Computer Architecture

A note according to Pokhara University's course syllabus

Compiled By: Deepak Kumar Karn

©2013 http://deepakkarn.wordpress.com/ All rights reserved

About this note:

This note covers almost all topics of Computer Architecture's Course Syllabus according to Pokhara University, Nepal.

This note may not give you the detail Knowledge of Computer Architecture but will help you in Exams.

I'm not writer of this note book. I just compiled it from different resources.

References:

Teachers:

[1]Assistant Professor Nitu Bharti, Nepal Engineering College, Nepal

[2]Assistant Professor Dayaram Budhathoki, Nepal Engineering College, Nepal

Books:

[1]Computer System Architecture (3rd Ed) by M Morris Mano
[2] Computer Organization and Architecture by William Stallings

And: Google search (www.google.com)

Computer Architecture By Deepak Kr. Karn

Contents **Pages**

Computer Architecture By Deepak Kr. Karn

Chapter 1 :- Introduction

1.) Expanded Structure of IAS computer:-

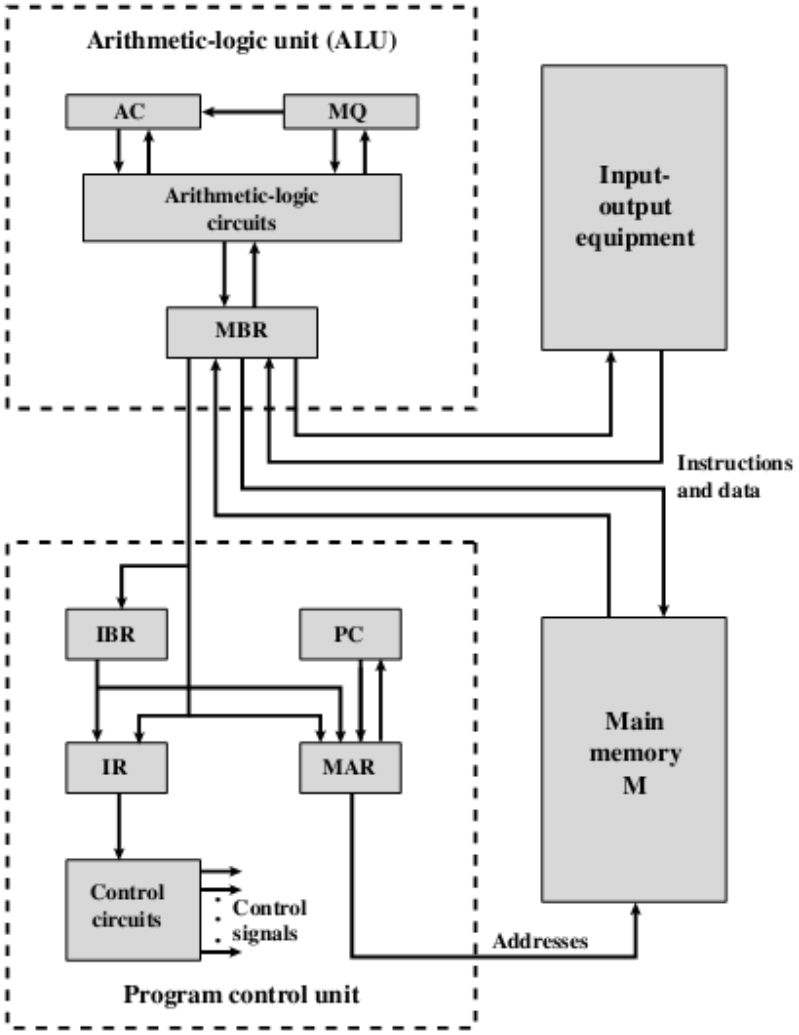


Figure :- Expanded structure of IAS computer

Memory buffer register (MBR):-

Contains a word to be stored in memory or sent to the I/O unit, or is used to receive a word from memory or from the I/O unit.

Memory address register (MAR) :-

Specifies the address of memory in the word to be written from or read into the MBR.

Instruction register (IR) :-

Contains the 8 bits Opcode instruction being executed.

Instruction buffer register (IBR):-

Employed to hold temporarily the right- hand instruction from a word in memory.

Program counter (PC):-

Contains the address of the next instruction-pair to be fetched from memory.

Accumulator (AC) and multiplier quotient (MQ):-

Employed to hold temporarily operands and results of ALU operations. For example, the result of multiplying two 40-bit numbers is an 80-bit number; the most significant 40 bits are stored in the AC and the least significant in the MQ.

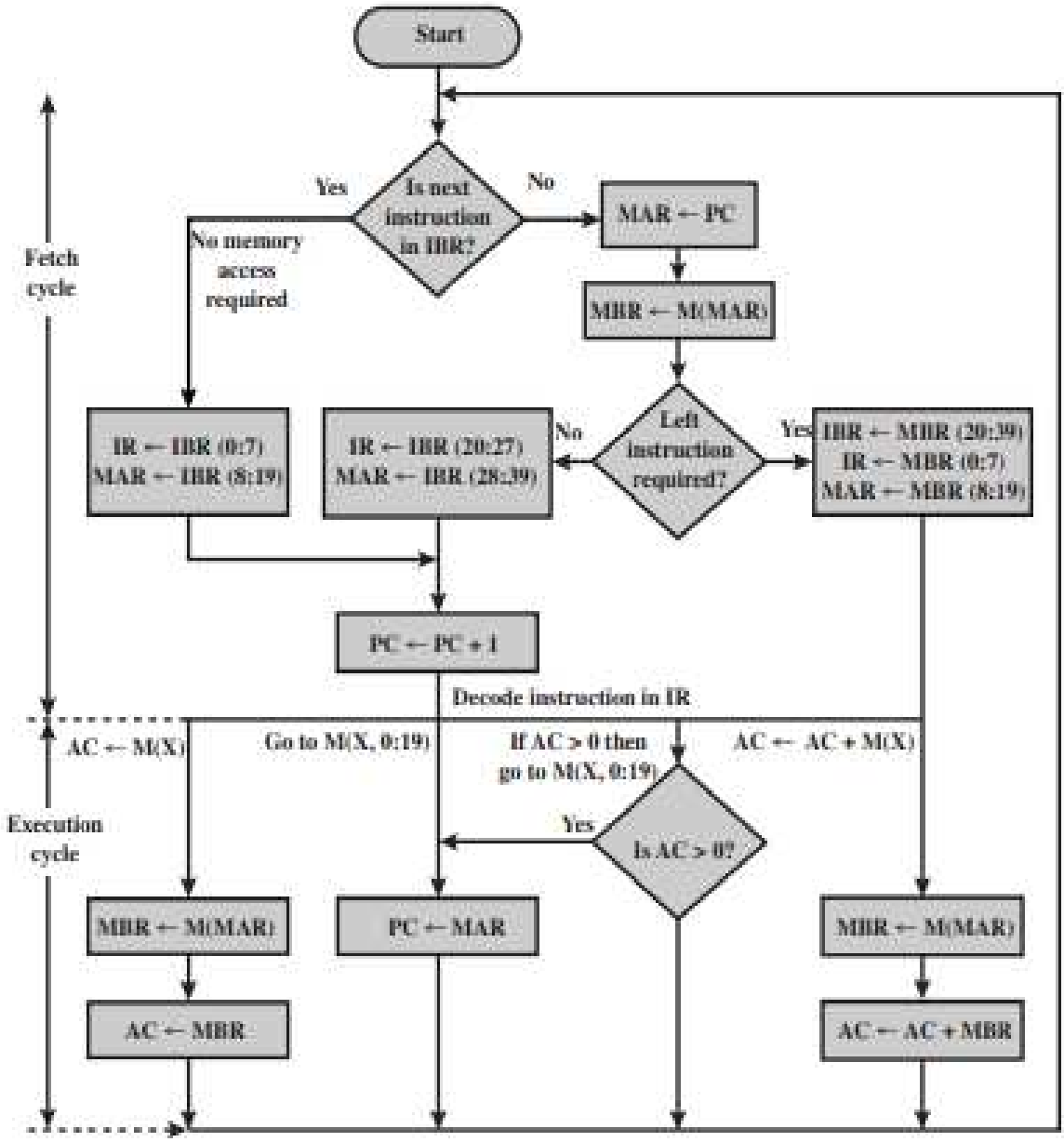
Central processing unit (CPU):-

Controls the operation of the computer and performs its data processing functions; often simply referred to as processor.

I/O:-

Moves data between the computer and its external environment.

2.) Flow chart of IAS Computer:-



M(X) = contents of memory location whose address is X
(i:j) = bits i through j

Figure 2.4 Partial Flowchart of IAS Operation

The IAs operates by respectively performing as instruction cycle. Each instruction cycle consists of two sub-cycles.

Fetch Cycle:-

The opcode of next instruction is loaded into the IR and the address portion is loaded into the MAR. This instruction may be taken from the IBR, or it can be obtained from memory by loading a word into the MBR, and then down to the IBR, IR and MAR.

Execute Cycle:-

The control circuitry interprets the Opcode & executes the instruction by sending out the appropriate control signals to cause data to be moved or an operation to be performed by the ALU.

3.)IAS instruction format:-

The memory of IAS consists of 1000 storage locations called words of 40 binary digits each. Both data & instructin are stored there, Number are represented in binary form.

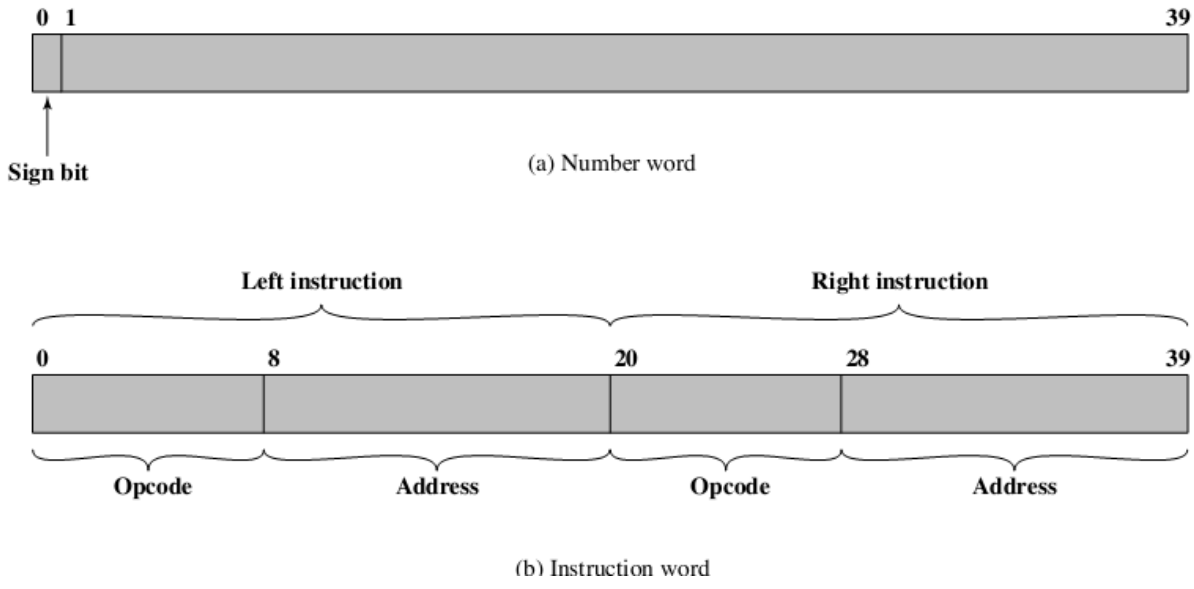


Fig. IAS Format

Each number is represented by a sign bit and a 39 bit value. A word may also contain 20 bit instruction consisting of an 8 bit operation code(opcode) specifying the operation to be performed and 12 bit address designing one of the words in memory (numbered from 0 to 999)

IAS computer had a total of 21 instruction that can be grouped as follows.

1. Data transfer:-

Move data between memory and ALU register or between two ALU registers.

2. Unconditional branch:-

The control unit executes instruction in sequence from memory. There sequence can be changed by a branch instruction which facilitates repetitive operations.

3. Conditional branch:-

Branch can be made dependent on a condition thus allowing decision part.

4. Arithmetic:-

It comprises operation performed by ALU

5. Address modify:-

It permits address to be computed in ALU and then inserted into instruction stored in memory.

4.) The difference between computer Organization and Computer Architecture:-

Computer organization

- Deals with all physical components of computer systems that interacts with each other to perform various functionalities
- The lower level of computer organization is known as micro-architecture which is more detailed and concrete.
- Examples of Organizational attributes includes Hardware details transparent to the programmer such as control signal and peripheral.

Computer architecture

- Refers as a set of attributes of a system as seen by programmer
- Examples of the Architectural attributes include the instruction set, the no of bits used to represent the data types, Input Output mechanism and technique for addressing memories.

The difference between architecture and organization is best described by a non-computer example. Is the gear level in a motorcycle part of it is architecture or organization? The architecture of a motorcycle is simple; it transports you from A to B. The gear level belongs to the motorcycle's organization because it implements the function of a motorcycle but is not part of that function

5.)Scale Of Integration:-

The number of components fitted into a standard size IC represents its integration scale, in other words it's a density of components. It is classified as follows:

→ SSI – Small Scale Integration

It have less than 100 components (about 10 gates).

→MSI – Medium Scale Integration

It contains less than 500 components or have more than 10 but less than 100 gates.

→ LSI – Large Scale Integration

Here number of components is between 500 and 300000 or have more than 100 gates.

→ VLSI – Very Large Scale Integration

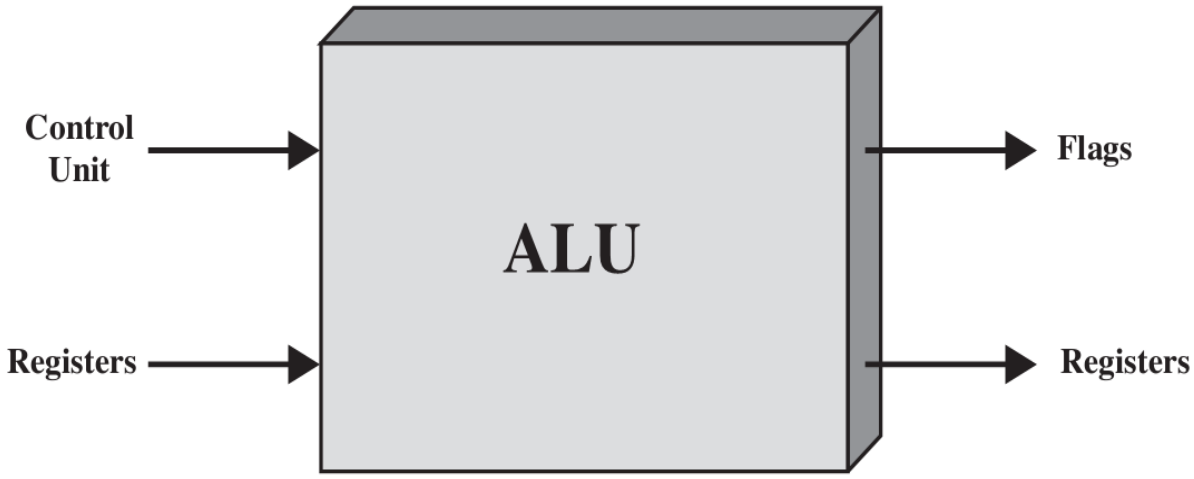
It contains more than 300000 components per chip

→VVLSI - Very Very Large Scale Integration

It contains more than 1500000 components per chip.

Chapter2:- Central Processing Unit

1.) ALU:-



The ALU is the part of computer that actually performs arithmetic & logical operations. All of other elements in the computer system, control unit, register, memory, I/O are there mainly to bring data into ALU for it to process and then to take results back.

2.)Sign-magnitude representation:-

→The most significant(left side) bit in the word as a sign bit.

Sign bit 0→ +Ve number

1→ -ve number

Eg.

+18 → 00010010

-18 → 10010010

Drawbacks:

Addition and subtraction require a consideration of both the sign of the numbers and their magnitude to carry out the required operation.

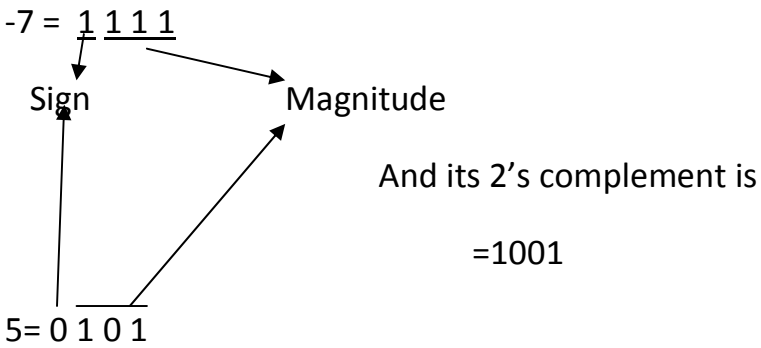
+0 = 0000 0000

-0= 1000 0000

This is inconvenient because it is slightly more difficult to test for 0, (which appears frequently on computers)

3.) Two's complement representation:-

Like sign magnitude, two's complement representation uses the most significant bit as a sign bit, making it easy to test whether it is +ve or – ve.



And its 2's complement is

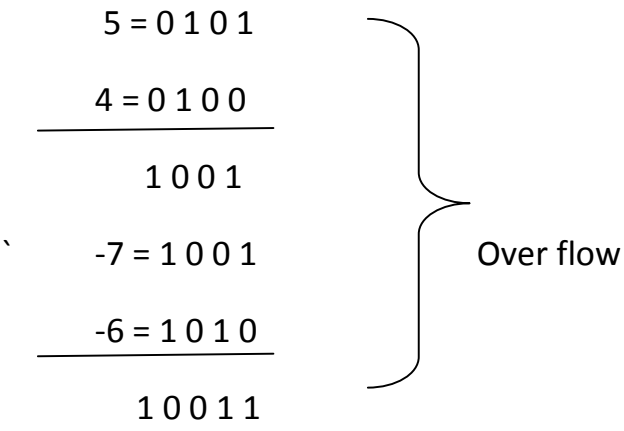
=0101

Addition:-

-7 = 1 0 0 1	-4 = 1 1 0 0
+ 5 = 0 1 0 1	+ 4 = 0 1 0 0
-2 = 1 1 1 0	-0= (1) 0 0 0 0

→The result may be larger that can be held in the word size being used. This condition is overflow.

→**Overflow rule**:- If two number are added, and they are both positive or negative than overflow occurs if and only if the result has opposite sign.



Subtraction:-

To subtract one number (subtrahend) from another number (minuend), take the 2's complement (negative) of the subtrahend and add it to the minuend.

M=2 s= 7

-7 = 2's complement of 0111

= 1000

\+ 1

1001

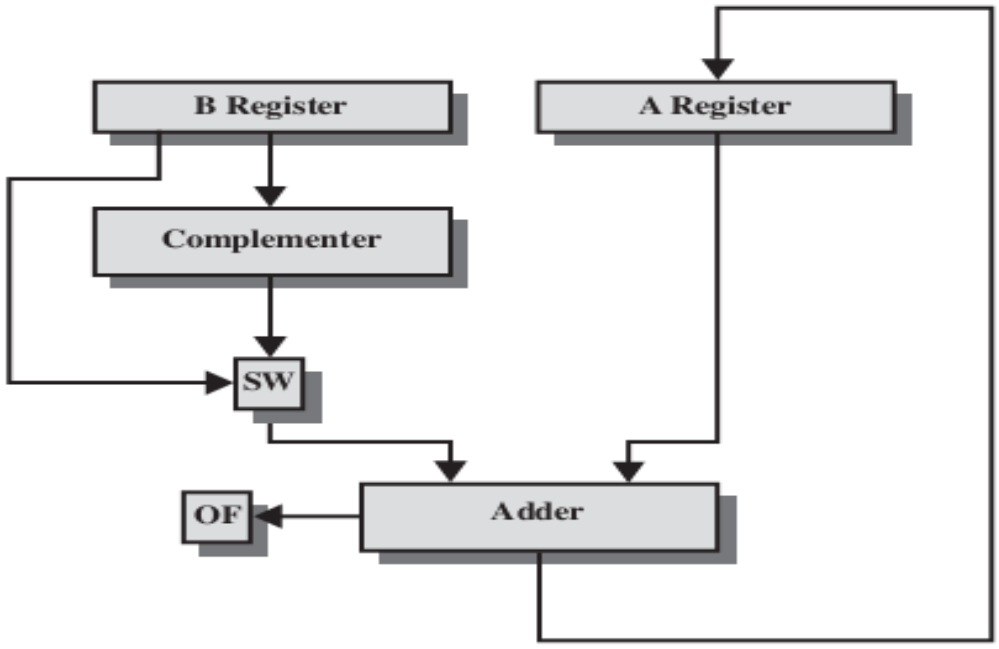
Therefore, 2-7 = 2+ (-7)

0010

1001

1011

Block Diagram of H/W Addition Subtraction:



OF = overflow bit
SW = Switch (select addition or subtraction)

Q. Explain the algorithm for signed magnitude number addition/subtraction with suitable examples.

Ans.:-

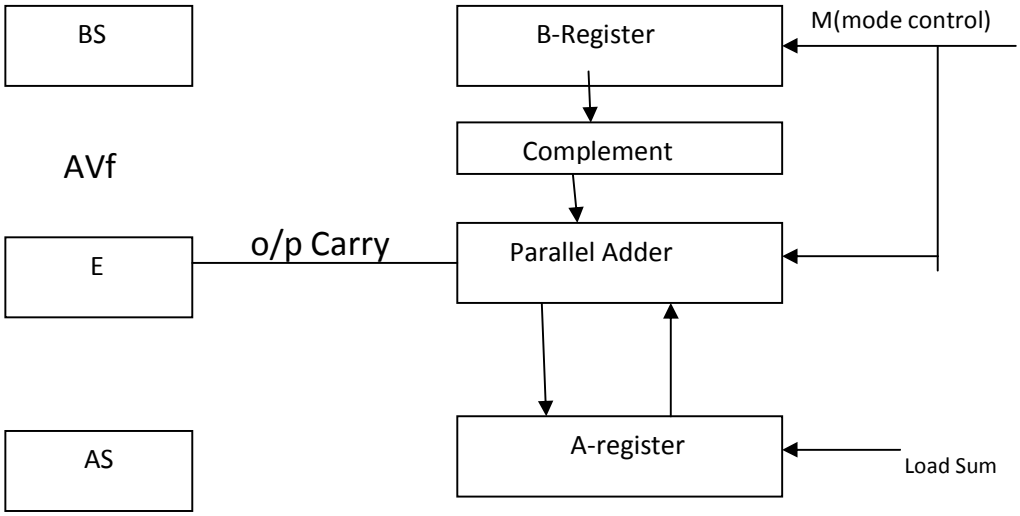


Fig: Hardware for signed magnitude addition & subtraction.

→Above block diagram for addition and subtraction consist of register A&B and sign flip-flop AS & BS subtraction is done by adding A to the 2's complement of B.

→The o/p carry is transfer flip-flop E where it can be checked to determine the relative magnitude of two magnitude.

→The add overflow flip-flop(AOF) holds the overflow when A & B are added.

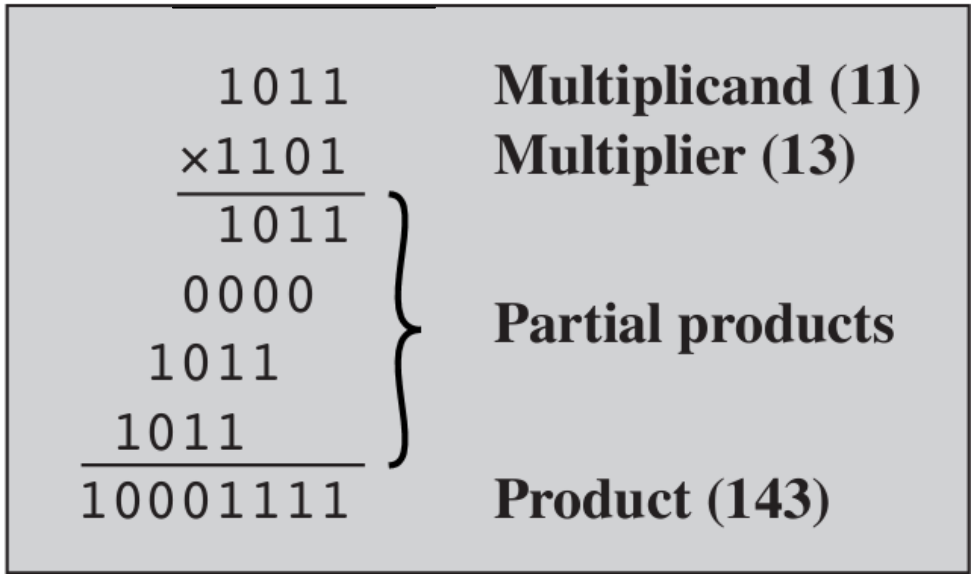
→The A register provides other micro operation that may be needed when we specify sequence of steps.

→The addition of A+B is done through the parallel adder.

→The complement provides the o/p of B or the complement of B depending on the state of mode control M. The M signal also applied to the i/p carry of the adder when M=0, the value of B is transfer to the adder, the i/p carry is zero & the o/p of the adder is equal to sum A+B.

→When M=1, the one's complement of B is applied to the adder, the i/p carry is I & O/P = A+B+I which is equal to A+2's complement of B. Again which is equivalent to the subtraction A-B.

Multiplication:

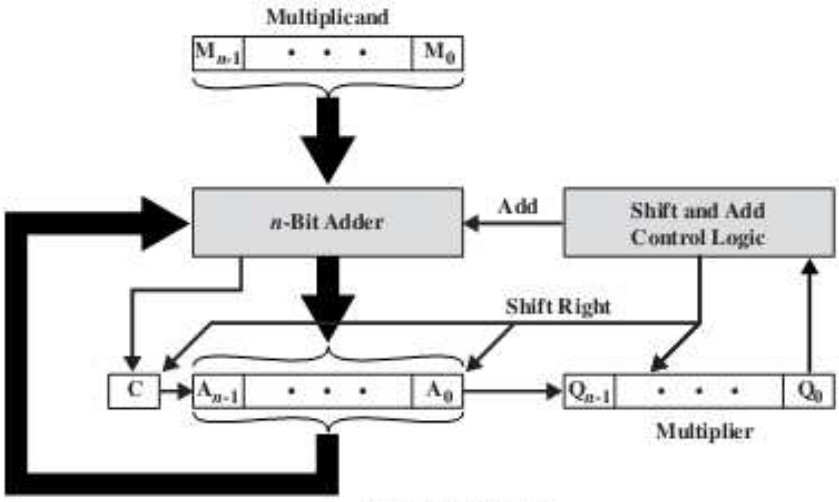


Example1:- Multiplication Of Unsigned Binary Integers

$(1011)_2 * (1101)_2 = (10001111)_2$

(11*13) = (143)

Block Diagram of multiplication:-



(a) Block Diagram

C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second Cycle
0	1101	1111	1011	Add	Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth Cycle
0	1000	1111	1011	Shift	

(b) Above Example 1

Fig. Hardware Implementation of Unsigned Binary Multiplication

Another Example of Multiplication:-

1010*1100

	C	A	Q	M	
	0	0000	1010	1100	
Since,Q0=0	0	0000	0101	1100 → 1st Cycle	
Since,Q0=1		C,A ← A+M			
	0	1100	0101	1100—add	
	0	0110	0010	1100—shift	2nd Cycle
Since,Q0=0					
	0	0011	0001	1100—shift	3rd Cycle
Since,Q0=1		C,A ← A+M			
	0	1111	0001	1100—add	4th Cycle
	0	0111	1000	1100—shift	

Answer = 0111 1000

Flowchart of Multiplication:-

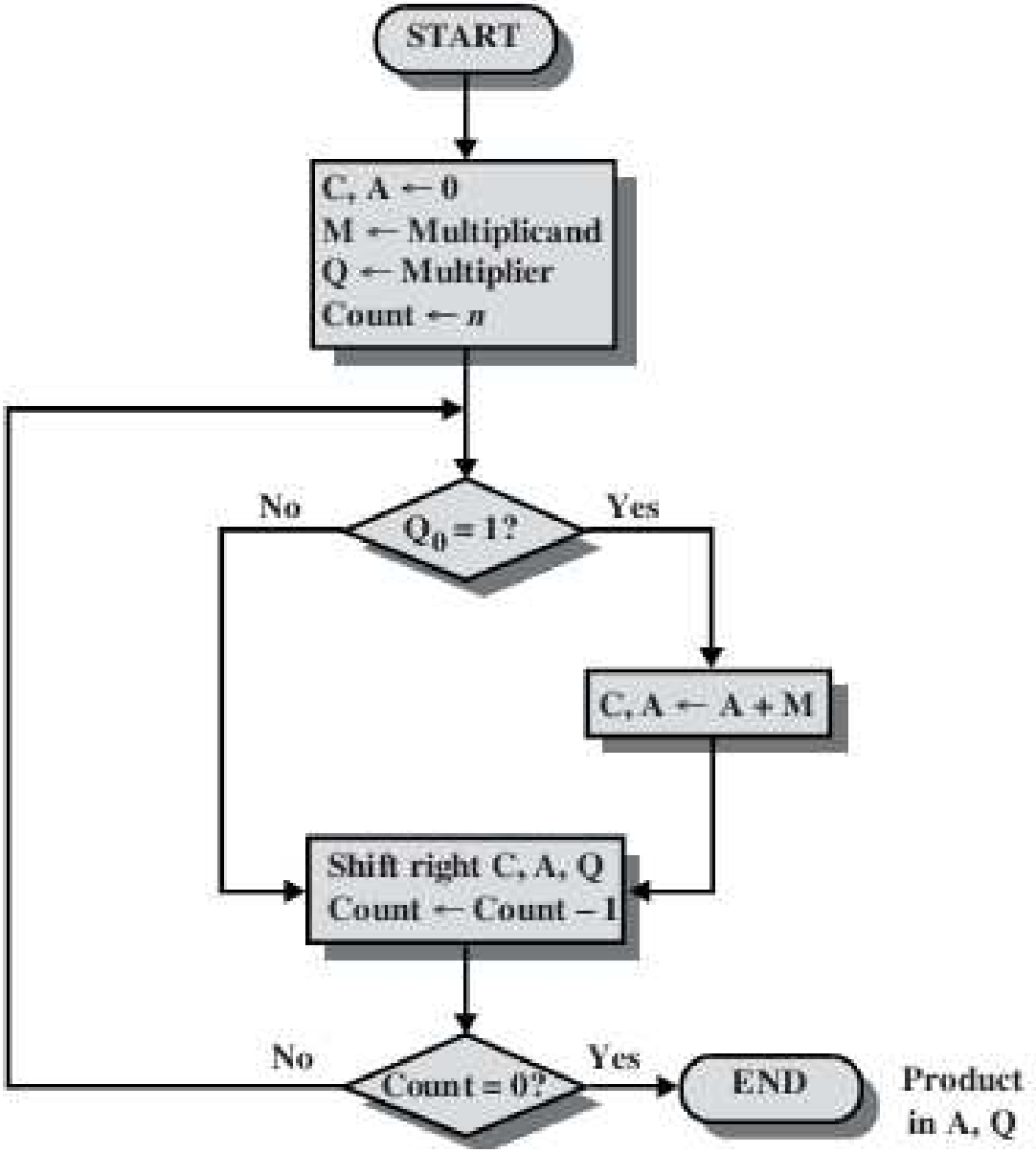


Fig. Flow chart of unsigned binary multiplication

#. Two's complement Multiplication Booth's Algorithm:-

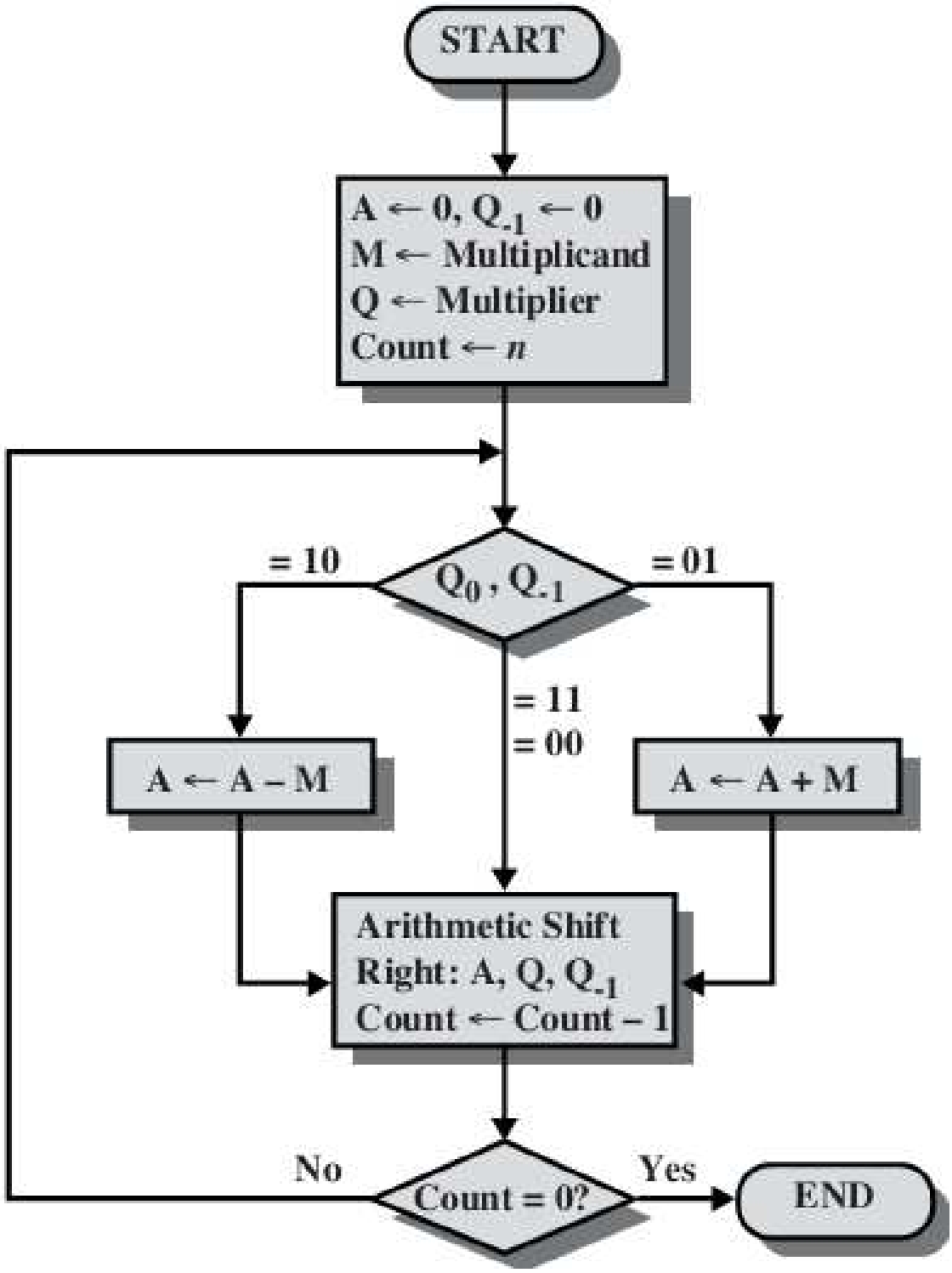


Fig. Booth's Algorithm for twos Complement Multiplication

Example:

Booth's Algorithm (7*3)

A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	A ← A - M	First Cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	Second Cycle
0101	0100	1	0111	A ← A + M	Third Cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	Fourth Cycle

→Since, QQ_{-1} =11

→Since, QQ_{-1} =01

→Since, QQ_{-1}=00

AQ= 0001 0101

Answer

Another Example:

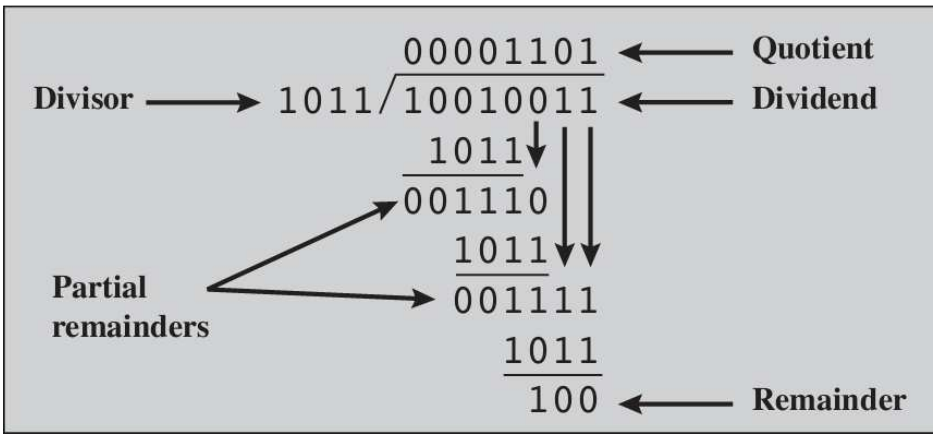
-7*3

	A	Q	Q_{-1}	M	
	000	0011	0	1001	
QQ_{-1}=10	A← A-M				
	0111	0011	0	1001	A←A-M
	0011	1001	1	1001	Shift
QQ_{-1} = 11					
	0001	1100	1	1001	Shift
QQ_{-1}=01	A←A+M				
	1010	1100	1	1001	
	1101	0110	0	1001	
QQ_{-1}=00					
	1110	1011	0	1001	

Note:

In either of the case, the right shift is such that the leftmost bit of A namely A_{n-1}, not only is shifted into A_{n-2} but also remains in A_{n-1}. This is required to preserve the sign of the number in A&Q. It is known as arithmetic shift it preserves sign bit.

Division: Division of unsigned binary division.



Flowchart of unsigned Binary division.

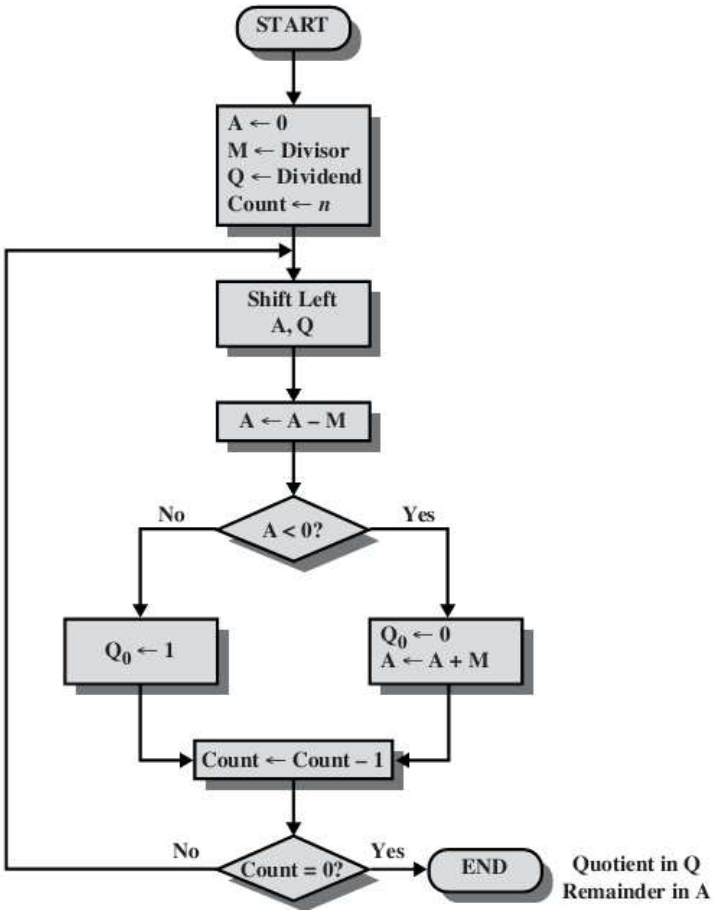


Fig. FLOWCHART UNSIGNED BINARY DIVISION

Two's complement division Algorithm:

1. M←Divisor

   A, Q←Dividend (expressed as 2n bits)

   For eg.

   0111 becomes 0000 0111

   1001 becomes 1111 1001

2. Shift A, Q left 1 Bit position.

3. If M & A has same sign perform

   A←A-M,

   Otherwise, A←A+M

4. The step 3 is successful if the sign of A is same before and after the operation.

   a. If Successful or A=0, then set Q=1

   b. If unsuccessful and A≠0, set Q←0 & restore previous value of A.

5. Repeat step 2 through 4 for 'n' times.

6. The reminder is in A. If the sign of divisor & dividend are same quotient is in Q. Otherwise quotient is 2's complement of Q.

D=Q*V+R

Where, D=Dividend V= Divisor

Q= Quotient R=Reminder.

Examples of twos complement Division

A	Q	M = 0011
0000	0111	Initial value
0000	1110	shift
1101		subtract
0000	1110	restore
0001	1100	shift
1110		subtract
0001	1100	restore
0011	1000	shift
0000		subtract
0000	1001	set $Q_0 = 1$
0001	0010	shift
1110		subtract
0001	0010	restore

(a) (7)/(3)

A	Q	M = 1101
0000	0111	Initial value
0000	1110	shift
1101		add
0000	1110	restore
0001	1100	shift
1110		add
0001	1100	restore
0011	1000	shift
0000		add
0000	1001	set $Q_0 = 1$
0001	0010	shift
1110		add
0001	0010	restore

(b) (7)/(–3)

A	Q	M = 0011
1111	1001	Initial value
1111	0010	shift
0010		add
1111	0010	restore
1110	0100	shift
0001		add
1110	0100	restore
1100	1000	shift
1111		add
1111	1001	set $Q_0 = 1$
1111	0010	shift
0010		add
1111	0010	restore

(c) (–7)/(3)

A	Q	M = 1101
1111	1001	Initial value
1111	0010	shift
0010		subtract
1111	0010	restore
1110	0100	shift
0001		subtract
1110	0100	restore
1100	1000	shift
1111		subtract
1111	1001	set $Q_0 = 1$
1111	0010	shift
0010		subtract
1111	0010	restore

(d) (–7)/(–3)

4.)Floating Points:

Floating point numbers: Two Uses:

1. Real no. with non zero fractions

   3.1416 , $6.62*10^{-23}$

2. Really high number

   $3*10^{8}$, $6.02*10^{23}$ ← Exponent

   Mantissa Radix

Floating point representation.

(a) Format

→+/- .significand x $2^{exponent}$

→Misnomer

→Point is actually fixed between sign bit and body of mantissa

→Exponent indicates place value (point position)

examples

$1.1010001 \times 2^{10100} = 0\ 10010011\ 10100010000000000000000 = 1.6328125 \times 2^{20}$

$-1.1010001 \times 2^{10100} = 1\ 10010011\ 10100010000000000000000 = -1.6328125 \times 2^{20}$

$1.1010001 \times 2^{-10100} = 0\ 01101011\ 10100010000000000000000 = 1.6328125 \times 2^{-20}$

$-1.1010001 \times 2^{-10100} = 1\ 01101011\ 10100010000000000000000 = -1.6328125 \times 2^{-20}$

#Addition and subtraction:

Four phases of the algorithm for addition and subtraction.

1. Check for zeros.

→Addition ~ subtraction except sign change.

→If either operand is 0, the other is reported as result.

2. Significand Alignment

→Manipulate numbers so that two exponent are equel.

Eg:- $(123 * 10^0) + (456 * 10^{-2})$

$= (123 * 10^0) + (4.56 * 10^0)$

$=127.56 * 10^0$

→Alignment may be achieved by shifting either smaller number to right (increasing its exponent) or shifting the larger number to left.

3. Addition

→Significands added taking account their sign bit also.

→Possibility of significand overflow by 1 digit. If so, The Significand of result is shifted right and exponent is incremented

4. Normalization.

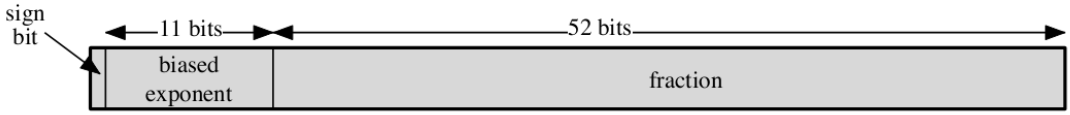
→ Significand digits are shifted left until MSD is non-zero.

→Each shift causes a decrement of the exponent and thus could cause exponent underflow.

→Finally the result must be rounded off and then reported.

IEEE 754 standard format for floating point.

(a) Single format



(b) Double format

Example : Represent $(47.625)_{10}$ *in Single Precision Floating Point (SPFP) format?*

Step:1

47=101111

0.625*2=1.250 →1

0.25*2=0.5 →0

0.5*2=1 →1

$(47.625)_{10}$= 101111.101

Step:2

1.01111101*2^5 (Normalization form)

=1.01111101*2^5

Step:3

Real Exp= 5+127 ← Biased

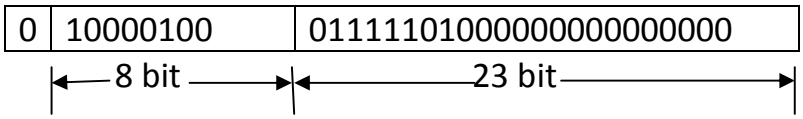
=132

Binary of 132

128	64	32	16	8	4	2	1
1	0	0	0	0	1	0	0

132=1000100

Sign=+ve



5.) Flowchart of floating point addition/subtraction

The floating point addition/subtraction are more complex than multiplication and division.

This is because of the need for alignment. There are four basic phases of the algorithm for addition and subtraction. These phases are clearly shown in below figure (flowchart). Namely

1. Check for zeros
2. Align for significands
3. Add or subtract the Significands
4. Normalization the result

The first phase is zero check. Because addition and subtraction are identical except for a sign change, the process begins by the subtratend if it is a subtract operation. Next, if either operand is 0, the other reported as the result.

The second phase is significand alignment. The numbers are manipulated so that the two exponents are equal. Alignment can be achieved by shifting either the smaller number to the right (increasing its exponent) or shifting

the layer number to the last. Because either operation may result in the loss of digits, it is the smaller that is shifted, any digits that are lost are therefore or relatively small significance. The alignment is achieved by repeatedly shifting the magnitude portion of the significand right 1 digit & incrementing the exponent until the two exponents are equal. If this process results in a 0 value for the significand then the other number is reported as the result. If two numbers have exponents that differ significantly, the lesser number is lost.

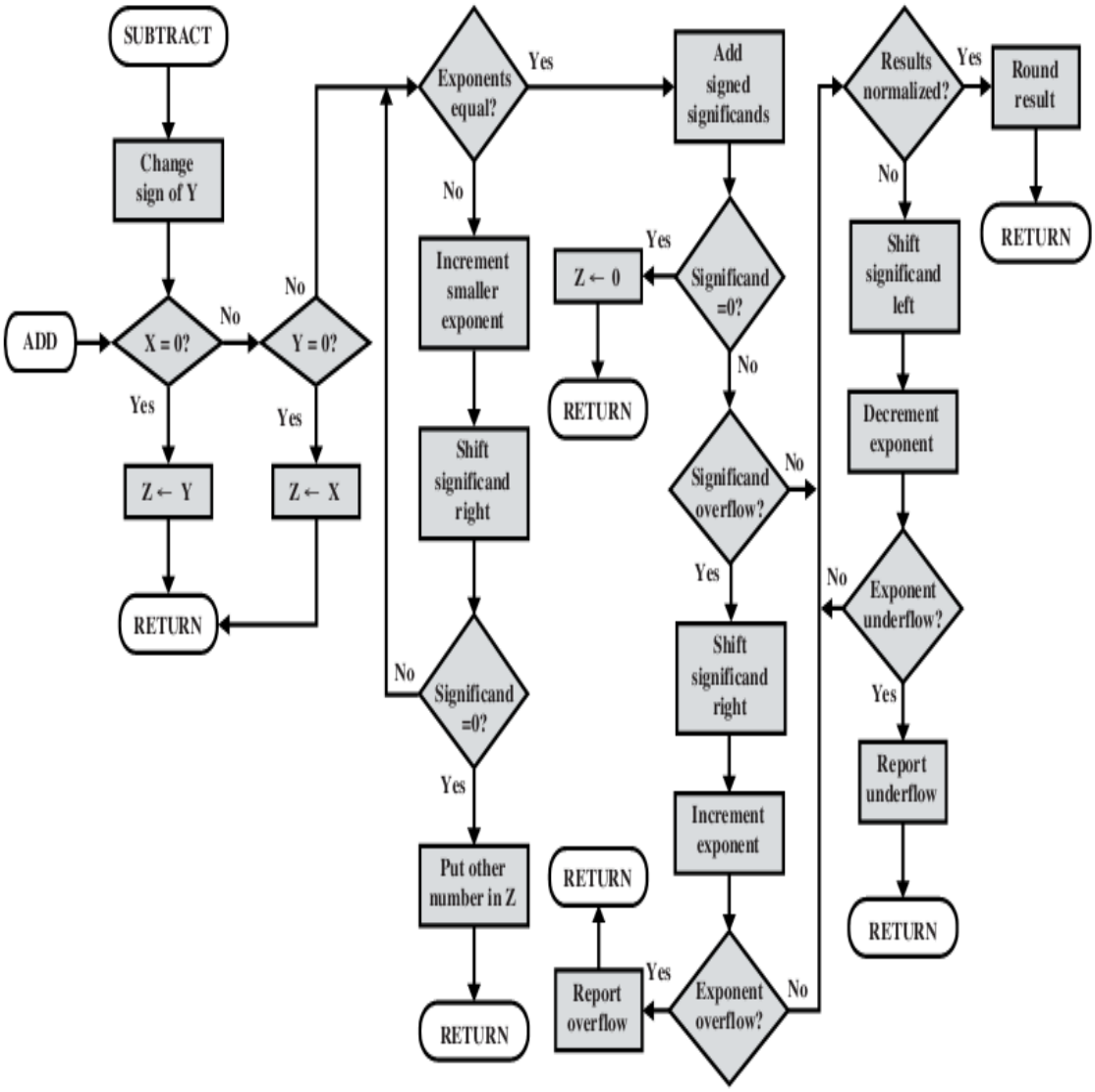


Figure :- Floating point addition subtraction(Z←X±Y)

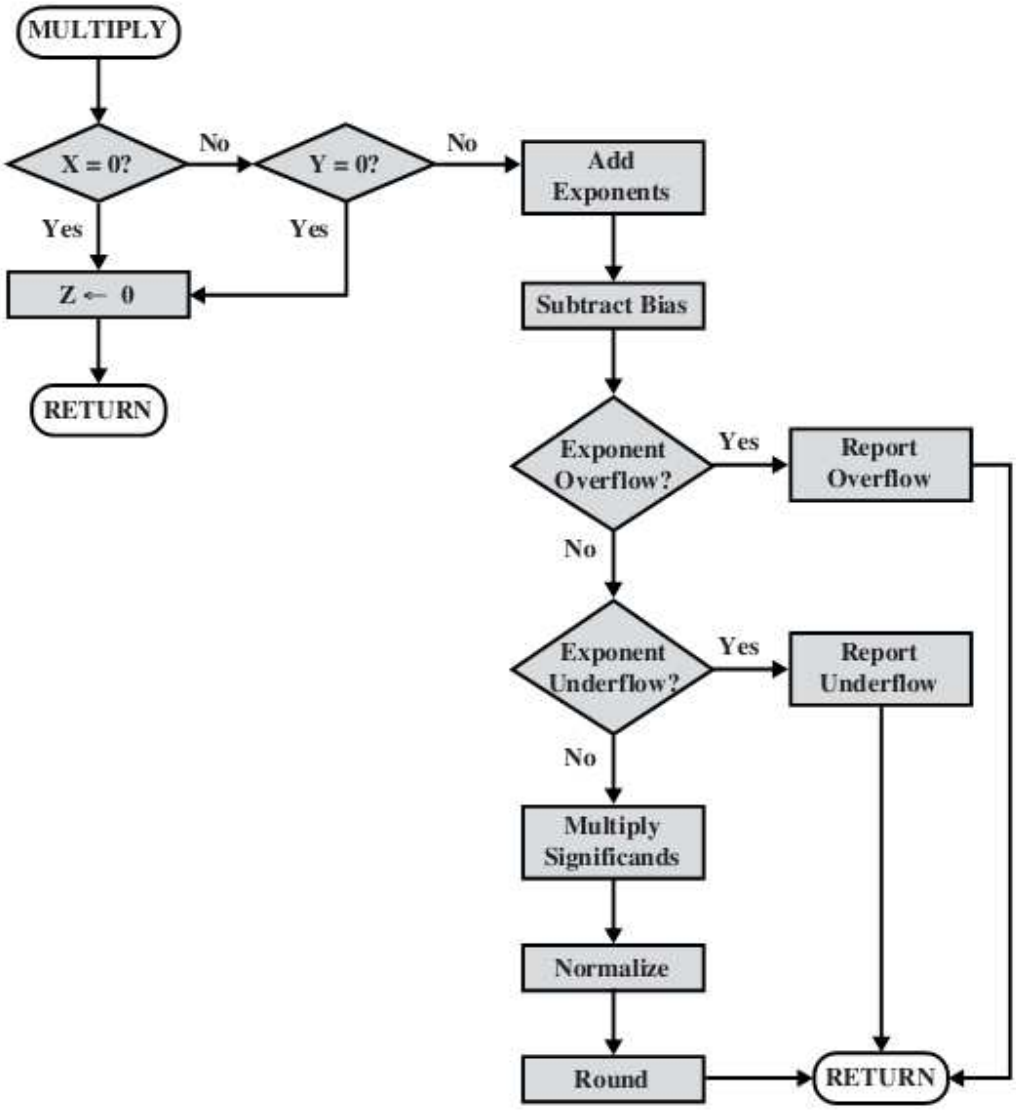
The third phase is addition in which the two significands are added together taking into account their signs. Because the sign may differ, the result may be a 0. There is also the possibility of significand of overflow by 1 digit. If so, the significand of the result is shifted right and the exponent is incremented. An exponent overflow could occur as a result; this would be reported & the operation halted.

The last and final phase, I.e. fourth phase is normalization. The result is normalized in this phase. Normalization consists shifting significand digits left until the most significant digit(bit, or 4 bit for base-16 exponent) is non-zero. Each shifts causes a decrement of the exponent & thus could cause an exponent underflow. Finally, the result must be rounded off & then reported.

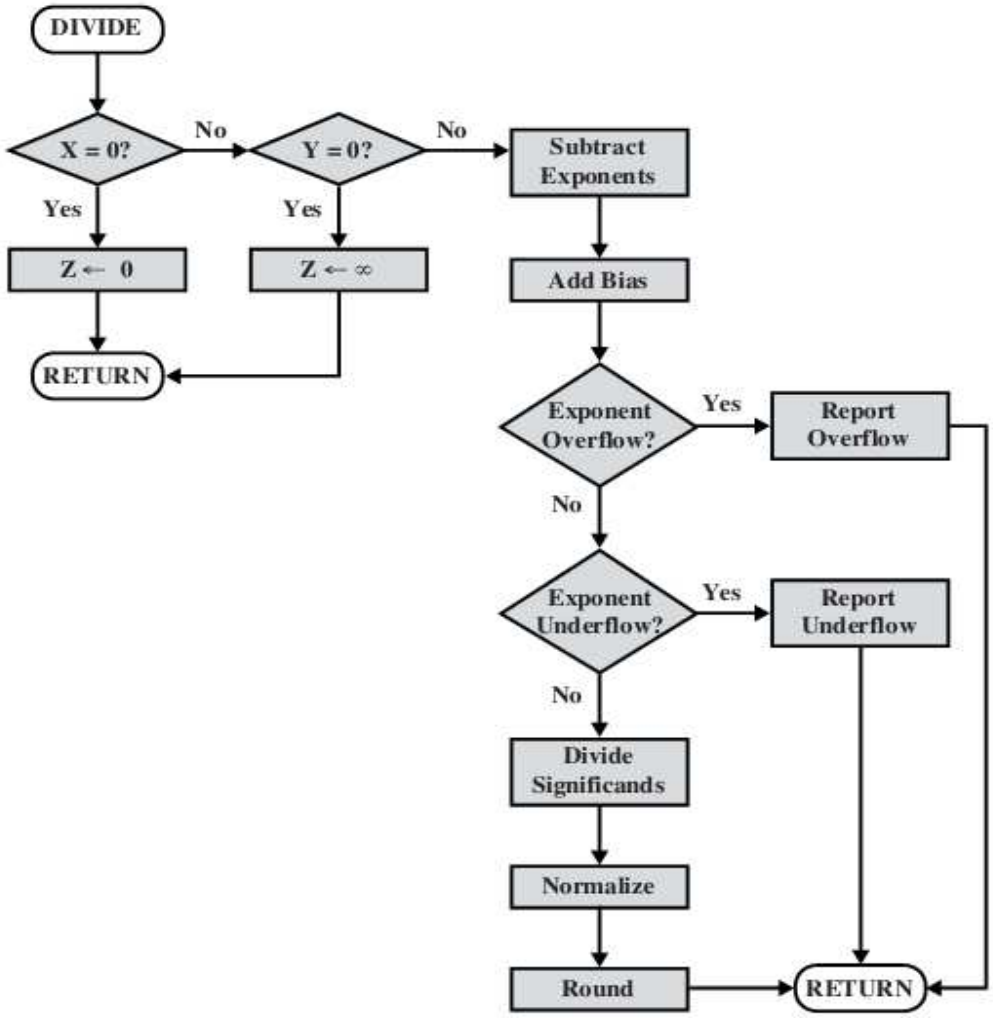
6.) Floating Point Multiplication/Division:-

- Check for zero
- Add/subtract exponents
- Multiply/divide significands (watch sign)
- Normalize
- Round
- All intermediate results should be in double length storage

Floating point multiplication (Z←X*Y)



Floating point Division:- (Z←X/Y)



7.)Instruction Sets:

What is an Instruction Set?

- The complete collection of instructions that are understood by a CPU
- Machine Code
- Binary
- Usually represented by assembly codes.

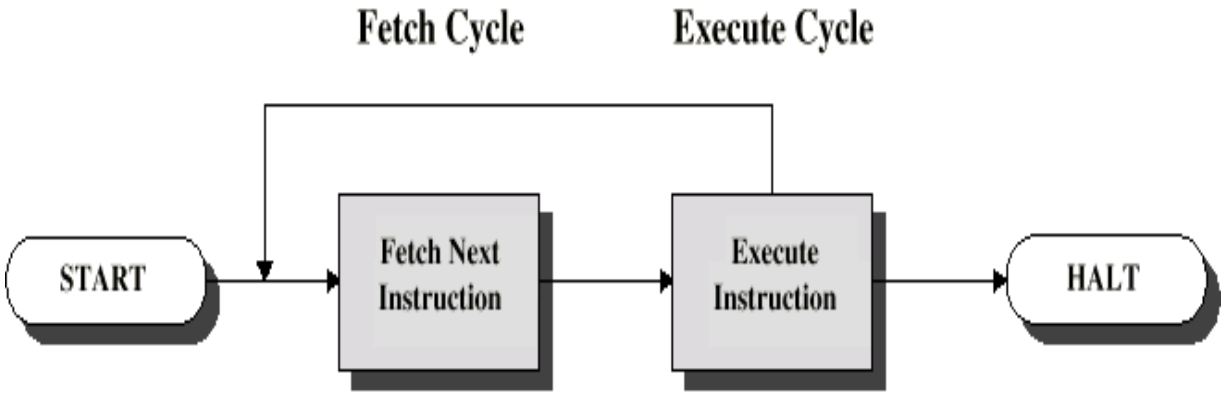
Elements of an Instruction

- Operation code (Op code)
  - — Do this
- Source Operand reference
  - — To this
- Result Operand reference
  - — Put the answer here
- Next Instruction Reference
  - — When you have done that, do this...

The operation of CPU is determined by instruction it executes referred to machine instruction or computer instruction.

The collection of different instruction that CPU can execute is referred to a CPU's intrusion set.

Basic instruction cycle:-



Fetch Cycle

- Program Counter (PC) holds address of next instruction to fetch
- Processor fetches instruction from memory location pointed to by PC
- Increment PC
    - — Unless told otherwise
- Instruction loaded into Instruction Register (IR)
- Processor interprets instruction and performs required actions

Execute Cycle

- Processor-memory
    - — data transfer between CPU and main memory
- Processor I/O
    - — Data transfer between CPU and I/O module
- Data processing
    - — Some arithmetic or logical operation on data
- Control
    - — Alteration of sequence of operations
    - — e.g. jump
- Combination of above

Instruction cycle state diagram:-

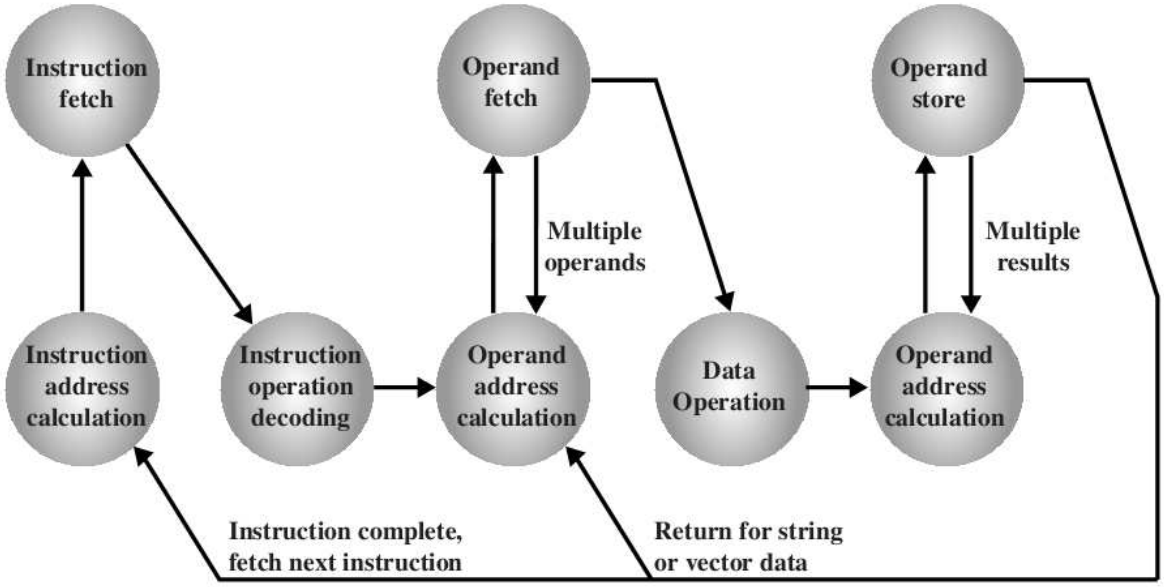
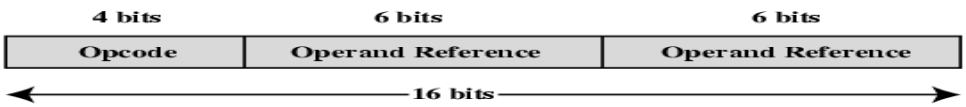


Figure :- Instruction cycle state diagram.

- ➔ Instruction cycle calculation determines the address of next instruction to be executed. Usually this involves adding a fix no. to address of previous instruction.
- ➔ Instruction fetch: Read instruction from its memory location into the processor.
- ➔ Instruction operation decoding: Analyze instruction to determine type of operation to be performed & operand to be executed.
- ➔ Operand address calculation: If operation involves reference to an operand in memory i/o the determine address of operand.
- ➔ Operand store: write result in memory in or out to i/p
- ➔ Data operation: Perform operation indicated instruction
- ➔ Operand store: Write result into memory or out to i/p.

#Simple Instruction Format:-



Instruction Types

- Data processing
- Data storage (main memory)
- Data movement (I/O)
- Program flow control

Number of Addresses (a)

- 3 addresses
  - Operand 1, Operand 2, Result
  - a = b + c;
  - May be a forth - next instruction (usually implicit)
  - Not common
  - Needs very long words to hold everything

Number of Addresses (b)

- 2 addresses
  - One address doubles as operand and result
  - a = a + b
  - Reduces length of instruction
  - Requires some extra work
    - Temporary storage to hold some results

Number of Addresses (c)

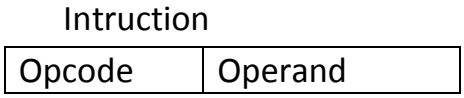
- 1 address
  - Implicit second address
  - Usually a register (accumulator)
  - Common on early machines

Number of Addresses (d)

- 0 (zero) addresses
  - All addresses implicit
  - Uses a stack
  - e.g. push a
  - push b
  - add
  - pop c
  - c = a + b

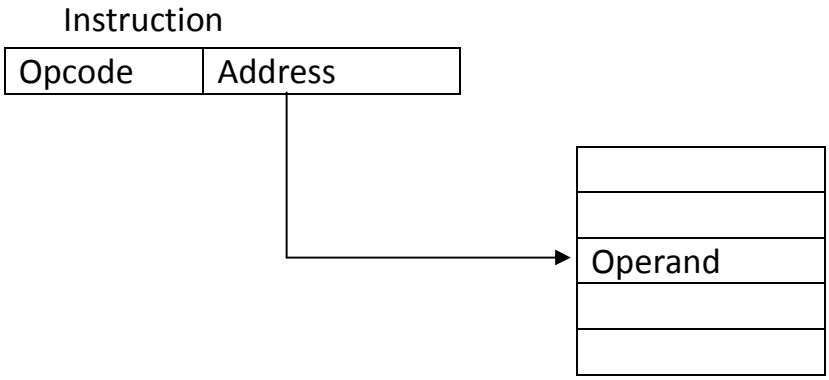
8.)Addressing Modes and Formats

1. Immediate Addressing



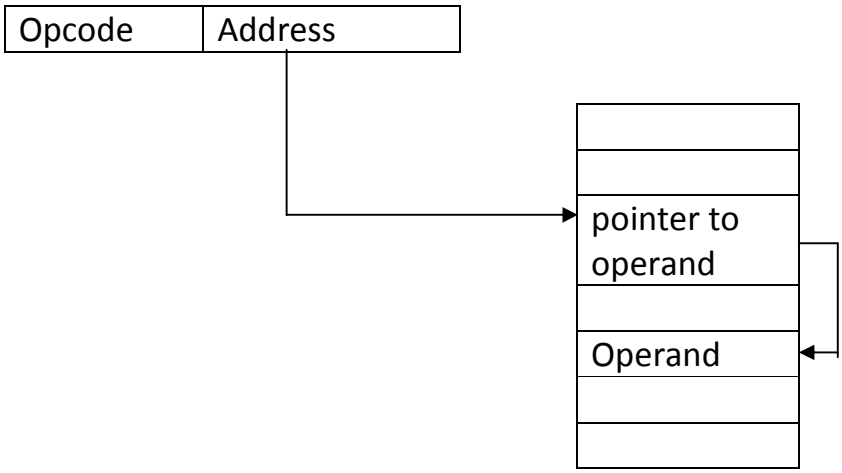
- Operand is part of instruction
- Operand = address field
- e.g. ADD 5
  - — Add 5 to contents of accumulator
  - — 5 is operand
- No memory reference to fetch data
- Fast
- Limited range

2. Direct Addressing



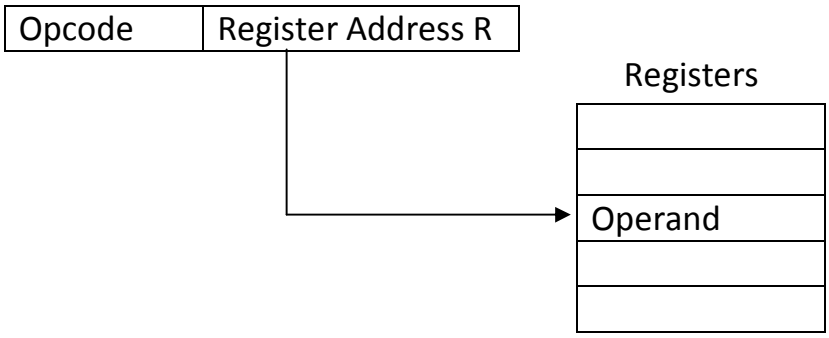
- Address field contains address of operand
- Effective address (EA) = address field (A)
- e.g. ADD A
  - — Add contents of cell A to accumulator
  - — Look in memory at address A for operand
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space

3. Indirect addressing:



- EA = (A)
  - ⎯ Look in A, find address (A) and look there for operand
- e.g. ADD (A)
  - ⎯ Add contents of cell pointed to by contents of A to accumulator
- Multiple memory accesses to find operand
- Hence slower

4. Register addressing mode

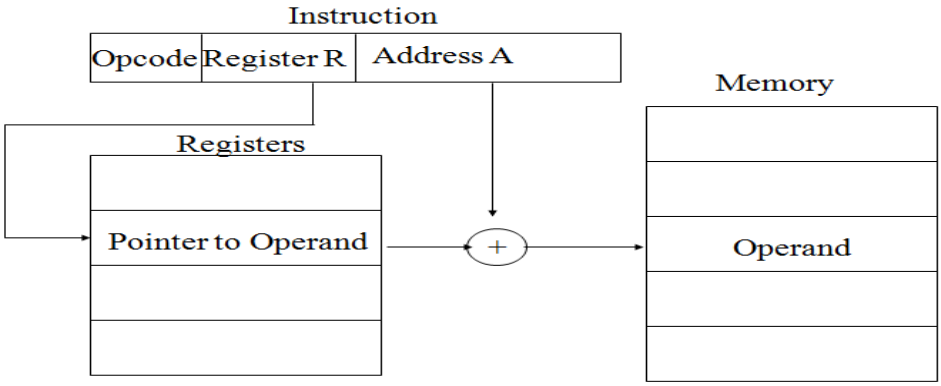


- Operand is held in register named in address filed
- EA = R
- Limited number of registers
- Very small address field needed
  - ⎯ Shorter instructions

— Faster instruction fetch

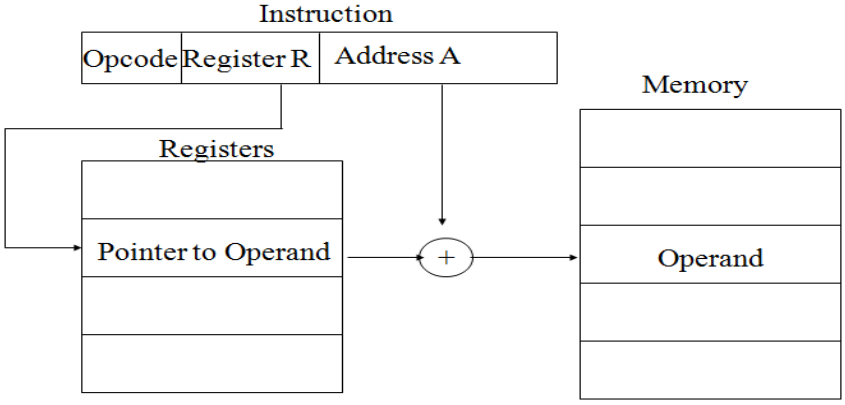
- No memory access
- Very fast execution
- Very limited address space

5. Register indirect addressing



- EA = (R)
- Operand is in memory cell pointed to by contents of register R
- Large address space (2^n)
- One fewer memory access than indirect addressing

6. Displacement addressing



- EA = A + (R)
- Address field hold two values
  - — A = base value
  - — R = register that holds displacement
  - — or vice versa

7. Relative addressing
   - A version of displacement addressing
   - R = Program counter, PC
   - EA = A + (PC)
8. Base- register addressing
   - A holds displacement
   - R holds pointer to base address
   - R may be explicit or implicit
9. Stack addressing
   - A = base
   - R = displacement
   - EA = A + R
   - Good for accessing arrays
     - EA = A + R
     - R++

6). Pentium and Power PC Evolution

Pentium incorporates the sophisticated design and serves as an excellent example of CISC design.

- 8080 → First general purpose microprocessor.
- → 8 bits
- 8086 → 16 bits
- →Wider data path and larger registers.
- →Also has instruction cache.
- 80286→Extension of 8086
- →can address 16 MB of memory
- 80386→ 32 bits
- →supports multitasking
- 80486→Uses Cache & Instruction pipelining
- →comes in with built math co-processor
-

Pentium:

- Super scalar technique to execute multiple instruction in parallel.

- Pentium Pro: features like register renaming branch prediction, data flow analysis, speculative execution.
- Pentium –II : incorporates MMX technology designed to process audio-video and graphics data.
- Pentium-III:- incorporates additional floating point instruction to support 3D graphics
- Merceed: it is 64 n-bit organization.
-

Power PC:

- It is direct dependant of RISC system. IBM 8018 is the best design & most powerful RISC based design system.
- 601:→it is 32 bita
- →it brought power PC evolution to market place
- 603 :→intended for low desktop & portable computer
- →32 bits
- 604:→32 bit machine
- →Intended for desktop & low servers.
- →used for superscalar design
- 620:→intended for high end server.
- →64 bit machine.
- 740/750(G3)-integrates two level of cache n main processor.
- G4:→Increased parallelism & interval speed.

Some questions:

1. What is processor organization? Describe the register organization of CPU??

ANS:

→ The organization of a processor is called processor organization.

→ALU:- Perform arithmetic and logical operation of data.

→Control Unit:- The CU controls the data into and out of CPU and Controls Operation of ALU.

→Registers:- These are the minimal internal memory consisting a set of storage locations used to store operand and result.
→Internal CPU bus: It helps transfer of data b/w various register and ALU

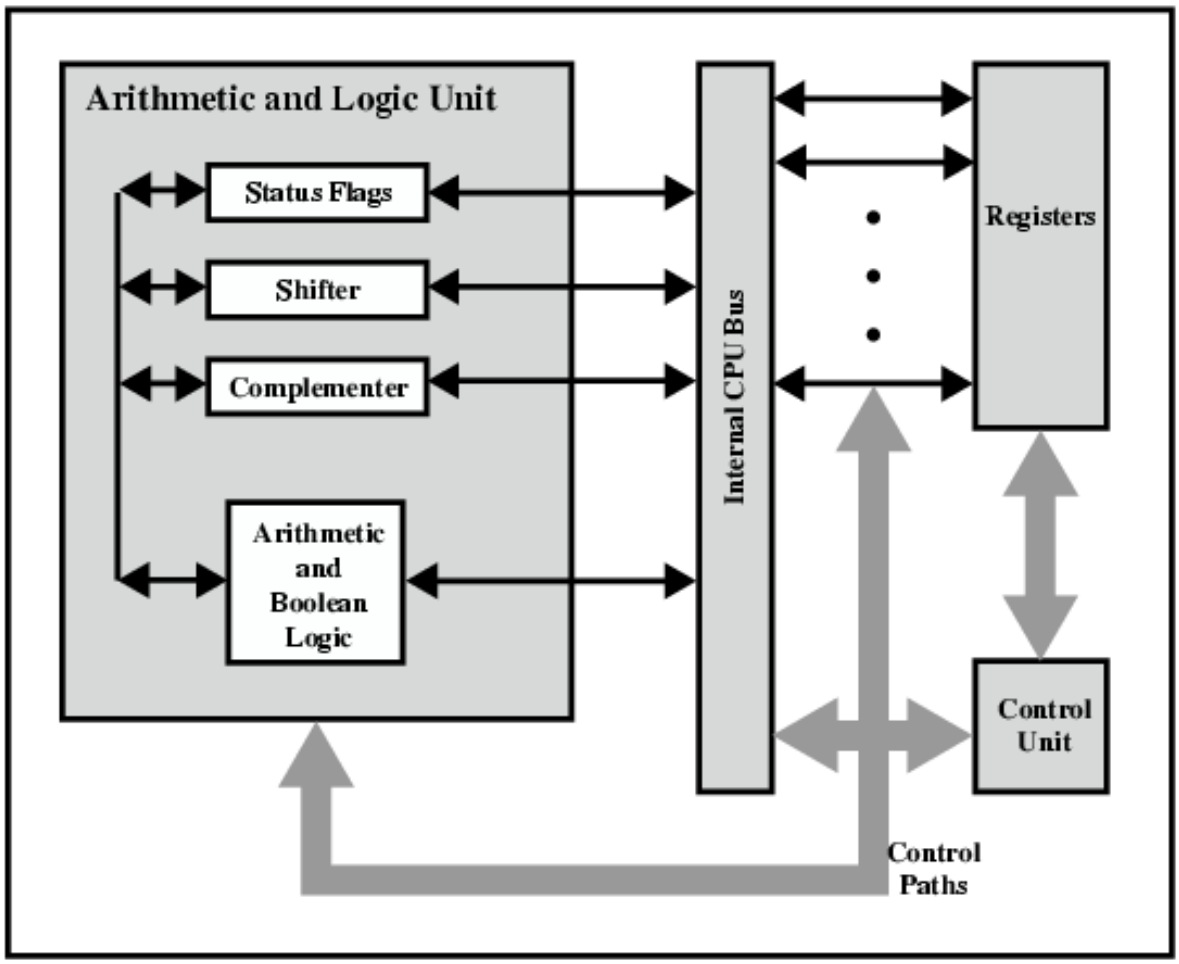


Fig:- Internal memory of CPU

To understand the organization of the processor, let us consider the requirement placed on the processor, the things that it must do.
→Fetch instruction: processor reads an instruction from memory (cache, main)
→Interpret instruction: the instruction is decoded to determine what action is enquired.
→Fetch data: the execution of an instruction may require reading data from memory or I/O module.

→Process data: the execution of an instruction require performing some arithmetic or logic operation on data.
→write data: the result of an execution may require writing data to memory or an I/O module.

Register Organization

The register in processor performs two roles:

1. User visible register:

A user visible register is the one that may be referenced by means of the machine language that the CPU executes ,it can be categorized as:

→General Purpose: Register that hold the operand and in some case use for addressing purpose.
→Data register: These registers may be used only to hold data and cannot be used in the calculation of operand address.
→Address register: They are devoted to a particular addressing mode. For eg: Segment pointer, stack pointer & index register.
→Condition codes (flags): These are the bits set by the CPU as a result of an operation. For eg: An arithmetic operation may produce a positive, negative, zero or overflow results.

2.Status & control register:

Control registers are used to control the operation of CPU. Most of these aren't visible to the user. For eg: Pc, instruction registers, memory address register, memory buffer register. All CPU contains a register called program status word(PSW)that contains condition codes plus other status information. For eg: sign, zero, carry, overflow interrupt etc

2. What is difference between zero address, one address and two address instruction?

ANS

→Zero address instruction:

They are applicable to a special memory organization called a stack. The stack is a last in first out set of location. Zero address instruction reference the top to stack elements.

For eg : A+B+C

Push A

Push B

ADD

Push C

ADD

→One-address instruction:

In one address instruction, the second order must be implicit the accumulation contains one of the operands and is used to store the result.

For eg: Load B

ADD A

→Two –address instruction:

With 2 –address instruction, one address must do double duty as both as operand and a result. To avoid altering the value of an operand, a 'MOV' instruction is used to move one of the value to a result or temporary location before performing the operation.

For eg: SUB A,B

⇨ A=A-B

3. Discuss the factors that influence the design of an instruction format.

ANS:

→One of the most interesting and most analyzed aspects of the computer design is instruction set design. The design of an instruction set is very complicated.

The most common factors are:

→Operation repertoire: how many and which operation to provide and how complex operation should be.

→Data types: The various types of data on which operation is to be performed.

→Instruction format: Instruction length (in bits), no of address, size of various field and others.

→Resistors: No. of CPU register that can be referenced by instructions and their use.

→Addressing: The mode or modes by which the address of an operand is specified.

4 .Explain role of stack in programming.

ANS:

→Computers with stack organization have PUSH & POP instruction which requires an address field.

For e.g.

PUSh X;

POP←M[X]

Pop of stack

→This instruction PUSH pushes the word or data at address X to the top of stack.

→The stack pointer (SP) is automatically updated.

→The operation instruction doesn't contain any address field because the operation is performed on to top most operand s of the stack.

5. Define Stack. Explain how arithmetic operation (A-B)/(C+A*D +B/C) is calculated by implementing.

ANS:

→

Three address	Comment
SUB Y, A, B;	Y←M[A]-M[B]
MPY T, D, E;	T←M[D]-M[E]
ADD T, T, C;	T←T+M[C]

DIV Y, Y, T;	Y←Y/T

Two address	Comment
MOVE Y, A	Y←A
SUB Y, B	Y←Y-B
MOV T, D	T←D
MPY T, E	T←T*E
ADD T, C	T←T+C
DIV Y, T	Y←Y/T

One address	Comment
LOAD D	AC←D
MPY E	AC←AC*E
ADD C	AC←AC+C
STOR Y	Y←AC
LOAD A	AC←A
SUB B	AC←AC-B
DIV Y	AC←AC/Y
STOR Y	Y←AC

Zero address	Comment
PUSH D	TOS←D
PUSH E	TOS←E
MPY	TOS←D*E
PUSH C	TOS←C
ADD	TOS←D*E+C
POP Y	Y←TOS
PUSH A	TOS ←A
PUSH B TOS	TOS←B
SUB	TOS ←A-B
DIV	TOS←(A-B)/Y
POP Y	Y←TOS

6. Multiply (-9)(-13) using Booth's Algorithm*

ANS:

A	Q(-9)	Q	M(-13)	
00000000	11111001	0	11110011	(initial)
00001101	11110111	0	11110011	A←A-M
00000110	11111011	1	11110011	shift(1)
00000011	01111101	1	11110011	shift(2)
00000001	10111110	1	11110011	shift(3)
11110100	10111110	1	11110011	A←A+M
11111010	01011111	0	11110011	shift A
00000111	01011111	0	11110011	A←A-M
00000011	10101111	1	11110011	shift(5)
00000001	11010111	1	11110011	shift(6)
00000000	11101011	1	11110011	shift(7)
00000000	01110101	1	11110011	shift(8)

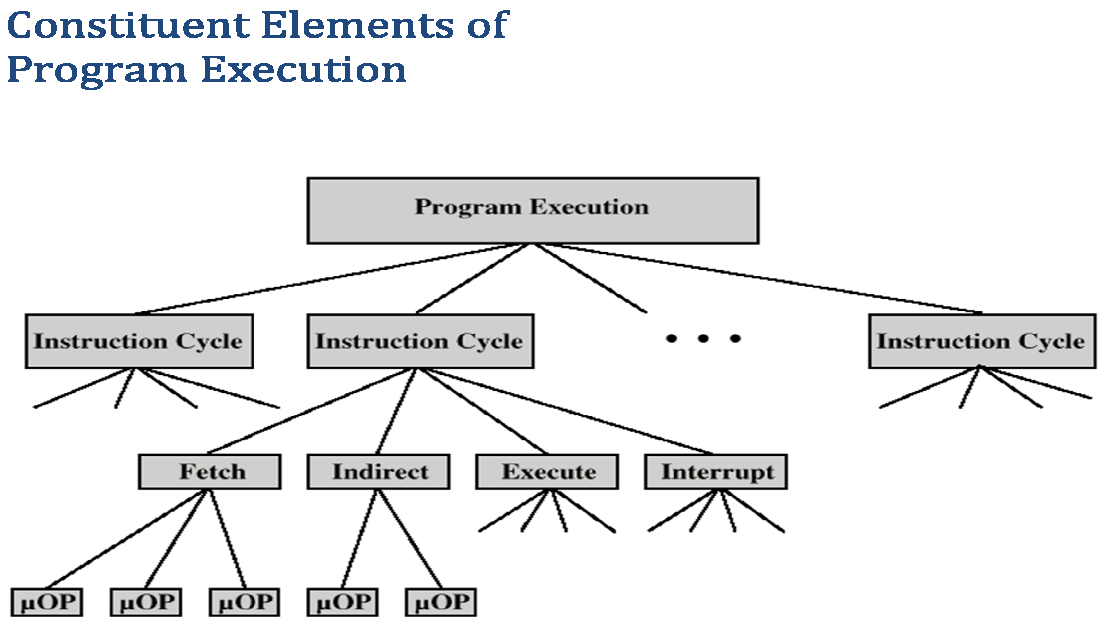
Product in A-Q →00000000 01110101
→01110101
=117

Chapter3 :- Control Unit Design

1.)Micro-Operations:-

The execution of instruction involves the execution of a sequence of sub-steps generally called cycles. For e.g. an execution may consist of fetch, indirect execute & interrupt cycle. Each cycle is in turn made up of a sequence of more fundamental operation called micro operation.

A single micro-operation generally involves a transfer between register, a transfer between register & an external bus or a single ALU operation.



Fetch cycle:-

The beginning of the each instruction cycle and causes an instruction to be fetched from memory which consist.

MAR (Memory Address Register)

MBR (Memory Buffer Register)

PC (Program Counter)

IR (Instruction Register)

Micro-operations in fetch cycle:

t1: MAR ← (PC)

t2: MBR ← memory

PC ←(PC) +1

t3: IR ← MBR

→The address of next instruction to be executed is in PC.

→First step is to move that address to MAR. The desired address (in the MAR) is placed on the address bus. The control unit issue READ command on the control bus & the result appear on the data bus & is copied into the MBR.

→We also need to increment PC by 1.

→The third step is to move the contents of MBR to IR.

Indirect cycle:-

When instruction is fetched the next is to fetch source code. The instruction specifies an indirect address, then an indirect cycle must precede the execute cycle as follows:

Micro operation of indirect cycle:

t1: MAR←(IR address)

t2: MBR← memory

t3: IR (address) ←(MBR (address))

Interrupt cycle:-

At the completion of execute cycle, the test is made to determine whether any enabled interrupts have occurs. The nature of the cycle gently varies from one machine cycle to another.

Micro operation of interrupt cycle:

t1: MBR←(PC)

t2:MAR←Save address

PC: Routine Address

t3:Memory←MBR

Execute cycle:

The fetch, indirect and interrupt cycles are simple and predictable. Each involves a small, fixed sequence of micro- operations and in each case, the same micro operations are repeated each time around for ADD R1, X.

t1: MAR ←(IR address)

t2: MBR←memory

t3:R1←(R1) + (MBR)

Instruction cycle:

- Each phase decomposed into sequence of elementary micro-operations
- E.g. fetch, indirect, and interrupt cycles
- Execute cycle
  - One sequence of micro-operations for each opcode
- Need to tie sequences together
- Assume new 2-bit register

- Instruction cycle code (ICC) designates which part of cycle processor is in
  - 00: Fetch
  - 01: Indirect
  - 10: Execute
  - 11: Interrupt.

Note*: In exam Question may ask like : Describe the different methods to generate address of next micro-instructions in microcode memory(Control memory).*

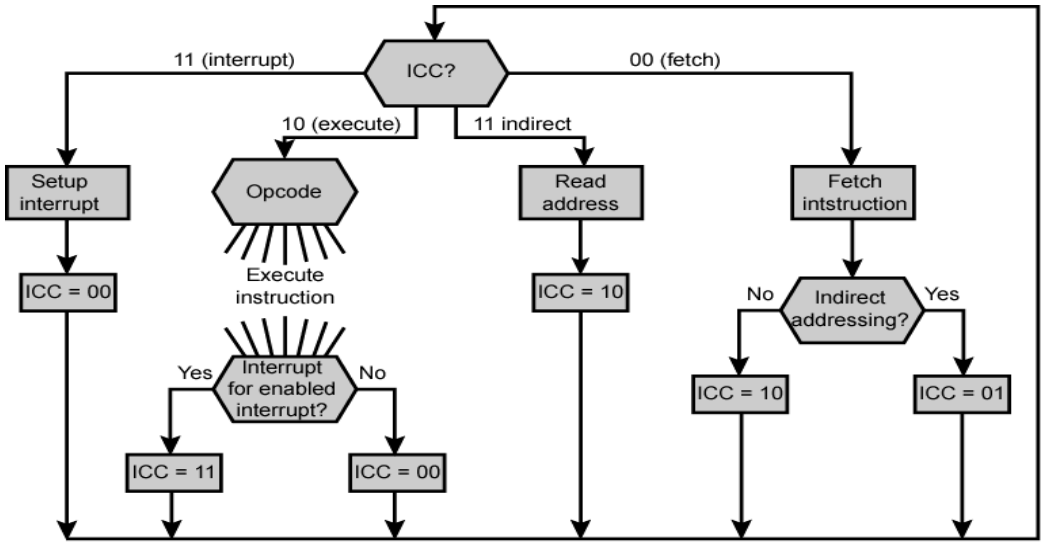
Ans : write above all cycles in your answer sheet.

2.) Flow chart for Instruction cycle:

We assume 2 bit register called instruction cycle code (ICC). It designates the state of processor in terms of which portion of cycle it is in

00: Fetch 10: Execute

01: Indirect 11: Interrupt



3.) Control Unit:-

It performs two tasks

a.) Sequencing:

The control unit causes the processor to step through a series of micro-operation in the proper sequence, based on the program being executed.

b.) Execution:-

The control unit causes each Micro-operation to be performed.

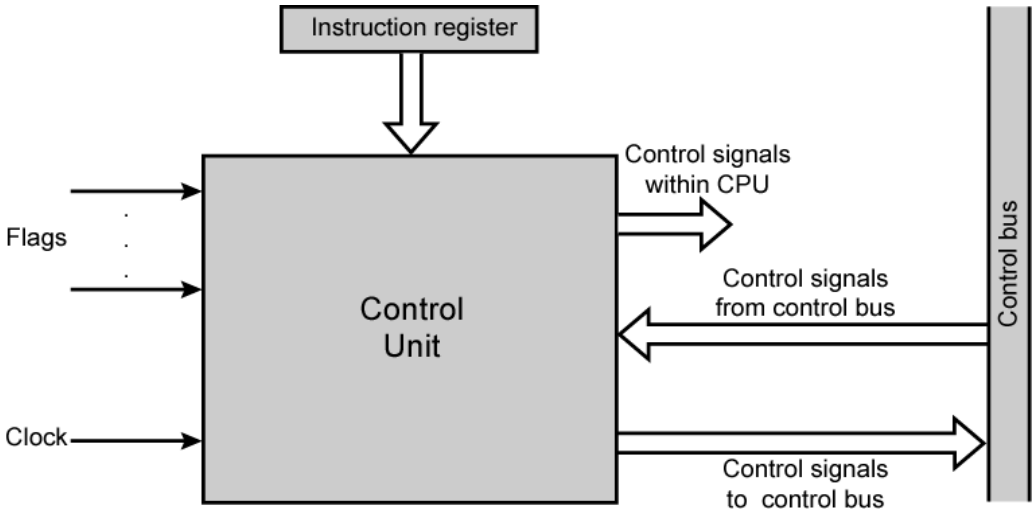


Fig. Block diagram of a control unit.

These two tasks (Sequencing & Execution) are done by Control Signals:

- Clock
  - — One micro-instruction (or set of parallel micro-instructions) per clock cycle
- Instruction register

— Op-code for current instruction

— Determines which micro-instructions are performed

- Flags

  — State of CPU

  — Results of previous operations

- Control signal from control bus

  — Interrupts

  — Acknowledgements

i.e. in control unit's **inputs** are Clock, Instruction register, Flags and Control Signal from control bus.

Now the **outputs** of Control units are:

- Within CPU

  — Cause data movement

  — Activate specific functions

- Via control bus

  — To memory

  — To I/O modules

Note : *Question may ask like: explain various i/p and o/p for CU.*

ANS: *Draw figure of CU and then write in Inputs section:-(Clock, Instruction Register, Flags, Control signal from control bus.*

Outputs sections:- (within CPU, Vai Control bus)

4.) Control signal example: (shown in next page)

Fetch	M-Op	Active Control Signals
	t1:MAR ←(PC)	C2
	t2:MBR←Memory	C5,CR
	PC: (PC +1)	
	t3: IR←MBR	C4

CR-Read control signal to system bus

CW-write

Similarly For interrupt &execute cycles… ☺

#Control Unit Implementation :

Two ways:

→Hardwired Implementation

→Micro Programmed Implementation

5.)Hardwired Implementation:-

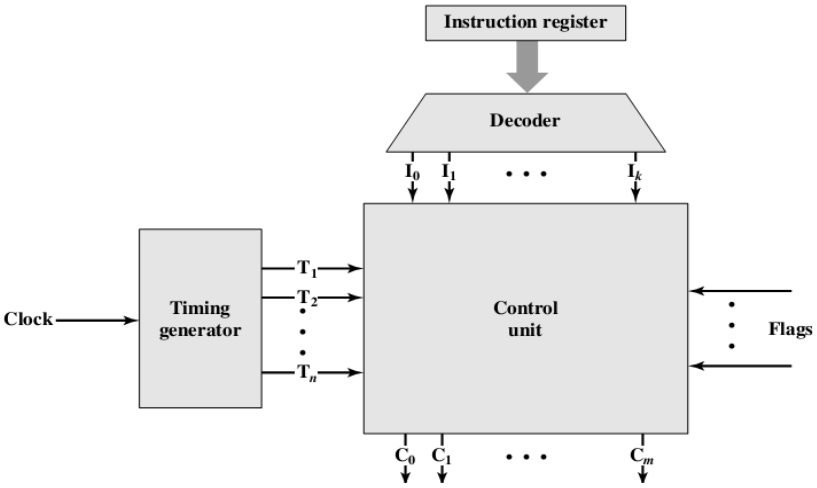


Figure: Control unit with decode i/p

In Hardwired implementation, the CU is essentially combinatorial ckt. Its i/p logic signals are transformed into a set of logic signals, which are the control signals. It is implemented through the use of finite no. of gates.

Its design uses a fixed architecture. It requires change in the wiring if the instruction set is modified or change. This architecture is performed as RISC (Reduced Instruction Set Computer).

Control Unit I/p:- It is given a decoded o/p from IR.

Control Unit logic:-

PQ= 00	Fetch
01	Indirect
10	Execute
11	Interrupt

Then the Boolean expression that defines C5: $\bar{P}\,\bar{Q}\,.\,T2 + \bar{P}.\,Q\,T2$ (That is control signals C5 will be asserted during the second time Unit and both fetch and indirect cycle)

6.)Micro programmed control (Firm ware):-

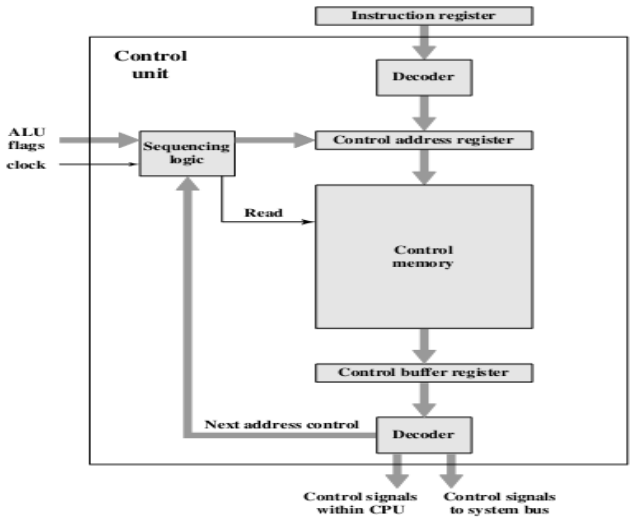
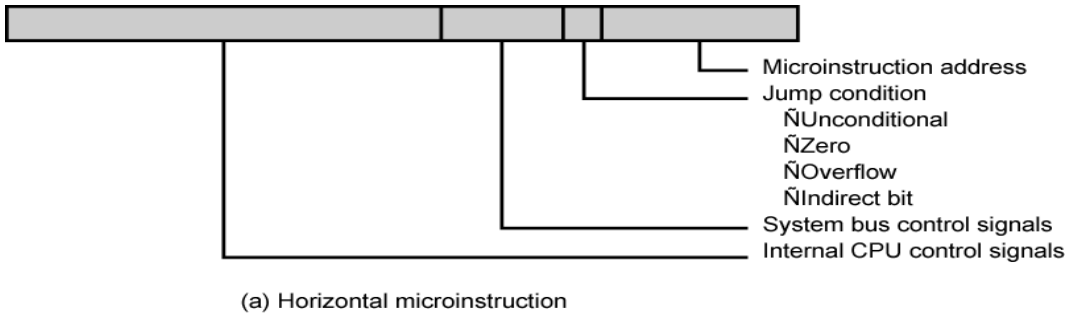
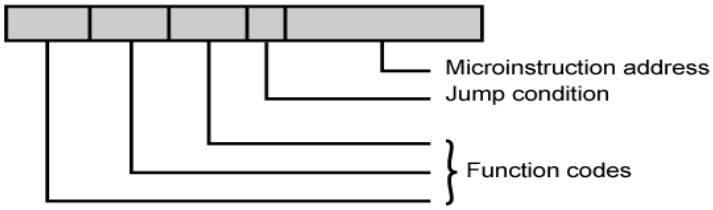


Figure: Functioning of micro programmed control unit

Micro instruction types:



(a) Horizontal microinstruction



(b) Vertical microinstruction

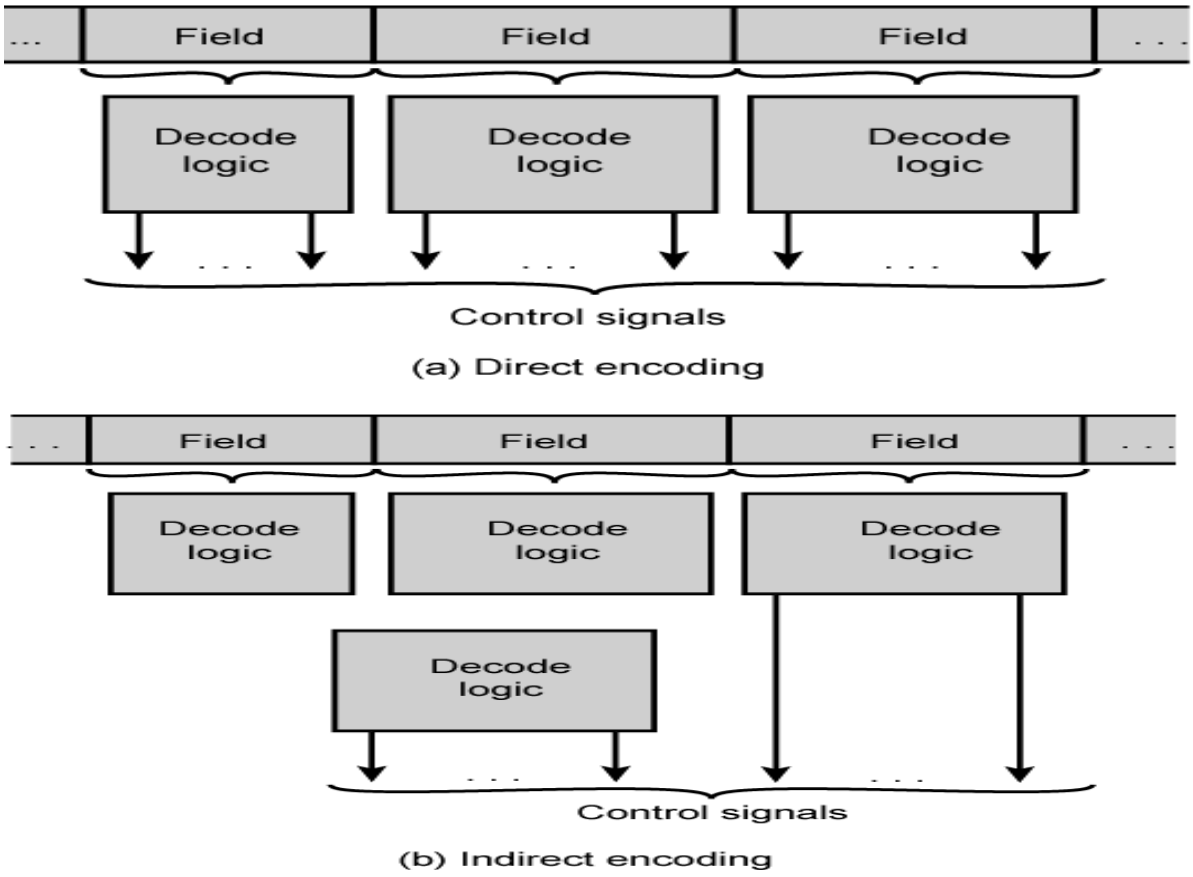
a.) Horizontal Micro-programming:

- Wide memory word
- High degree of parallel operations possible
- Little encoding of control information

b.) Vertical Micro-programming:

- Width is narrow
- n control signals encoded into $\log_2 n$ bits
- Limited ability to express parallelism
- Considerable encoding of control information requires external memory word decoder to identify the exact control line being manipulated

Micro Instruction encoding:-



In micro programmed control unit, the logic of CU is specified by micro program.

A micro program consists of a sequence of instruction in a micro programming language. These are the instruction that specify micro operation.

Micro instruction sequencing means getting the next micro-instruction from the control memory. Based on current micro instruction, condition flags, and the contents of the instruction register, a control memory address must be generated for the next micro instruction. A wide variety of techniques have been used. We can group them into three

general categories. These categories are based on the **format** of the address information in micro- instruction.

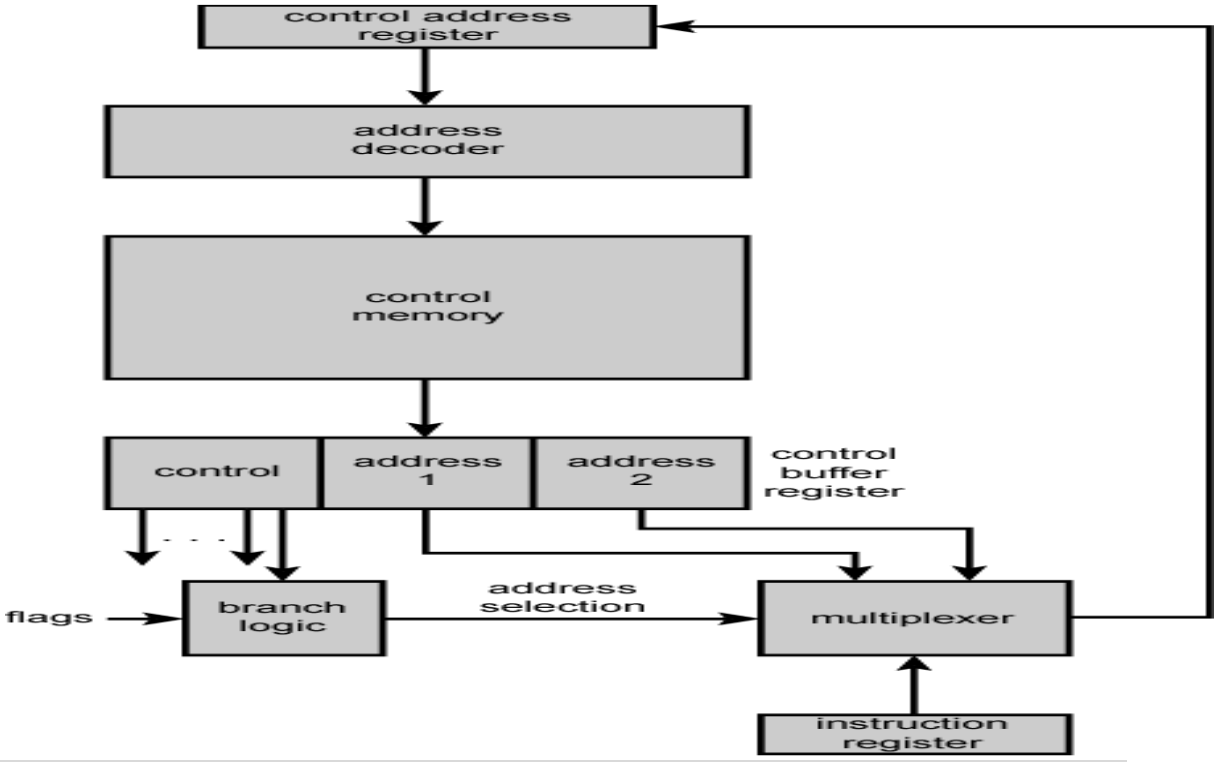
→ i.) Two address field.

→ ii.) Single address filed.

→iii.) Variable field.

i.) Two address field:-

The simplest approach is to provide two address field in each micro-instruction. A multiplexer is provided that search as the destination for both the address plus the instruction register. Based on an address selection i/p, the multiplexer transmit either the opcode or one of the two address CAR(Control address register). The CAR is subsequently decoded to provide the next micro-instruction address. The address selection signals are provided by a branch logic module whose o/p consist of flags plus bits from the control portion of micro-instruction. (figure below shows the branch control logic- two address format).



ii.) Single address format:-

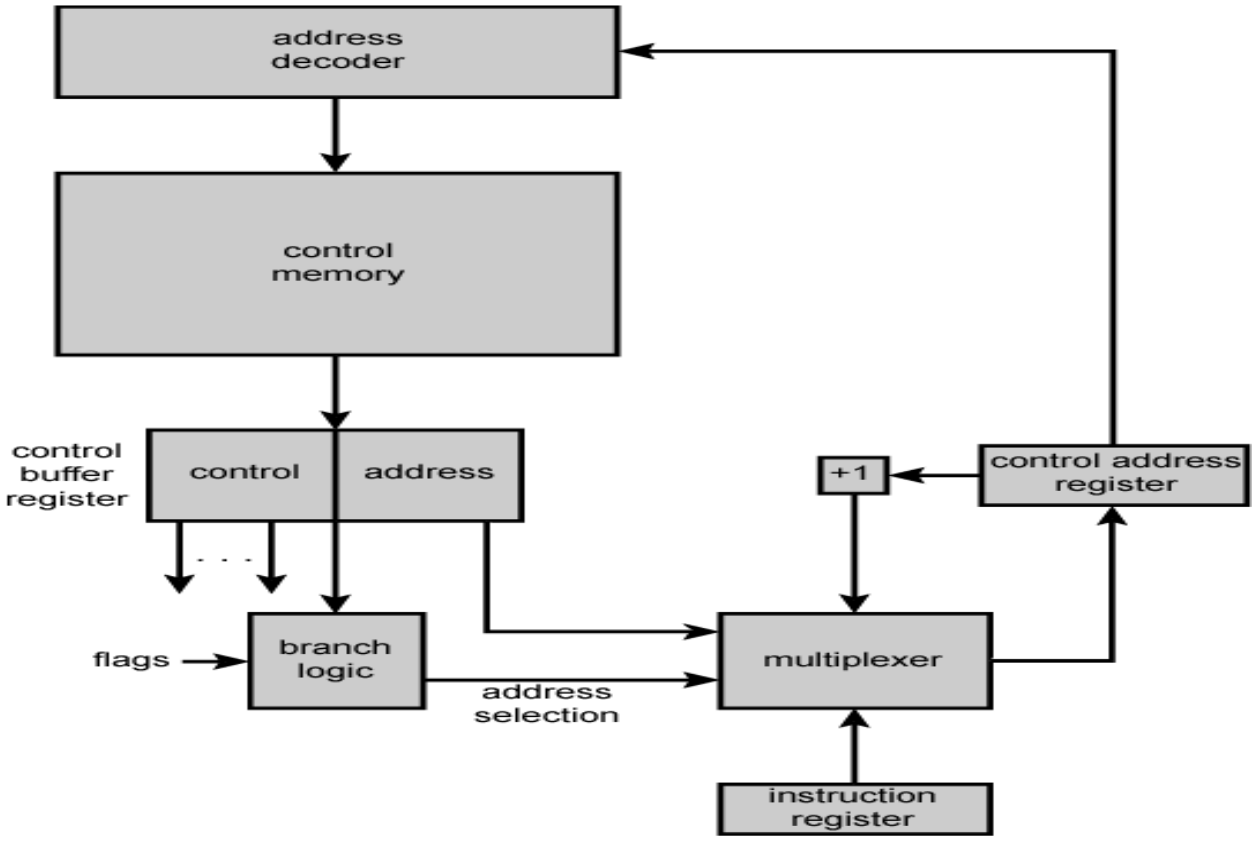


Figure : Branch control logic single address format

With single address field the option of next address are as follows : (i) address field, (ii)instruction filed, (iii)next sequential address.

The address selection signal determines which option is selected. This approach reduces the no. of address filed to one. However that the address filed often will not be used. Thus, there is some inefficiency in micro instruction coding scheme.

iii.) Variable address filed:-

It provides two entirely different micro instruction format one bit designed which format being used. In one format the remaining bits are used to activate control signal. In other format some bits drives the branches logic & remaining bits provides the address.

With the 1st format, the next address is either the next sequential address or an address derive from the instruction register. With the 2nd format, either conditional or unconditional branch being specified.

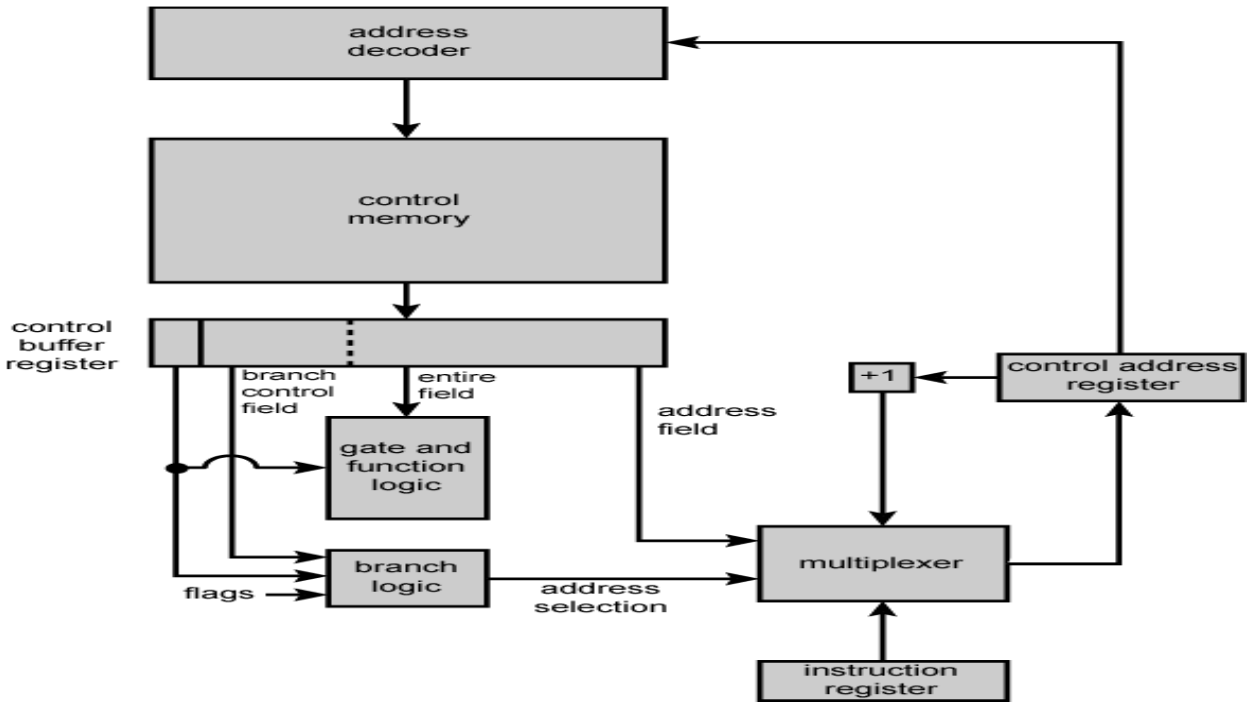


Figure: Branch control logic variable format.

Micro instruction execution: the micro instruction is the basic event on the micro programmed processor. Each cycle is made up of two parts : fetch & execute. The fetch portion is determined by the generation of micro instruction address.

The effect of execution of a micro-instruction is to generate control signals. Some of these signals control point interval to the processor. The remaining signals go to the external control bus or other external interface. As an incidental function, the address of next micro-instruction is determined.

The sequencing logic module contains the logic to perform functions. It generates address of next micro instruction, using as inputs: the instruction register, ALU flags, the control address register(for

incrementing), and the control buffer register. Control buffer register provide an actual address, control bits or both. The module is driven by clock that determines the timing of micro instruction cycle.

The control logic module generates control signals as a function of some of the bits in the micro instruction. The format and content of micro instruction determines complexity of control logic module.

Note: *Question may ask like: describe the block diagram of micro programmed or micro instruction sequencing and execution with suitable block diagram.*

ANS:- *write the micro instruction sequencing(*Two address field, Single address filed, Variable field). *And micro execution only..* ☺

Some questions:

1.) What is a micro- operation? Write the sequence of micro-operation for fetch & execute cycles of ADD X, R1 where R1 is a register and X is a location in memory.

ANS:

The execution of instruction involves the execution of a sequence of sub steps generally called cycles. For e.g. an execution may consist of fetch indirect execute & interrupt cycle. Each cycle is in turn made up of a sequence of more fundamental operations called micro operations.

Micro-operation of execute cycle for ADD X, R1:

t1:MAR←(IR address)

t2:MBR←memory

t3:R1← R1+MBR

micro-operation of fetch cycle for ADD X, R1:

t1:MAR←PC

t2:MBR←memory

t3:IR←MBR

2)What do you mean by micro programming language? Describe the applications of micro programming?

ANS:

In addition to the use of control of control signals, each micro operation is described in symbolic motation. The notation suspiciously looks like programming language. That language is known as micro programming language.

The application of micro programming are:

i.)Realization of computer.

ii.) imulation

iii.) Operating system support

iv.) Micro diagnostic

3.)what are advantage & disadvantages of hardwired, and micro-programmed control? Why is micro programmed control becoming increasingly more popular?

ANS:

=> Advantages of micro-programmed control:-

→ simplifies design of CU

→cheaper

→less error prone to implement

=>Disadvantages of micro- programmed control:-

→slower compared to hardwired control

=>Advantages of hardwired control:-

→faster than micro-programmed control

=>Disadvantages of hardwired control:-

→instruction set and control logic are directly hid together using complicated ckts that are difficult to design and modify.

Most of the computers today are micro-programmed. The reason is basically one of flexibility. Once the CU of hardwired computer is designed and built, it is virtually impossible to alter its architecture and instruction set. In micro-programmed computer, however, we can change the computer's instruction set by altering the micro-program stored in control memory. Lets take basic computer as an example, we notice that its four bit op-code permits Up-to 16 instructions. Therefore we could add seven more instruction to the instruction set by simply expanding its micro-program, To do this with hard wired version of computer would require a complete redesign of controller ckt hardware.

Another advantage to using micro-programmed control is the fact that the task of designing the computer in 1st place is simplified. The process of specifying the architecture and instruction set is now one of software (micro programming) as opposed to hardware design. Nevertheless for certain applications hard-wires computers are still used. Hence, micro-programmed control is becoming popular because of above reasons.

4.) What are the differences between horizontal and vertical micro instruction. Explain with their diagram.

ANS:

Horizontal micro instruction	**Vertical micro instruction**
1. wide memory card	1. width is narrow
2. little encoding of control information	2. considerable encoding of control information.
3. High degree of parallel operations possible Draw the figure	3. Limited ability to express parallelism Draw the figure.

Chapter 4:- Input / Output Organization

1)Input / Output:-

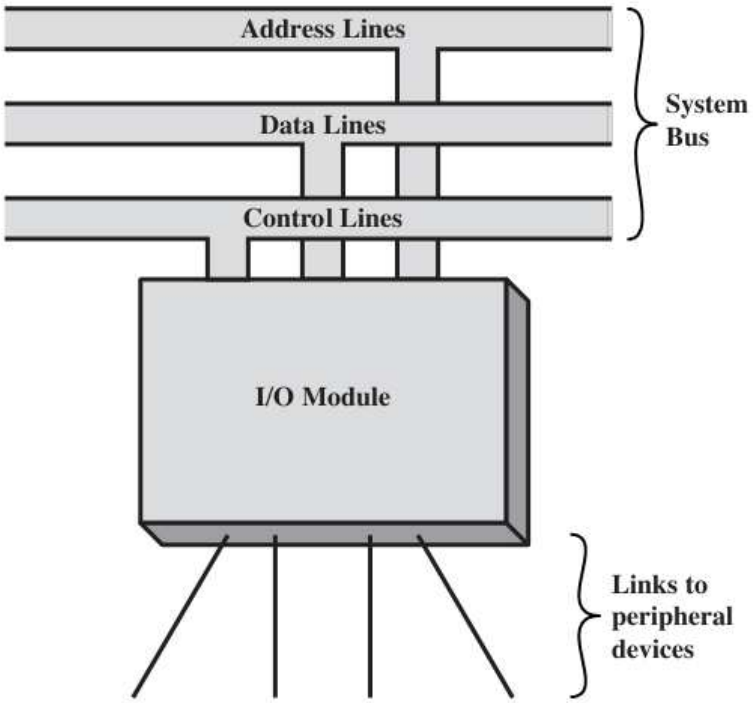


Fig: Generic model of I/O model

I/O module:-

→Interface to CPU & memory.

→Interface to one or more peripherals.

Why I/O module:- Instead of directly connecting I/O devices to the system bus, they are connected through I/O module because

a) Wide variety of peripherals.

b) Delivering different amounts of data.

c) In different speed and formats.

External devices:- We can broadly classify external devices into three categories:

- Human readable: Suitable for communicating with the computer user
- Machine readable: Suitable for communicating with equipment
- Communication: Suitable for communicating with remote devices

Examples of human-readable devices are video display terminals (VDTs) and printers. Examples of machine-readable devices are magnetic disk and tape systems, and sensors and actuators, such as are used in a robotics application.

Communication devices allow a computer to exchange data with a remote device, which may be a human-readable device, such as a terminal, a machine-readable device, or even another computer.

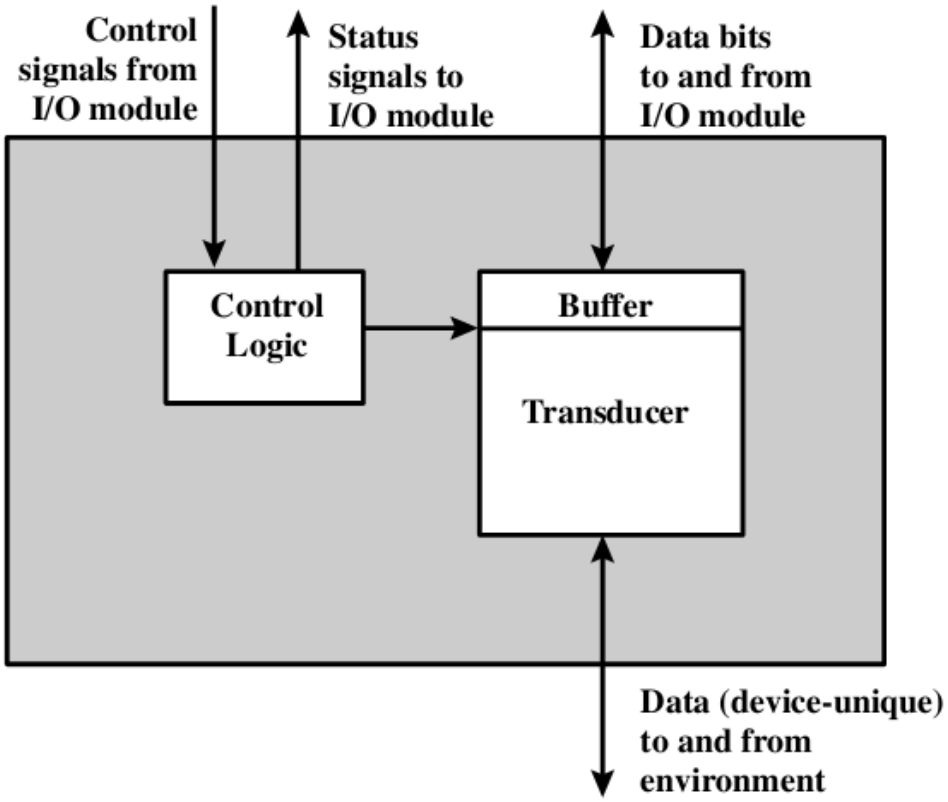


Figure 1: Block diagram of external devices.

In very general terms, the nature of an external device is indicated in Figure1. The interface to the I/O module is in the form of control, data, and status signals. Control signals determine the function that the device will perform, such as send data to the I/O module (INPUT or READ), accept data from the I/O module (OUTPUT or WRITE), report status, or perform some control function particular to the device (e.g., position a disk head). Data are in the form of a set of bits to be sent to or received from the I/O module. Status signals indicate the state of the device. Examples are READY/NOT-READY to show whether the device is ready for data transfer.

Control logic associated with the device controls the device's operation in response to direction from the I/O module. The transducer converts data from electrical to other forms of energy during output and from other forms to electrical during input. Typically, a buffer is associated with the transducer to temporarily hold data being transferred between the I/O module and the external environment; a buffer size of 8 to 16 bits is common.

2.) Functions of I/O module:-

The major functions or requirements for an I/O module fall into the following categories:

- Control and timing
- Processor communication
- Device communication
- Data buffering
- Error detection

The internal resources, such as main memory and the system bus, must be shared among a number of activities, including data I/O. Thus, the I/O function includes a **control and timing** requirement, to coordinate the flow of traffic between internal re-sources and external devices.

If the system employs a bus, then each of the interactions between the processor and the I/O module involves one or more bus arbitrations. The preceding simplified scenario also illustrates that the I/O module must communicate with the **processor** and with the external device.

On the other side, the I/O module must be able to perform **device communication**. This communication involves commands, status information, and data.

An essential task of an I/O module is **data buffering**. The transfer rate into and out of main memory or the processor is quite high, the rate is orders of magnitude lower for many peripheral devices and covers a wide range. Data coming from main memory are sent to an I/O module in a rapid burst. The data are buffered in the I/O module and then sent to the peripheral device at its data rate. In the opposite direction, data are buffered so as not to tie up the memory in a slow transfer operation. Thus, the I/O module must be able to operate at both device and memory speeds. Similarly, if the I/O device operates at a rate higher than the memory access rate, then the I/O module performs the needed buffering operation.

Finally, an I/O module is often responsible for **error detection** and for subsequently reporting errors to the processor. One class of errors includes mechanical and electrical malfunctions reported by the device (e.g., paper jam, bad disk track).Another class consists of unintentional changes to the bit pattern as it is transmitted from device to I/O module. Some form of error-detecting code is often used to detect transmission errors. A simple example is the use of a parity bit on each character of data. For example, the IRA character code occupies 7 bits of a byte. The eighth bit is set so that the total number of 1s in the byte is even (even parity) or odd (odd parity). When a byte is received, the I/O module checks the parity to determine whether an error has occurred.

3.) I/O Steps, block diagram of I/O and Three Techniques of I/O:

I/O steps

- CPU checks I/O module device status
- I/O module returns status
- If ready, CPU requests data transfer
- I/O module gets data from device
- I/O module transfers data to CPU
- Variations for output, DMA, etc.

Block Diagram of I/O

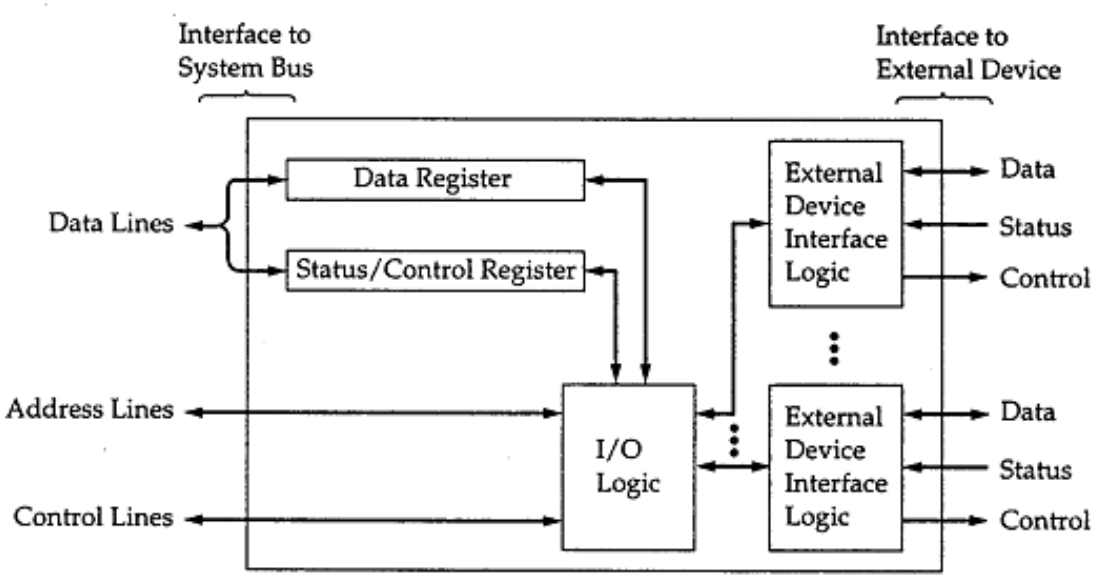


Figure shows the structure of an I/O module. It varies considerably in complexity and the no. of external devices they control.

A status register may also function a control register to accept detailed control information from processor. An I/O module function to allow the processor to view wide range of device in a simple minded way. I/O module may hide the detail of timing formats and the electro-mechanic of an external device so that the processor can function in terms of simple read/write commands. It all logical instruction that is needed for external

devices and processor relief from same extra work and speed up the overall performance of computer.

Three techniques of I/O operation:

i.) Programmed I/O:-

Programmed I/O (PIO) refers to data transfers initiated by a CPU under driver software control to access registers or memory on a device.

The CPU issues a command then waits for I/O operations to be complete. As the CPU is faster than the I/O module, the problem with programmed I/O is that the CPU has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data. The CPU, while waiting, must repeatedly check the status of the I/O module, and this process is known as Polling. As a result, the level of the performance of the entire system is severely degraded.

Programmed I/O basically works in these ways:

→CPU requests I/O operation

→I/O module performs operation

→I/O module sets status bits

→CPU checks status bits periodically

→I/O module does not inform CPU directly

→I/O module does not interrupt CPU

→CPU may wait or come back later

(see figure below (a))

ii.) Interrupt driven I/O:-

The CPU issues commands to the I/O module then proceeds with its normal work until interrupted by I/O device on completion of its work.

For input, the device interrupts the CPU when new data has arrived and is ready to be retrieved by the system processor. The actual actions to perform depend on whether the device uses I/O ports, memory mapping.

For output, the device delivers an interrupt either when it is ready to accept new data or to acknowledge a successful data transfer. Memory-mapped and DMA-capable devices usually generate interrupts to tell the system they are done with the buffer.

Although Interrupt relieves the CPU of having to wait for the devices, but it is still inefficient in data transfer of large amount because the CPU has to transfer the data word by word between I/O module and memory.

Below are the basic operations of Interrupt:

→CPU issues read command

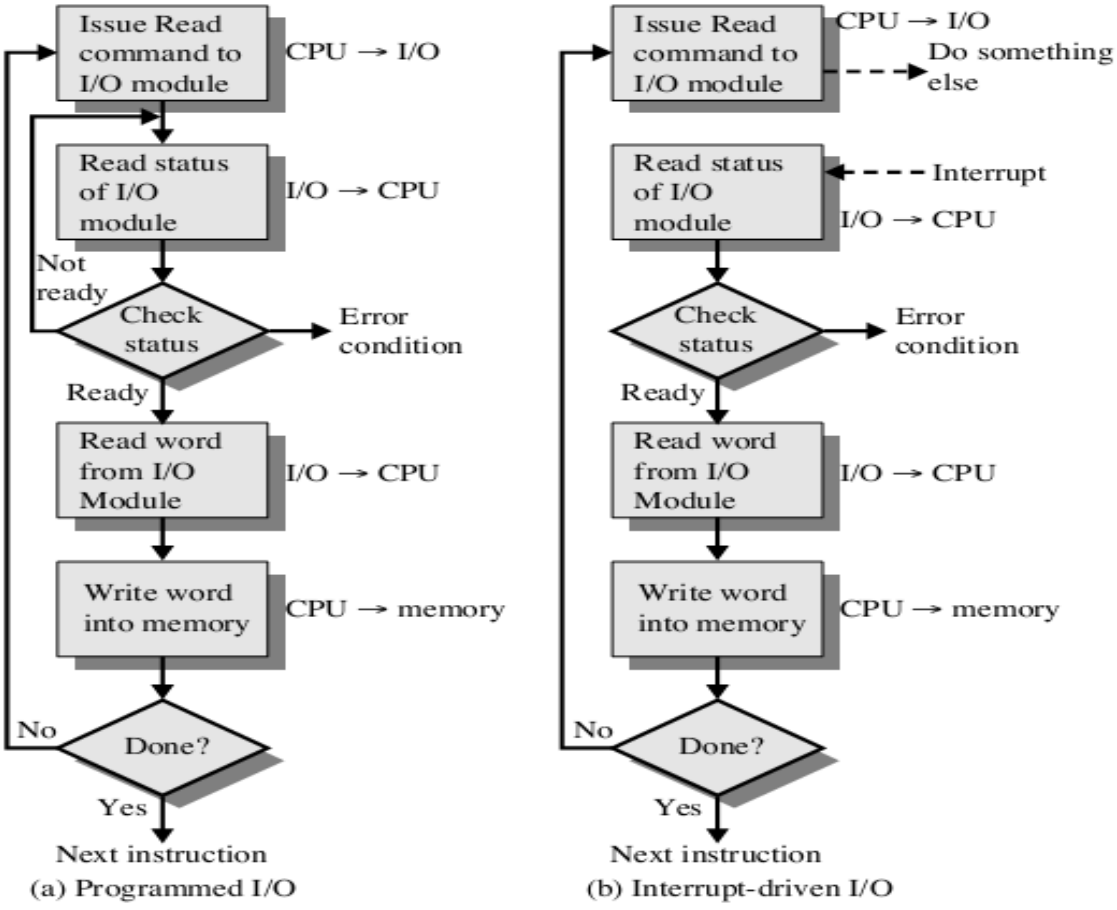
→I/O module gets data from peripheral whilst CPU does other work

→I/O module interrupts CPU

→CPU requests data

→I/O module transfers data

(See figure below (b))



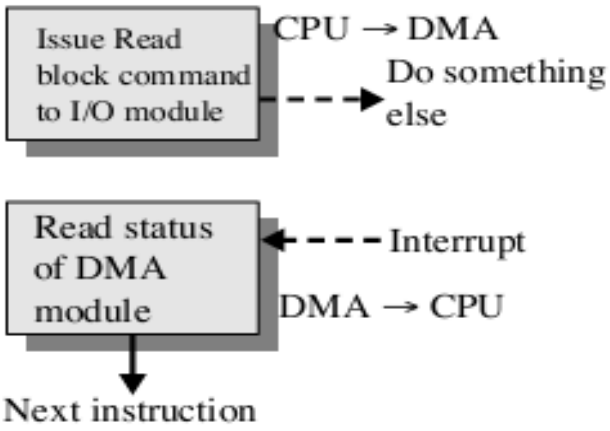
iii.) Direct Memory Access:-

Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module controls exchange of data between main memory and the I/O device. Because of DMA device can transfer data directly to and from memory, rather than using the CPU as an intermediary, and can thus relieve congestion on the bus. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of

transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.

DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory busses. Hardware design is complicated because the DMA controller must be integrated into the system, and the system must allow the DMA controller to be a bus master. Cycle stealing may also be necessary to allow the CPU and DMA controller to share use of the memory bus.



(c) Direct memory access

4.)Simple Interrupt Processing and Interrupt Design Issue:

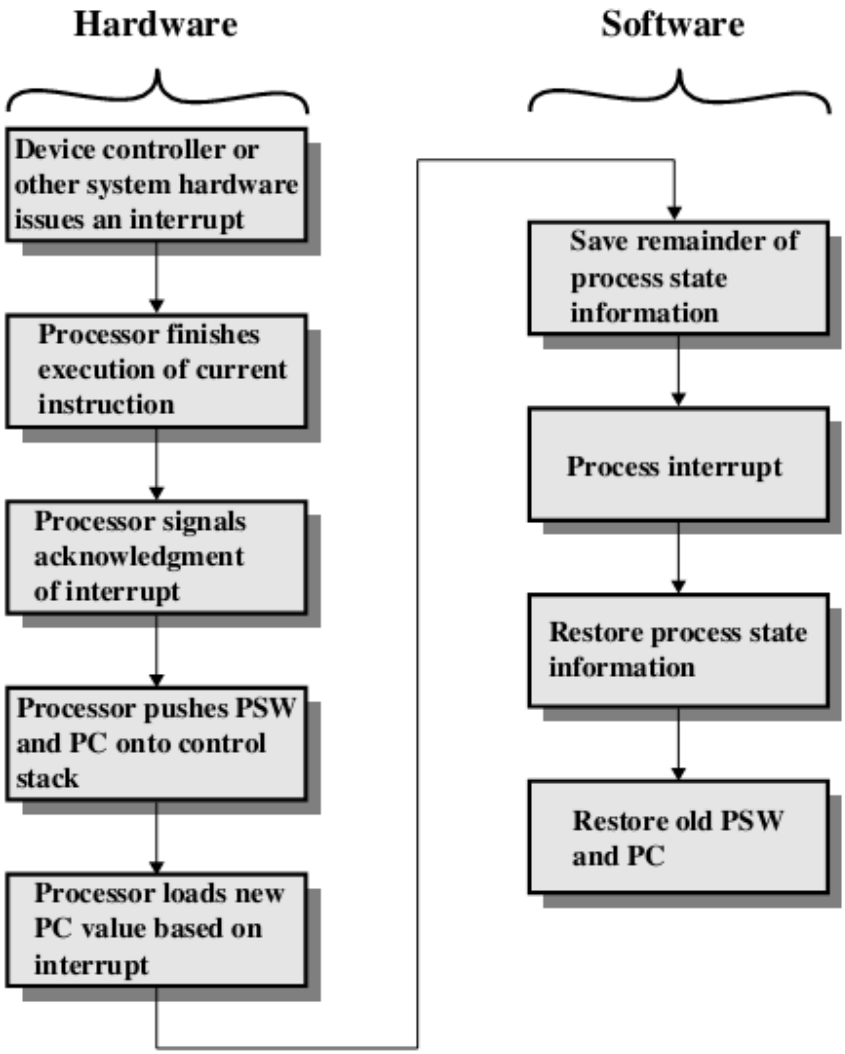
Interrupt Design Issue:

I.) How does processor determine which device issued the interrupt?

II.) If multiple interrupt occurs, how does processor decide which one to process??

Simple Interrupt Processing:

Shown in figure



5.) Direct Memory Access (DMA):-

In both programmed I/o & Interrupt driver I/O processor is involved. In DMA, the I/O module & main memory exchange data directly without processor involvement.

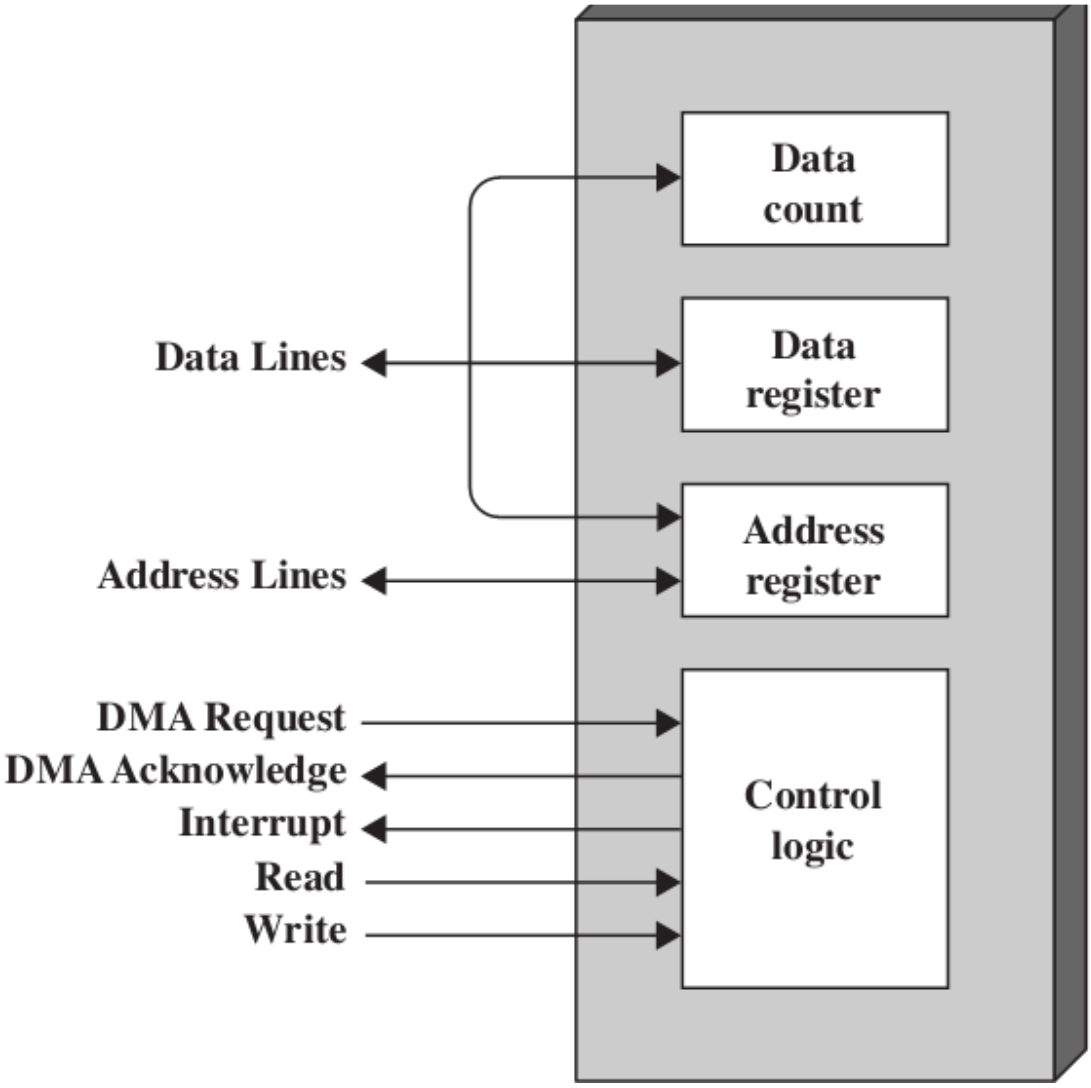


Figure: Typical DMA block diagram.

→When Processor wishes to read or write a block of data, It issues a command to the DMA module.

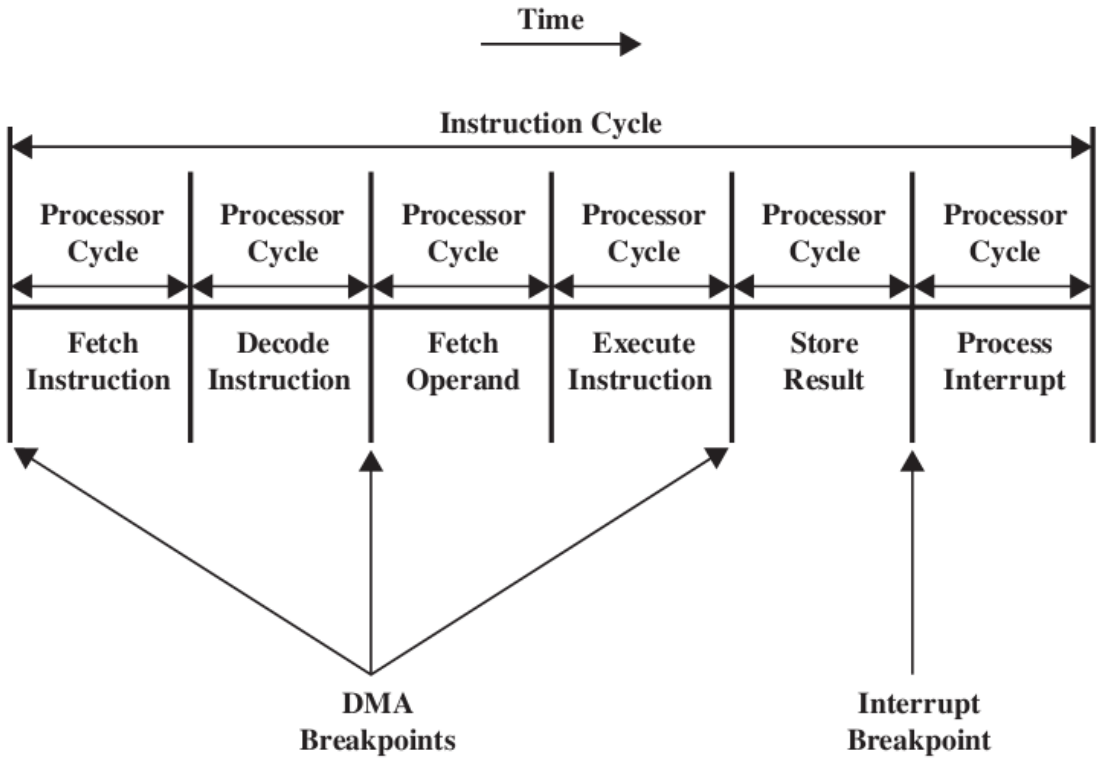
→Read or Write is requested through the read or write control.

→Data lines used for communication between I/O devices.

→The no. of words to be read or written are stored in data count registers.

→Address register store starting location in memory to be need or written.

→DMA involves an additional module on the system bus. DMA module takes control of the system from the processor. For this purpose the DMA module must force the processor to suspend operation temporarily. This technique is referred to as cycle stealing because the DMA module in effect steals a bus cycle.

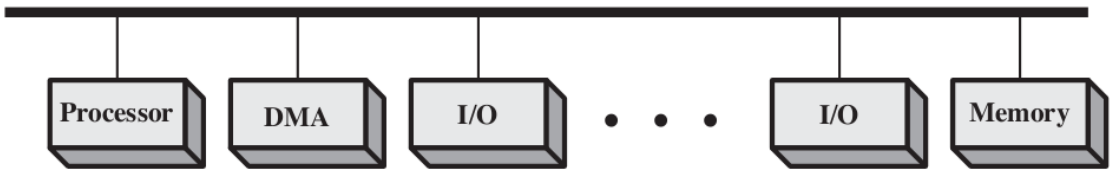


DMA and Interrupt Breakpoints During an Instruction Cycle

Types of DMA Transfer modes:

DMA can be configured in variety of ways:

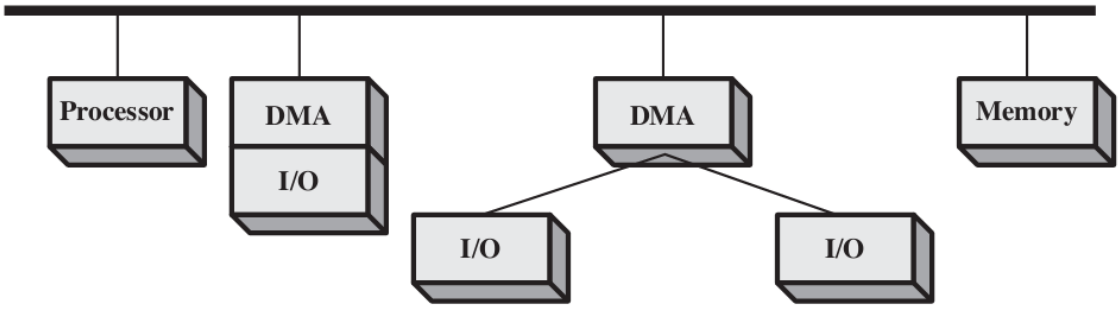
i) **Single Bus detached DMA:**



(a) Single-bus, detached DMA

As with processor controlled I/O each transfer requires two bus cycle.

ii.) **Single-Bus integrated DMA:**



(b) Single-bus, Integrated DMA-I/O

6.) I/O Channels:-

→I/O channels represents an extension of DMA concept.

→Channel has the ability to execute I/O instruction which gives it complete control over I/O operation.

→In a computer system with such instruction are stored in main memory to be executed by a special purpose processor in the I/O channel itself.

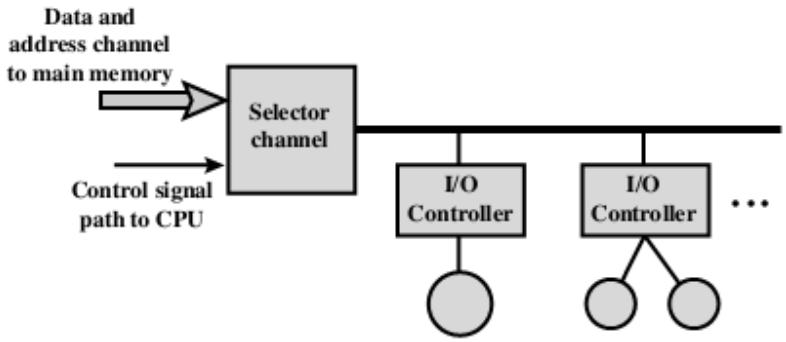
→Thus, CPU initiate an I/O transfer by instructing the I/O channel to execute a program in memory. The I/O channel follows these instruction & control the data transfer.

Two types of I/O channel. They are:

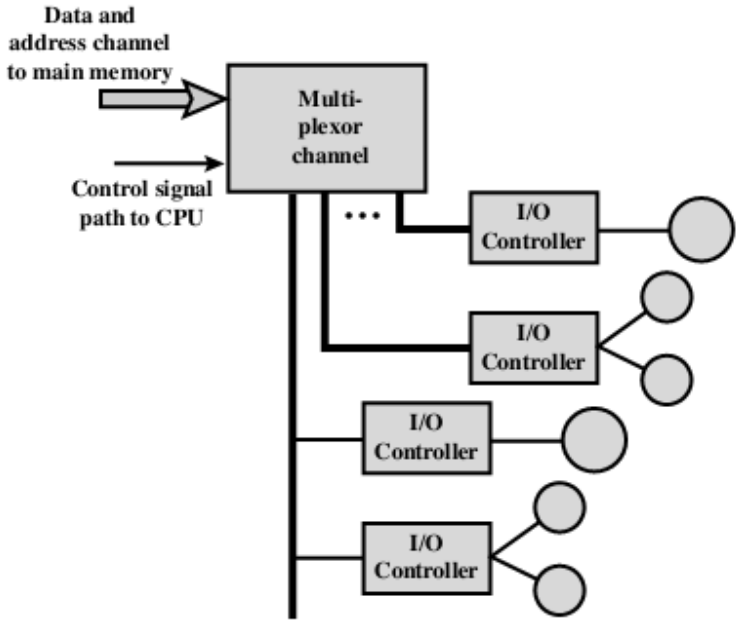
a) Selector

b) Multiplexer

Selector & Multiplexer



(a) Selector



(b) Multiplexor

→A selector channel control multiple high speed devices & at any one time is dedicated to transfer of data with one of those devices. Each device or a small set of devices is handled by a controller or I/O module.

→A multiplexer channel can handle I/O with multiple devices at the same time. For low speed devices, a byte multiplexer accepts or transmit character as fast as possible to multiple devices.

Some Questions:

1.) Explain the operation of DMA data transfer?

ANS:

DMA controllers vary as to the type of DMA transfers and the number of DMA channels they support. The two types of DMA transfers are flyby DMA transfers and fetch-and-deposit DMA transfers. The three common transfer modes are single, block, and demand transfer modes. These DMA transfer types and modes are described in the following paragraphs.

The fastest DMA transfer type is referred to as a single-cycle, single-address, or flyby transfer. In a flyby DMA transfer, a single bus operation is used to accomplish the transfer, with data read from the source and written to the destination simultaneously. In flyby operation, the device requesting service asserts a DMA request on the appropriate channel request line of the DMA controller. The DMA controller responds by gaining control of the system bus from the CPU and then issuing the pre-programmed memory address. Simultaneously, the DMA controller sends a DMA acknowledge signal to the requesting device. This signal alerts the requesting device to drive the data onto the system data bus or to latch the data from the system bus, depending on the direction of the transfer. In other words, a flyby DMA transfer looks like a memory read or write cycle with the DMA controller supplying the address and the I/O device reading or writing the data. Because flyby DMA transfers involve a single memory cycle per data transfer, these transfers are very efficient; however, memory-to-memory transfers are not possible in this mode.

The second type of DMA transfer is referred to as a dual-cycle, dual-address, flow-through, or fetch-and-deposit DMA transfer. As these names imply, this type of transfer involves two memory or I/O cycles. The data being transferred is first read from the I/O device or memory into a temporary data register internal to the DMA controller. The data is then written to the memory or I/O device in the next cycle. Figure 3 shows the

fetch and- deposit DMA transfer signal protocol. Although inefficient because the DMA controller performs two cycles and thus retains the system bus longer, this type of transfer is useful for interfacing devices with different data bus sizes. For example, a DMA controller can perform two 16-bit read operations from one location followed by a 32-bit write operation to another location. A DMA controller supporting this type of transfer has two address registers per channel (source address and destination address) and bus-size registers, in addition to the usual transfer count and control registers. Unlike the flyby operation, this type of DMA transfer is suitable for both memory-to-memory and I/O transfers.

In addition to DMA transfer types, DMA controllers have one or more DMA transfer modes. Single, block, and demand are the most common transfer modes. Single transfer mode transfers one data value for each DMA request assertion. This mode is the slowest method of transfer because it requires the DMA controller to arbitrate for the system bus with each transfer. This arbitration is not a major problem on a lightly loaded bus, but it can lead to latency problems when multiple devices are using the bus. Block and demand transfer modes increase system throughput by allowing the DMA controller to perform multiple DMA transfers when the DMA controller has gained the bus. For block mode transfers, the DMA controller performs the entire DMA sequence as specified by the transfer count register at the fastest possible rate in response to a single DMA request from the I/O device. For demand mode transfers, the DMA controller performs DMA transfers at the fastest possible rate as long as the I/O device asserts its DMA request. When the I/O device un-asserts this DMA request, transfers are held off.

2.) Why can peripherals not connected to the system bus?

ANS:

The reason for using I/O module to connect external device, not directly to the system bus are as follows:

→There are wide range of peripherals with various methods of operation. It would be impractical to available all logics in processors.

→The data transfer rate of peripherals is often much slower than that of processor. Thus it is impractical to use high speed system bus to communicate directly with a peripherals.

→On the other hand, the data transfer rate of some peripherals is faster than that of memory and again mismatch would occur.

→Peripherals often use different data format and word length than the computer to which they are attached.

3.) compare and contrast between programmed & interrupt driven I/O.

ANS:

In programmed I/O, the instruction of I/O modules execute by issuing a command to the appropriate I/O module, the I/O module will perform the requested action and then set the appropriate bit in the I/O status register. Then I/O module takes no further action to the microcontroller. So it responsibility of the processor periodically to check the status of I/O module until it find operation is complete.

But in interrupt driven I/O the processor has not wait long time for I/O module of concern to be ready for either reception or transmission of data. In this type of I/O processor issue an I/O command to a module and then go on to do some useful work. The I/O module interrupts the processor to request service when it is ready to exchange data with processor.

4.) Explain I/O module and its usage?

ANS:

Input-output interface provides a method for transferring information be-tween internal storage and external I/O devices. Peripherals connected to a computer need special communication links for interfacing them with the central processing unit. The purpose of the communication link is to

resolve the differences that exist between the central computer and each peripheral. The major differences are:

1. Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory, which are electronic devices. Therefore, a conversion of signal values may be required.

2. The data transfer rate of peripherals is usually slower than the transfer rate of the CPU, and consequently, a synchronization mechanism may be needed.

3. Data codes and formats in peripherals differ from the word format in the CPU and memory.

4. The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

Input/Output Module

- Interface to CPU and Memory
- Interface to one or more peripherals
- GENERIC MODEL OF I/O DIAGRAM 6.1

I/O Module Function

- Control & Timing
- CPU Communication
- Device Communication
- Data Buffering
- Error Detection

I/O Steps

- CPU checks I/O module device status
- I/O module returns status
- If ready, CPU requests data transfer
- I/O module gets data from device
- I/O module transfers data to CPU
- Variations for output, DMA, etc.

I/O Module Decisions

- Hide or reveal device properties to CPU
- Support multiple or single device
- Control device functions or leave for CPU
- Also O/S decisions

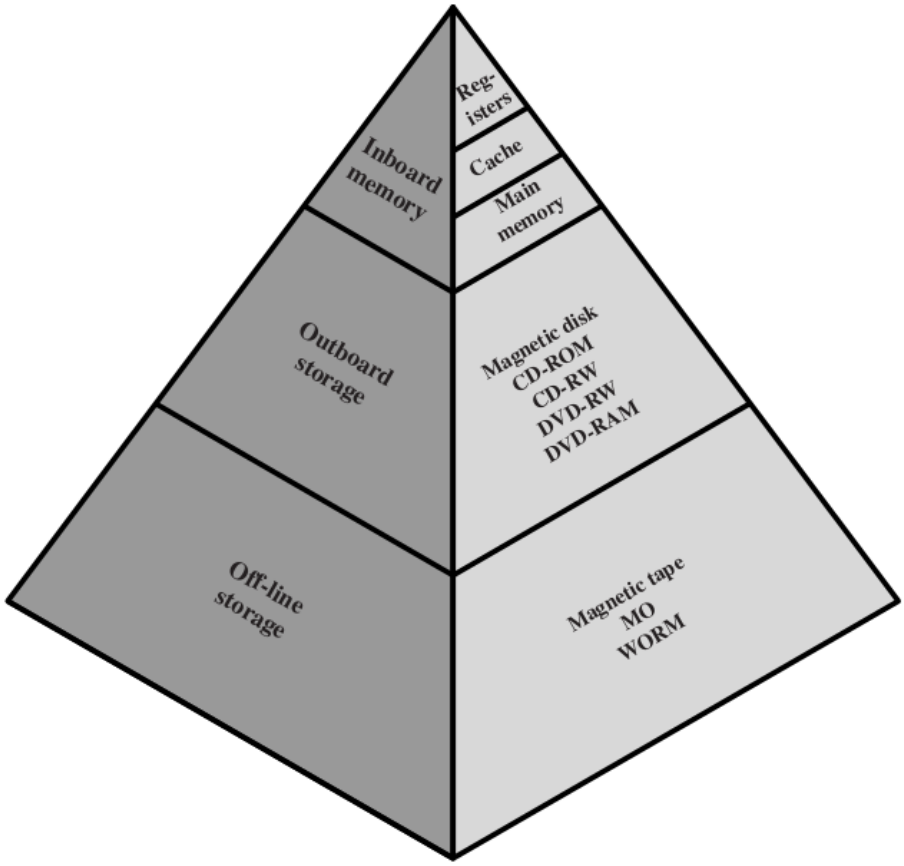
Q e.g. Unix treats everything it can as a file

Input Output Techniques

- Programmed
- Interrupt driven
- Direct Memory Access (DMA)

Chapter 5: Memory Organization

1.) Memory Hierarchy:-



Memory of computer is broadly divided into two categories.

→Internal

→External

Internal memory is used by CPU to perform task & External memory is used to store bulk information.

Main Memory Hierarchy:

i.) Register

ii.) Cache memory

iii.) Main memory

iv.) Magnetic Disk

v.) Removal Disk.

2.) Cache Memory:

- Small amount of fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or module

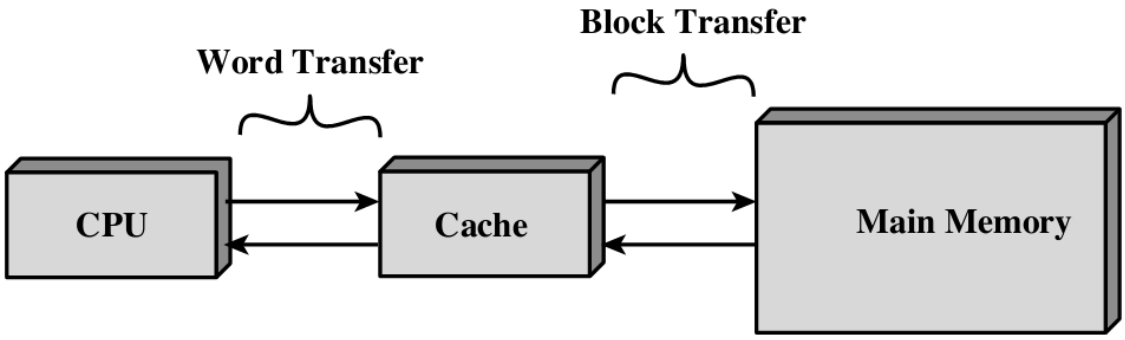


Figure: Single Cache

Cache memory is random access memory (RAM) that a computer microprocessor can access more quickly than it can access regular RAM. As the microprocessor processes data, it looks first in the cache memory and if it finds the data there (from a previous reading of data), it does not have to do the more time-consuming reading of data from larger memory.

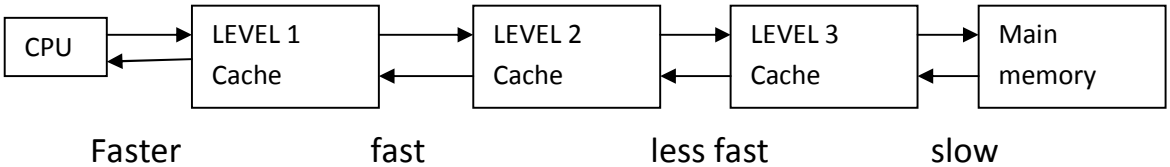


figure: Three level cache Organization.

Cache Read/ Write Policy:

Read policy: when a processor starts a read cycle the cache checks to see if that address is a cache hit.

Cache HIT: If the cache contains the memory location, then cache will respond to the read cycle & terminates the bus cycle.

Miss: If the cache does not contain the memory location, then main memory will respond to the processor & terminate the bus cycle. The cache will snarf (write itself) the data, so next time the processor request this data; it will be a cache hit.

Write policy: two common policies

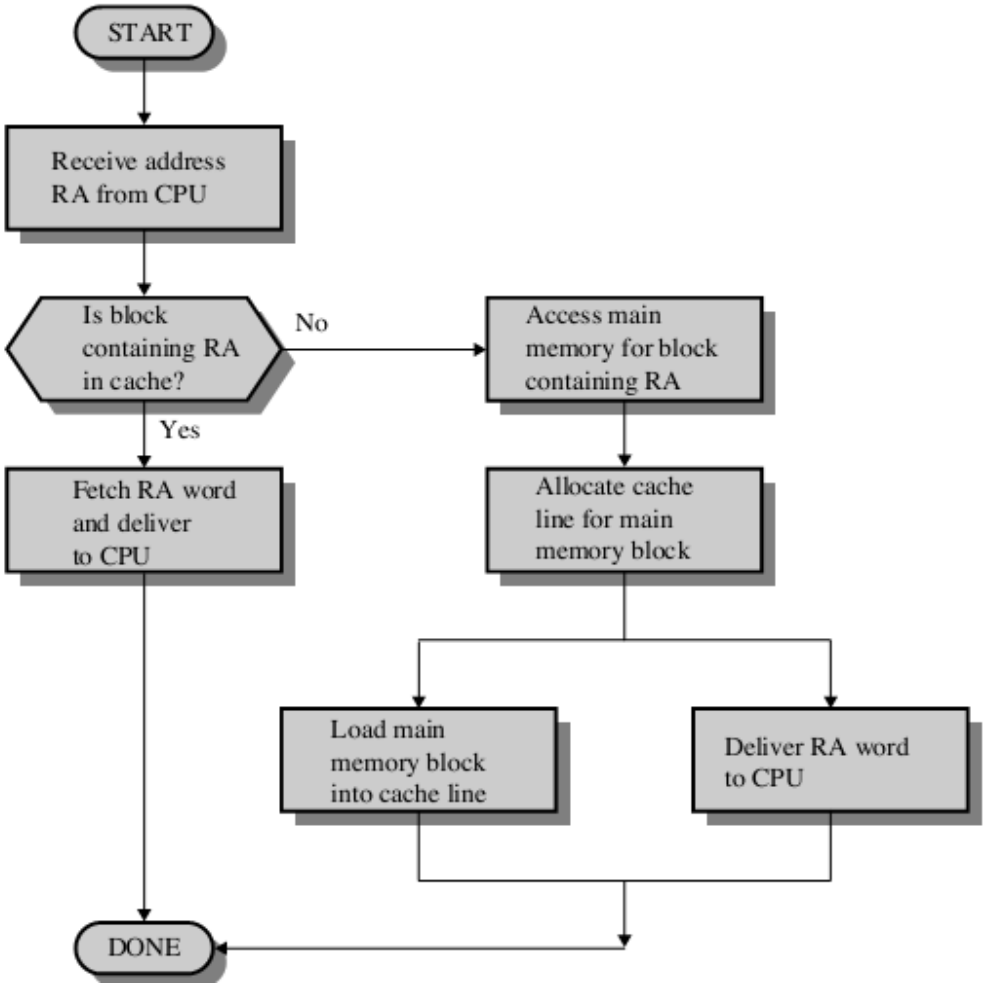
i.)Write-Back Policy:

It acts like a buffer i.e. when the processor starts the write cycle the cache receives the data and terminates the cycle. The cache then writes the data back to main memory, when the data system bus is available & provides greatest performance but it increases complexity & cost.

ii.)Write-Through Policy:

The processor writes through the cache to main memory. This method is less complex and expensive but has lower performance.

Figure shown below shows the Cache Read Operation



3.) Cache-Main Memory Structure:

→Main memory consists of upto 2^n addressable words with each word having a unique n-bit address.

→For mapping purpose, this memory is considered to consist of a number of fixed length of K words each.

→There are M= 2^n/K blocks in main memory.

→The Cache consist of m blocks called lines.

→Each line contains K words plus a tag of few bits.

→The length of a line not including tag and control bits is the line size.

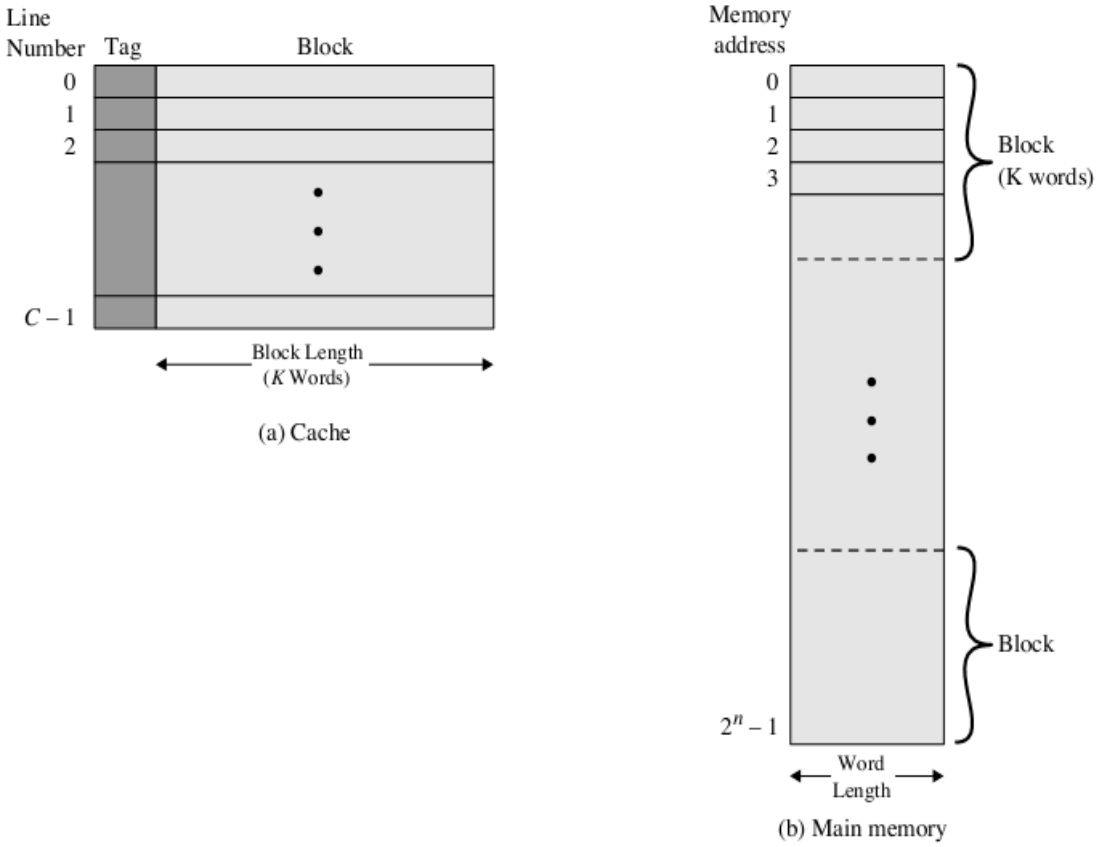
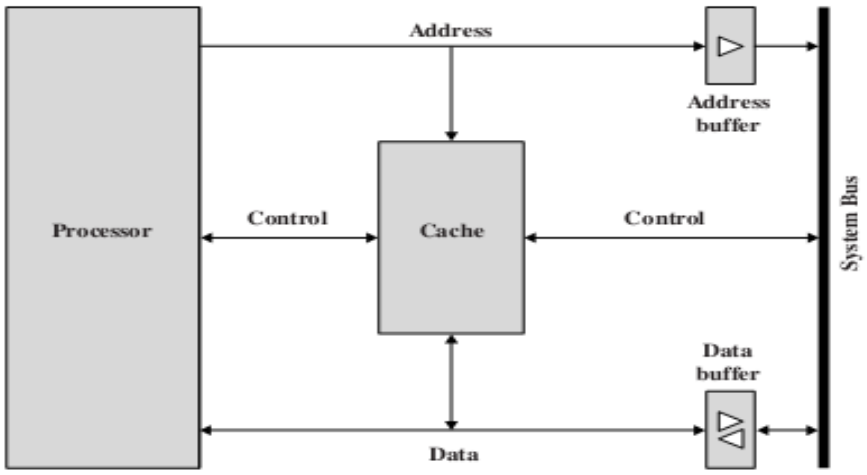


Figure Cache/Main-Memory Structure

CPU Organization of Cache:-



4.) Mapping function:-

Used to map a particular block of main memory to a particular block of main memory to a particular block of cache. The mapping function is used to transfer the block from main memory to cache memory. Three mapping function exists.

i.) Direct mapping

ii.) Associative mapping

iii.) Set Associative mapping

i.) Direct Mapping:

- Address length = (s + w) bits
- Number of addressable units = 2s+w words or bytes
- Block size = line size = 2w words or bytes
- Number of blocks in main memory = 2s+ w/2w = 2s
- Number of lines in cache = m = 2r
- Size of tag = (s – r) bits

In direct mapping, block K of main memory maps into block k modulo m (K%m) of the cache. Here m is the total no. of blocks in cache. Here value of m is 128.

Direct mapping pros n cons:

- Simple
- Inexpensive
- Fixed location for given block
    - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high

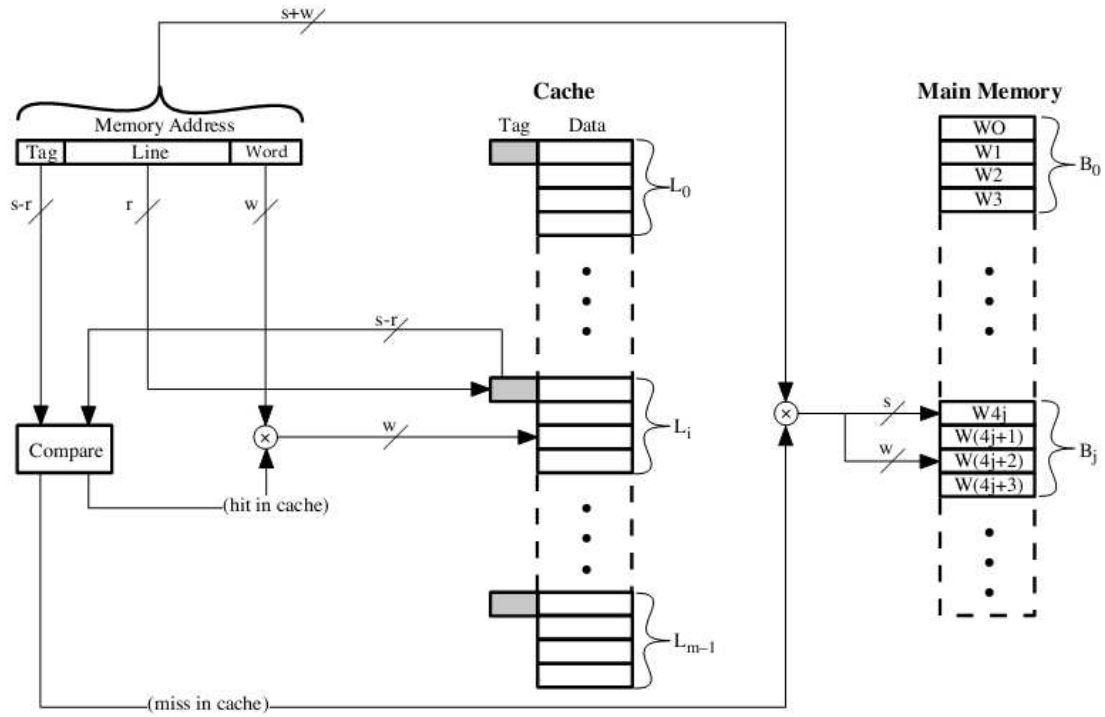


Figure: Direct mapping.

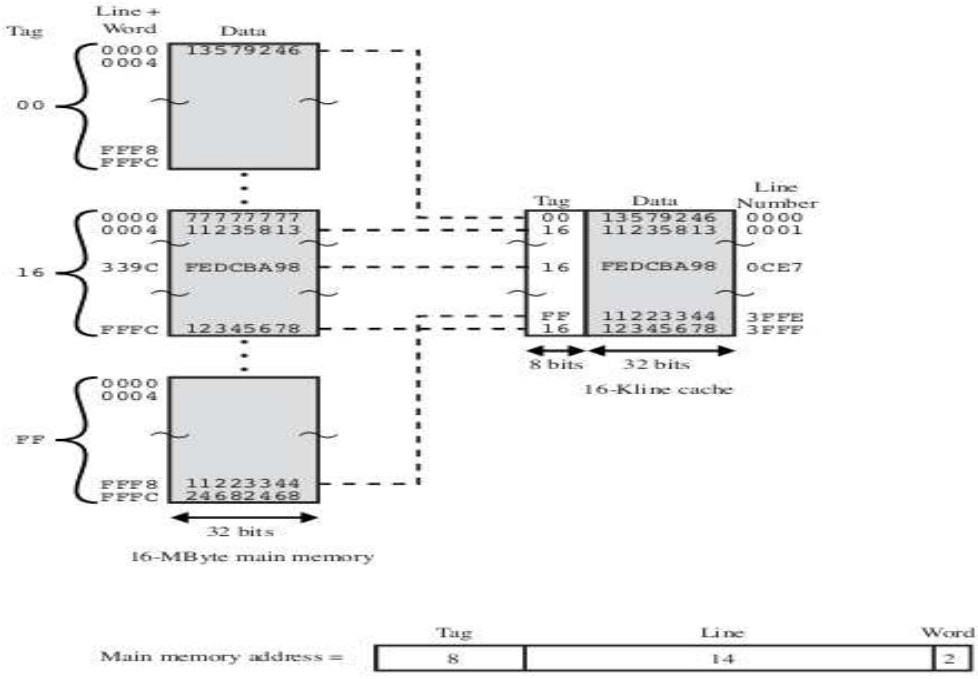


Figure: example of direct mapping.

ii.)Associative Mapping:

- Address length = (s + w) bits
- Number of addressable units = 2s+w words or bytes
- Block size = line size = 2w words or bytes
- Number of blocks in main memory = 2s+ w/2w = 2s
- Number of lines in cache = undetermined
- Size of tag = s bits

In this mapping technique, any block of main memory can potentially reside in any cache block position. It is much more flexible mapping technique.

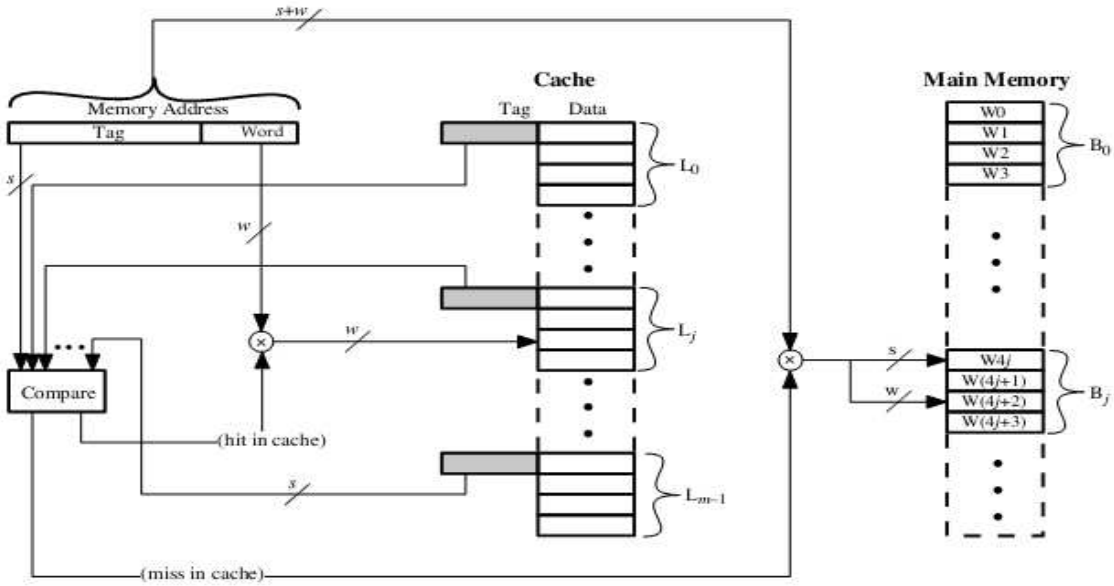


Figure: Associative Mapping

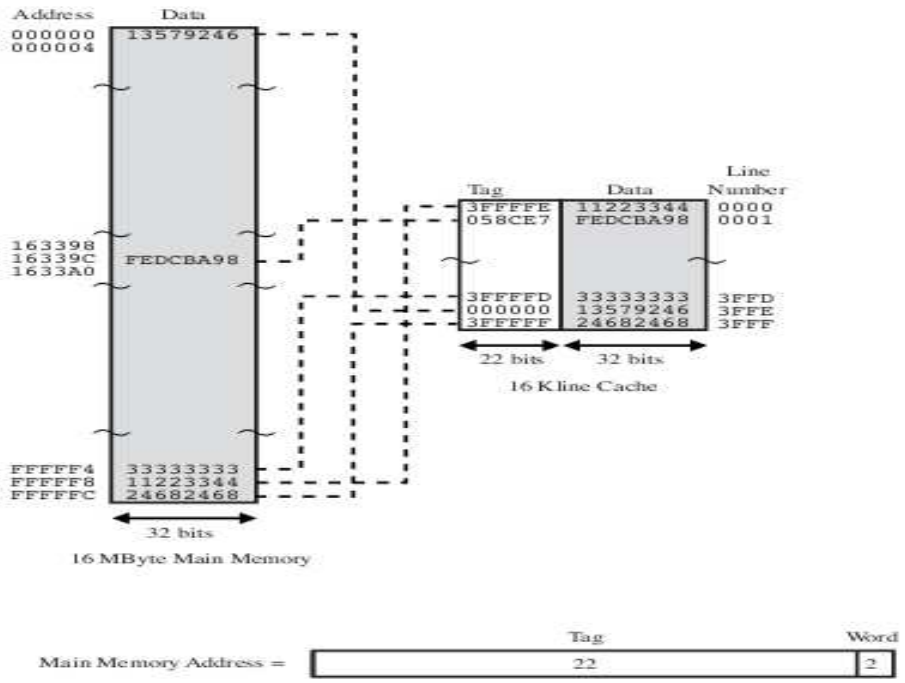


Figure: Example of Associative mapping

When the processor wants an address, all tag filed in the cache are checked to see if the data is already in cache.

<u>iii.) Set Associative mapping:</u>

- Address length = (s + w) bits
- Number of addressable units = 2s+w words or bytes
- Block size = line size = 2w words or bytes
- Number of blocks in main memory = 2d
- Number of lines in set = k
- Number of sets = v = 2d
- Number of lines in cache = kv = k * 2d
- Size of tag = (s – d) bits

In this method blocks of cache are grouped into sets, & the mapping allows a block of main memory to reside in any block of a specific set. From the flexibility point of view it is between the other two method.

This reduced searching overhead, because the search is restricted to no. of sets, instead of no. of block.

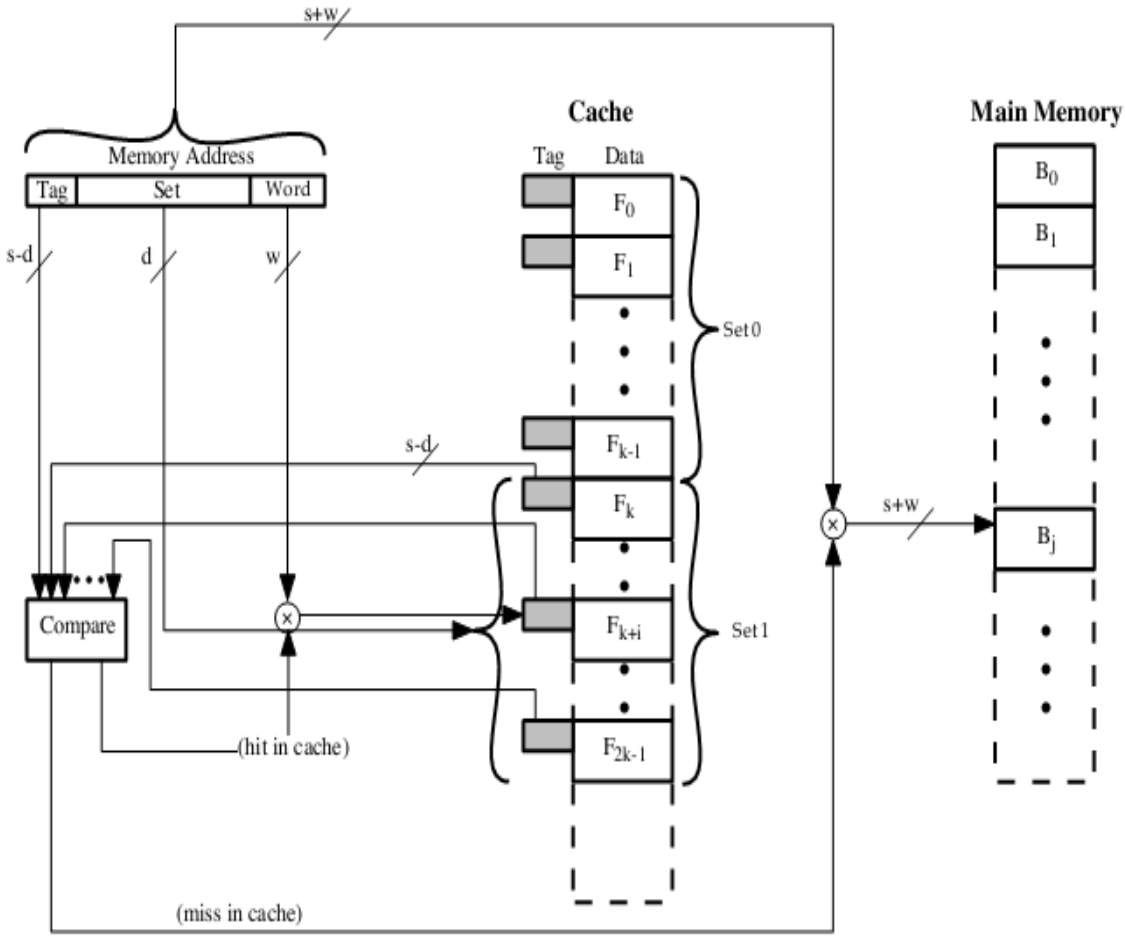


figure : set associative mapping

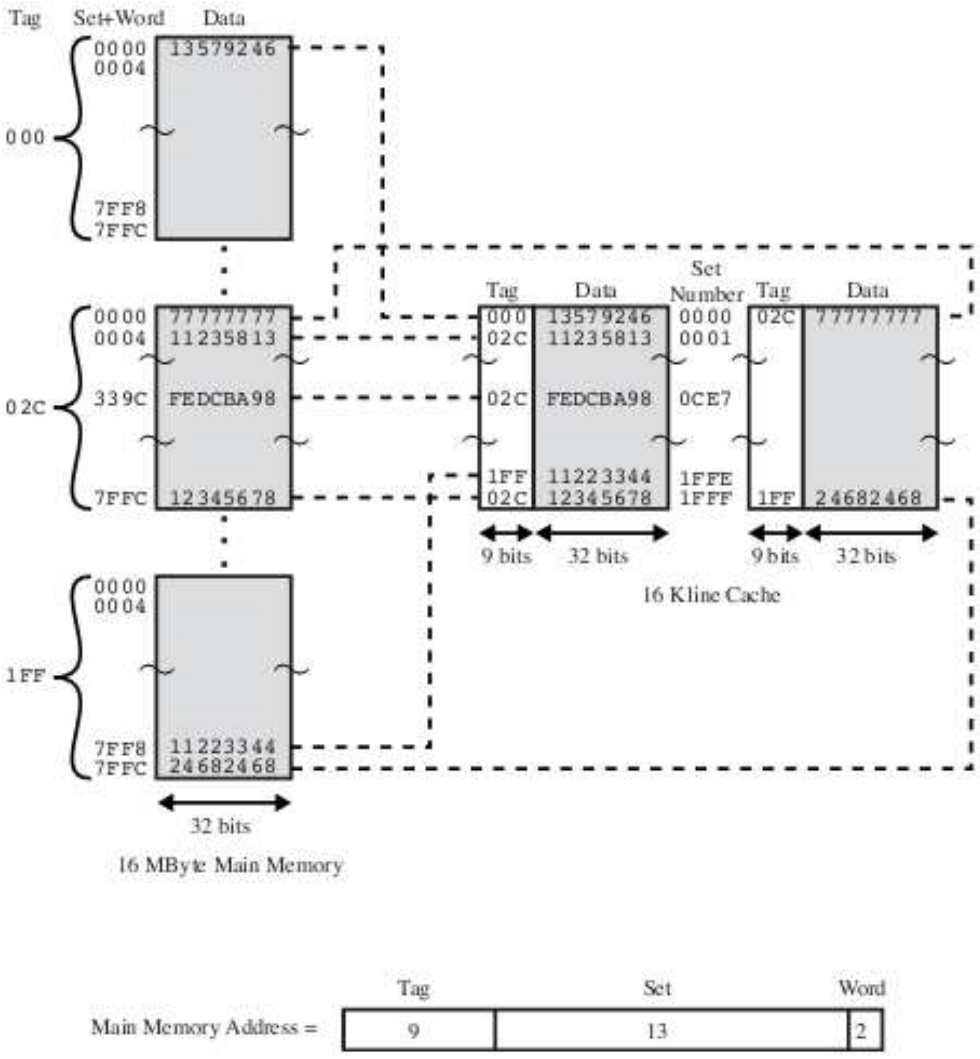


Figure: Example of set-associative

Some questions:

1. what are page replacement in cache?

ANS:

Replacement Algorithms
Direct mapping

- No choice

- Each block only maps to one line
- Replace that line

Replacement Algorithms
Associative & Set Associative

- Hardware implemented algorithm (speed)
- Least Recently used (LRU)
- e.g. in 2 ways set associative
    - Which of the 2 block is lru?
- First in first out (FIFO)
    - replace block that has been in cache longest
- Least frequently used
    - replace block which has had fewest hits
- Random

Chapter 6: RISC, CISC and Pipelining Techniques

1.) Parallel processing or multi processing:

→A parallel processing system is able to perform concurrent data processing to achieve faster execution time.

→The system may have two or more ALU and be able to execute two or more instruction at the same time.

→Parallel processing increase the amount of h/w required.

→Parallel processing can be achieved through

i.) Pipelining processing

ii.)Vector processing

iii.) Array processor

Pipelining: It is a technique of decomposing a sequential process into sub-operations with each sub-process being execute in a special dedicated segment that operates concurrently with all other segments.

The result obtained from the computation in each segment is transferred to the next segment in the pipeline.

Example:

Ai*Bi+C For i=1,2,3,…..7

The sub-operations performed in each segments are:

R1←Ai; , R2←Bi

R3←R1*R2 , R4←Ci

R5←R3+R4

Clock pulse no.	*Segment 1* R1	R2	*Segment 2* R3	R4	*Segment 3* R5
1	A1	B1			
2	A2	B2	A1*B1	C1	
3	A3	B3	A2*B2	C2	A1*B1+C1
4	A4	B4	A3*B3	C3	A2*B2+C2
5	A5	B5	A4*B4	C4	A3*B3+C3
6	A6	B6	A5*B5	C5	A4*B4+C4
7	A7	B7	A6*B6	C6	A5*B5+C5

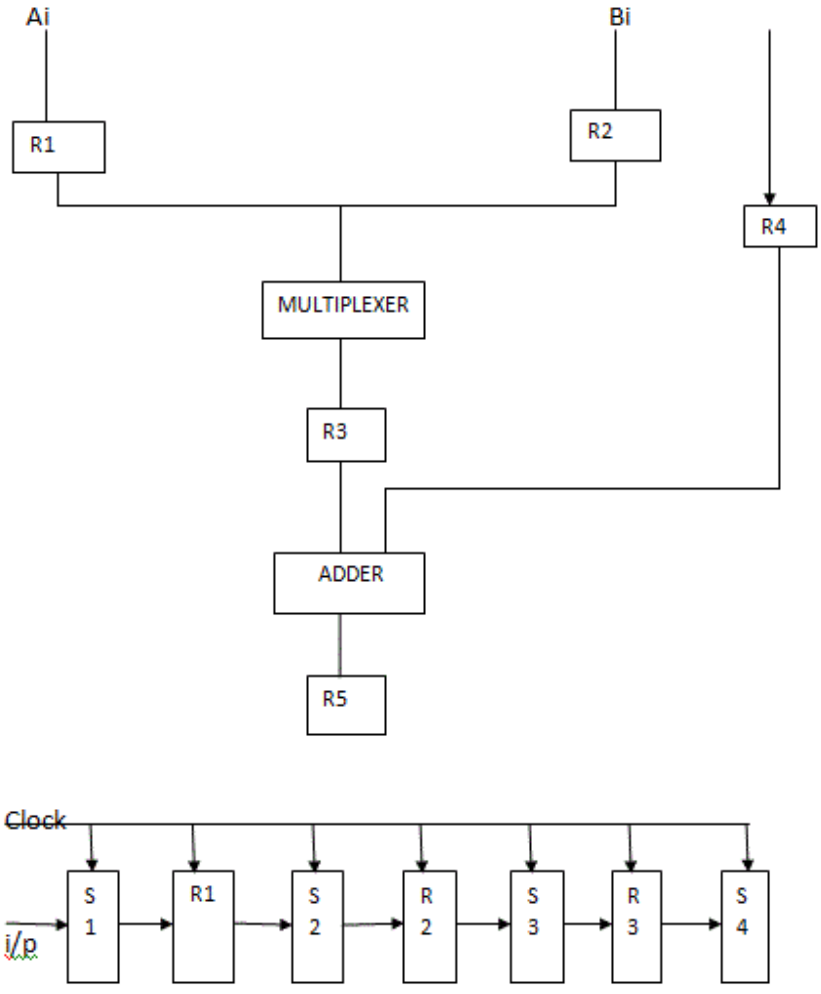


Figure : four segment pipeline

Segments	1	2	3	4	5	6	7	8	9
1	T1	T2	T3	T4	T5	T6			
2		T1	T2	T3	T4	T5	T6		
3			T1	T2	T3	T4	T5	T6	
4				T1	T2	T3	T4	T5	T6

Let us consider 6 tasks T1 through T6 executed 4 segments.

Serial execution total time Ts=? = 6*4 = 24 time units

Parallel execution total time= Tp = (4+6-1)= 9 time unit

Speed Up Ratio= Ts/Tp = 24/9

Q. A non pipeline system takes 50 ns to process a task. The same task can be processed in six segment pipeline with a clock cycle of 10 ns. Determine the speed up ratio of the pipeline for 100 tasks. What is maximum speed up that can be achieved?

ANS:

tn=50 ns, k=6, tp=10ns, n=100

Then,

Speed Up Ratio= n tn/[(k+n-1)*10]

= 100*10/[(6+99)*10]

= 4.76

Smax= tn/tp

= 50/10 =5

Therefore, Smax= 5

Arithmetic Pipeline:

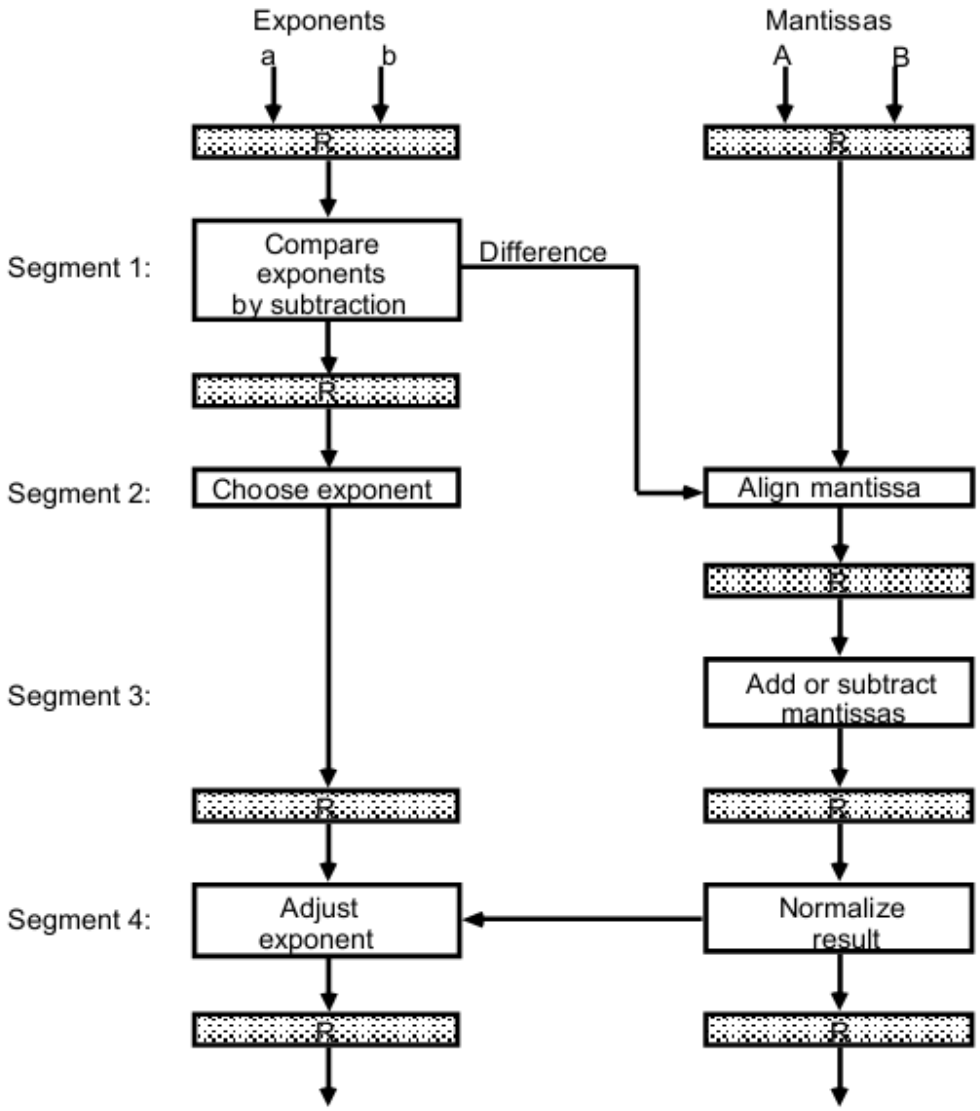


Figure : Pipeline for floating point addition & subtraction

→X=0.9504*10^3

Y=0.8260*10^2

→The two exp are subtracted in segment 1 to get 3-2=1

→The larger exp 3 is choose as the exp result.

→Segment 2 shifts the mantissa of Y to the right to obtain Y=0.0820*10^3

→Mantissas are now aligned.

→Segment 3 produces the sum

Z= 1.0324*10^3

→Segment 4 normalizes the result by shifting the mantissa once to the right & incrementing the exp by 1, to obtain,

Z=0.1032*10^4

Instruction Pipeline:

Following steps are needed to process each instruction.

i.)Fetch the instruction from memory.

ii.)Decode the instruction.

iii.)Calculate the effective address.

iv.)Fetch the operand from memory.

v.)Execute the instruction

vi.)Store the result in proper place.

Step 2 & 3 and 5 & 6 can be merged as

i.)FI is the segment that fetches instruction.

ii.)DA is the segment that decodes instruction and calculated the effective address.

iii.)FO is the segment that fetches operands

iv.)EX is the segment that execute & store results.

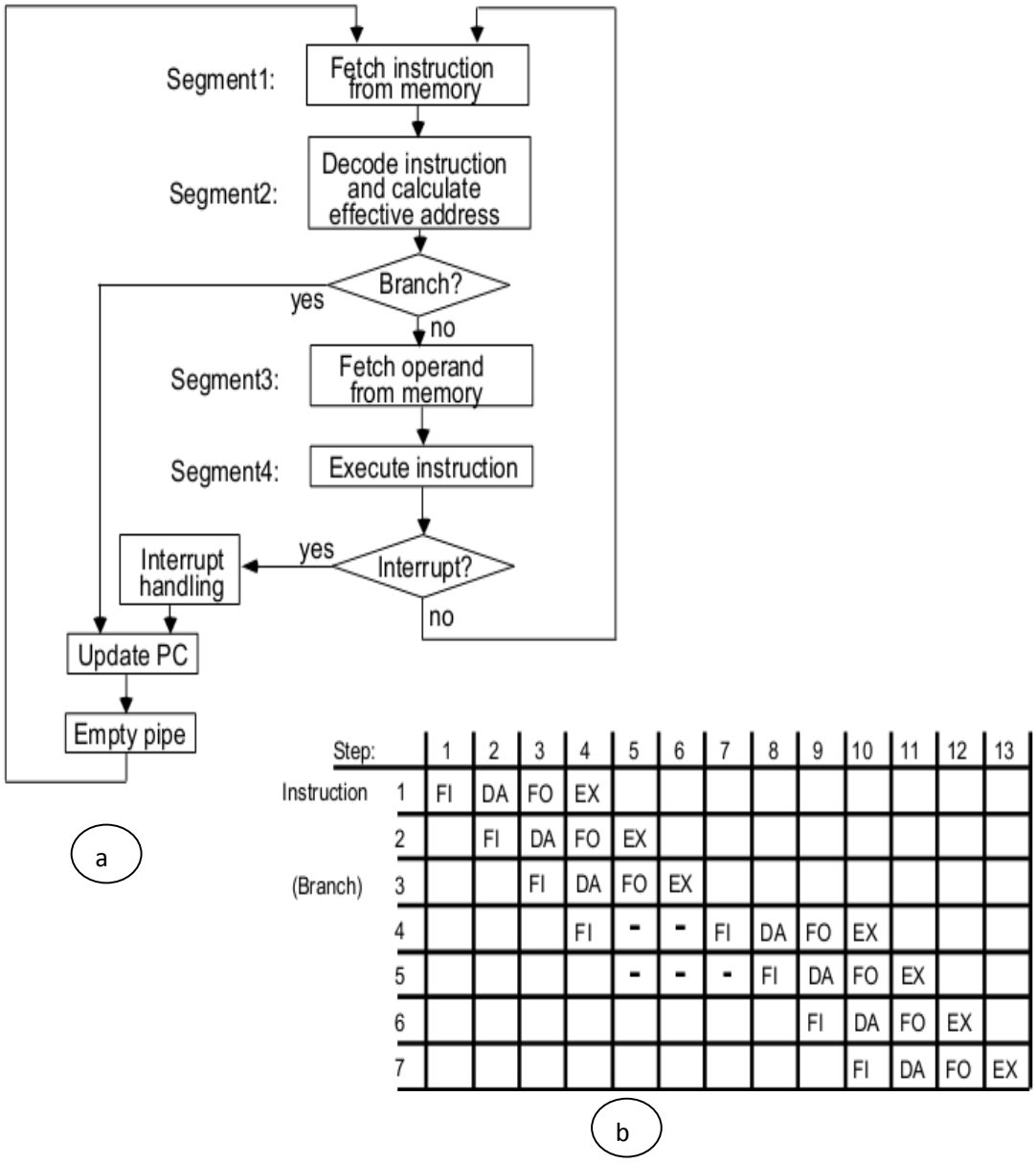


Figure : a.) Timing of instruction pipeline

b.) 4 segment CPU pipeline.

Pipelining Hazards:

These are the situations, called Hazard that prevent the next instruction in the instruction stream from being executing during its designated clock cycles. Hazards reduce the performance from ideal speed up gained by the processor.

Three types of Hazards

i.) Structural Hazards:

- Due to the non-availability of appropriate hardware
- One obvious way of avoiding structural hazard is to insert additional hardware into the pipeline.

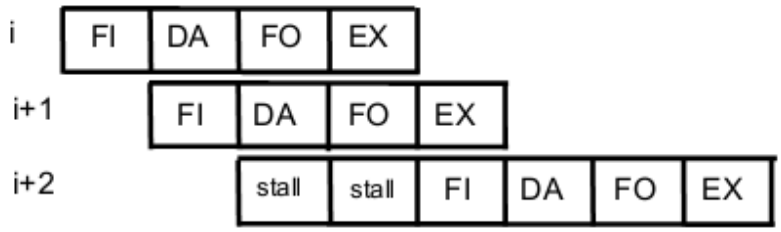
or

Occur when some resource has not been duplicated enough to allow all combinations of instructions in the pipeline to execute.

OR simply

Hazards occur when instruction needs a resource being used by another instruction.

Example: With one memory-port, a data and an instruction fetch cannot be initiated in the same clock



The Pipeline is stalled for a structural hazard

← Two Loads with one port memory

→Two-port memory will serve without stall

ii.)Data Hazards:

They arise when an instruction depends on the result of previous instruction.

3- Situations:

1. Read after write (RAW)

I1:R2←R1+R3

I2:R3←R2+R3

2. Write after read (WAR)

I1:R4←R1+R3

I2:R3←R1+R2

3. Write after write (WAW)

I1:R2←R4+R7

I2:R2←R1+R3

ii.) Control Hazards:

- Conditional branch instructions
  - The target address of branch will be known only after the evaluation of the condition.
- The ways to solve control hazards
  - The pipeline is frozen
  - The pipeline predicts that the branch will not be taken.
  - It would be to start fetching the target instruction sequence into a buffer while the non-branch sequence is being fed into the pipeline.

2.) RISC and CISC:

Reduced Instruction Set Computer (RISC)

Original idea to reduce the ISA

- Provide ***minimal set of instructions*** that could carry out all essential operations

Instruction complexity is reduced by

1. Having *few simple* instructions that are the *same length*

2. Allowed memory access only with *explicit* load and store instructions

Hence each instruction performs less work but instruction execution time among different instructions is consistent

The complexity that is removed from ISA is moved into the domain of the assembly programmer/compiler

Examples: LC3, MIPS, PowerPC (IBM), SPARC (Sun)

Complex Instruction Set Computer (CISC)

Memory in those days was expensive

- bigger program->more storage->more money

Hence needed to *reduce* the number of instructions per program

Number of instructions are reduced by having *multiple operations* within a single instruction

Multiple operations lead to many different kinds of instructions that access memory

- In turn making instruction length variable and fetch-decode-execute time unpredictable – making it more complex
- Thus hardware handles the complexity

Example: x86 ISA

RISC vs. CISC

The difference between CISC and RISC becomes evident through the basic computer performance equation:

$$\text{CPU Time} = \frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{avg. cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

RISC systems shorten execution time by reducing the *clock cycles per instruction* (i.e. simple instructions take less time to interpret)

CISC systems shorten execution time by reducing the *number of instructions per program*

RISC vs. CISC Summary

RISC

- Simple instructions, few in number
- Fixed length instructions
- Complexity in compiler
- Only **LOAD/STORE** instructions access memory
- Few addressing modes

CISC

- Many complex instructions
- Variable length instructions
- Complexity in microcode
- Many instructions can access memory
- Many addressing modes

RISC PIPELINE

RISC

- Machine with a very fast clock cycle that executes at the rate of one instruction per cycle

<- Simple Instruction Set

Fixed Length Instruction Format

Register-to-Register Operations

Instruction Cycles of Three-Stage Instruction Pipeline

→Data Manipulation Instructions

I: Instruction Fetch

E: Decode, Read Registers, ALU Operations

D: Write a Register

→ Load and Store Instructions

I: Instruction Fetch

E: Decode, Evaluate Effective Address

D: Register-to-Memory or Memory-to-Register

→Program Control Instructions

I: Instruction Fetch

E: Decode, Evaluate Branch Address

D: Write Register (PC)

Sequential Execute(shown below)

Load A←M I E D

Load B←M I E D

Add C←A+B I E

Store M ←C I E D

Branch X I E

Two way pipeline (shown below)

Load A←M I E D

Load B←M I E D

Add C←A+B I E

Store M ←C I E D

Branch X I E

Here I & E of two different instructions are performed simultaneously.

3-way pipelined timing (shown below)

Load A←M I E D

Load B←M I E D

Noop I E

Add C←A+B I E

Store M ←C I E D

Branch X I E

Noop I E

3.) Flynn's Taxonomy:-

- Single Instruction stream, Single Data stream (SISD)
  - Conventional uniprocessor
  - Although ILP is exploited
- Single Program Counter -> Single Instruction stream
- The data is not "streaming"

- Single Instruction stream, Multiple Data stream (SIMD)
  - Popular for some applications like image processing
  - One can construe vector processors to be of the SIMD type.
  - MMX extensions to ISA reflect the SIMD philosophy
    - Also apparent in "multimedia" processors (Equator Map-1000)
  - "Data Parallel" Programming paradigm
- Multiple Instruction stream, Single Data stream (MISD)
  - Until recently no processor that really fits this category
  - "Streaming" processors; each processor executes a kernel on a

stream of data

– Maybe VLIW?

- Multiple Instruction stream, Multiple Data stream (MIMD)
  – The most general
  – Covers:
    • Shared-memory multiprocessors
    • Message passing multicomputers (including networks of workstations cooperating on the same problem; grid computing)

Some Questions

1.) Explain the followings..

ANS:

Prefetch Target Instruction

– Fetch instructions in both streams, branch not taken and branch taken
– Both are saved until branch branch is executed. Then, select the right instruction stream and discard the wrong stream

Branch Target Buffer(BTB; Associative Memory)

– Entry: Addr of previously executed branches; Target instruction and the next few instructions

– When fetching an instruction, search BTB.

– If found, fetch the instruction stream in BTB;

– If not, new stream is fetched and update BTB

Loop Buffer(High Speed Register file)

– Storage of entire loop that allows to execute a loop without accessing memory

Branch Prediction

– Guessing the branch condition, and fetch an instruction stream based on

the guess. Correct guess eliminates the branch penalty

Delayed Branch

– Compiler detects the branch and rearranges the instruction sequence by inserting useful instructions that keep the pipeline busy

in the presence of a branch instruction

2.)Compare and contrast RISC Vs CISC processor.

ANS:

RISC	**CISC**
1. Multiple register set often consisting of more than 256 registers.	1. Single resistor set typically 6 to 16 registers in total
2. three register operands allowed per instruction e.g. Add R1, R2, R3	2. One or two register allowed per instruction. e.g. Add R1, R2
3. Hardware control	3. micro programmed control

4. Highly pipelined	4. less pipelined
5. Simple in instructions that are few in number	5. many complex instructions
6. Fixed length instruction	6. variable length instructions
7. complexity in compiler	7. complexity in micro-code
8. Few address modes	8. Many addressing modes
9. only load and stored instruction can access memory	9. Many instruction can access memory
10. single cycle instructions (except for load and store)	10. Multiple cycle instruction
11. parameter passing through efficient on chip register windows	11. parameter passing through inefficient chip memory
12. reduced instruction set computer	12. complex instruction set computer