SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

COMPUTER ARCHITECTURE AND ORGANIZATION (SCSA1402)

UNIT – I – Central Processing Unit – SCSA1402

UNIT.1 INTRODUCTION

Central Processing Unit - Introduction - General Register Organization - Stack organization -- Basic computer Organization - Computer Registers - Computer Instructions - Instruction Cycle. Arithmetic, Logic, Shift Microoperations- Arithmetic Logic Shift Unit -Example Architectures: MIPS, Power PC, RISC, CISC

Central Processing Unit

The part of the computer that performs the bulk of data-processing operations is called the central processing unit CPU. The CPU is made up of three major parts, as shown in Fig.1

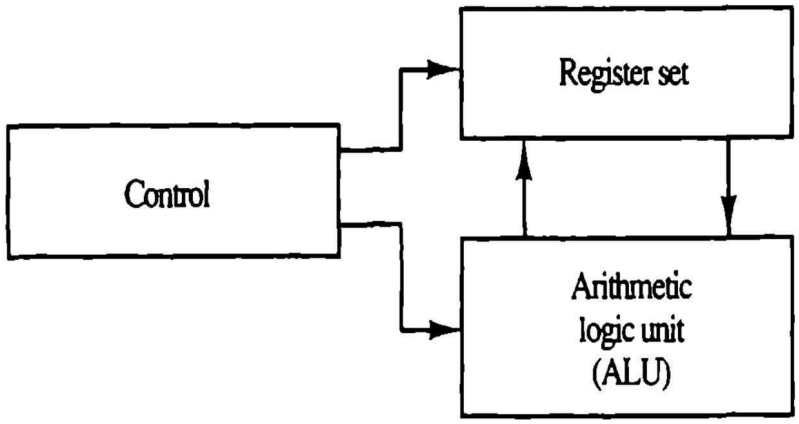


Fig 1. Major components of CPU.

- The register set stores intermediate data used during the execution of the instructions.
- The arithmetic logic unit (ALU) performs the required microoperations for executing the instructions.
- The control unit supervises the transfer of information among the registers and instructs the ALU as to which operation to perform.

General Register Organization

When a large number of registers are included in the CPU, it is most efficient to connect them through a common bus system. The registers communicate with each other not only for direct data transfers, but also while performing various microoperations.

Hence it is necessary to provide a common unit that can perform all the arithmetic, logic, and shift microoperations in the processor. A bus organization for seven CPU registers is shown in Fig.2.

- The output of each register is connected to two multiplexers (MUX) to form the two buses A and B.
- The selection lines in each multiplexer select one register or the input data for the particular bus. The A and B buses form the inputs to a common arithmetic logic unit (ALU).

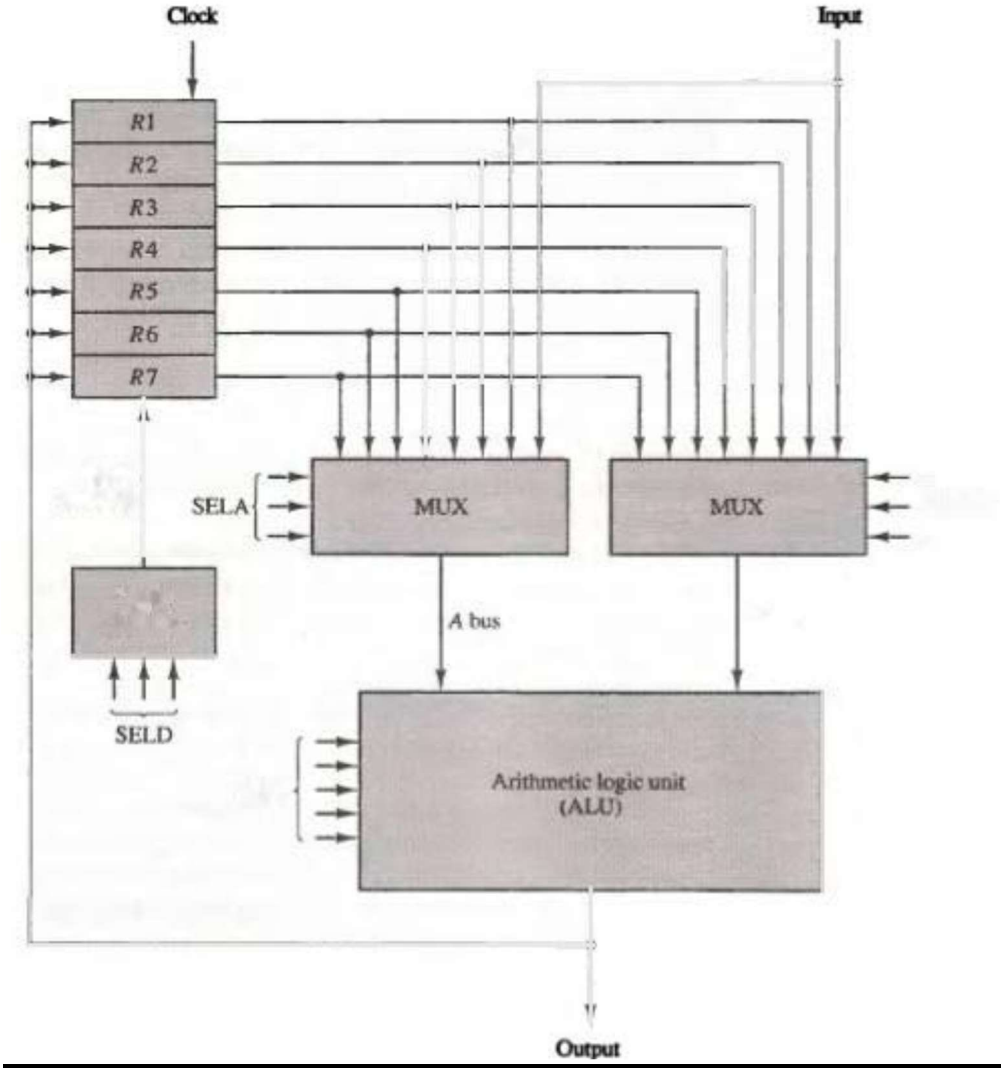


Fig 2 Register set with common ALU.

- The operation selected in the ALU determines the arithmetic or logic microoperation that is to be performed. The result of the microoperation is available for output data and also goes into the inputs of all the registers. The register that receives the information from the output bus is selected by a decoder. The decoder activates one of the register load inputs, thus providing a transfer path between the data in the output bus and the inputs of the selected destination register.
- The control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the system.
- For example, to perform the operation

R 1 <--R2 + R3

1. MUX A selector (SELA): to place the content of R2 into bus A.
2. MUX B selector (SELB): to place the content o f R 3 into bus B.
3. ALU operation selector (OPR): to provide the arithmetic addition A + B.
4. Decoder destination selector (SELD): to transfer the content of the output bus into R 1.

The four control selection variables are generated i n the control unit and must be available at the beginning of a clock cycle.

Control Word

There are 14 binary selection inputs in the unit, and their combined value specifies a control word. The 14-bit control word is defined in Fig. 4.

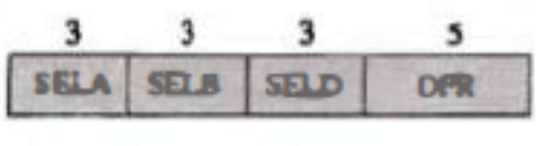


Fig 4. Control Word Format

TABLE 1 Encoding of Register Selection Fields

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

The encoding of the register selections is specified in Table 1. The 3-bit binary code listed in the first column of the table specifies the binary code for each of the three fields. The register selected by fields SELA, SELB, and SELD is the one whose decimal number is equivalent to the binary number in the code. When SELA or SELB is 000, the corresponding multiplexer selects the external input data. When SELD = 000, no destination register is selected but the contents of the output.

The ALU provides arithmetic and logic operations. The shifter may be placed in the input of the ALU to provide a preshift capability, or at the output of the ALU to provide postshifting capability. In some cases, the shift operations are included with the ALU. The function table for this ALU is listed in Fig.5. The encoding of the ALU operations for the CPU is specified in Table. The OPR field has five bits and each operation is designated with a symbolic name.

OPR Select	Operation	Symbol
00000	Transfer *A*	TSFA
00001	Increment *A*	INCA
00010	Add *A* + *B*	ADD
00101	Subtract *A* − *B*	SUB
00110	Decrement *A*	DECA
01000	AND *A* and *B*	AND
01010	OR *A* and *B*	OR
01100	XOR *A* and *B*	XOR
01110	Complement *A*	COMA
10000	Shift right *A*	SHRA
11000	Shift left *A*	SHLA

Fig.5 Encoding of ALU Operations

Stack Organization:

A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved. The operation of a stack can be compared to a stack of trays. The last tray placed on top of the stack is the first to be taken off. The register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top item in the stack. The two operations of a stack are the insertion and deletion of items. The operation of insertion is called push. The operation of deletion is called pop.

Register Stack

A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers. Figure 6 shows the organization of a 64-word register stack. The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack. Three items are placed in the stack: A, B, and C, in that order. Item C is on top of the stack so that the content of SP is now 3. To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of SP. Item B is now on top of the stack since SP holds address 2. To insert a new item, the stack is pushed by incrementing SP and writing a word in the next-higher location in the stack. Note that item C has been read out but not physically removed. This does not matter because when the stack is pushed, a new item is written in its place.

The one-bit register FULL is set to 1 when the stack is full, and the one-bit register EMTY is set to 1 when the stack is empty of items. DR is the data register that holds the binary data to be written into or read out of the stack.

Initially, SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0, so that SP points to the word at address 0 and the stack is marked empty and not full. If the stack is not full (if FULL = 0), a new item is inserted with a push operation. The push operation is implemented with the following sequence of microoperations:

DR <--M [SP] Read item from the top of stack

SP <--SP – 1 Decrement stack pointer

If (SP = 0) then (EMTY <--1) Check if stack is empty

FULL <--0 Mark the stack not full

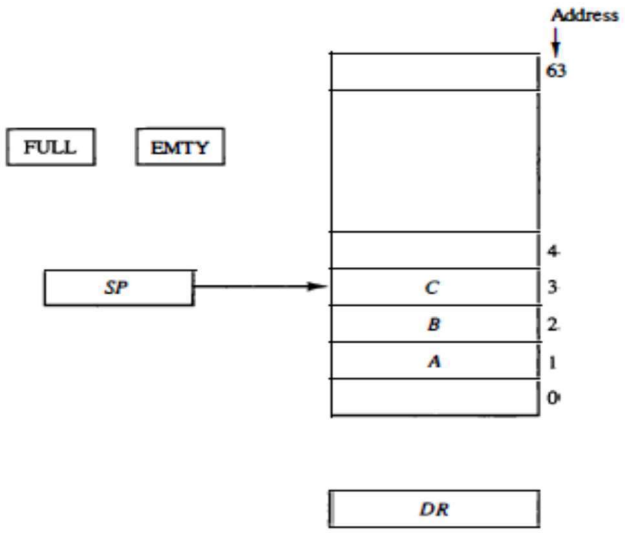


Fig.6 Block diagram of a 64 word stack.

The top item is read from the stack into DR. The stack pointer is then decremented. If its value reaches zero, the stack is empty, so EMTY is set to 1. This condition is reached if the item read was in location L Once this item is read out, SP is decremented and reaches the value 0, which is the initial value of SP.

Memory Stack:

A stack can exist as a stand-alone unit as in Fig. 6 or can be implemented in a random-access memory attached to a CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. Fig 7 shows a portion of computer memory partitioned into three segments: program, data, and stack. The program counter PC points at the address of the next instruction in the program. The address register AR points at an array of data. The stack pointer SP points at the top of the stack. The three registers are connected to a common address bus, and either one can provide an address for memory. PC is used during the fetch phase to read an instruction. AR is used during the execute phase to read an operand. SP is used to push or pop items into or from the stack. As shown in Fig.7, the initial value of SP is 4001 and the stack grows with decreasing addresses. Thus the first item stored in the stack is at address 4000, the second item is stored at address 3999, and the last address that can be used for the stack is 3000. No provisions are available for stack limit checks.

We assume that the items in the stack communicate with a data register DR. A new item is inserted with the push operation as follows

SP <-SP - 1

M [SP] <-DR

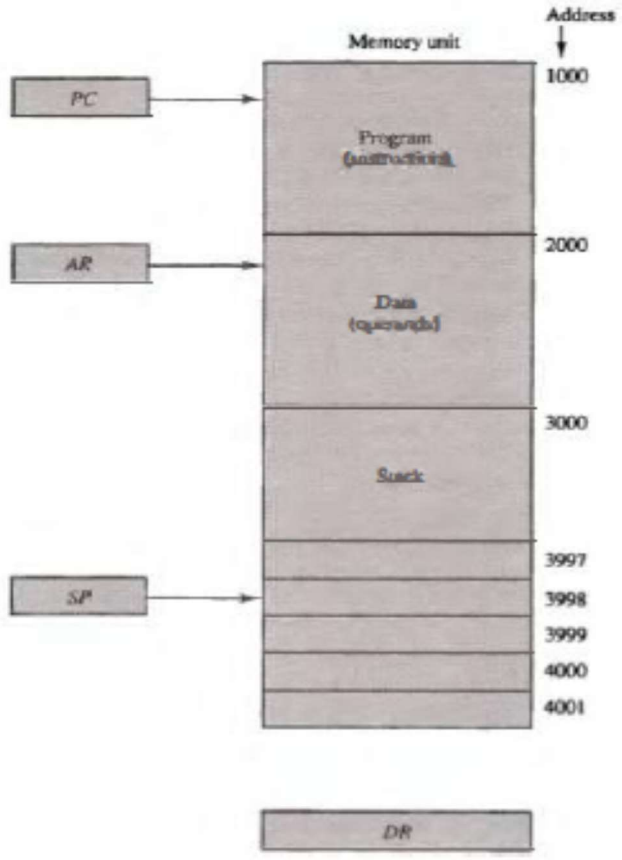


Fig .7 Computer memory with Program, data, and sack segments

The stack pointer is decremented so that it points at the address of the next word. A memory write operation inserts the word from DR into the top of the stack. A new item is deleted with a pop operation as follows:

DR <-M [SP]

SP <-SP + 1

Instruction Formats:

The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register. The bits of the instruction are divided into groups called fields. The most common fields found in instruction formats are:

1. An operation code field that specifies the operation to be performed.
2. An address field that designates a memory address or a processor register.
3. A mode field that specifies the way the operand or the effective address is determined.

The operation code field of an instruction is a group of bits that define various processor operations, such as add, subtract, complement, and shift.

Three-Address Instructions

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand. The program in assembly language that evaluates X = (A + B) • (C + D) is shown below, together with comments that explain the register transfer operation of each instruction.

```
ADD     R 1, A, B    R 1 <--M [A] + M [B]
ADD     R2, C, D     R2 <--M [C] + M [D]
MUL     X, R 1, R 2  M [X] <--R 1 • R 2
```

It is assumed that the computer has two processor registers, R 1 and R2. The symbol M [A] denotes the operand at memory address symbolized by A. The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

Two-Address Instructions

Two-address instructions are the most common in commercial computers. Here again each address field can specify either a processor register or a memory word. The program to evaluate X = (A + B) • (C + D) is as follows:

```
MOV     R 1, A   R 1 <--M [A]
ADD     R 1, B   R 1 <--R 1 + M [B]
```

MOV R 2, C R2 <--M [C]

ADD R2, D R2 <--R 2 + M [D]

MUL R1, R2 R 1 <--R 1 • R 2

MOV X, R 1 M [X] <--R 1

The MOV instruction moves or transfers the operands t o and from memory and processor registers. The first symbol listed in an instruction is assumed to be both a source and the destination where the result of the operation is transferred.

One-Address Instructions

One-address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication and division there is a need for a second register. The program to evaluate

X = (A + B) • (C + D) is

L O A D A AC <- M [A]

A DD B A C <- A C + M [B]

S T O R E T M [T] <- A C

L O A D C A C <- M [C]

A D D D A C <- A C + M [D]

M U L T A C <- A C • M [T]

S T O R E X M [X] <- A C

Zero-Address Instructions

A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack. The following program shows how X = (A + B) • (C + D) will be written for a stack organized computer. (TOS stands for top of stack.)

P U S H A T O S <- A

PUSH	B	TOS <- B
ADD		TOS <- (A + B)
PUSH	C	TOS <- C
PUSH	D	TOS <- D
ADD		TOS <- (C + D)
MUL		TOS <- (C + D) • (A + B)
POP	X	M [X] <- TOS

Computer Registers

The memory unit has a capacity of 4096 words and each word contains 16 bits. Twelve bits of an instruction word are needed to specify the address of an operand. This leaves three bits for the operation part of the instruction and a bit to specify a direct or indirect address. The data register (DR) holds the operand read from memory. The accumulator (AC) register is a general purpose processing register. The instruction read from memory is placed in the instruction register (IR). The temporary register (TR) is used for holding temporary data during the processing.

The memory address register (AR) has 12 bits since this is the width of a memory address. The program counter (PC) also has 12 bits and it holds the address of the next instruction to be read from memory after the current instruction is executed. The PC goes through a counting sequence and causes the computer to read sequential instructions previously stored in memory. Instruction words are read and executed in sequence unless a branch instruction is encountered. A branch instruction calls for a transfer to a nonconsecutive instruction in the program. The address part of a branch instruction is transferred to PC to become the address of the next instruction. To read an instruction, the content of PC is taken as the address for memory and a memory read cycle is initiated. PC is then incremented by one, so it holds the address of the next instruction in sequence. Two registers are used for input and output. The input register (INPR) receives an 8-bit character from an input device. The output register (OUTR) holds an 8-bit character for an output device.

Register symbol	Number of bits	Register name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

Fig .8 List of Registers for the Basic Computer

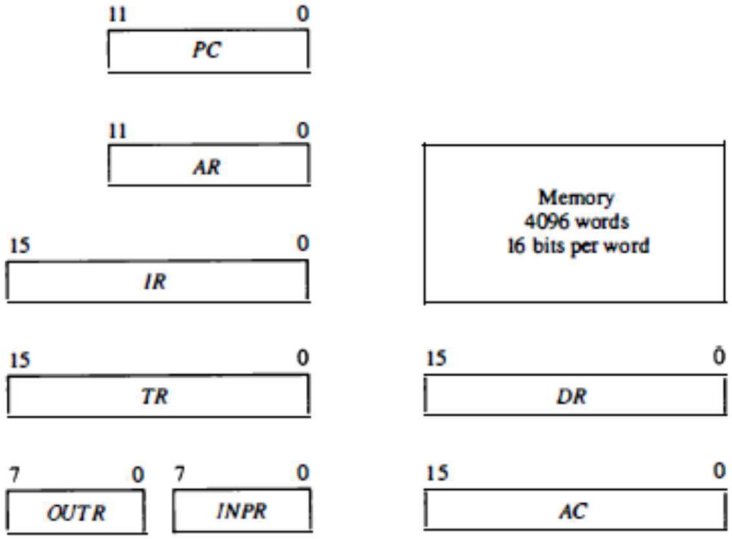


Figure 9. Basic computer registers and memory.

Computer Instructions:

The basic computer has three instruction code formats, as shown in Fig. 10. Each format has 16 bits. The operation code (opcode) part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered.

- A memory-reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I. I is equal to 0 for direct address and to 1 for indirect address. The register reference instructions are recognized by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction.

- A register-reference instruction specifies an operation on or a test of the AC register. An operand from memory is not needed; therefore, the other 12 bits are used to specify the operation or test to be executed.
- A input-output instruction does not need a reference to memory and is recognized by the operation code Ill with a 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type of input-output operation or test performed.

The type of instruction is recognized by the computer control from the four bits in positions 12 through 15 of the instruction. If the three opcode bits in positions 12 through 14 are not equal to 111, the instruction is a memory-reference type and the bit in position 15 is taken as the addressing mode I. If the 3-bit opcode is equal to 111, control then inspects the bit in position 15. If this bit is 0, the instruction is a register-reference type. If the bit is 1, the instruction is an input -output type. Note that the bit in position 15 of the instruction code is designated by the symbol I but is not used as a mode bit when the operation code is equal to 111.

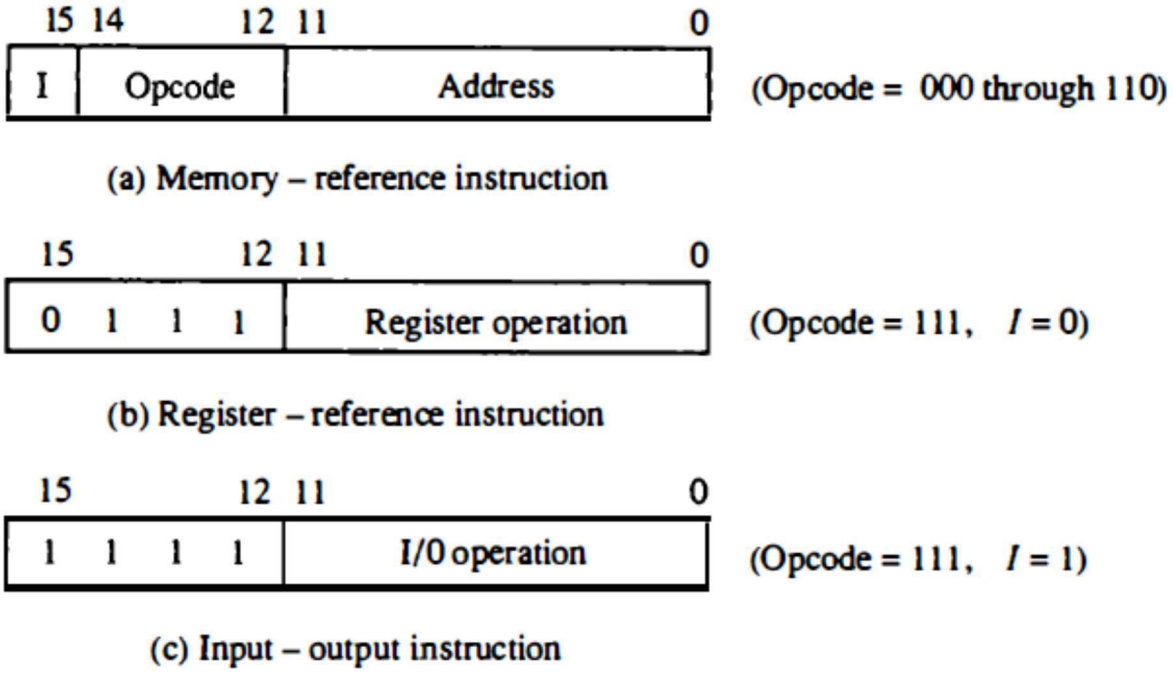


Fig 10. Basic computer instruction formats.

Symbol	Hexadecimal code $I = 0$	$I = 1$	Description
AND	0xxx	8xxx	AND memory word to *AC*
ADD	1xxx	9xxx	Add memory word to *AC*
LDA	2xxx	Axxx	Load memory word to *AC*
STA	3xxx	Bxxx	Store content of *AC* in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear *AC*
CLE	7400		Clear *E*
CMA	7200		Complement *AC*
CME	7100		Complement *E*
CIR	7080		Circulate right *AC* and *E*
CIL	7040		Circulate left *AC* and *E*
INC	7020		Increment *AC*
SPA	7010		Skip next instruction if *AC* positive
SNA	7008		Skip next instruction if *AC* negative
SZA	7004		Skip next instruction if *AC* zero
SZE	7002		Skip next instruction if *E* is 0
HLT	7001		Halt computer
INP	F800		Input character to *AC*
OUT	F400		Output character from *AC*
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

Fig 11.Basic Computer Instructions

Stack Organization:

Stack is a storage device that stores information in a way that the item is stored last is the first to be retrieved (LIFO). Stack in computers is actually a memory unit with address register (stack pointer SP) that can count only. SP value always points at top item in stack.

The two operations done on stack are,

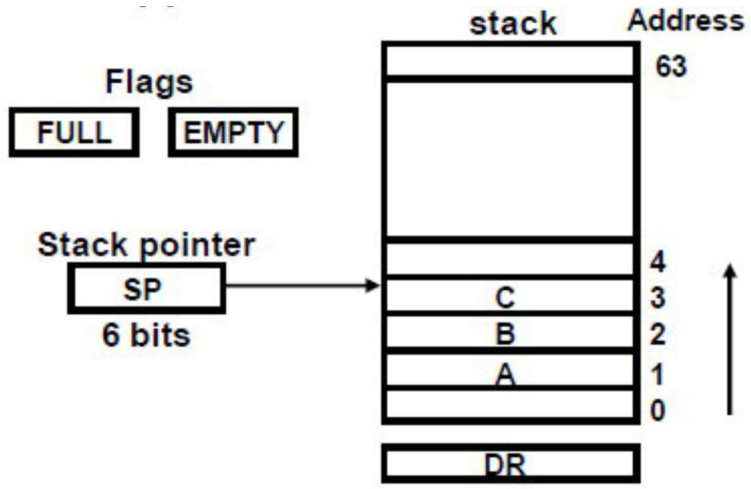
PUSH (Push Down), operation of insertion of items into stack

POP (Pop Up), operation of deletion item from stack

Those operation are simulated by INC and DEC stack register (SP).

1. Register stack:

A stand alone unit that consists of collection of finite number of registers. The next example shows 64 location stack unit with SP that stores address of the word that is currently on the top of stack.



Note that 3 items are placed in the stack A, B, and C. Item C is in top of stack so that SP holds 3 which the address of item C. To remove top item from stack (popping stack) we start by reading content of address 3 and decrementing the content of SP. Item B is now in top of stack holding address 2.

To insert new item (pushing the stack) we start by incrementing SP then writing a new word where SP now points to (top of stack).

Note that in 64-word stack we need to have SP of 6 bits only (from 000000 to 111111). If 111111 is reached then at next push SP will be 000000, that is when the stack is FULL. Similarly, when SP is 000001 then at next pop SP will go to 000000 that is when the stack is EMTY.

Initially, SP = 0, EMPTY = 1, FULL =0

Procedures for pushing stack

SP <- SP + 1

M[SP] <- DR

IF (SP = 0) THEN (FULL = 1)

EMTY <- 0

Note that:

1. Always we use DR to pass word into stack
2. M[SP] memory word specified by address currently in SP
3. First item stored in stack is at address 1
4. Last item stored in stack is at address 0. That is FULL = 1
5. Any push to stack means EMTY = 0

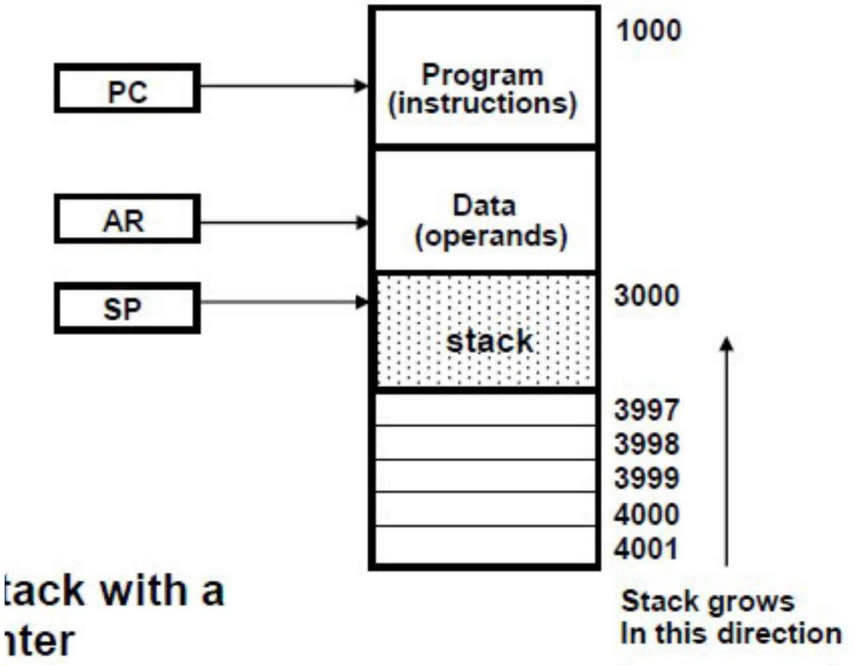
2. Memory Stack :
   - Stack can be implemented in RAM memory attached to CPU. Only by assigning special part of it for stack operations.
   - Next figure shows of main memory divided into program, data, and stack.
   - PC points to next instruction in instruction part
   - AR points to array of data of operands
   - SP points to top of stack All are connected to common address bus
   - Stack grows (pushed) with decreasing address and empties (pops) with increasing address.
   - New item is inserted with push operation by decrementing SP then a write to SP address is done

     SP <-SP -1

     M [SP]<- DR
   - Last item is removed from stack with pop operation by removing item by reading from memory location addressed by SP then SP is incremented.

     DR <- M [SP]

     SP <- SP +1



- As shown in figure initial value of SP is 4001 and first item when pushed in stack stores at address 4000 and second one stores at address 3999. The last address pushed into will be 3000. (See limitation danger?)
- Most computers are not supported by hardware to sense stack overflow and underflow. But can be implemented by saving the 2 limits in 2 registers. After each push or pop the SP is compared with the limit to see if stack has reached its limits. So must be taking care of using software.
- Always in this way we load SP with bottom address of stack portion of memory

Reverse Polish Notation:

- Very useful notation to utilize stacks to evaluate arithmetic expressions.

We write in infix notation such as:

$$A*B + C*D$$

We compute A*B, store product, compute C*D, then sum two products. So we have to scam back and forth to see which operation comes first.

The 3 notations to evaluate expressions

1. A + B Infix notation
2. +AB Prefix notation (Polish notation)
3. AB+ Postfix notation (reverse Polish

Reverse Polish Notation is in a form suitable for stack manipulation. Starts by scanning expression from left to right. When operator is found then perform

Instruction Format:

Operation with 2 operands in left of operator and replace result place of 2 operands and operator. Then you can continue this until you reach final answer.

Example

Expression A*B + C*D is written in RPN as AB*CD*+. And will computed as

(A*B) CD *+

(A*B)(C*D) +

Example

Convert infix notation expression (A + B)*(C * (D + E) + F) to RPN?

AB+ DE+ C * F+*.

Will be computed as

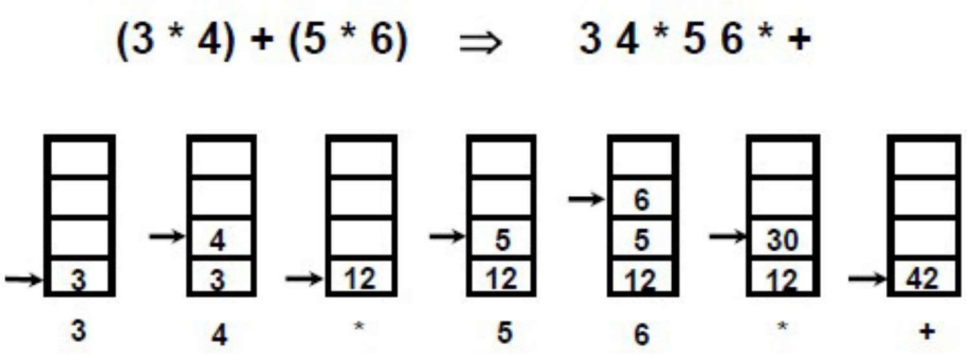
(A+B) (D+E) C * F + *

- Reverse polish notation combined with stack comprised of registers is most efficient way to evaluate expression. Stacks are good for handling long and complex problems involving chain calculations. But need first to convert arithmetic expressions into parenthesis-free reverse polish notation.
- This procedure is employed in some scientific calculators and some computers.

Example

Convert (3*4) (5*6) to RPN

34*56*+



The Instruction coding fields in today's computers follow the next format

1. Operation code field to specify operation
2. Address field that specifies operand address field or register
3. Mode field to specify effective address

In general, most processors are organized in one of 3 ways

- Single register (Accumulator) organization
  - Basic Computer is a good example
  - Accumulator is the only general-purpose register
- General register organization
  - Used by most modern computer processors
  - Any of the registers can be used as the source or destination for computer operations
- Stack organization
  - All operations are done using the hardware stack
  - For example, an OR instruction will pop the two top elements from the stack, do a logical OR on them, and push the result on the stack
- We are interested with address field of instructions with multiple address fields in instructions. The number of address fields in the instruction format depends on the internal organization of CPU. Some CPU combines features from more of one structure.

Instruction Cycle:

A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle for each instruction. Each instruction cycle in turn is subdivided into a sequence of subcycles or phases. In the basic computer each instruction cycle consists of the following phases:

1. Fetch an instruction from memory.
2. Decode the instruction.
3. Read the effective address from memory if the instruction has an indirect address.
4. Execute the instruction.

Fetch and Decode:

Initially, the program counter PC is loaded with the address of the first instruction in the program. The sequence counter SC is cleared to 0, providing a decoded timing signal To. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T0, T1, T2, and so on. The micro-operations for the fetch and decode phases can be specified by the following register transfer statements.

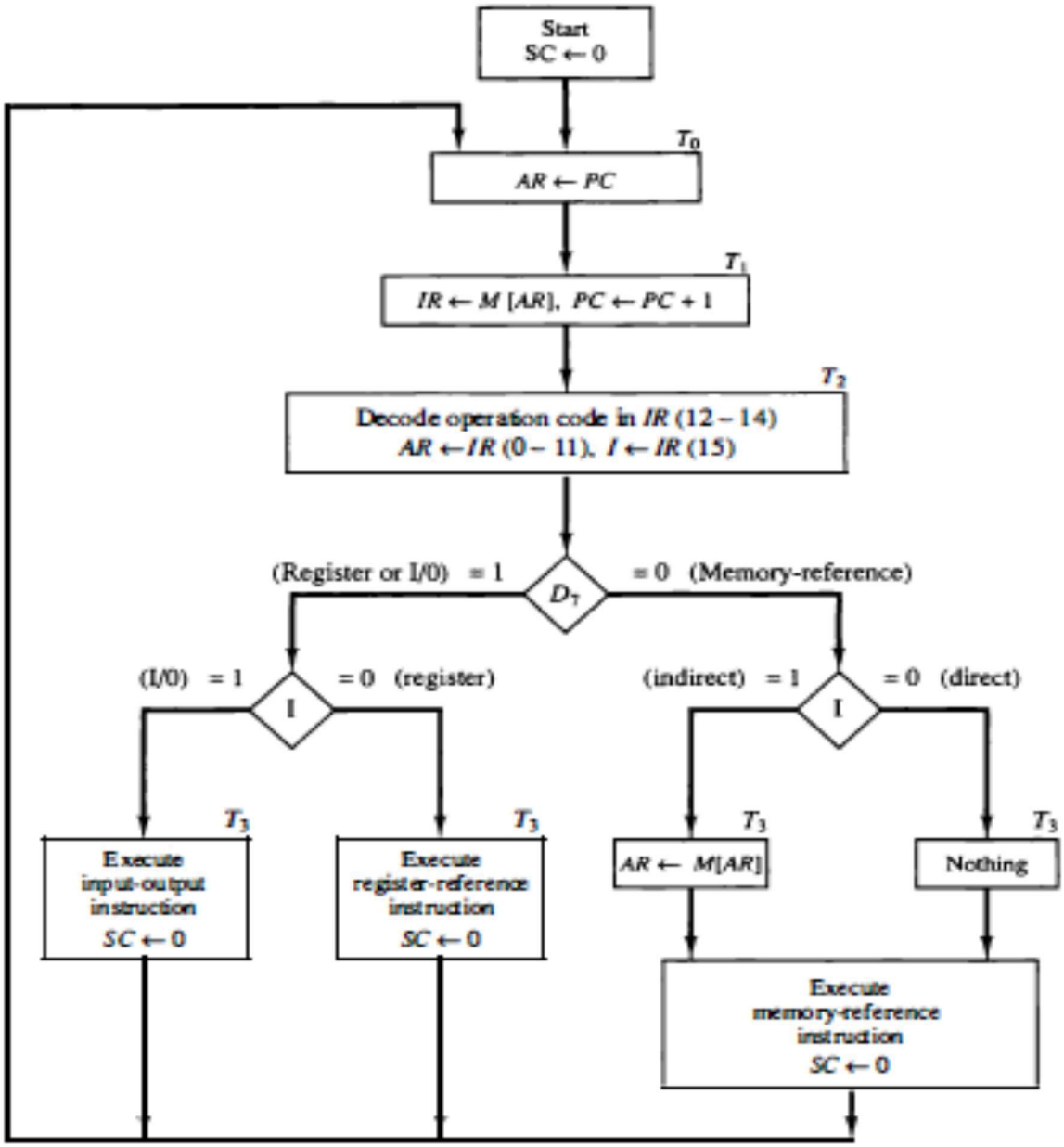
T0: AR <- PC

T,: IR <-M[AR], PC <- PC + 1

T2: D0, • • • , D7 <- Decode IR(12-14), AR <--- IR(0-11), 1 <--- IR(lS)

Since only AR is connected to the address inputs of memory, it is necessary to transfer the address from PC to AR during the clock transition associated with timing signal T0• The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing signal T1•

At the same time, PC is incremented by one to prepare it for the address of the next instruction in the program. At time T2, the operation code in IR is decoded, the indirect bit is transferred to flip-flop I, and the address part of the instruction is transferred to AR . Note that SC is incremented after each clock pulse to produce the sequence To, T1, and T2•



TEXT / REFERENCE BOOKS

1. M.Morris Mano, ;Computer System Architecture",Prentice-Hall Publishers,Third Edition.
2. John P Hayes , 'Computer Architecture and Organization', McGraw Hill international edition, Third Edition.
3. Kai Hwang and Faye A Briggs ,'Computer Architecture and Parallel Processing', McGraw Hill international edition,1995.

RISC Architecture

RISC (Reduced Instruction Set Computer) is used in portable devices due to its power efficiency. For Example, Apple iPod and Nintendo DS. RISC is a type of microprocessor architecture that uses highly-optimized set of instructions. RISC does the opposite, reducing the cycles per instruction at the cost of the number of instructions per program Pipelining is one of the unique feature of RISC. It is performed by overlapping the execution of several instructions in a pipeline fashion. It has a high performance advantage over CISC.

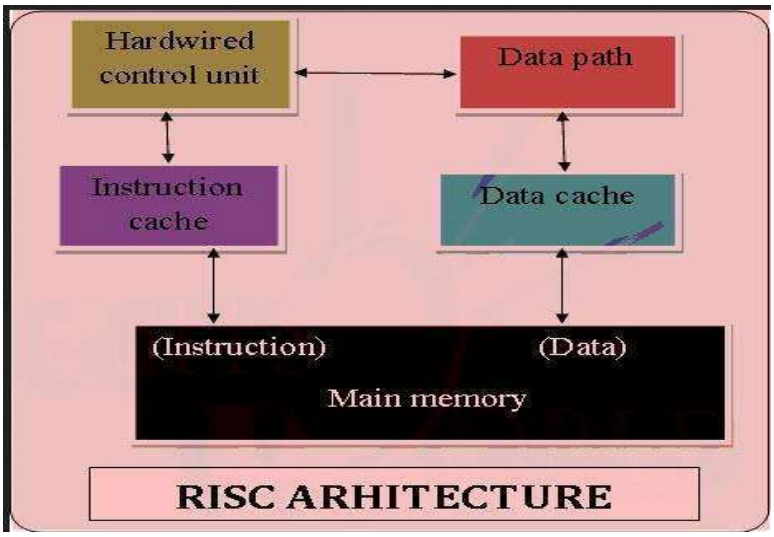


Figure : RISC Architecture

RISC ARCHITECTURE CHARACTERISTICS

Simple Instructions are used in RISC architecture. RISC helps and supports few simple data types and synthesize complex data types. RISC utilizes simple addressing modes and fixed length instructions for pipelining. RISC permits any register to use in any context. One Cycle Execution Time The amount of work that a computer can perform is reduced by separating "LOAD" and "STORE" instructions. RISC contains Large Number of Registers in order to prevent various number of interactions with memory. In RISC, Pipelining is easy as the execution of all instructions will be done in a uniform interval of time i.e. one click. In RISC, more RAM is required to store assembly level instructions. Reduced instructions need a less number of transistors in RISC. RISC uses Harvard memory model means it is Harvard Architecture. A compiler is used to perform the conversion operation means to convert a high-level language statement into the code of its form.

RISC & CISC COMPARISON

CISC	RISC
It is prominent on Hardware	It is prominent on the Software
It has high cycles per second	It has low cycles per second
It has transistors used for storing Instructions which are complex	More transistors are used for storing memory
LOAD and STORE memory-to-memory is induced in instructions	LOAD and STORE register-register are independent
It has multi-clock	It has a single - clock

MUL instruction is divided into three instructions
"LOAD" – moves data from the memory bank to a register
"PROD" – finds product of two operands located within the registers
"STORE" – moves data from a register to the memory banks
The main difference between RISC and CISC is the number of instructions and its complexity.

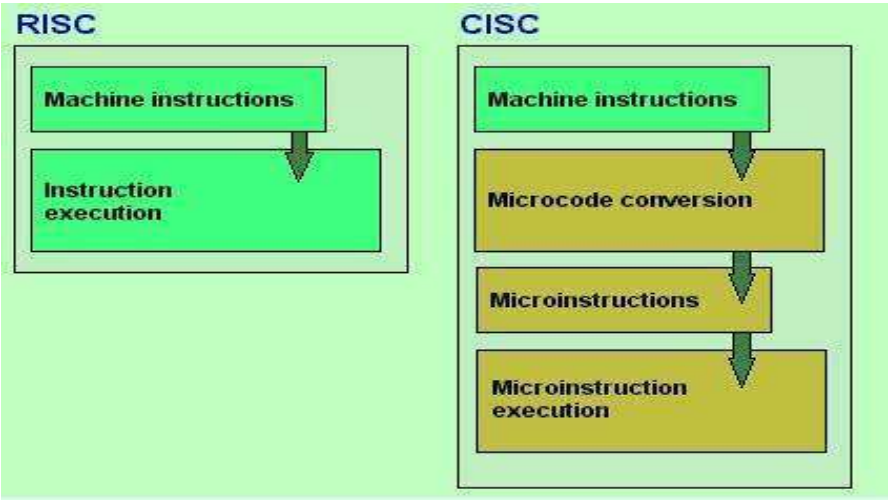
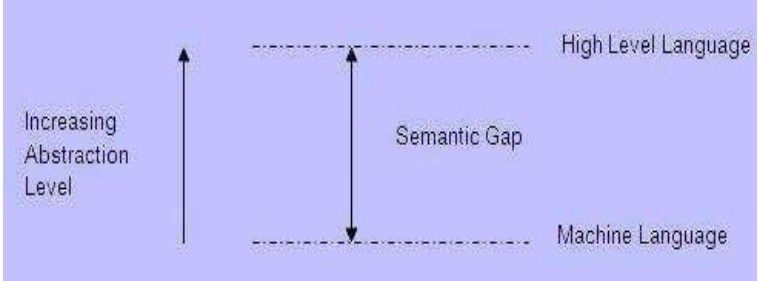


Figure: RISC & CISC COMPARISON

SEMANTIC GAP

Both RISC and CISC architectures have been developed as an attempt to cover the semantic gap.



With an objective of improving efficiency of software development, several powerful programming languages have come up, viz., Ada, C, C++, Java, etc. They provide a high level of abstraction, conciseness and power. By this evolution the semantic gap grows. To enable efficient compilation of high level language programs, CISC and RISC designs are the two options.

The features of RISC include the following

- The demand of decoding is less
- Uniform instruction set
- Few data types in hardware
- General purpose register Identical
- Simple addressing nodes

Advantages of RISC architecture:

- RISC(Reduced instruction set computing)architecture has a set of instructions, so high-level language compilers can produce more efficient code
- It allows freedom of using the space on microprocessors because of its simplicity.
- Many RISC processors use the registers for passing arguments and holding the local variables.
- RISC functions use only a few parameters, and the RISC processors cannot use the call instructions, and therefore, use a fixed length instruction which is easy to pipeline.
- The speed of the operation can be maximized and the execution time can be minimized. Very less number of instructional formats, a few numbers of instructions and a few addressing modes are needed.

The Disadvantages of RISC architecture:

- Mostly, the performance of the RISC processors depends on the programmer or compiler as the knowledge of the compiler plays a vital role while changing the CISC code to a RISC code

- While rearranging the CISC code to a RISC code, termed as a code expansion, will increase the size. And, the quality of this code expansion will again depend on the compiler, and also on the machine's instruction set.
- The first level cache of the RISC processors is also a disadvantage of the RISC, in which these processors have large memory caches on the chip itself. For feeding the instructions, they require very fast memory systems.

CISC Architecture

- The CISC approach attempts to minimize the number of instructions per program, sacrificing the number of cycles per instruction. Computers based on the CISC architecture are designed to decrease the memory cost. Because, the large programs need more storage, thus increasing the memory cost and large memory becomes more expensive. To solve these problems, the number of instructions per program can be reduced by embedding the number of operations in a single instruction, thereby making the instructions more complex.

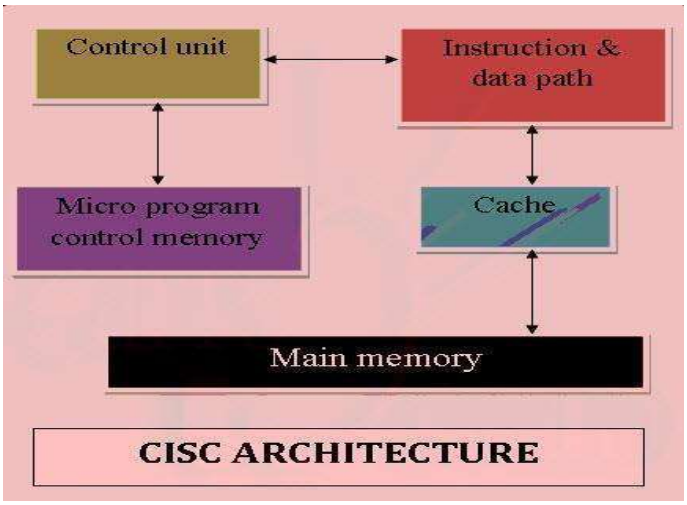


Figure: CISC Architecture

MUL loads two values from the memory into separate registers in CISC. CISC uses minimum possible instructions by implementing hardware and executes operations. Instruction Set Architecture is a medium to permit communication between the programmer and the hardware. Data execution part, copying of data, deleting or editing is the user commands used in the microprocessor and with this microprocessor the Instruction set architecture is operated. The main keywords used in the above Instruction Set Architecture are as below

Instruction Set: Group of instructions given to execute the program and they direct the computer by manipulating the data. Instructions are in the form – Opcode (operational code) and

Operand. Where, opcode is the instruction applied to load and store data, etc. The operand is a memory register where instruction applied.

Addressing Modes: Addressing modes are the manner in the data is accessed. Depending upon the type of instruction applied, addressing modes are of various types such as direct mode where straight data is accessed or indirect mode where the location of the data is accessed. Processors having identical ISA may be very different in organization. Processors with identical ISA and nearly identical organization is still not nearly identical.

CPU performance is given by the fundamental law

$$CPU\ Time = \frac{Seconds}{Program} = \frac{Instructions}{Program} X \frac{Cycles}{Instructions} X \frac{Seconds}{Cycle}$$

Thus, CPU performance is dependent upon Instruction Count, CPI (Cycles per instruction) and Clock cycle time. And all three are affected by the instruction set architecture.

Instruction Count of the CPU

	Instruction Count	CPI	Clock
Program	X		
Compiler	X	X	
Instruction Set Architecture	X	X	X
Microarchitecture		X	X
Physical Design			X

Examples of CISC PROCESSORS

IBM 370/168 – It was introduced in the year 1970. CISC design is a 32 bit processor and four 64-bit floating point registers.
VAX 11/780 – CISC design is a 32-bit processor and it supports many numbers of addressing modes and machine instructions which is from Digital Equipment Corporation.
Intel 80486 – It was launched in the year 1989 and it is a CISC processor, which has instructions varying lengths from 1 to 11 and it will have 235 instructions.

CHARACTERISTICS OF CISC ARCHITECTURE

Instruction-decoding logic will be Complex. One instruction is required to support multiple addressing modes. Less chip space is enough for general purpose registers for the instructions that are 0operated directly on memory. Various CISC designs are set up two special registers for the stack pointer, handling interrupts, etc. MUL is referred to as a "complex instruction" and requires the programmer for storing functions. CISC designs involve very complex architectures, including a large number of instructions and addressing modes, whereas RISC designs involve simplified instruction set and adapt it to the real requirements of user programs.

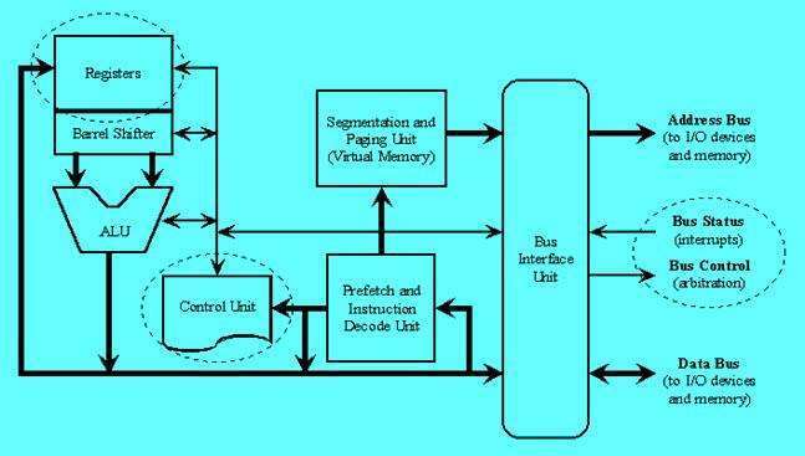


Figure: CISC and RISC Design

Advantages of CISC architecture

- Microprogramming is easy assembly language to implement, and less expensive than hard wiring a control unit.
- The ease of micro coding new instructions allowed designers to make CISC machines upwardly compatible:
- As each instruction became more accomplished, fewer instructions could be used to implement a given task.

Disadvantages of CISC architecture

- The performance of the machine slows down due to the amount of clock time taken by different instructions will be dissimilar
- Only 20% of the existing instructions is used in a typical programming event, even though there are various specialized instructions in reality which are not even used frequently.
- The conditional codes are set by the CISC instructions as a side effect of each instruction which takes time for this setting – and, as the subsequent instruction changes the

condition code bits – so, the compiler has to examine the condition code bits before this happens.

Memory Unit

RISC has no memory unit and uses a separate hardware to implement instructions. CISC has a memory unit to implement complex instructions

Program

RISC has a hard-wired unit of programming. CISC has a microprogramming unit

Design

RISC is a complex compiler design. CISC is an easy compiler design

Calculations

RISC calculations are faster and more precise. CISC calculations are slow and precise

Decoding

RISC decoding of instructions is simple. CISC decoding of instructions is complex

Time

Execution time is very less in RISC. Execution time is very high in CISC.

External memory

RISC does not require external memory for calculations. CISC requires external memory for calculations.

Pipelining

RISC Pipelining does function correctly. CISC Pipelining does not function correctly.

Stalling

RISC stalling is mostly reduced in processors. CISC processors often stall.

Code Expansion

Code expansion can be a problem in RISC whereas, in CISC, Code expansion is not a problem.

Disc space

Space is saved in RISC whereas in CISC space is wasted. The best examples of CISC instruction set architecture include VAX, PDP-11, Motorola 68k,And your desktop PCs on Intel's x86

architecture, whereas the best examples of RISC architecture include DEC Alpha, ARC, AMD 29k, Atmel AVR, Intel i860, Blackfin, i960, Motorola 88000, MIPS, PA-RISC, Power, SPARC, SuperH, and ARM too.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING
DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

UNIT – II – Computer Arithmetic – SCSA1402

II. Computer Arithmetic

Introduction:

Data is manipulated by using the arithmetic instructions in digital computers.Data is manipulated to produce results necessary to give solution for the computation problems. The Addition, subtraction, multiplication and division are the four basic arithmetic operations. If we want then we can derive other operations by using these four operations. To execute arithmetic operations there is a separate section called arithmetic processing unit in central processing unit. The arithmetic instructions are performed generally on binary or decimal data. Fixed-point numbers are used to represent integers or fractions. We can have signed or unsigned negative numbers. Fixed-point addition is the simplest arithmetic operation. If we want to solve a problem then we use a sequence of well-defined steps. These steps are collectively called algorithm.

Data types

- Fixed-point binary
  - ✓ Signed-magnitude representation ‹
  - ✓ Signed-2's complement representation „
- Floating-point binary „
- Binary-coded decimal (BCD)

Addition and Subtraction

There are three ways of representing negative fixed-point binary numbers: Signed-magnitude, signed-l's complement, or signed-2's complement. Most computers use the signed-2's complement representation when performing arithmetic operations with integers. For floating-point operations, most computers use the signed-magnitude representation for the mantissa.

Addition and Subtraction with Signed-Magnitude Data

Where the signed numbers are added or subtracted, we find that there are eight different conditions to consider, depending on the sign of the numbers and the operationperformed. These conditions are listed in the first column of Table 2.1. The other columns in the table show the actual operation to be performed with the magnitude of the numbers. The last column is needed to present a negative zero. In other words, when two equal numbers are subtracted, the result should be +0 not -0. The algorithms for addition and subtraction are derived from the table and can be stated as follows

Table 2.1: Addition and Subtraction of Signed-Magnitude Numbers

Operation	Add Magnitudes	Subtract Magnitudes		
		When A > B	When A < B	When A = B
(+ A) + (+ B)	+(A + B)			
(+ A) + (– B)		+ (A – B)	– (B – A)	+ (A – B)
(– A) + (+ B)		– (A – B)	+ (B – A)	+ (A – B)
(– A) + (– B)	– (A + B)			
(+ A) – (+ B)		+ (A – B)	– (B – A)	+ (A – B)
(+ A) – (– B)	+(A + B)			
(– A) – (+ B)	– (A + B)			
(– A) – (– B)		– (A – B)	+ (B – A)	+ (A – B)

When the signs of A and B are same, add the two magnitudes and attach the sign of result is that of A. When the signs of A and B are not same, compare the magnitudes and subtract the smaller number from the larger. Choose the sign of the result to be the same as A, if A > B or the complement of the sign of A if A < B. If thetwo magnitudes are equal, subtract B from A and make the sign of the result will be positive.

HARDWARE IMPLEMENTATION:-

First, a parallel-adder is needed to perform the micro operation A + B . Second, a comparator circuit is needed to establish if A > B, A = B, or A < B. Third, two parallel- subtractor circuits are needed to perform the micro operations A - B and B - A. The sign relationship can be determined from an exclusive OR gate with A, and B, as inputs.

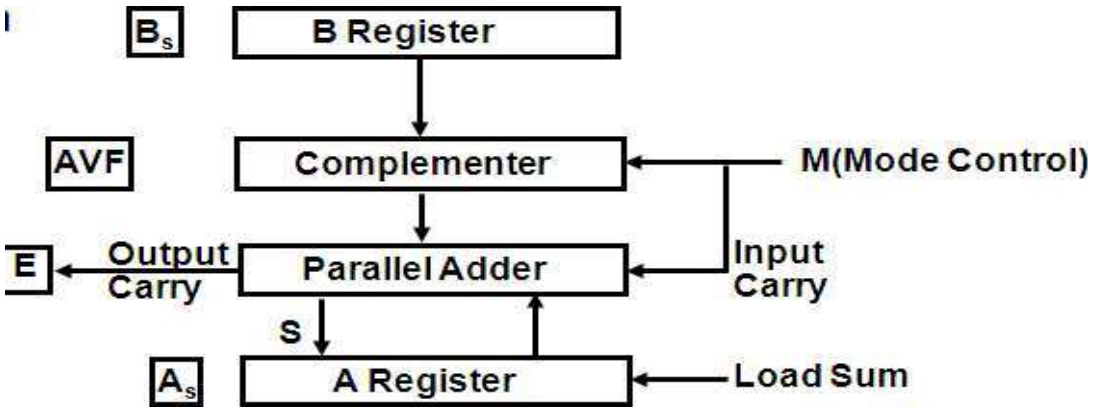


Fig. 2.1 Hardware for signed magnitude addition and subtraction

block diagram of the hardware for implementing the addition and subtraction operations. It consists of registers A and B and sign flip-flops A, and B, . Subtraction is done by adding A to the 2' s complement of B. The output carry is transferred to flip-flop E, where it can be checked to determine the relative magnitudes of the two numbers. The add-overflow flip-flop AVF holds the overflow bit when A and B are added. The A register provides other microoperations that may be needed when we specify thesequence of steps in the algorithm.

Description

A_S Sign of A , B_S Sign of B ,A_S & A Accumulator , AVF Overflow bit for A + BE

Output carry for parallel adder

Data representation Signed magnitude – consists of the magnitude and negative sign(sign bit in binary, '0' for positive and '1' for negative)

– E.g. +14 = 0 0001110, -14= 1 0001110

Signed 1's complement – leaving out the sign bit, convert all 1's to 0's and 0's to 1's inthe signed magnitude form of the data

– E.g. -14 = 1 1110001

Signed 2's complement – Add 1 to signed 1's complement representation of the data –E.g. -14 = 1 1110010

Hardware algorithm:

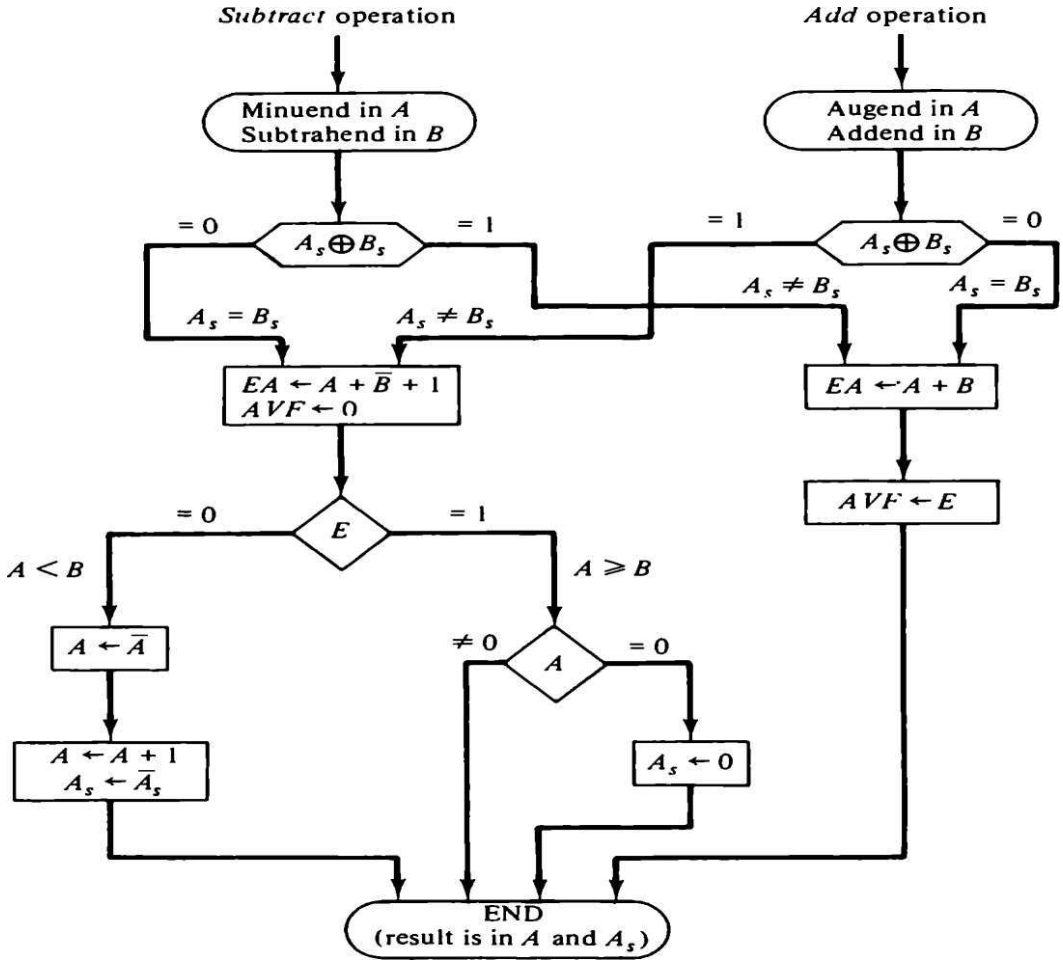


Fig.2.2 Flowchart for add and subtract operations

Signed magnitude addition and subtraction

- For an add operation, identical signs dictate that the magnitudes be added,different signs require that the magnitudes be subtracted
- For subtraction operation, different signs dictate that magnitudes be added, identical signs require that magnitudes be subtracted

AVF – Add-overflow flip-flop holds the overflow bit when A and B are added

Addition of A and B is done through parallel adder

2's complement addition and subtraction:

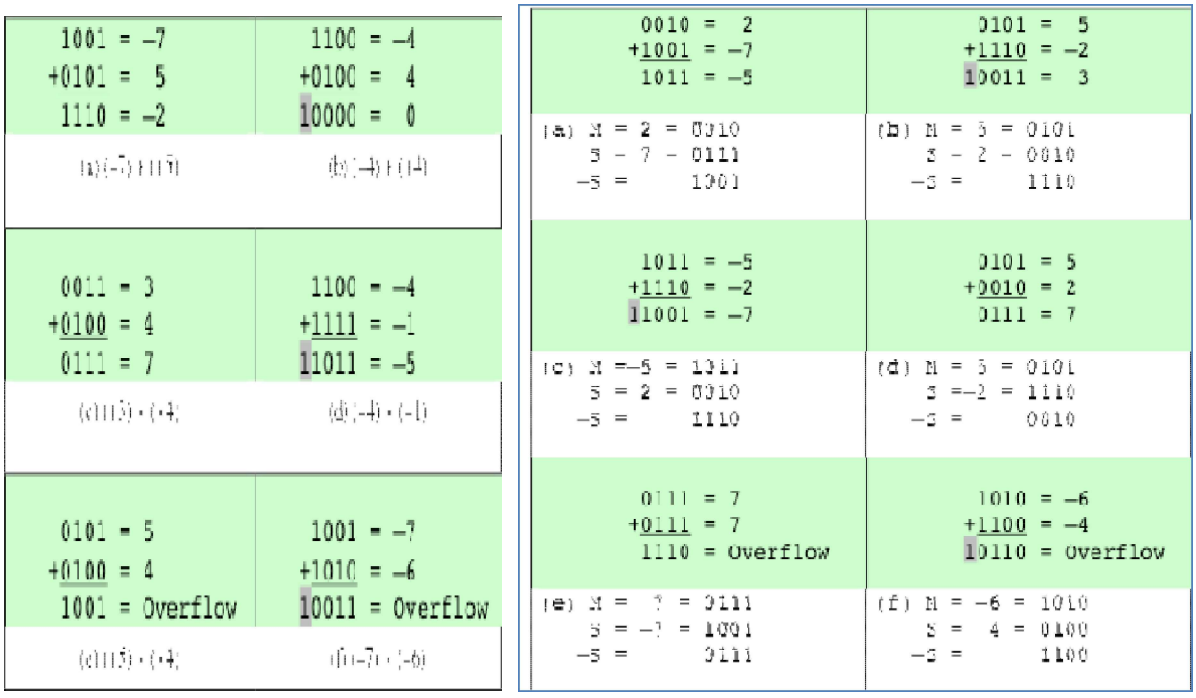


Fig. 2.3 Addition and subtraction of numbers in 2's complement representation

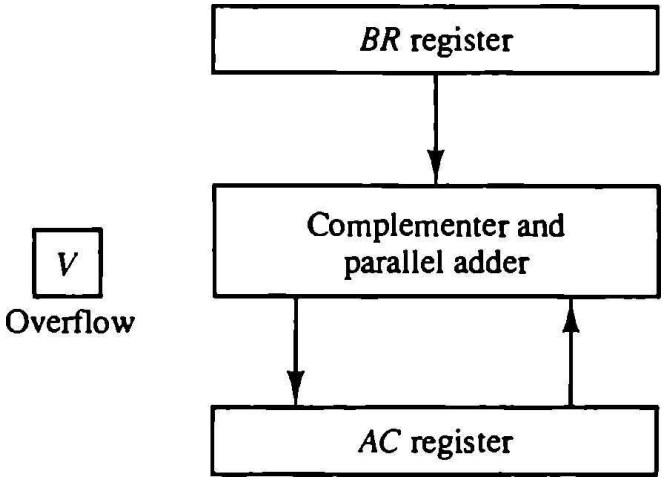


Fig.2.4 Block diagram of hardware for signed 2's complement addition and subtraction

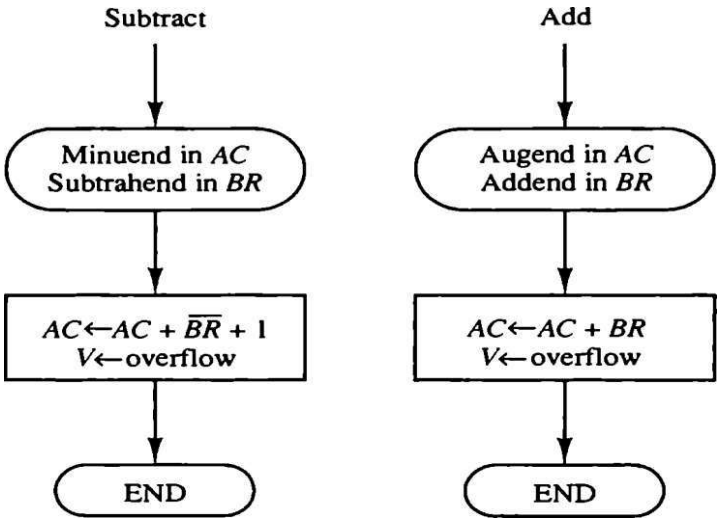


Fig.2.5 Algorithm for signed 2's complement addition and subtraction

Multiplication Algorithm:

A binary example:

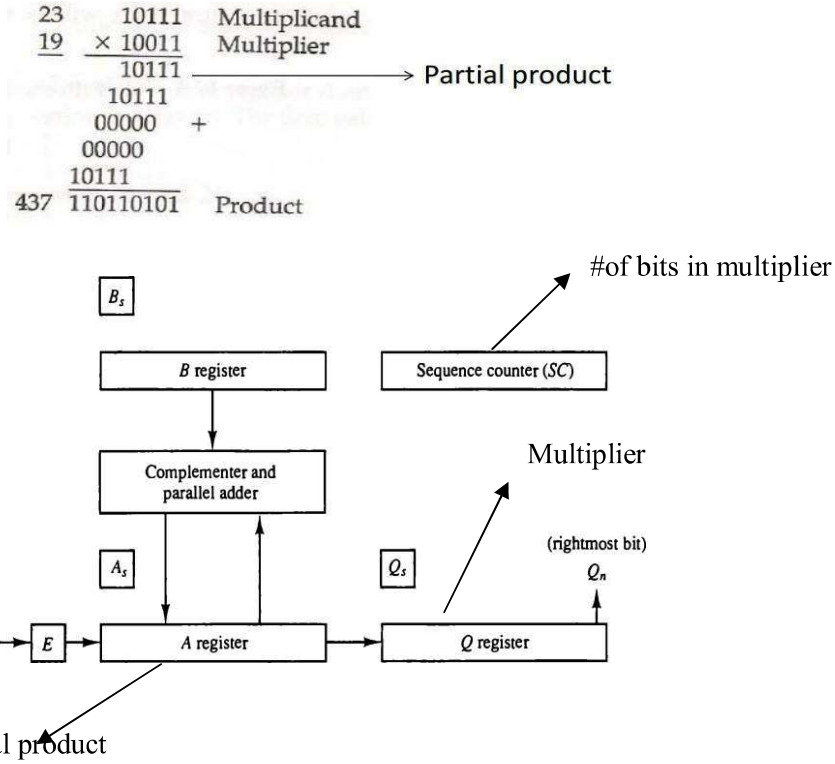
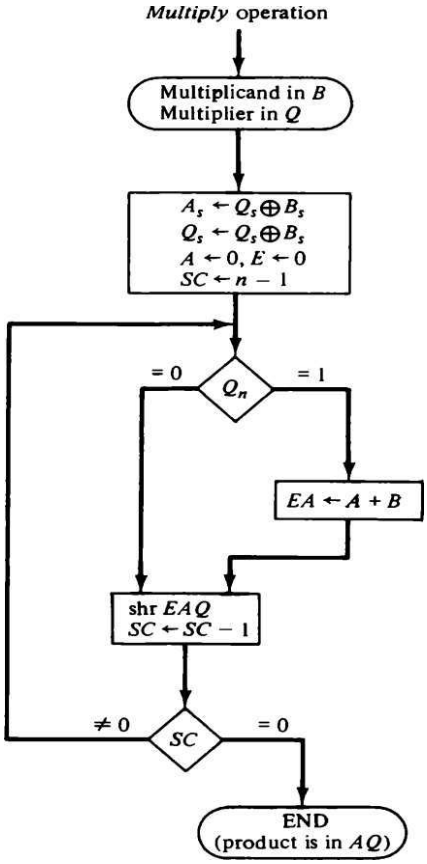


Fig.2.6 Block diagram of hardware for multiply operation



	B=11011 Q=00111
4	Q4=1,A=0,Qs=1 EA=A+B=1011 EAQ= 0 1011 0111 Shr EAQ= 0 0101 1011
3	Q3=1 EA = 1 0000 EAQ 1 0000 1011 Shr EAQ 0 1000 0101
2	Q2=1 EA= 1 0011 EAQ 1 0011 01 01 shr EAQ 0 1001 101 0
1	Q1=0 Shr EAQ 0 0100 1101=77
0	

Fig.2.7 Flow chart for multiply operation Table

2.2 Numerical example for Binary multiplier

Multiplicand B = 10111	E	A	Q	SC
Multiplier in Q	0	00000	10011	101
Q_n = 1; add B		10111		
First partial product	0	10111		
Shift right EAQ	0	01011	11001	100
Q_n = 1; add B		10111		
Second partial product	1	00010		
Shift right EAQ	0	10001	01100	011
Q_n = 0; shift right EAQ	0	01000	10110	010
Q_n = 0; shift right EAQ	0	00100	01011	001
Q_n = 1; add B		10111		
Fifth partial product	0	11011		
Shift right EAQ	0	01101	10101	000
Final product in AQ = 0110110101				

Booth multiplication algorithm :

A= 00011 B= 00111 => A*B= A*(7) =A* (8-1) =A*8-A*1

Booth algorithm requires examination of the multiplier bits and shifting of the partial product

Q_n - LSB of multiplier Extra flip flop Q_{n+1} is appended to the multiplier bits to facilitate double bit inspection of the multiplier.

Compare bits of Q_n and Q_{n+1}

Rules are:

1. The multiplicand is subtracted from the partial product upon encountering the firstleast significant 1 in a string of 1's in the multiplier
2. The multiplicand is added to the partial product upon encountering the first 0(provided that there was a previous 1) in a string of 0's in the multiplier
3. The partial product does not change when the multiplier bit is identical to the previous multiplier bit

   In 2's complement representation, we can use Booth algorithm without change

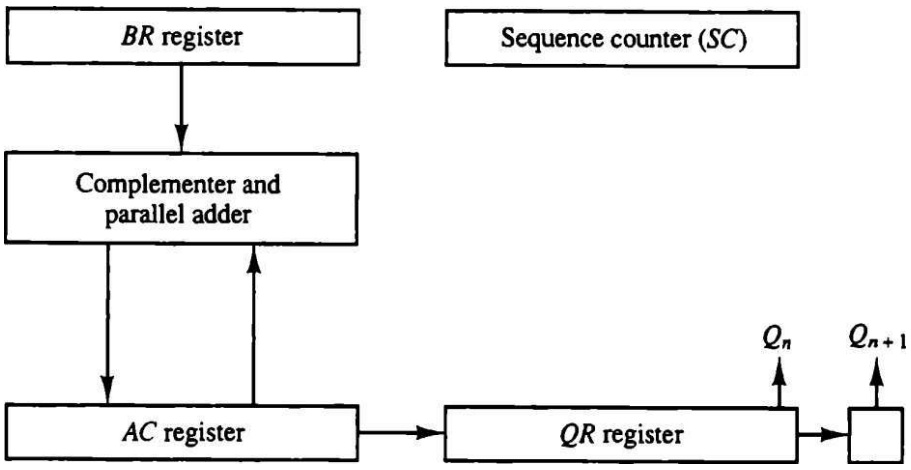


Fig.2.8 Block diagram of hardware for Booth algorithm

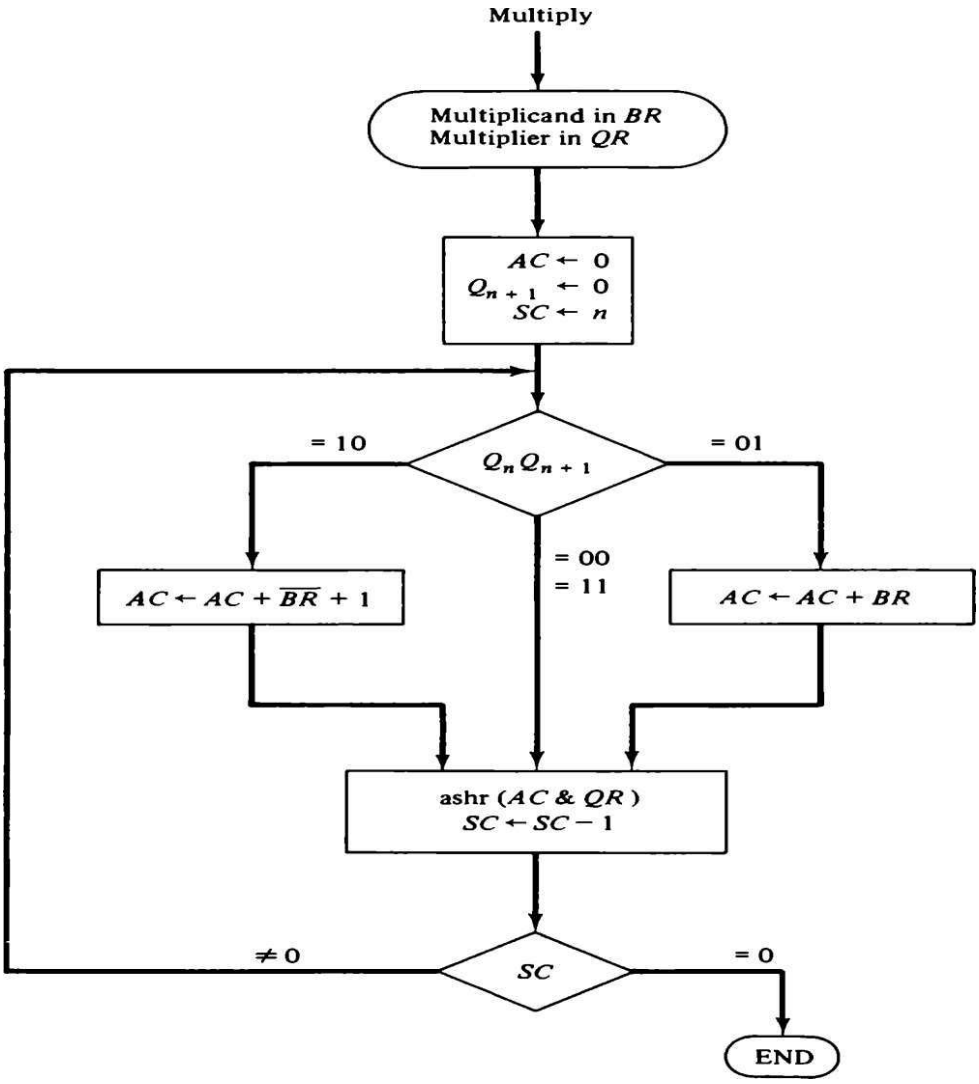


Fig. 2.9 Booth algorithm for multiplication of signed 2's complements numbersTable

2.3 Example of multiplication with Booth Algorithm

$Q_n Q_{n+1}$	$BR = 10111$ $\overline{BR} + 1 = 01001$	AC	QR	Q_{n+1}	SC
	Initial	00000	10011	0	101
1 0	Subtract BR	01001			
		01001			
	ashr	00100	11001	1	100
1 1	ashr	00010	01100	1	011
0 1	Add BR	10111			
		11001			
	ashr	11100	10110	0	010
0 0	ashr	11110	01011	0	001
1 0	Subtract BR	01001			
		00111			
	ashr	00011	10101	1	000

Array multiplier: Fast approach

To check the bits of the multiplier one at a time and forming partial products is a sequential operation requiring a sequence of add and shift micro-operations. The multiplication of two binary numbers can be done with one micro-operation by using combinational circuit that forms the product bits all at once. This is a fast way since all it takes is the time for the signals to propagate through the gates that form the multiplication array. However, an array multiplier requires a large number of gates, and so it is not an economical unit for the development of ICs

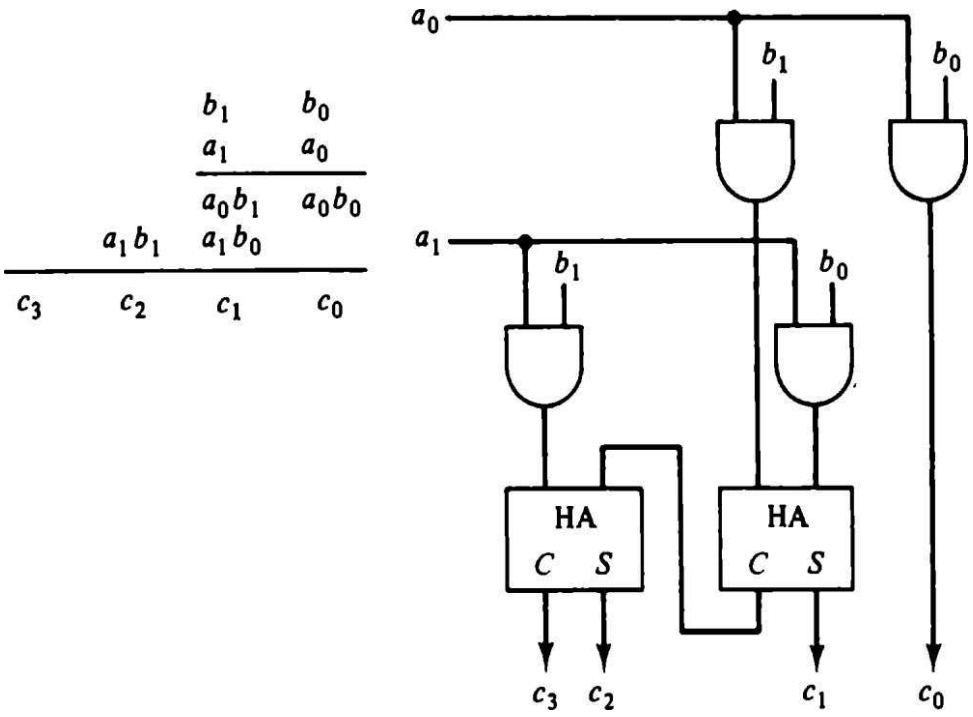


Fig. 2.10 : 2 bit by 2 bit array multiplier

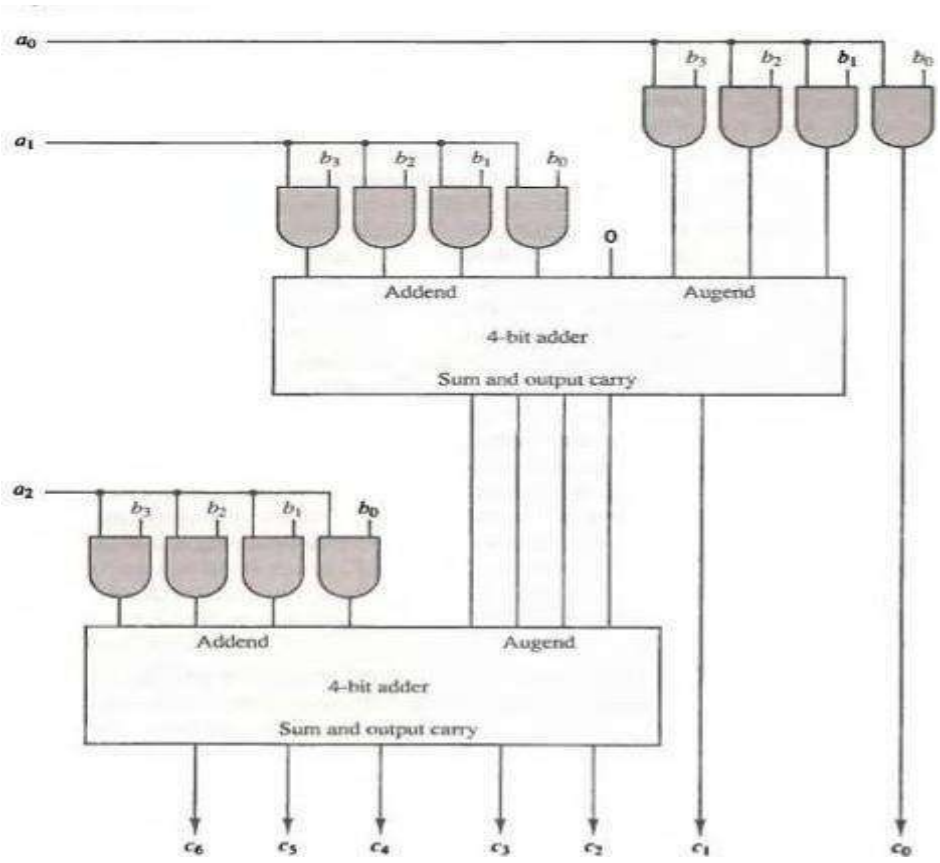


Fig. 2.11: 4 bit by 3bit array multiplier

Division algorithm:

Binary division – simpler because the quotient digits are either 0 or 1 and there is no need to estimate how many times the dividend or partial remainder fits into the divisor

Division operation may result in a quotient with an overflow.

Divide overflow flip flop (DVF) is used to detect overflow

Divisor – B register, Dividend – A and Q register

If the signs of divisor and dividend are alike, the sign of the quotient is plus. Otherwiseit is minus.

Best way to avoid divide overflow is to use floating point data.

Example for binary division :

Divisor:	11010	Quotient = Q
B = 10001	)0111000000	Dividend = A
	01110	5 bits of $A < B$, quotient has 5 bits
	011100	6 bits of $A \geq B$
	-10001	Shift right B and subtract; enter 1 in Q
	-010110	7 bits of remainder $\geq B$
	--10001	Shift right B and subtract; enter 1 in Q
	--001010	Remainder $< B$; enter 0 in Q; shift right B
	---010100	Remainder $\geq B$
	----10001	Shift right B and subtract; enter 1 in Q
	----000110	Remainder $< B$; enter 0 in Q
	-----00110	Final remainder

Divisor B = 10001, $\overline{B} + 1$ = 01111

	E	A	Q	SC
Dividend:		01110	00000	5
shl EAQ	0	11100	00000	
add $\overline{B} + 1$		01111		
E = 1	1	01011		
Set Q_n = 1	1	01011	00001	4
shl EAQ	0	10110	00010	
Add $\overline{B} + 1$		01111		
E = 1	1	00101		
Set Q_n = 1	1	00101	00011	3
shl EAQ	0	01010	00110	
Add $\overline{B} + 1$		01111		
E = 0; leave Q_n = 0	0	11001	00110	
Add B		10001		2
Restore remainder	1	01010		
shl EAQ	0	10100	01100	
Add $\overline{B} + 1$		01111		
E = 1	1	00011		
Set Q_n = 1	1	00011	01101	1
shl EAQ	0	00110	11010	
Add $\overline{B} + 1$		01111		
E = 0; leave Q_n = 0	0	10101	11010	
Add B		10001		
Restore remainder	1	00110	11010	0
Neglect E				
Remainder in A:		00110		
Quotient in Q:			11010	

Fig.2.12 Example of binary division with digital hardware

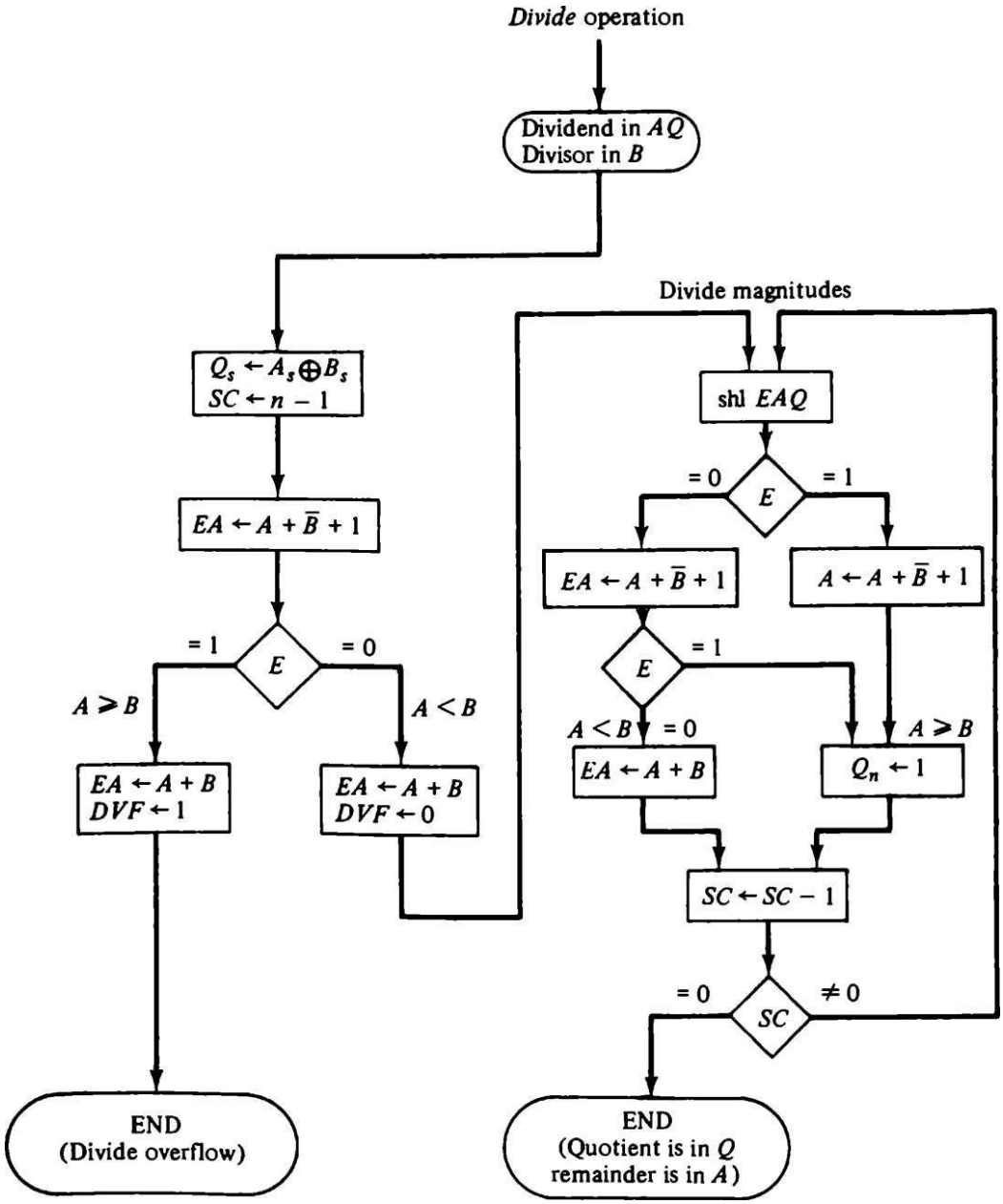


Fig. 2.13 Flow chart for divide operation

Floating Point Arithmetic Operations:

- Numbers too large for standard integer representations or that have fractional components are usually represented in scientific notation, a form used commonly by scientists and engineers.
- Examples: 4.25×10^1
- Addition and subtraction are more complex than multiplication and division

• Need to align mantissas

• Algorithm: — Check for zeros — Align significant (adjusting exponents) — Add or subtract significant — Normalize result

$$F = m \times r^e$$

where m: Mantissa, r: Radix, e: Exponent

It is necessary to make two exponents be equal before the mantissas can be added. We can either shift the first number three positions to the left, or shift the second number three positions to the right. When we store the mantissas in registers, shifting to the left causes a loss of most significant digits. Shifting to the right causes a loss of least significant digits. The second method is preferable because it only reduces the accuracy, while the first method may cause an error. The usual alignment procedure is to shift the mantissa that has the smaller exponent to the right by a number of places equal to the difference between the exponents. Now, the mantissas can be added.

$$\begin{array}{r} .5372400 \times 10^2 \\ +\ .0001580 \times 10^2 \\ \hline .5373980 \times 10^2 \end{array}$$

When two normalized mantissas are added, the sum may contain an overflow digit. An overflow can be corrected easily by shifting the sum once to the right and incrementing the exponent. When two numbers are subtracted, the result may contain most significant zeros as shown in the following example:

$$\begin{array}{r} .56780 \times 10^5 \\ -\ .56430 \times 10^5 \\ \hline .00350 \times 10^5 \end{array}$$

An underflow occurs if a floating-point number that has a 0 in the most significant position of the mantissa. To normalize a number that contains an underflow, we shift the mantissa to the left and decrement the exponent until a nonzero digit appears in the first position. Here, it is necessary to shift left twice to obtain $.35000 \times 10^3$. In most computers a normalization procedure is performed after each operation to ensure that all results are in a normalized form.

Floating-point multiplication and division need not do an alignment of the mantissas. Multiplying the two mantissas and adding the exponents can form the product. Dividing the mantissas and subtracting the exponents perform division.

Register Configuration:

Three registers are there, BR, AC, and QR. Each register is subdivided into two parts. The mantissa part has the same uppercase letter symbols as in fixed-point representation. The exponent part may use corresponding lower-case letter symbol.

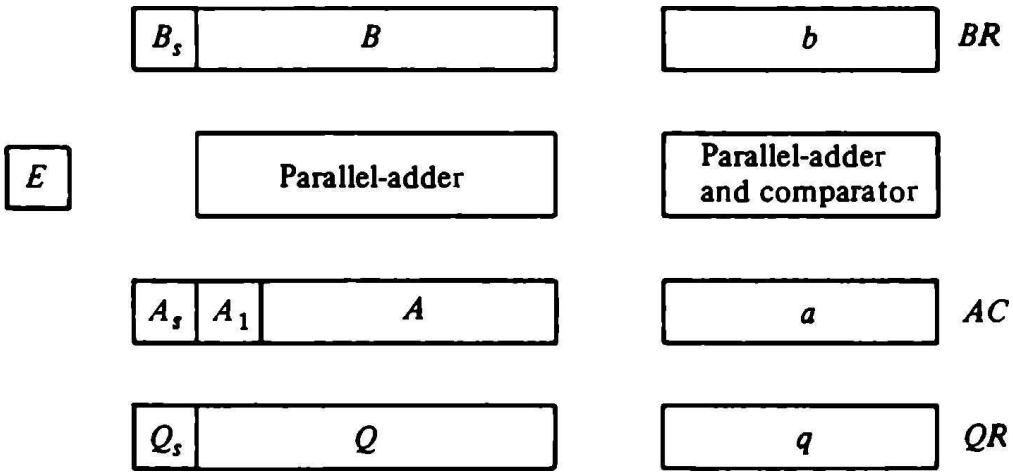


Fig. 2.14 Registers for floating point arithmetic operations

Assuming that each floating-point number has a mantissa in signed-magnitude representation and a biased exponent. Thus the AC has a mantissa whose sign is in A_s, and a magnitude that is in A. The diagram shows the most significant bit of A, labeled by A1. The bit in his position must be a 1 to normalize the number. Note that the symbol AC represents the entire register, that is, the concatenation of A_s, A and a. In the similar way, register BR is subdivided into B_s, B, and b and QR into Q_s, Q and q. A parallel-adder adds the two mantissas and loads the sum into A and the carry into E. **Addition and Subtraction of Floating Point Numbers**

During addition or subtraction, the two floating-point operands are kept in AC andBR. The sum or difference is formed in the AC.

The algorithm can be divided into four consecutive parts:

1. Check for zeros.

2. Align the mantissas.

3. Add or subtract the mantissas

4. Normalize the result

A floating-point number cannot be normalized, if it is 0. If this number is used for computation, the result may also be zero. Instead of checking for zeros during the normalization process we check for zeros at the beginning and terminate the process if necessary. The alignment of the mantissas must be carried out prior to their operation. After the mantissas are added or subtracted, the result may be un-normalized. The normalization procedure ensures that the result is normalized before it is transferred to memory.

For adding or subtracting two floating-point binary numbers, if BR is equal to zero, the operation is stopped, with the value in the AC being the result. If AC = 0, we transfer the content of BR into AC and also complement its sign we have to subtract the numbers. If neither number is equal it to zero, we proceed to align the mantissas. The magnitude comparator attached to exponents a and b gives three outputs, which show their relative magnitudes.

If the two exponents are equal, we go to perform the arithmetic operation. If the exponents are not equal, the mantissa having the smaller exponent is shifted to the right and its exponent incremented. This process is repeated until two exponents are equal.

The addition and subtraction of the two mantissas is similar to the fixed-point addition and subtraction algorithm presented in the following Fig

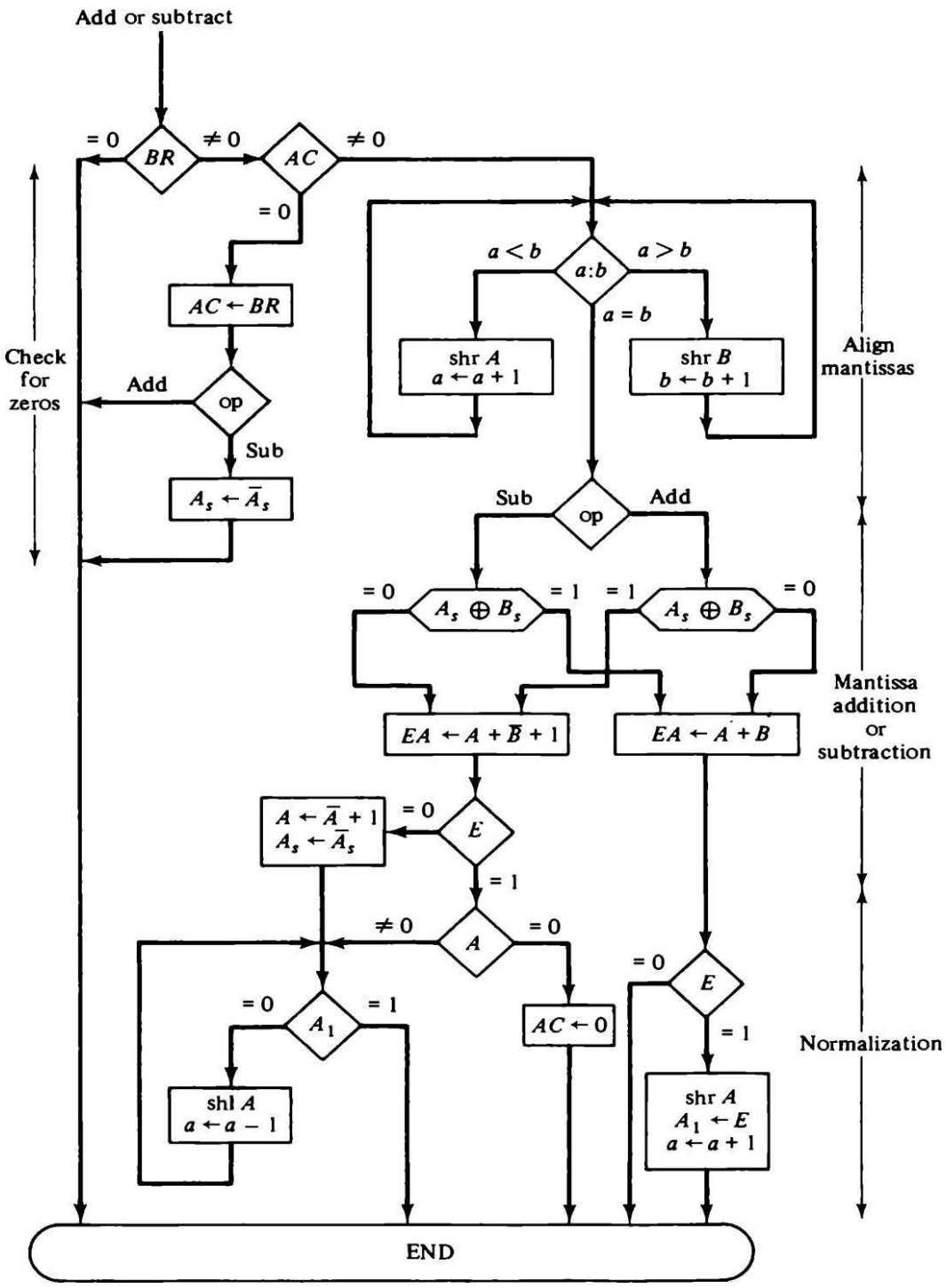


Fig. 2.15 Addition and subtraction of floating point numbers

Multiplication:

The algorithm can be divided into four consecutive parts:

1. Check for zeros.
2. Add the exponents.
3. Multiply the mantissas
4. Normalize the result

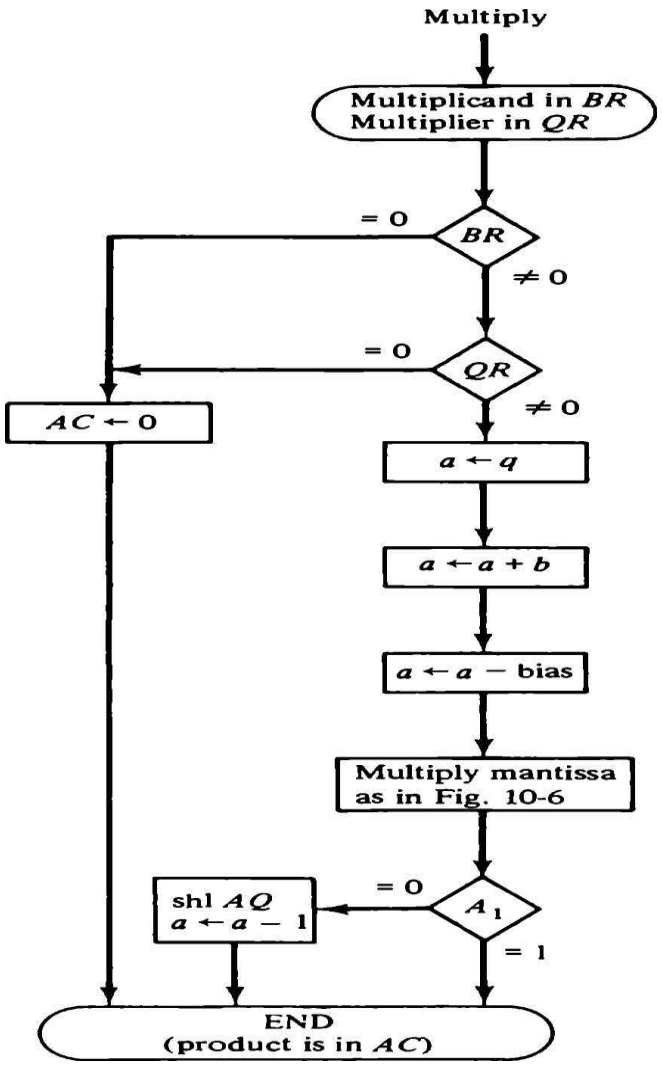


Fig. 2.16 Multiplication of floating point numbers

Division:

The algorithm can be subdivided into five consecutive parts:

1. Check for zeros.
2. Initialize the register and evaluate the sign
3. Align the dividend
4. Subtract the exponents.
5. Divide the mantissa

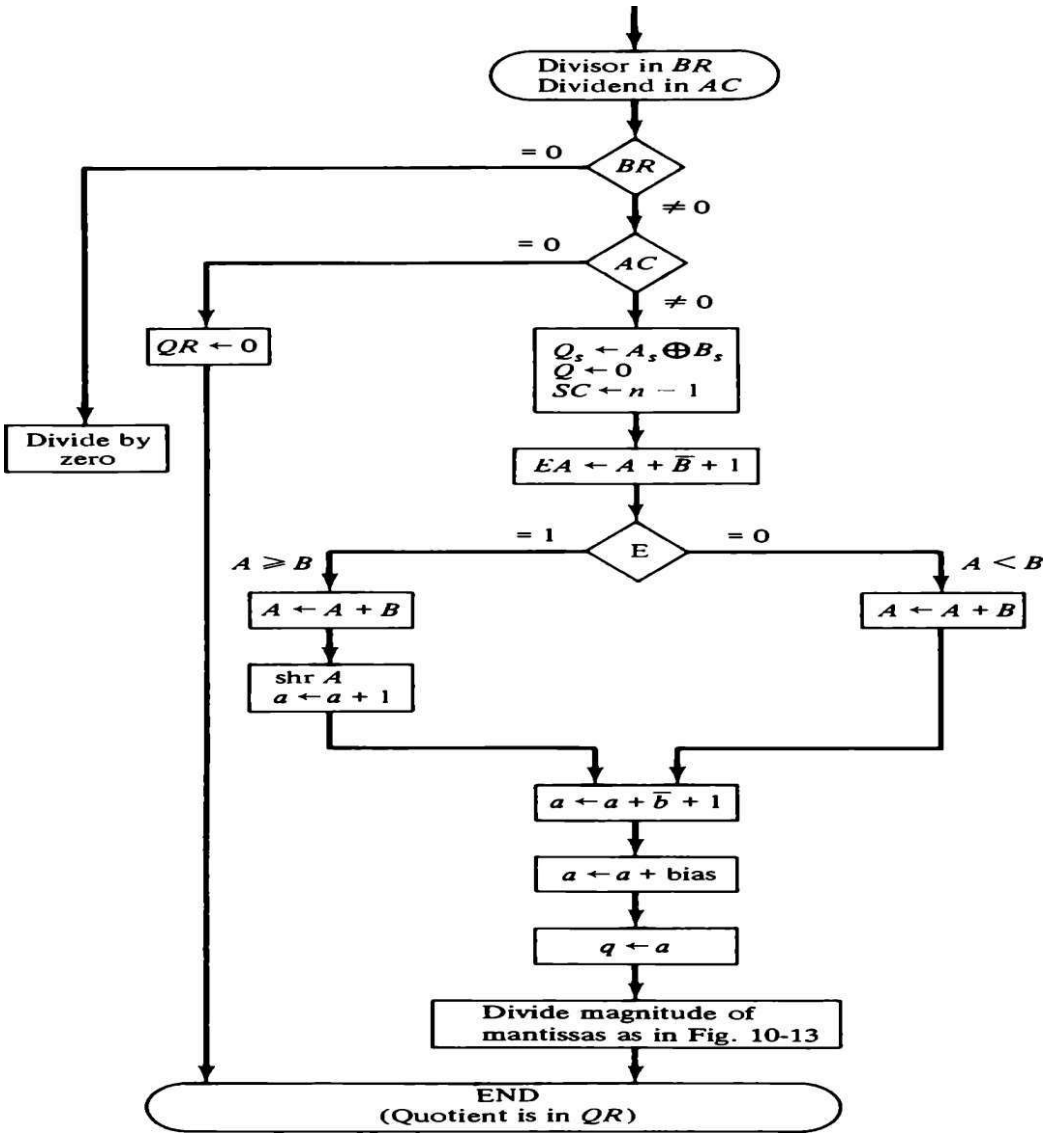


Fig. 2.17 Division of floating point numbers

Microprogrammed Control

Introduction:

Microprogram

- Program stored in memory that generates all the control signals required toexecute the instruction set correctly
- Consists of microinstructions

Microinstruction

- Contains a control word and a sequencing word

Control Word - All the control information required for one clock cycle Sequencing Word - Information needed to decide the next microinstruction adress

Control Memory(Control Storage: CS)

- Storage in the microprogrammed control unit to store the microprogram

Writeable Control Memory(Writeable Control Storage:WCS)

- CS whose contents can be modified

-> Allows the microprogram can be changed

-> Instruction set can be changed or modified

Dynamic Microprogramming

- Computer system whose control unit is implemented with a microprogram in WCS
- Microprogram can be changed by a systems programmer or a user

a. Control Memory

◆ Control Unit

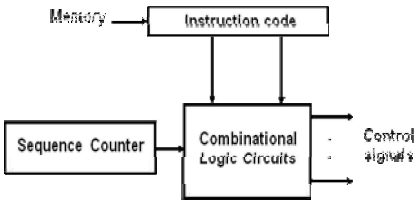
- Initiate sequences of microoperations

» Control signal *(that specify microoperations)* in a bus-organized system

o groups of bits that select the paths in multiplexers, decoders, and arithmetic logic units

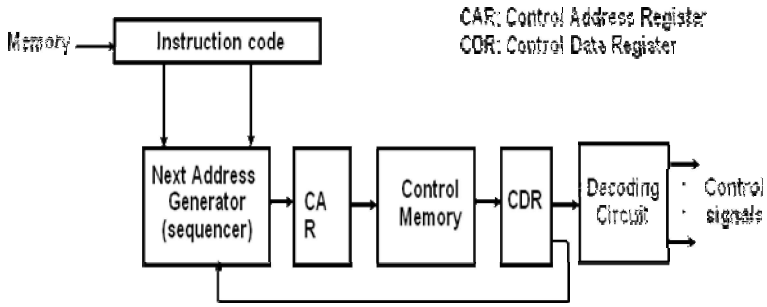
- Two major types of Control Unit

» Hardwired Control :



- The control logic is implemented with gates, F/Fs, decoders, and other digitalcircuits
- \+ Fast operation, - Wiring change(if the design has to be modified)

» Microprogrammed Control



- The control information is stored in a control memory, and the control memoryis programmed to initiate the required sequence of microoperations
- \+ Any required change can be done by updating the microprogram in controlmemory,
  - Slow operation

◆ Control Word

- The control variables at any given time can be represented by a string of 1'sand 0's.

◆ Microprogrammed Control Unit

- A control unit whose binary control variables are stored in memory (*control memory*).

◆ Microinstruction : *Control Word in Control Memory*

- The microinstruction specifies one or more microoperations

- Microprogram
  - A sequence of microinstruction
  - Dynamic microprogramming : *Control Memory* = RAM
- RAM can be used for writing (*to change a writable control memory*)
- Microprogram is loaded initially from an auxiliary memory such as a magnetic disk
  - Static microprogramming : *Control Memory* = ROM
- Control words in ROM are made permanent duringthe hardware production.
- Microprogrammed control Organization :

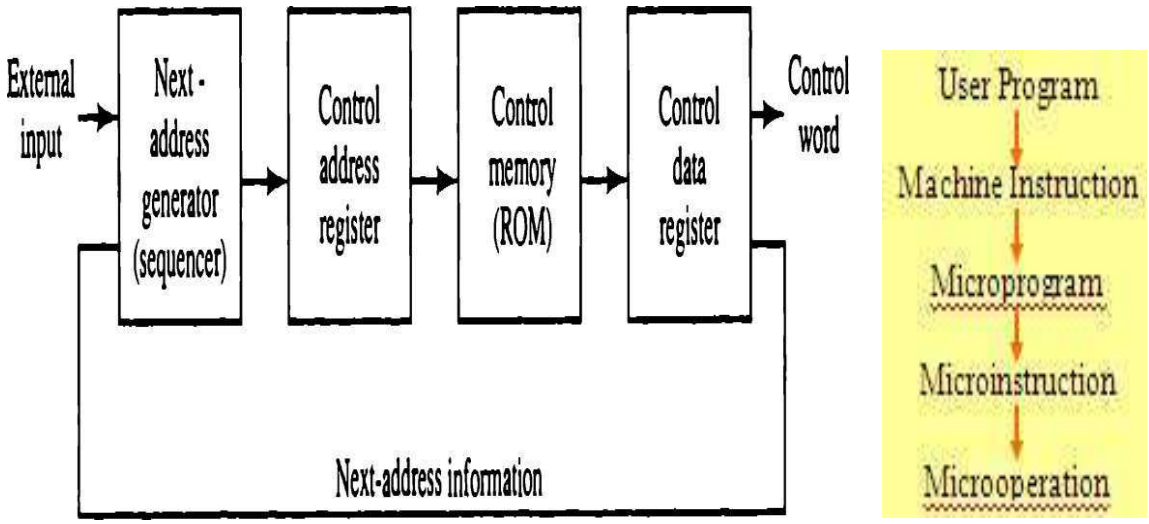


Fig. 2.18 Microprogrammed control operation

1) Control Memory
   - A memory is part of a control unit : *Microprogram*이
   - Computer Memory (*employs a microprogrammed control unit*)
   - Main Memory : for storing user program (*Machine instruction/data*)
   - Control Memory : for storing microprogram (*Microinstruction*)

2) Control Address Register
   - Specify the address of the microinstruction

3) Sequencer (= *Next Address Generator*)
   - Determine the address sequence that is read from control memory

- Next address of the next microinstruction can be specified several way depending on the sequencer input

4) Control Data Register (= *Pipeline Register*)

- Hold the microinstruction read from control memory
- Allows the execution of the microoperations specified by the control word ***simultaneously*** with the generation of the next microinstruction

◆ RISC Architecture Concept

RISC(Reduced Instruction Set Computer) system use hardwired control rather than microprogrammed control :

b. Address Sequencing

◆ Address Sequencing = Sequencer : Next Address Generator

- Selection of address for control memory

◆ Routine → *Subroutine: program used by other ROUTINES*

- Microinstruction are stored in control memory in groups

◆ Mapping

- Instruction Code - Address in control memory(*where routine is located*)

◆ Address Sequencing Capabilities : ***control memory address***

1) Incrementing of the control address register

2) Unconditional branch or conditional branch, depending on status bit conditions

3) Mapping process (bits *of the instruction address for control memory*)

4) A facility for subroutine return

◆ Selection of address for control memory :

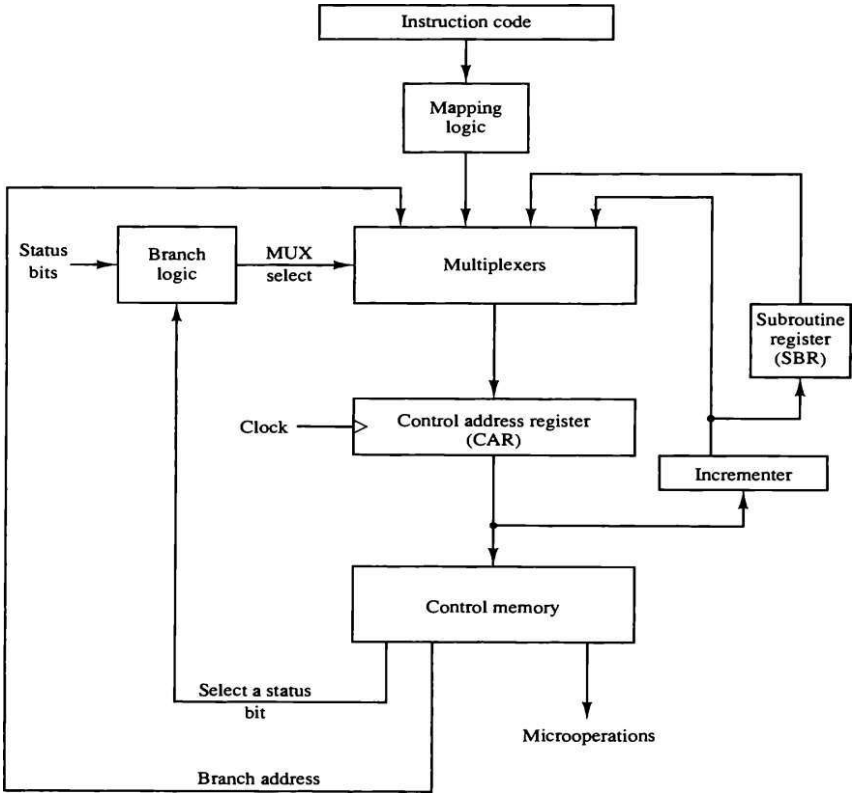


Fig. 2.19 Selection of address for control memory

- Multiplexer
  - CAR Increment
  - JMP/CALL
  - Mapping
  - Subroutine Return
- CAR : Control Address Register
  - » CAR receive the address from4 different paths
    1) Incrementer
    2) Branch address from control memory
    3) Mapping Logic
    4) SBR : Subroutine Register
- SBR : Subroutine Register
  - » Return Address can not be stored in ROM

» Return Address for a subroutine is stored in SBR

◆ Conditional Branching

- Status Bits
  - » Control the conditional branch decisions generated in the ***Branch Logic***
- Branch Logic
  - » Test the specified condition and Branch to the indicated address if the condition is met ; otherwise, the control address register is just incremented.
- Status Bit Test - Branch Logic
  - » 4 X 1 Mux - Input Logic

◆ Mapping of Instruction :

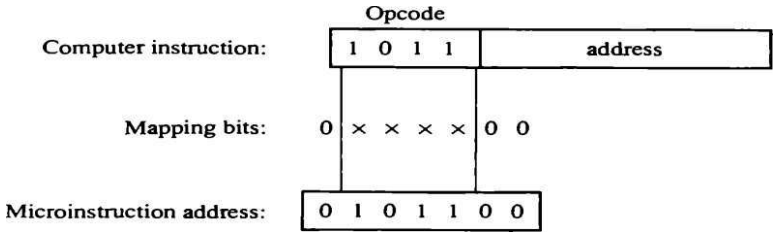


Fig. 2.20 Mapping from instruction code to microinstruction address

- 4 bit Opcode = specify up to 16 distinct instruction
- Mapping Process : Converts *the 4-bit Opcode* to *a 7-bit control memory address*
  - » 1) Place a "0" in the most significant bit of the address
  - » 2) Transfer 4-bit Operation code bits
  - » 3) Clear the two least significant bits of the CAR
- Mapping Function : Implemented by *Mapping ROM* or *PLD*

Control Memory Size : 128 words (= 2^7)

◆ Subroutine

- Subroutines are programs that are used by other routines

» Subroutine can be called from any point within the mainbody of the microprogram

- Microinstructions can be saved by subroutines that use common section of microcode
- Subroutine must have a provision for
  - » storing the return address during a subroutine call
  - » restoring the address during a subroutine return
    - Last-In First Out(LIFO) Register Stack

c. Microprogram Example

◆ Computer Configuration :

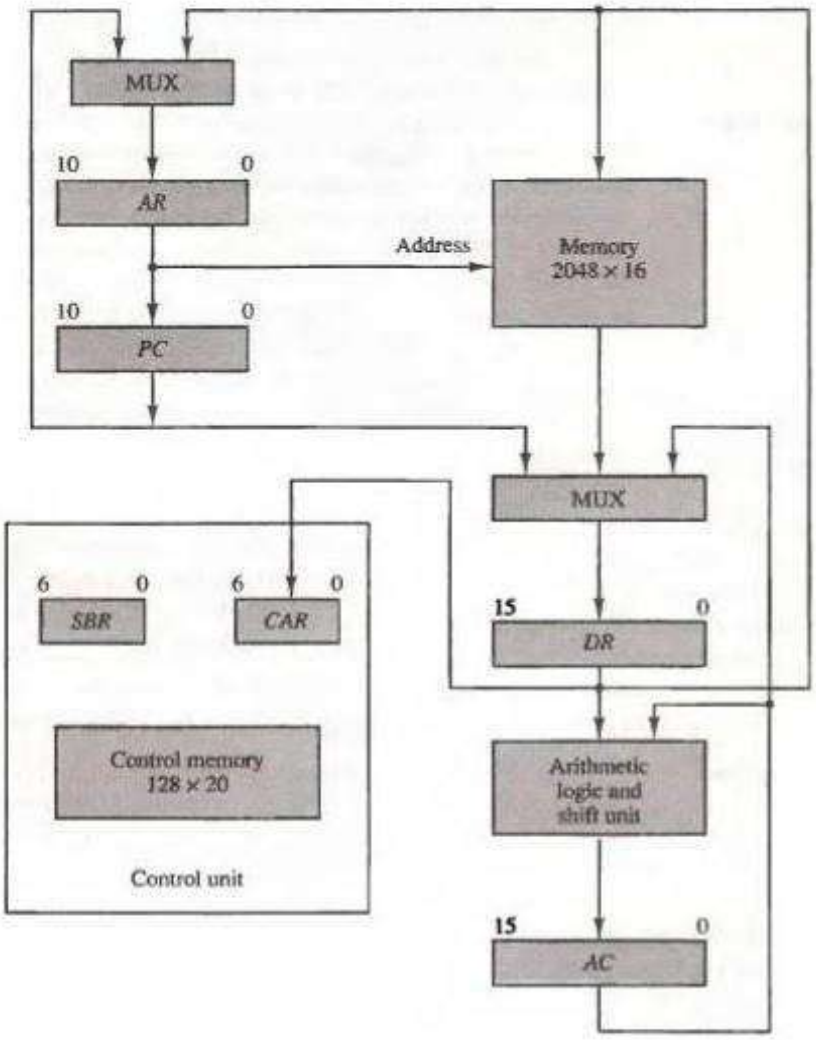
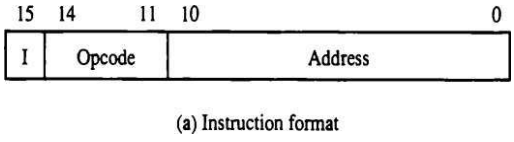


Fig. 2.21 Computer hardware configuration

- 2 Memory : Main memory(*instruction/data*), Control memory(*microprogram*)
  - » Data written to memory come from DR, and Data read from memory can go only to DR
- 4 CPU Register and ALU : DR, AR, PC, AC, ALU
  - » DR can receive information from AC, PC, or Memory (*selected by MUX*)
  - » AR can receive information from PC or DR (*selected by MUX*)
  - » PC can receive information only from AR
  - » ALU performs microoperation with data from AC and DR
- 2 Control Unit Register : SBR, CAR

◆ Instruction Format

- Instruction Format : ***Fig. 2.22***



(a) Instruction format

Fig. 2.22 Computer instruction format

  - » I : 1 bit for indirect addressing
  - » Opcode : 4 bit operation code
  - » Address : 11 bit address for system memory
- Computer Instruction : Fig 2.23

Symbol	Opcode	Description
ADD	0000	$AC \leftarrow AC + M[EA]$
BRANCH	0001	If $(AC < 0)$ then $(PC \leftarrow EA)$
STORE	0010	$M[EA] \leftarrow AC$
EXCHANGE	0011	$AC \leftarrow M[EA], M[EA] \leftarrow AC$

EA is the effective address

Fig. 2.23 Computer instruction- four computer instruction

◆ Microinstruction Format : Fig. 2.24

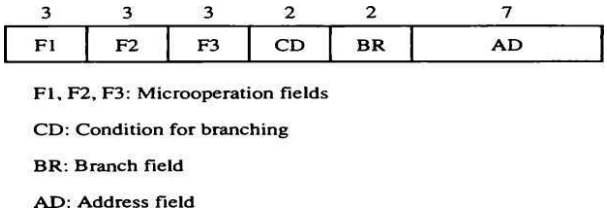


Fig. 2.24 Microinstruction code format (20 bits)

- 3 bit Micro operation Fields : F1, F2, F3
  - » 21Microoperation : ***Tab. 2.4***
  - » two or more conflicting microoperations can not be specified simultaneously
    - 010 001 000
  - » Clear AC to 0 and subtract DR from AC at the same time
  - » Symbol **DRTAC**(F1 = 100)
    - stand for a transfer from DR to AC (*T* = *to*)
- » 2 bit Condition Fields : CD
  - » 00 : Unconditional branch, **U** = 1
  - » 01 : Indirect address bit, **I** = DR(15)
  - » 10 : Sign bit of AC, **S** = AC(15)
  - » 11 : Zero value in AC, **Z** = AC = 0
- » 2 bit Branch Fields : BR
  - » 00 : **JMP**
    - Condition = 0 :
    - Condition = 1 :
  - » 01 : **CALL**
    - Condition = 0 :
    - Condition = 1 :
  - » 10 : **RET**
  - » 11 : **MAP**
- » 7 bit Address Fields : AD128 word : 128 X 20 bit

Table 2.4 Symbols and binary code for microinstruction fields

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0\text{–}10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0\text{-}10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow$ shl AC	SHL
100	$AC \leftarrow$ shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	$DR(15)$	I	Indirect address bit
10	$AC(15)$	S	Sign bit of AC
11	$AC = 0$	Z	Zero value in AC

BR	Symbol	Function
00	JMP	$CAR \leftarrow AD$ if condition = 1
		$CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow AD$, $SBR \leftarrow CAR + 1$ if condition = 1
		$CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR(2\text{–}5) \leftarrow DR(11\text{–}14)$, $CAR(0,1,6) \leftarrow 0$

◆ Symbolic Microinstruction

- Label Field : Terminated with a colon (**:**)
- Microoperation Field : one, two, or three symbols, separated by commas
- CD Field : U, I, S, or Z
- BR Field : JMP, CALL, RET, or MAP
- AD Field
  a. Symbolic Address : Label (= Address)
  b. Symbol "NEXT" : next address
  c. Symbol "RET" or "MAP" : AD field = 0000000

• ORG : Pseudoinstruction(*define the origin, or first address of routine)*

◆ Fetch (**Sub**)Routine

• Memory Map(*128 words*) : ***Tab. 2.5, Tab. 2.6***

Table 2.5 Symbolic micro program

TABLE 7-2 Symbolic Microprogram (Partial)

Label	Microoperations	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ACTDR, DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	
INDRCT:	READ	U	JMP	NEXT
	DRTAR	U	RET	

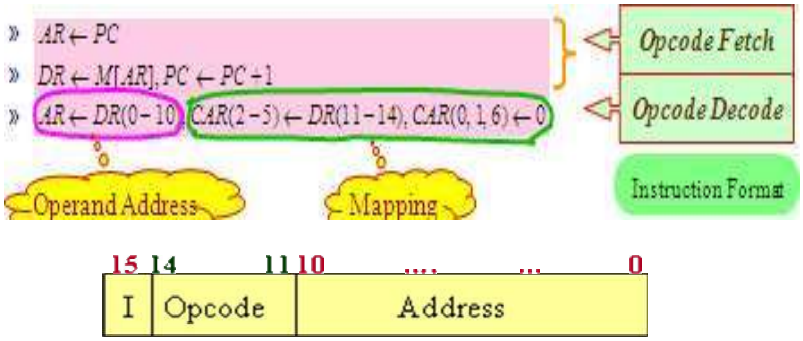
Table 2.6 Binary micro program for control memory (Partial)

Micro Routine	Address		Binary Microinstruction					
	Decimal	Binary	F1	F2	F3	CD	BR	AD
ADD	0	0000000	000	000	000	01	01	1000011
	1	0000001	000	100	000	00	00	0000010
	2	0000010	001	000	000	00	00	1000000
	3	0000011	000	000	000	00	00	1000000
BRANCH	4	0000100	000	000	000	10	00	0000110
	5	0000101	000	000	000	00	00	1000000
	6	0000110	000	000	000	01	01	1000011
	7	0000111	000	000	110	00	00	1000000
STORE	8	0001000	000	000	000	01	01	1000011
	9	0001001	000	101	000	00	00	0001010
	10	0001010	111	000	000	00	00	1000000
	11	0001011	000	000	000	00	00	1000000
EXCHANGE	12	0001100	000	000	000	01	01	1000011
	13	0001101	001	000	000	00	00	0001110
	14	0001110	100	101	000	00	00	0001111
	15	0001111	111	000	000	00	00	1000000
FETCH	64	1000000	110	000	000	00	00	1000001
	65	1000001	000	100	101	00	00	1000010
	66	1000010	101	000	000	00	11	0000000
INDRCT	67	1000011	000	100	000	00	00	1000100
	68	1000100	101	000	000	00	10	0000000

» Address 0 to 63 : Routines for the 16 instruction

» Address 64 to 127 : Any other purpose (*Subroutines : FETCH, INDRCT*)

Microinstruction for FETCH Subroutine



- Fetch Subroutine : address 64

```
        ORG 64
FETCH:  PCTAR          U   JMP   NEXT
        READ, INCPC    U   JMP   NEXT
        DRTAR          U   MAP
```

- Symbolic Microprogram : ***Tab. 7-2***
  - The execution of MAP microinstruction in FETCH subroutine
    - » Branch to address 0**xxxx**00 (*xxxx = 4 bit Opcode*)
      - ADD : 0 **0000** 00 = 0
      - BRANCH : 0 **0001** 00 = 4
      - STORE : 0 **0010** 00 = 8
      - EXCHANGE : 0 **0011** 00 = 12, (*16, 20, ... , 60*)
  - Indirect Address : **I** = 1
  - Indirect Addressing : $AR \leftarrow M[AR]$
  - INDRCT subroutine

Label	Microoperation	CD	BR	AD
INDRCT:	READ	U	JMP	NEXT
	DRTAR	U	RET	0

$$DR \leftarrow M[AR]$$
$$AR \leftarrow DR$$

- Execution of Instruction
  - ADD instruction
  - BRANCH instruction
  - STORE instruction
  - EXCHANGE instruction

d. Design of Control Unit

- Decoding of Microinstruction Fields : ***Fig. 2.25***

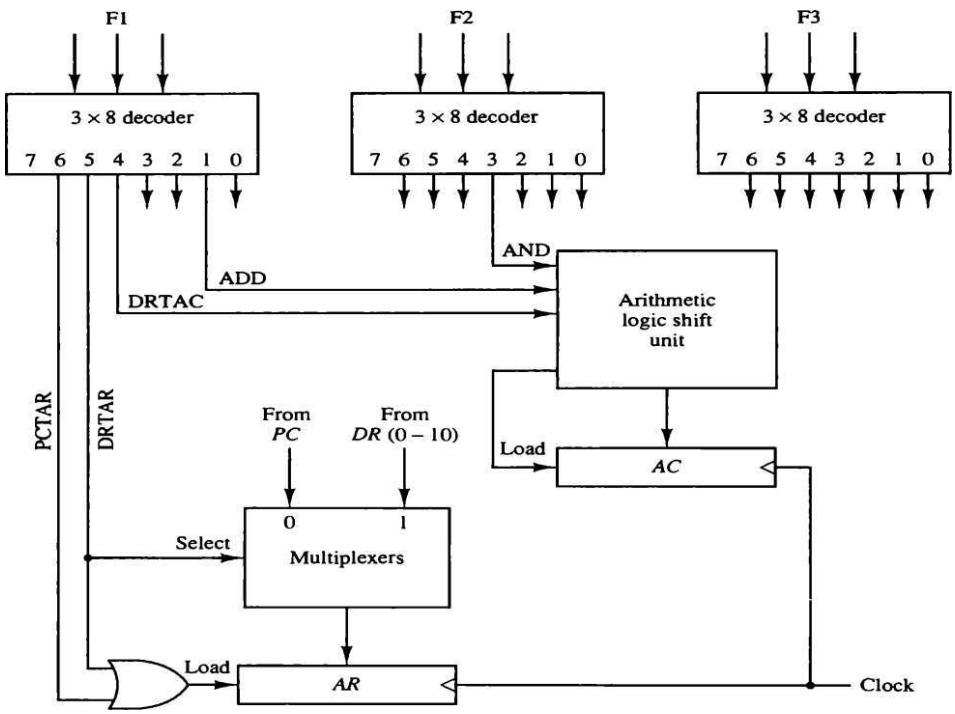


Fig. 2.25 Decoding of micro operation fields

- F1, F2, and F3 of Microinstruction are decoded with a 3 x 8 decoder
- Output of decoder must be connected to the proper circuit to initiate the corresponding microoperation
- F1 = 101 (5) : **DRTAR**
- F1 = 110 (6) : **PCTAR**
- Output 5 and 6 of decoder F1 are connected to the load input of AR(*two input of OR gate*)
- Multiplexer select the data from DR when output 5 is active
- Multiplexer select the data from AC when output 5 is inactive
- Arithmetic Logic Shift Unit
- Control signal of ALU in *hardwired control*
- Control signal will be now come from the *output of the decoders* associated with the AND, ADD, and DRTAC.

◆ Microprogram Sequencer : ***Fig. 2.26***

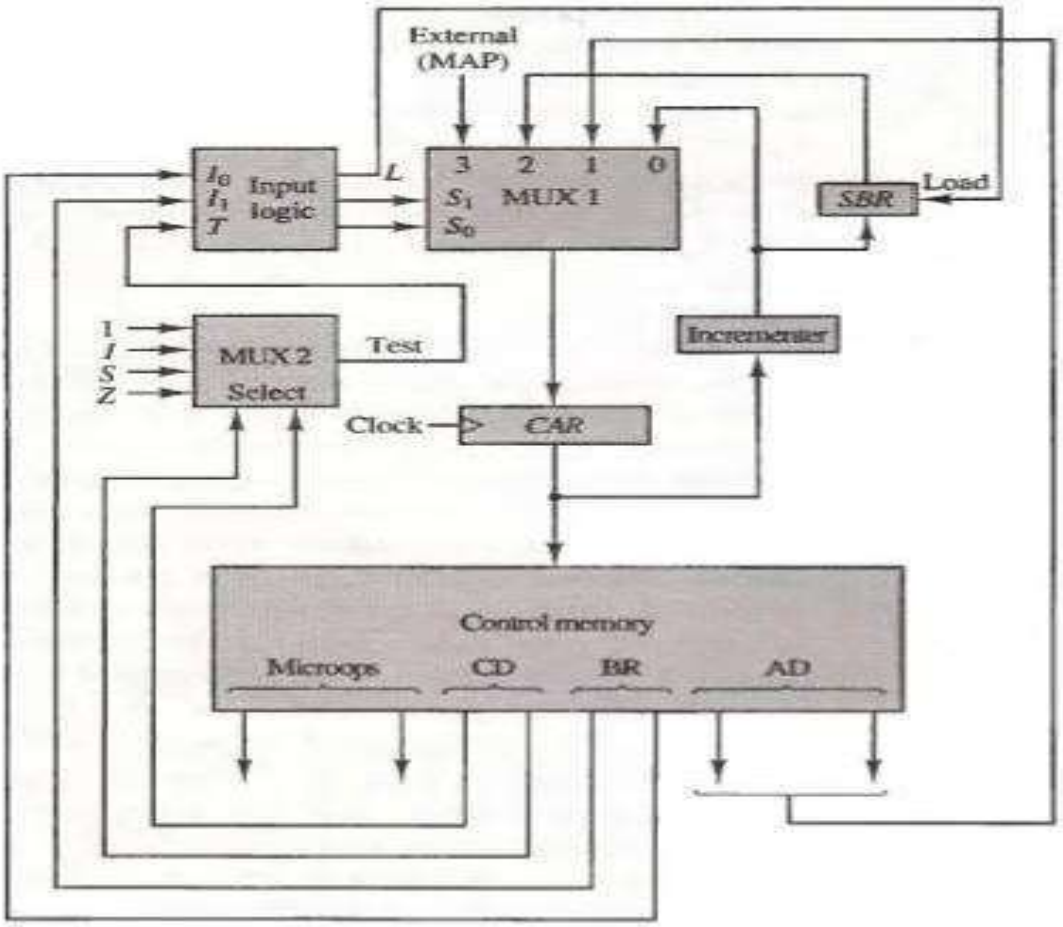


Fig. 2.26 Microprogram sequencer for a control memory

- Microprogram Sequencer select the next address for control memory
- MUX 1
  - Select an address source and route to CAR
    - CAR + 1
    - JMP/CALL
    - Mapping
    - Subroutine Return
- MUX 2
  - Test a status bit and the result of the test is applied to aninput logic circuit
  - One of 4 Status bit is selected by Condition bit (**CD**)
- Design of Input Logic Circuit

- Select one of the source address(**S_0, S_1**) for CAR
- Enable the load input(**L**) in SBR
- Input Logic Truth Table : ***Tab. 2.7***
  - » Input :
    - I_0, I_1 from Branch bit (**BR**)
    - T from MUX 2 (**T**)
  - » Output :
    - MUX 1 Select signal (**S_0, S_1**)

$S1 = I_1I_0' + I_1I_0 = I_1(I_0' + I_0) = I_1$

$S0 = I_1'I_0'T + I_1'I_0T + I_1I_0$

$= I_1'T(I_0' + I_0) + I_1I_0$

$= I_1'T + I_1I_0$

- SBR Load signal (**L**)

$L = I_1'I_0T$

$$S_1 = I_1$$
$$S_0 = I_1I_0 + I_1'T$$
$$L = I_1'I_0T$$

Table 2.7: Input logic truth table for micro program sequencer

BR Field	Input			MUX 1		Load *SBR*
	I_1	I_0	T	S_1	S_0	L
0 0	0	0	0	0	0	0
0 0	0	0	1	0	1	0
0 1	0	1	0	0	0	0
0 1	0	1	1	0	1	1
1 0	1	0	×	1	0	0
1 1	1	1	×	1	1	0

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – III – Memory Organization – SCSA1402

UNIT III Memory Organization

Memory Hierarchy - Main memory - auxiliary Memory - Associative Memory –Cache Memory - Virtual memory

Memory Hierarchy

The memory unit is an essential component in any digital computer since It Is needed for storing programs and data. The memory unit that communicates directly with the CPU is called the main memory. Devices that provide backup storage are called auxiliary memory. The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. They are used for storing system programs, large data files, and other backup information. The memory hierarchy system consists of all storage devices employed in a computer system from the slow but high-capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory accessible to the high-speed processing logic. Figure 1 illustrates the components in a typical memory hierarchy

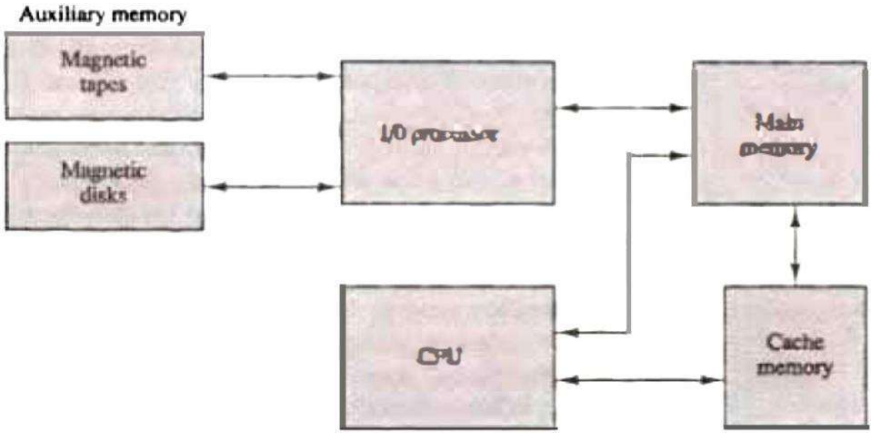


Fig.1 Memory hierarchy in a typical Computer system

Main memory

Main memory is the central storage unit in a computer system. It is a relatively large and fast memory used to store programs and data during the computer operation. The principal technology used for the main memory is based on semi conductor integrated circuits. Integrated circuits RAM chips are available in two possible operating modes, static and dynamic.

• Static RAM – Consists of internal flip flops that store the binary information.

• Dynamic RAM – Stores the binary information in the form of electric charges that are applied to capacitors.

Most of the main memory in a general purpose computer is made up of RAM integrated circuit chips, but a portion of the memory may be constructed with ROM chips.

• Read Only Memory –Store programs that are permanently resident in the computer and for tables of constants that do not change in value once the production of the computer is completed.

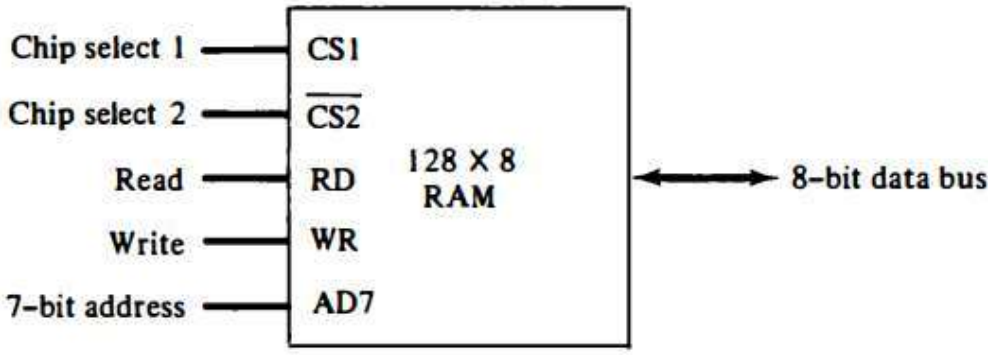
The ROM portion of main memory is needed for storing an initial program called a Bootstrap loader.

• Boot strap loader –function is start the computer software operating when power is turned on.

• Boot strap program loads a portion of operating system from disc to main memory and control is then transferred to operating system.

RAM and ROM CHIP

• RAM chip –utilizes bidirectional data bus with three state buffers to perform communication with CPU



(a) Block diagram

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

(b) Function table

Fig.2: Block Diagram of a RAM Chip

The block diagram of a RAM Chip is shown in Fig.2. The capacity of memory is 128 words of eight bits (one byte) per word. This requires a 7-bit address and an 8-bit bidirectional data bus. The read and write inputs specify the memory operation and the two chips select (CS) control inputs are enabling the chip only when it is selected by the microprocessor. The read and write inputs are sometimes combined into one line labelled R/W. The function table listed in Fig.12-2(b) specifies the operation of the RAM chip. The unit is in operation only when CS1=1 and CS2=0.The bar on top of the second select variable indicates that this input is enabled when it is equal to 0. If the chip select inputs are not enabled, or if they are enabled but the read or write inputs are not enabled, the memory is inhibited and its data bus is in a high-impedance state. When CS1=1 and CS2=0, the memory can be placed in a write or read mode. When the WR input is enabled, the memory stores a byte from the data bus into a location specified by the address input lines. When the RD input is enabled, the content of the selected byte is placed into the data bus. The RD and WR signals control the memory operation as well as the bus buffers associated with the bidirectional data bus.

Fig.3: Typical ROM Chip

A ROM chip is organized externally in a similar manner. However, since a ROM can only read, the data bus can only be in an output mode. The block diagram of a ROM chip is shown in fig.3. The nine address lines in the ROM chip specify any one of the512 bytes stored in it. The two chip select inputs must be CS1=1 and CS2=0 for the unit to operate. Otherwise, the data bus is in a high-impedance state.

Memory Address Map

The interconnection between memory and processor is then established from knowledge of the size of memory needed and the type of RAM and ROM chips available. The addressing of memory can be established by means of a table that specify the memory address assigned to each chip. The table called Memory address map, is a pictorial representation of assigned address space for each chip in the system.

Table 1: Memory address map for microcomputer

Component	Hexadecimal address	Address bus									
		10	9	8	7	6	5	4	3	2	1
RAM 1	0000–007F	0	0	0	x	x	x	x	x	x	x
RAM 2	0080–00FF	0	0	1	x	x	x	x	x	x	x
RAM 3	0100–017F	0	1	0	x	x	x	x	x	x	x
RAM 4	0180–01FF	0	1	1	x	x	x	x	x	x	x
ROM	0200–03FF	1	x	x	x	x	x	x	x	x	x

The memory address map for this configuration is shown in table. The component column specifies whether a RAM or a ROM chip is used. The hexadecimal address column assigns a range of hexadecimal equivalent addresses for each chip. The address bus lines are listed in the third column. The RAM chips have 128 bytes and need seven address lines. The ROM chip has 512 bytes and needs 9 address lines.

Memory Connection to CPU

RAM and ROM chips are connected to a CPU through the data and address buses. The low order lines in the address bus select the byte within the chips and other lines in the address bus select a particular chip through its chip select inputs.

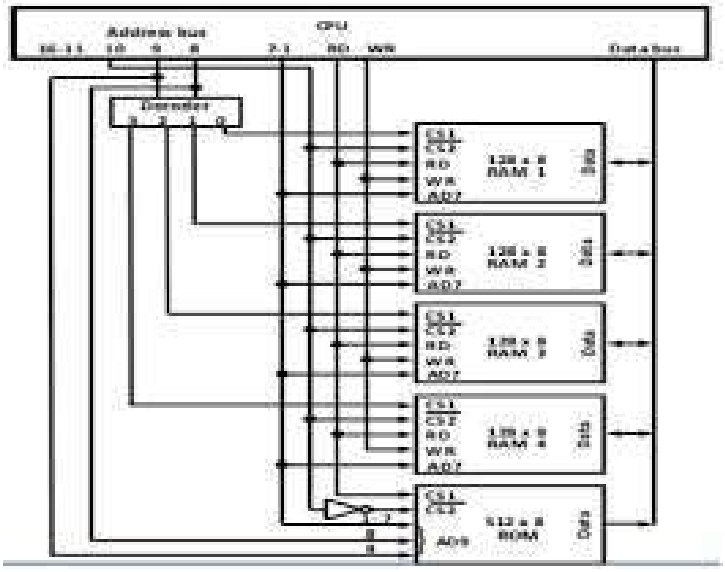


Fig.4: Memory connection to the CPU

The connection of memory chips to the CPU is shown in the above figure. This configuration gives a memory capacity of 512 bytes of RAM and 512 bytes of ROM. Each RAM receives the seven low-order bits of the address bus to select one of 128 possible bytes. The particular RAM chip selected is determined from lines 8 and 9 in the address bus. This is done through a 2 X 4 decoder whose outputs go to the CS1 inputs in each RAM chip. Thus, when address lines 8 and 9 are equal to 00, the first RAM chip is selected. When 01, the second RAM chip is select, and so on. The RD and WR outputs from the microprocessor are applied to the inputs of each RAM chip. The selection between RAM and ROM is achieved through bus line 10. The RAMs are selected when the bit in this line is 0, and the ROM when the bit is 1. Address bus lines 1 to 9 are applied to the input address of ROM without going through the decoder. The data bus of the ROM has only an output capability, whereas the data bus connected to the RAMs can transfer information in both directions.

Auxiliary Memory

The time required to find an item stored in memory can be reduced considerably if stored data can be identified for access by the content of the data itself rather than by an address. A memory unit accessed by content is called an associative memory or content addressable memory (CAM).

• CAM is accessed simultaneously and in parallel on the basis of data content rather than by specific address or location

• Associative memory is more expensive than a RAM because each cell must have storage capability as well as logic circuits

• Argument register –holds an external argument for content matching

• Key register –mask for choosing a particular field or key in the argument word

Hardware Organization

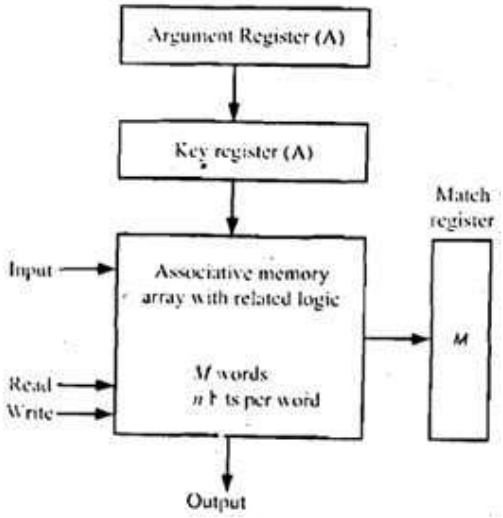


Fig.5: Block Diagram of associative memory

It consists of a memory array and logic for m words with n bits per word. The argument register A and key register K each have n bits, one for each bit of a word. The match register M has m bits, one for each memory word. Each word in memory is compared in parallel with the content of the argument register. The words that match the bits of the argument register set a corresponding bit in the match register. After the matching process, those bits in the match register that have been set b indicate the fact that their corresponding words have been matched. Reading is accomplished by a sequential access to memory for those words whose corresponding bits in the match register have been set

Read and Write operation Read Operation

If more than one word in memory matches the unmasked argument field , all the matched words will have 1's in the corresponding bit position of the match register

• In read operation all matched words are read inλ sequence by applying a read signal to each word line whose corresponding Mi bit is a logic 1

• In applications where no two identical items are stored in the memory , only one word may match , in which case we can use Mi output directly as a read signal for the corresponding word

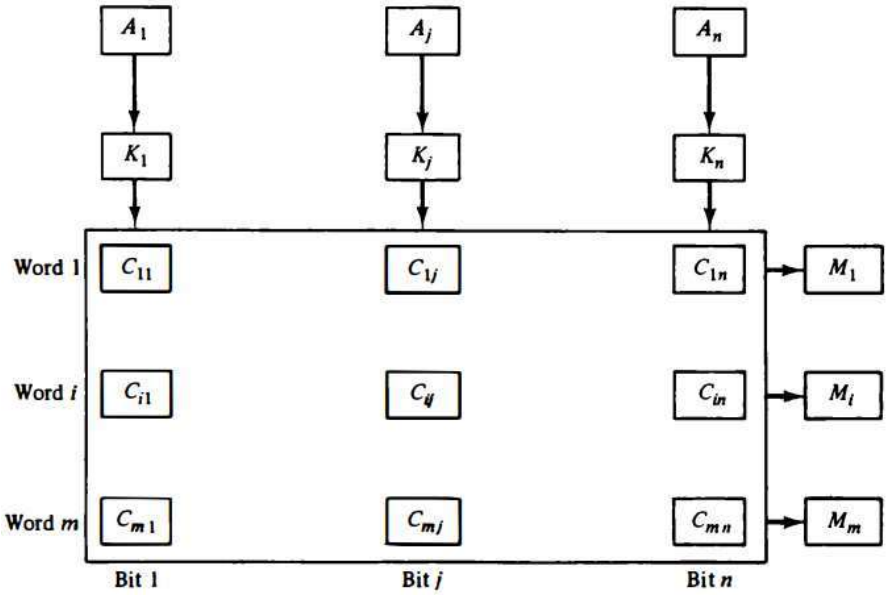


Fig.6:Associative memory of m word, n cells per word

The relation between the memory array and external registers in an associative memory is shown in Fig.6. The cells in the array are marked by the letter C with two subscripts. The first subscript gives the word number and second specifies the bit position in the word. Thus cell Cij is the cell for bit j in word i. A bit Aj in the argument register is compared with all the bits in column j of the array provided that kj =1.This is done for all columns j=1,2,….n. If a match occurs between all the unmasked bits of the argument and the bits in word I, the corresponding bit Mi in the match

register is set to 1. If one or more unmasked bits of the argument and the word do not match, Mi is cleared to 0.

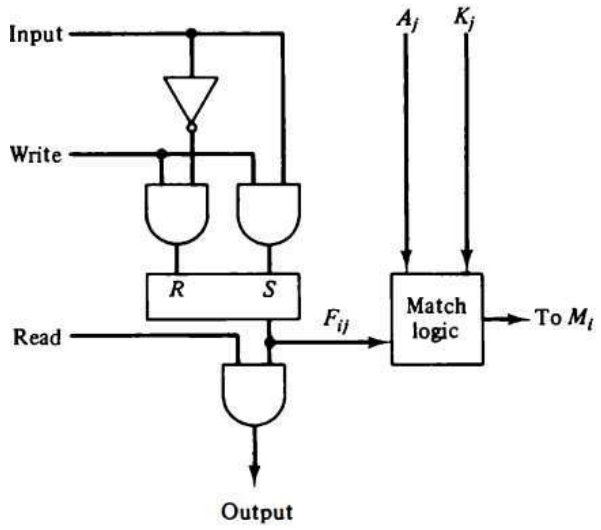


Fig.7: One cell of associative memory

It consists of flip-flop storage element Fij and the circuits for reading, writing, and matching the cell. The input bit is transferred into the storage cell during a write operation. The bit stored is read out during a read operation. The match logic compares the content of the storage cell with corresponding unmasked bit of the argument and provides an output for the decision logic that sets the bit in Mi.

Read and Write operation

Read Operation

If more than one word in memory matches the unmasked argument field all the matched words will have 1's in the corresponding bit position of the match register

• In read operation all matched words are read in λ sequence by applying a read signal to each word line whose corresponding Mi bit is a logic 1

• In applications where no two identical items are stored in the memory , only one word may match , in which case we can use Mi output directly as a read signal for the corresponding word

Write Operation

Can take two different forms

1. Entire memory may be loaded with new information

2.Unwanted words to be deleted and new words to be inserted

1.Entire memory : writing can be done by addressing each location in sequence – This makes it random access memory for writing and content addressable memory for reading – number of lines needed for decoding is d Where m = 2 d , m is number of words.

2.Unwanted words to be deleted and newλ words to be inserted :

• Tag register is used which has as many bits as there are words in memory

• For every active (valid) word in memory , the corresponding bit in tag register is set to 1

• When word is deleted the corresponding tag bit is reset to 0

• The word is stored in the memory by scanning the tag register until the first 0 bit is encountered After storing the word the bit is set to 1.

Cache Memory

Effectiveness of cache mechanism is based on a property of computer programs called "locality of reference"

• The references to memory at any given time interval tend to beλ confined within a localized areas

• Analysis of programs shows that most of their execution time is spent on routines in which instructions are executed repeatedly These instructions may be – loops, nested loops , or few procedures that call each other

• Many instructions in localized areas of program are executed

• Repeatedly during some time period and reminder of the program is accessed infrequently

This property is called "Locality of Reference".

A special very- high- speed memory called a Cache is sometimes used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate. The cache memory is employed in computer systems to compensate for the speed differential between main memory access time and processor logic. CPU logic is usually faster than main memory access time, with the result that processing speed is limited primarily by the speed of main memory.

A technique used to compensate for the mismatch in operating speeds is to employ an extremely fast, small cache between the CPU and main memory whose access time Is dose to processor logic dock cycle time. The cache is used for storing segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations.

Analysis of a large number of typical programs has shown that the references to memory at any given interval of time tend to be confined within a few localized areas in memory. This phenomenon is known as the property of locality of reference locality of reference. When a program loop is executed, the CPU repeatedly refers to the set of instructions in memory that constitute the loop. Every time a given subroutine is called, its set of instructions are fetched from memory. Thus loops and subroutines tend to localize the references to memory for fetching instructions. The result of all these observations is the locality of reference property, which states that over a short interval of time, the addresses generated by a typical program refer to a few localized areas of memory repeatedly, while the remainder of memory is accessed relatively infrequently.

If the active portions of the program and data are placed in a fast small memory, the average memory access time can be reduced, thus reducing the total execution time of the program. Such a

fast small memory is referred to as a cache memory. It is placed between the CPU and main memory

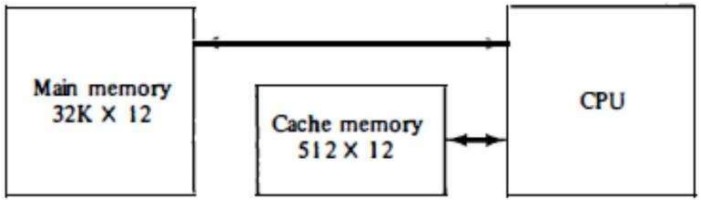


Fig.8:Example of cache memory

The main memory can store 32k words of 12 bits each. The cache is capable of storing 512 of these words at any given time. For every word stored , there is a duplicate copy in main memory. The CPU communicates with both memories. It first sends a 15 bit address to cache. If there is a hit, the CPU accepts the 12 bit data from cache. If there is a miss, the CPU reads the word from main memory and the word is then transferred to cache.

• When a read request is received from CPU, contents of a block of memory words containing the location specified are transferred in to cache

• When the program references any of the locations in this block , the contents are read from the cache Number of blocks in cache is smaller than number of blocks in main memory

• Correspondence between main memory blocks and those in the cache is specified by a mapping function

• Assume cache is full and memory word not in cache is referenced

• Control hardware decides which block from cache is to be removed to create space for new block containing referenced word from memory

• Collection of rules for making this decision is called “Replacement algorithm ”

Read/ Write operations on cache

• Cache Hit Operation

• CPU issues Read/Write requests using addresses thatλ refer to locations in main memory

• Cache control circuitry determines whether requested word currently exists in cache

• If it does, Read/Write operation is performed on the appropriate location in cache (Read/Write Hit)

Read/Write operations on cache in case of Hit

• In Read operation main memory is not involved.

• In Write operation two things can happen.

1.Cache and main memory locations are updated λ simultaneously (“ Write Through ”) OR

2. Update only cache location and mark it as " Dirty or λ Modified Bit " and update main memory location at the time of cache block removal (" Write Back " or " Copy Back ") .

Read/Write operations on cache in case of Miss Read Operation

• When addressed word is not in cache Read Miss occurs there are two ways this can be dealt with

1.Entire block of words that contain the requested word is copied from main memory to cache and the particular word requested is forwarded to CPU from the cache (Load Through) (OR)

2.The requested word from memory is sent to CPU first and then the cache is updated (Early Restart)

Write Operation

• If addressed word is not in cache Write Miss occurs

• If write through protocol is used information is directly written in to main memory

• In write back protocol , block containing the word is first brought in to cache , the desired word is then overwritten.

Mapping Functions

• Correspondence between main memory blocks and those in the cache is specified by a memory mapping function

• There are three techniques in memory mapping

1. Direct Mapping

2. Associative Mapping

3. Set Associative Mapping

Direct mapping:

A particular block of main memory can be brought to a particular block of cache memory. So, it is not flexible.

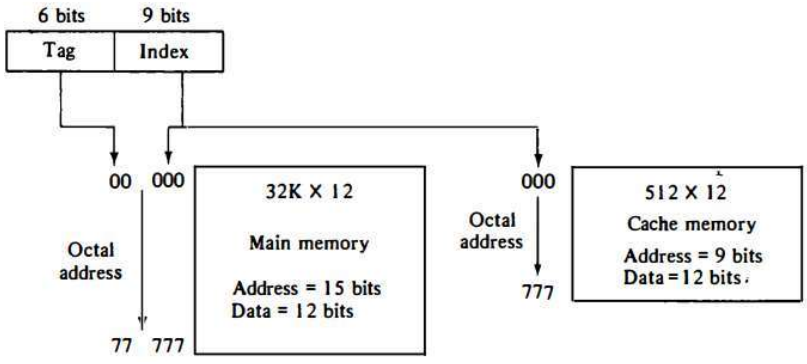


Fig.9:Addressing relationship between main and cache memories

The CPU address of 15 bits is divided into two fields. The nine least significant bits constitute the index field and remaining six bits form the tag field. The main memory needs an address that includes both the tag and the index bits. The number of bits in the index field is equal to the number of address bits required to access the cache memory.

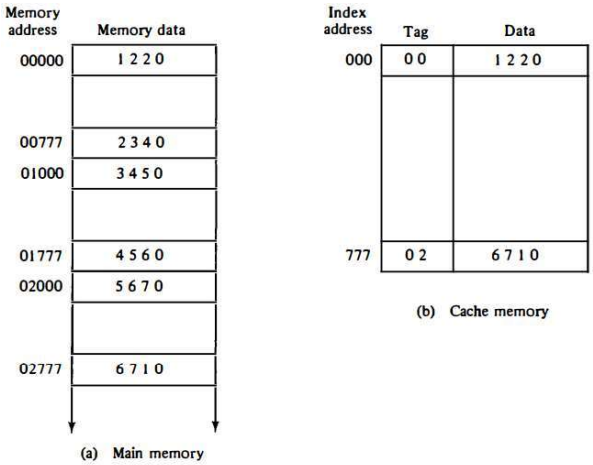


Fig.10:Direct mapping cache organization

The direct mapping cache organization uses the n- bit address to access the main memory and the k-bit index to access the cache. Each word in cache consists of the data word and associated tag. When a new word is first brought into the cache, the tag bits are stored alongside the data bits. When the CPU generates a memory request, the index field is used the index field is used for the address to access the cache. The tag field of the CPU address is compared with the tag in the word read from the cache. If the two tags match, there is a hit and the desired data word is in cache. If there is no match, there is a miss and the required word is read from main memory.

Associative mapping

In this mapping function, any block of Main memory can potentially reside in any cache block position. This is much more flexible mapping method.

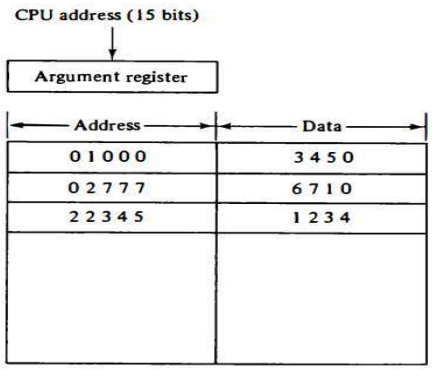


Fig.11:Associative mapping cache (all numbers in octal)

The associative memory stores both address and content(data) of the memory word. This permits any location in cache to store any word from main memory. The diagram shows three words presently stored in the cache. The address value of 15 bits is shown as a five-digit ctal number and its corresponding 12-bit word is shown as a four-digit octal number. A CPU address of 15-bits is placed in the argument register and the associative memory is searched for a matching address. If address is found, the corresponding 12-bit data is read and sent to the CPU. If no match occurs, the main memory is accessed for the word.

Set-associative mapping

In this method, blocks of cache are grouped into sets, and 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 in between to the other two methods.

Fig.12:2 way set associative cache memory

The octal numbers listed in Fig.12-15 are with reference to the main memory contents. When the CPU generates a memory request, the index values of the address is used to access the cache. The tag field of the CPU address is then compared with both tags in the cache to determine if a match occurs. The comparison logic dine by an associative search of the tags in the set similar to an associative memory search thus the name “Set Associative”

Replacement Policies

• When the cache is full and there is necessity to bring new data to cache , then a decision must be made as to which data from cache is to be removed

• The guideline for taking a decision about which data is to be removed is called replacement policy Replacement policy depends on mapping

• There is no specific policy in case of Direct mapping as we have no choice of block placement in cache Replacement Policies In case of associative mapping

• A simple procedure is to replace cells of the cache in round robin order whenever a new word is requested from memory

• This constitutes a First-in First-out (FIFO) replacement policy

In case of set associative mapping

• Random replacement

• First-in First-out (FIFO) (item chosen is the item that has been in the set longest)

• Least Recently Used (LRU) (item chosen is the item λ that has been least recently used by CPU)

Virtual Memory

Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program generated addresses are translated automatically to the corresponding machine addresses. The size of virtual storage is limited by the addressing scheme of the computer system and amount of secondary memory is available not by the actual number of the main storage locations.

It is a technique that is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory. All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time. This means that a process can be swapped in and out of main memory such that it occupies different places in main memory at different times during the course of execution.

A process may be broken into number of pieces and these pieces need not be continuously located in the main memory during execution. The combination of dynamic run-time address translation and use of page or segment table permits this

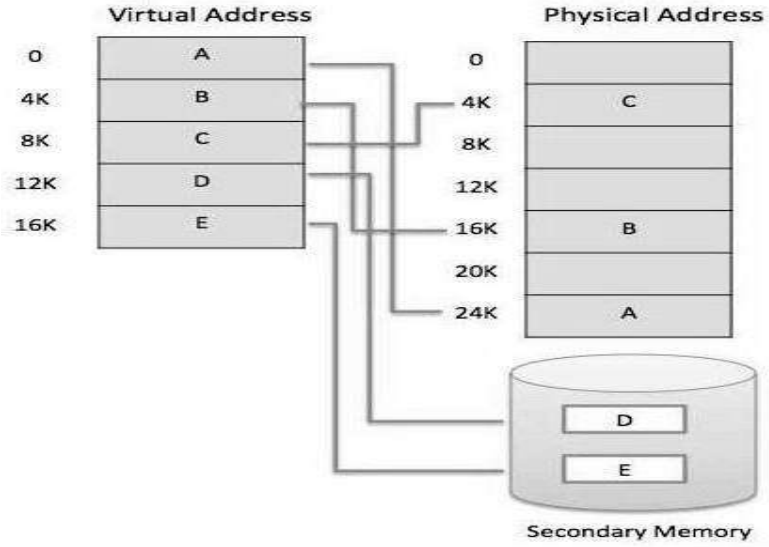


Fig.13: virtual memory

Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – IV – INPUT AND OUTPUT ORGANIZATION – SCS1402

Introduction I/O System

Peripherals:

The input and output devices that are attached to the computer are called peripheral devices.

Eg: Keyboard, printer, display unit.

- Peripherals that provide auxiliary storage to the system are magnetic disks and tapes.
- Printers provide a permanent record on paper of computer output data or text.
  The 3 types of printers:
  1) Daisy wheel
  2) Dot matrix
  3) Laser
- Magnetic tapes are used for storing files of data.
  Access – Sequential
  Slowest and cheapest method.
  Tapes can be removed when not in use.

- Magnetic disks are used for bulk storage of programs and data.
  Access – by moving the read/write mechanism to a track in the magnetized surface.

  ASCII –American Standard Code for Information Interchange
  It is a 7 bit code. Most computers manipulate an 8 bit quantity as a single unit called a byte. ASCII characters are often stored one per byte.

Input Output Interface

It provides a method for transferring information between internal storage and external I/O devices. Peripherals need special communication links for interfacing with CPU. The purpose of the communication link is to resolve the differences between the central computer and each peripheral.

The major differences are:

1. Peripherals are electromagnetic and electromechanical devices, whereas CPU and memory are electronic devices.
2. The data transfer rate of peripherals is slower than that of the CPU.

3. The operating modes of peripherals are different from each other.
4. Data codes and formats in peripherals differ from the word format of CPU and memory.

To resolve the differences, special hardware components are included between CPU and peripherals to supervise and synchronize all the input and output transfers called interface units.

I/O Bus and Interface Modules

The communication between the processor and several peripherals through the I/O Bus is shown in the following figure.

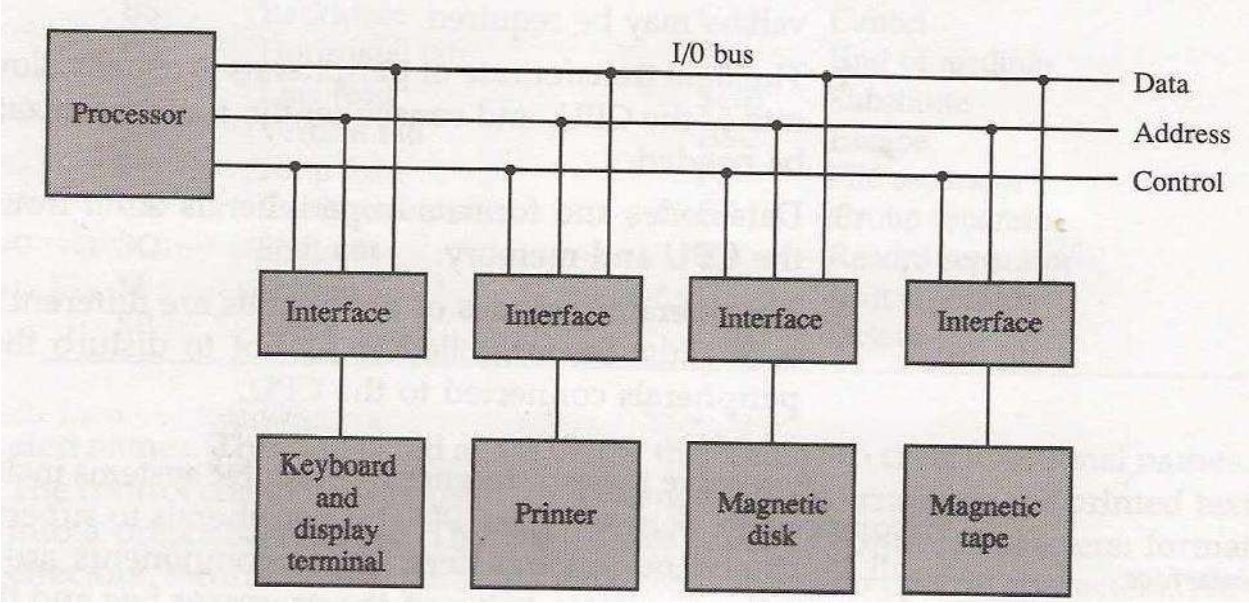


Figure 1 I/O Bus and Interface Modules

I/O Command:

The function code is referred to as an I/O command. It is an instruction that is executed in the interface and its attached peripheral unit.

An interface may receive four types of commands. They are control, status, data input and data output

Control command: It is issued to activate the peripheral and to inform it what to do.

Status command: It is used to test various status conditions in the interface and the peripheral.

Data input command: The interface receives an item of data from the peripheral and places it in its buffer register.

Data output command: The command causes the interface to respond by transferring data from the bus into one of its registers.

I/O versus memory bus

There are 3 ways that computer buses can be used to communicate with memory and I/O.

1. Use two separate buses, one for memory and the other for I/O.
2. Use one common bus for both memory and I/O but have separate control lines for each.
3. Use one common bus for memory and I/O with common control lines.

Isolated versus Memory mapped I/O

Isolated I/O :

1) The CPU has distinct input and output instructions.
2) Isolates memory and I/O addresses.
3) Each has its own address space.

Memory mapped I/O

1) No specific input output instructions.
2) Use memory related instuctions for accessing data.
3) Do not distinguish between memory and I/O addresses.
4) Memory and I/O share the same set of addresses.

Example of I/O interface :

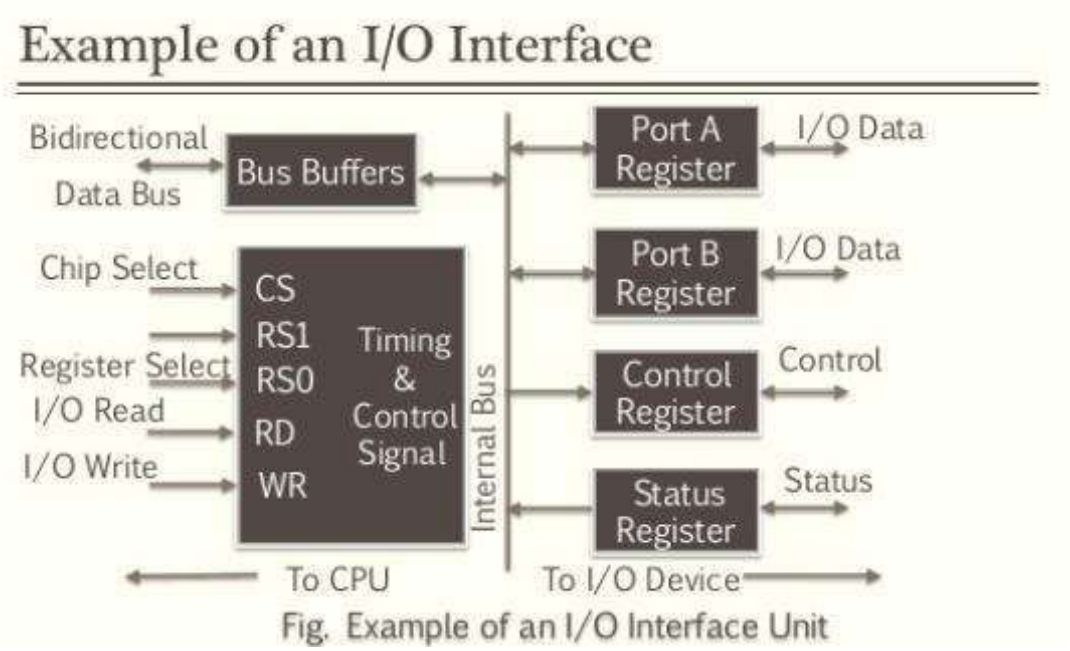


Figure 2 I/O interface

An example of an I/O interface unit is shown in the following block diagram.

CS	RSI	RS0	Register selected
0	x	x	None: dauAs inhigh—inpedance
1	0	0	Port *A* register
1	0	1	Port *B* register
1	1	0	Control registcr
l	i	1	Sums register

Asynchronous data transfer :

- Control signals are transmitted between the two communicating units to indicate the time at which data is being transmitted.
- A strobe pulse is supplied by one of the units to indicate to the other unit when the transfer has to occur.
- The unit receiving the data item responds with another control signal to acknowledge receipt of the data.
- This type of agreement between two independent units is referred to as handshaking.

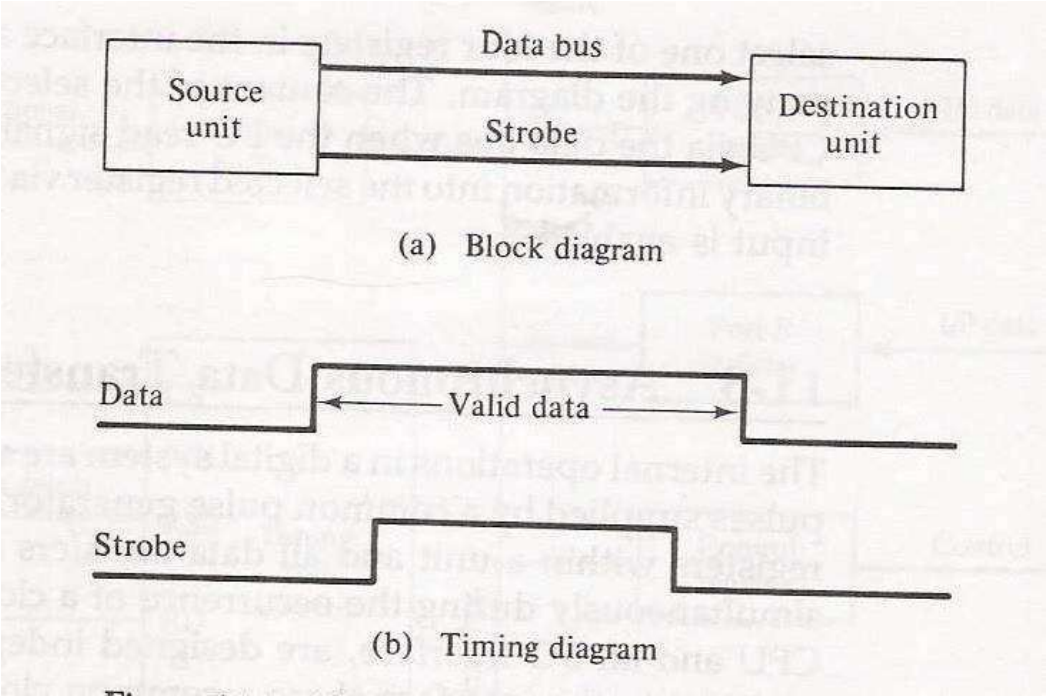


Figure 3 Asynchronous data transfer

Source initiated strobe for data transfer

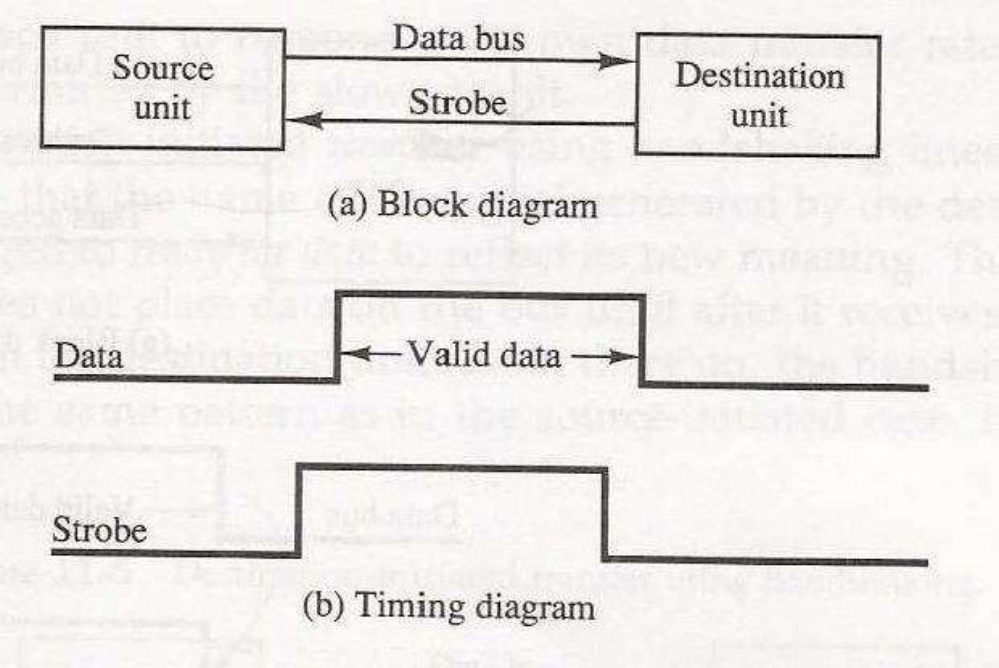


Figure 4 Timing Diagram

Destination initiated strobe for data transfer

Disadvantage of strobe method:

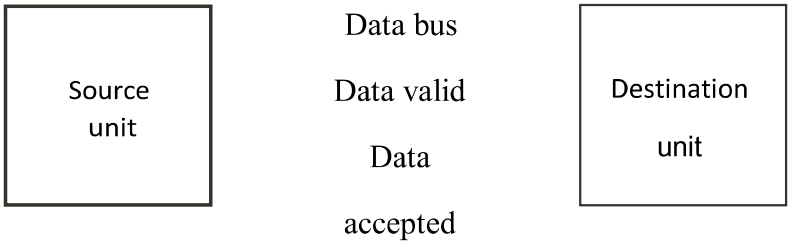
The source/ destination unit that initiates the transfer has no way of knowing whether the destination/source unit has actually received the data item that was placed in the bus.

Handshaking:

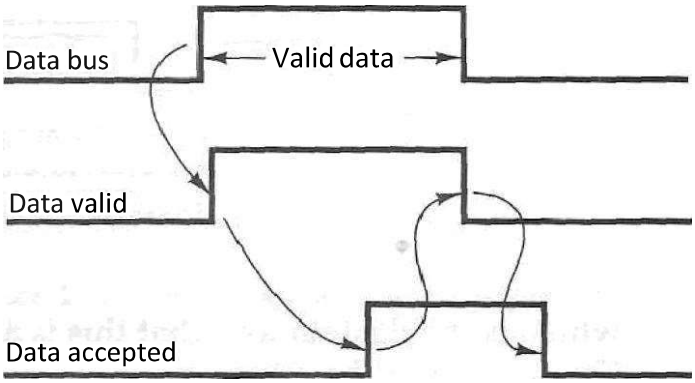
The two handshaking lines are data valid, which is generated by the source unit and data accepted, generated by the destination unit.

1) The source unit initiates the transfer by placing the data on the bus and enabling its data valid signal.
2) The data accepted signal is activated by the destination unit after it accepts the data from the bus.
3) The source unit then disables its data valid signal, which invalidates the data on the bus.
4) The destination unit then disables its data accepted signal and the system goes into initial state.
5) The source does not send the next data item until after the destination unit shows its readiness to accept new data by disabling its data accepted signal,

This scheme allows arbitrary delays from one state to the next and permits each unit to respond at its own data transfer rate. The rate of transfer is determined by the slowest unit.



(a) Block diagram



(b) Timing diagram

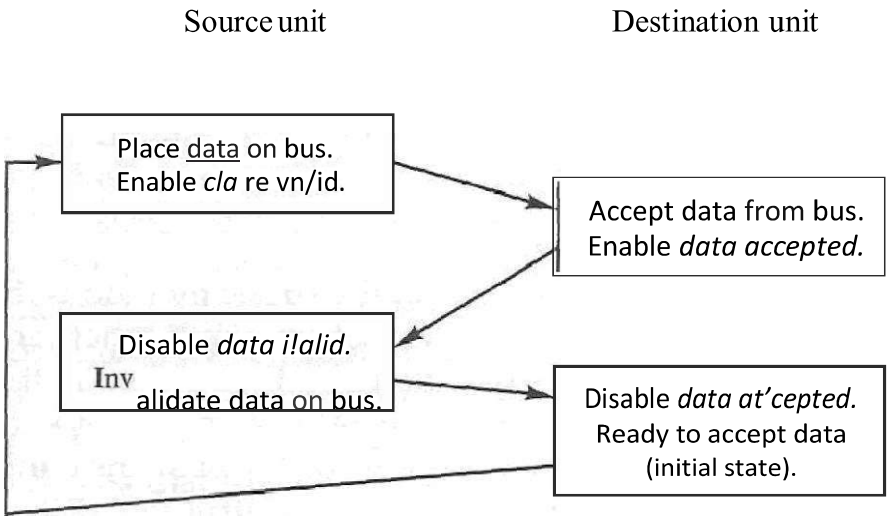


Figure 5 Sequence of events

Source initiated transfer using handshaking

Asynchronous serial transfer

- Serial transmission may be synchronous or asynchronous.
- In serial synchronous transmission, two units share a common clock frequency and the bits are transmitted at a rate dictated by the clock pulses.
- A serial asynchronous data transmission technique used in many interactive terminals employs special bits that are inserted at both ends of the character code.
- Each character consists of three fields : a start bit, the character bits and stop bits.
- The transmitter rests at the 1- state when no characters are transmitted.
- The first bit, called the start bit is always a 0 and is used to indicate the beginning of a character,
- The last bit called the stop bit is always a 1.

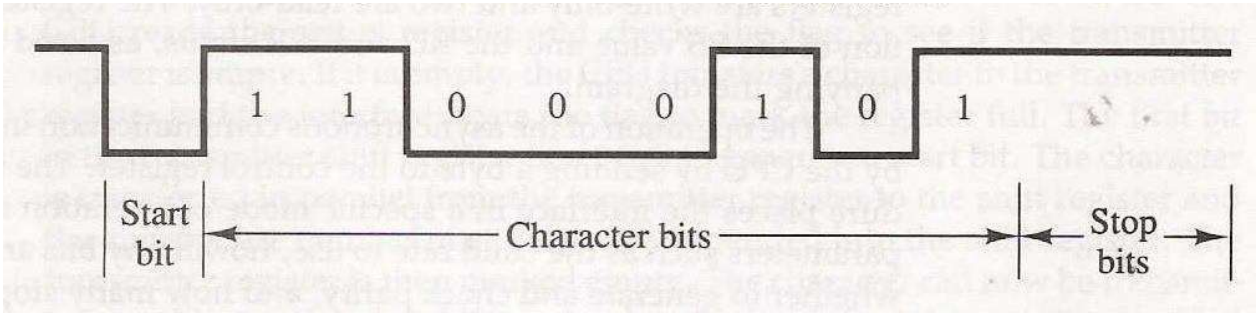


Figure 6 Asynchronous serial transfers

Asynchronous serial transmission

UART:

The integrated circuits that are specially designed to provide the interface between computer and similar interactive terminals is called an asynchronous communication interface or a universal asynchronous receiver – transmitter (UART)

Baud rate

The rate at which serial information is transmitted and is equivalent to the data transfer in bits per second.

FIFO buffer

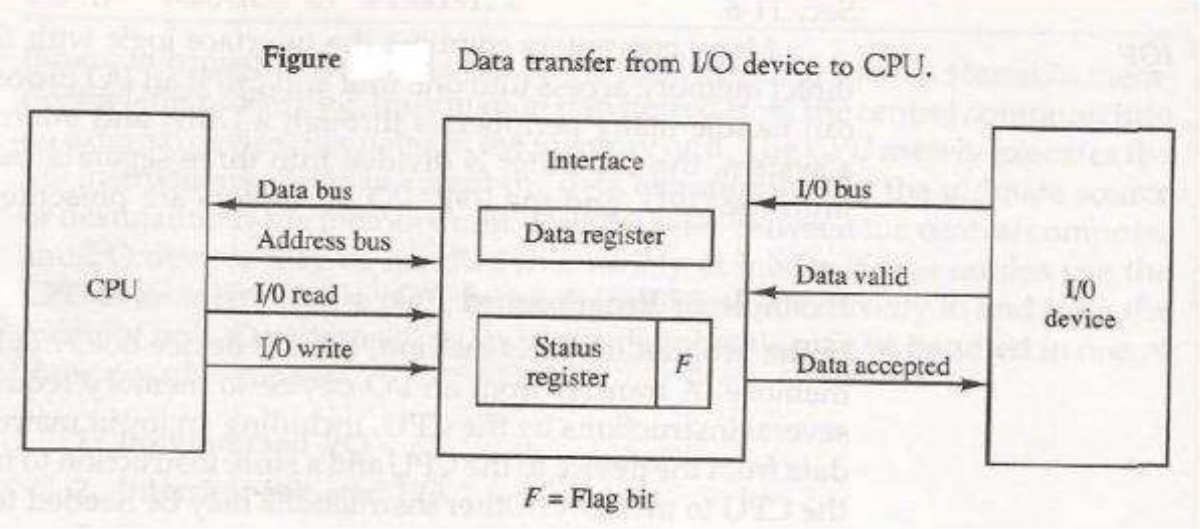
- A first in first out (FIFO) buffer is a memory unit that stores information in such a manner that the item first in is the item first out.
- A FIFO buffer comes with separate input and output terminals.
- It can input data and output data at two different rates and the output data are always in the same order in which the data entered the buffer.
- Useful in some applications when data are transferred asynchronously.

Modes of transfer

Data transfer to and from peripherals may be handled in one of the three modes

1. Programmed I/O
2. Interrupt initiated I/O
3. Direct memory access

Programmed I/O



- Each data item is initiated by an instruction in the program.
- The transfer is to and from a CPU register and peripheral.
- Transferring data under program control requires constant monitoring of the peripheral by the CPU.
- The CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer.
- Time consuming process since it keeps the processor busy needlessly.

Interrupt initiated I/O

- The interface monitors the device.
- When the device is ready for data transfer, the interface generates an interrupt request to the computer.
- Upon detecting the external interrupt signal, the CPU momentarily stops the task it is processing, branches to a service program to process the I/O transfer and then returns to the task it was originally performing.

DMA

- The interface transfers data into and out of the memory unit through the memory bus.
- The CPU initiates the transfer by supplying the interface with the starting address and the number of words needed to be transferred and then proceeds to execute other tasks.
- When the transfer is made, the DMA requests memory cycles through the memory bus.
- When the request is granted by the memory controller, the DMA transfers the data directly into memory.
- The CPU delays its memory access operation to allow the direct memory I/O transfer.

Priority Interrupt

There are number of IO devices attached to the computer. They are all capable of generating the interrupt.

When the interrupt is generated from more than one device, priority interrupt system is used to determine which device is to be serviced first.

Devices with high speed transfer are given higher priority and slow devices are given lower priority. Establishing the priority can be done in two ways:

- Using Software
- Using Hardware

A pooling procedure is used to identify highest priority in software means.

Polling Procedure : There is one common branch address for all interrupts.

Branch address contain the code that polls the interrupt sources in sequence.The highest priority is tested first.

The particular service routine of the highest priority device is served.

The disadvantage is that time required to poll them can exceed the time to serve them in large number of IO devices.

Using Hardware: Hardware priority system function as an overall manager

It accepts interrupt request and determine the priorities.

To speed up the operation each interrupting devices has its own interrupt vector.

No polling is required, all decision are established by hardware priority interrupt unit.

It can be established by serial or parallel connection of interrupt lines.

Serial or Daisy Chaining Priority:

Device with highest priority is placed first.

Device that wants the attention send the interrupt request to the CPU.

CPU then sends the INTACK signal which is applied to PI(priority in) of the first device. If it had requested the attention, it place its VAD(vector address) on the bus.

And it block the signal by placing 0 in PO(priority out) If not it pass the signal to next device through PO(priority out) by placing 1.

This process is continued until appropriate device is found.

The device whose PI is 1 and PO is 0 is the device that send the interrupt request.

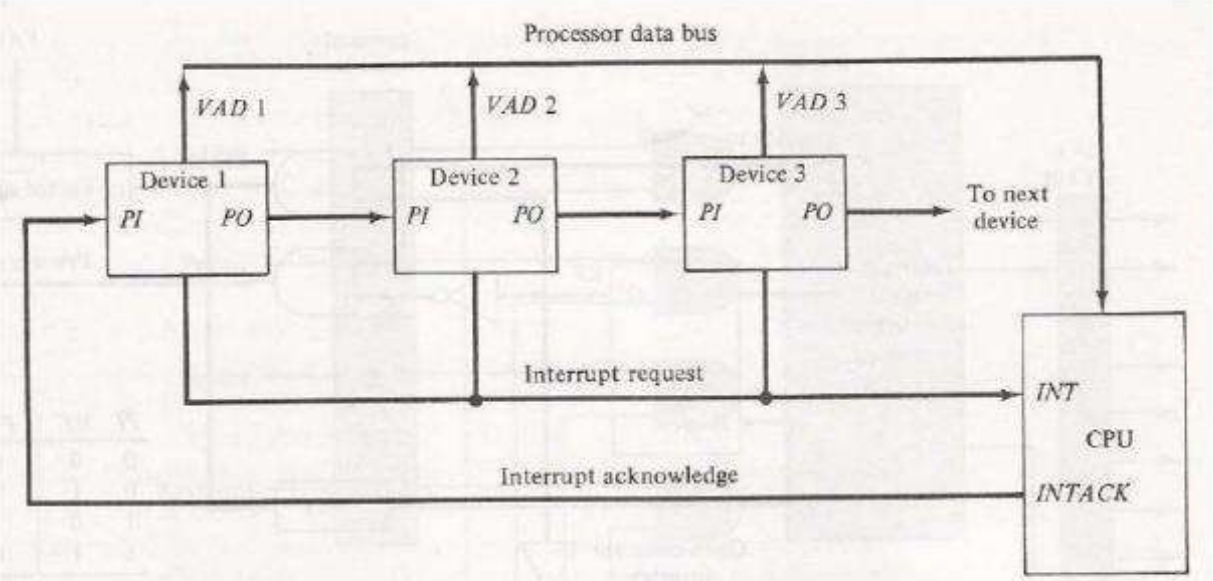


Figure Daisy-chain priority interrupt.

Parallel Priority Interrupt :

It consist of interrupt register whose bits are set separately by the interrupting devices.

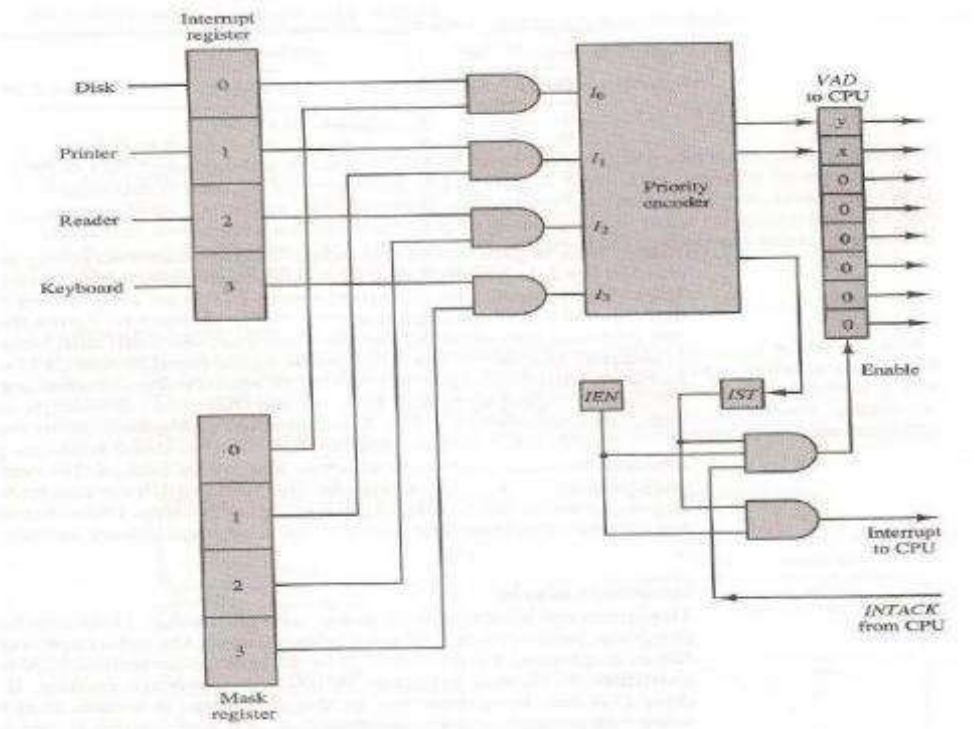
Priority is established according to the position of the bits in the register

Mask register is used to provide facility for the higher priority devices to interrupt when lower priority device is being serviced or disable all lower priority devices when higher is being serviced.

Corresponding interrupt bit and mask bit are ANDed and applied to priority encoder.

Priority encoder generates two bits of vector address.

Another output from it sets IST(interrupt status flip flop).



Priority Encoder Truth Table

Inputs				Outputs			Boolean functions
I_0	I_1	I_2	I_3	x	y	IST	
1	×	×	×	0	0	1	
0	1	×	×	0	1	1	$x = I_0' I_1'$
0	0	1	×	1	0	1	$y = I_0' I_1 + I_0' I_2'$
0	0	0	1	1	1	1	$(IST) = I_0 + I_1 + I_2 + I_3$
0	0	0	0	×	×	0	

IOP

- The Intel 8089 I/O processor is contained in a 40 pin integrated circuit package.
- There are two independent units called channels.

- Reads the message from memory, carries out the operation and notifies the CPU when it has finished.
- Contains 50 basic instructions that can operate on individual bits, on bytes or 16 bit words.
- It can execute programs in a manner similar to a CPU except that the instruction set is specifically chosen to provide efficient input-output processing.
- It provides efficient data transfer between any two components attached to the system bus, such as I/O to memory, memory to memory or I/O to I/O.
- In the Intel 8086/8089 microcomputer system, the 8086 fuctions as the CPU and the 8089 as the IOP.
- The two units share a common memory through a bus controller connected to a system bus, called a “multibus” by Intel.
- The IOP uses a local bus to communicate with various interface units connected to I/O devices.
- The CPU communicates with the IOP by enabling the channel attention line.
- The CPU uses the select line to select one of the two channels in IOP.
- The IOP gets the attention of the CPU by sending an interrupt request.
- The CPU and IOP communicates with each other by writing messages for one another in system memory.
- The CPU prepares the message area and signals the IOP by enabling the channel attention line.
- The IOP reads the message, performs the required I/O functions and executes the appropriate channel program .
- When the channel has completed its program, it issues an interrupt request to the CPU.
- The communication scheme consists of program sections called blocks.
- Each block contains control and parameter information as well as an address pointer to its successor block.
- The address of the control block is passed to each IOP channel during initialization.
- The busy flag indicates whether the IOP is busy or ready to perform a new I/O operation.
- The CCW (Channel command word) is specified by the CPU to indicate the type of operation required from the IOP.

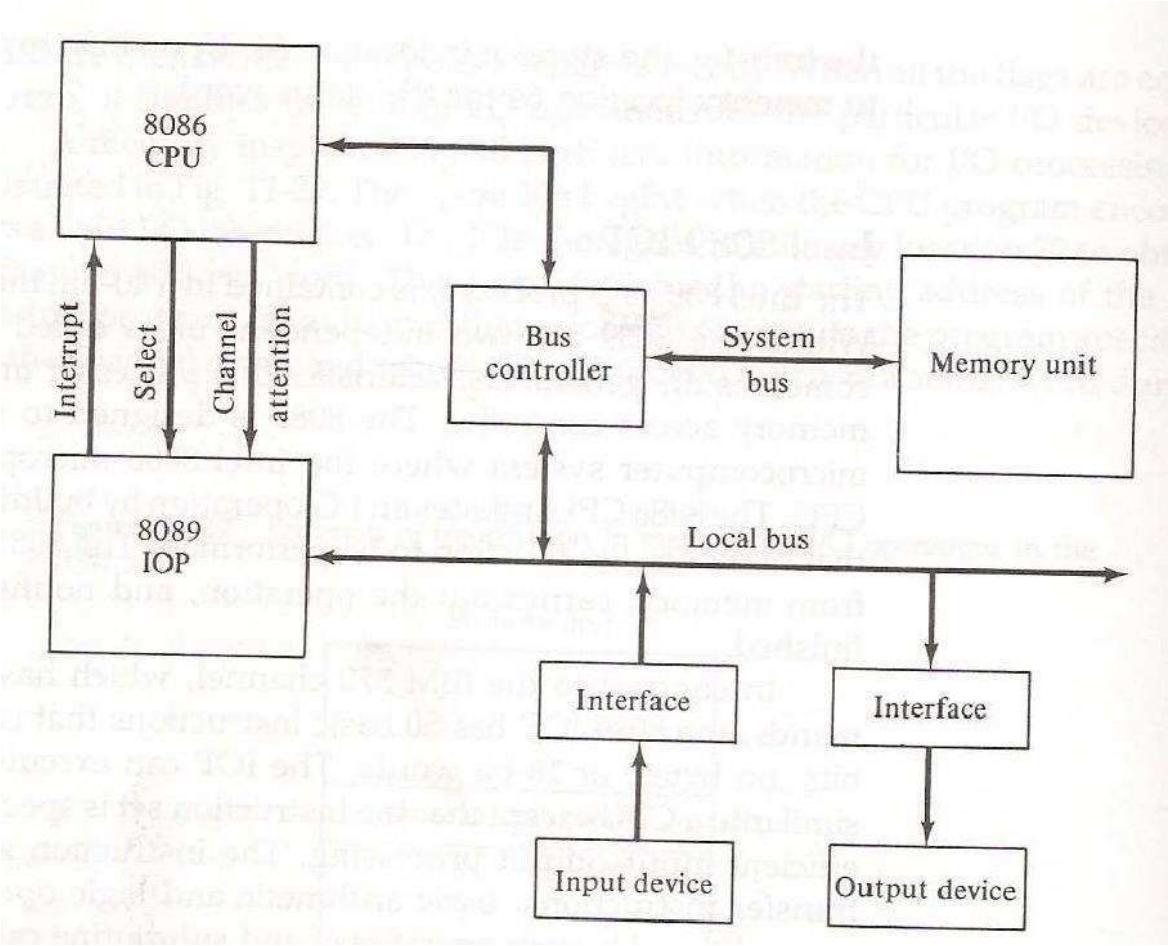


Figure 7 Intel 8086/8089 microcomputer block diagram

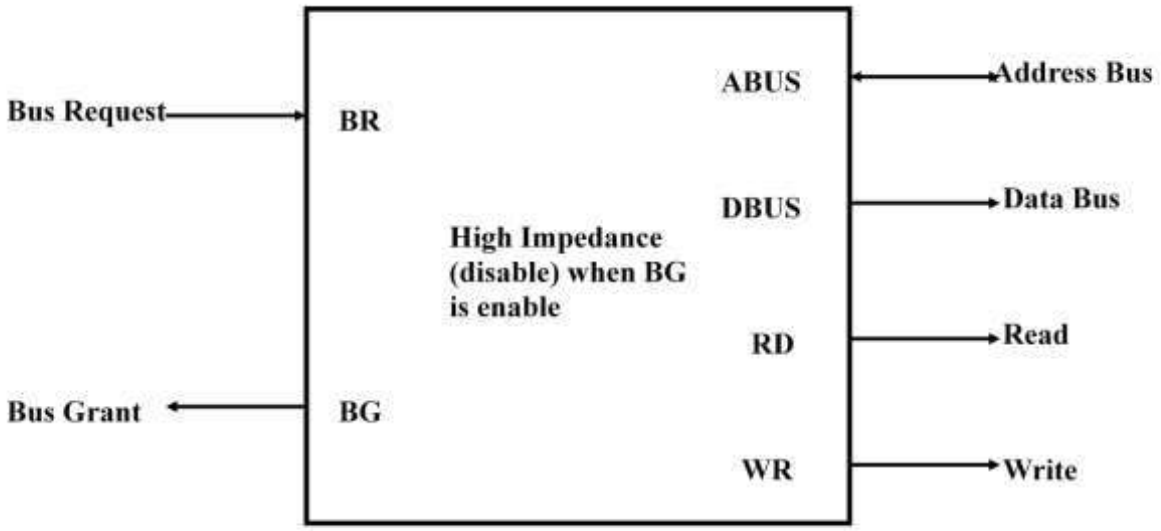
Direct Memory Access (DMA)

Direct memory access (**DMA**) is a feature of computer systems that allows certain hardware subsystems to access main system memory (RAM) independently of the central processing unit (CPU).

Without DMA, when the CPU is using programmed input/output, it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU first initiates the transfer, then it does other operations while the transfer is in progress, and it finally receives an interrupt from the DMA controller when the operation is done. This feature is useful at any time that the CPU cannot keep up with the rate of data transfer, or when the CPU needs to perform useful work while waiting for a relatively slow I/O data transfer. Many hardware systems use DMA, including disk drive controllers, graphics cards, network cards and sound cards. DMA is also used for intra-chip data transfer in multi-core

processors. Computers that have DMA channels can transfer data to and from devices with much less CPU overhead than computers without DMA channels. Similarly, a processing element inside a multi-core processor can transfer data to and from its local memory without occupying its processor time, allowing computation and data transfer to proceed in parallel.

DMA can also be used for "memory to memory" copying or moving of data within memory. DMA can offload expensive memory operations, such as large copies or scatter-gather operations, from the CPU to a dedicated DMA engine. An implementation example is the I/O Acceleration Technology.

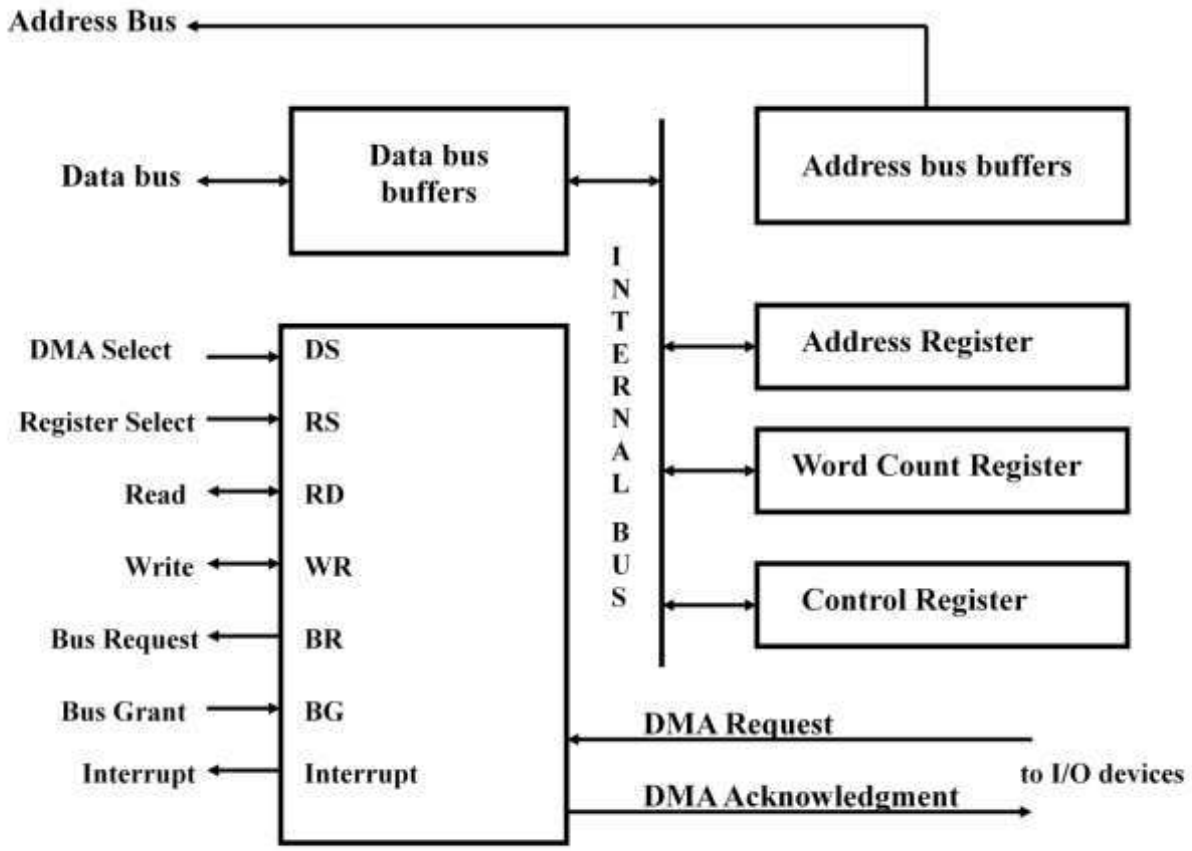


CPU bus Signals for DMA Transfer

Types of modes:

Burst mode

An entire block of data is transferred in one contiguous sequence. Once the DMA controller is granted access to the system bus by the CPU, it transfers all bytes of data in the data block before releasing control of the system buses back to the CPU, but renders the CPU inactive for relatively long periods of time. The mode is also called "Block Transfer Mode". It is also used to stop unnecessary data.



Block Diagram of DMA Controller

Cycle stealing mode

The *cycle stealing mode* is used in systems in which the CPU should not be disabled for the length of time needed for burst transfer modes. In the cycle stealing mode, the DMA controller obtains access to the system bus the same way as in burst mode, using BR (Bus Request) and BG (Bus Grant) signals, which are the two signals controlling the interface between the CPU and the DMA controller. However, in cycle stealing mode, after one byte of data transfer, the control of the system bus is deasserted to the CPU via BG. It is then continually requested again via BR, transferring one byte of data per request, until the entire block of data has been transferred. By continually obtaining and releasing the control of the system bus, the DMA controller essentially interleaves instruction and data transfers. The CPU processes an instruction, then the DMA controller transfers one data value, and so on. On the one hand, the data block is not transferred as quickly in cycle stealing mode as in burst mode, but on the other hand the CPU is not idled for as long as in burst mode. Cycle stealing mode is useful for controllers that monitor data in real time.

Transparent mode

The *transparent mode* takes the most time to transfer a block of data, yet it is also the most efficient mode in terms of overall system performance. The DMA controller only transfers data when the CPU is performing operations that do not use the system buses. It is the primary advantage of the transparent mode that the CPU never stops executing its programs and the DMA transfer is free in terms of time. The disadvantage of the transparent mode is that the hardware needs to determine when the CPU is not using the system buses, which can be complex.

Serial Communication

Data transmission between two points occurs in three different modes

1) Simplex – This line carries information in one direction only.
   Ex: radio and TV broadcasting
2) Half-duplex – This transmission system is capable of transmitting in both directions , but data can be transmitted in only one direction at a time.
   Ex: Modem
3) Full-duplex – This transmission system can send and receive data in both directions simultaneously.
   Ex: a two wire circuit

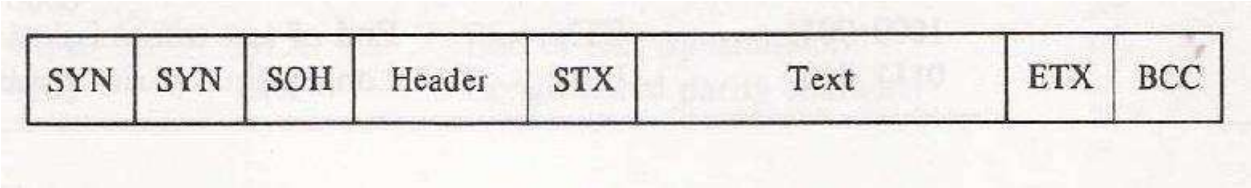


Figure 8 Typical message format for character oriented protocol

- There are a number of control characters used for message formation.
- Each character, including the control characters, is transmitted serially as an 8- bit binary code which consists of the 7 bit ASCII code plus an odd parity bit in the eighth most significant position.
- The two SYN characters are used to synchronize transmitter and receiver.
- The heading starts with the SOH character and continues with two characters that specify the address of the terminal.

Bit oriented protocol

- A frame starts with the 8 bit flag 01111110 followed by an address and control sequence.
- The frame check field is a CRC (cyclic redundancy check) sequence used for detecting errors in transmission.
- The ending flag indicates to the receiving station that the 16 bits just received constitute the CRC bits.
- The ending frame can be followed by another frame, another flag or a sequence of consecutive 1's.
- A frame must have a minimum of 32 bits between flags to accomodate the address, control and frame check fields.

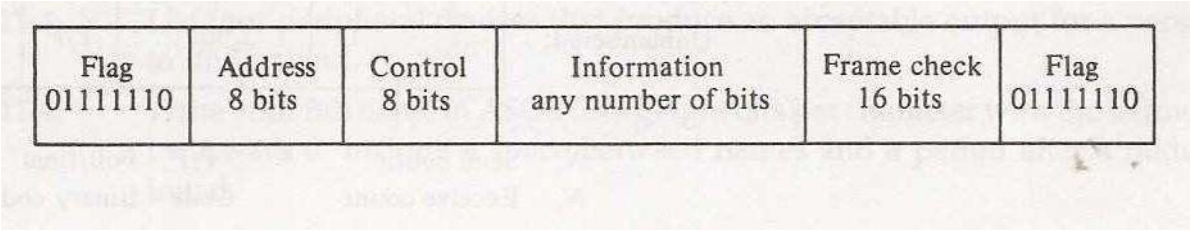


Figure 9 Frame format for bit oriented protocol

REFERENCES :

1. COMPUTER SYSTEM ARCHITECTURE, MORRIS M. MANO, 3RD EDITION, PRENTICE HALL INDIA.

2. HTTP://NPTEL.AC.IN/COURSES

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING
DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

UNIT – V – Characteristics of Multiprocessors – SCSA1402

V. Characteristics of multiprocessors

5.1 Multiprocessor:

- A set of processors connected by a communications network

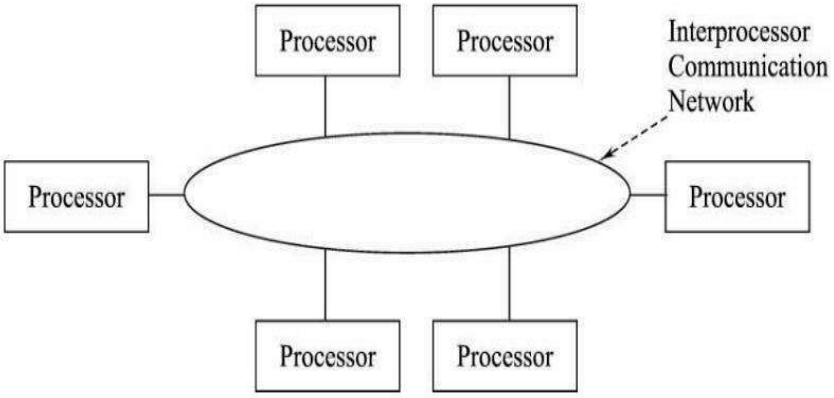


Fig. 5.1 Basic multiprocessor architecture

- A multiprocessor system is an interconnection of two or more CPU's with memory and input-output equipment.
- Multiprocessors system are classified as multiple instruction stream, multiple data stream systems(MIMD).
- There exists a distinction between multiprocessor and multicomputers that though both support concurrent operations.
- In multicomputers several autonomous computers are connected through a network and they may or may not communicate but in a multiprocessor system there is a single OS Control that provides interaction between processors and all the components of the system to cooperate in the solution of the problem.
- VLSI circuit technology has reduced the cost of the computers to such a low Level that the concept of applying multiple processors to meet system performance requirements has become an attractive design possibility.

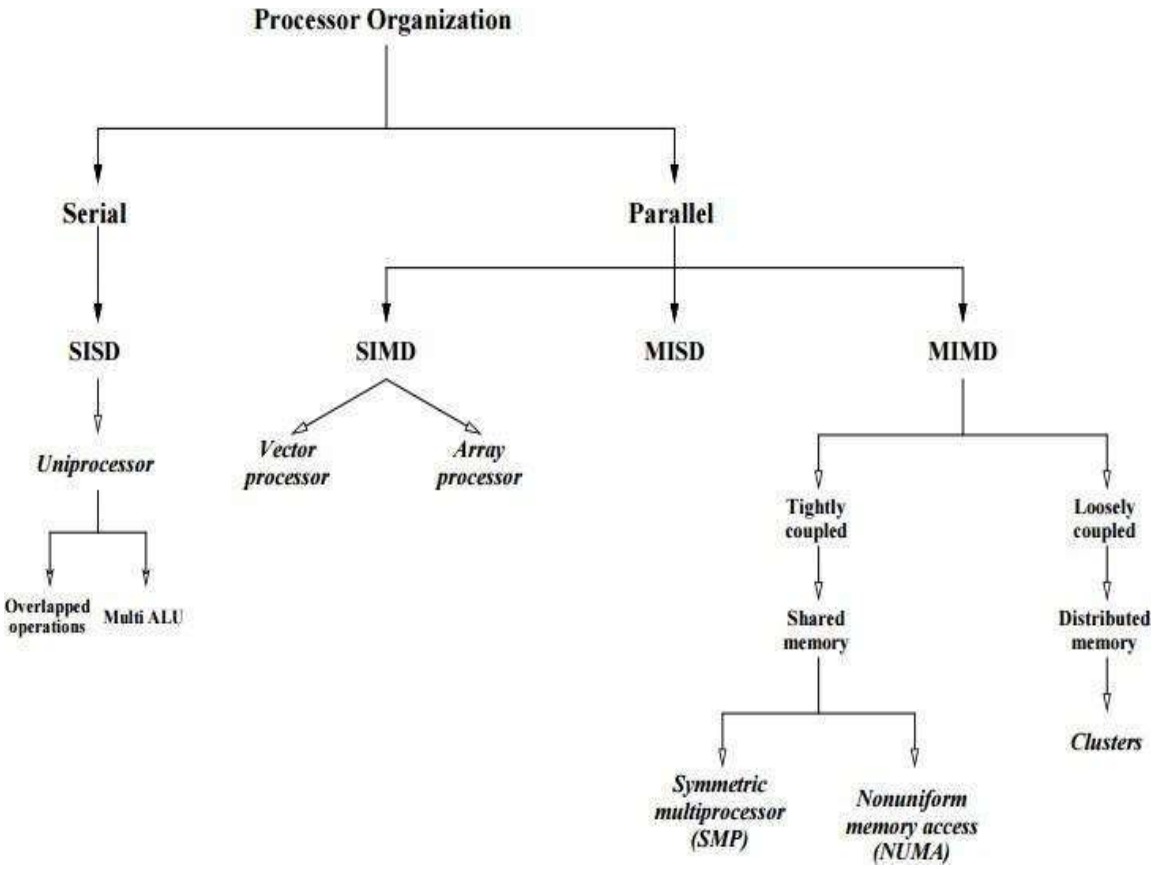


Fig. 5.2 Taxonomy of mono- mulitporcessor organizations

Characteristics of Multiprocessors:

Benefits of Multiprocessing:

1. Multiprocessing increases the reliability of the system so that a failure or error in one part has limited effect on the rest of the system. If a fault causes one processor to fail, a second processor can be assigned to perform the functions of the disabled one.

2. Improved System performance. System derives high performance from the fact that computations can proceed in parallel in one of the two ways:

a) Multiple independent jobs can be made to operate in parallel.

b) A single job can be partitioned into multiple parallel tasks.

This can be achieved in two ways:

- The user explicitly declares that the tasks of the program be executed in parallel

- The compiler provided with multiprocessor s/w that can automatically detect parallelism in program. Actually it checks for Data dependency

COUPLING OF PROCESSORS

Tightly Coupled System/Shared Memory:

- Tasks and/or processors communicate in a highly synchronized fashion
- Communicates through a common global shared memory
- Shared memory system. This doesn't preclude each processor from having its own local memory(cache memory)

Loosely Coupled System/Distributed Memory

- Tasks or processors do not communicate in a synchronized fashion.
- Communicates by message passing packets consisting of an address, the data content, and some error detection code.
- Overhead for data exchange is high
- Distributed memory system

Loosely coupled systems are more efficient when the interaction between tasks is minimal, whereas tightly coupled system can tolerate a higher degree of interaction between tasks.

Shared (Global) Memory

- A Global Memory Space accessible by all processors
- Processors may also have some local memory

Distributed (Local, Message-Passing) Memory

- All memory units are associated with processors
- To retrieve information from another processor's memory a message must be sent there

Uniform Memory

- All processors take the same time to reach all memory locations

Non-uniform (NUMA) Memory

- Memory access is not uniform

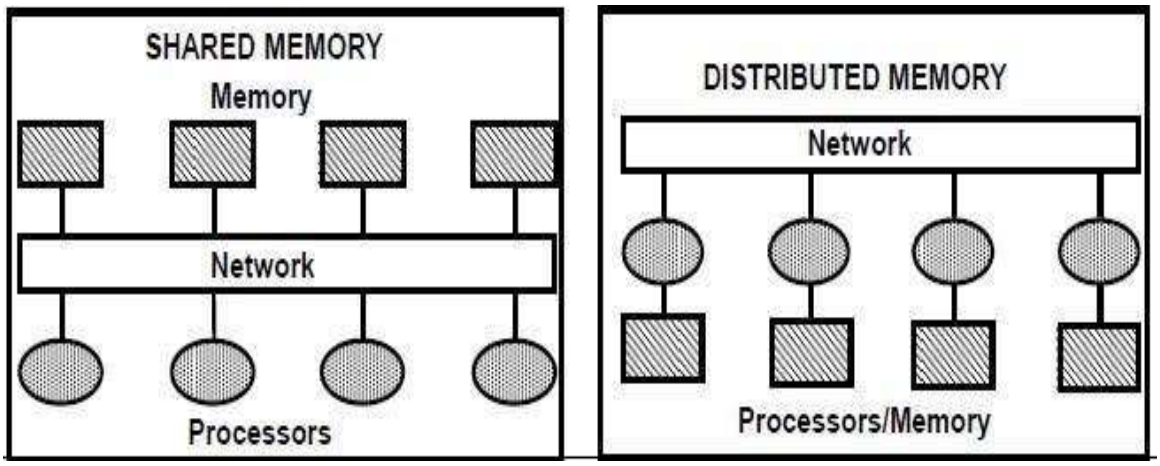


Fig. 5.3 Shared and distributed memory

Shared memory multiprocessor:

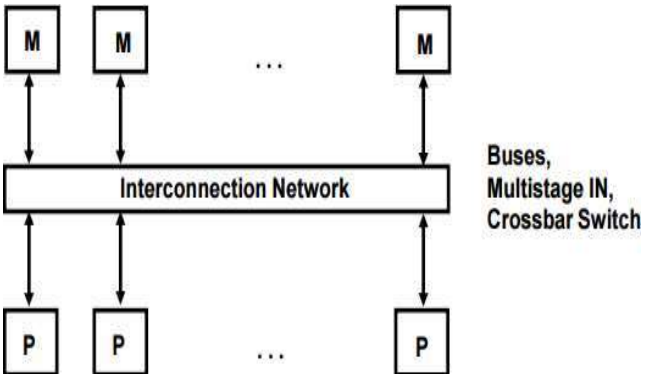


Fig 5.4 Shared memory multiprocessor

Characteristics

- All processors have equally direct access to one large memory address space

Limitations

- Memory access latency; Hot spot problem

5.2 Interconnection Structures:

The interconnection between the components of a multiprocessor System can have different physical configurations depending n the number of transfer paths that are available between the processors and memory in a shared memory system and among the processing elements in a loosely coupled system.

Some of the schemes are as:

- Time-Shared Common Bus
- Multiport Memory
- Crossbar Switch
- Multistage Switching Network
- Hypercube System

a. Time shared common Bus

- All processors (and memory) are connected to a common bus or busses
- Memory access is fairly uniform, but not very scalable
- A collection of signal lines that carry module-to-module communication
- Data highways connecting several digital system elements
- Operations of Bus

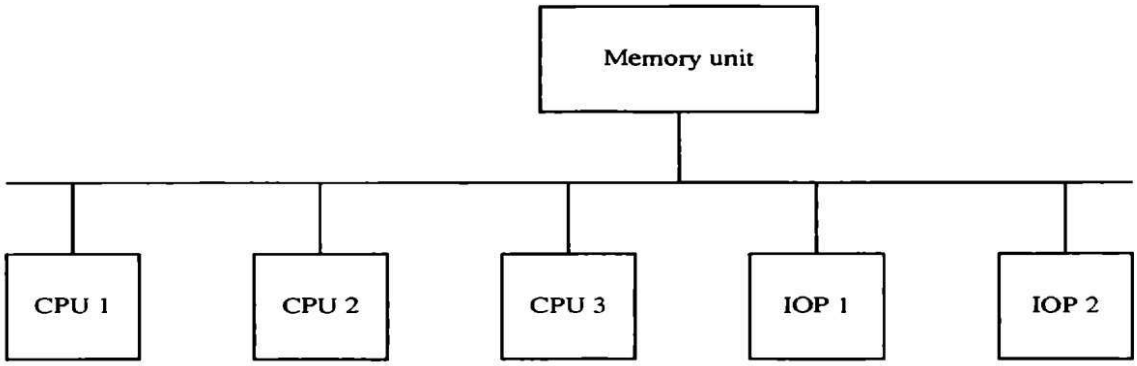


Fig. 5.5 Time shared common bus organization

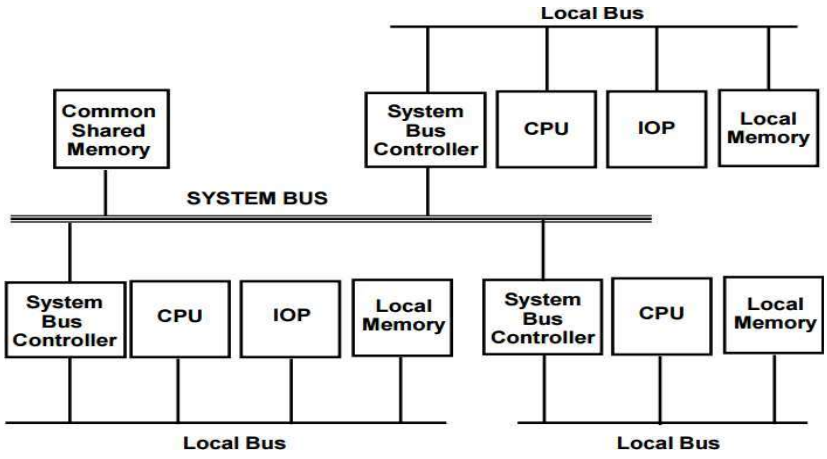


Fig. 5.6 system bus structure for multiprocessor

In the above figure we have number of local buses to its own local memory and to one or more processors. Each local bus may be connected to a CPU, an IOP, or any combinations of processors. A system bus controller links each local bus to a common system bus. The I/O devices connected to the local IOP, as well as the local memory, are available to the local processor. The memory connected to the common system bus is shared by all processors. If an IOP is connected directly to the system bus the I/O devices attached to it may be made available to all processors

Disadvantage.:

- Only one processor can communicate with the memory or another processor at any given time.
- As a consequence, the total overall transfer rate within the system is limited by the speed of the single path

b. Multiport Memory:

Multiport Memory Module

- Each port serves a CPU

Memory Module Control Logic

- Each memory module has control logic
- Resolve memory module conflicts Fixed priority among CPUs

Advantages

- The high transfer rate can be achieved because of the multiple paths.

Disadvantages:

- It requires expensive memory control logic and a large number of cables and connections

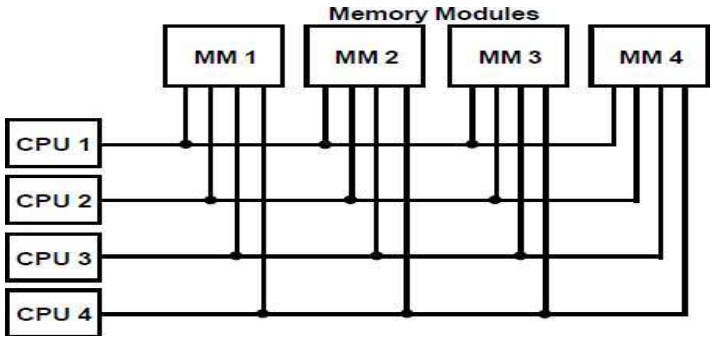
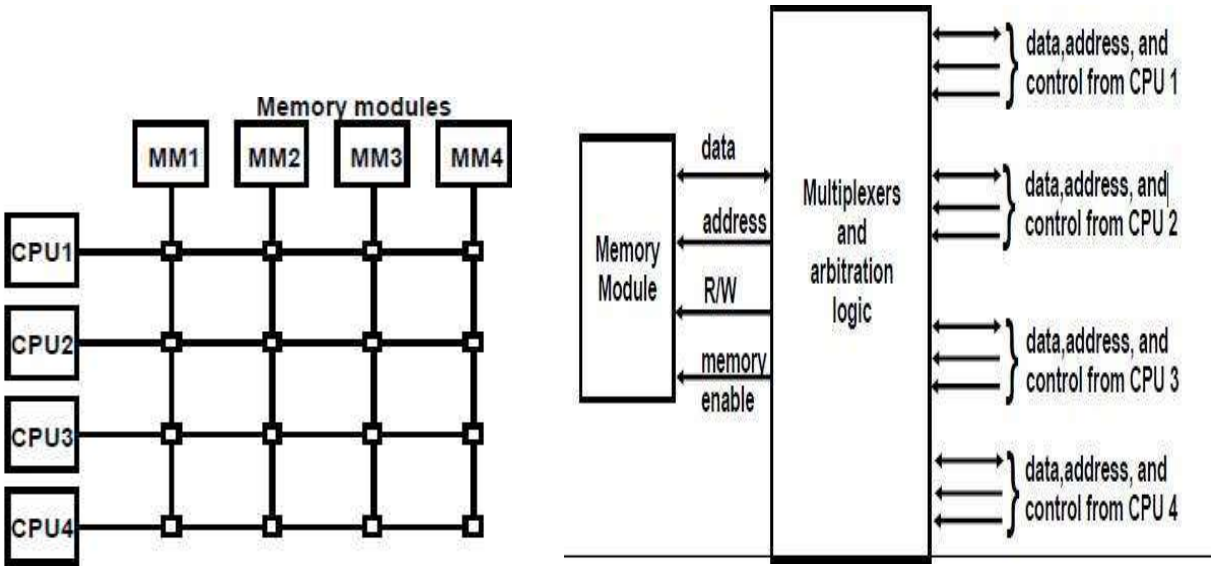


Fig. 5.7 Multiport memory

c. Crossbar switch:

- Each switch point has control logic to set up the transfer path between a processor and a memory.
- It also resolves the multiple requests for access to the same memory on the predetermined priority basis.
- Though this organization supports simultaneous transfers from all memory modules because there is a separate path associated with each Module.
- The H/w required to implement the switch can become quite large and complex



a) b)

Fig. 5.8 a) cross bar switch b) Block diagram of cross bar switch

Advantage:

- Supports simultaneous transfers from all memory modules

Disadvantage:

- The hardware required to implement the switch can become quite large and complex.

d. Multistage Switching Network:

- The basic component of a multi stage switching network is a two-input, two-output interchange switch.

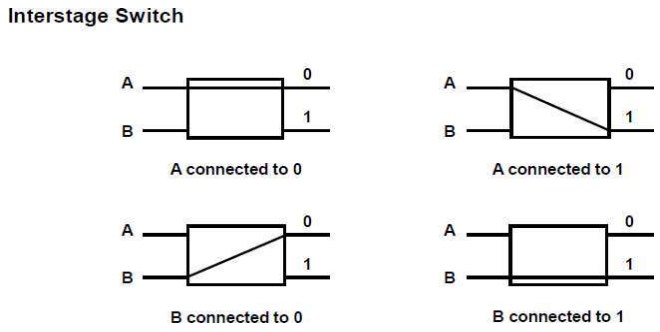


Fig. 5.9 operation of 2X2 interconnection switch

Using the 2x2 switch as a building block, it is possible to build a multistage network to control the communication between a number of sources and destinations.

- To see how this is done, consider the binary tree shown in Fig. below.
- Certain request patterns cannot be satisfied simultaneously.

  i.e., if P1 → 000~011, then P_2 → 100~111

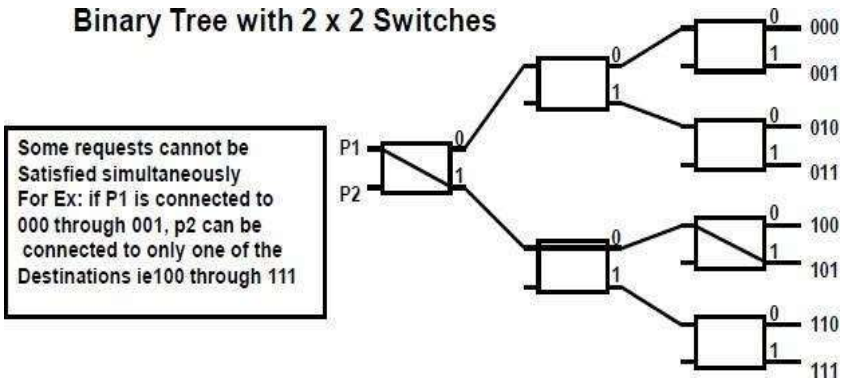


Fig 5.10 Binary tree with 2x2 switches

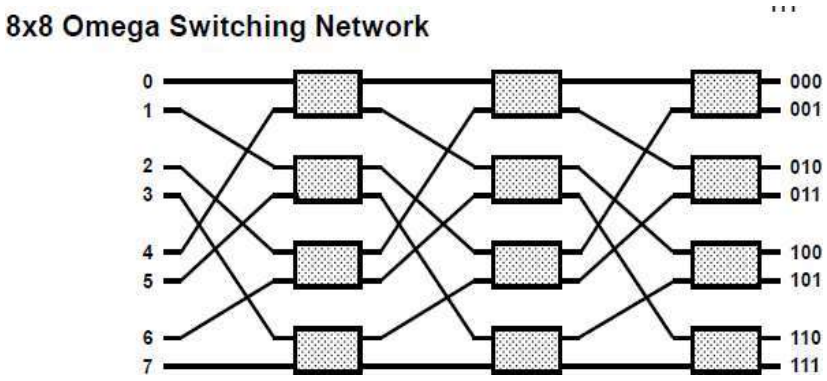


Fig. 5.11 8X8 Omega switching network

- Some request patterns cannot be connected simultaneously. i.e., any two sources cannot be connected simultaneously to destination 000 and 001
- In a tightly coupled multiprocessor system, the source is a processor and the destination is a memory module.
- Set up the path → transfer the address into memory → transfer the data
- In a loosely coupled multiprocessor system, both the source and destination are Processsing elements.

e. Hypercube System:

The hypercube or binary n-cube multiprocessor structure is a loosely coupled system composed of N=2n processors interconnected in an n-dimensional binary cube.

- Each processor forms a node of the cube, in effect it contains not only a CPU but also local memory and I/O interface.
- Each processor address differs from that of each of its n neighbors by exactly one bit position.
- Fig. below shows the hypercube structure for n=1, 2, and 3.
- Routing messages through an *n*-cube structure may take from one to *n* links from a source node to a destination node.
- A routing procedure can be developed by computing the exclusive-OR of the source node address with the destination node address.
- The message is then sent along any one of the axes that the resulting binary value will have 1 bits corresponding to the axes on which the two nodes differ.
- A representative of the hypercube architecture is the Intel iPSC computer complex.
- It consists of 128(*n*=7) microcomputers, each node consists of a CPU, a floating point processor, local memory, and serial communication interface units

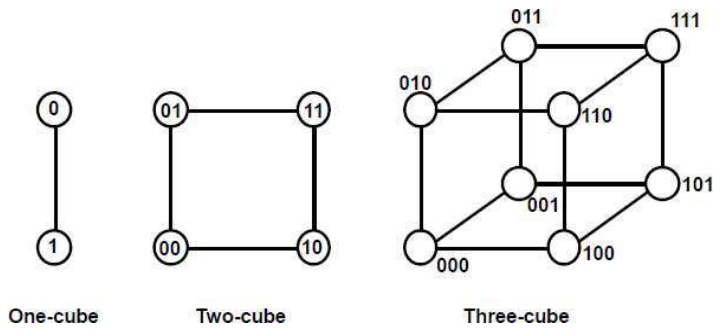


Fig. 5.12 Hypercube structures for n=1,2,3

5.3 Inter-processor Arbitration

- Only one of CPU, IOP, and Memory can be granted to use the bus at a time
- Arbitration mechanism is needed to handle multiple requests to the shared resources to resolve multiple contention
- SYSTEM BUS:
    - A bus that connects the major components such as CPU's, IOP's and memory
    - A typical System bus consists of 100 signal lines divided into three functional groups: data, address and control lines. In addition there are power distribution lines to the components.
- Synchronous Bus
    - Each data item is transferred over a time slice
    - known to both source and destination unit
    - Common clock source or separate clock and synchronization signal is transmitted periodically to synchronize the clocks in the system
- Asynchronous Bus
    - Each data item is transferred by Handshake mechanism
        - Unit that transmits the data transmits a control signal that indicates the presence of data
        - Unit that receiving the data responds with another control signal to acknowledge the receipt of the data

o Strobe pulse -supplied by one of the units to indicate to the other unit when the data transfer has to occur

Table 5.1 IEEE standard 796 multibus signals

	Signal name
Data and address	
Data lines (16 lines)	DATA0–DATA15
Address lines (24 lines)	ADRS0–ADRS23
Data transfer	
Memory read	MRDC
Memory write	MWTC
IO read	IORC
IO write	IOWC
Transfer acknowledge	TACK
Interrupt control	
Interrupt request (8 lines)	INT0–INT7
Interrupt acknowledge	INTA
Miscellaneous control	
Master clock	CCLK
System initialization	INIT
Byte high enable	BHEN
Memory inhibit (2 lines)	INH1–INH2
Bus lock	LOCK
Bus arbitration	
Bus request	BREQ
Common bus request	CBRQ
Bus busy	BUSY
Bus clock	BCLK
Bus priority in	BPRN
Bus priority out	BPRO
Power and ground (20 lines)	

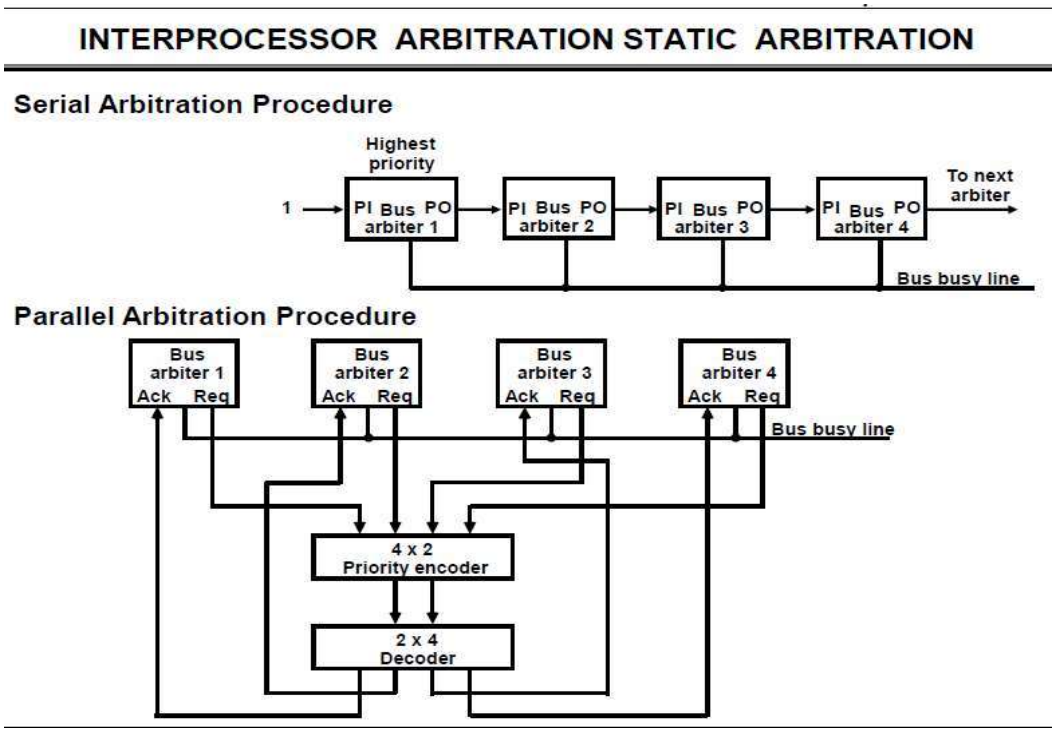


Fig. 5.13 Inter-processor arbitration static arbitration

Interprocessor Arbitration Dynamic Arbitration

- Priorities of the units can be dynamically changeable while the system is in operation
- Time Slice
    - Fixed length time slice is given sequentially to each processor, round-robin fashion
- Polling
    - Unit address polling -Bus controller advances the address to identify the requesting unit. When processor that requires the access recognizes its address, it activates the bus busy line and then accesses the bus. After a number of bus cycles, the polling continues by choosing a different processor.
- LRU
    - The least recently used algorithm gives the highest priority to the requesting device that has not used bus for the longest interval.
- FIFO
    - The first come first serve scheme requests are served in the order received. The bus controller here maintains a queue data structure.
- Rotating Daisy Chain
    - Conventional Daisy Chain -Highest priority to the nearest unit to the bus controller
    - Rotating Daisy Chain –The PO output of the last device is connected to the PI of the first one. Highest priority to the unit that is nearest to the unit that has most recently accessed the bus(it becomes the bus controller)

5.4 Inter processor communication and synchronization:

- The various processors in a multiprocessor system must be provided with a facility for *communicating* with each other.
    - A communication path can be established through *a portion of memory* or *a common input-output channels*.

- The sending processor structures a request, a message, or a procedure, and places it in the memory mailbox.
    - *Status bits* residing in common memory
    - The receiving processor can check the mailbox *periodically*.
    - The response time of this procedure can be time consuming.
- A more efficient procedure is for the sending processor to alert the receiving processor directly by means of an *interrupt signal*.
- In addition to shared memory, a multiprocessor system may have other shared resources.
    - e.g., a magnetic disk storage unit.
- To prevent conflicting use of shared resources by several processors there must be a provision for assigning resources to processors. i.e., operating system.
- There are three organizations that have been used in the design of operating system for multiprocessors: *master-slave configuration*, *separate operating system*, and *distributed operating system*.
- In a master-slave mode, one processor, master, always executes the operating system functions.
- In the separate operating system organization, each processor can execute the operating system routines it needs. This organization is more suitable for *loosely coupled systems*.
- In the distributed operating system organization, the operating system routines are distributed among the available processors. However, each particular operating system function is assigned to only one processor at a time. It is also referred to as a *floating operating system*.

Loosely Coupled System

- There is *no shared memory* for passing information.
- The communication between processors is by means of message passing through *I/O channels*.
- The communication is initiated by one processor calling a *procedure* that resides in the memory of the processor with which it wishes to communicate.

- The communication efficiency of the interprocessor network depends on the *communication routing protocol, processor speed, data link speed, and the topology of the network*.

Interprocess Synchronization

- The instruction set of a multiprocessor contains basic instructions that are used to implement communication and synchronization between cooperating processes.
    - Communication refers to the exchange of data between different processes.
    - Synchronization refers to the special case where the data used to communicate between processors is control information.
- Synchronization is needed to enforce the *correct sequence of processes* and to ensure *mutually exclusive access* to shared writable data.
- Multiprocessor systems usually include various mechanisms to deal with the synchronization of resources.
    - Low-level primitives are implemented directly by the hardware.
    - These primitives are the basic mechanisms that enforce mutual exclusion for more complex mechanisms implemented in software.
    - A number of hardware mechanisms for mutual exclusion have been developed.
        - A binary semaphore

Mutual Exclusion with Semaphore

- A properly functioning multiprocessor system must provide a mechanism that will guarantee orderly access to shared memory and other shared resources.
    - Mutual exclusion: This is necessary to protect data from being changed simultaneously by two or more processors.
    - Critical section: is a program sequence that must complete execution before another processor accesses the same shared resource.
- A *binary variable* called a *semaphore* is often used to indicate whether or not a processor is executing a critical section.

- Testing and setting the semaphore is itself a critical operation and must be performed as a single indivisible operation.
- A semaphore can be initialized by means of a *test and set instruction* in conjunction with a hardware *lock* mechanism.
- The instruction TSL SEM will be executed in two memory cycles (the first to read and the second to write) as follows:

$$R \leftarrow M[SEM],\ M[SEM] \leftarrow 1$$

5.5 Cache Coherence

cache coherence is the consistency of shared resource data that ends up stored in multiple local **caches**. When clients in a system maintain **caches** of a common memory resource, problems may arise with inconsistent data, which is particularly the case with CPUs in a multiprocessing system.

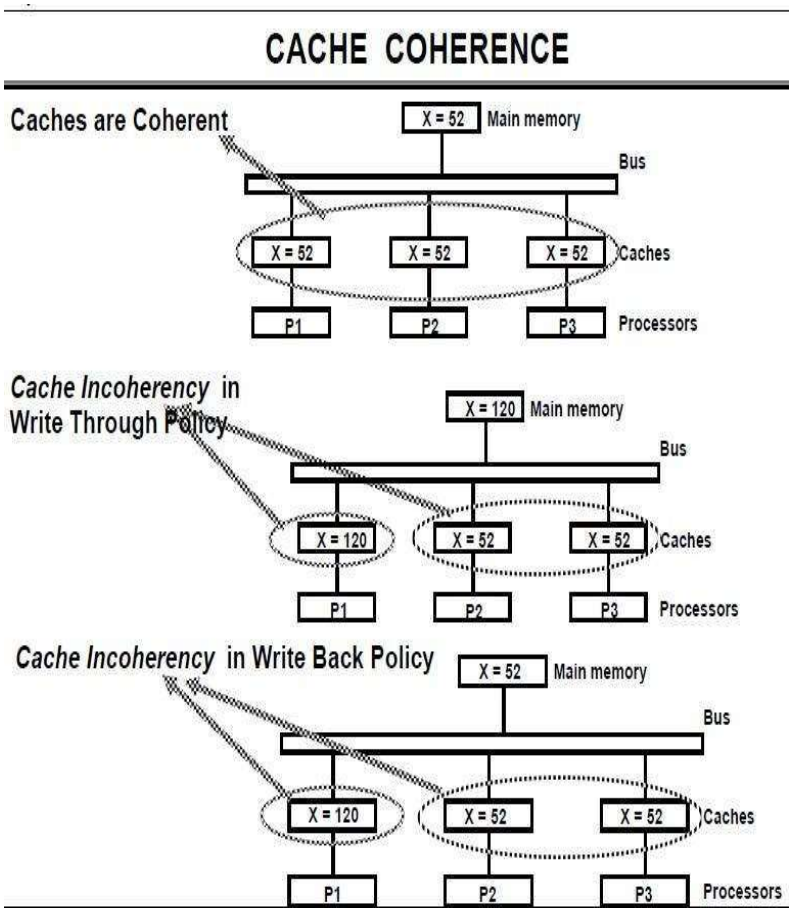


Fig. 5.14 cache coherence

Shared Cache

-Disallow private cache

-Access time delay

Software Approaches

* Read-Only Data are Cacheable
  - Private Cache is for Read-Only data
  - Shared Writable Data are not cacheable
  - Compiler tags data as cacheable and noncacheable
  - Degrade performance due to software overhead

* Centralized Global Table
  - Status of each memory block is maintained in CGT: RO(Read-Only); RW(Read and Write)
  - All caches can have copies of RO blocks
  - Only one cache can have a copy of RW block
  - Hardware Approaches

* Snoopy Cache Controller
  - Cache Controllers monitor all the bus requests from CPUs and IOPs
  - All caches attached to the bus monitor the write operations
  - When a word in a cache is written, memory is also updated (write through)
  - Local snoopy controllers in all other caches check their memory to determine if they have a copy of that word; If they have, that location is marked invalid(future reference to this location causes cache miss)