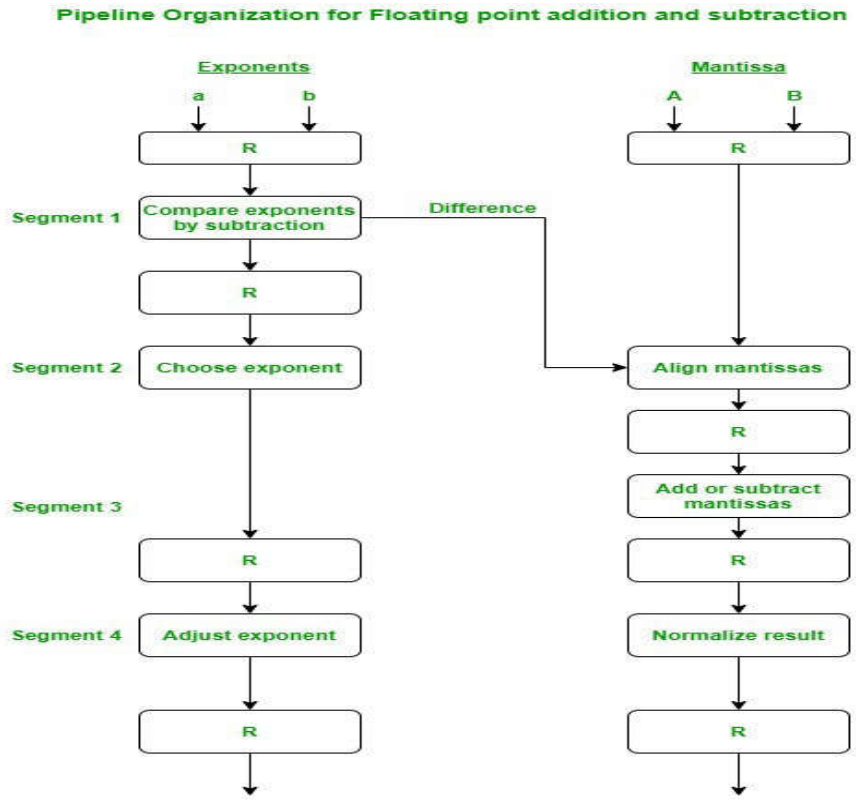


Fig 2.5 Pipelining for floating point addition and subtraction.

Floating point addition using arithmetic pipeline

The following sub operations are performed in this case:

1. Compare the exponents.
2. Align the mantissas.
3. Add or subtract the mantissas.
4. Normalise the result

Space for learners:

First of all the two exponents are compared and the larger of two exponents is chosen as the result exponent. The difference in the exponents then decides how many times we must shift the smaller exponent to the right. Then after shifting of exponent, both the mantissas get aligned. Finally the addition of both numbers take place followed by normalisation of the result in the last segment.

Example:
Let us consider two numbers,
X=0.3214*10^3 and Y=0.4500*10^2

Explanation:

First of all the two exponents are subtracted to give 3-2=1. Thus 3 becomes the exponent of result and the smaller exponent is shifted 1 times to the right to give

Y=0.0450*10^3

Finally the two numbers are added to produce

Z=0.3664*10^3

As the result is already normalized the result remains the same.

Space for learners:

2.8.1.2 Instruction Pipeline

In this a stream of instructions can be executed by overlapping *fetch*, *decode* and *execute* phases of an instruction cycle. This type of technique is used to increase the throughput of the computer system.

An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline. Thus we can execute multiple instructions simultaneously. The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.

In this a stream of instructions can be executed by overlapping fetch, decode and execute phases of an instruction cycle. This type of technique is used to increase the throughput of the computer system. An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline. Thus we can execute multiple instructions simultaneously. The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.

In the most general case computer needs to process each instruction in following sequence of steps:

1. Fetch the instruction from memory (FI)
2. Decode the instruction (DA)
3. Calculate the effective address
4. Fetch the operands from memory (FO)
5. Execute the instruction (EX)
6. Store the result in the proper place

The flowchart for instruction pipeline is shown below.

Space for learners:

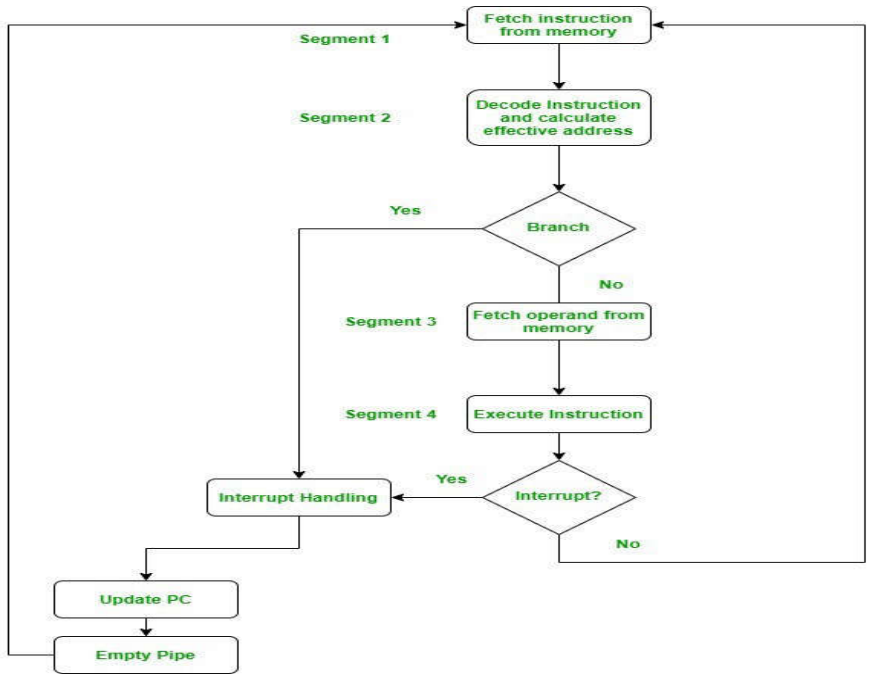


Fig 2.6 Flowchart for instruction Pipelining.

Let us see an example of instruction pipeline.

Example:

	Stage	1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
Branch	3			FI	DA	FO	EX							
	4				FI	---	---	FI	DA	FO	EX			
	5								FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

Here the instruction is fetched on first clock cycle in segment 1. Now it is decoded in next clock cycle, then operands are fetched and finally the instruction is executed. We can see that here the fetch and decode phase overlap due to pipelining. By the time the first instruction is being decoded, next instruction is fetched by the pipeline.

Space for learners:

In case of third instruction we see that it is a branched instruction. Here when it is being decoded 4th instruction is fetched simultaneously. But as it is a branched instruction it may point to some other instruction when it is decoded. Thus fourth instruction is kept on hold until the branched instruction is executed. When it gets executed then the fourth instruction is copied back and the other phases continue as usual.

Space for learners:

2.8.2 Pipeline Conflicts

There are some factors that cause the pipeline to deviate its normal performance. Some of these factors are given below:

(i) Timing Variations

All stages cannot take same amount of time. This problem generally occurs in instruction processing where different instructions have different operand requirements and thus different processing time.

(ii) Data Hazards

When several instructions are in partial execution, and if they reference same data then the problem arises. We must ensure that next instruction does not attempt to access data before the current instruction, because this will lead to incorrect results.

(iii) Branching

In order to fetch and execute the next instruction, we must know what that instruction is. If the present instruction is a conditional branch, and its result will lead us to the next instruction, then the next instruction may not be known until the current one is processed.

(iv) Interrupts

Interrupts set unwanted instruction into the instruction stream. Interrupts effect the execution of instruction.

(v) Data Dependency

It arises when an instruction depends upon the result of a previous instruction but this result is not yet available.

Space for learners:

2.8.3 Advantages of Pipelining

1. The cycle time of the processor is reduced.
2. It increases the throughput of the system
3. It makes the system reliable.

2.8.4 Disadvantages of Pipelining

1. The design of pipelined processor is complex and costly to manufacture.
2. The instruction latency is more.

2.9 CHAINING TECHNIQUE

In computing, chaining is a technique used in computer architecture in which scalar and vector registers generate interim results which can be used immediately, without additional memory references which reduce computational speed.

Chaining allows the results of one vector operation to be directly used as input to another vector operation. A convoy is a set of vector instructions that can potentially execute together. Only structural hazards cause separate convoys as true dependences are handled via chaining in the same convoy.

2.10 GATHER-SCATTER OPERATION

Gather and scatter are two fundamental data-parallel operations, where a large number of data items are read (gathered) from or are written (scattered) to given locations.

Gather-scatter is also a type of memory addressing operation that often arises when addressing vectors in sparse linear algebra operations. It is the vector-equivalent of register indirect addressing, with gather involving indexed reads and scatter indexed writes. Vector processors (and some SIMD units in CPUs) have hardware support for gather-scatter operations, providing instructions such as *Load Vector Indexed* for gather and *Store Vector Indexed* for scatter.

2.10.1 The basic concepts of Gather-scatter

We are generally used to organizing our memories by row. Caches are built from rows so if we want one piece of data, we get the whole row. If we want to manage our performance tightly, then we try to have as many related variables as possible on the same row so that we get more bangs for our caching buck and reduce our cache misses.

The nice thing about a row of memory is that, especially with vector structures like SIMD (single-instruction, multiple data), we can operate on multiple pieces of data at the same time, in parallel. At the very least, if we can't do it in parallel, then we can loop along the row for the operation without further fetching hassles.

But there are several contexts where the world doesn't cooperate with this row-by-row structure. What if we want to be able to do is exactly that same thing, but without the requirement that addresses be contiguous?

Space for learners:

This isn't so easy to do, since we need lots of fetches to populate a vector; we can't just copy over a chunk of memory and get busy operating on it. The idea is to find a way to "gather" data from far-flung locations, work with them as a single vector, and then, if we desired, take the results and "scatter" them back out into their original far-flung locations.

Space for learners:

2.10.2 Different gather-scatter applications

Some application of gather-scatter operations are given below:

- A single block of in-memory data may represent data from a file that has been fractured into various sectors across the storage medium.
- A single in-memory buffer, if too large, may cause problems due to memory fragmentation. It can be more easily managed if it is stored in smaller fragments, but this requires management to make them look contiguous.
- Network traffic streams may be split up as they arrive, with various buckets in memory. This is referred to as "Scatter/gather I/O." In a way, this is the reverse of other applications. In other applications, scattered data is brought together in the processor. With this streaming version, it's a unified stream that then gets scattered about as it arrives at the processor.
- Embedded systems may require low-level access to data that's scattered throughout DRAM, treating it as contiguous. The illustrations above reflect this application. As we'll see, vision is a major driver of this usage.

CHECK YOUR PROGRESS:

Multiple Choice Questions:

1. A processor, which works on simple instruction at a time, which operates on single data items is known as
   (A) Scalar (B) Vector (C) Array
   (D) Superscalar

2.A processor that has the ability to execute the complete vector input in a single instruction is called
   (A) Scalar (B) Vector (C) Normal
   (D) Superscalar

3.In memory to memory architecture, source operands, intermediate and final results are retrieved (read) directly from

   (A) Main memory (B) Register
   (C)Cache (D) Secondary memory

4. SIMD means
   (A) Single Instruction Many Data
   (B) Simple Instruction Multiple Data
   (C) Single-Instruction, Multiple Data
   (D) None of above

5.A technique where multiple instructions are overlapped during execution is known as

   (A) Gathering (B) Scattering
   (C) Chaining (D) Pipelining

Space for learners:

2.11 SUMMING UP

- Vector processor is basically a central processing unit that has the ability to execute the complete vector input in a single instruction. So, we can say vector processing allows operation on multiple data elements by the help of single instruction.
- Scalar CPUs can manipulate one or two data items at a time, which is not very efficient. Also, simple instructions

like *ADD A to B*, and store into *C* are not practically efficient.

- Vector computing technique, not only use instruction pipeline, but it also pipelines the data, working on multiple data at the same time.
- Vector processor is basically a central processing unit that has the ability to execute the complete vector input in a single instruction. More specifically we can say, it is a complete unit of hardware resources that executes a sequential set of similar data items in the memory using a single instruction.
- Vector processing occurs when arithmetic or logical operations are applied to vectors. It is distinguished from scalar processing which operates on one or one pair of data. The conversion from scalar code to vector code is called vectorization.
- Programs size is small as it requires less number of instructions. Vector instructions also hide many branches by executing a loop in one instruction.
- Vector processors need high speed on-chip memory which are expensive.
- It is difficult to package the processors with such high speed. In the past, vector manufactures have employed expensive designs for this.
- The classification of vector processor relies on the ability of vector formation as well as the presence of vector instruction for processing. So, depending on these criteria, vector processing is classified as follows:

(iii) Register to Register Architecture (Vector register processors)and

Space for learners:

(iv) Memory to Memory Architecture(Memory-memory vector processors)

- In memory to memory architecture, source operands, intermediate and final results are retrieved (read) directly from the main memory.
- Register to register architecture is highly used in vector computers. As in this architecture, the fetching of the operand or previous results indirectly takes place through the main memory by the use of registers.
- In Superscalar processor, multiple such pipelines are introduced for different operations, which further improves parallel processing.
- Vector registers are the storage areas in a CPU core that contain the operands for vector computations, as well as the results. The size of the vector registers determines the level of SIMD instructions that can be supported by a given processor's CPUs.
- Scalar registers can also provide data as input to the vector functional units, as well as compute addresses to pass to the vector load-store unit.

- An array processor is a CPU which implements an instruction set that are designed to operate efficiently and effectively on large one-dimensional arrays of data called vectors.
- An attached array processor is a processor which is attached to a general purpose computer and its purpose is to enhance and improve the performance of that computer in numerical computational tasks.
- The objective of the attached array processor is to provide vector manipulation capabilities to a conventional computer at a fraction of the cost of supercomputer.

Space for learners:

- Single-instruction, multiple data (SIMD) is the organization of a single computer containing multiple processors operating in parallel.
- Pipelining is the process of accumulating instruction from the processor through a pipeline. It allows storing and executing instructions in an orderly process. It is also known as pipeline processing.
- Pipelining is a technique where multiple instructions are overlapped during execution.
- Arithmetic pipelines are usually found in most of the computers. They are used for floating point operations, multiplication of fixed point numbers etc.
- An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline.
- In computing, chaining is a technique used in computer architecture in which scalar and vector registers generate interim results which can be used immediately, without additional memory references which reduce computational speed.
- Gather and scatter are two fundamental data-parallel operations, where a large number of data items are read (gathered) from or are written (scattered) to given locations.

Space for learners:

2.12 ANSWER TO CHECK YOUR PROGRESS

Answer: 1 (A), 2 (B), 3 (A), 4 (C), 5 (D).

2.13 POSSIBLE QUESTIONS

Short Type Questions:

1. What is vector computing? How it differ from scalar computing?
2. What do you mean vector processor?
3. What are the advantages of vector processor?
4. What is array processor? What are its different categories?
5. What do you mean by pipelining in vector processing?

Long Answer Type Questions:

1. Explain about the arithmetic pipeline and instruction pipeline techniques.
2. Explain how vector-register processor differs from memory vector processor.
3. Explain about the working principle of array processor.
4. What do you mean by chaining? Explain about the scatter-gather techniques.

Space for learners:

2.14 REFERENCES AND SUGGESTED READINGS

- M. Morris Mano, "Computer System Architecture", 3nd Edition, Pearson,2006.
- William Stalling, "Computer Organization and Architecture", 8th Edition, Pearson, 2010.
- John P. Hayes, "Computer Architecture and Organization", 2nd Edition, McGraw-Hill International Edition, 1988.

---×---

UNIT 3: ADVANCE CONCEPT OF COMPUTER ARCHITECTURE PLICITPARALLELISM

Unit Structure:

Space for learners:

3.1 INTRODUCTION

Implicit parallelism allows programmers to write down their programs without any worry about parallelism exploitation. The exploitation of parallelism is instead automatically performed by the compiler and the runtime system. Thus, the parallelism is transparent to the programmer, maintaining the complication of software development at the same level as standard sequential programming. Implicit parallelism generally facilitates the design of parallel programs and therefore substantially improves programmer efficiency and productivity. Different applications utilize different aspects of parallelism - e.g., data-intensive applications utilize high aggregate throughput, server applications utilize high aggregate network bandwidth, and scientific applications typically utilize high processing and memory system performance. It is important to realize each of these performance bottlenecks and their interacting effect.

In this unit, you will learn about the pipelining technique, and the comparison/ discussion of super pipeline and super scalar pipeline will describe in this unit. Various classes of superscalar architecture will be discussed in this unit. You will learn the measurement of the performance of pipeline architecture. Some of the benefits and drawbacks of the superscalar pipeline will be pointed out in this unit. You will learn the need for Branch prediction in the pipeline. Different branch prediction techniques (static and dynamic prediction) will be discussed with the proper example and diagram in this unit. You will learn various hazards (structural hazards, control hazards, and data hazards)that occur in the pipelining. Some of the delay slots will be discussed in this unit. You also learn the out-of-order execution and register renaming concept with an example at the last of the unit.

Space for learners:

3.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- understand the needs of implicit parallelism techniques.
- describe the basic structure of the pipeline

- know different stages of instruction in the pipeline
- understand the design concept of super pipeline and superscalar pipeline
- understand the branch prediction logic
- understand the various Hazards in the pipeline
- know the idea of delay slots
- Describe out-of-order execution and Register Renaming

Space for learners:

3.3 INTRODUCTION OF PIPELINING

Pipelining is the practice of accumulating instruction from the processor through a pipeline. In pipelining, storing and executing of the instructions allows being in an orderly process. It is also known as pipeline processing. Multiple instructions are overlapped during execution in pipelining that's why process microprocessor begins executing a second instruction before the first instruction has been completed. A pipeline is separated into stages, and these stages are attached to one another to form a pipe like structure. Instructions enter from one end and exit from another end. Pipelining improved the overall instruction throughput.

In the pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and a combinational circuit performs operations on it. The output of the combinational circuit is applied to the input register of the next segment.

A processing circuit of a given stage is connected to the input latch of the next stage **(Figure 3.1)**. A clock signal is connected to each input latch. Every stage transfers its intermediate result to the input latch of the next stage on every clock pulse. This way, the final outcome is produced after the input data have passed through the whole Pipeline, finishing one stage for every clock pulse. The clock pulse period should be large enough to grant sufficient time for a

signal to go across through the slowest stage where the most extended amount of time to need complete (bottleneck stage). Also, there should be sufficient time for a latch to store its input signals.

If the clock's period, P, is expressed as $\mathbf{P = t_b + t_l}$, then $\mathbf{t_b}$ should be bigger than the utmost delay of the bottleneck stage, and t_l should be sufficient for storing data into a latch.

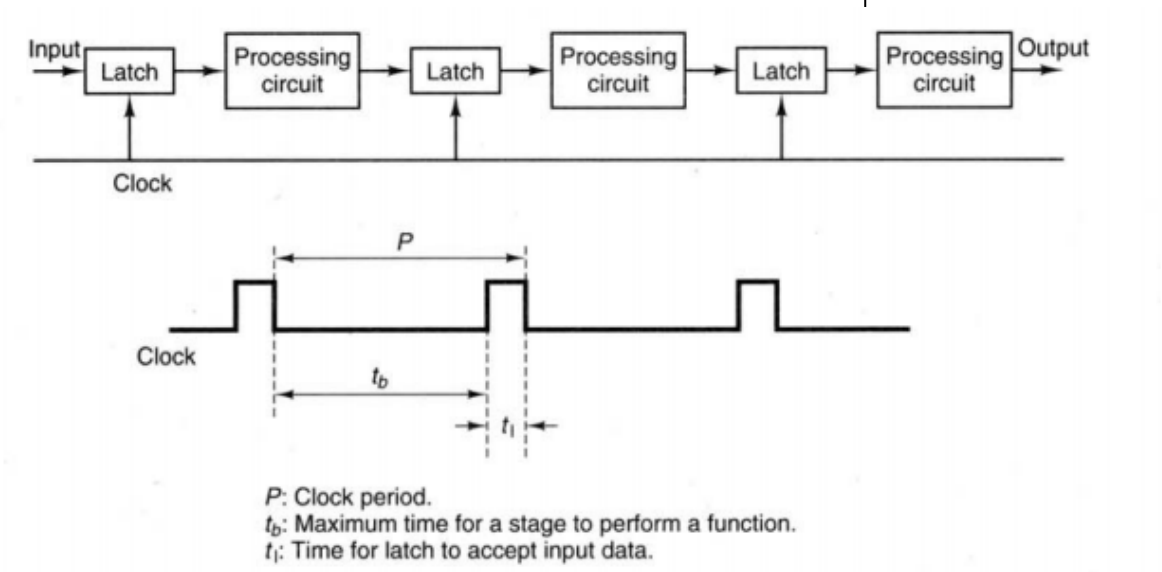


Figure 3.1: Basic structure of a pipeline. *adapted from [1]*

The instruction in pipelining is divided into five subtasks likely

1. Instruction Fetch (IF): In this subtask, the instruction is fetched.

2. Instruction Decode(ID): Here, the fetched instruction is decoded.

3. Operand Fetch (OF): In this stage, the operand is fetched of the instruction.

4. Instruction Execute (IE): In this stage,arithmetic and logical operations are performed on the operands to execute the instruction.

5.Operand Store (OS): The result of the earlier stage is stored in the memory.

Space for learners:

Let us visualize how pipelining is done for *N* numbers of instructions. In the **Figure 3.2**given below four instructions are pipelined. The instruction-1 gets completed in 5 clock cycles. After the first instruction is completed in every new clock, the proceeding instruction(i.e., Instruction 2, 3 & 4) completes its execution.

Pipeline system is like some assembly line set up in different factories. For example, in an automobile manufacturing industry, huge assembly lines are arranged and at each point, there are robotic arms to perform a particular task, and then the product moves on ahead to the next arm. Pipeline techniques are categories into 2 types. One is arithmetic pipeline, and the other is instruction pipeline.

- ✓ Arithmetic pipelinedesigned to act upon high-speed floating-point addition, multiplication and division. Multiple arithmetic logic units (ALUs) are built to perform the parallel arithmetic computation in various data formats in this Pipeline. Examples of arithmetic pipelined processors are Cray-1, Cyber-205, Star-100, TI-ASC.The floating point addition and subtraction is done in 4 parts: Compare the exponents, align the mantissas, add or subtract mantissas and produce the result.
- ✓ In instruction Pipeline, the number of instructions are pipelined, and the subsequent instruction execution overlaps the execution of current instruction. It is also known as instruction lookahead.

Space for learners:

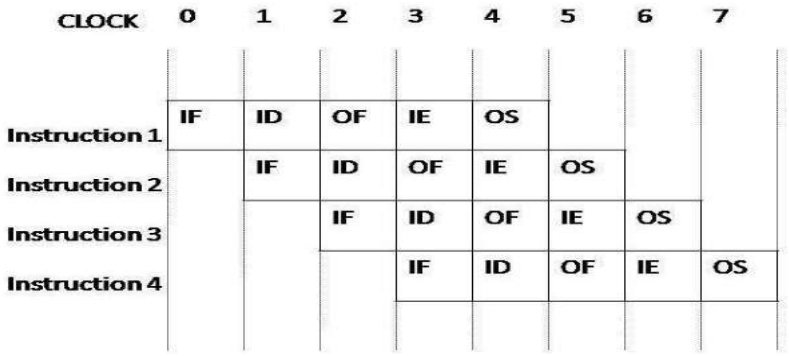


Figure 3.2: Pipelining of four Instructions

3.3.1 Register File

The register file is a hardware device with two read ports and one write port (corresponding to the two inputs and one output of the ALU). The register file and the ALU together comprise the two elements required to compute MIPS R-format ALU instructions. The register file is included of a set of registers that can be read or written by supplying a register number to be accessed, as well (in the case of write operations) as a write authorization bit. A block diagram of the register file is shown in Figure 3.3

Since reading of a register-stored value does not change the register state, no "safety mechanism" is needed to prevent inadvertent overwriting of stored data, and we need only supply the register number to obtain the data stored in that register. However, when writing to a register, we need:

1. A register number.
2. An authorization bit for safety (because the write operation overwrites the previous contents of the register selected for writing).
3. A clock pulse that controls the writing of data into the register.

Space for learners:

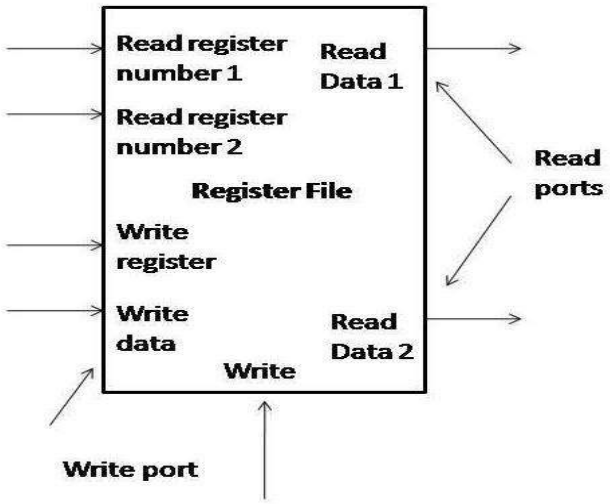


Figure 3.3. Register with two read ports and one write port,*adapted from [2]*.

Space for learners:

3.3.2 Datapath Design

The datapath is the "brawn" of a processor, since it implements the fetch-decode-execute cycle. The general discipline for datapath design is to

a. Determine the instruction classes and formats in the ISA,
b. Design datapath components and interconnections for each instruction class or format, and
c. Compose the datapath segments designed in Step 2) to yield a composite datapath.

Simple datapath components include *memory* (stores the current instruction), *PC* or program counter (stores the address of current instruction), and *ALU* (executes current instruction). The interconnection of these simple components to form a basic datapath is illustrated in Figure 3.4.

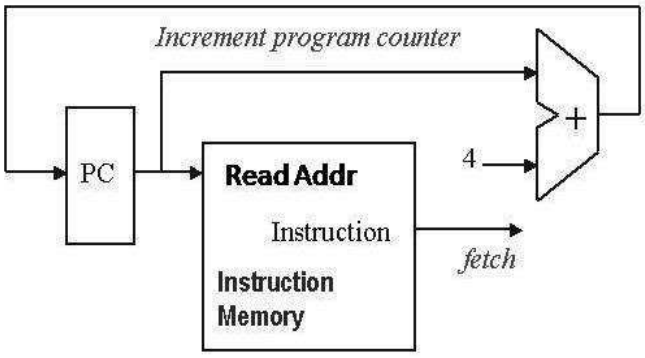


Figure 3.4: Schematic high-level diagram of MIPS datapath from an implementation perspective, *adapted from [2]*.

3.4. SUPER PIPELINING

Super pipelining is another approach to reach better (faster) performance. Super-pipelining is the breaking of stages of a given pipeline into more miniature stages(thus making the pipeline deeper) to shorten the clock period and thus to enhance the instruction throughput by keeping more and more instruction in flight at a time.

For example, if we divide each stage into two, the clock cycle period t will be reduced to half, t/2; hence, at the maximum capacity, the pipeline produces a result every t/2 s.

For a given architecture and the subsequent instruction set, there is an optimal number of pipeline stages; increasing the number of stages over this boundary decrease the overall performance. Superscalar architecture is a solution to further improve speed. Given a pipeline stage time T, it may be possible to execute at a higher rate by starting operations at intervals of T/n. This can be accomplished in two ways:

Space for learners:

Space for learners:

1. Further divide each of the pipeline stages into n sub stages.

This approach requires faster logic and the capability to subdivide the stages into segments with consistent latency. Here also Complex inter-stage interlocking and stall-restart logic required.

2. Make available n pipelines that are overlapped.

In this approach could be viewed in a sense as staggered superscalar operation, and has associated with it all of the same requirements except that instructions and data can be fetched with a slight offset in time.

Unavoidably, super pipelining is controlled by the speed of logic, and the frequency of unpredictable branches. The Stage time cannot effectively grow shorter than the interstage latch time, and accordingly this is a limit for the number of stages. The MIPS R4000 is sometimes called a super pipelined machine, although its 8 stages really only split the I-fetch and D-fetch stages of the pipe and add a tag check stage. Nevertheless, the further stages enable it to operate with higher throughput. The UltraSPARC's 9-stage pipe definitely qualifies it as a super pipelined machine, and in fact it is a Super-Super design because of its superscalar issue. The Pentium 4 splits the pipeline into 20 stages to enable increased clock rate. The benefit of such extensive pipelining is really only gained for very regular applications such as graphics. On more irregular applications, there is little performance advantage.

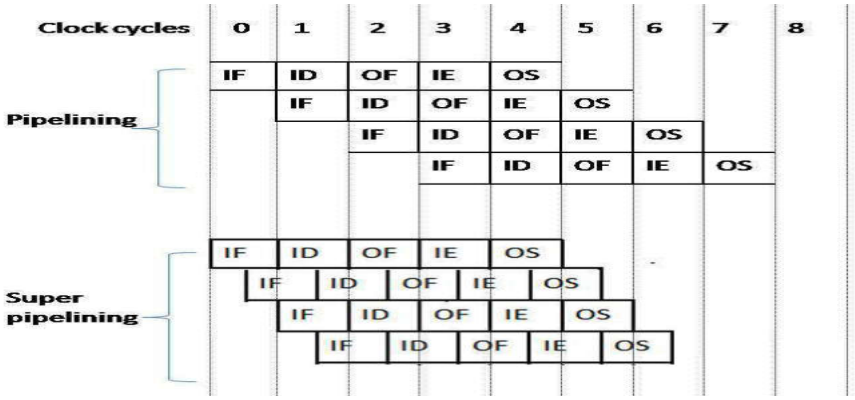


Figure 3.5: Comparison of normal pipeline and Super pipeline.

3.5 PERFORMANCE OF A PIPELINED PROCESSOR

Consider a K segment pipeline with clock cycle time as T_p and N tasks to be completed in the pipelined processor.

Here, the first instruction is about to take K cycles to come out of the Pipeline but the other $N–1$ instructions will take only one cycle each, i.e., a total of $N-1$ cycles. So, time is taken to N instructions in a pipelined processor:

$ET_P = K + N – 1$ cycles
$= (K + N – 1)\ T_P$
In the same case, the Execution time of N instructions in a non-pipelined processor, will be:
$ET_{NP} = N * K * T_P$.

Here $\mathbf{ET_P}$stand for estimate time taken in pipeline processor and $\mathbf{ET_{NP}}$stand for estimate time taken in non- pipeline processor.

therefore, *speedup(S)* of the pipelined processor over the non-pipelined processor, when N tasks are executed on the same processor is:

Space for learners:

$$S = \frac{\text{Performance of pipelined processor}}{\text{Performance of Non-pipelined processor}}$$

Since the performance of a processor is inversely proportional to the execution time, we have,

$S = ET_{NP} / ET_P$

$$\Rightarrow S = \frac{N * K * T_P}{(K + N - 1)\, T_P}$$

$$S = \frac{N * K}{(K + N - 1)}$$

We can ignore *(K-1)* When the number of tasks N are considerably larger than K, that is, $N >> K$

$$S = \frac{N * K}{N}$$

$S = K$, where K is the number of stages in the Pipeline.

Theoretically, maximum speedup ratio will be k where k are the total number of segments in which process is divided.

Again,

Efficiency = Given speed up / Max speed up $= S / S_{Max}$

We already know that $S_{Max} = K$

as a result, Efficiency $= \frac{S}{K}$

$$\textbf{Throughput} = \frac{\text{Number of instructions}}{\text{Total time to complete the instructions}}$$

hence, Throughput $= N / (K + N - 1) * T_P = N/T_P$

In ideal case as N -> 1 the throughout is equal to $1/ T_P$ that is equal to frequency. Thus

maximum throughput is obtained is there is one output per clock pulse.

Problem 1: A non-pipeline system takes 60 ns to process a task. The same task can be processed in six segment pipeline with a clock cycle of 10 ns. Determine the speedup ratio of the pipeline for 100 tasks. What is the maximum speed up that can be achieved?

Space for learners:

Solution:
Total time taken by for non pipeline to complete 100 task is
= 100 * 60 = 6000 ns
Total time taken by pipeline configuration to complete 100 task is
= (100 + 6 –1) *10 = 1050 ns
Thus speed up ratio will be = 6000 / 1050 = 4.76
The maximum speedup that can be achieved for this process is = 60 / 10 = 6
Thus, if total speed of non pipeline process is same as that of total time taken to complete a
process with pipeline than maximum speed up ratio is equal to number of segments.

Space for learners:

3.6 SUPERSCALAR ARCHITECTURE

Superscalar architecture is a system of parallel computing used in many processors together. In a superscalar computer, the central processing unit manages multiple instruction pipelines to execute several instructions concurrently during a clock cycle. It is achieved by feeding the different pipelines through several execution units within the processor. To successfully implement a superscalar architecture, the CPU's instruction fetching mechanism must intelligently retrieve and allot instructions. Otherwise, pipeline stalls may occur, resulting in execution units that are often inactive.

With each instruction that a superscalar processor issues, it must check the instruction's operands get in the way with the operands of any other instruction in flight. Once an instruction is independent of all other ones in flight, the hardware must also decide precisely when and on which available functional unit to execute the instruction. To fully utilize a superscalar processor of degree N must issue N instructions per cycle to execute in parallel at all times. This situation may not be accurate in every clock cycle. In that case, some of the pipelines may be stalling in a wait state. The simple operation latency should require only one cycle in a superscalar

processor, as in the base scalar processor. The superscalar processor depends more on an optimizing compiler to exploit parallelism to achieve a higher degree of instruction-level parallelism in program.

Space for learners:

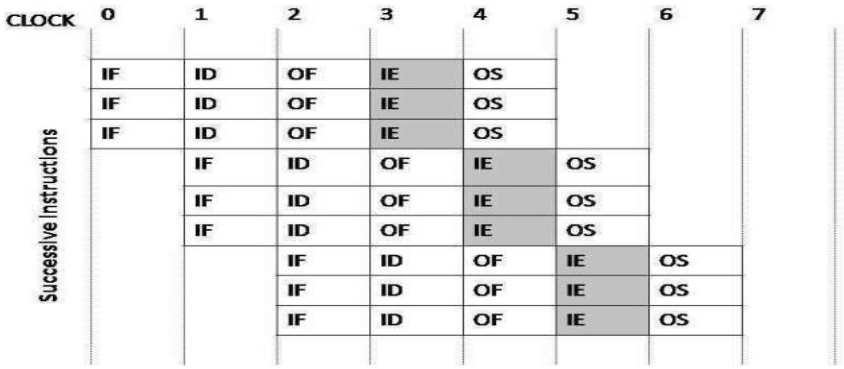


Figure 3.6: Pipeline structure of superscalar processor of degree-3

3.6.1 Structure Superscalar Architecture

Consider a machine organization capable of issuing more than one instruction per cycle depicted in Figure 3.7. Assume that the instruction set executed by the processor is $\mathbf{I = (I_1, I_2,.....I_N)}$ and that at most k instructions can be issued per cycle described by the k-tuple $\mathbf{P = (i_1, i_2.... i_k)}$, with $\mathbf{i_j \epsilon\ I,\ \ j = 1,2,....k}$. Furthermore, assume that at least k instructions are fetched into an instruction buffer and that a decision is reached on whether or not a k-instruction tuple can be issued and executed in parallel. This decision-making process is performed by the "Decode & Issue" logic. It is usually based on: the opcodes of the instructions, on availability of resources, and the structural and data dependencies.

Space for learners:

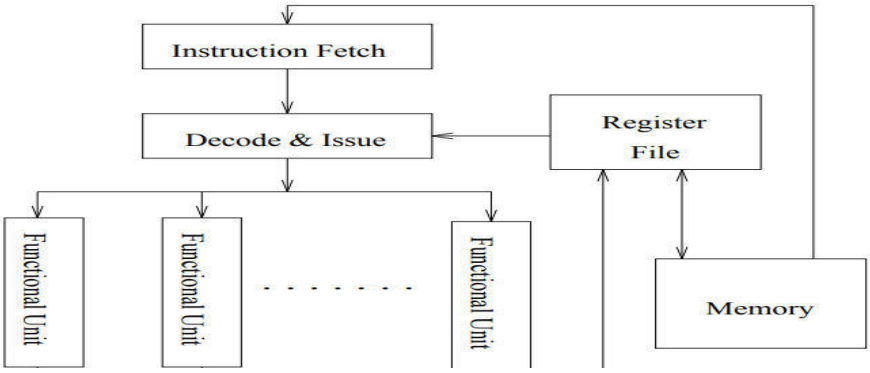


Figure 3.7: Basic Superscalar Architecture

We can classify superscalar processors into some classes of varying complexity.

1. Static Superscalar — In this processor issue and execute instructions in program order. So, for example, in a degree 3 machine, it is possible to issue and execute three instructions simultaneously: given instructions i1,i2 and i3, we may choose to issue all, or only i1 (depending on the presence of hazards). We may *not* just issue i2 or i3. The instruction issues look dynamic because the hardware has a choice about issuing instructions. However, as the actual execution of instructions is in order, we state that scheduling is *static*.

2. Dynamic Superscalar — These types of machines permit out-of-order program execution, but they usually still *issue* instructions in program order. Since we can potentially re-order the execution, so we now state scheduling is dynamic.

3. Dynamic with Speculation — These machines add the capability to speculate beyond branches, using different techniques.

3.6.2 Advantages of Superscalar Architecture

- The compiler can keep away from many hazards through well-judged choice and order of instructions.
- The compiler should make every effort to interleave floating-point instructions and integer instructions. This would facilitate the dispatch unit to maintain both the integer units and floating-point units active most of the time.
- On the whole, high performance is achieved if the compiler can arrange program instructions to take maximum assistance of the available hardware units.

3.6.3 Disadvantages of Superscalar Architecture

- In a Superscalar processor, the unfavorable effect on the performance of various hazards becomes even more pronounced.
- The problem in scheduling can occur because of this complex architecture.

CHECK YOUR PROGRESS-I

1. What is pipelining?
2. What are the 5 pipeline stages?
3. What is meant by ILP?
4. Define Superscalar processor.

3.7 BRANCH PREDICTION

The existence of program transfer instructions, e.g., JMP, RET, CALL, etc., can reduce the gain produced by Pipelining. These instructions change the sequence causing all the other instructions that entered the Pipeline after program transfer instructions are

Space for learners:

worthless. Thus no effort is done as the pipeline stages are reloaded.

keep away from this trouble, Pentium uses a scheme called Dynamic Branch Prediction. In this process, a prediction is prepared for the branch instruction currently within the Pipeline. The prediction will either be taken or not taken. If the prediction became true, then the Pipeline will not be flushed, and no clock cycles will be gone astray. If the prediction is false, then the Pipeline is flushed and starts over with the present instruction.

Mainly Branch Prediction predicts two problems one is Direction predicting and other one is calculating the target address.

3.7.1 Types of Branch Prediction

Basically there are two types of Branch prediction schemes :

1. Static branch schemes and
2. Dynamic branch schemes.

3.8. STATIC BRANCH SCHEME

A static branch scheme is a software-based technique which very simple and easy. This scheme assembles the more significant part of the data/information earlier to the program's execution or during the compile time and it does not need any hardware. In the Static branch prediction technique, underlying hardware assumes that either the branch is not always taken or the branch is always taken.

Let us understand branch prediction with an example code:

Space for learners:

```
//Code
int  number=0;
while(number <10)
 {
   //branch instruction, condition either true or false
   if(number%3= = 0)
        {
    Some statement ;
    }
   number=number++;
 }
```

Output:

Let us consider that underlying hardware has assumed that branch is taken constantly. The output predicted through underlying hardware, and actual output is shown in **figure3.8.**

Number	0	1	2	3	4	5	6	7	8	9
Actual Prediction	T	NT	NT	T	NT	NT	T	NT	NT	T
Static Prediction	T	T	T	T	T	T	T	T	T	T
	✔	×	×	✔	×	×	✔	×	×	✔

Figure 3.8: Prediction result of static branch prediction

3.9 DYNAMIC BRANCH SCHEME

A dynamic branch scheme is hardware-based technique based on the hardware and assembles the information during the program's run-time. Dynamic schemes are more assorted as they keep track during run-time of the program execution. In Dynamic branch prediction technique, prediction by underlying hardware is not rigid, rather it

Space for learners:

changes dynamically. The accuracy of this technique has high than the static technique.

Some of the dynamic branch prediction techniques are listed below:

a. 1-bit branch prediction technique
b. 2-bit branch prediction technique and
c. Correlating branch prediction technique

Space for learners:

3.9.1 1-Bitbranch prediction technique

In this technique hardware changes its assumption immediately after one false assumption. For instance if hardware assumes branch to be taken but in reality, branch is not taken, then after that step, hardware assumes branch to be not taken. Similarly, if hardware assumes branch not to be taken but in reality, branch is taken, then after that step, hardware assumes branch to be taken.

1-bit branch prediction Technique is shown in the Fig 3.8 below:

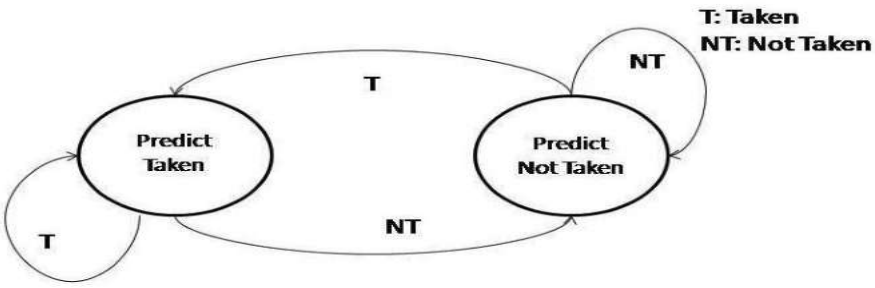


Figure3.9: Transition diagram of 1-bit prediction technique

Explanation –

In the beginning, let us declare hardware assume branch to be taken and so at number=0, branch is taken.
At number=1, hardware assumes branch to be taken but branch is not taken.

So now at number=2 hardware assumes branch not to be taken and also branch is not taken.

At number =3 hardware assumes branch not to be taken but branch is taken.

At number=4 hardware assumes branch to be taken but branch is not taken.

At number=5 hardware assumes branch not to be taken and branch is not taken.

In this way, the prediction is made till number=9.

The output predicted through underlying hardware, and actual output is shown in **Figure 3.9**:

Space for learners:

3.9.2 2-bit branch prediction technique

This predictor changes its earlier prediction only on two successive mispredictions occur and vice-versa. Two bits called as history Bit are maintained in the prediction buffer and there are 4-different states where Two states related to a taken state and two related to not taken state.

2-bit branch prediction technique is shown in the figure:

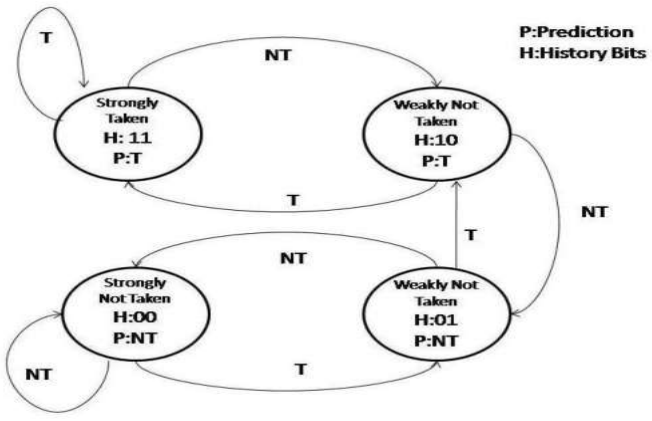


Figure 3.10: Transition diagram of 1-bit prediction technique

Explanation –

1. Let's assume that when number=0, everything is reset(00), so hardware assumes branch strongly not to be taken and the real branch is taken. As a result, the current state is (01).
2. When number =1, hardware assumes branch weakly not taken and in the real branch is not taken. Therefore the current state is (00).
3. When number =2, hardware assumes branch strongly not to be taken and branch is not taken in real. As a result, the current state remains (00).
4. When number =3, hardware assumes branch strongly not to be taken and in the real, branch is taken. As a result, the current state is (01).
5. When number =4, hardware assumes branch weakly not taken and in the real branch is not taken. So the current state is (00)
6. When number =5, hardware assumes branch strongly not to be taken and in the real branch is not taken. So the current state is (00). In this way, the prediction is done till number=9.

The comparison of the performance of Branch prediction schemes are given in the fig below:

Number	0	1	2	3	4	5	6	7	8	9
Actual Prediction	T	NT	NT	T	NT	NT	T	NT	NT	T
Static Prediction	T	T	T	T	T	T	T	T	T	T
	✔	×	×	✔	×	×	✔	×	×	✔
Output Predicted in 1-Bit Prediction	T	T	NT	NT	T	NT	NT	T	NT	NT
	✔	×	✔	×	×	✔	×	×	✔	×
Output Predicted in 2-Bit Prediction	NT 00	NT 01	NT 00	NT 00	NT 01	NT 00	NT 00	NT 01	NT 00	NT 00
	×	✔	✔	×	✔	✔	×	✔	✔	×

Figure 3.11: Comparison of performance of static,1-bit and 2-bit prediction scheme.

Space for learners:

3.9.3 Correlating Branch Prediction Technique

Due to interference with other branches, it is impossible to get significant accuracy from the 2-bit branch predictor. So correlating branch prediction comes into the picture which prediction accuracy is enhanced as it considers the recent activities of other branches. It uses *k* least significant bits of branch target addresswhich is fetched before. Also, it uses local history table (LHT), a table of shift registers where shift register refers to the last effect of m branches having similar k least significant bits. It also uses local prediction table(LPT) to predict the result depending on its present state.

CHECK YOUR PROGRESS-II

5. Define Branch prediction.
6. Define register file.
7. Define static branch prediction.
8. Define Dynamic branch prediction.

3.10 HAZARDS IN PIPELINING AND ITS TYPES

The situation that prevents the next instruction in the instruction stream from executing during its selected clock cycle is Hazard. Hazards decrease the performance from the ideal speedup gained by pipelining.

- **Structural Hazards**: Structural Hazards arises from resource conflicts when the hardware cannot support all possible combinations of instructions in the Pipeline requiring the same resource Due to some functional unit not being fully pipelined. Then the sequence of instructions using that un-pipelined unit cannot proceed at one per clock cycle rate. This generally isn't a problem with simple pipelines, but if some instructions take longer than

Space for learners:

others, this can become a problem. Another common way that it may appear is when some resources are not duplicated enough to allow all combinations of instructions in the Pipeline to execute. So by fully pipelining the stages and duplicating resources will avoid a structural Pipeline.

- **Data hazards**: Data hazards occur when instructions that show signs of data dependence modify data in different stages of a pipeline. This Hazard cause delays in the Pipeline., Data hazards occur when the Pipeline changes the order of read/write accesses to operands so that the order differs from the order seen by sequentially executing instructions on an un-pipelined processor. It can be minimized by a simple hardware technique called forwarding or by adding stalls.

  There are generally three types of data hazards:

  1) Read after Write (RAW)
  2) Write after Read (WAR)
  3) Write after Write (WAW)

  Let , there be two instructions I_1 and I_2, such that I_2 follow I_1.

  Then,

  - RAW Hazard occurs when instruction I_2 tries to read data before instruction I_1 writes it.
    E.g.,:

    I_1: R2 ← R1 + R3
    I_2: R4 ← R2 + R3

  - WAR hazard occurs when instruction I_2 tries to write data before instruction I_1 reads it.
    E.g.:
    I_1: R2 ← R1 + R3
    I_2: R3 ← R4 + R5

  - WAW hazard occurs when instruction I_2 tries to write output before instruction I_1 writes it.
    E.g.:

Space for learners:

I_1: R2 ← R1 + R3
I_2: R2 ← R4 + R5

WAR and WAW hazards occur during the out-of-order execution of the instructions.

Control hazards: It is caused by a delay between the fetching of instructions and decisions about changes in control flow (branches and jumps). Here instruction depends on the results of previous instruction in a way exposed by the overlapping of instructions in the Pipeline. Control Hazards are also known as Branch Hazards. The simplest method to handle branches hazard is to freeze or flush the Pipeline, holding or deleting any instructions after the branch until the branch destination is identified. In this case branch penalty is fixed and cannot be reduced by software. The other scheme is the predicted-not-taken or predicted-untaken and *delayed branch*. The number of stalls introduced during the branch operations in the pipelined processor is known as branch penalty.

3.11 DELAY SLOT

An instruction slot being executed devoid of the effects of a preceding instruction is known as a delay slot. The most familiar form is a particular arbitrary instruction located without delay after a branch instruction on a DSP or RISC architecture; this instruction will execute even if the prior branch is taken. In that way, by design, the instructions appear to execute in an incorrect or illogical order. It is usual for assemblers to automatically rearrange instructions by default, hiding the unease from assembly developers and compilers.[3]

Space for learners:

Load Delay Slot

In pipelined architecture, the load word instruction loads a word from memory to the specified register. The next instruction executes concurrently with the current instruction; if the following instruction uses the LW destination register as one of its source registers, it cannot continue before the LW data is fetched from memory and written back to the destination register; otherwise, it will read invalid data.

Branch Delay Slot

The branch delay slot is also a consequence of the branch hazard. A simple design would insert stalls into the Pipeline after a branch instruction until the new branch target address is computed and loaded into the program counter. Each cycle where a stall is inserted is considered one branch delay slot. A more sophisticated design would execute program instructions that are not dependent on the branch instruction. This optimization can be performed in software at compile time by moving instructions into branch delay slots in the in-memory instruction stream if the hardware supports this. Another side effect is that special handling is needed when managing breakpoints on instructions and stepping while debugging within branch delay slot[3].

Space for learners:

3.12 OUT-OF-ORDER EXECUTION

A processor that executes the instructions one after the other may use the resources inefficiently, leading to poor performance of the processor. To improve the performance of the processor, this can be done in two ways. One is by executing different sub-steps of sequential instructions simultaneously, and others execute the instructions entirely simultaneously. Additionally, improvement in the processor can be achieved through out-of-order execution by

executing the instruction in a diverse from the original order they appear out-of-order execution can be achieved. The approach of an out-of-orderexecution,used in high-performance microprocessors.

Here in this approach efficiently uses instruction cycles and minimized costly delay. As an alternative of the original order of the instructions, a processor will execute the instructions in an order of accessibility of data or operands. The processor will avoid being idle while data is retrieved for the next instruction in a program. In other words, a processor that uses multiple execution units completes the processing of instructions in the wrong order. For example, I_1 and I_2 are the two instructions where I_1 comes first, then I_2. In the out-of-order execution, a processor can execute I_2 instruction before I_1 instruction has been executed. This flexibility of allowing execution with less waiting time will improve the performance of the processor. The main benefit of the out-of-order processor is it avoids instruction waits when the data needs to perform an operation is unavailable.

STOP TO CONSIDER

Differentiate in-Order Execution from Out-of-Order execution.

Out-of-order execution is a situation in pipelined execution when an instruction is blocked from executing does not cause the following instructions to wait. It preserves the data flow order of the program.

In-order execution requires the instruction fetch and decode unit to issue instructions in order, which allows dependences to be tracked, and requires the commit unit to write results to registers and memory in program fetch order. This conservative mode is called in-order commit.

Space for learners:

CHECK YOUR PROGRESS-III

9. Define hazard and its types.

10. Define data hazard.

11. What is meant by delayed branch?

12. What is pipeline stall?

Space for learners:

3.13 REGISTER RENAMING

To deals with data dependences in pipelining between instructions by renaming their register operands is known as register renaming. *Renaming* replaces architectural register names by, in effect, value names with a new value name for each instruction destination operand. This process eliminates the name dependences (anti-dependences and output dependences) between instructions and automatically recognizes true dependences. An assembly language programmer or a compiler specifies these operands using *architectural registers* - the registers are explicit in the instruction set architecture.

The identification of true data dependences between instructions allows a more flexible life cycle for instructions. Maintaining a status bit for each value indicating whether or not it has been computed yet allows the execution phase of two instruction operations to be performed out of order when there are no true data dependences between them.

Being more explicit about the action precise by an instruction data dependences can be realized. The action specified by an instruction is more apparent if we illustrate instructions in terms of values rather than registers. We have to name the values in a manner that captures changes in register contents over time.

By replacing register names in all operands with value names, we can confine the intent of a sequence of instructions. For this, we use a table that records the value names assigned to each register name. Then we apply the following algorithm.

i. Replace each source operand with the most recent value name in the designated register column.
ii. Replace the destination operand with a new name and place the new name in the designated register column.

It is essential that **step i** is done first. Otherwise, when the same register is used both as a source operand and a destination operand, we indicate that the instruction execution phase cannot begin until its result is ready. This makes it impossible for the execution phase to begin.[4]

For an Example:

We will start with the following instructions.

MUL R6, R0, R2
DIV R4, R2, R0
ADD R0, R6, R2

Instruction	R0	R2	R4	R6	Renamed Instruction
Initial values	P0	P1	P2	P3	-----
MUL R6, R0, R2				P4	MUL P4, P0, P1
DIV R4, R2, R0			P5		DIV P5, P1, P0
ADD R0, R6, R2	P6				ADD P6,P4,P1

Space for learners:

3.13.1 Advantages of register renaming

The instructions with value names capture the intent of a sequence of instructions by specifying relationships between register values rather than just registers. This simplifies determining when the execution of an instruction can begin. We only need to check if its source operand values have been computed. The name dependencies no longer complicate the picture.

To determine when source operands have been computed, the value registers contain a status bit in addition to a data value. The status bit is initialized to 0 (not ready) when the value register is allocated for an instruction. It is set to 1 when a functional unit writes a result.[4]

Space for learners:

3.14 SUMMING UP

In this unit, efforts have been made to acquaint you with the advanced concept of computer architecture implicit parallelism. Here in this unit, you learn the basic pipeline structure. Pipeline techniques are categories into two types. One is arithmetic pipeline, and the other is instruction pipeline. Super-pipelining is the breaking of stages of a given pipeline into more miniature stages to shorten the clock period and thus to enhance the instruction throughput by keeping more and more instruction in flight at a time. Superscalar architecture is a system of parallel computing used in many processors together. Here the central processing unit manages multiple instruction pipelines to execute several instructions concurrently during a clock cycle. Here in this unit you learned the different branch prediction techniques. In the static branch prediction technique, underlying hardware assumes that either the branch is not always taken or the branch is always taken. A dynamic

branch scheme is hardware-based technique based on the hardware and assembles the information during the program's run-time. The comparison of static prediction,1- bit branch prediction and 2-bit branchprediction also elaborate with an example. Hazard is the situation that prevents the next instruction in the instruction stream from executing during its selected clock cycle. An instruction slot being executed devoid of the effects of a preceding instruction is known as a delay slot. Here we discuss the out-of-order execution that avoids instruction waits when the data needs to perform an unavailable operation. At the last of the unit you learned the register renamingprocess, which deals with data dependences.

Space for learners:

3.15 KEY TERM

PIPELINING: Pipelining is the practice of accumulating instruction from the processor through a pipeline. In Pipelining, Storing and executing of the instructions allows being in an orderly process.

SUPER PIPELINING :Super pipelining is another approach to reach better performance. Super-pipelining is the breaking of stages of a given pipeline into more miniature stages to shorten the clock period and thus to enhance the instruction throughput by keeping more and more instruction in flight at a time.

SUPERSCALAR ARCHITECTURE : Superscalar architecture is a system of parallel computing used in many processors together. In a superscalar computer, the central processing unit manages multiple instruction pipelines to execute several instructions concurrently during a clock cycle.

STATIC BRANCH PREDICTION : This is the simplest branch prediction technique because it predicts the outcome of a branch based solely on the branch instruction. It does not rely on information about the dynamic history of code executing.

DYNAMIC BRANCH PREDICTION : This is hardware-based technique based on the hardware and assembles the information during the program's run-time.

HAZARD :The situation that prevents the next instruction in the instruction stream from executing during its selected clock cycle is Hazard.

STALL*:* It is a delay in execution of an instruction in order to resolve a hazard.

BRANCH PENALTY: The number of stalls introduced during the branch operations in the pipelined processor is known as branch penalty.

DELAY SLOT: An instruction slot being executed devoid of the effects of a preceding instruction.

REGISTER RENAMING: Register Renaming is a process to deals with data dependences in pipelining between instructions by renaming their register operands.

Space for learners:

3.16 ANSWERS TO CHECK YOUR PROGRESS

1. "Pipelining, also known as "pipeline processing", is the process of collecting instruction from the processor through a pipeline. It stores and executes instructions in an orderly process."

2. The 5 stages of instruction execution in a pipelined processor are:
   a. Instruction Fetch (IF)
   b. Instruction Decode(ID)
   c. Operand Fetch (OF)

d. Instruction Execute (IE)
e. Operand Store (OS)

3. Pipelining exploits the potential parallelism among instructions. This parallelism is called instruction-level parallelism (ILP). There are two primary methods for increasing the potential amount of instruction-level parallelism. a. Increasing the depth of the pipeline to overlap more instructions. b. Multiple issue.

4. There are processors which are capable of achieving an instruction executing throughput of more than one instruction per cycle. They are known superscalar processor.

5. It is a technique for reducing the branch penalty associated with conditional branches is to attempt to predict whether or not a particular branch will be taken.

6. The processor‘s 32 general-purpose registers are stored in a structure called a register file. A register file is a collection of registers in which any register can be read or written by specifying the number of the register in the file. The register file contains the register state of the computer.

7. The branch prediction decision is always the same every time a given instruction is executed. Any approach that has this characteristic is called static branch prediction.

8. The branch prediction where decision may change depending on execution history is called Dynamic branch prediction.

9. Any condition that causes the pipeline to stall is called a hazard. Its types are:

a. Data hazard
b. Instruction hazard
c. Structural hazard

Space for learners:

10. A data hazard is any condition in which either the source or the destination operands of an instruction are not available at the time expected in the pipeline. A data hazard is a situation in which the pipeline is stalled because the data to be operated on are delayed for some reason.

11. Delayed branch is a type of branch where the instruction immediately following the branch is always executed, independent of whether the branch condition is true or false.

12. Pipeline stall, also called bubble, is a stall initiated in order to resolve a hazard. They can be seen elsewhere in the pipeline.

Space for learners:

3.17 POSSIBLE QUESTIONS

Multiple Choice Questions:

1. Arithmetic Pipeline is used for
   a. floating point operations
   b. integer operations
   c. character operations
   d. None of the above

2. Which of the following is not a Pipeline Conflicts
   a. Timing Variations
   b. Branching
   c. Load Balancing
   d. Data Dependency

3. How many types of Pipelining exist
   a. 2
   b. 3
   c. 4
   d. 5

4. Which of the following is disadvantage of Pipelining
   a. cycle time of the processor is reduced.
   b. The design of pipelined processor is complex and costly to manufacture.

c. The instruction latency is more.
d. Both b and c

5. Which of the following is an advantage of pipelining
   a. Instruction throughput increases.
   b. Faster ALU can be designed when pipelining is used.
   c. Pipelining increases the overall performance of the CPU.
   d. All of the above

6. In arithmetic pipeline, the floating point addition and subtraction is done in _____ parts.
   a. 2
   b. 3
   c. 4
   d. 5
7. _____ have been developed specifically for pipelined systems.
   a. Utility software
   b. Speed up utilities
   c. Optimizing compilers
   d. None of the above

8. The pipelining process is also called as _____
   a. Assembly line operation
   b. Von Neumann cycle
   c. Superscalar operation
   d. None of the above
9. Each stage in pipelining should be completed within _____ cycle.
   a. 1
   b. 2
   c. 3
   d. 4
10. The periods of time when the unit is idle is called as _____
    a) Stalls
    b) Bubbles
    c) Hazards
    d) Both Stalls and Bubbles

Answers (1. a, 2. c, 3. a, 4. d, 5.d, 6. c, 7.c, 8.a, 9.a, 10.d)

Space for learners:

Fill in the blanks:

1. The situation wherein the data of operands are not available is called _____.
2. The contention for the usage of a hardware device is called _____
3. Each stage in pipelining should be completed within _____ .
4. The fetch and execution cycles are interleaved with the help of _____.
5. The pipelining process is also called as _____ .
6. A pipeline _____ is a delay in execution of an instruction in order to resolve a hazard.
7. The number of stalls introduced during the branch operations in the pipelined processor is known as _____ .

Answers:

1. Data hazard 2. Structural hazard 3. 1 cycle 4. clock 5. Assembly line operation 6. Stall 7. branch penalty

Short answer type questions:

1. What do you mean by implicit parallelism?
2. Write a brief note on pipelining.
3. Explain Basic structure of pipelining technique.
4. Write short notes on arithmetic pipeline and instruction pipeline.
5. How do you calculate performance of a pipeline.
6. Short note on super pipelining techniques.
7. Differentiate between normal pipeline and super pipeline.
8. Compare super pipeline with superscalar architecture.
9. What are the advantages and disadvantages of superscalar architecture?
10. write down different type of superscalar processor.
11. Why do we require branch prediction?

Space for learners:

12. What are the types of branch prediction scheme?
13. What is static branch prediction?
14. What is dynamic branch prediction?
15. What are the different types of dynamic branch prediction?
16. Write a short note on correlating branch prediction.
17. What do you mean by hazards in pipeline ?
18. What are the different types of hazard in the pipeline?
19. What do you mean by delay slot.
20. What is the need of register renaming?

Long answer type questions:

1. Explain pipeline structure with diagram.
2. How many sub-tasks of instruction are there in pipeline. Explain.
3. Explain super pipeline technique. What are the benefit over normal pipeline.
4. Explain basic structure of superscalar architecture.
5. Explain 1-bit branch and 2-bit branch prediction technique with example.
6. Explain different type of hazards occurs in pipeline.
7. Explain data hazard with their types.
8. Explain different delay slots in pipeline.
9. Describe out-of-order execution.
10. What is register renaming? Explain how register renaming is done.

Space for learners:

3.18 REFERENCES AND SUHHESTED READINGS

[1]Pipelining. cs.siu.edu/~cs401/Textbook/ch3.pdf

[2]*Course Notes,* Mafla, E. *CDA3101*, at cise.ufl.edu/~emafla/

[3] Delay slot - WikiMili, The Best Wikipedia Reader. wikimili.com/en/Delay_slot

[4] Register Renaming ,University of Minnesota, d.umn.edu/~gshute/arch/register-renaming.html

[5] "Advanced Computer Architecture" Hwang ,Publisher Tata McGraw-Hill Education, 2003 ISBN:007053070X, 9780070530706

[6] "Computer Organization and Design – The Hardware / Software Interface", David A. Patterson and John L. Hennessy, 4th.Edition, Morgan Kaufmann, Elsevier, 2009.

[7] "Computer system Architecture", Mano, M. Morish, 3rd Edition, Pearson Education,1993

---×---

Space for learners:

UNIT 4: ADVANCED CONCEPTS OF PIPELINING SCHEDULE

Unit Structure:

Space for learners:

Space for learners:

4.1. INTRODUCTION

In this unit, you will get to learn in detail about pipelining scheduling which is a very important concept of parallelism in computer organization and architecture (COA). As you already know that in pipelining, the instructions are accumulated through a pipeline from the processor. Many instructions are overlapped with each other. Performances of the CPU are improved due to the use of pilelines. So we will discuss the main concepts of pipelining through the dynamic scheduling approaches.

After going through the chapter, you will get to learn some of the important concepts of pipelining scheduling such as

- **DATA DEPENDENCY** – Data dependency is a concept that is applied to check whether a block works properly even if the instructions present in that block are rearranged.
- **SCOREBOARD** – When the data dependencies are not present and when sufficient resources are present in the system, the score-boarding technique allows the execution of out-of-order performances.
- **SLACK TIME** – When the time of a process gets delayed without other processes getting delayed, is termed slack time.

- **RATE MONOTONIC** – Rate monotonic (RM)is a type of static scheduling algorithm in which the instructions that have the smallest job or rate are given more priority than the bigger jobs.

Space for learners:

4.2. UNIT OBJECTIVES

Studying this unit, you will be able to:

- Understand the concept of pipelining and pipelining scheduling.
- Discuss the different data structures related to pipelining processes.
- Know the different dynamic and static scheduling algorithms.

4.3. PIPELINING

Pipelining is a technique where instruction overlapping occurs at the time of execution. Instructions are accumulated from a processor into the input registers through a pipeline, and therefore this process is known as pipelining. The order in which the instructions are stored and executed is defined as the pipelining processing [1]. Different stages are linked together to form a single-stage pipeline and the instructions enter through one end of the pipeline and come out through the other end. Each stage of the pipeline consists of some input registers which hold the instructions at every stage and are then operated by some combinational circuit. When a combinational circuit works on a register, the output of it is shifted to the next registers present in the lined-up segments. All the instruction inside the pipeline works concerning some clock time[1].

4.3.1. Types of Pipeline

Since instructions encountered in the pipeline are of different types, so to cope up with this situation, the pipeline is split into two types. They are Arithmetic Pipelining and Instruction Pipelining, which are explained as follows [1]:

4.3.1.1. Arithmetic Pipelining

When arithmetic operations come as instruction into the pipeline they are then stored in the Arithmetic Pipeline. Arithmetic operations may include addition, subtraction, operations on floating-point numbers, etc. [1].

4.3.1.2. Instruction Pipelining

Instruction pipelining helps in increasing the throughput of the system. Fetch, execute and decode instructions are overlapped in the instruction cycle. When a new instruction is present in the memory then it is read by an instruction pipeline, and the instructions that were already existing are executed in the segments present in the pipeline. The efficiency of the pipeline will increase if the instruction cycle is split into the equal time clock. By doing this we execute multiple instructions simultaneously. That is, we can say that parallel processing occurs in the pipeline thus increasing the efficiency of the system along with an increasing throughput [1].

CHECK YOUR PROGRESS

1) What do you understand by pipelining?
2) What do you mean by pipelining schedule?
3) What is arithmetic pipelining?
4) What is instruction pipelining?
5) Fill in the blanks:

a) When the time of a process gets delayed without other processes actually getting delayed, is termed as the _____.

b) Each stage of the pipeline consists of some _____ which holds the instructions at every stage and then operated by some _____ circuit.

Space for learners:

4.4. PIPELINING PROCESSOR

Depending upon the work it follows, the pipelining processors are divided into two categories, one is the scalar processor and the other is the vector(array) processor[1].

4.4.1. Scalar Processor

The simple processor which executes one instruction at a time and that too simple instructions are known as the scalar processor. But as it works on single instruction at a time therefore it proves to be an inefficient processor. The speed of the processor is also very slow.

For example, we need to add two numbers and store the answer in the third location which requires only simple calculation[1].

ADD B, D and store it in E.

4.4.2. Vector (Array) Processor

When complex instructions are executed on numerous data synchronously, then a vector processor is used. This processor executes the instructions very fast as compared to the scalar processor and has much efficiency.

In the Instruction pipelining, at a particular time, different works are performed by the processor on the different data. Vector processor uses the instruction pipeline for data processing. Here the CPU remains busy all the time[1].

4.5. ADVANTAGES OF PIPELINING [1]

- Using pipelining, the total time of the processor's instruction cycle gets reduced thereby increasing the throughput of the instruction. In an actual case, multiple instructions are executed simultaneously and it looks like that the total time gets reduced.

Space for learners:

- The time delay in between two instructions is greatly reduced hence increasing the throughput.
- Nowadays for a faster and more complex design Arithmetic and Logic Unit, the pipeline is developed into several stages.
- Performance of the pipeline increases, meaning the clock cycle also increases.
- The speed at which the clock cycle of the RAM works is much lower than the clock cycle of the pipelining processor hence increasing the performance.

Space for learners:

4.6. DISADVANTAGES OF PIPELINING [1]

- Branching delay can occur in a pipelined processor. For reducing this branching delay address of the target branch need to be pre-fetched at the stage of decoding. Doing this the delay occurred may be reduced until 1 clock cycle.
- The flip-flops that are inserted between the data modules increase the latency in the instructions.
- In pipelined processing, you may get some unexpected performances.
- When there are many branches in the stages of the pipeline, then the throughput gets reduced.
- Memory delay can occur in the pipelined processor. Cache miss occurs when searched data or instructions are not present in the cache memory and therefore searched in the main memory which then consumes more number of the instruction cycle. This is known as the Memory delay which becomes the reason for the delay for the other data or instructions that are lined up.
- When the pipeline does not validate the assumptions of the instructions, then incorrect behavior of the program might occur, which leads to hazards.

4.7. PIPELINING SCHEDULING

Space for learners:

Pipelining scheduling is a type of mechanism where executions are overlapped for different inputs and the computations are performed at different stages. It improves the performances of the machine that have parallel instructions usually termed as instruction pipelines. Let us explain this pipelining scheduling with the help of the following example. Suppose you have to manufacture a washing machine by developing two models[2].

a) **For model 1**, suppose you have designed the washing machine in such a way it washes (W), dries (D), and iron (I) one cloth at a time (T). That is, for performing the mentioned operations on(n) number of clothes, the time required would be **(n.T)**.

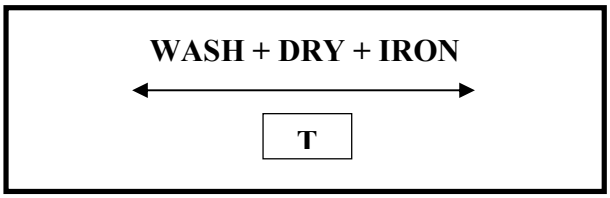


Figure 1: Model 1 for pipelining example

b) **For model 2**, suppose you have split the work of one washing machine into different machines that can wash (W), dry (D), and iron (I) the clothes separately. For each separate machine, the mentioned work is performed on more than one number of clothes in time (T). Now the time, that is required by **each** machine to perform the above task (for 1 cloth at a time) will be (T/3)[2]. And the time required for performing the operations on (n) number of clothes will be

$$\{T_3 = (2 + n).\ T/3\}.$$

For a larger number of clothes, (2 + n) will become 'n'. Then the time required will be

$$\{T_3 = n.\ T/3\}$$

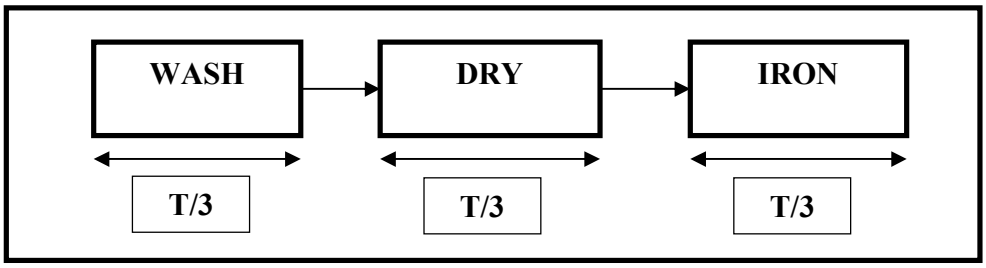


Figure 2: Model 2 for pipelining example

Here, model 2 explains the pipelining process. Let us explain this as follows[2].

- For time T/3, the cloth is washed in the machine.
- After it finishes the process of washing, it enters the second stage that is the dry stage, and works there for time T/3. When the second cloth was in the dry stage another cloth has entered the washing stage and took the same time T/3. It is being pipelined.
- When the first cloth entered the iron stage for time T/3, the second cloth is in the dry stage for time T/3 (it is being pipelined) and the third cloth is in the washing stage for time T/3 (it is being pipelined).
- Then cloth one finishes its task, cloth two is pipelined and enters into the iron stage. Meanwhile, cloth 3 enters into the dry stage and new cloth 4 enters into the washing stage.
- Simultaneously all the machines are working having each time T/3.
- In this way, all the machines are working keeping themselves busy without remaining idle.
- By keeping in mind the formula (**T_3 = (2 + n). T/3), where n is several clothes.**
- We can say that Cloth one took (for all the three stages) time

  T = (2 + 1) T/3 = **3T/3**

- Cloth two took time; T = (2 + 2) T/3 = **4T/3**
- Cloth three took time; T = (2 + 3) T/3 = **5T/3**and so on.

This explains the pipelining process.

Space for learners:

4.7.1. Data Dependency

Data dependency is a concept that is applied to check whether a block works properly even if the instructions present in that block are rearranged. As said, there are three types of data dependencies[4].

- **Read After Write (RAW)** – At first, suppose Instruction 1 writes an instruction. That same instruction is read by Instruction 2 later. After Instruction 1 writes the value, then only Instruction 2 will read, therefore Instruction 1 must be written first otherwise instead of reading the new value, Instruction 2 will read the old value.
- **Write After Read (WAR)** – At first, the location of a value is read by Instruction 1. After that Instruction 2 again rewrites the value. Instruction 1 must be written first, otherwise, instead of reading the new value, Instruction 2 will read the old value.
- **Write After Write (WAW)** – Both Instruction 1 and 2 when write a value in the same location, then this dependency is termed as write after write and it must be in the same order as the original order.

CHECK YOUR PROGRESS

1) What is a scalar processor?
2) What is a vector processor?
3) State the advantages of pipelining.
4) State the disadvantages of pipelining.
5) What are the different data dependencies?

4.8. DYNAMIC SCHEDULING

At the time of compilation, sometimes some dependencies occur in the system and we are unable to recognize these dependencies. In this case, handling of the dependencies is performed by the dynamic scheduler and hence the process is termed dynamic scheduling. For instructions having simple pipelining techniques, the major drawback is that all instructions are scheduled in some order, and once the instructions are

Space for learners:

pipelined then no new instructions or instructions after the scheduled instructions can be executed earlier than the pipelined instructions. If two or more instructions are spaced closely and have the same dependencies, then it might so happen that the instructions might come to a halt or become idle[3]. When hardware is taken into account, dynamic scheduling comes into force.

Space for learners:

4.8.1. Out of Order Completion

The WAR and WAW hazards create the possibility of out-of-order execution. For handling the exceptions, major complications are created by the out-of-order completion. There are two possible cases where non-precise exceptions might occur[3].

- Suppose there are many instructions in a pipeline and it may so happen that one instruction present in the pipeline can cause exceptions. The possibility of a non-precise exception might occur when instructions that are present after the 'exception instruction' have been executed first[3].
- Another possibility might occur where there are many instructions present in a pipeline and it may so happen that one instruction present in the pipeline can cause exceptions. The possibility of a non-precise exception might occur when some instructions present in the pipeline before the 'exception instruction' are not executed at all[3].

Execution of out-of-order is allowed if the five stages pipeline is transformed into two stages in the following ways[3].

- Issue – Instruction decoding and to check whether any structural hazards are present in the pipeline or not.
- Read operands – The pipeline will wait till no data hazards are encountered and then the operands will be read.

For the dynamic scheduling, the instruction in the pipeline should pass in an ordered way through the Issue stage and then into the read operands, which is the second stage.

4.8.2. Dynamic Scheduling Algorithms

As the name suggests, a dynamic scheduler helps in making efficient decisions during the runtime of the system. Therefore, the system that works on dynamic scheduling is more flexible but at the same time calculation overhead also occurs. It checks which instruction has the most priority than the other and simultaneously works on that instruction at first. As it takes instruction during the runtime therefore the priority of executing the instruction might also change accordingly[5]. There are many dynamic scheduling algorithms based on different approaches, some of them are discussed below.

Space for learners:

4.8.2.1. Earliest Deadline First (EDF)

EDF is a type of dynamic scheduling algorithm in which the instructions that have the nearest deadline to complete are given the task of highest priority and are executed first. When the current process gets completed and new processes are scheduled then this algorithm is worked upon. It is applicable for real-time systems. The CPU is utilized fully making sure that all the tasks are completed. An optimal feasible schedule is processed where all the tasks are executed within the stipulated deadline. The task must mention its deadline once it is made ready for execution and given a fixed CPU burst timing. Preemption can occur in EDF, and any instances that are scheduled for later but are engaged with an earlier deadline get ready for execution and becomes active[5].

But there are some limitations of the Earliest Deadline First Algorithm such as

- Overloading problems for the transient might occur.
- There might be some problems when resources are shared.
- Sometimes implementations are not done efficiently.

Let us explain the EDF algorithm with the help of an example by taking a flowchart[5].

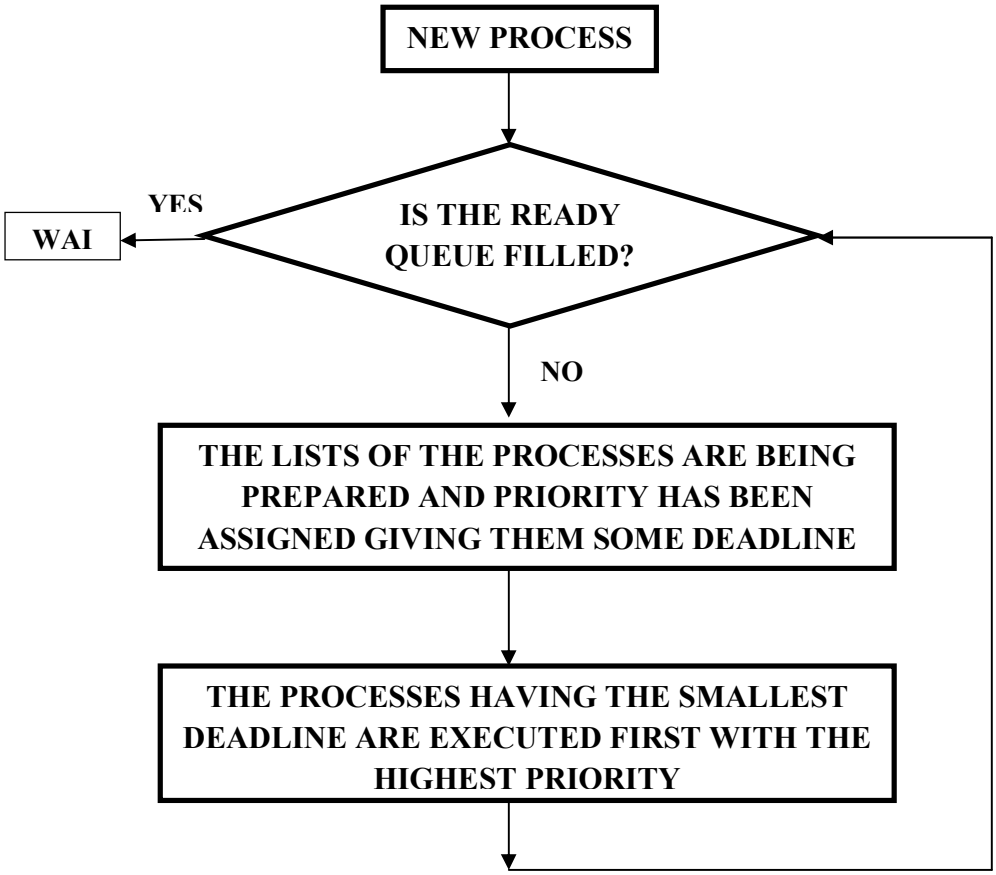


Figure 3: Flowchart of the Earliest Deadline First Algorithm

Space for learners:

4.8.2.2. Least Slack Time First (LST)

LST is a type of dynamic scheduling algorithm in which the instructions that have the smallest slack time are given the task of highest priority and are executed first. When the time of a process gets delayed without other processes getting delayed, is termed as the slack time. Like that of the EDF, when the current process that has the lowest slack time gets completed and new processes are scheduled then this algorithm is worked upon. For the slack time to be given as *l,* starting time is given as *t,* deadline interval is given to be *d,* and the remaining execution time is given to be *c*, the formula is given as ($l = d - c - t$) [5]. The algorithm is somewhat complex therefore requires extra information like the deadline and the execution timing. In real-time systems, it is sometimes difficult to predict the burst time. If the processes have the same slack time, then first cum first serve (FCFS) algorithm is applied together with LST.

Let us explain the LST algorithm with the help of an example by taking a flowchart [5]

Space for learners:

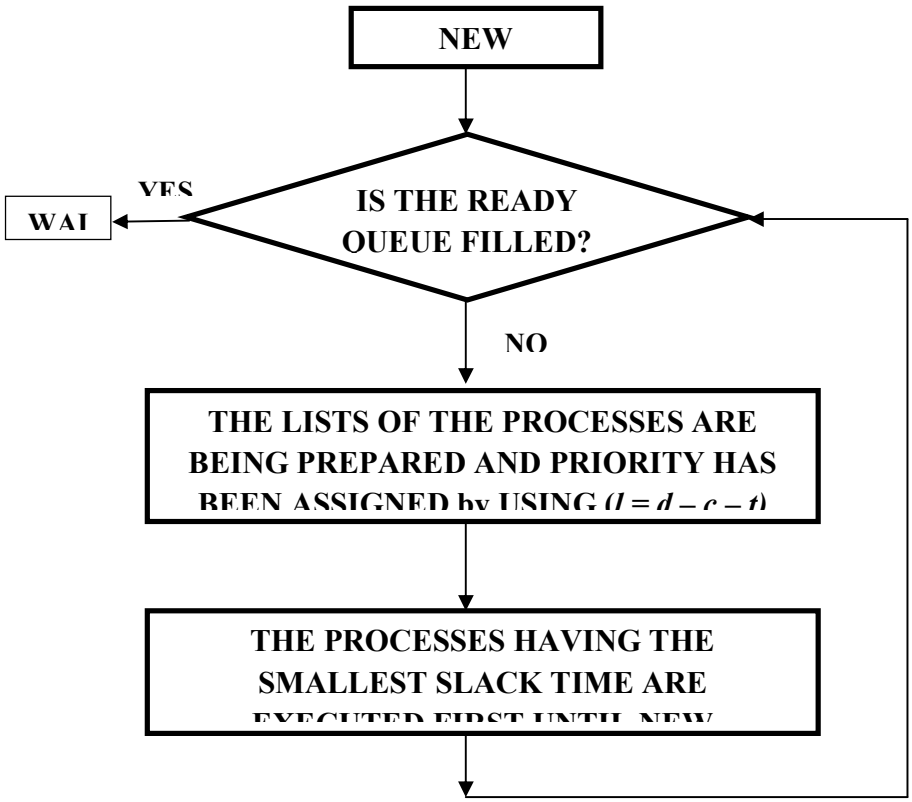


Figure 4: Flowchart of the Least Slack Time Algorithm.

4.8.3. Advantages of Dynamic Scheduling[6]

- Unknown dependencies during compile time are handled by dynamic scheduling because memory references are included.
- It simplifies the performance of the compiler.
- Codes on a pipeline are compiled so efficiently that they can run on different pipelines.
- Hardware speculations are often built on dynamic scheduling.

4.8.4. Disadvantages of Dynamic Scheduling[7]

- The complexities of the hardware increase substantially.
- Dynamic scheduling surely complicates exception handling.
- WAW and the WAR dependencies are created for out-of-order execution as well as out-of-order completion.

4.9. STATIC SCHEDULING

In static scheduling, all the processes are fixed for a particular stage in the pipeline. Before the execution takes place, the processes are given the tasks. They are usually processor non-preemptive. The overall time of the execution is minimized by the static algorithm. It tries to indicate the behavior of the execution of the program such as the execution time, process, and communication delays during the compile time. The smaller tasks are partitioned for reducing the communication costs. Processes are allocated to the processors. Static scheduling has a more efficient execution time environment as compared to the dynamic scheduling algorithm [5].

4.9.1. Static Scheduling Algorithms

Just like the dynamic scheduler, the priority scheduler works on the tasks that have more priority than the other but the value of the priority does not change. The static scheduler can make an efficient decision before runtime as well. There are many static scheduling algorithms, some of them are discussed below[5].

4.9.1.1. The Rate Monotonic (RM)

RM is a type of static scheduling algorithm in which the instructions that have the smallest job or rate are given more priority than the bigger jobs. The size or rate of the job is already scheduled in the RTOS. When the current process that has the smallest job gets completed and new processes are scheduled then this algorithm is worked upon[5]. The priorities are assigned just before the execution and remain the same throughout its execution period. Rate monotonic works based on the preemption, that is, during the execution time, if a shorter job is

Space for learners:

encountered by the system, then that job is given more priority for the execution. A job that has more time period has less priority and a job that has a lesser time period have more priority. The implementation of it is very much easy.

Let us explain the rate monotonic algorithm with the help of an example by taking a flowchart [5]

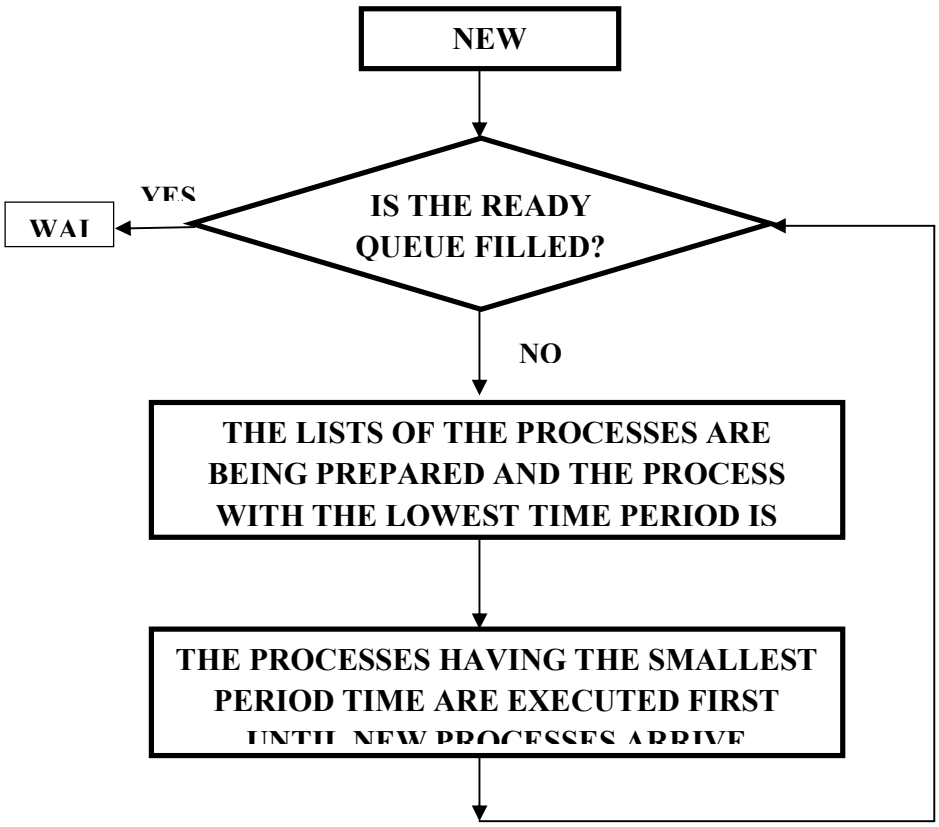


Figure 5: Flowchart of the Rate Monotonic algorithm

4.9.1.2. The Shortest Job First (SJF)

SJF is a type of static algorithm in which the instructions that have the smallest execution time are executed first. The time of the job is already scheduled in the RTOS. It is kept as the CPU time. When the current process that has the smallest job time gets completed and new processes

Space for learners:

are scheduled then this algorithm is worked upon[5]. This algorithm is suitable for a processor having batch-type processing and the waiting time for the jobs is not critical. SJF can be applied in both preemptive and non-preemptive scheduling algorithms. Starvation of the processes might occur if the processes have a larger burst time.

Let us explain the SJF algorithm with the help of an example by taking a flowchart [5].

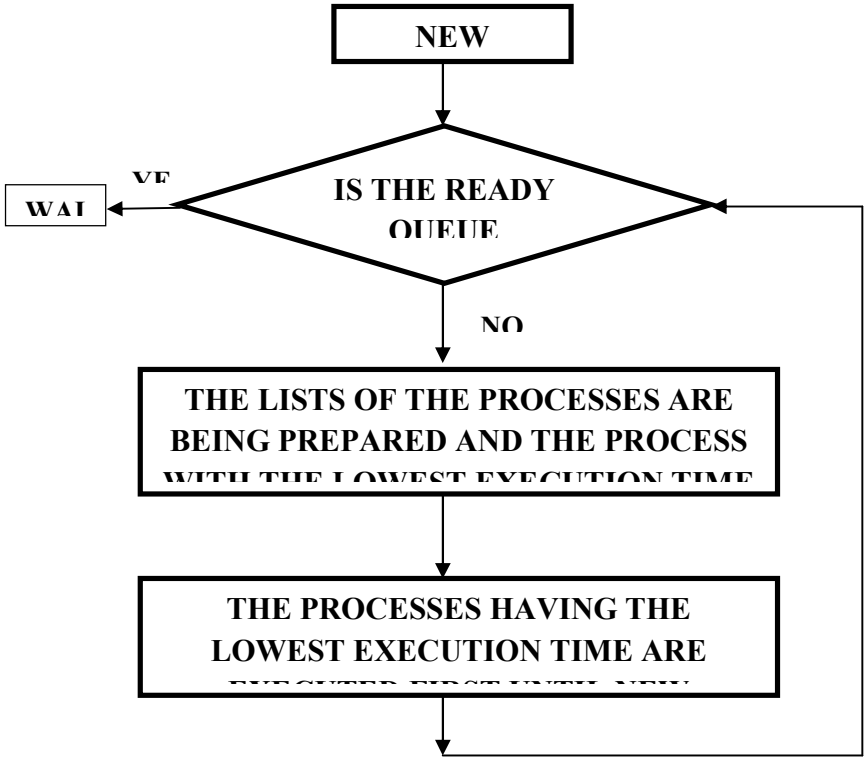


Figure 6: Flowchart of the Shortest Job First algorithm

CHECK YOUR PROGRESS

1) What is out of order completion?
2) Describe the different dynamic scheduling algorithms.
3) Describe the different static scheduling algorithms.
4) What are the advantages of dynamic scheduling?
5) What are the disadvantages of dynamic scheduling?

Space for learners:

4.10. TOMASULO'S ALGORITHM

Space for learners:

A scientist named Robert Tomasulo invented the Tomasulo Algorithm to be used in IBM 360/91.Tomasulo's algorithm is a type of hardware algorithm in computer architecture that isused for implementing dynamic scheduling allowing out-of-order execution and enabling multiple execution units. The hardware includes the reservation stations, the register renaming, and a common data bus (CDB) for carrying the values towards the reservation stations. Because of the presence of this hardware architecture, parallel processing is possible. WAR and WAW hazards are removed using register renaming. And this register renaming is done through reservation stations. Register Renaming is implemented through reservation stations. Tomasulo's algorithm work in some stages which are discussed below [8]. Here reservation station provides the register renaming.

1) **The Issue stage** – Instructions are present in a queue (FIFO queue) in which all the instructions are given some space with some deadline. When one instruction completes its job, the next instruction remains at the head of the queue. The work of the Issue stage is to call the instruction from the queue that is present at the head of the queue. If the reservation station matches the called instruction, then the instruction is issued some operand values in the renaming register [3]. One condition is that; the reservation station must remain free when the instruction is called. If the reservation station is not free, then the instruction stalls, and subsequently structural hazards occur. If there is a problem in issuing one instruction, then the instructions that are lined up in the queue will not get executed. Another case may occur; the instruction waits in the reservation stations if the values of the operands are not found from the common data bus [9].

2) **The Execution stage** – If all the values of the operands are available by the CDB into the reservation stations, then execution of the processes takes place. Until and unless operand values are not available, execution does not occur [3].With the help of effective addressing, load and store are

maintained.Executions are not initiated by the instructions until and unless the previous instructions are completed that were in order [9].

3) **The Write Result stage**– Once the results are obtained through execution, the result is written on the CDB and then transferred into the registers and then into the reservation stations that contain the store buffers. At this write result step, the data is written into the memory. As soon as the data values and addresses are present the data is transferred to the memory and in this stage, the storage gets completed[3].

Space for learners:

4.10.1. Out-of-Order Execution Implementation

For the complexities of the pipeline to be enhanced, the out-of-order processor needs to be implemented. Now the WAR and WAW hazards can be tackled because the system can reorder the instructions. The following are some of the issues that have been tackled in the pipelining structure and which are very important for Tomasulo's algorithm.

4.10.1.1. Reservation Stations

Reservation stations are one of the features of the CPU which permits the register renaming and Tomasulo's algorithm uses these reservation stations for use in dynamic scheduling. Reservation stations work as the data buffer that fetches and stores the instruction operand values as soon as they are made available and it does not allow the data to get stored in the register. One instruction specifies a single reservation station and the operands once available are sent for its execution and the completed instructions are stored in the buffer of the reservation stations. When there are many instructions and when all of their needs to write in the same register than by the terms, logically only the last instruction can be written in that same register[10].

Sequential instructions are issued to the reservation stations in Tomasulo's algorithm that helps in buffering the instructions. Reservation stations checks the common data bus for the availability of

the data operand and if it is available in the buffer then only the instructions get activated.

There are some fields of register present in the reservation stations which are explained as follows[3]

- Op – Op is the operations that need to be performed on the operand data. It is the functional unit of the associated reservation station. The functional unit can be the arithmetic as well as the logical interpretations such as {AND, OR, NOT, ADD, SUBTRACT, etc.).
- Qj, Qk–In these fields of the reservation station, the source operands are produced and the value of zero indicates that the reservation station has delivered the value to its corresponding source. It produces the source registers.
- Vj, Vk – The source operands have some actual values depicted as Vj and Vk. The actual values will be only valid if the Qj and Qk have the value zero which indicates that the value of the source has arrived in the reservation station.
- A – The 'A' field holds the address of the memory information for a load or a store. Until and unless effective address computation occurs, instructions containing in the immediate field only hold.
- Busy – It works in two Boolean conditions that are True or False. If the condition is True that means the reservation station is occupied or busy. And if the condition is False that means the reservation station is not occupied. The value 1 indicates that the station is busy and o indicates that the station is not busy.
- Qi –all the results of the reservation station are stored in this register. If the value is 0 that means the value present in the register is the actual value of that register. At this point, the register is not renamed.

4.10.1.2. Register Renaming

Instruction results that are stored in the registers are particularly renamed. There may be more than one type of name in the registers that might be used in the system hardware. The reservation stations and the registers are mapped after which the renaming is performed. For

Space for learners:

correctly performing the out-of-order executions, the register renaming is usually applied by Tomasulo's algorithm[11].

It is a pipelining technique that renames the register operands by dealing with the dependences of the data. The operands are specified with the help of a compiler using the architectural registers that are explicit instructions. The renaming register restores the name of these architectural registers by a new value name for the operands of each instruction. It recognizes the true dependencies automatically. It removes the WAR and WAW hazards by dynamically assigning values to the registers [11].

4.10.1.3. Common Data Bus

The functional units and the reservation stations are connected directly with the help of the common data bus (CDB). Tomasulo's algorithm depicts that the CDB "preserves precedence while encouraging concurrency". It can be in two different structures [11].

- Operation results can be accessed by the functional unit without demanding any register with a floating-point and allow multiple functional units to access the register file [11].
- In CDB, control execution and hazard detection are distributed while controlling of the execution of the instruction are done by the reservation stations rather than by a hazard unit [11].

4.10.1.4. Scoreboarding

The scoreboard also follows the dynamic scheduling technique or we can say helps in implementing it so that the execution of the out-of-order can be performed with the condition that no conflicts occur and there is the availability of the hardware. During scoreboard, if data dependencies occur then it is tracked, logged, and observed very often. The scoreboard monitors the system every time to check whether any instruction got stalled or not and tries to resolve the dependencies before any instruction gets stalled[12]. It monitors every instruction that waits for it to get dispatched.

The scoreboard keeps all the latest information into its registers and also determines the time period when the instruction will begin and end.

Space for learners:

The scoreboard contains some stages in which the instruction must pass through it. The stages are given as follows[13].

a) Issue – In the issue stage, the scoreboard checks whether any hazards such as the WAW hazards are available or not. If it ispresent, then the instruction gets stalled[13].

b) Read operands – The scoreboard reads or finds whether any source operands are available or not. If it is present, the functional units are instructed by the scoreboard to check the register file and read the operands so that it can start its execution. The RAW hazards are corrected in this stage. If instructions do not write or use any operand, then that operand is said to be free or available and is present in the register file. If multiple instructions come to the register file, then ambiguity might occur as to which instruction will get the preference to write the operand[13].

c) Execution – The scoreboard gets notified here by the functional unit as to when the execution gets over[13].

d) Write result – As soon as the scoreboard gets notified from the execution stage that the execution has finished, the scoreboard investigates whether any WAR hazards are present or not. If WAR hazards are present, then the functional unit is instructed to get stall by the scoreboard until the hazards are being cleared[13].

The main difference between Tomasulo's algorithm and scoreboarding is that there is no distribution system in scoreboarding. Scoreboarding keeps the track of all instructions and information within itself and is the sole control unit. Whereas Tomasulo's algorithm is a distributed system. All the functional control is distributed among different registers.

Space for learners:

4.11. REORDER BUFFER (ROB)

The reorder buffer creates an apparition to the Users that their instructions are working in order. When a system encounters an

instruction, the instruction gets renamed and decoded and then gets transferred to the ROB as well as the issue queue and simultaneously marked as busy. ROB receives the information once the instruction gets executed and the ROB is marked as not busy. Not busy means that it is now 'committed' and the architectural state gets visible. But if an exceptional instruction remains at the head of the ROB then the architectural changes are not visible[14].

The structure of the ROB is normally a circular buffer that keeps the track of all instructions in order, while the commit head points to the oldest instruction and simultaneously new instructions will be managed within the ROB.

Like the other forms, reorder buffer has also some stages that help in the smooth working of it. They are explained as follows[14]:

a) Exception State –The oldest instruction in the pipeline when gets encountered by the ROB and is pointed to the head pointer then an exception is thrown by the system. A single bit is used to depict the instruction that has entered the ROB or not but the oldest exception instruction is only tracked by the additional exception state. By doing this saves space[14].

b) PC Storage – Branch and Jump instruction are used to access the information into the ROB's PC file at the time of register read[14].

c) Commit Stage – When the head of the ROB does not contain any instruction then it can be committed which means that any changes that occur in the system are made available. ROB releases single instruction in the pipeline but does not check for multiple instructions to get committed. The instruction gets stored into the memory only when the commit is performed. After commit, the instruction physically releases the register[14].

d) Exception and flushes – When ROB contains the instruction at the commit head then only exceptions are handled. The ROB gets emptied by flushing the pipeline. Reset of the rename map table must be done. Control status register (CSR) receives the accepting instruction if the instruction is an architectural exception and if it is a micro-

Space for learners:

architectural exception re-fetching is done of the failing instructions and execution can begin afresh[14].

e) Point of no return – For marking the instruction for which exception might be generated, another pointer head runs just in front of the ROB commit head which is known as the point-of-no-return. It includes memory operations that are untranslated and branches that are unresolved. RoCC instructions that do not tolerate miss peculationare nowadays used by the PNR which means that instruction that has passed the PNR head only gets issued by the ROB[14].

f)

Space for learners:

CHECK YOUR PROGRESS

Fill in the following blanks.:

1. Pipelining is a technique where instruction overlapping occurs at the time of its _____.

2. The efficiency of the pipeline will _____ if the instruction cycle is split into the equal time clock.

3. _____ delay can occur in pipelined processor.

4. The flip-flops that are inserted between the data modules increases the _____ in the instructions.

5. _____ algorithm is a type of hardware algorithm in computer architecture that is used for implementing dynamic scheduling allowing out-of-order execution and enabling multiple execution units.

6. If the reservation station matches the called instruction, then the instruction is issued some operand values in the _____.

7. Reservation Stations checks the _____ for the availability of the data operand.

8. The reservation stations and the registers are mapped after which the _____ is performed.

9. The structure of the _____ is normally a circular buffer that keeps the track of all instructions in order.

10. _____ receives the excepting instruction if the instruction is architectural exception.

Space for learners:

4.12. SUMMING UP

- The order in which the instructions are stored and executed is defined as pipelining processing.
- All the instruction inside the pipeline works concerning some clock time.
- When arithmetic operations come as instruction into the pipeline they are then stored in the arithmetic pipeline.
- When complex instructions are executed on numerous data synchronously, then a vector processor is used. This processor executes the instructions very fast as compared to the scalar processor and has much efficiency.
- The time delay in between two instructions in a pipeline is greatly reduced hence increasing the throughput.
- Branching delay can occur in a pipelined processor.
- Data dependency is a concept that is applied to check whether a block works properly even if the instructions present in that block are rearranged.
- If two or more instructions are spaced closely and have the same dependencies, then it might so happen that the instructions might come to a halt or become idle.
- The WAR and WAW hazards create the possibility of out-of-order execution.

- If the processes have the same slack time, then first cum first serve (FCFS) algorithm is applied together with LST.

Space for learners:

4.13. KEY TERMS

- PIPELINING - It is a technique where instruction overlapping occurs at the time of execution. Instructions are accumulated from a processor into the input registers through a pipeline, and therefore this process is known as pipelining.
- PIPELINING SCHEDULE – Pipeliningscheduling is a type of mechanism where executions are overlapped for different inputs and the computations are performed at different stages. It improves the performances of the machine that have parallel instructions usually termed as instruction pipelines.
- DATA DEPENDENCY – Data dependency is a concept that is applied to check whether a block works properly even if the instructions present in that block are rearranged.
- SCOREBOARD – When the data dependencies are not present and when sufficient resources are present in the system, the score-boarding technique allows the execution of out-of-order performances.
- EARLIEST DEADLINE FIRST – EDF is a type of dynamic scheduling algorithm in which the instructions that have the nearest deadline to complete are given the task of highest priority and are executed first.
- LEAST SLACK TIME FIRST – LST is a type of dynamic scheduling algorithm in which the instructions that have the smallest slack time are given the task of highest priority and are executed first.
- SLACK TIME – When the time of a process gets delayed without other processes getting delayed, is termed Slack time.
- RATE MONOTONIC – RM is a type of static scheduling algorithm in which the instructions that have the smallest job or rate are given more priority than the bigger jobs.

- SHORTEST JOB FIRST – SJFis a type of static algorithm in which the instructions that have the smallest execution time are executed first.

Space for learners:

4.14. ANSWERS TO CHECK YOUR PROGRESS

1. Execution, 2. Increase, 3. Branching, 4. Latency, 5. Tomasulo, 6. Renaming Register, 7. Common Data Bus, 8. Renaming, 9. Reorder Buffer, 10. Control Status Register.

4.15. POSSIBLE QUESTIONS

Short Type Questions:

1) What do you mean by pipelining?
2) What is pipelining processing?
3) What are the two types of a pipeline?
4) Explain in brief the data dependency.
5) Explain in brief the dynamic scheduling.
6) What are the different types of dynamic scheduling and static scheduling algorithms?
7) Write the function of the Issue stage in Tomasulo'salgorithm.
8) What is a reservation station?
9) How does the register renaming work in the pipelining schedule?
10) What do you mean by reorder buffer?
11) What is the function of a common data bus?
12) What do you mean by scoreboarding? Explain in brief.
13) What are the different dependency hazards?
14) What do you understand by point of no return? Explain in brief.

Long Type Questions:

1) What do you mean by pipelining? Explain the different types of pipelines.

2) Discuss the advantages and disadvantages of pipelining.

3) Explain the pipelining scheduling with a relevant example.

4) What are the different types of data dependencies?

5) What are the advantages and disadvantages of dynamic scheduling?

6) Explain the earliest deadline first algorithm.

7) Explain the least slack time first algorithm.

8) Explain the rate monotonic algorithm.

9) Explain the shortest job first algorithm.

10) Explain Tomasulo'salgorithm.

11) What do you mean by reservation station? Explain all its stages.

12) What do you mean by register renaming?

13) What do you understand by reorder buffer?

Space for learners:

4.16. REFERENCES AND SUGGESTED READING

[1] https://www.lkouniv.ac.in/site/writereaddata/siteContent/20200422161333844 5rohit_engg_pipelining_and_hazzard.pdf

[2] https://www.quora.com/What-is-pipelining-scheduling-in-computer-architecture

[3] https://www.brainkart.com/article/Dynamic-Scheduling_8832/

[4] https://en.wikipedia.org/wiki/Instruction_scheduling

[5] Teraiya, J., & Shah, A. (2020). Analysis of dynamic and static scheduling algorithms in soft real-time system with its implementation. In *Soft Computing: Theories and Applications* (pp. 757-768). Springer, Singapore.

[6] https://www.cs.umd.edu/~meesh/411/CA-online/chapter/advanced-concepts-of-ilp-dynamic-scheduling/index.html

[7] COMPUTER ORGANIZATION AND ARCHITECTURE DESIGNING FOR PERFORMANCE EIGHTH EDITION, BY WILLIAM STALLINGS, published by Prentice Hall (an imprint of Pearson)

[8] https://www.cse.iitk.ac.in/users/biswap/CS422/L12-Tomasulo.pdf

[9] http://www.cs.umd.edu/~meesh/411/CA-online/chapter/dynamic-scheduling example/index.html

[10] https://www.cs.umd.edu/~meesh/cmsc411/website/projects/dynamic/tomasulo.html

[11] https://en.wikipedia.org/wiki/Tomasulo_algorithm

[12] https://en.wikipedia.org/wiki/Scoreboarding

[13] https://www.cs.umd.edu/~meesh/cmsc411/website/projects/dynamic/scoreboard.html

[14] https://docs.boom-core.org/en/latest/sections/reorder-buffer.html

---×---

Space for learners:

UNIT 5 – ADVANCED CPU ARCHITECTURES

Space for learners:

Unit Structure:

5.1 INTRODUCTION

In last three decades, the architectures of CPU design have been implemented on an unprecedented scale on a single chip due to the advancement of Integrated Circuit fabrication technology. So, this becomes very much relevant for you to learn about different instruction set architectures. Moreover, it is also very important to know about the interconnection between multiple processors & cache memory and the cache coherence problem which may arise with such interconnections.

This unit is divided into two parts. In the first part, we will take a close took CPU architectures. Our primary focus is Very Large Instruction Word (VLIW) architecture. You will get a brief introduction to different instruction set architectures like CISC and RISC, which are implemented in superscalar processors. The detailed architecture of VLIW processors will be discussed in this unit along with basic working, instruction format, advantages and disadvantages. Additionally, the implementation details of Explicitly Parallel Instruction Computing (EPIC) architecture will also be discussed, which is a VLIW inspired architecture, developed by HP and Intel. You will learn how EPIC differs from VLIW and how EPIC overcomes certain limitations of VLIW.

The second part of the unit focusses on the concept of Multiprocessor Systems, which is based on Multiple Instruction stream, Multiple Data stream (MIMD)design scheme. We will discuss about the different classification of multiprocessors – namely tightly coupled and loosely coupled. You will also learn about different interconnection structures between multiprocessors along with pros and cons of each of the structure. We will also discuss how cache memory is used in uniprocessor and multiprocessor systems to increase the performance along with the cache coherence problem. We will also look into different hardware and software-based solutions to the cache coherence problem.

Space for learners:

5.2 UNIT OBJECTIVES

The objective of this unit is to give an introduction to advanced CPU architectures and multiprocessor systems. After completing this unit, you will be able to

- Learnabout the VLIW architecture and how it differs from the superscalars
- Know the EPIC architectures and how it differs from VLIW
- Learn the concept of multiprocessor systems and its types
- Understand the different interconnection structures in multiprocessor systems with pros and cons.
- Define cache coherence problem and its solutions.

5.3 INTRODUCTION TO ADVANCED CPU ARCHITECTURES

In the field of Computer Science, an abstract model of a computer system is defined by Instruction Set Architecture (ISA), also known as Computer Architecture. Implementation of ISA corresponds to the realization of ISA, such as CPU, registers, main memory, data types to be supported, etc. An ISA is like a contract between the set of microprocessor implementation of an architecture and the class of programs that are written for that architecture. ISA defines a basic set of operations that must be performed by the system and serves as boundary between hardware and software. The set of operation may include arithmetic, logical, branching and memory operations. The ISA provides details about how a machine code over the implementation of a particular instruction set architecture doesn't depend on the prime characteristics of that implementation. Thus, it allows multiple implementations of ISA, which may differ in physical size, overall performance and prices, but can run the same machine code or software.

Space for learners:

5.3.1 Classification of Instruction Set Architectures:

1. **Complex Instruction Set Computer (CISC):** In CISC architecture, there are hundreds of instructions or commands of variable lengths, that instructs the system to perform addition of numbers, storing and displaying results. This approach is carried out in order to save the memory since all instructions of same length will contribute to wastage of memory. Here, simple commands may require 8-bits and complex commands may require 120 bits. An implementation of CISC architecture is *Intel x86*, which was introduced in 1978. CISC provides convenient addressing modes and enables copying the block of instructions through support for functions using CALL instructions. Thus, in CISC, it is easy to expand the ISA.

2. **Reduced Instruction Set Computer (RISC):** In RISC architecture, the computer system uses sets of instructions which are highly optimized and CPU design focuses on raw performance. In contrast to CISC, RISC uses relatively simple, fixed length instructions of 32-bits. Although fixed

length instructions may mean more space wastage, however the instructions are faster & easier to execute. Moreover, in terms of CPU design perspective, RISC integrated chips requires a smaller number of transistors as compared to CISC, since RISC implementation deals with only handful types of instructions and delivers high performance. However, due to short instruction size, a greater number of instructions are executed compared to CISC, in order to accomplish a given function. Example of RISC architectures includes Sun Microsystem's SPARC, IBM/Motorola's PowerPC, Hewlett-Packard's PA-RISC,SGI's MIPS, ARM architecture, etc. In recent times, almost all low-end portable devices are based on ARM architecture, which includes most Android-based systems, Apple's iPhone and iPads, Nintendo's video gaming console Switch, Raspberry Pi and many more.

Please note that the simplicity of RISC allows to easily design superscalar processor tha tcan execute more than one command or instruction at a time. This concept is known as *Instruction-level parallelism (ILP)*. In modern times, almost all CISC & RISC processors are superscalar in nature, however, this has introduced new levels of design complexity for CPU architects.

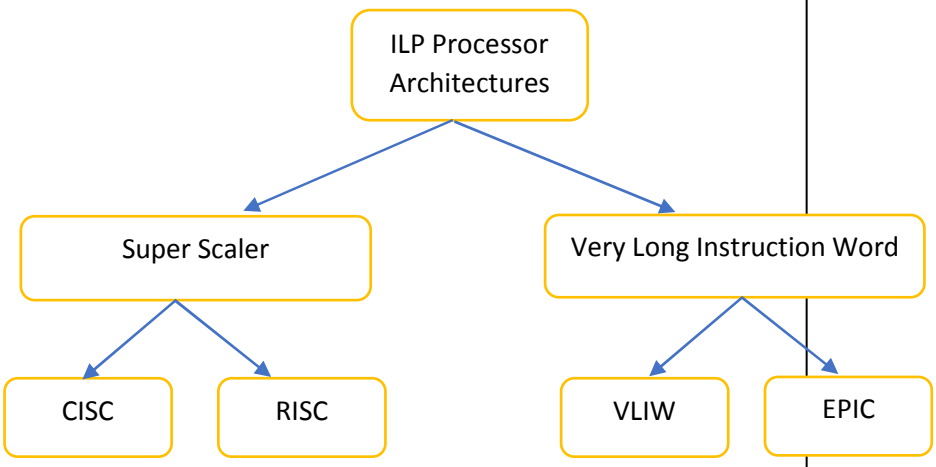


Fig. 1. Classification of ILP Processor Architecture

Now, there are two most significant types of ILP processors, namely *Superscalar* processors and *Very Long Instruction Word (VLIW)* processors. We have already come across superscalar processors, an implementation for ILP processor architectures in which programs

Space for learners:

doesn't have any explicit information about parallel execution of instruction and it is the responsibility of the system hardware to detect and construct action plans for any ILPs to be exploited for parallelism. On the other hand, VLIW processors are built on an architecture in which programs contain explicit information about parallelism and it is the responsibility of the software, called compiler to identify and communicate it to the hardware by specifying all the independent operations. Thus, the hardware doesn't have to check further on the operations which can execute in the same cycle, since the information is already provided by the compiler. Let's explore VLIW architecture in details in next section.

Space for learners:

Check Your Progress

1. An abstract model of a computer system is defined by _____

2. What type of operations are defined by an instruction set architecture?

3. What are the different class of Superscalars based on Instruction Set Architecture?

4. RISC stands for _____

5. Give one example each for implementation of CISC and RISC architecture.

State TRUE or FALSE:

6. CISC uses simple, fixed length instructions.

7. In RISC, CPU design focuses on raw performance and the instructions are highly optimized.

8. Instruction-level parallelism is the ability of a processor to execute more than one instruction at a time.

5.4 VLIW ARCHITECTURE

In the early 1980s, John Fisher, a faculty from Yale University, invented the architectural concept and coined the term VLIW among his research group. He later joined HP Labs. VLIW refers to a processor architecture designed to take advantage of instruction-level parallelism (ILP). It is less complex approach to allow higher

performance i.e., multiple operations are performed simultaneously or level of parallelism increases.

In VLIW Processor,

- Instruction consists of multiple independent operations grouped together.
- There are multiple independent functional units.
- Each operation in the instruction is assigned to different functional units.
- All functional units share the use of a common large register file.

For example –

ADD R1, R2; SUB R5, R6; LD R7, data; STR R8, data;

In this example, there are four operations. ADD (Addition) and SUB (Subtraction) are arithmetic operations, which corresponds to Arithmetic & Logic Unit (ALU). Similarly, LD (Load) and STR (Store) are memory operations, which corresponds to Memory Unit (MU). Here, we can see that independent operations are grouped together in a single instruction word. Now, the CPU will assign each of these operations to different independent functional units to execute the operations parallelly, thus to achieve instruction level parallelism (ILP) and higher performance.

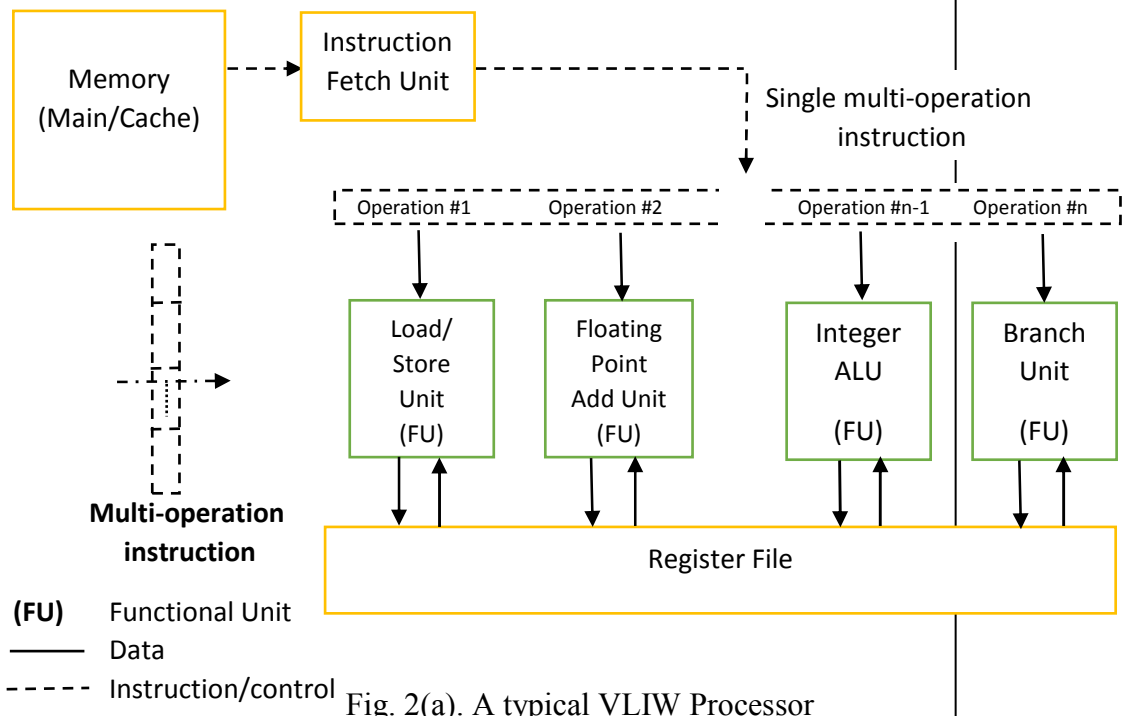


Fig. 2(a). A typical VLIW Processor

Space for learners:

Space for learners:

Load / Store	Floating-Point Addition	Floating-Point Multiplication	Branch		Integer ALU

Fig. 2(b). A VLIW Instruction Format

In VLIW Processor, compiler is responsible for static scheduling of instructions i.e., compiler finds out which operations can be executed in parallel in the program. Compiler groups together these independent operations in a single instruction (VLIW) which is the VLIW. It also makes sure that before the operands are ready, an operation is not issued.

VLIW instruction contains operands & operations to be performed by the various functional units. One VLIW instruction encodes at least one operation for each functional (or execution) unit on each cycle. So, length of the instruction increases with the number of functional (or execution) units. For example, as we have seen earlier if we have two ALU and two Load/Store functional units in our VLIW architecture, then VLIW instructions length will be four. These operations are assigned to functional units by the position in the given fields within the long instruction word. This is known as *slotting*.

Stop to Consider

- ✓ The instructions within a VLIW instruction are issued and executed in parallel.
- ✓ Since in VLIW processor, one VLIW instruction word encodes multiple operations, which allows them to be initiated in a single clock cycle.
- ✓ The start of execution of the operations is bound by the VLIW instruction in which it appears, and all the operations in a VLIW start executing together in parallel
- ✓ VLIW instructions are at least around 64 bits wide and 1024 wide in some architecture.

5.4.1 Example of VLIW code:

RISC Code	VLIW Code	
MUL R1, R3, 3	MUL R1, R3, 3	SUB R3, R3, 1
LD R4, 0(R1)	LD R4, 0(R1)	NOP
ADD R2, R2, R4	NOP	NOP
SUB R3, R3, 1	ADD R2, R2, R4	BNEZ R3, -4
BNEZ R3, -4		

In the above example for RISC Code, content of Register R3 is multiplied by 3 and is stored in R1. The R4 is loaded with the data stored in the address that R1 contains. Then the content of registers R2 and R4 are added and stored in R2. The content is R3 acts as counter and is decremented by 1. BNEZ instruction is a conditional branch which checks if content of R3 is not equal zero and if the condition satisfies, the control is passed back to -4 instructions from the top i.e., to the MUL instruction at the beginning. In VLIW code, this sequence is divided by the compiler in such a way that similar task can be carried out parallelly on different execution or functional units to achieve high performance.

5.4.2 Examples of VLIW Processors:

- VLIW Mini Supercomputers – Multiflow TRACE 7/300, 14/300 and 28/300
- Single Chip VLIW Processors – Philip's LIFE Chips
- DSP Processors - Analog Devices' SHARC DSP, Texas Instruments' C6000 DSP family
- Intel's Itanium IA-64 EPIC (embedded & nonembedded)
- Tilera TILE Pro

5.4.3 Advantages of VLIW

1. Compiler determines data dependency checks and other instruction issues; it becomes a lot simpler.
2. Reduces hardware complexity
3. Compiler is used to schedule according to functional units.
4. Compiler issues instructions corresponding to the position of functional units.

Space for learners:

5. Ensures low power consumption due to reduction of hardware complexity.
6. Increases potential clock rate.

5.4.4 Disadvantages of VLIW

1. Higher complexity of the compiler, which are hard to design.'
2. VLIW processors cannot react on dynamic or unscheduled events. It can work only on static instructions. Unscheduled events, for example a cache miss could lead to a stall which will stall the entire processor.
3. Large memory bandwidth & more registers for software pipelining, etc.
4. Increased program code size.
5. The number of instructions in a VLIW instruction word is usually fixed.
6. If issued bandwidth is not met, padding of VLIW instruction word is needed, which results in increase in code size.
7. In case of un-filled opcodes in a VLIW, padding of VLIW instructions with No-Ops (No Operations) is required, for which there is waste of memory space and instruction bandwidth.

5.4.5 Applications of VLIW Processors

- It is suitable for Digital Signal Processing Applications.
- It is used for tasks, which involves processing of media data, like compression /decompression of image and speech data.

SAQ

1. Draw a typical VLIW processor and explain in brief about the architecture

2. What is the role of a compiler in a VLIW processor architecture?

3. State the advantages and disadvantages of VLIW architecture.

4. Explain in brief about the VLIW instructions format and slotting.

5. State the applications of VLIW Processors.

Space for learners:

5.5 EPIC Architecture:

Explicitly Parallel Instruction Computing (EPIC)is a term proposed by Hewlett Packard & Intel, which formed an alliance in early 90s for the research and implementation of Intel Itanium architecture (IA-64). In 2001, IA-64 was launched as a collection of 64-bit Intel Itanium microprocessors. Though the original ISA specifications were by HP, but it was later evolved and implement as a new processor micro architecture by Intel.

5.5.1 EPIC vs VLIW

EPIC is inspired by VLIW architecture at roots, so it permits execution of instructions in parallel using a compiler instead of complex circuits, which were earlier used to control instruction level parallelism (ILP). In contrast to VLIW, apart from identifying and grouping the independent operation in a single instruction, the compiler communicates this via explicit information in the instruction set. That's why EPIC is also known as "independence architecture" *(Fisher & Rau)*. Unlike VLIW, EPIC retains backward compatibility across different implementations like superscalars, but doesn't need any hardware for *dependency checks* like superscalars. EPIC is a mix of software & hardware, incorporating the advantages of both superscalars and VLIW architectures, while fixing several shortcomings of VLIW.

1. VLIW instructions had a backward compatibility issue between wider and narrower implementations. Wider implementation uses greater number of execution units (EU), which also increases the size of an instruction since the number of operations to run in parallel also increases. Such a wider instruction set doesn't work well with narrower implementations with lesser number of execution units.
2. The static scheduling by the compiler for load instructions became quite difficult since memory operations need to work with several devices from memory hierarchy, like CPU cache memory and DRAM, which doesn't have any deterministic delay for load responses. In other words, the compiler couldn't predict the delay in response time efficiently for the load instructions using different memory technologies.

Space for learners:

So, although EPIC evolved from VLIW architecture, it tries to retain some properties from superscalar architecture. There are several additions to features of EPIC architecture in contrast of VLIW as discussed in next section.

Space for learners:

5.5.2 EPIC architectural details

In EPIC architecture, we have a "*bundle*" of multiple software instructions. Each of these bundles includes a stop bit to indicate if there is some interdependencies between two subsequent bundles. The dependency information is calculated by the compiler. This information could help in issuance of multiple bundles in future implementations. Typically, a bundle is of 128 bits, with thee 41-bit instructions per bundle and only two bundles can be issued at once. For data prefetch, software prefetch instruction is used, which not only increases cache hit for load operation, but also indicates the requirements of temporal locality in different cache levels. For these purpose, two types of load instructions, namely *speculative load instruction* and *check load instructions* are used in EPIC to bypass control and data dependencies.

Moreover, EPIC follows a fully predicated instruction set architecture, that enables *predicated execution,* which decreases the occurrence of branching and increase *speculative execution* of instructions.

Stop to Consider

- ✓ **Predication:**
  "In computer science, Predication is an architectural feature that provides alternative to conditional transfer of control, implemented by machine instructions such as conditional jump, conditional call, conditional return and branch tables. It means if a register condition bit is set, the instruction is executed; if the bit is clear, it is not." – Predication *(on Wikipedia)*

- ✓ **Speculation:**
  "Speculative execution is an optimization technique where a computer system performs some tasks that may not be needed. Work is done before it is known whether it is actually needed, so as to prevent a delay that would have to be incurred by doing the work after it is known that it is needed. If it turns out the work was not needed after all, most changes made by the work are reverted and the results are ignored. The objective is to provide more concurrency if extra resources are available. This approach is employed in a variety of areas, including branch prediction in pipelined processors, value prediction for exploiting value locality, prefetching memory and files, etc." – Speculation *(on Wikipedia)*

- ✓ **Register renaming:**

  Register renaming is a technique of managing data dependencies between the instructions in the pipeline by renaming the register operands.

Space for learners:

In this architecture, the register files are very large and there are wide range of registers at disposal to avoid *register renaming*.Registers include 128 integer and floating-point registers, 128 additional registers for loop unrolling & optimization, 8 indirect branch registers and other miscellaneous registers. Moreover, predication (or multi-way branch instruction) improves the prediction of branch instruction by combining branches as alternate instruction in one bundle.

Lastly, let us revise the difference between Superscalars, EPIC & VLIW.

	Grouping of instructions *(Checking dependencies between instructions to find group able instructions for*	**Assigning of functional unit** *(Assigning instructions to the functional or execution units of the*	**Initiation of execution** *(Determining when the execution starts or instructions are initiated)*

	parallel execution)	*hardware)*	
Superscalar	Hardware	Hardware	Hardware
EPIC	Compiler	Hardware	Hardware
VLIW	Compiler	Compiler	Compiler

Check Your Progress

9. EPIC stands for_____

10. The first implementation of EPIC architecture is _____ family of processors.

11. In VLIW, _____issues instructions corresponding to the position of functional units.

12. EPIC is developed as a joint collaboration between _____

13.EPIC follows a fully _____ instruction set architecture

State TRUE or FALSE:

14. VLIW is inspired by EPIC architecture

15. In EPIC, the functional units are assigned by compiler.

Space for learners:

5.6 INTRODUCTION TO MULTIPROCESSOR SYSTEMS

A multiprocessor system is a computer system with more than one processor (typically two or more), where each processor is linked with one another. The connection between these processors is known interconnection network. The primary focus of a multiprocessor system is to achieve parallel processing, which enhances the overall performance. Apart from high performance, the multiprocessor system focusses on –

1. *Fault Tolerance and graceful degradation:* These systems have high fault tolerance since multiple processors are at play. In case of system failure, the system can continue to run in low power, until it stops completely.

2. *Scalability and modular growth:* The number of processors, memory units, etc. can be added or removed at any point of time. This modularity allows for scalable enhancements in future.

Multiprocessor system falls under MIMD architecture. It is one of the types of parallelism as per Flynn's classification of computer organization. The MIMD refers to multiple control units and multiple execution units or processors. There are multiple instruction and data steams as shown in figure below.

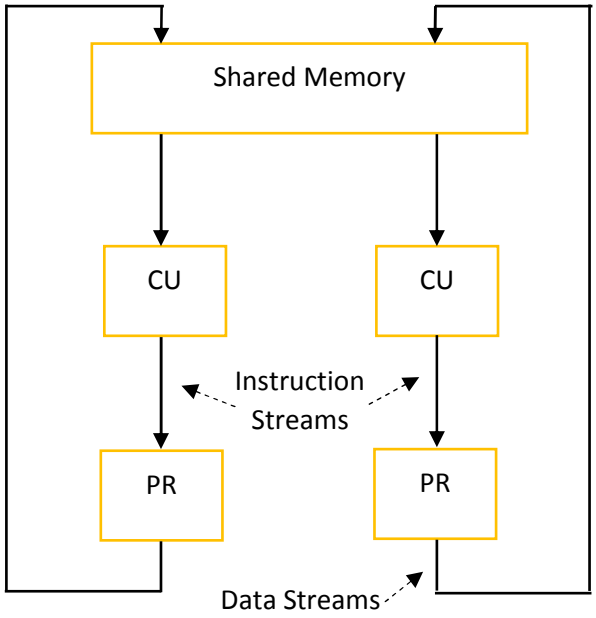


Fig. 3. MIMD with shared memory

The MIMD refers to multiple control units and multiple execution units or processors. There are multiple instruction and data steams as shown in figure 3.

Stop to Consider

- ✓ Please note that *multiprocessor* and *multicomputer* systems may sound similar, but there exists an important difference.
- ✓ Both of them support concurrent operations, but a multicomputer system is a system with multiple computers and a multiprocessor system is a system with multiple processors.

Space for learners:

- ✓ In multiprocessor systems, there is a single operating system, which provides interaction between processors and all the components of the system cooperate in the solution of a problem.
- ✓ In multicomputer system, each computer has a separate operating system, however these computers work together as a single entity.

Space for learners:

5.6.1 Classification of Multiprocessors

The following figure shows different types of multiprocessors. They are primarily divided into two-types: *tightly coupled system* and *loosely coupled system*.

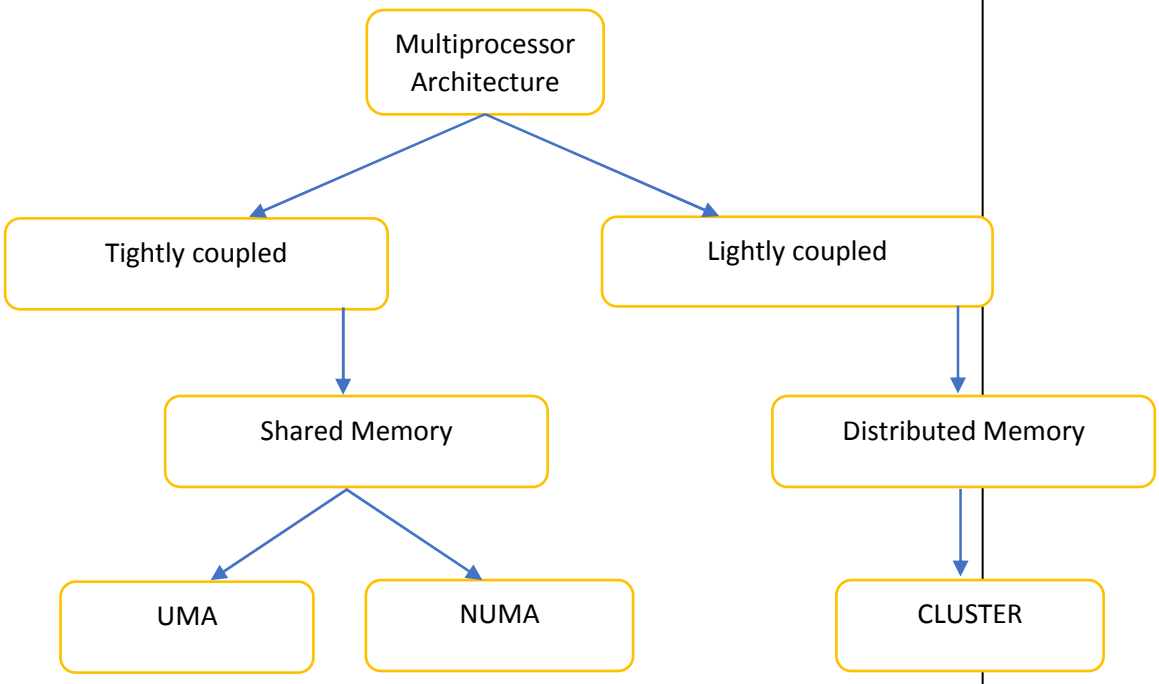


Fig. 4. Classification of Multiprocessors

A *tightly coupled* multiprocessor, also known as shared memory multiprocessor system, share information between multiple processors via a shared or global memory. Here, all processors share a single memory address space and communicate among themselves through shared variable in memory. Each of the processors can access any location in the shared memory. Apart from shared

memory, each processor can also have a dedicated local memory which other processors cannot access. Please note that all the processors in the multiprocessors system communicate to perform tasks in a highly synchronized fashion.

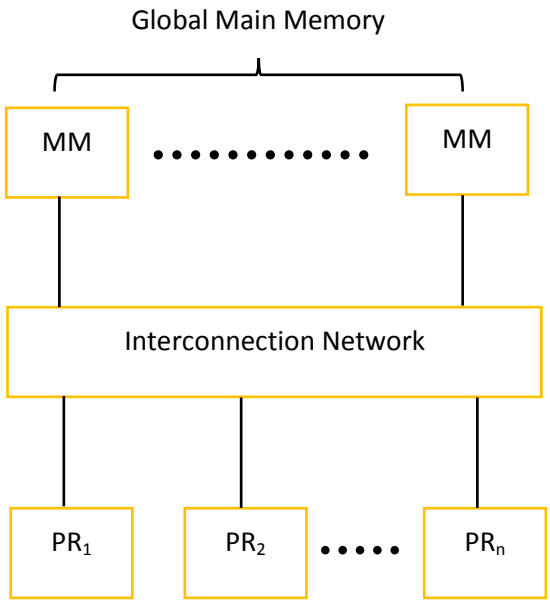


Fig. 5. Tightly coupled multiprocessor system

In tightly-coupled multiprocessors, we have Uniform Memory Access (UMA). In a UMA multi-processor system, the access time of memory is equal for all the processors irrespective of which processor accesses which portion of the common memory. Although the access time of memory is almost equal, the memory access in UMA is bit slow due to the use of a single memory controller. We also have Symmetric Multiprocessor (SMP) system, which is an UMA multi-processor system with identical, homogenous processors, which are capable of performing similar functions and utilizes a centralised shared main memory.

There is also another type of tightly-coupled multi-processor system known as Non-Uniform Memory Access (NUMA) system. In NUMA multi-processor systems, the memory area is virtually divided into faster access area and slower access area. The faster access areas are assigned to the processors and the slower common area is used for the exchange of data. Several memory controllers are used for this purpose for allowing local faster memories to be used as actual main memories. This enables NUMA to manage workloads to achieve higher performance than UMA multi-

Space for learners:

processor systems. These systems are also known as Distributed Shared Memory (DSM). In DSM multiprocessor system, the processors have a shared address space for all the memories.

A *loosely coupled* multiprocessor system, also known as the distributed memory multi-processor system, doesn't share information between multiple processors via a shared memory, since each processor has its local dedicated memory, which together forms a distributed memory. Please note that all the processors in the multiprocessors system do not communicate to perform tasks in a highly synchronized fashion. Processors communicate and share explicit information among each other using a common message passing protocol via interconnection network, for which the overhead of data exchange is high.

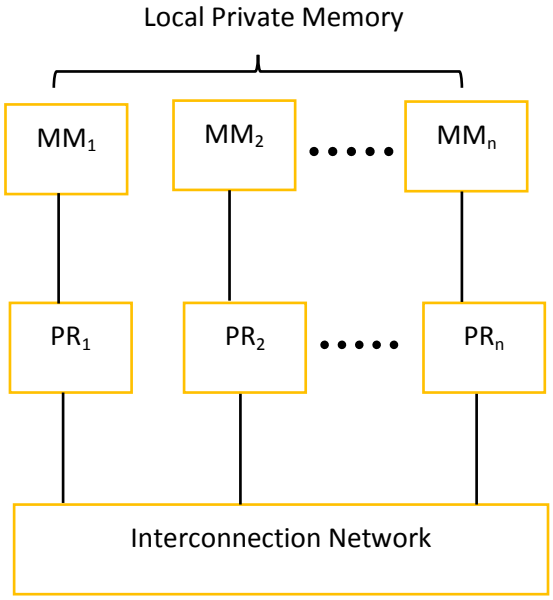


Fig. 6. Loosely coupled multiprocessor system

The loosely coupled multiprocessor system has physically distributed memories like in the case of cluster. A cluster consists of a set of computers connected over a local area network (LAN) which function as a single large multiprocessor. In the cluster system, there is no sharing of address space and each cluster node works together, although it can also work independently. Since a cluster act like a multiprocessor, it can provide the benefits of multiprocessor system along with additional benefits like load sharing and better fault tolerance.

Space for learners:

Stop to Consider

- ✓ Tightly-coupled multiprocessor systems use a shared memory (can be a virtually distributed shared memory) and Loosely-coupled multiprocessor systems use physically dedicated distributed memory.
- ✓ In literature, the terms UMA and SMP are used interchangeably, since access to shared memory is balanced in both the cases.
- ✓ NUMA can be considered as a tightly coupled form of cluster.
- ✓ Cluster is not same as a Computer Network. The primary objective of a computer network is *resource sharing* but for Cluster, it is *parallel computing*.

Space for learners:

5.7 INTERCONNECTION TYPES

In multiprocessor systems, the components like CPU and I/O Ports are connected to I/O devices and a memory unit, which can be shared or distributed in nature. The interconnection between the components be of different physical configurations, described as follows:

a) Time-Shared Common Bus Structure:

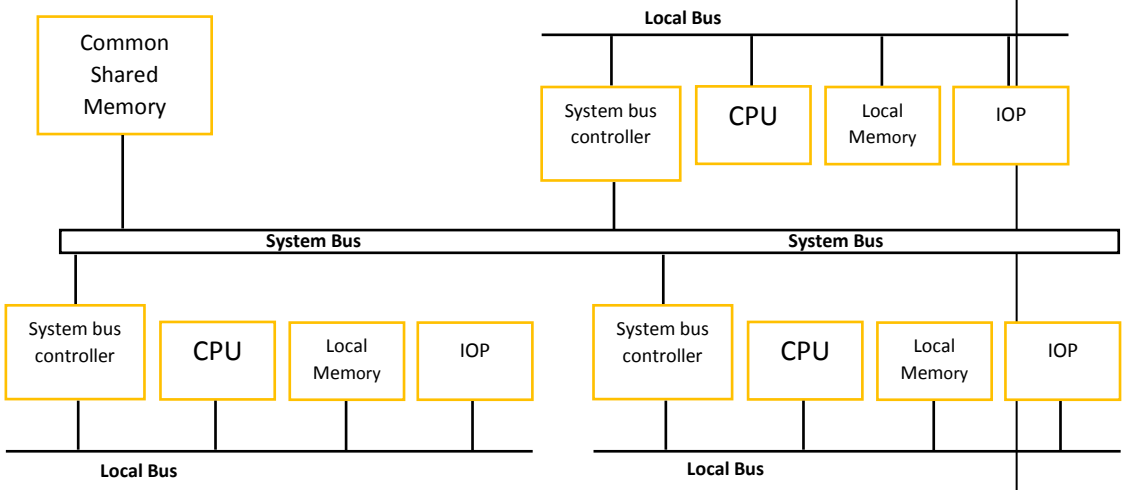


Fig. 7. Time-shared common bus structure

In this structure, all the processors in the microprocessor system are connected to shared memory and other common resources using a

common interconnecting path, called as common system bus. In this structure, only one processor out of others can communicate with the shared memory or any other processor over the system bus at a given time, thus *time-shared*. Each processor can also have a local bus to communicate with its local memory and local I/O. The benefit of this is while one processor is working on system bus, other processors can communicate with local memory and local I/O through local bus. Please note that a part of local memory can be designed as cache and can be attached to CPU to reduce the average access time of the local memory.

Pros

1. The design is simple due to the use of single common system bus.
2. It is a cheap and affordable structure.

Cons

1. Since only one processor at a time can transfer or communicate over the system bus, the communication is quite slow. It means when one processor is accessing the shared memory using the bus, others can't perform any other operation using the bus.

b) Multiport Memory Structure:

In this structure, the system has separate buses between each memory module and the processors. For example, if we have 4 processors and 4 memory modules, then each memory module will have 4 ports each connecting to each of the processor bus. The processor bus consists of data, address and control lines. Each of the memory module has an internal control or priority logic to determine which processor request will be granted i.e., which port will have access to memory module at a given time, when there is a conflict of simultaneous requests from multiple processors in the system. Generally, a fixed priority is assigned to each memory ports to avoid memory access conflicts. Moreover, each processor is associated with a priority of the memory access, which is determined by the physical position of the port that the processor bus occupies in each module. So, processor P1 will have the highest priority and priority of the processor P4 will be the lowest.

Space for learners:

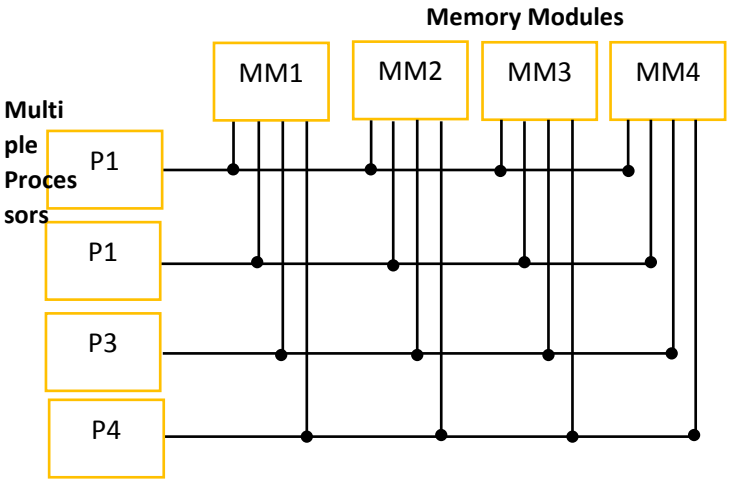


Fig. 8. Multiport memory structure

Pros

1. Due to multiple paths between the processors and memory modules, multiple processors can simultaneously access the memory, thus, high transfer rate can be achieved through this organization.

Cons

1. It requires a huge number of interconnecting cables to connect all the processors with memory modules. Thus, it is suitable for systems with small number of processors.
2. It also requires large hardware in memory modules in the form of memory control logic, which is very expensive in cost.

c) Crossbar Switch Structure:

In this structure, a number of crosspoints are placed at the intersection of memory paths and processor buses. At each crosspoint, there is a control logic to set the desired path between a memory module and a processor. This control logic is basically a switch, which is an electronic circuit. A switch can also resolve the conflict of simultaneous requests from multiple processors to access same memory module in the system based on a fixed priority basis. The following figure shows a crossbar switch interconnection for a system with 4 processors and 4 memory modules with 16 switches represented by small-squares marked by S_1 to S_{16}.

Space for learners:

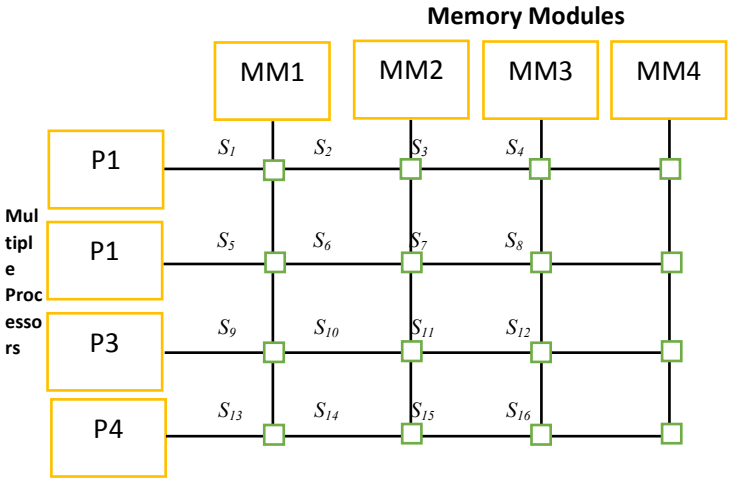


Fig. 9. Crossbar switch structure

Pros

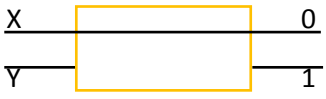
1. Since there exists a separate path associated with each memory module, simultaneous transfer from all memory module is possible.

Cons

1. The entire connection here relies on switches. So, if large number of processors are present, then the design & implementation of switch requires large hardware and becomes complex.

d) Multistage Switching Network Structure

In this structure, we use a switch which can interchange *two-inputs, two-outputs*, in contrast to that of crossbar switches, which allows one stage of electronic switches – either input or output - to determine the path between multiple processors and multiple memory modules. Hence the name, multistage switching network since it allows to build different possible stages for different combination of inputs & outputs. Let us take the example of 2 x 2 interchange switch, which has 2 inputs – X & Y and 2 outputs – 0 and 1.



X is connected to 0

X is connected to 1

Space for learners:

Y is connected to 0 Y is connected to 1

Fig. 10. Interchange switch states

You can see that how four different states are possible in a single switch. Now, in place of X & Y, if we have two processors connected say P1 and P2, then we can have definite control to reach a particular memory module from a processor. These interchange switches allow to connect a source to a destination through multiple stages using a control logic.

A very popular topology is called omega switching network which allows exactly one path from each source to any particular destination. Simultaneous connections by two sources to two destinations connected to same switch is prohibited.

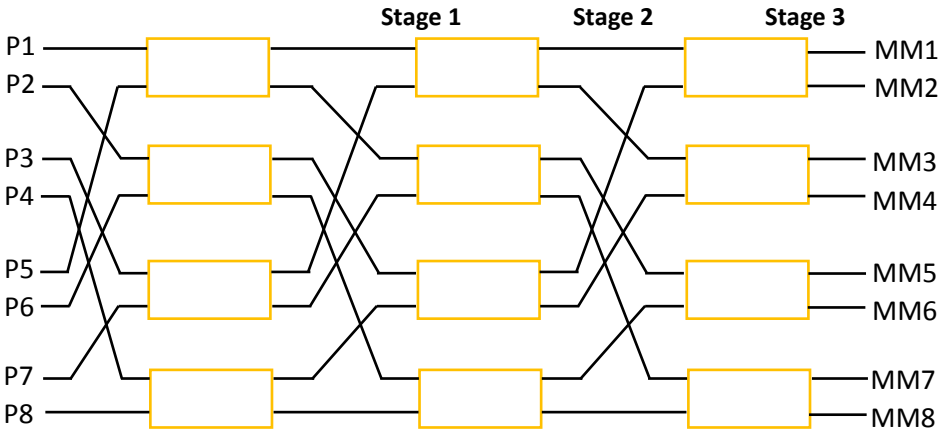


Fig. 11. An 8 x 8 Omega Multistage Switching Network

Pros:

1. The structure is cost effective since we can connect multiple sources to multiple destinations with less amount of wiring compared to crossbar switch structure.

Space for learners:

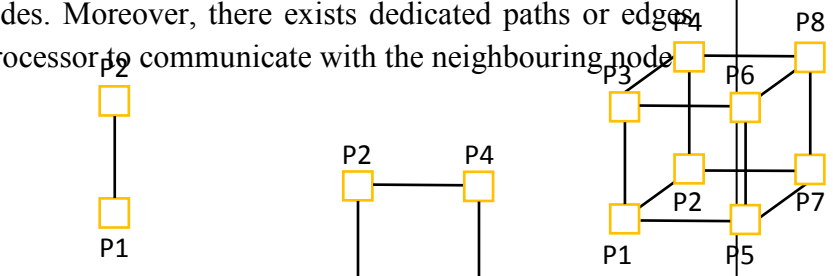
Cons:

1. There is a restriction on the number of simultaneous connections, since simultaneous connections by two sources to two destinations connected to same switch is prohibited.

Space for learners:

e) Hypercube Network Structure

In this structure, a loosely coupled system is realized with the help of a concept called hypercube. A hypercube structure is comprised of $N = 2^n$ numbers of processors interconnected to each other in a N-dimensional cube. This structure is also known as a *binary N-node* multiprocessor structure. A node of the cube is represented by a processor and an edge of the cube is a communication path connection two nodes. Moreover, there exists dedicated paths or edges for a processor to communicate with the neighbouring node.



n=1, N=2
Binary two-node cube
(One Dimensional)

n=2, N=4
Binary four-node cube
(Two Dimensional)

n=3, N=3
Binary 8-node cube
(Three Dimensional)

Fig. 12. Hypercube structures

Pros:

1. It is easy to scale the current network to higher configurations simply by increasing the value of *n* which is the dimension of the cube.
2. Since it is a loosely coupled system, intelligent communication protocols could be easily implemented.

Cons:

1. The multiple paths between processors increases the routing complexity.

Check Your Progress

16. MIMD stands for_____

17. Uniform Memory Access corresponds to _____ multiprocessors and Clusters corresponds to _____ multiprocessors.

18. A hypercube contains _____ numbers of processors.

State TRUE or FALSE:

19. In multistage switching network, an interchange switch has two-inputs, two-outputs.

20. In multiport memory structure with 3 processors, one memory module will have 3 ports connecting to each processor.

Space for learners:

5.8 CACHE MEMORY: UNIPROCESSOR VS MULTIPROCESSOR

A cache memory is a faster memory which sits in between a processor and main memory. Its primary role is to reduce the average access time for a particular data. In a system without a cache memory, the processor might have to higher access time given that the access to a particular data is needed on consecutive execution of instructions. The cache tends to hold those data in itself which has high probability of being asked by processor in next cycle. Also note that cache is also realised as a random-access memory, so the time taken to access any part of cache is almost same. Both this factor makes cache memory quite efficient solution to reduce the average access time.

In a system with single processor, the read and write operation on cache works as follows. During a read operation, a *word* from cache line is sent to the processor, the main memory is not involved in the transfer. During the write operation, two widely-used policies – *write back* and *write through* – are used. In *write back*, the cache memory is regularly updated after every write operation and all the changes made in cache are marked and is updated on main memory later. On the other hand, in *write through*, the cache and the main memory are updated simultaneously.

However, in a multiprocessor system, we can have a common shared main memory among all the processors. Each processor can also have local cache memory in order to reduce the average access time on an instruction cycle time. We already know for a processor writes its cache memory during a write operation. During the execution of an instruction, if any processor locally writes its cache, the new values must be made available to the all the other processors to maintain the consistency of the system. In case if the new value is not updated in common shared memory, then the other processors will receive and use the old values in their cache, which should not be allowed. Thus, when any of the processor makes modification in its cache,

a. All the other processors should either update their cache with the new modified value, or
b. Mark the old data in their cache as invalid.

Stop to Consider

- ✓ In a uniprocessor system, the main memory is for use by a single processor. In multiprocessor, the main memory is shared among all the processors.
- ✓ Each processor has its own cache memory and can incorporate either *write through* or *write back policy* to update its own cache.

Space for learners:

5.8.1 Cache Coherence Problem:

Cache coherence is a condition which states that all the cache lines with a particular shared main memory block must contain same information at any given point of time. This ensures that a multiprocessor system can perform memory operation correctly, by keeping identical multiple copies of information in the caches of the processors involved in execution of a particular instruction. Cache coherence problem occurs when cache coherence is not maintained, i.e., a processor updates its cache and other processors doesn't get an updated copy of newly modified data in their cache. This hampers the uniformity of data in all the caches of processors. Cache coherence problem happens in a multiprocessor system, since multiple processor access and works on non-identical multiple

copies of data. Now as the cache coherence problem has been discussed, let's see the solutions for this problem.

Space for learners:

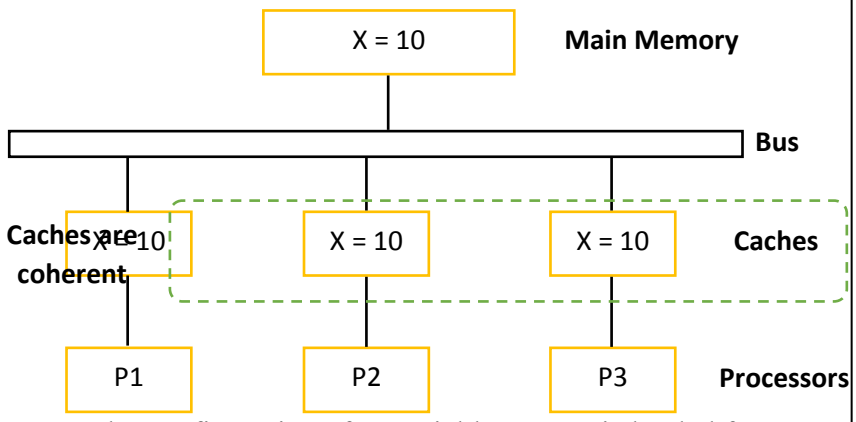


Fig. 13. Cache configuration after variable X = 10 is loaded from Main Memory

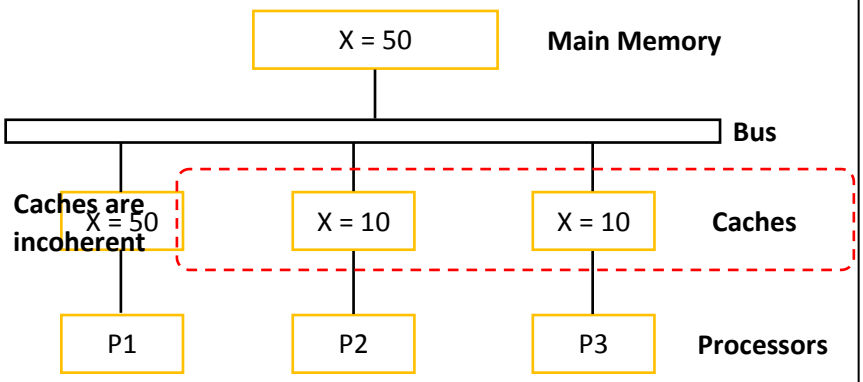


Fig. 14. Write-through policy. Modified value X=50 in Cache & Main Memory

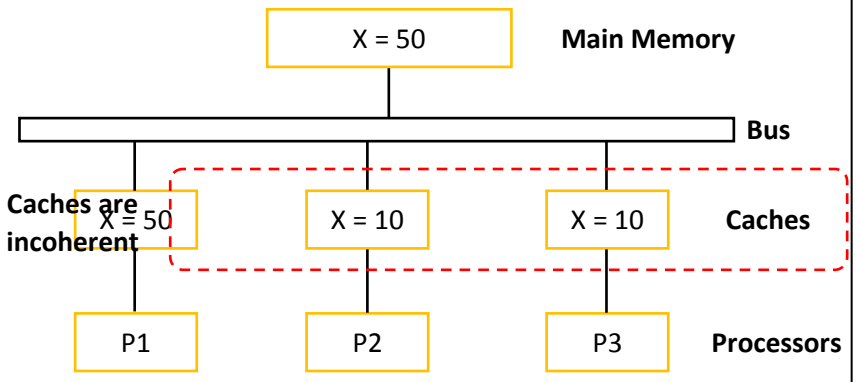


Fig. 15. Write-back policy. Modified value X=50 only in Cache, MM to be updated later.

5.8.2. The "All-is-well" Solution:

One of simple scheme can be to restrict the association of local caches for each processor and force them to use a shared cache memory instead. However, this simply overshadows the idea of having cache memory close to the processor, since a common cache will increase the average access time as compared to local cache. Thus, this scheme simply ignores cache coherence problem.

Considering the significance of performance, it is better to allow local caches in each processor and move towards more practical *software* & *hardware* solutions.

5.8.3. Software-based solutions:

The compiler is used to analyse the source code as the object code is generated in order to identify the parts of the program which uses shared items. These writable shared items are marked with a tag as *non-cacheable*, i.e., processors cannot write *non-cacheable* data into their local caches and have to access it directly from main memory for both read and write operations. A shared item can be identified by the processors using the tags associated with it. This is cheap to implement and can be achieved during the compilation process. However, it increases the average access time since during execution of an instruction, the processors have to access the main memory instead of their local caches. It is also an extra overhead on the software which also affects the system's performance. Please note that the program also uses non-sharable and read-only items, which are marked as *cacheable* i.e., these data are allowed to be stored in the local cache of the processor. Only non-cacheable items remain in main memory.

Stop to Consider

- ✓ The *all-is-well* approach is not a viable solution for cache coherence solution, since a common cache memory in multiprocessor system decreases performance.
- ✓ The *software-based* solutions for cache coherence problem are cheap but slow.
- ✓ The *hardware-based* solutions for cache coherence problems are costly but fast.

Space for learners:

5.8.4. Hardware Solutions:

Space for learners:

1. Cache Snooping Protocol:

Here, a bus controller is assigned to each processor, which monitors the write operations on the bus by other processors. This bus controller is known as *snoopy cache controller*. The snoopy cache controller is responsible to identify if a shared item is being modified by any processor and ensures that all other cache controllers have the most recent updated copy of the shared item to avoid the usage of outdated information from their caches. There are two methods as discussed below, which can either be followed as a snooping cache protocol.

a) *Write-update protocol (or Write-broadcast protocol)*:
In this protocol, whenever a processor writes to a shared item in its cache, it broadcasts to all the other cache controllers about the updated value of the shared item through the system bus. All the cache controllers update their local cache accordingly. This scheme makes the update value readily available in caches of other processors, thus consumes more bandwidth in terms of memory. A solution to this over consumption of memory bandwidth is to keep tracks of the shared items to avoid unnecessary re-broadcasts. An example of write-update protocol is Firefly Protocol, which is used by SPARC center 2000.

b) Write-invalidate protocol:
In this protocol, whenever a processor (let's say P1) writes into a shared item (word) in its cache, it informs all the other cache controller about the location (let's say 3000) of the updated word in its cache. All the cache controllers' snoops on the bus for write operation. They will check if they have a copy of the word which has been overwritten by P1. If yes, then they mark the location of that word in their cache as *invalid* for future reference and *removes* the word from their caches. Afterwards, whenever another processor (let's say P2) tries to access the invalid word (which was a copy of the word from location 3000), it will result in a cache miss and any one of the following operations will

i. If the cache follows *write-through* policy, then the updated item will be transferred to processor P2 from

the main memory. Here, the updated item will be available in both - cache memory of processor P1 and main memory, but main memory is the preferred choice in an event of cache miss.

ii. If the cache follows *write-back* policy, then the updated item will be transferred from the cache memory of Processor P1 to Processor P2 via main memory, since at any time, the latest value of the word will only be available in cache of P1 and will be updated in main memory later.

An example of write-invalidate protocol is MESIprotocol (Modified Exclusive Shared Invalid), which is used byIntel Pentium 4 and PowerPC.

Space for learners:

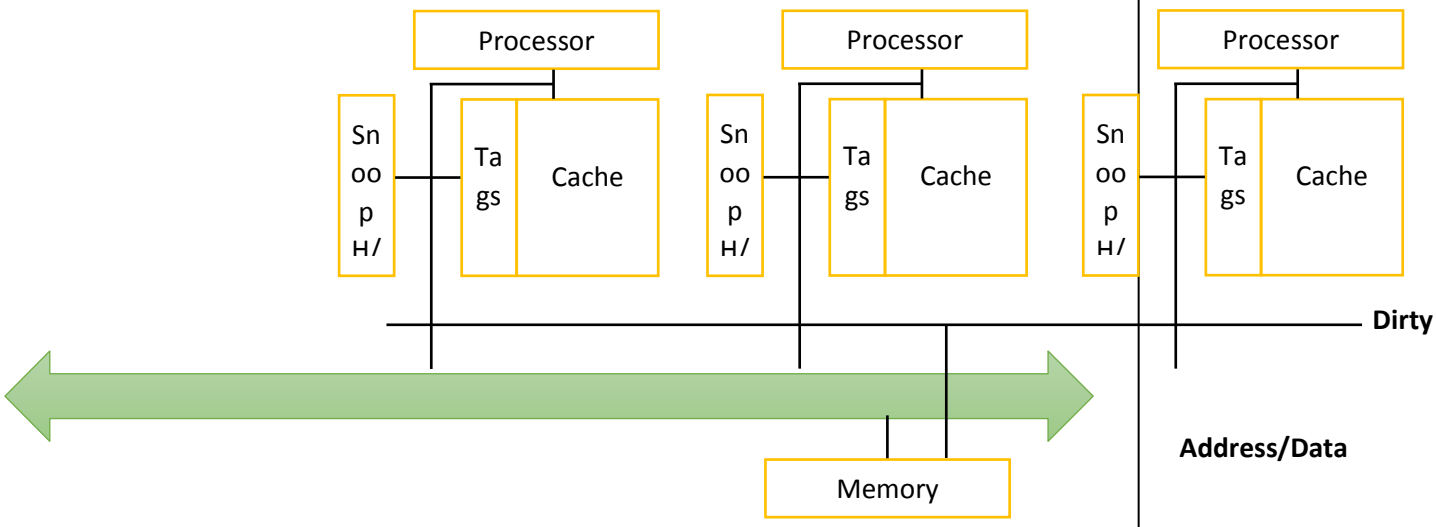


Fig.16 Cache Snooping Protocol

2. Directory Protocol:

Here, a centralized approach is considered by maintaining a*directory* in the main memory. We define one directory per cache to keep track of state (or information) of every block of main memory present in that cache. In other words, the information in a directory is about the cache memories of processors containing same block from main memory and the state of the block - either *valid* or

invalid.In order to prevent bottleneck in a directory, the entries in the directory can be distributed.

Whenever an information in the cache is modified by a processor, it the responsibility of the directory controller to check the directory and identify the affected processors. Then the affected processor receives an explicit information from the directory controller about the appropriate action to be taken in order to avoid any incoherency in cache.

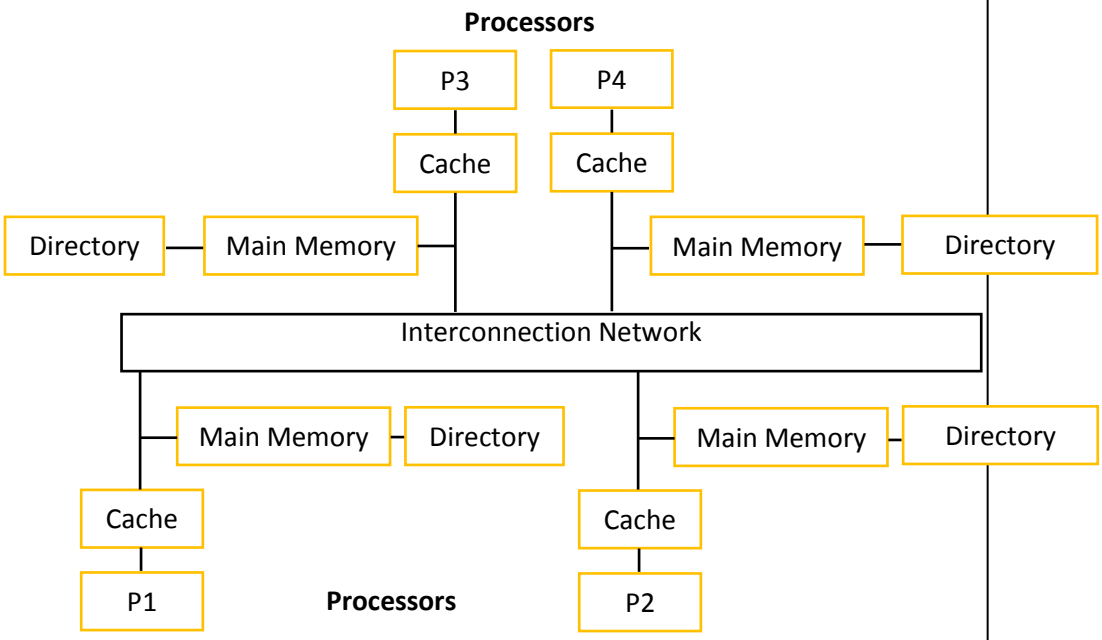


Fig 17. Distributed Directory Protocol

SAQ

1. What is Cache Coherence? How does it differ from Cache Coherence Problem?
2. State the working of Cache Snooping Protocol.
3. State the working of Directory Protocol.

5.9 SUMMING UP

- Instruction Set Architecture (ISA) defines a basic set of operations - like arithmetic, logical, branching and memory operations, that must be performed by the system and also

Space for learners:

provides details about how a machine code doesn't depend on the prime characteristics of the implementation of a particular ISA. Based on architectural complexity, ISA can be classified into CISC and RISC.

- CISC stands for Complex Instruction Set Computer. This approach attempts to reduce the number of instructions per program. In order to do so, the number of cycles per instruction increases. CISC takes several clock cycles to execute instruction. In CISC architecture, the instructions are of variable lengths (from 8-bits to 120-bits)
- RISC stands for Reduced Instruction Set Computer. This approach is needed to minimize the cycles per instruction. In order to do so, the number of instructions per program increases. RISC takes single clock cycle to execute an instruction In RISC architecture, the instructions are of fixed lengths (32-bits)
- In modern times, almost all CISC & RISC processors are superscalar in nature. Superscalar is an implementation for ILP processor architectures in which programs doesn't have any explicit information about parallel execution of instruction and it is the responsibility of the system hardware to detect and construct action plans for any ILPs to be exploited for parallelism.
- VLIW processors are built on an architecture in which programs contain explicit information about parallelism and it is the responsibility of the software, called compiler to identify and communicate it to the hardware by specifying all the independent operations.
- In VLIW Processor, Instruction consists of multiple independent operations grouped together. There are multiple independent functional units. Each operation in the instruction is assigned to different functional units. All functional units share the use of a common large register file.
- One VLIW instruction encodes at least one operation for each functional unit on each cycle. So, length of the instruction increases with the number of functional units. These operations are assigned to functional units by the position in the given fields within the long instruction word. This is known as slotting.
- EPIC stands for Explicitly Parallel Instruction Computing and is implemented by Hewlett Packard & Intel as Intel

Space for learners:

Itanium architecture (IA-64). EPIC is a mix of software & hardware, incorporating the advantages of both superscalars and VLIW architectures.

- Like VLIW, EPIC it permits execution of instructions in parallel using a compiler. However, in EPIC apart from identifying and grouping the independent operation in a single instruction, the compiler communicates this via explicit information in the instruction set. That's why EPIC is also known as "independence architecture".
- Unlike VLIW, EPIC retains backward compatibility across different implementations like superscalars, but doesn't need any hardware for dependency checks like superscalars.
- A multiprocessor system is a computer system with more than one processor (typically two or more), where each processor is linked with one another via interconnection network. The focus of a multiprocessor system is to achieve parallel processing, Fault Tolerance, graceful degradation, scalability and modular growth. Multiprocessor system falls under MIMD architecture (Multiple Instruction stream, Multiple Data stream). They are primarily divided into two-types: tightly coupled system and loosely coupled system.
- A tightly coupled multiprocessor, also known as shared memory multiprocessor system, share information between multiple processors via a shared or global memory. Example of tightly-coupled multiprocessor system - Unform Memory Access (UMA) and Non-Uniform Memory Access (NUMA). Symmetric Multiprocessor (SMP) system is an UMA multi-processor system with identical, homogenous processors, which are capable of performing similar functions and utilizes a centralised shared main memory.
- A loosely coupled multiprocessor system, also known as the distributed memory multi-processor system, doesn't share information between multiple processors via a shared memory, since each processor has its local dedicated memory, which together forms a distributed memory. Example of loosely-coupled multiprocessor system - Clusters. A cluster consists of a set of computers connected over a local area network (LAN) which function as a single large multiprocessor.
- In multiprocessor systems, the components like CPU and I/O Ports are connected to I/O devices and a memory unit, which

Space for learners:

can be shared or distributed in nature. The interconnection between the components be of different physical configurations - Time-Shared Common Bus, Multiport Memory, Crossbar Switch, Multistage Switching Network, Hypercube Network

- In time-shared common bus structure, all the processors in the microprocessor system are connected to shared memory and other common resources using a common interconnecting path, called as common system bus. Only one processor at a time can communicate over the bus. The design is simple due to the use of single common system bus. It is a cheap and affordable structure. Since only one processor at a time can transfer or communicate over the system bus, the communication is quite slow.
- In multiport memory structure, the system has separate buses between each memory module and the processors. Each of the memory module has an priority logic to resolve conflict of simultaneous requests from multiple processors. A fixed priority is assigned to each memory ports to avoid memory access conflicts. Due to multiple paths between the processors and memory modules, multiple processors can simultaneously access the memory with high transfer rate. But it is expensive in cost due to huge interconnecting cables requirements.
- In crossbar switch structure, a number of crosspoints are placed at the intersection of memory paths and processor buses. At each crosspoint, there is a control logic to set the desired path between a memory module and a processor. This control logic is basically a switch, which is an electronic circuit. A switch can also resolve the conflict of simultaneous requests from multiple processors to access same memory module in the system based on a fixed priority basis.
- In multistage switching network structure, we use a switch which can interchange two-inputs, two-outputs to determine the path between multiple processors and multiple memory modules. Hence the name, multistage switching network since it allows to build different possible stages for different combination of inputs & outputs. A very popular topology is called omega switching network which allows exactly one path from each source to any particular destination. The

Space for learners:

structure is cost effective since we can connect multiple sources to multiple destinations with less amount of wiring. But there is a restriction on the number of simultaneous connections to two destinations connected to same switch is prohibited.

- In hypercube structure, a loosely coupled system comprised of N = 2n numbers of processors are interconnected to each other in a N-dimensional cube. A node of the cube is represented by a processor and an edge of the cube is a communication path connection two nodes. The advantage of hypercube is that is easy to scale the current network to higher configurations and intelligent communication protocols could be easily implemented. However, the multiple paths between processors increases the routing complexity.
- Cache coherence is a condition which states that all the cache lines with a particular shared main memory block must contain same information at any given point of time. This ensures that a multiprocessor system can perform memory operation correctly, by keeping identical multiple copies of information in the caches of the processors involved in execution of a particular instruction.
- Cache coherence problem occurs when cache coherence is not maintained, i.e., a processor updates its cache and other processors doesn't get an updated copy of newly modified data in their cache. This hampers the uniformity of data in all the caches of processors. Cache coherence problem happens in a multiprocessor system, since multiple processor access and works on non-identical multiple copies of data.
- All-is-well approach One of simple scheme can be to restrict the association of local caches for each processor and force them to use a shared cache memory. This scheme simply ignores cache coherence problem and moreover increases average access time.
- In software-based solution, the compiler is used to mark data as cacheable and non-cacheable. The cacheable items are allowed to be stored in the local cache of the processor but the non-cacheable items can't be stored in cache and remain in main memory. All the non-sharable & read-only items are

Space for learners:

tagged as cacheable and the writable shared items are tagged as non-cacheable.

- The cache snooping protocol is a hardware-based solution. Here a bus controller is assigned to each processor, which monitors the write operations on the bus by other processors. This bus controller is known as snoopy cache controller. The snoopy cache controller is responsible to identify if a shared item is being modified by any processor and ensures that all other cache controllers have the most recent updated copy of the shared item to avoid the usage of outdated information from their caches. There is two ways to implement this protocol - write-update protocol (or Write-broadcast protocol) and write-invalidate protocol.
- In write-update/write-broadcast protocol, whenever a processor writes to a shared item in its cache, it broadcasts all the other cache controllers about the updated value of the shared item through the system bus. All the cache controllers update their local cache accordingly.
- In write-invalidate protocol, whenever a processor writes into a shared word in its cache, it informs all the other cache controller about the location of the updated word in its cache. All the cache controllers' checks if they have a copy of that old word. If yes, then they mark the location of that word in their cache as invalid for future reference and removes the word from their caches. Afterwards, whenever another processor tries to access the invalid word, there will be cache miss and actions will be taken depending on whether write-back or write-through policy is followed.
- Directory Protocol is also a hardware solution for cache coherence problem. Here, a centralized approach is considered by maintaining a directory in the main memory. We define one directory per cache to keep track of state (either valid or invalid) of every block of main memory present in that cache. The entries in the directory can be distributed. A central memory controller checks the directory to find affected processors in case of any modification of shared data in its cache by a processor and sends explicit instruction to the affected processors to avoid cache incoherence.

Space for learners:

5.10 ANSWER TO CHECK YOUR PROGRESS

1. Instruction Set Architecture
2. The set of operation defined by instruction set architecture may include arithmetic, logical, branching and memory operations.
3. RISC & CISC
4. Reduced Instruction Set Computer
5. Example of CISC: Intel x86, Example of RISC: ARM
6. False
7. True
8. True
9. Explicitly Parallel Instruction Computing
10. Intel Itanium
11. Compiler
12. Hewlett Packard (HP) & Intel
13. Predicated
14. False
15. False
16. Multiple Instruction stream, Multiple Data stream
17. Tightly-coupled, Loosely-coupled
18. 2^n
19. True
20. True

5.11 POSSIBLE QUESTIONS

1. What is instruction-set architecture? Why is it important?
2. Explain the different types of ISAs.
3. Define Instruction-level parallelism (ILP). How VLIW takes advantage of ILP?
4. Explain the VLIW architecture and the instruction format.
5. State the advantages, disadvantages and applications of VLIW architecture.
6. What is EPIC? How does it differ from VLIW?
7. Write a short note of EPIC architecture.
8. Explain in brief about multiprocessor system. How does it differ from multicomputer system?
9. Differentiate between tightly-coupled and loosely-coupled microprocessor

Space for learners:

10. How does uniform memory access differ from non-uniform memory access?
11. Explain in brief about different interconnection structures in multiprocessor systems.
12. What is the difference between the cross-switch and multistage-switch?
13. What are the two widely-used policies of cache write operation?
14. What is the software-based approach to solve the cache coherence problem?
15. Write a short note of hardware-based solutions for cache coherence problem.

Space for learners:

5.12 REFERENCES AND SUGGESTED READINGS

1. Mano, M. Morris. *Computer System Architecture, 3E.* Pearson Education India, 2007.
2. Govindarajalu, B. *Comp Arch and Org, 2E*. Tata McGraw-Hill Education, 2010.
3. Hamacher, V. Carl, Zvonko G. Vranesic, and Safwat G. Zaky. *Computer organization and Embedded Systems, 6E*. McGraw-Hill, Inc., 2012.
4. Al-Hothali, Samaher. "Snoopy and directory-based cache coherence protocols: A critical analysis." *Journal of Information & Communication Technology (JICT)* 4.1 (2010): 11.
5. Semiconductors, Philips. "An introduction to very-long instruction word (VLIW) computer architecture." *Philips Semiconductors* (1997).
6. Smotherman, Mark. "Understanding EPIC architectures and implementations." *40th Annual Southeast ACM Conference*. 2002.
7. Halfhill, Tom R. "VLIW Microprocessors" *Computerworld India*, 14 Feb. 2000, https://www.computerworld.com/article/2593626/vliw-microprocessors.html.
8. "Instruction set architecture" *Wikipedia*, https://en.wikipedia.org/wiki/Instruction_set_architecture. Accessed 01 Aug. 2021.

9. “Very long instruction word” *Wikipedia*, https://en.wikipedia.org/wiki/Very_long_instruction_word. Accessed 01 Aug. 2021.
10. “Explicitly Parallel Instruction Computing" *Wikipedia*, https://en.wikipedia.org/wiki/Explicitly_parallel_instruction_computing. Accessed 01 Aug. 2021.
11. Zaccone, Giancarlo. *Python parallel programming cookbook*. Packt Publishing Ltd, 2015.
12. Beckmann, Nathan. “Static Scheduling & VLIW.” *Carnegie Mellon University*, https://www.cs.cmu.edu/afs/cs/academic/class/15740-s17/www/lectures/13-static-scheduling.pdf. Accessed 02 Aug. 2021
13. Shanthi, A. P. “Multiple Issue Processors II” Univeristy of Maryland,https://www.cs.umd.edu/~meesh/411/CA-online/chapter/multiple-issue-processors-ii/index.html. Accessed 01 Aug. 2021.
14. "VLIW Processors" Slideshare, https://www.slideshare.net/shudhanshu29/vliw-processors. Accessed 01 Aug. 2021.
15. Schlansker, Michael S., and B. Ramakrishna Rau. *EPIC: An architecture for instruction-level parallel processors*. Hewlett-Packard Laboratories, 2000.

---×---

Space for learners: