

# COMPUTER ARCHITECTURE SYLLABUS

**Objective:** To gain knowledge about the architecture of computer and to understand the concepts of CPU, ALU Design, I/O Instruction format and different processors.

**UNIT I Basic Computer Organisation And Design :** Instruction codes - Computer Registers - Computer Instructions - Timing and Control - Instruction Cycle - Control Memory-Address Sequencing (12L)

**UNIT II Central Processing Unit :** General Register Organization – Stack Organization – Instruction Formats – Addressing Modes – Data transfer and manipulation – Program Control. (12L)

**UNIT III Computer Arithmetic :** Hardware Implementation and Algorithm for Addition, Subtraction, Multiplication, Division-Booth Multiplication Algorithm-Floating Point Arithmetic.

**UNIT IV Input Output Organization :** Input – Output Interface – Asynchronous data transfer – Modes of transfer – Priority Interrupt – Direct Memory Access (DMA). (12L)

**Unit V Memory Organisation:** Memory Hierarchy - Main memory - Auxillary memory - Associative memory - Cache memory - Virtual memory. (12L)

**Text Book:** Computer system Architecture - by Morris Mano, Third Edition. P.H.I Private Limited.

## Reference Books:

1. Computer System Architecture P.V.S. Rao PHI
2. Nirmala Sharma, "Computer Architecture", First Edition, 2009, University Science Press
3. Nicholas Carter, "Computer Architecture" , 2006, TMH Publication.

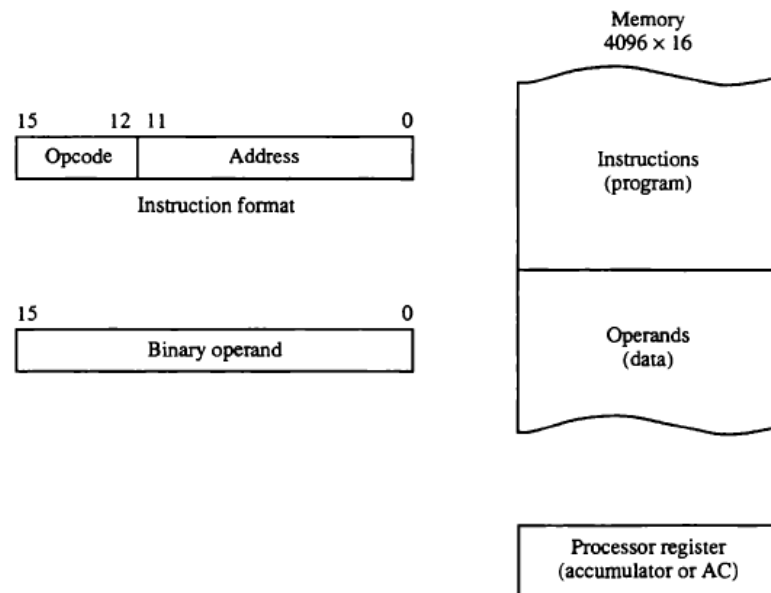
# COMPUTER ARCHITECTURE

## 1. INSTRUCTION CODES

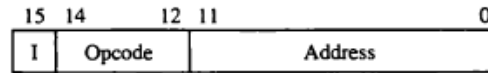
- The organisation of the computer is defined by its internal registers, timing and control structure and the set of instructions that it uses.
- The internal organization of a digital system is defined by the sequence of micro operations it performs on data stored in its registers.
- The general purpose digital computer is capable of executing various micro operations and also can be instructed as to what specific sequence of operations it must perform.
- The user of a computer can control the process by means of a program. A program is a set of instructions that specify the operations, operands and the sequence of the processing.
- A computer instruction is a binary code that specifies a sequence of micro operations for the computer.
- Instruction codes together with data are stored in memory. The computer reads each instruction from memory and places it in a control register.
- The control interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of micro operations.
- Every computer has its own unique instruction set.
- The ability to store and execute instructions, the stored program concept is the most important property of a general purpose computer.
- **Instruction Code:** An instruction code is a group of bits that instructs the computer to perform a specific operation. It is usually divided into parts. The most basic part of an instruction code is its operation part.
- The **operation code** of an instruction is a group of bits that define such operations as add, subtract, multiply, shift and complement.
- The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer. The operation code consists of at least  $n$  bits for a given  $2^n$  distinct operations.
- Eg. For ADD operation, the operation code consists of six bits 110010 when the operation code is decoded in the control unit. The computer issues control signals to read an operand from memory and add the operand to a processor register.
- An operation is part of an instruction stored in computer memory. It is a binary code that tells the computer to perform a specific operation.
- The control unit receives instruction from memory and interprets the operation code bits. It then issues a sequence of control signal to initiate the micro operations in internal computer registers for the hardware implementation. So it is called as macro operations because it specifies a set of micro instructions.
- Instruction code not only specifies the operation but also the registers or the memory words where the operands are to be found and also the register or memory word where the result is stored.

### a) Stored Program Organisation

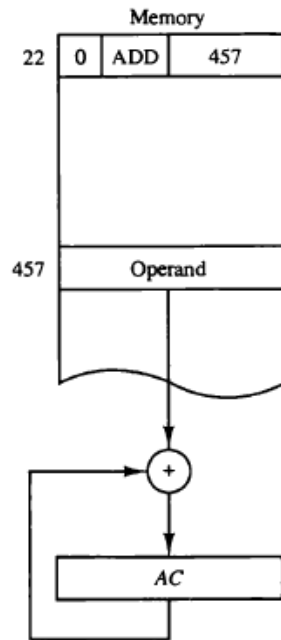
- Simple way to organize a computer is to have one processor register and an instruction code format with two parts. The first part specifies the operation to be performed and the second specifies an address.
- The memory address tells the control where to find an operand in memory. The operand is read from memory and used as the data to be operation on together with the data stored in the processor register.



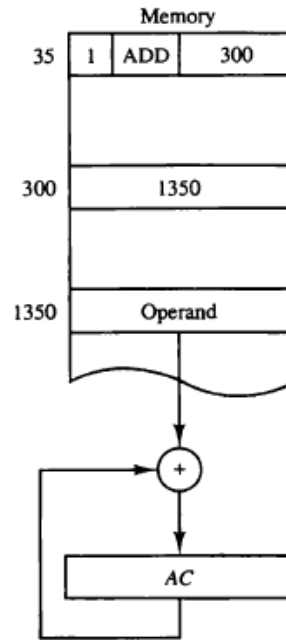
- In the above figure, Instructions are stored in one section of memory and data in another.
- For a memory unit with 4096 words, 12 bits specify an address  $2^{12}=4096$ , if we store each instruction code in one 16 bit memory word, 4 bits for operation code and 12 bits for address of an operand.
- Computers that have a single processor register usually assign to it the name **Accumulator** and label it **AC**. The operation is performed with the memory operand and the content of AC.
- If an operation in an instruction code does not need an operand from memory, rest of the bits can be used for other purposes. Eg. Operations such as Clear AC, complement AC and increment AC operate on data stored in AC register.
- It is convenient to use the address bits of an instruction code not as an address but as **actual operand**.
- When the second part of the instruction code specifies an operand, the instruction is called as **immediate operand**.
- When the second part of an instruction specifies the address of an operand, the instruction is said to have a **direct address**.
- When the second part of an instruction designate an address of a memory word in which the address of the operand is found, this instruction is called as **Indirect Address**.



(a) Instruction format



(b) Direct address



(c) Indirect address

- In the above figure a, consists of a 3 bit operation code, a 12 bit address and an indirect address mode bit designated by I. The mode bit is 0 for **direct address** and 1 for an **indirect address**.
- In figure b, it is placed in address 22 in memory. The I bits is 0, so the instruction is **direct address**. The opcode specified in an ADD instruction and the address part is the binary equivalent of 457. The control finds the operand in memory at address 457 and adds it to the content of AC.
- In figure c, the instruction in address 35 has a mode bit I=1, it is recognized as an **indirect address**. The address part is the binary equivalent of 300. The control goes to address 300 to find the address of the operand which is 1350 is then added to the content of AC. The indirect address instruction needs two references to memory to fetch an operand. First reference is to read the address of the operand and the second is the operand itself.
- **Effective address** is the address of the operand or the target address in a branch type instruction. So effective address of figure b is 457 and figure c is 1350.

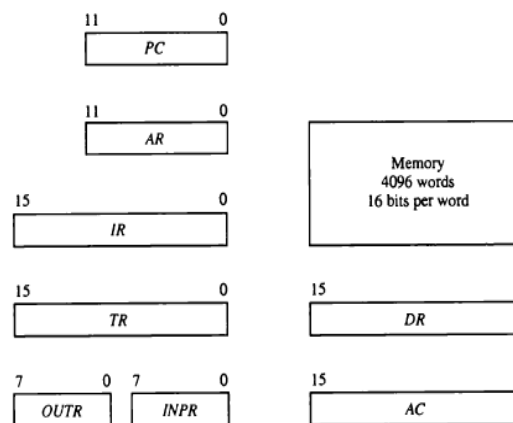
## 2. COMPUTER REGISTERS

- The computer needs processor registers for manipulating data and a register to hold the memory address. The lists of registers for the basic computer are



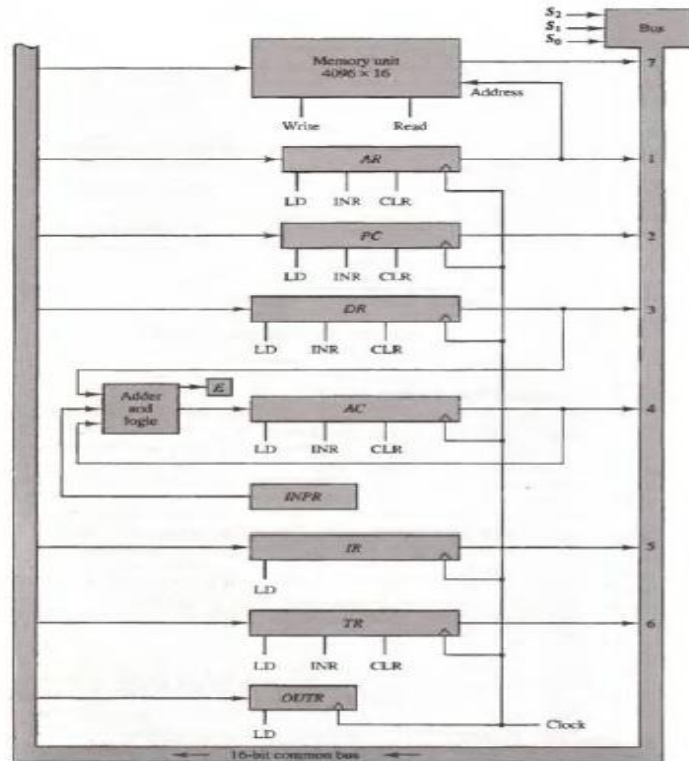
Register symbol	Number of bits	Register name	Function
<i>DR</i>	16	Data register	Holds memory operand
<i>AR</i>	12	Address register	Holds address for memory
<i>AC</i>	16	Accumulator	Processor register
<i>IR</i>	16	Instruction register	Holds instruction code
<i>PC</i>	12	Program counter	Holds address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

- The memory unit has a capacity of 4096 words and each word contains 16 bits. Twelve bits specify the address of an operand. 3 bits for the operation part of the instruction and one bit to specify direct or indirect address.
- Data Register (DR) holds the operand read from memory
- Accumulator register (AC) 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 which is the memory address.
- The program Counter (PC) has 12 bits and it holds the address of the next instruction to be read from memory after the current instruction is executed.
- Two registers used for input and output. The input register(INPR) receives an 8 bit character from an input devices. The output register (OUTR) holds an 8 bit character for an output device.



#### a) Common Bus System

- The basic computer has 8 registers, a memory unit and a control unit. Paths must be provided to transfer information from one register to another and between memory and registers.
- The outputs of 7 registers and memory are connected to the common bus.
- The specific output is selected for the bus lines at any given using the binary value of the selection variables  $S_2, S_1, S_0$ . The output of DR is 3, then the 16 bit outputs of DR on the bus lines  $S_2S_1S_0=011$ .



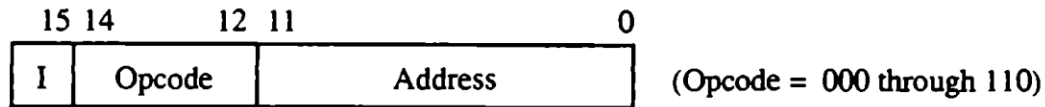
- Four Registers DR, AC, IR and TR have 16 bits each. Two registers AR and PC have 12 bits. The input register INPR and output register OUTR have 8 bits each.
- 16 lines of the common bus receive information from 6 registers and the memory unit. The bus lines are connected to the inputs of 6 registers and memory.
- Five registers have 3 control inputs LD(load), INR(increment) and CLR(clear).
- The input data and output data of the memory are connected to the common bus, but the memory address is connected to AR.
- Any register can receive the data from memory after a read operation except AC.
- The 16 inputs of AC come from an adder and logic circuit.
- One set of 16 bit inputs come from the outputs of AC.
- Second set of 16 bits inputs come from data register DR. They are used for arithmetic and logic micro operations such as ADD DR to AC or AND DR to AC. The result of addition is transferred to AC and end carry out of addition is transferred to flip flop E.
- Third set of 8 bit inputs come from the input register INPR.

Eg.  $DR \leftarrow AC$  and  $AC \leftarrow DR$ .

- This can be done by placing the content of AC on the bus enabling the LD input of Dr, transferring the content of DR through the adder and logic circuit into AC and enabling the LD input of AC, all during the same clock cycle.

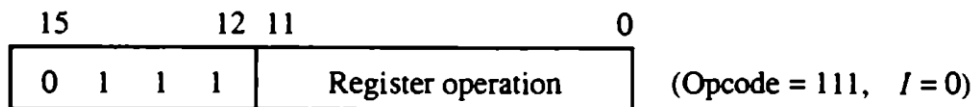
### 3. COMPUTER INSTRUCTIONS

- The basic computer has three instruction code formats. Each format has 16 bits. The operation code part of the instruction contains three bits and the remaining 13 bits depends on operation code.
- A **memory reference instruction** uses 12 bits to specify an address and one bit to specify the addressing mode I. I=0 for direct address and I=1 for indirect address.



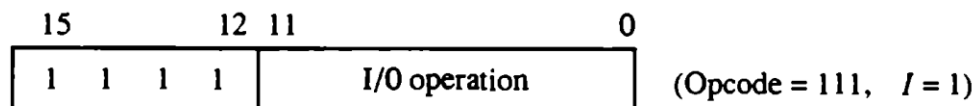
#### (a) Memory – reference instruction

The **register reference instruction** specifies an operation on or a test of the AC register. An operand from memory is not needed. The other 12 bits are used to specify the operation or test to be executed.



#### (b) Register – reference instruction

- Similarly an **Input Output instruction** does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction. Remaining 12 bits are used to specify the type of input-output operation or test performed.



#### (c) Input – output 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 three 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.
- Only three bits of the instruction are used for the operation code.
- The basic computer instructions are listed in table below.

Symbol	Hexadecimal code		Description
	<i>I</i> = 0	<i>I</i> = 1	
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

- The hexadecimal code is equal to the equivalent hexa decimal number of the binary code used for the instruction.
- A memory reference instruction has an address part of 12 bits. The address part is denoted by three xxx's and stand for the 3 hexadecimal digits of 12 bit address. Last bit is designated by symbol I.
- When I=0 the last four bits of an instruction have a hexadecimal digit equivalent from 0 to 6, since the last bit is 0. When I=1. The last four bits of the instruction ranges from 8 to E, since the last bit is 1.

### Instruction Set Completeness

The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories.

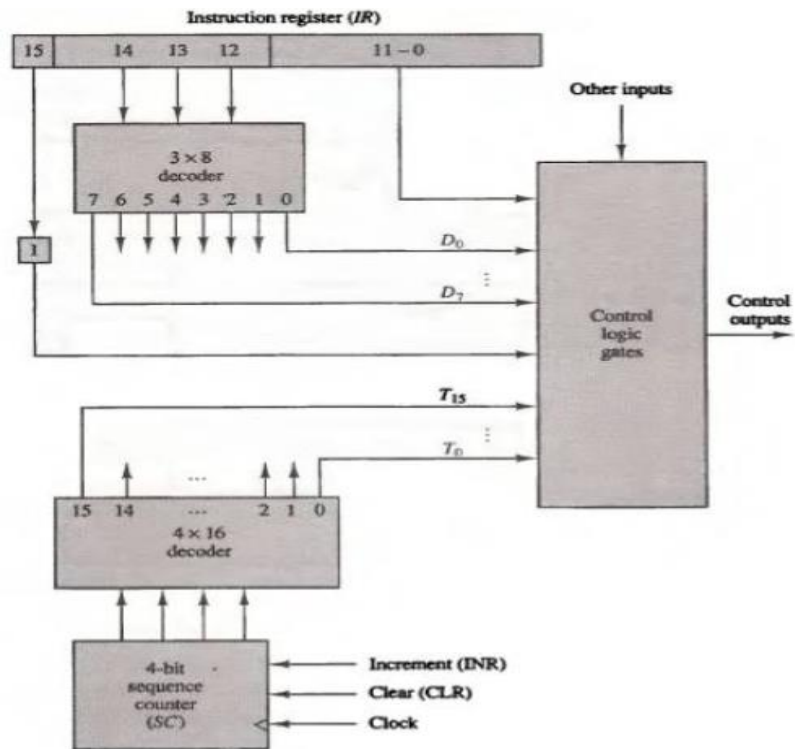
1. Arithmetic, logical and shift instructions
2. Instructions for moving information to and from memory and processor registers
3. Program control instructions together with instructions that check status conditions
4. Input and output instructions.

- There is one arithmetic instruction ADD and two related instructions complement AC(CMA) and increment AC(INC).
- Circulate instructions CIR and CIL used for arithmetic shifts . Multiplication and division can be performed using addition subtraction and shifting.
- There are 3 logic operations AND, Complement AC(CMA) and clear AC(CLA), AND and complement provides a NAND operation.
- Moving information from memory to AC is AC(LDA) instruction
- Storing information from AC into memory is done with the store AC(STA) instruction.
- Branch instructions BUN, BSA and ISZ together with the four skip instructions.
- Input (INP) and Output(OUT) instructions cause information to be transferred between the computer and external devices.

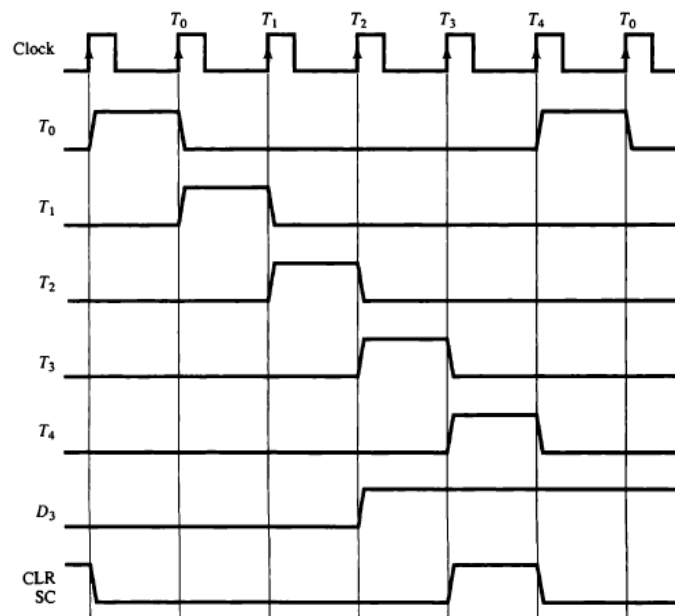
#### 4. TIMING AND CONTROL

- The timing for all registers in the basic computer is controlled by a master clock generator. The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit.
- The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers and micro operations for the accumulator.
- There are two major types control organization: hardwired control and micro programmed control.
- In the **hardwired organization**, control information is stored in flip-flops, decoders and other digital circuits. It required changes in the wiring among the various components if the design has been modified or changed.
- In the **microprogrammed organization**, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of microoperations.
- The block diagram of the control unit is shown in the figure. It consists of two decoders, a sequence counter and a number of control logic gates.
- An instruction read from memory is placed in the instruction register(IR). The position of this register is divided into three parts: the I bit, the operation code and bits 0 through 11.
- The operation code in bits 12 through 14 are decoded with 3x8 decoder. The eight outputs of the decoder are designated by D<sub>0</sub> through D<sub>7</sub>.
- Bit 15 of the instruction is transferred to flipflop designated by the symbol I. Bits 0 through 11 are applied to the control logic gates.
- The 4 bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T<sub>0</sub> through T<sub>15</sub>.
- The sequence counter SC can be incremented or cleared synchronously. The counter is incremented to provide the sequence of timing signals out of the 4x16 decoder.

- Once in a while the counter is cleared to 0, causing the next active timing signal to be  $T_0$ .
- SC is incremented to provide timing signals  $T_0, T_1, T_2, T_3$  and  $T_4$  in sequence. At time  $T_4$ , SC is cleared to 0, if decoder output  $D_3$  is active.  $D_3 T_4; SC \leftarrow 0$ .



- The timing diagram shows the time relationship of the control signals.



- The sequence counter SC responds to the positive transition of the clock. Initially the CLR input of SC is active.
- The first positive clock transition clears SC to 0, which in turn activates the timing signal  $T_0$  out of the decoder.
- $T_0$  is active during one clock cycle. SC is incremented with every positive clock transition signals  $T_0, T_1, T_2, T_3, T_4$  and so on. The timing signals will continue with  $T_5, T_6$  upto  $T_{15}$  and back to  $T_0$ .
- A memory read or write cycle will be initiated with the rising edge of a timing signal.
- For example the register transfer statement.  $T_0: AR \leftarrow PC$  specifies a transfer of the content of PC into AR if timing signal  $T_0$  is active.
- During this time the content of PC is placed onto the bus with  $S_2S_1S_0=010$  and the LD input of AR is enabled.
- The actual transfer does not occur until the end of the clock cycle when the clock goes through a positive transition.
- This same clock transition increments the sequence counter SC from 0000 to 0001, The next clock cycle has  $T_1$  active and  $T_0$  inactive.

## 5. 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 is subdivided into a sequence of sub cycles or phases. The phases are
  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

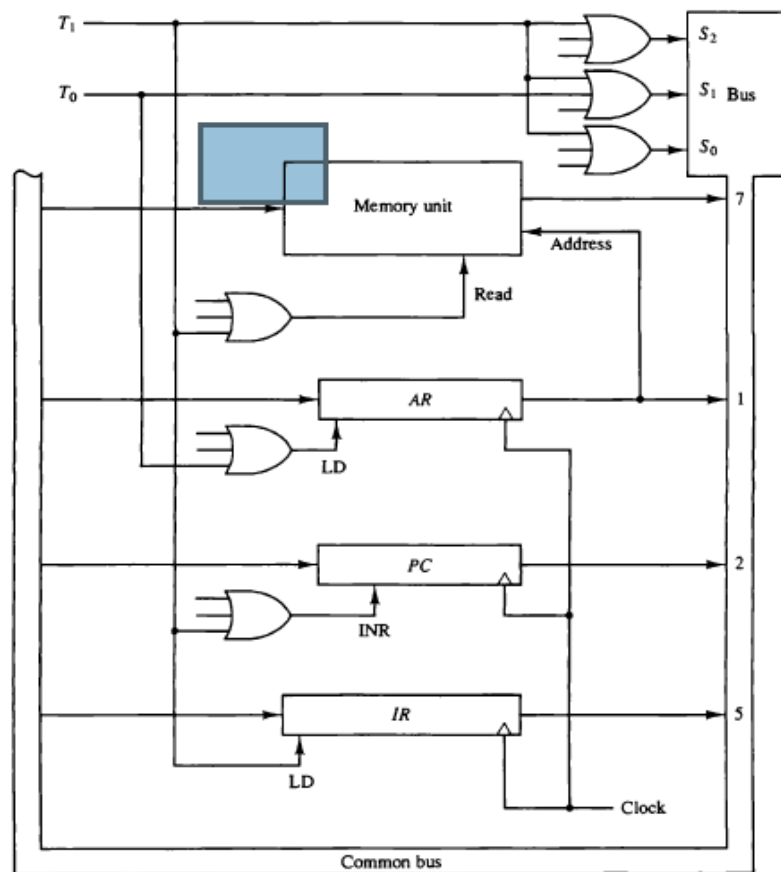
### 1. Fetch and Decode

- Initially the program Counter PC is loaded with the address of the first instruction in the program. SC is cleared to 0 with timing signal  $T_0$ .
- After each clock pulse, SC is incremented by one, so timing signals go through a sequence  $T_0, T_1, T_2$ .

$$\begin{aligned}
 T_0: & AR \leftarrow PC \\
 T_1: & IR \leftarrow M[AR], \quad PC \leftarrow PC + 1 \\
 T_2: & D_0, \dots, D_7 \leftarrow \text{Decode } IR(12-14), \quad AR \leftarrow IR(0-11), \quad I \leftarrow IR(15)
 \end{aligned}$$

- Since AR is connected to the address inputs of memory, it is necessary to transfer the address from PC to AR at  $T_0$ .
- The instruction read from memory is placed in the Instruction Register IR at  $T_1$ . At the same time PC is incremented by one for the next instruction.
- At time  $T_2$  operation code in IR is decoded, the indirect bit is transferred to flipflop I and the address part of the instruction is transferred to AR.
- The figure shows how the first two register transfer statements are implemented in the bus system.

1. Place the content of PC onto the bus by making the bus selection inputs  $S_2S_1S_0$  equal to 010.
2. Transfer the content of the bus to AR by enabling the LD input of AR.



$$T_1: IR \leftarrow M[AR], PC \leftarrow PC + 1$$

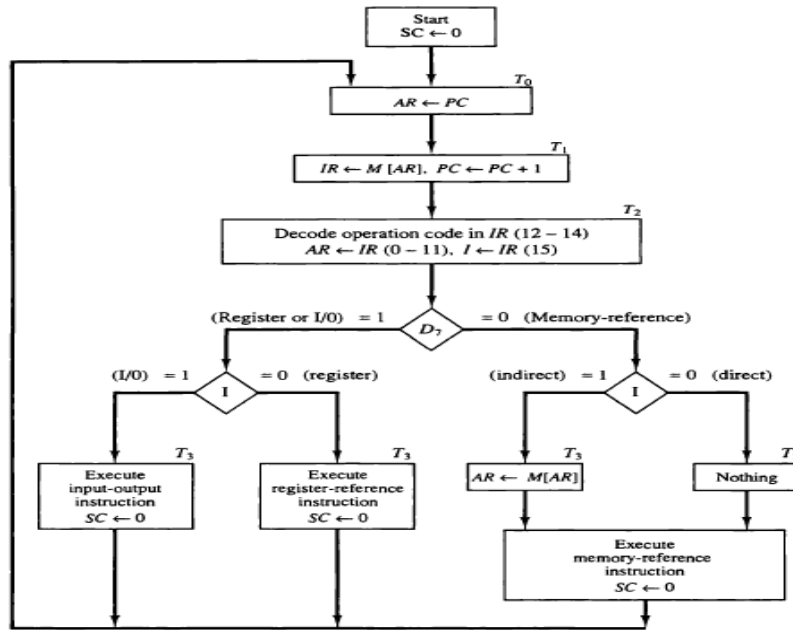
To use the timing signal  $T_1$ , provide the following connections in the bus system.

1. Enable the read input of memory.
2. Place the content of memory on to the bus by making  $S_2S_1S_0=111$
3. Transfer the content of the bus to IR by enabling the LD input of IR.
4. Increment PC by enabling the INR input of PC.

### Determine the type of instruction

- Decoder output  $D_7=1$ , if the operation code is equal to 111, the instruction must be a register reference instruction or input output type.
- IF  $D_7=0$ , the operation code must be one of the seven values 000 to 110 specifying a memory reference instruction. If  $D_7=0$  and  $I=1$  we have a memory reference instruction with an indirect address.





- The micro operation for the indirect address condition is  $AR \leftarrow M[AR]$ . Initially AR holds the address part of the instruction. This address is used for memory read operation.
- The three instruction types are subdivided into four separate paths at timing T3.

$D_7 I T_3$ :  $AR \leftarrow M[AR]$   
 $D_7 I' T_3$ : Nothing  
 $D_7 I' T_3$ : Execute a register-reference instruction  
 $D_7 I T_3$ : Execute an input-output instruction

### Register-Reference Instructions

- This has  $D_7=1$  and  $I=0$ . These instructions use bits 0 through 11 of the instruction code to specify one of 12 instructions. These 12 bits are available in  $IR(0-11)$ .
- The first seven register-reference instructions perform clear, complement, circular shift and increment microoperations on the AC or E registers. the HLT instruction clears a start stop flip flop S and stops the sequence counter from counting.

TABLE 5-3 Execution of Register-Reference Instructions

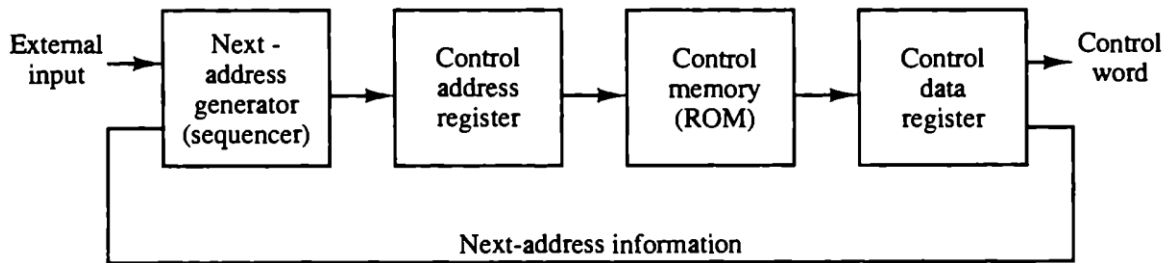
$D_7 I' T_3 = r$ (common to all register-reference instructions) $IR(i) = B_i$ [bit in $IR(0-11)$ that specifies the operation]			
	$r$ :	$SC \leftarrow 0$	Clear SC
CLA	$rB_{11}$ :	$AC \leftarrow 0$	Clear AC
CLE	$rB_{10}$ :	$E \leftarrow 0$	Clear E
CMA	$rB_9$ :	$AC \leftarrow \overline{AC}$	Complement AC
CME	$rB_8$ :	$E \leftarrow \overline{E}$	Complement E
CIR	$rB_7$ :	$AC \leftarrow shr AC, AC(15) \leftarrow E, E \leftarrow AC(0)$	Circulate right
CIL	$rB_6$ :	$AC \leftarrow shl AC, AC(0) \leftarrow E, E \leftarrow AC(15)$	Circulate left
INC	$rB_5$ :	$AC \leftarrow AC + 1$	Increment AC
SPA	$rB_4$ :	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	$rB_3$ :	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	$rB_2$ :	If $(AC = 0)$ then $PC \leftarrow PC + 1$	Skip if AC zero
SZE	$rB_1$ :	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	$rB_0$ :	$S \leftarrow 0$ (S is a start-stop flip-flop)	Halt computer

## 6. Control Memory

- The function of the control unit in a digital computer is to initiate sequences of microoperations.
- The number of different types of microoperations that are available in a given system is finite.
- The control function that specifies a microoperation is a binary variable.
- When it is in one binary state the corresponding binary operation is executed.
- The activate state of a control variable may be either the 1 state or the 0 state.
- The control unit initiates a series of sequential steps of microoperations.
- The control variable at any given time can be represented by a string of 1s and 0s called a control word.
- The control unit initiates a series of sequential steps of microoperations.
- The control words can be programmed to perform various operations on the components of the system.
- Each word in control memory contains within a microinstruction.
- The microinstruction specifies one or more microoperations for the system. \
- A sequence of microinstructions constitutes a microprogram.
- Control memory can be a read only memory(ROM)
- The content of the words in ROM are fixed and cannot be altered by simple programming.
  - ROM words are made permanent during the hardware production of the unit.
  - Dynamic microprogramming permits a microprogram to be loaded initially from an auxiliary memory such as a magnetic disk.
- A memory that is part of a control unit is referred to as a control memory.
- A microprogrammed control unit will have two separate memories: a main memory and a control unit.
- The main memory is available to the user for storing the programs. The contents of main memory may alter when the data are manipulated.
- The control memory holds a fixed microprogram and cannot be altered by user.
- Each microprogram consists of microinstructions that specify various internal control signals for execution of register microoperations.
- These microinstructions generate the microoperations to fetch the instruction from main memory; to evaluate the effective address to execute the operations specified by the instruction. The block diagram of the microprogrammed control organisation is shown below.

- The control address register holds the address of the next microinstruction and the control data

**Figure 7-1** Microprogrammed control organization.



- Register holds the microinstruction read from memory. The microinstruction contains a control word that specifies one or more microoperations for the data processor. Once these operations are executed, the control must determine the next address.
- The next address generator is called a microprogram sequencer as it determines the address sequence that is read from control memory.
- The control data register holds the present microinstruction while the next address is computed and read from memory. The data register is called a pipeline register.

## 7. ADDRESS SEQUENCING

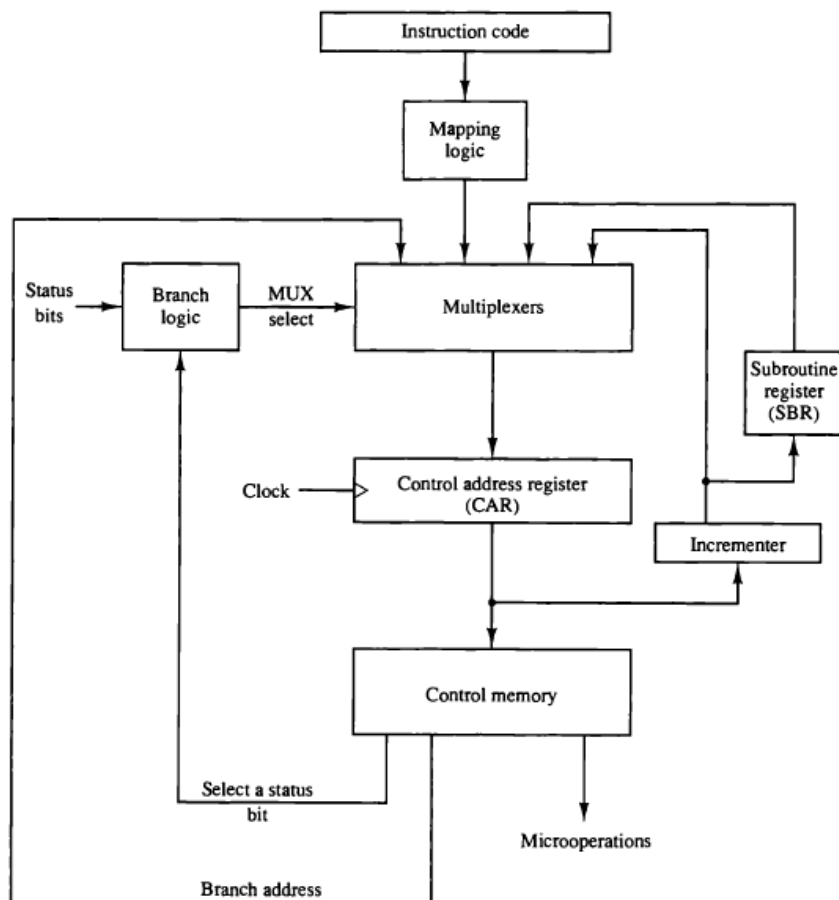
- Microinstructions are stored in control memory in groups.
- Each computer instruction has its own microprogram routine in control memory to generate the microoperations that execute the instruction
- The hardware that controls the address sequencing of the control memory must be capable of sequencing the microinstructions within a routine and be able to branch from one routine to another.
- An initial address is loaded into the control address register when power is turned on in the computer.
- The fetch routine may be sequenced by incrementing the control address register through the rest of its microinstructions.
- At the end of the fetch routine, the instruction is in the instruction register of the computer.
- The next step is to generate the microoperations that execute the instruction fetched from memory.
- The microoperation steps to be generated in processor registers depend on the operation code part of the instruction.
- The mapping procedure is a rule that transforms the instruction code into a control memory address.
- Once the required routine is reached, the microinstructions that execute the instruction may be sequenced by incrementing the control address register.

- Microprogram that employ subroutines require an external register for storing the return address.
- The address sequencing capabilities required in control memory are

1. Incrementing the control address register
2. Unconditional branch or conditional branch depending on status bit conditions
3. A mapping process from the bits of the instruction to an address fro control memory
4. A facility for subroutine call and return.

### Conditional Branching:

- The branch logic provides decision making capabilities in the control unit.
- The status conditions are special bits in the systems that provide parameter information such as carry-out of an adder, sign bit of a number, mode bits, input output status conditions.
- The branch logic is test the specified condition and branch to the indicated address if the condition is met, otherwise the address register is incremented.



- Three bits in the microinstruction are used to specify any one of eight status bit conditions.

- ### Mapping of instruction:

## UNIT - 2

**Central Processing Unit :** General Register Organization – Stack Organization – Instruction Formats – Addressing Modes – Data transfer and manipulation – Program Control. (12L)

### 1. GENERAL REGISTER ORGANISATION

- 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 for various microoperations.
- The bus organization for seven CPU registers is shown in figure. The output of each register is connected to two multiplexers 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.

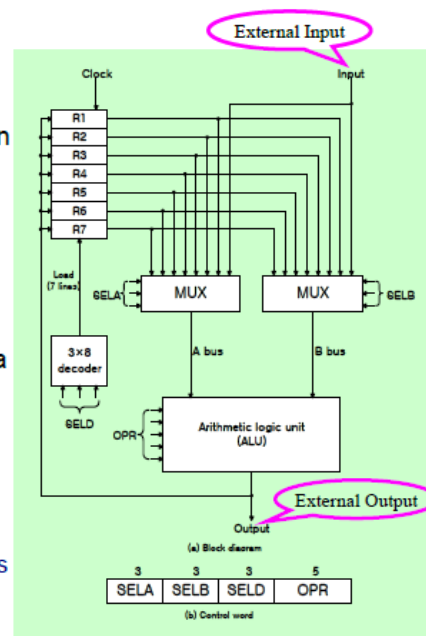
#### General Register Organization

##### Register

- Memory locations are needed for storing *pointers, counters, return address, temporary results, and partial products* during multiplication (in the programming examples of Chap. 6)
- Memory access is the most time-consuming operation in a computer
- More convenient and efficient way is to store intermediate values in processor registers

##### Bus organization for 7 CPU registers : Fig. 8-2

- **2 MUX** : select one of 7 register or external data input by **SELA** and **SELB**
- **BUS A and BUS B** : form the inputs to a common ALU
- **ALU** : **OPR** determine the arithmetic or logic microoperation
  - » The result of the microoperation is available for external data output and also goes into the inputs of all the registers
- **3 X 8 Decoder** : select the register (by **SELD**) that receives the information from ALU



- The operation selected in the ALU determines the arithmetic or logic microoperation that is 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 3x8 decoder.

- 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 eg.  $R1 \leftarrow R2 + R3$**

1. MUX A selector (SELA) to place the content of R2 into bus A.
  2. MUX B selector (SELB) to place the content of R3 into bus B.
  3. ALU operation selector (OPR) to provide arithmetic addition  $A+B$
  4. Decoder destination selector (SELD) to transfer the content of the output bus into R1.
- The four control selection variables are generated in the control unit must be available at the beginning of a clock cycle.
  - The data from the two source registers propagate through the gates in the multiplexers and the ALU to output bus and into the inputs of the destination register all during the clock cycle interval.
  - **Control word:** There are 14 selection inputs in the unit and their combined values specify a control word.
  - Three fields contain three bits each and one field five bits.
  - The three bits of SELA select a source register for the A input of ALU.
  - The three bits of SELB select a register for the B input of the ALU.
  - Three bits of SELD select a destination register using the decoder

◆ **Binary selector input** |  $R1 \leftarrow R2 + R3$

- 1) MUX A selector (**SELA**) : to place the content of R2 into BUS A
- 2) MUX B selector (**SELB**) : to place the content of R3 into BUS B
- 3) ALU operation selector (**OPR**) : to provide the arithmetic addition  $R2 + R3$
- 4) Decoder selector (**SELD**) : to transfer the content of the output bus into R1

◆ **Control Word**

- 14 bit control word (4 fields) : **Fig. 8-2(b)**
  - » **SELA (3 bits)** : select a source register for the A input of the ALU
  - » **SELB (3 bits)** : select a source register for the B input of the ALU
  - » **SELD (3 bits)** : select a destination register using the 3 X 8 decoder
  - » **OPR (5 bits)** : select one of the operations in the ALU
- Encoding of Register Selection Fields : **Tab. 8-1**
  - » **SELA or SELB = 000 (Input)** : MUX selects the external input data
  - » **SELD = 000 (None)** : no destination register is selected but the contents of the output bus are available in the external output

- Encoding of ALU Operation (OPR) : **Tab. 8-2**

◆ **Examples of Microoperations** : **Tab. 8-3**

- TSFA (Transfer A) :  $R7 \leftarrow R1$ , External Output  $\leftarrow R2$ , External Output  $\leftarrow$  External Input
- XOR :  $R5 \leftarrow 0$  ( $XOR\ R5 \oplus R5$ )

**TABLE 8-1** 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 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 bus are available in the external output.
- ALU provides arithmetic and logic operations and CPU must provide shift operations.
- The OPR field has five bits and each operation is designated with a symbolic name.

**TABLE 8-2** Encoding of ALU Operations

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



TABLE 8-3 Examples of Microoperations for the CPU

Microoperation	Symbolic Designation				Control Word
	SELA	SELB	SELD	OPR	
$R1 \leftarrow R2 - R3$	R2	R3	R1	SUB	010 011 001 00101
$R4 \leftarrow R4 \vee R5$	R4	R5	R4	OR	100 101 100 01010
$R6 \leftarrow R6 + 1$	R6	—	R6	INCA	110 000 110 00001
$R7 \leftarrow R1$	R1	—	R7	TSFA	001 000 111 00000
$\text{Output} \leftarrow R2$	R2	—	None	TSFA	010 000 000 00000
$\text{Output} \leftarrow \text{Input}$	Input	—	None	TSFA	000 000 000 00000
$R4 \leftarrow \text{shl } R4$	R4	—	R4	SHLA	100 000 100 11000
$R5 \leftarrow 0$	R5	R5	R5	XOR	101 101 101 01100

## 2. STACK ORGANISATION:

A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved. The stack in digital computer is a memory unit with an address register that can count only after an initial value is loaded into it. 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 the stack are the insertion(push) and deletion(pop) of items.

1. Register Stack,

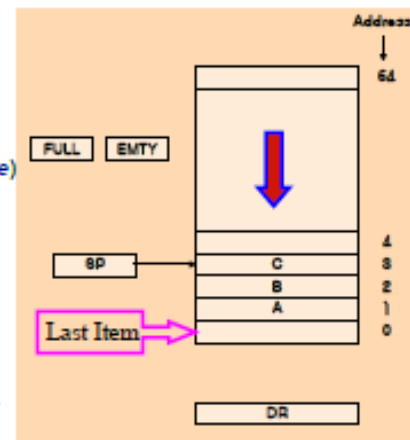
2. Memory Stack

### ◆ Stack or LIFO (Last-In, First-Out)

- A storage device that stores information
  - » The item stored last is the first item retrieved = a stack of tray
- Stack Pointer (SP)
  - » The register that holds the address for the stack
  - » SP always points at the top item in the stack
- Two Operations of a stack : *Insertion and Deletion of Items*
  - » PUSH : Push-Down = Insertion
  - » POP : Pop-Up = Deletion
- Stack의 종류
  - » 1) Register Stack (*Stack Depth가 제한*)
    - a finite number of memory words or register (*stand alone*)
  - » 2) Memory Stack (*Stack Depth가 유동적*)
    - a portion of a large memory

### ◆ Register Stack : Fig. 8-3

- PUSH :  $SP \leftarrow SP + 1$  : Increment SP
- $M[SP] \leftarrow DR$  : Write to the stack
- If ( $SP = 0$ ) then ( $FULL \leftarrow 1$ ) : Check if stack is full
- $EMPTY \leftarrow 0$  : Mark not empty



### **1. 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.
- 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 . 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.
- In a 64-word stack,  $2^6=64$ , the stack pointer contains 6 bits because it has only six bits it cannot exceed a number greater than 63.
- When 63 is incremented by 1, the result is 0.
- 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.

PUSH operation and POP operation are given in the figures.

### **2. Memory Stack:**

- A portion of the 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 three registers are connected to a common address bus and either one can provide 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.
- The push and pop operation is shown in figure below.

Also the stack limits is given in the figure below.

» The first item is stored at **address 1**, and the last item is stored at **address 0**

- **POP** :  $DR \leftarrow M[SP]$  : Read item from the top of stack  
 $SP \leftarrow SP - 1$  : Decrement Stack Pointer  
 If  $(SP = 0)$  then  $(EMPTY \leftarrow 1)$  : Check if stack is empty  
 $FULL \leftarrow 0$  : Mark not full

\* Memory Stack  
 PUSH = Address 감소  
 \* Register Stack  
 PUSH = Address 증가

#### ◆ Memory Stack : Fig. 8-4

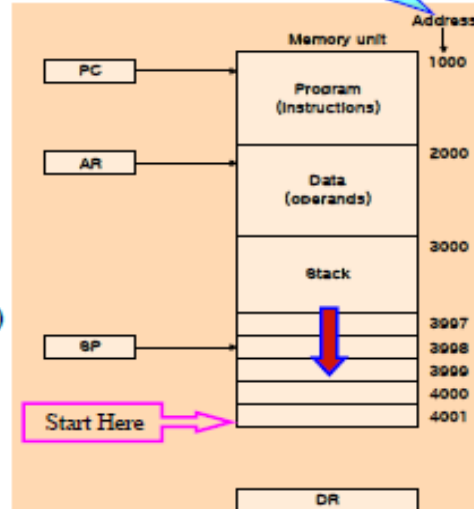
- **PUSH** :  $SP \leftarrow SP - 1$   
 $M[SP] \leftarrow DR$
- » The first item is stored at **address 4000**

- **POP** :  $DR \leftarrow M[SP]$   
 $SP \leftarrow SP + 1$

#### ◆ Stack Limits

- Check for stack overflow(**full**)/underflow(**empty**)
  - » Checked by using two register
    - Upper Limit and Lower Limit Register
  - » After PUSH Operation
    - SP compared with the upper limit register
  - » After POP Operation
    - SP compared with the lower limit register

\* Error Condition  
 PUSH when FULL = 1  
 POP when EMPTY = 1

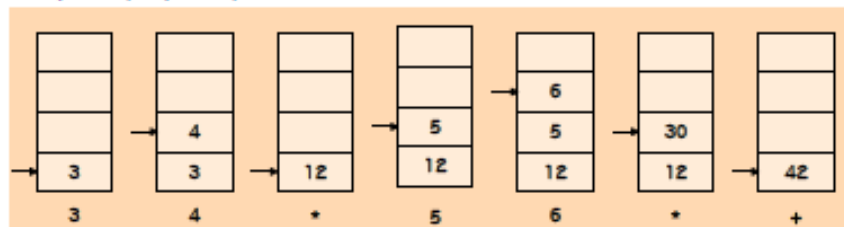


### 3. Reverse Polish Notation(RPN)

- Stack organisation is very effective for evaluating arithmetic expressions.
- Common arithmetic expressions are written in infix notation with each operator written between the operands.  $A*B+C*D$
- Polish mathematician Lukasiewicz showed that arithmetic expressions represented in prefix notation is referred to as polish notation.
- Postfix notation is referred to as Reverse Polish Notation(RPN)  $AB*CD*+$

#### ◆ RPN (Reverse Polish Notation)

- The common mathematical method of writing arithmetic expressions imposes difficulties when evaluated by a computer
- A stack organization is very effective for evaluating arithmetic expressions
- 예제)  $A * B + C * D \rightarrow AB * CD * +$  : Fig. 8-5  
 »  $(3 * 4) + (5 * 6) \rightarrow 34 * 56 * +$



The procedure for RPN is

- Scan the expression from left to right.

- When an operator is reached, perform the operation with two operands found on the left side of the operator.
- Remove the two operands and the operator and replace them by the number obtained from the result of the operation.
- Continue to scan the expression and repeat the procedure for every operator until there are no more operators.

### 3. INSTRUCTION FORMATS

A Computer will have a variety of instruction code formats. The bits of the instruction are divided into groups called fields.

The most common fields found in instruction formats are

#### ◆ Fields in Instruction Formats

- 1) **Operation Code Field** : specify the operation to be performed
- 2) **Address Field** : designate a memory address or a processor register
- 3) **Mode Field** : specify the operand or the effective address (*Addressing Mode*)

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

Computers may have instructions of several different lengths containing varying number of addresses. Most computers fall into three types of CPU organizations:

#### ◆ 3 types of CPU organizations

- 1) Single AC Org. : **ADD X**      $AC \leftarrow AC + M[X]$
- 2) General Register Org. : **ADD R1, R2, R3**      $R1 \leftarrow R2 + R3$
- 3) Stack Org. : **PUSH X**      $TOS \leftarrow M[X]$

X = Operand Address

#### ◆ The influence of the number of addresses on computer instruction

$$X = (A + B) * (C + D)$$

- 4 arithmetic operations : ADD, SUB, MUL, DIV
- 1 transfer operation to and from *memory* and *general register* : MOV
- 2 transfer operation to and from *memory* and *AC register* : STORE, LOAD
- Operand memory addresses : A, B, C, D
- Result memory address : X



Instructions may have three address, two address, one address, zero address and RISC instructions.

### 1. Three-Address Instructions:

Computers with three address instruction formats can use each address field to specify either a processor register or a memory operand.

#### • 1) Three-Address Instruction

ADD	R1, A, B	$R1 \leftarrow M[A] + M[B]$
ADD	R2, C, D	$R2 \leftarrow M[C] + M[D]$
MUL	X, R1, R2	$M[X] \leftarrow R1 * R2$

- » Each address fields specify either a processor register or a memory operand
- »  Short program
- »  Require too many bit to specify 3 address

2. **Two-address instructions:** are common in commercial computers.

Eg. To evaluate  $X=(A+B)*(C+D)$  is as follows:

#### • 2) Two-Address Instruction

MOV	R1, A	$R1 \leftarrow M[A]$
ADD	R1, B	$R1 \leftarrow R1 + M[B]$
MOV	R2, C	$R2 \leftarrow M[C]$
ADD	R2, D	$R2 \leftarrow R2 + M[D]$
MUL	R1, R2	$R1 \leftarrow R1 * R2$
MOV	X, R1	$M[X] \leftarrow R1$

- » The most common in commercial computers
- » Each address fields specify either a processor register or a memory operand

3. **One-Address instruction:** use an implied Accumulator Register(AC) for all data manipulation. The program to evaluate  $X=(A+B)*(C+D)$  is as follows:

LOAD	A	$AC \leftarrow M[A]$
ADD	B	$AC \leftarrow AC + M[B]$
STORE	T	$M[T] \leftarrow AC$
LOAD	C	$AC \leftarrow M[C]$
ADD	D	$AC \leftarrow AC + M[D]$
MUL	T	$AC \leftarrow AC * M[T]$
STORE	X	$M[X] \leftarrow AC$

- » All operations are done between the AC register and memory operand

4. **Zero-Address Instruction:** A stack organized computer does not use an address field for the instruction Add and MUL. The PUSH and POP instructions need an address to specify the

operand that communicates with the stack. The program to evaluate  $X=(A+B)*(C+D)$  is as follows:

<b>PUSH</b>	<b>A</b>	$TOS \leftarrow A$
<b>PUSH</b>	<b>B</b>	$TOS \leftarrow B$
<b>ADD</b>		$TOS \leftarrow (A + B)$
<b>PUSH</b>	<b>C</b>	$TOS \leftarrow C$
<b>PUSH</b>	<b>D</b>	$TOS \leftarrow D$
<b>ADD</b>		$TOS \leftarrow (C + D)$
<b>MUL</b>		$TOS \leftarrow (C + D) * (A + B)$
<b>POP</b>	<b>X</b>	$M[X] \leftarrow TOS$

- » Stack-organized computer does not use an address field for the instructions **ADD**, and **MUL**
- » **PUSH**, and **POP** instructions need an address field to specify the operand
- » Zero-Address : absence of address ( **ADD**, **MUL** )

4. **RISC instructions:** The advantages of a reduced instruction set computer architecture is restricted to the use of load and store instructions when communicating within the registers of the CPU without referring to memory.

- Only use **LOAD** and **STORE** instruction when communicating between memory and CPU
- All other instructions are executed within the registers of the CPU without referring to memory
- Program to evaluate  $X = (A + B) * (C + D)$

<b>LOAD</b>	<b>R1, A</b>	$R1 \leftarrow M[A]$
<b>LOAD</b>	<b>R2, B</b>	$R2 \leftarrow M[B]$
<b>LOAD</b>	<b>R3, C</b>	$R3 \leftarrow M[C]$
<b>LOAD</b>	<b>R4, D</b>	$R4 \leftarrow M[D]$
<b>ADD</b>	<b>R1, R1, R2</b>	$R1 \leftarrow R1 + R2$
<b>ADD</b>	<b>R3, R3, R4</b>	$R3 \leftarrow R3 + R4$
<b>MUL</b>	<b>R1, R1, R3</b>	$R1 \leftarrow R1 * R3$
<b>STORE</b>	<b>X, R1</b>	$M[X] \leftarrow R1$

#### 4. ADDRESSING MODES

The way the operands are chosen during program execution is dependent on the addressing mode of the instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced. Computers use addressing mode technique for the purpose of the following provisions:

### ◆ Addressing Mode

- 1) To give programming versatility to the user
  - » pointers to memory, counters for loop control, indexing of data, ....
- 2) To reduce the number of bits in the addressing field of the instruction

The control unit of a computer is designed to go through an instruction cycle that is divided into three major phases:

### ◆ Instruction Cycle

- 1) Fetch the instruction from memory and  $PC + 1$
- 2) Decode the instruction
- 3) Execute the instruction



There is one register in the computer called the program counter or PC that has the following functions.

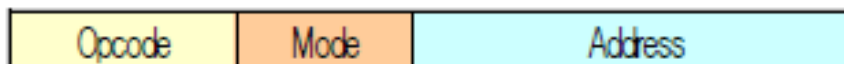
### ◆ Program Counter ( $PC$ )

- PC keeps track of the instructions in the program stored in memory
- PC holds the address of the instruction to be executed next
- PC is incremented each time an instruction is fetched from memory

The addressing mode is shown in figure. The operation code specifies the operation to be performed. The mode field is used to locate the operands needed for the operation. There may or may not be an address field in the instruction. It may designate memory address or a processor register.

### ◆ Addressing Mode of the Instruction

- 1) Distinct Binary Code
    - » Instruction Format 에 Opcode 와 같이 별도로 Addressing Mode Field
  - 2) Single Binary Code
    - » Instruction Format에 Opcode 와 Addressing Mode Field가 섞여 있음
- ◆ Instruction Format with mode field : Fig. 8-6



There are two modes that need no address field at all. These are the implied and immediate modes.




### 1. Implied Mode:

- Operands are specified implicitly in definition of the instruction
- *Examples*
  - » **COM** : Complement Accumulator
    - Operand in AC is implied in the definition of the instruction
  - » **PUSH** : Stack push
    - Operand is implied to be on top of the stack


### 2. Immediate Mode:

- Operand field contains the actual operand
- Useful for initializing registers to a constant value
- *Example* : LD #NBR

### 3. Register Mode:

- Operands are in registers
- Register is selected from a register field in the instruction
  - » k-bit register field can specify any one of  $2^k$  registers
- *Example* : LD R1 AC  $\leftarrow$  R1 

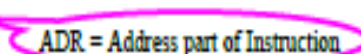
### 4. Register Indirect Mode:

- Selected register contains the address of the operand rather than the operand itself
-  Address field of the instruction uses fewer bits to select a memory address
  - » Register select 하는 것이 bit 수가 적게 소요됨
- *Example* : LD (R1) AC  $\leftarrow$  M[R1]

### 5. Autoincrement or Autodecrement Mode:

- Similar to the register indirect mode except that
  - » the register is *incremented after* its value is used to access memory
  - » the register is *decrement before* its value is used to access memory

### 6. Direct Address Mode

- Effective address is equal to the address field of the instruction (Operand)
- Address field specifies the actual branch address in a branch-type instruction
- *Example* : LD ADR AC  $\leftarrow$  M[ADR] 



## 7. Indirect Addressing Mode

- Address field of instruction gives the address where the effective address is stored in memory
- *Example* : LD @ADR  $AC \leftarrow M[M[ADR]]$

Effective address = address part of the instruction + content of CPU register



## 8. Relative Addressing Mode

- PC is added to the address part of the instruction to obtain the effective address
- *Example* : LD \$ADR  $AC \leftarrow M[PC + ADR]$

## 9. Indexed Addressing Mode

- XR (*Index register*) is added to the address part of the instruction to obtain the effective address
- *Example* : LD ADR(XR)  $AC \leftarrow M[ADR + XR]$

## 10. Base Register Addressing Mode;

- the content of a base register is added to the address part of the instruction to obtain the effective address
- Similar to the indexed addressing mode except that the register is now called a *base register* instead of an *index register*
  - » index register (XR) : LD ADR(XR)  $AC \leftarrow M[ADR + XR]$  
    - index register hold an index number that is relative to the address part of the instruction
  - » base register (BR) : LD ADR(BR)  $AC \leftarrow M[BR + ADR]$  
    - base register hold a base address
    - the address field of the instruction gives a displacement relative to this base address

### ◆ Numerical Example

Addressing Mode	Effective Address	Content of AC
Immediate Address Mode	201	600
Direct Address Mode	600	800
Indirect Address Mode	800	300
Register Mode		400
Register Indirect Mode	400	700
Relative Address Mode	702	325
Indexed Address Mode	600	900
Autoincrement Mode	400	700
Autodecrement Mode	399	450

R1 = 400  
 R1 = 400 (after)  
 R1 = 400 - 1 (prior)

500 + 202 (PC)  
 500 + 100 (XR)

Address	Memory
200	Load to AC
201	Address = 500
202	Next instruction
300	450
400	700
500	800
600	900
702	325
800	300

## 5. DATA TRANSFER AND MANIPULATION

Most computer instructions can be classified into three categories:

1. Data Transfer
2. Data Manipulation
3. Program Control instructions

### 1. Data Transfer instructions:

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

• Typical Data Transfer Instruction : *Tab. 8-5*

- » **Load** : transfer from memory to a processor register, usually an AC (*memory read*)
- » **Store** : transfer from a processor register into memory (*memory write*)
- » **Move** : transfer from one register to another register
- » **Exchange** : swap information between two registers or a register and a memory word
- » **Input/Output** : transfer data among processor registers and input/output device
- » **Push/Pop** : transfer data between processor registers and a memory stack

The eight addressing modes of the Load instruction are

**TABLE 8-6** Eight Addressing Modes for the Load Instruction

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	$AC \leftarrow M[ADR]$
Indirect address	LD @ADR	$AC \leftarrow M[M[ADR]]$
Relative address	LD \$ADR	$AC \leftarrow M[PC + ADR]$
Immediate operand	LD #NBR	$AC \leftarrow NBR$
Index addressing	LD ADR(X)	$AC \leftarrow M[ADR + XR]$
Register	LD R1	$AC \leftarrow R1$
Register indirect	LD (R1)	$AC \leftarrow M[R1]$
Autoincrement	LD (R1)+	$AC \leftarrow M[R1], R1 \leftarrow R1 + 1$

## 2. Data Manipulation Instruction

a) Arithmetic   b) Logical and bit manipulation   c) shift instruction

a) Arithmetic Instructions:

Name	Mnemonic
<b>Increment</b>	<b>INC</b>
<b>Decrement</b>	<b>DEC</b>
<b>Add</b>	<b>ADD</b>
<b>Subtract</b>	<b>SUB</b>
<b>Multiply</b>	<b>MUL</b>
<b>Divide</b>	<b>DIV</b>
<b>Add with carry</b>	<b>ADDC</b>
<b>Subtract with borrow</b>	<b>SUBB</b>
<b>Negate (2's complement)</b>	<b>NEG</b>

## b) Logical and bit manipulation Instructions

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

## c) Shift Instructions:

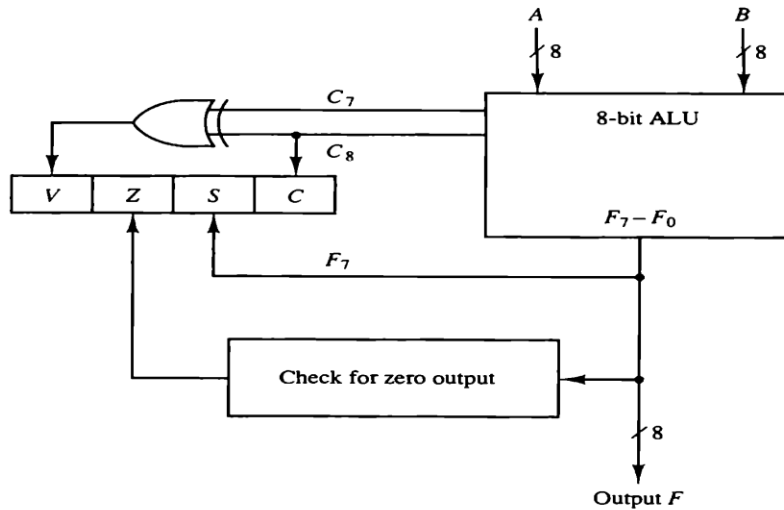
Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

## 6. PROGRAM CONTROL

Program Control instructions are the branch and jump instructions.

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

a) **Status Bit conditions:** Status bits are called condition code bits or flag bits. The block diagram of an 8 bit ALU with a 4 bit status register. The four status bits are symbolized by C, S, Z and V. The bits are set or cleared as a result of an operation performed in the ALU.



Condition code bit or flag bit. The bits are set or cleared as a result of an operation performed in the ALU.

#### ◆ 4-bit status register

- Bit C (*carry*) : set to 1 if the end carry  $C_8$  is 1
- Bit S (*sign*) : set to 1 if  $F_7$  is 1
- Bit Z (*zero*) : set to 1 if the output of the ALU contains all 0's
- Bit V (*overflow*) : set to 1 if the exclusive-OR of the last two carries ( $C_8$  and  $C_7$ ) is equal to 1

- *Flag Example* :  $A - B = A + (2's \text{ Comp. Of } B)$  :  $A = 11110000, B = 00010100$

$11110000$

$+ 11101100$  (2's comp. of B)

$1\ 11011100$

$C = 1, S = 1, V = 0, Z = 0$

#### b) Conditional Branch Instructions

TABLE 8-11 Conditional Branch Instructions

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	$Z = 1$
BNZ	Branch if not zero	$Z = 0$
BC	Branch if carry	$C = 1$
BNC	Branch if no carry	$C = 0$
BP	Branch if plus	$S = 0$
BM	Branch if minus	$S = 1$
BV	Branch if overflow	$V = 1$
BNV	Branch if no overflow	$V = 0$

*Unsigned compare conditions ( $A - B$ )*

BHI	Branch if higher	$A > B$
BHE	Branch if higher or equal	$A \geq B$
BLO	Branch if lower	$A < B$
BLOE	Branch if lower or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$

*Signed compare conditions ( $A - B$ )*

BGT	Branch if greater than	$A > B$
BGE	Branch if greater or equal	$A \geq B$
BLT	Branch if less than	$A < B$
BLE	Branch if less or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$

c) Subroutine call and Return:

- CALL :  $SP \leftarrow SP - 1$  : Decrement stack point  
 $M[SP] \leftarrow PC$  : Push content of PC onto the stack  
 $PC \leftarrow \text{Effective Address}$  : Transfer control to the subroutine
- RETURN :  $PC \leftarrow M[SP]$  : Pop stack and transfer to PC  
 $SP \leftarrow SP + 1$  : Increment stack pointer

d) Program Interrupt

- Program Interrupt
  - » Transfer program control from a currently running program to another service program as a result of an external or internal generated request
  - » Control returns to the original program after the service program is executed
- Interrupt Service Program & Subroutine Call
  - » 1) An interrupt is initiated by an internal or external signal (*except for software interrupt*)
    - A subroutine call is initiated from the execution of an instruction (CALL)
  - » 2) The address of the interrupt service program is determined by the hardware
    - The address of the subroutine call is determined from the address field of an instruction
  - » 3) An interrupt procedure stores all the information necessary to define the state of the CPU
    - A subroutine call stores only the program counter (*Return address*)

- Program Status Word (PSW)
  - » The collection of all status bit conditions in the CPU
- Two CPU Operating Modes
  - » Supervisor (*System*) Mode : Privileged Instruction 실행
    - When the CPU is executing a program that is part of the operating system
  - » User Mode : User program

*CPU operating mode is determined from special bits in the PSW*

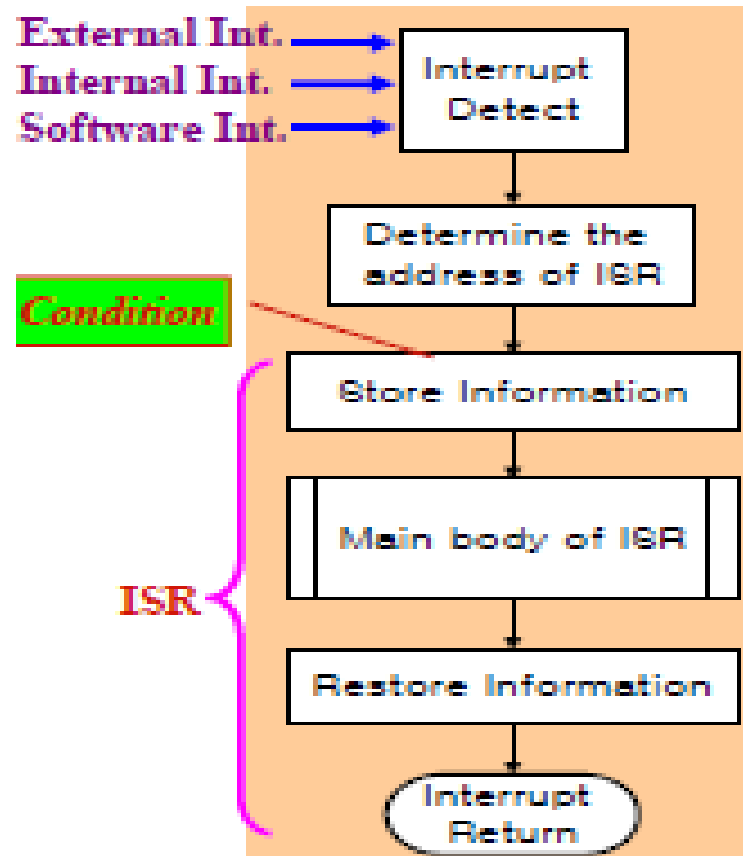
#### e) Types of Interrupt:

There are three major types of interrupts that cause a break in the normal execution of a program.

They can be classified as

1. External interrupts
2. Internal Interrupts
3. Software Interrupts

- 1) External Interrupts
  - » come from I/O device, from a timing device, from a circuit monitoring the power supply, or from any other external source
- 2) Internal Interrupts or TRAP
  - » caused by register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation
- 3) Software Interrupts
  - » initiated by executing an instruction (*INT* or *RST*)
  - » used by the programmer to initiate an interrupt procedure at any desired point in the program





# COMPUTER ARCHITECTURE

**UNIT III Computer Arithmetic :** Hardware Implementation and Algorithm for Addition, Subtraction, Multiplication, Division-Booth Multiplication Algorithm-Floating Point Arithmetic.

P.NO 342

## 1. ADDITION AND SUBTRACTION ALGORITHM:

Most computers use the signed 2's complement representation when performing arithmetic operations with integers.

### ADDITION AND SUBTRACTION WITH SIGNED-MAGNITUDE DATA

The magnitude of two numbers be A and B, when the signed numbers are added or subtracted, there are 8 different conditions depending on the sign of the numbers and the operation performed. The conditions are listed in the first column of the table.

The other columns show the actual operation to be performed with the magnitude of the numbers. Last column is needed to prevent negative zero. When two equal numbers are subtracted the result should be +0 not -0.

**TABLE 10-1** Addition and Subtraction of Signed-Magnitude Numbers

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

### **Addition (subtraction) algorithm:**

- When the signs of A and B are identical, add the two magnitudes and attach the sign of A to the result.
- When the signs of A and B are different, compare the magnitudes and subtract the smaller number from the larger. Choose the sign of the result to be same as A, if  $A > B$  or the complement of the sign of A if  $A < B$ .
- If the two magnitudes are equal, subtract B from A and make the sign of the result positive.
- The procedure to be followed for identical signs in the addition algorithm is the same as for different signs in the subtraction algorithm.

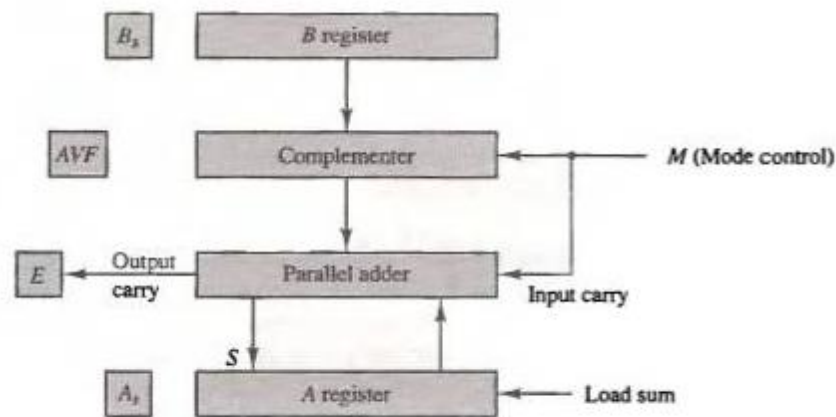
### **HARDWARE IMPLEMENTATION:**

To implement the two arithmetic operations with hardware, the two numbers be stored in registers. Let A and B be two registers that hold the magnitudes of the numbers and  $A_s$  and  $B_s$  be two flipflops that hold the corresponding signs. The result of the operation may be transferred to a third register, if the result is transferred into A and  $A_s$  together form accumulator register.

First a **parallel adder** is needed to perform the microoperation  $A+B$ . 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 microoperations  $A-B$  and  $B-A$ . The sign relationship can be determined from an **XOR** with  $A_s$  and  $B_s$  as inputs.

This procedure requires a **magnitude comparator**, an **adder** and **two subtractors**. Different procedure can be used with less equipment. First subtraction can be accomplished by means of **complement and add**. Second the result of the comparison can be determined from the end carry after the subtraction. So only **complementer** and **adder** is needed.

The block diagram for implementing addition and subtraction operation is shown in figure.



**Figure 10-1** Hardware for signed-magnitude addition and subtraction.

It consists of registers A and B and sign flipflops  $A_s$  and  $B_s$ . Subtraction is done by adding A to the 2's complement of B. The output carry is transferred to flip-flop E. The add-overflow flipflop AVF holds the overflow bit when A and B are added.

The addition of A plus B is done through the **parallel adder**. The S(sum) output of the adder is applied to the input of the A register. The **complementer** provides an output of B or the complement of B depending on mode control. The complementer consists of **XOR** gates and the parallel adder consists of full-adder circuits.

M signal is applied to the input carry of the adder.

- When  $M=0$ , the output of B is transferred to the adder, the input carry is 0, and the output of the adder is equal to the sum  $A+B$ .
- When  $M=1$ , the 1's complement of B is applied to the adder, the input carry is 1, the output  $S=A+B+1$ . This is equal to A plus the 2's complement of B, which is equivalent to the subtraction  $A-B$ .

## Hardware algorithm:

The flowchart for the hardware algorithm is shown below. The two signs  $A_s$  and  $B_s$  are compared by an exclusive OR gate. If the output of the gate is 0, the signs are identical; If it is 1, the signs are different.

For an add operation, identical signs dictate the magnitudes be added. For a subtract operation, different signs dictate that the magnitudes be added. The magnitudes are added with a microoperation  $EA \leftarrow A + B$  where EA is a register that combines E and A. The carry in E after the addition constitutes an overflow if it is equal to 1. The value of E is transferred into the add overflow flipflop AVF.

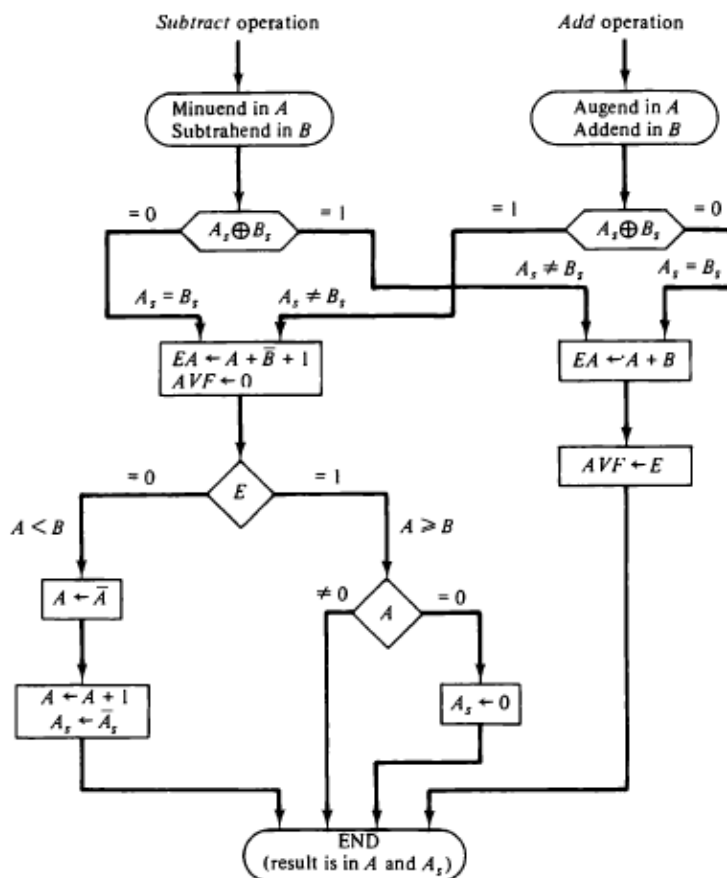


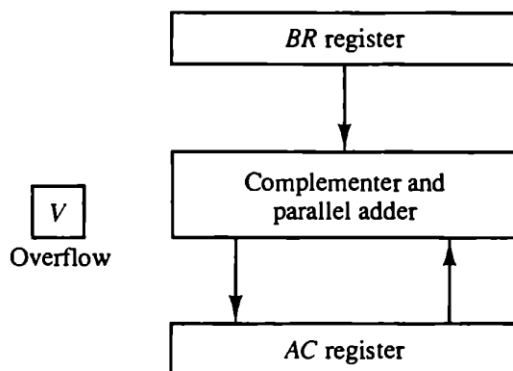
Figure 10-2 Flowchart for add and subtract operations.

- The two magnitudes are subtracted if the signs are different for an add operation or identical for a subtract operation. The magnitudes are subtracted by adding A to the 2's complement of B. No overflow can occur if the numbers are subtracted so AVF is cleared to zero.
- A 1 in E indicates that  $A \geq B$  and the number in A is the correct result. IF the number is zero, the sign A must be made positive to avoid a negative zero.
- A 0 in E indicates that  $A < B$ , it is necessary to takes the 2's complement of A as  $A \leftarrow \hat{A} + 1$ . A register has circuits for microoperations complement and increment.
- The final result is found in register A and its sign in  $A_s$ . The value in AVF provides an overflow indication. The final value of E is immaterial.

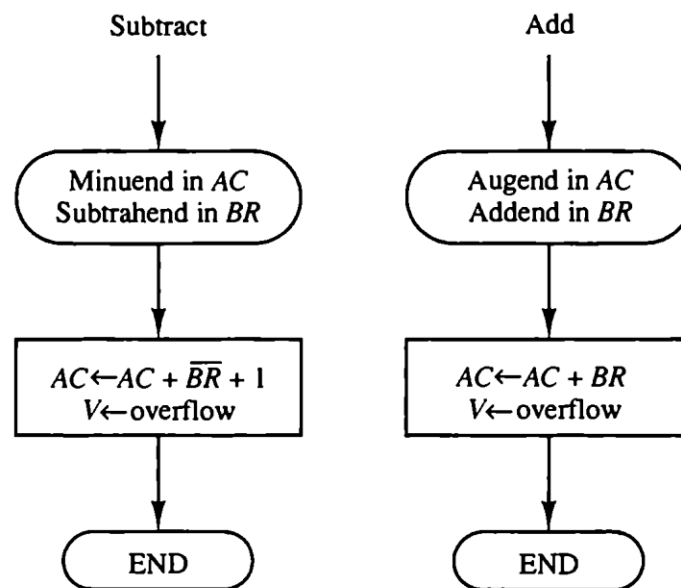
#### ADDITION AND SUBTRACTION WITH SIGNED 2'S COMPLEMENT DATA:

- The signed 2's complement representation of numbers together with arithmetic algorithms for addition and subtraction. The left most bit of a binary number represents the sign bit: 0 for positive and 1 for negative.
- If the sign bit is 1, the entire number is represented in 2's complement form. For example +33 is represented as 00100001 and -33 as 11011111 (is the 2's complement of 00100001).
- The register configuration of hardware implementation is shown in figure 10.3

**Figure 10-3** Hardware for signed-2's complement addition and subtraction.



- As shown in figure 10.4, sum is obtained by adding the contents of AC and BR. The overflow bit V is set to 1, if the XOR of the last two carries is 1 and it is cleared to 0 otherwise.
- The subtraction operation is obtained by adding the content of AC to the 2's complement of BR. Taking the 2's complement of BR has the effect of changing a positive number to negative and vice versa.



**Figure 10-4** Algorithm for adding and subtracting numbers in signed-2's complement representation.

## 2. MULTIPLICATION ALGORITHM

Multiplication of two fixed point binary numbers in signed magnitude representation is done by a process of successive shift and add operations.

$$\begin{array}{r}
 \begin{array}{r}
 23 \quad 10111 \\
 19 \quad \times 10011 \\
 \hline
 \end{array}
 \begin{array}{l}
 \text{Multiplicand} \\
 \text{Multiplier}
 \end{array} \\
 \begin{array}{r}
 10111 \\
 10111 \\
 00000 \quad + \\
 00000 \\
 10111 \\
 \hline
 \end{array} \\
 437 \quad 110110101 \quad \text{Product}
 \end{array}$$

The process consists of looking at successive bits of the multiplier, least significant bit first. If the multiplier bit is a 1, the multiplicand is copied down, otherwise zeros are copied down. The numbers copied down are shifted one position to the left from the previous number. Finally the numbers are added and their sum forms the product.

The sign of the product is determined from the signs of the multiplicand and multiplier. If they are alike, the sign of the product is positive. If they are unlike, the sign of the product is negative.

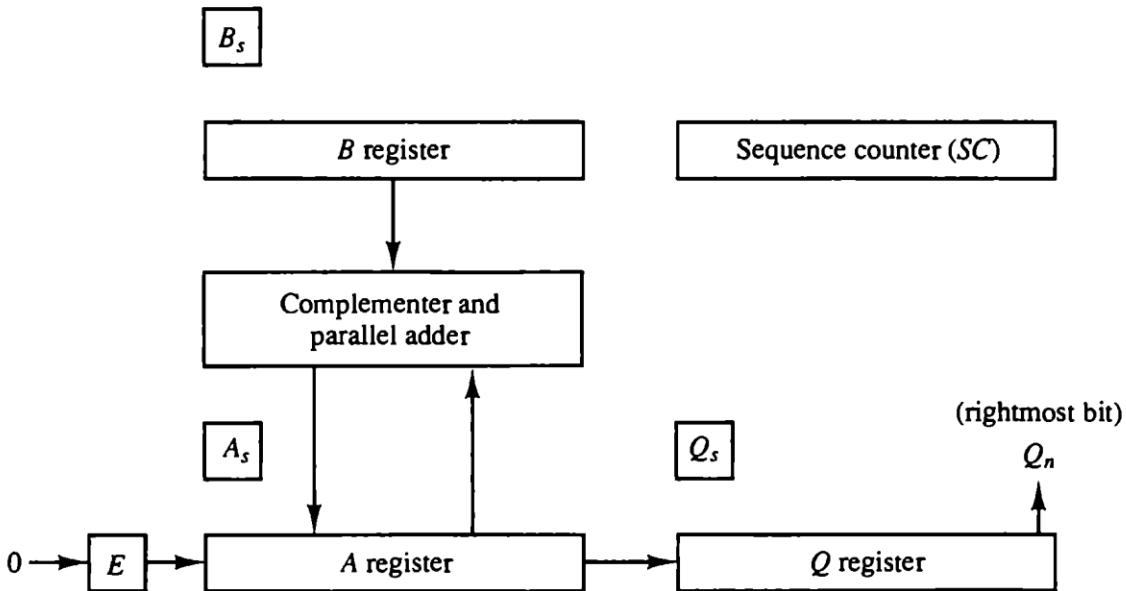
#### **HARDWARE IMPLEMENTATION OF SIGNED MAGNITUDE DATA:**

First instead of providing registers to store and add simultaneously as many binary numbers as there are bits in the multiplier, an adder is provided for the summation of only two binary numbers and successively accumulate the partial products in a register.

Second instead of shifting the multiplicand to the left, the partial product is shifted to the right which results in leaving the partial product and the multiplicand in the required relative positions

Third when the corresponding bit of the multiplier is 0, there is no need to add all zeros to the partial product since it will not alter its value.

**Figure 10-5** Hardware for multiply operation.



The hardware for multiplication consists of two more registers A and B. The multiplicand is stored in B register and multiplier in Q register. The sequence counter SC is initially set to a number equal to the number of bits in the multiplier. The counter is decremented by 1 after forming each partial product. When the content of the counter reaches zero, the product is formed and the process stops.

Initially the multiplicand is in B register and multiplier in Q. The sum of A and B forms a partial product is transferred to EA register, both partial product and multiplier are shifted to the right. The shift will be denoted by shr EAQ to designate the right shift. The least significant bit of A is shifted into the most significant position of Q bit from E is shifted into the most significant position of A and 0 is shifted into E. After the shift, one bit of the partial product is shifted into Q, pushing the multiplier bits one position to the right. The rightmost flipflop in register Q is Q<sub>n</sub> will hold the bit of the multiplier which is inspected next.



**Hardware algorithm:**

The flowchart of the hardware multiply algorithm. Initially the multiplicand is in B and the multiplier in Q. Their corresponding signs are in Bs and Qs. The signs are compared and both A and Q are set correspond to the sign of the product since a double length product will be stored in registers A and Q.

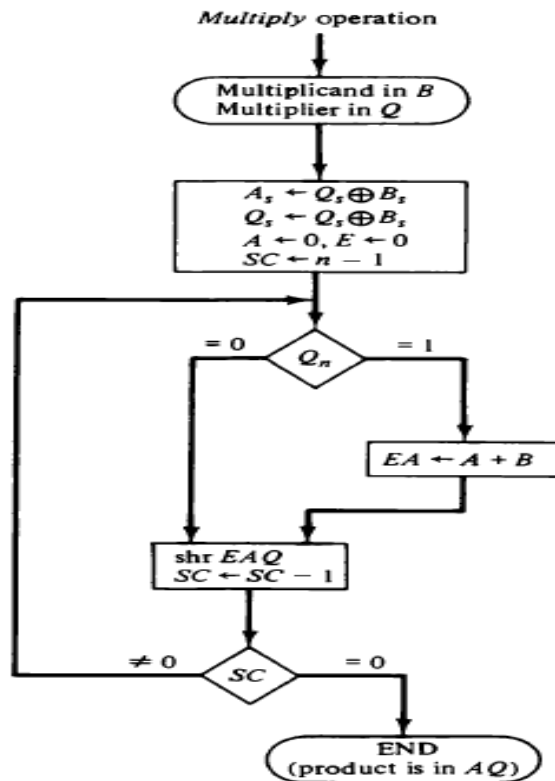
Registers A and E are cleared and the sequence counter SC is set to a number equal to the number of bits of the multiplier. After the initialization, the low order bit of the multiplier in Q<sub>n</sub> is tested. If it is a 1, the multiplicand in B is added to the present partial product in A. If it is a 0, nothing is done. Register EAQ is then shifted once to the right to form the new partial product.

The sequence counter is decremented by 1 and its new value checked.

If it is not equal to zero, the process is repeated and a new partial product is formed. The process stops when SC=0. The partial product formed in A is shifted into Q one bit at a time and eventually replaces the multiplier, The final product is available in both A and Q with A holding

the most significant bits and Q holding the least significant bits.

**Figure 10-6** Flowchart for multiply operation.



#### 4. BOOTH MULTIPLICATION ALGORITHM

Booth algorithm gives a procedure for multiplying binary integers in signed 2's complement representation. It operates that strings of 0's in the multiplier require not addition but just shifting and a string of 1's in the multiplier from bit weight  $2^k$  to  $2^m$  can be treated as  $2^{k+1} - 2^m$ .

TABLE 10-2 Numerical Example for Binary Multiplier

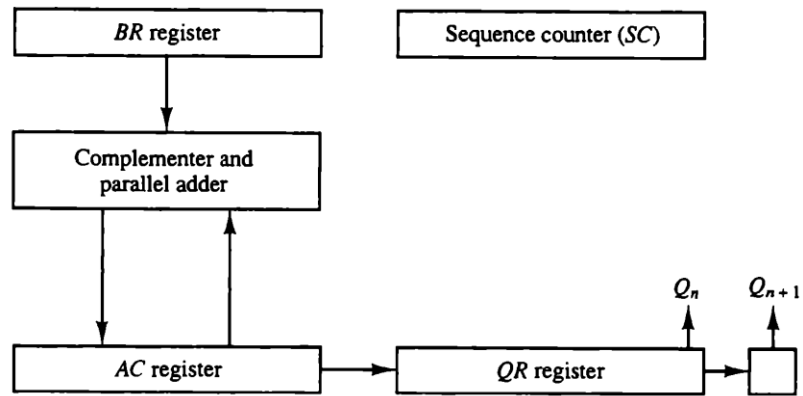
Multiplicand $B = 10111$	$E$	$A$	$Q$	$SC$
Multiplier in $Q$	0	00000	10011	101
$Q_n = 1$ ; add $B$		<u>10111</u>		
First partial product	0	10111		
Shift right $EAQ$	0	01011	11001	100
$Q_n = 1$ ; add $B$		<u>10111</u>		
Second partial product	1	00010		
Shift right $EAQ$	0	10001	01100	011
$Q_n = 0$ ; shift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

The binary number 001110(+14) has a string of 1's from  $2^3$  to  $2^1$  ( $k=3, m=1$ ). The number can be represented as  $2^{k+1} - 2^m = 2^4 - 2^1 = 16 - 2 = 14$ . The multiplication of  $M \times 14$  where  $M$  is the multiplicand and 14 the multiplier as  $M \times 2^4 - M \times 2^1$ . Thus the product can be obtained by shifting the binary multiplicand  $M$  four times to the left and subtracting  $M$  shifted left once. Prior to the shifting the multiplicand may be added to the partial product, subtracted from the partial product or left unchanged to the rules to

1. The multiplicand is subtracted from the partial product upon encountering the first least 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 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.

The hardware implementation of Booth algorithm is shown below.

Figure 10-7 Hardware for Booth algorithm.



Rename registers A, B and Q as AC, BR and QR. The flowchart for Booth algorithm is

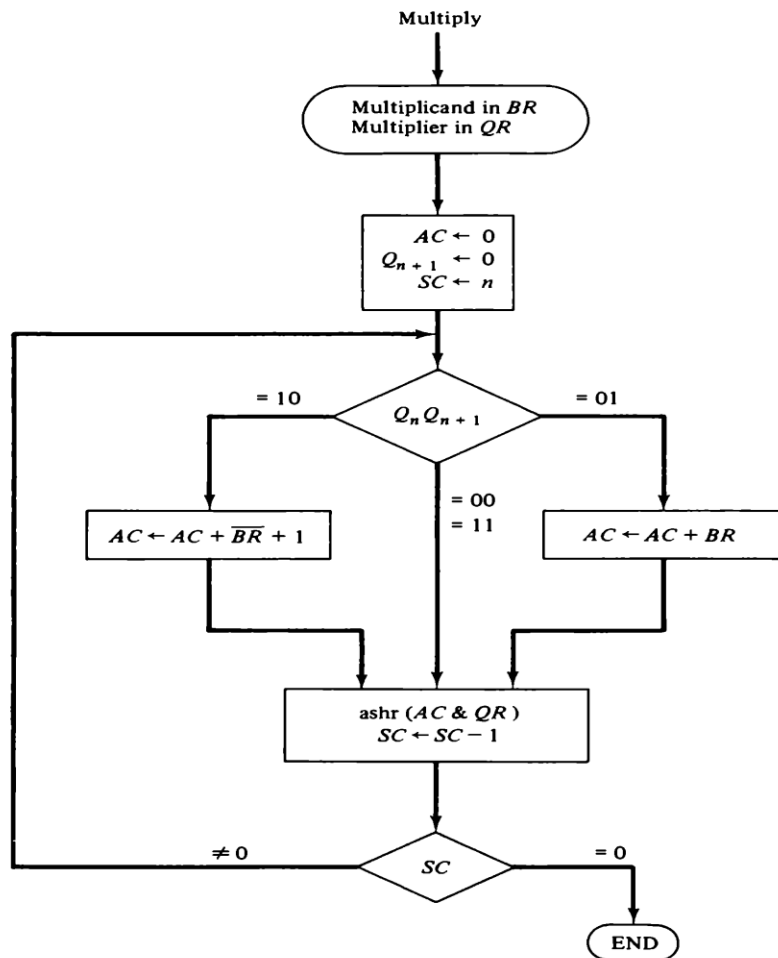


Figure 10-8 Booth algorithm for multiplication of signed-2's complement numbers.

Bit  $Q_{n+1}$  are initially cleared to 0 and the sequence counter SC is set to a number  $n$  equal to the number of bits in the multiplier. The two bits of the multiplier in  $Q_n$  and  $Q_{n+1}$  are inspected. If the two bits are equal to 10, the first string 1 in a string of 1's has been encountered. If the two bits are equal to 01, the first 0 in a string of 0's is encountered. When the two bits are equal, the partial product does not change. An overflow cannot occur because the addition and subtraction of the multiplicand follow each other.

## **5. FLOATING POINT ARITHMETIC OPERATIONS:**

### **a) Addition and Subtraction:**

During addition or subtraction, the two floating point operands are in AC and BR. 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 mantissa.
4. Normalize the result.

The flowchart for adding or subtracting two floating point binary numbers is shown below.

If BR is equal to zero, the operation is terminated with the value in the AC being the result. If AC is equal to zero, we transfer the content of BR into AC and also complement its sign if the numbers are to be subtracted. If neither number is equal to zero, align the mantissa. The magnitude comparator attached to exponents  $a$  and  $b$  provides three outputs. If the two

exponents are equal, perform arithmetic operation. If the exponents are not equal, the mantissa

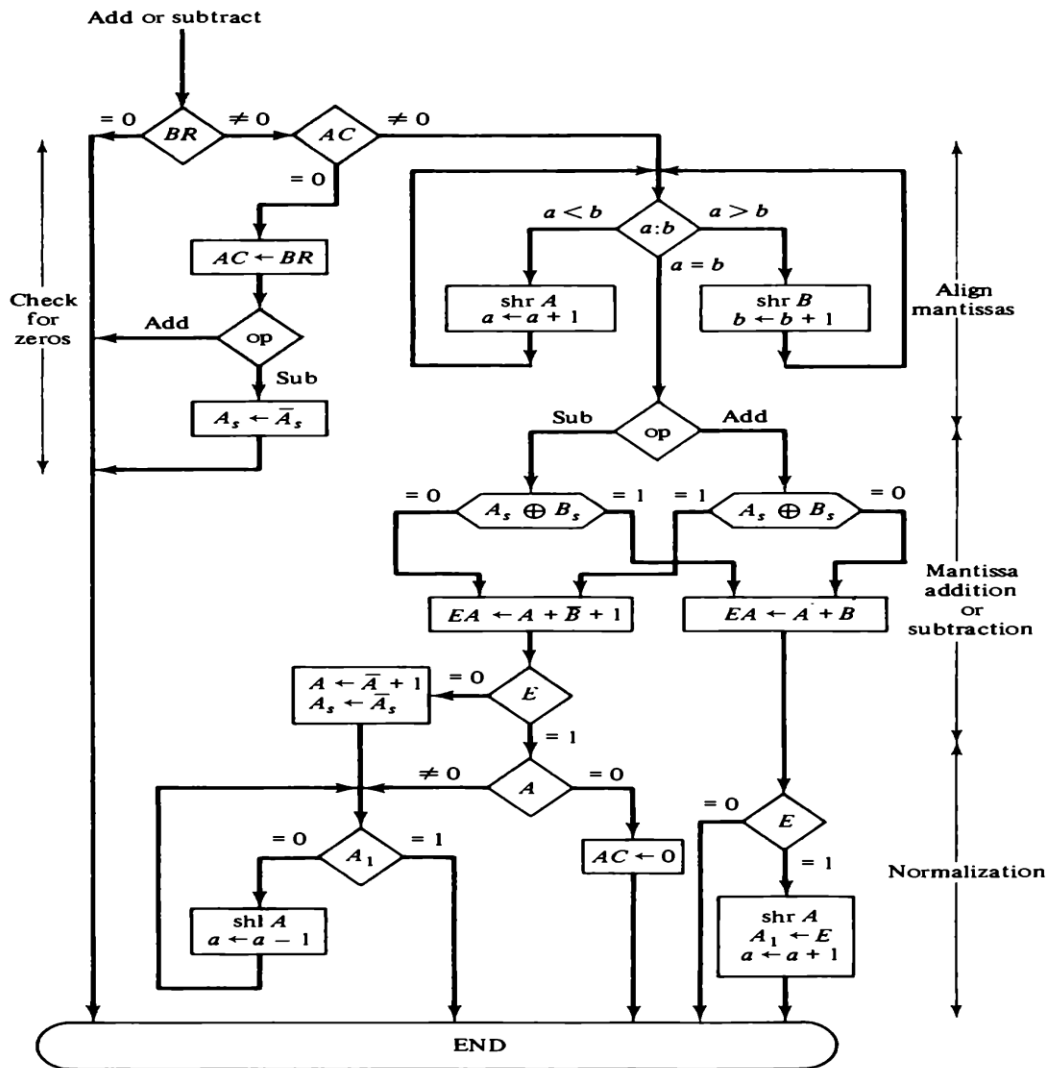


Figure 10-15 Addition and subtraction of floating-point numbers.

having the smaller exponent is shifted to the right and its exponent incremented. The process is repeated until the two exponents are equal. If an overflow occurs when the magnitudes are added, it is transferred to flip flop E. If E is equal to 1, the bit is transferred into A and all other bits of A are shifted right.

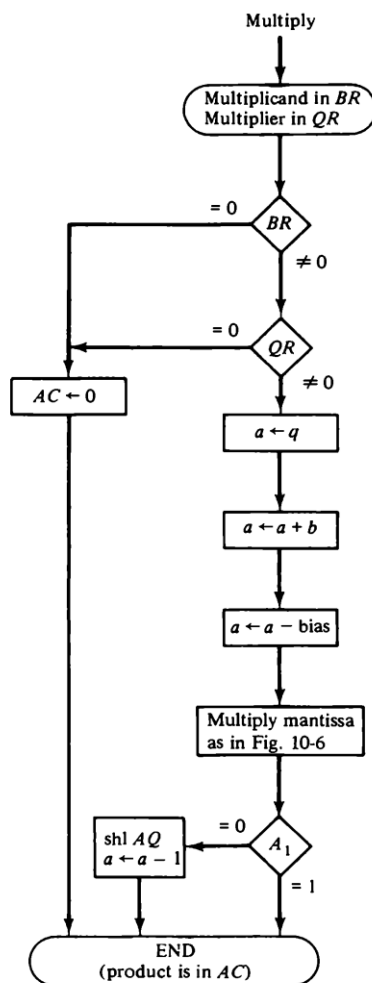
If the magnitudes were subtracted, the result may be zero or may have an underflow. If the mantissa is zero, the entire floating point number in the AC is made zero. Otherwise the mantissa must have at least one bit that is equal to 1.

## b) Multiplication:

Multiplication of two floating point numbers multiply the mantissa and add the exponents. The multiplication algorithm is subdivided into four parts.

1. check for zeros
2. Add the exponents
3. Multiply the mantissa
4. Normalize the product

Figure 10-16 Multiplication of floating-point numbers.



Steps 2 and 3 can be done simultaneously if separate adders are available for the mantissas and exponents. Flowchart for floating point multiplication is shown above,

The exponent of the multiplier is in  $q$  and the adder is between exponents  $a$  and  $b$ . To transfer the exponents from  $q$  to  $a$ , add the two exponents and transfer the sum into  $a$ .

### c) DIVISION:

Floating point division requires that the exponents be subtracted and the mantissa divided. The division algorithm can be subdivided into 5 parts.

1. check for zeros
2. Initialize registers and evaluate the sign.
3. Align the dividend
4. subtract the exponents
5. divide the mantissa

Flowchart for floating point division is

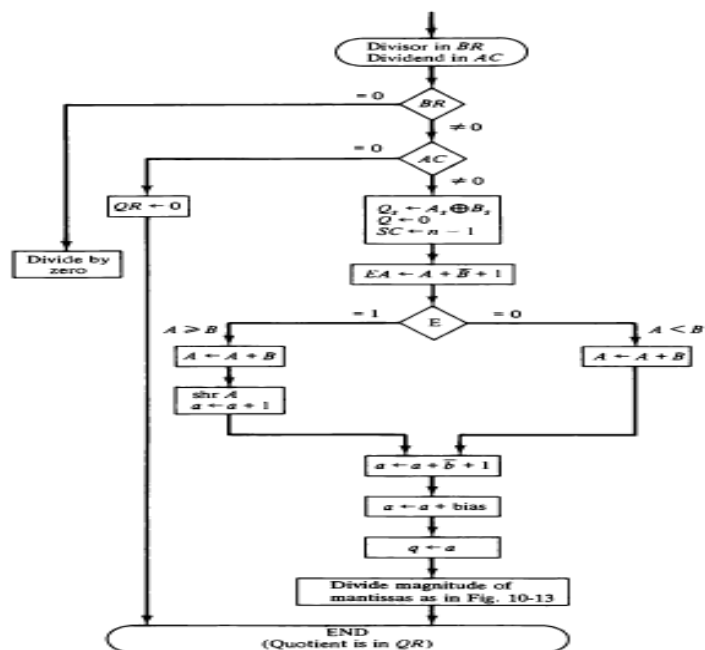


Figure 10-17 Division of floating-point numbers.



# COMPUTER ARCHITECTURE

## UNIT IV

**Input Output Organization** : Input – Output Interface – Asynchronous data transfer – Modes of transfer – Priority Interrupt – Direct Memory Access (DMA).

### 1. INPUT-OUTPUT INTERFACE:

Input output interface provides a method for transferring information between internal storage and external devices. Peripherals connected to a computer need special communication links for interfacing them with the CPU. Communication link resolves the differences exist between central computer and each peripheral.

The major differences are

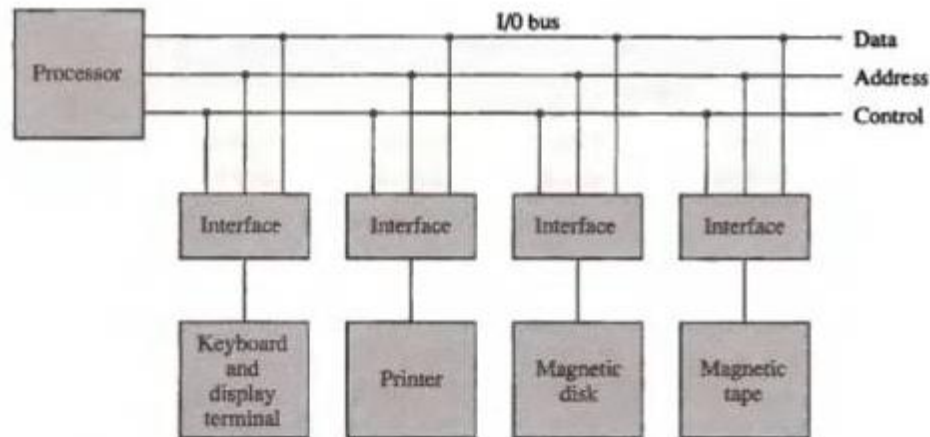
1. Peripherals are electromechanical and electromagnetic devices and their operation is different from the operation of the CPU.
2. The data transfer rate of peripherals is slower than the transfer rate of the CPU.
3. Data codes and formats in peripherals differ from the word format in CPU and memory.
4. The operating modes of peripherals are different from each other and each must be controlled not to disturb the operation of other peripherals.

To resolve these differences, computer systems include special hardware components to supervise and synchronize all input and output transfers. The components are called interface units because they interface between processor bus and the peripheral device. Each device have its own controller that supervises the operation in the peripheral.

### a) IO Bus and INTERFACE MODULES

The IO bus consists of data lines, address lines and control lines. The magnetic disk, printer and terminal are employed in any general purpose computer. Magnetic tape is used in some computers for backup storage. Each peripheral device is associated with an interface unit.

Figure 11-1 Connection of I/O bus to input-output devices.



- Each interface decodes the address and control received from the IO bus, interprets them for the peripheral and provides signals for the peripheral controller.
- It also synchronizes the data flow and supervises the transfer between peripheral and processor.
- The IO bus from the processor is attached to all peripheral interfaces.
- To communicate with a particular device, the processor places a device address on the address lines.
- Each interface attached to the IO bus contains an address decoder that monitors the address lines.
- When the interface detects its own address, it activates the path between the bus lines and the device that it controls.
- The function code is referred to as an IO command and is in essence an instruction that is executed in the interface and its attached peripheral unit.
- There are four types of commands that an interface may receive. They are classified as control, status, data output and data input.

1. A control command is issued to activate the peripheral and to inform it what to do. For example, a magnetic tape unit may be instructed to backspace the tape by one record, to rewind the type or to start the tape moving in the forward direction.

2. A status command is used to test various status conditions in the interface and the peripheral. During the transfer, one or more errors may occur which are detected by the interface.

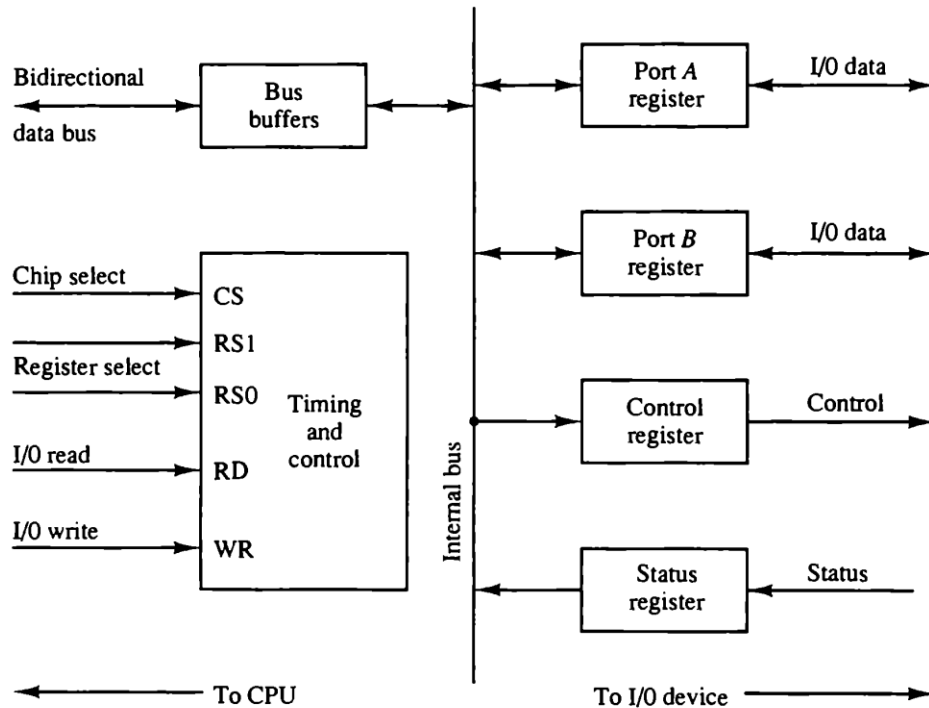
3. A data output command causes the interface to respond by transferring data from the bus into one of its registers. When the tape is in correct position, the processor issues a data output command. The interface then communicates with the tape controller and sends the data to be stored on the tape.

4. The data input command is the opposite of the data output. The interface receives an item of data from the peripheral and places it in the buffer register. The processor checks if data are available by means of a status command and then issues a data input command. The interface places the data on the data lines accepted by the processor.

## **b) IO VERSUS MEMORY BUS**

Also the processor must communicate with the IO and memory unit. The IO bus, memory bus contains data, address and read write control lines. There are **three** ways that computer buses can be used to communicate with memory and IO.

1. Use two separate buses, one for memory and the other for IO.
  2. Use one common bus for both memory and IO but have separate control lines for each.
  3. Use one common bus for memory and IO with common control lines.
- The computer has independent sets of data, address and control buses, one for accessing memory and the other for IO.
  - The memory communicates with both the CPU and the input output processor(IOP) in addition to the CPU. The IOP communicates with the input and output devices through a separate IO bus with its own address, data and control lines. The purpose of IOP is to provide an independent pathway for the transfer of information between external devices and internal memory.



CS	RS1	RS0	Register selected
0	x	x	None: data bus in high-impedance
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

**Figure 11-2** Example of I/O interface unit.

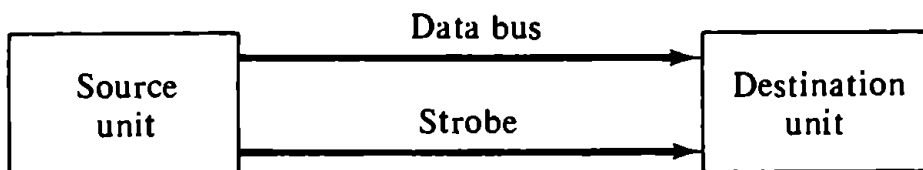
## 2. ASYNCHRONOUS DATA TRANSFER

- ✓ The internal operations in a digital system are organized by means of clock pulses supplied by a common pulse generator.
- ✓ Two units such as CPU and IO interface are designed independently of each other.
- ✓ If the registers in the interface share a common clock with the CPU registers, the transfer between the two units is said to be synchronous.
- ✓ Asynchronous data transfer between two independent units requires that control of signals be transmitted between the communicating units to indicate the time at which data is being transmitted;

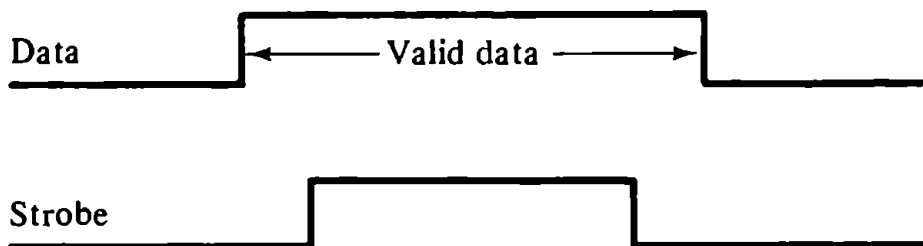
- ✓ Strobe pulse is supplied by one of the units to indicate to the other unit when the transfer has to occur.
- ✓ Another method is to accompany each data item being transferred with a control signal that indicates the presence of data in the bus.
- ✓ The unit receiving the data item responds with another control signal to acknowledge receipt of the data.
- ✓ This type of agreement between the two independent units is referred to as **handshaking**.

**a) Strobe control:**

- ✓ The strobe may be activated by either the source or the destination unit.
- ✓ A source initiated transfer, the data bus carries the binary information from source unit to the destination unit.
- ✓ As in the timing diagram, the source unit first places the data on the data bus.



**(a) Block diagram**

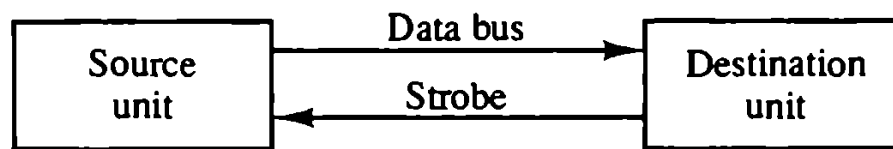


**(b) Timing diagram**

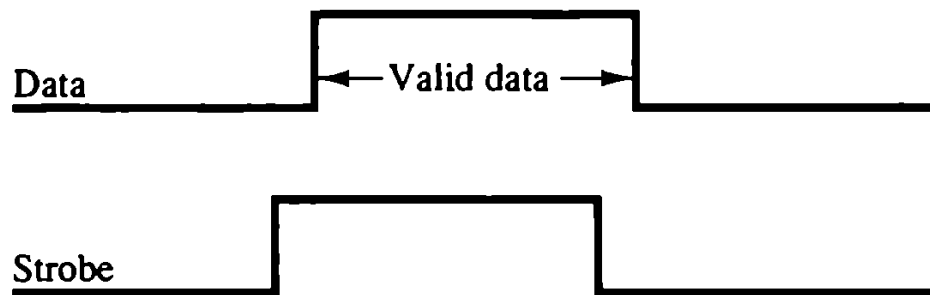
**Figure 11-3 Source-initiated strobe for data transfer.**

- ✓ The information on the data bus and the strobe signal remain in the active state for a sufficient time to allow the destination unit to receive the data.
- ✓ The data must be valid and remain in the bus long enough for the destination unit to accept it.

- ✓ Strobe pulses are actually controlled by the clock pulses in the CPU is always in control of the buses and informs the external units how to transfer data



(a) Block diagram

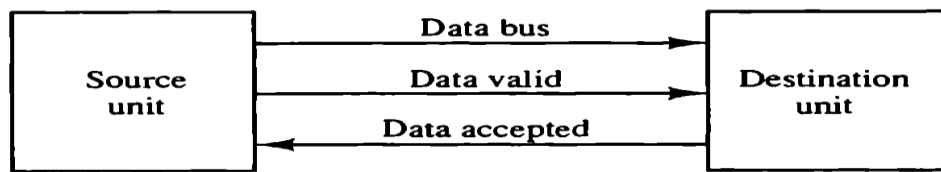


(b) Timing diagram

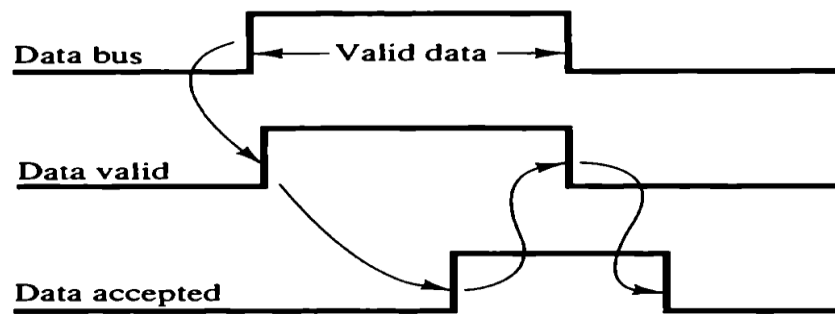
**Figure 11-4 Destination-initiated strobe for data transfer.**

**b) Handshaking:**

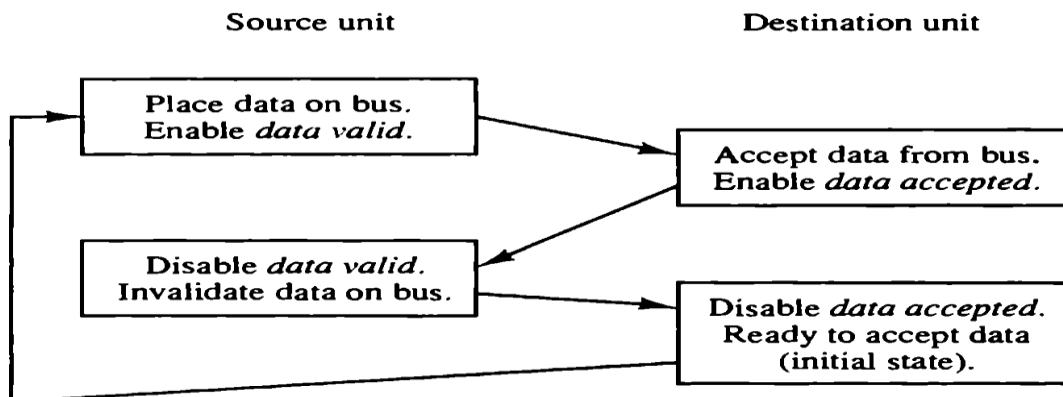
- ✓ The disadvantage of the strobe method is that the source unit that indicates the transfer has no way of knowing whether the destination unit has actually received the data item that was placed in the bus.
- ✓ Similarly a destination unit that indicates the transfer has no way of knowing whether the source unit has actually placed the data on the bus.
- ✓ The handshake method solves the problem by introducing a second control signal that provides a reply to the unit that indicates the transfer.
- ✓ The **data transfer procedure when initiated by the source**, the two handshaking lines are data valid, which is generated by the source unit and data accepted generated by the destination unit. The timing diagram shows the exchange of signals between the two units.



(a) Block diagram



(b) Timing diagram

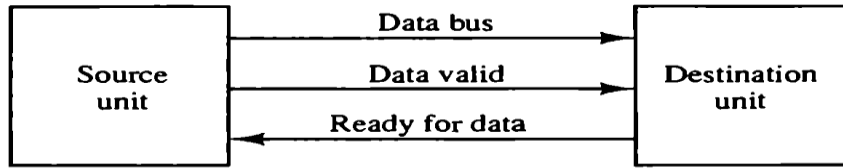


(c) Sequence of events

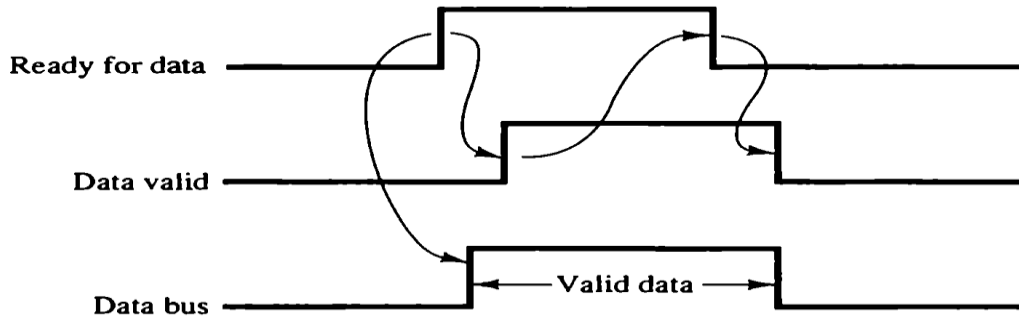
**Figure 11-5 Source-initiated transfer using handshaking.**

- ✓ The source unit initiates the transfer by placing the data on the bus and enabling its data valid signal. The data accepted signal is activated by the destination unit after it accepts the data from the bus.
- ✓ The source unit then disables the data valid signal which invalidates the data on the bus.
- ✓ The destination unit then disables its data accepted signal and the system goes into its initial state.
- ✓ The **destination initiated transfer** using handshaking lines is shown in figure.

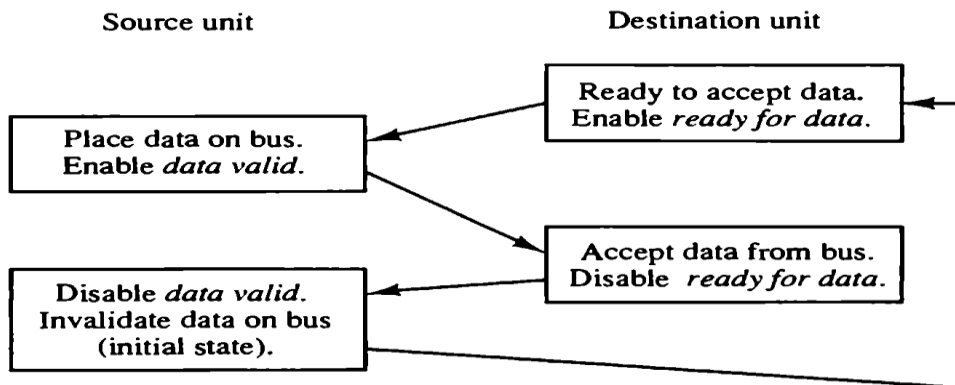
**Figure 11-6** Destination-initiated transfer using handshaking.



(a) Block diagram



(b) Timing diagram



(c) Sequence of events

- ✓ The name of the signal generated by the destination unit has been changed to ready for data to reflect its meaning.
- ✓ The handshaking scheme provides a high degree of flexibility and reliability because the successful completion of a data transfer relies on active participation of both units.
- ✓ If one unit is faulty the data transfer will not be completed. Such an error can be detected by means of a timeout mechanism which produces an alarm if the data transfer is not completed within a predetermined time.



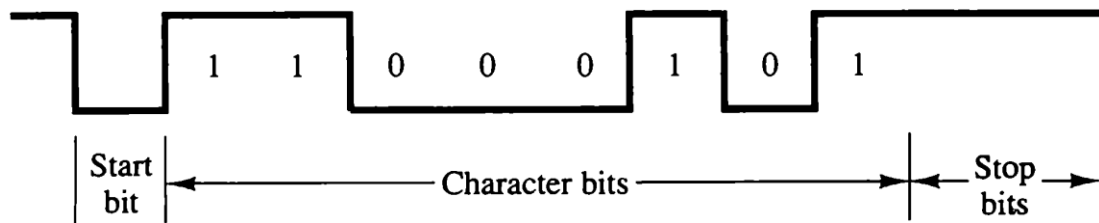
### c) Asynchronous serial transfer:

- ✓ The transfer of data between two units may be done in parallel or serial.
- ✓ In parallel data transmission, each bit of the message has its own path and the total message is transmitted at the same time. This means that an n-bit message must be transmitted through n separate conductor paths.
- ✓ In serial data transmission, each bit in the message is sent in sequence one at a time.
- ✓ Serial transmission can be synchronous or asynchronous.
- ✓ In synchronous transmission, the two units share a common clock frequency and bits are transmitted continuously at the rate dictated by the clock pulses.
- ✓ In asynchronous transmission, binary information is sent only when it is available and the line remains idle when there is no information to be sent.
- ✓ Each character consists of three parts: a start bit, a character bits and stop bits. The convention is that 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 is called the stop bit is always a 1.
- ✓ A transmitted character can be detected by the receiver from knowledge of the transmission rules.

1. When a character is not being sent, the line is kept in the 1 state.
  2. The initiation of a character transmission is detected from the start bit which is always 0.
  3. The character bit always follow the start bit.
  4. After the last bit of the character is transmitted, a stop bit is detected when the line returns to the 1 state for atleast one bit times.
- ✓ As an illustration, consider the serial transmission of a terminal whose transfer rate is 10 character per second. Each transmitted character consists of a start bit, eight information bits and two stop bits for a total of 11 bits.
  - ✓ The baud rate is defined as the rate at which serial information is transmitted and is equivalent to the data transfer in bits per second.

Ten characters per second with an 11 bit format has a transfer rate of 110 bauds.

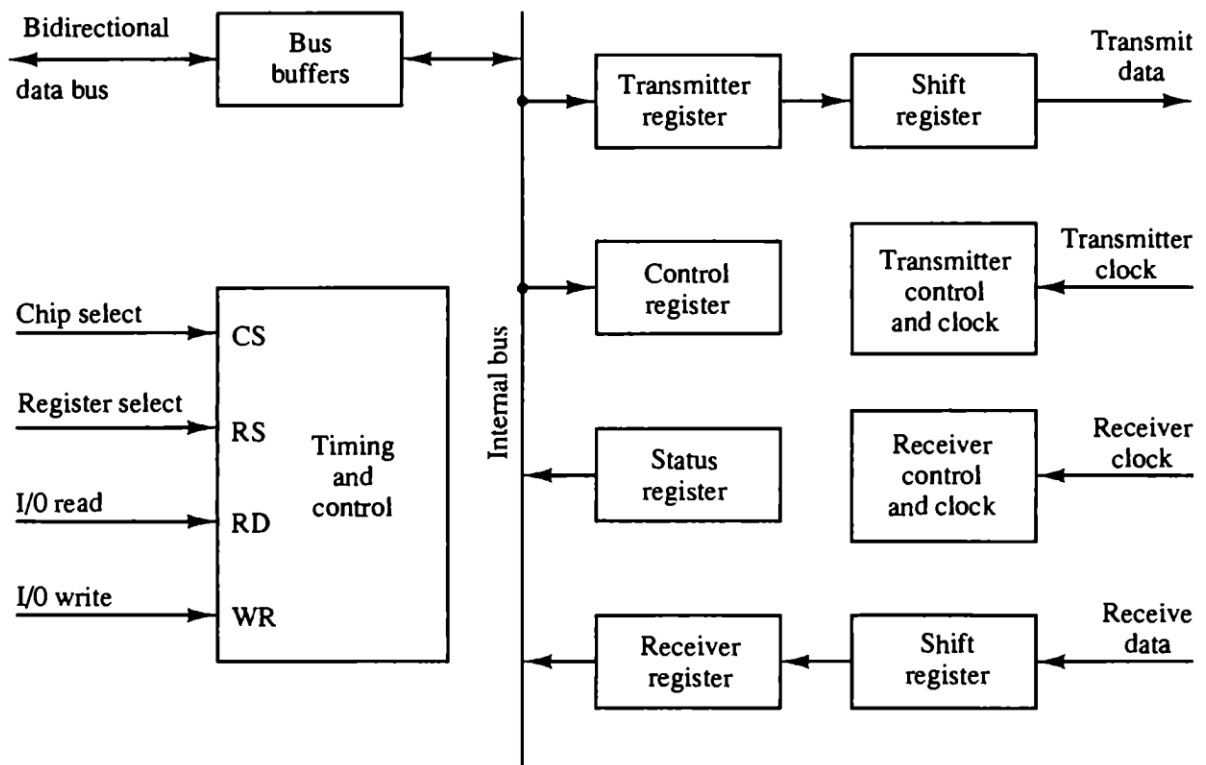
**Figure 11-7** Asynchronous serial transmission.



- ✓ The terminal has a keyboard and a printer. The terminal interface consists of a transmitter and a receiver. The **transmitter** accepts an 8 bit character from the computer and proceeds to send a serial 11 bit message into the printer line. The **receiver** accepts a serial 11 bit message from the keyboard line and forwards the 8 bit character code into the computer.
- ✓ Integrated circuits are available which are specifically designed to provide the interface between computer and similar interactive terminals. Such a circuit is called an asynchronous communication interface or a universal asynchronous receiver transmitter(**UART**),

**d) Asynchronous communication Interface:**

- ✓ The block diagram of an asynchronous communication interface is shown in figure. It functions as both a transmitter and a receiver. The interface is initialized for a particular mode of transfer by means of a control byte that is loaded into its control register. The transmitter register accepts a data byte from the CPU through the data bus.
- ✓ The CPU can read the status register to check the status of the flag bits and to determine if any errors have occurred. The chip select and the read and write control lines communicate with the CPU. The chip select(CS) input is used to select the interface through the address bus. The register select(RS) is associated with the read(RD) and Write(WR) controls. Two registers are write only and two are read only.
- ✓ The operation of the asynchronous communication interface is initialized by the CPU by sending a byte to the control register.



CS	RS	Operation	Register selected
0	x	x	None: data bus in high-impedance
1	0	WR	Transmitter register
1	1	WR	Control register
1	0	RD	Receiver register
1	1	RD	Status register

**Figure 11-8** Block diagram of a typical asynchronous communication interface.

- ✓ The initialization procedure places the interface in a specific mode of operations as it defines certain parameters such as the baud rate to use, how many bits are in each character, whether to generate and check parity and how many stop bits are appended to each character.
- ✓ Two bits in the status register are used as flags. One bit is used to indicate whether the transmitter register is empty and another bit is used to indicate whether the receiver register is full.

- ✓ The operation of the **transmitter** portion of the interface is as follows: The CPU reads the status register and checks the flag to see the transmitter register is empty. IF it is empty, the CPU transfers a character to the transmitter register and the interface clears the flag to mark the register full. The first bit in the transmitter shift register is set to 0 to generate a start bit. The character is transferred in parallel form the transmitter register to the shift register and the appropriate number of stop bits is appended into the shift register.
- ✓ The operation of the **receiver** portion of the interface is similar. The receive data input is in the 1-state when the line is idle. The receiver control monitors the receive data line for a 0 signal to detect the occurrence of a start bit. Once a start bit has been detected, the incoming
- ✓ Bits of the character are shifted into the shift register at the prescribed baud rate.
- ✓ The **interface** checks for any possible errors during transmission and sets appropriate bits in the status register. The CPU can read the status register at any time to check if any errors have occurred. **Three possible errors** that the interface checks during transmission are parity error, framing error and overrun error.
- ✓ **Parity error** occurs if the number of 1's in the received data is not in the correct parity. A **framing error** occurs if the right numbers of stop bits is not detected at the end of the received character. AN **overrun error** occurs if the CPU does not read the character from the receiver register before the next one becomes available in the shift register. Overrun error results in a loss of characters in the received data stream.

### 3. MODES OF TRANSFER

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

1. Programmed IO
2. Interrupt initiated IO
3. Direct Memory Access(DMA)

1. **Programmed IO**: operations are the result of IO instructions written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually the transfer is to and from a CPU register and peripheral. Other instructions are needed to transfer the data to and from CPU and memory.

In the **programmed IO method**, the CPU stays in a program loop until the IO unit indicates that it is ready for data transfer. This is a time consuming process since it keeps the processor busy needlessly. It can be avoided by using an interrupt facility and special commands to inform the interface to issue an interrupt request signal when the data are available from the device.

Transfer of data from programmed IO is between CPU and peripheral. In **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.

Example of programmed IO: in the programmed IO method, the IO device does not have direct access to memory. A transfer from an IO device to memory requires the execution of several instructions by the CPU including an input instruction to transfer the data from the device to the CPU and a store instruction to transfer the data from the CPU to memory.

An example of data transfer from an IO device through into the CPU is shown in the device transfers bytes of data on a time as they are available when a byte of data is available, the device is in the IO bus enables its data valid line.

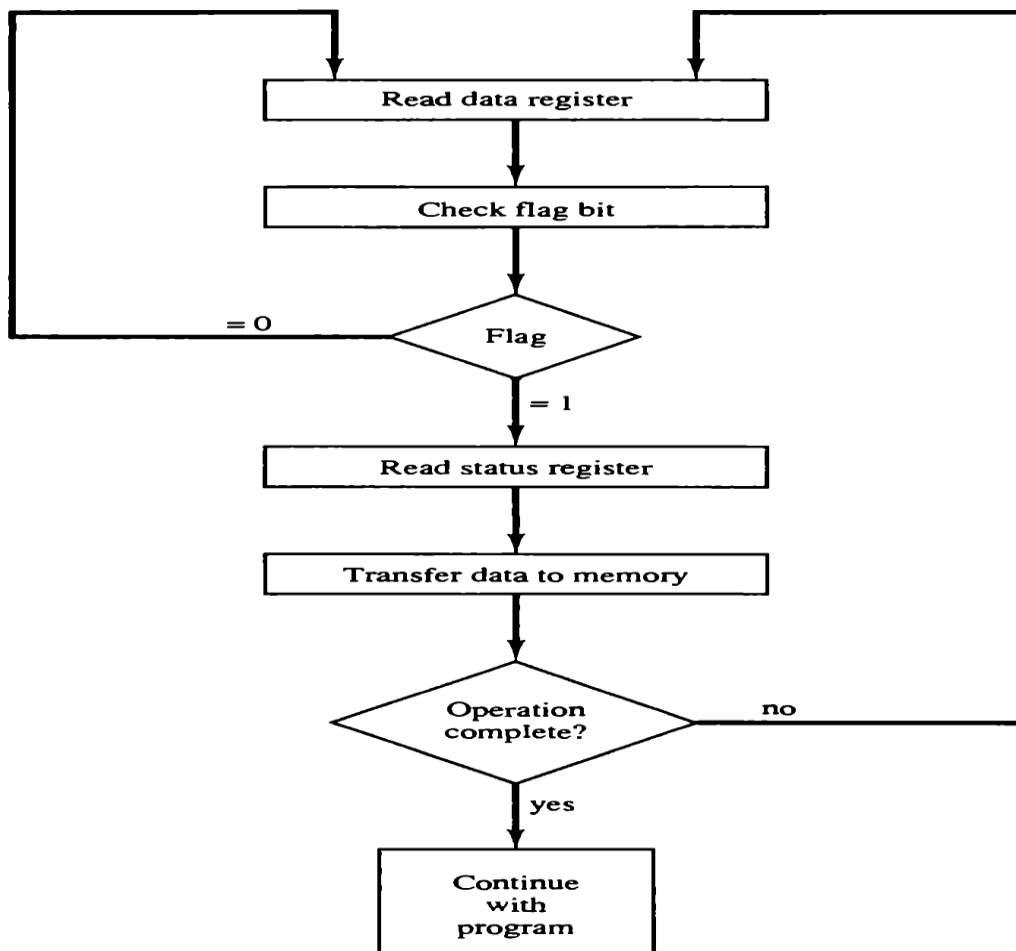
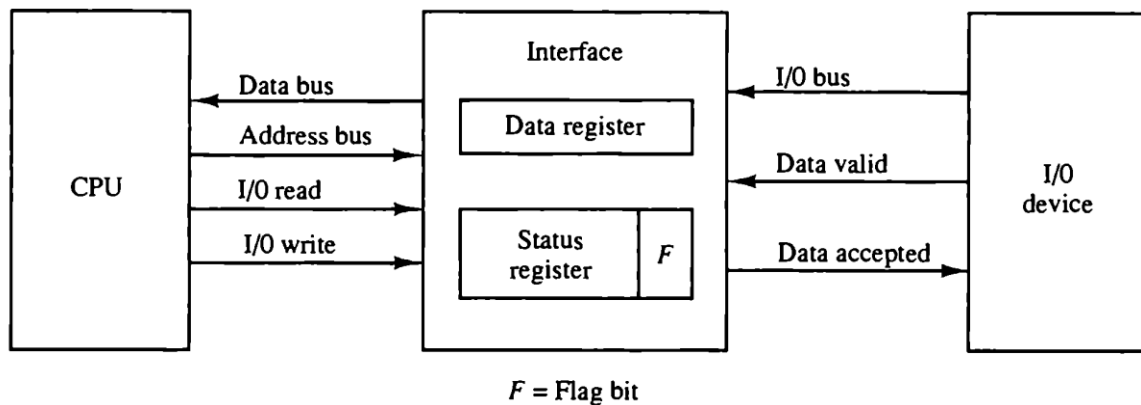
A program is written for the computer to check the flag in the data registered determine if a byte has been placed in the data registered by the device. This is done by reading the status register into a CPU register by checking the value of the flag bit. If the flag is equal to 1, the CPU takes data from the data register.

A flowchart of the program that must be written for the CPU is shown. It is assumed that the device is sending a sequence of bytes that be stored in memory. The transfer of each byte requires three instructions.

1. Read the status register.
2. Check the status of the flag bits and branch to step 1 if not set 0, if set.
3. Read the data register.

The programmed IO method is particularly useful in small low speed computers or in systems that are dedicated to monitor a device continuously.

**Figure 11-10** Data transfer from I/O device to CPU.



## 2) Interrupt Initiated IO:

An alternative to the CPU constantly monitoring the flag is to let the interface inform the computer when it is ready to transfer data. This mode of transfer uses the interrupt facility. While the CPU is running a program, it does not check the flag. However when the flag is set, the

computer is momentarily interrupted from proceeding with the current program and is informed that the flag has been set.

- The CPU responds to the interrupt signal by storing the return address from the program counter into a memory stack and then control branches to service routine that processes the required IO transfer.
- The way that the processor chooses the branch address of the service routine varies from one unit to another.
- In principle there are **two** methods for accomplishing this one is called vectored interrupt and the other non-vectored interrupt.
- In a **non-vectored interrupt**, the branch address is assigned to a fixed location in memory.
- In a **vectored interrupt**, the source that interrupts supplies the branch information to the computer. This information is called the **interrupt vector**.

### 3) DIRECT MEMORY ACCESS(DMA)

- The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. The transfer technique is called **Direct Memory Access(DMA)**. During DMA transfer, the CPU is idle and has no control of the memory buses.
- Two control signals in the CPU that facilitate the DMA transfer, the **bus request(BR)** input is used to relinquish control of the buses. When this input is active, the CPU terminates the execution of the current instruction and places the address buses.
- When this input is active, the CPU terminates the execution of the current instruction and places the address bus, the data bus and the read and write lines into a high impedance state.
- The CPU activates the **bus grant(BG)** output to inform the external DMA that the buses are in the high impedance state.
- In **DMA burst transfer**, a block sequence consisting of a number of memory words is transferred in a continuous burst while the DMA controller is master of the memory buses.
- This mode of transfer is needed for fast devices such as magnetic disks, where data transmission cannot be stopper or slowed down until an entire block is transferred.
- An alternative technique called **cycle stealing**, allows the DMA controller to transfer one data word at a time, after which it must return control of the buses to the CPU.

### **i) DMA Controller:**

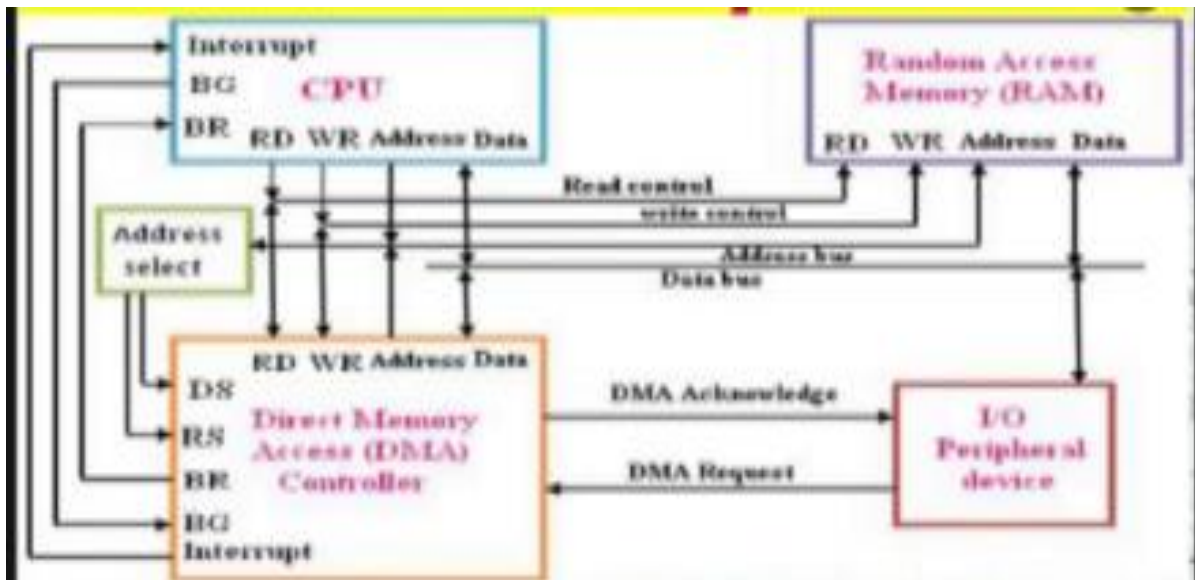
- The DMA controller needs the usual circuits of an interface to communicate with the CPU and IO device.
- The unit communicates with the CPU via the data bus and control lines. The registers in the DMA are selected by the CPU through the address bus by enabling the DS(DMA Select) and RS(Register Select) inputs. The RD(read) and WR(write) inputs are bidirectional.
- The DMA controller has three registers. An address register contains an address to specify the desired location in memory. The address bits go through bus buffers into the address bus. The address register is incremented after each word that is transferred to memory. The control register specifies the mode of transfer. All registers in the DMA appear to the CPU as IO interface registers. Thus the CPU can read from or write into the DMA registers under program control via the data bus.
- The DMA is first initialized by the CPU. After that the DMA starts and continues to transfer data between memory and peripheral unit until an entire block is transferred.
  1. The starting address of the memory block where data are available( for read) or where data are to be stored.
  2. The word count which is the number of words in memory block.
  3. Control to specify the mode of transfer such as read or write.
  4. A control to start the DMA transfer.

### **ii) DMA Transfer:**

- The CPU communicates with the DMA through the address and data buses as with any interface unit.
- The DMA has its own address, which activates the DS and RS lines. The CPU initialize the DMA through the data bus.
- Once the DMA receives the start control command it can start the transfer between the peripheral device and the memory.
- The peripheral device sends a DMA request. The DMA controller activates the BR line, informing the CPU to relinquish the buses.
- The CPU responds with its BG line informing the DMA that its buses are disabled.



- The DMA then puts the current value of its address register into the address bus, initiates the RD or WR signals and sends a DMA acknowledge to the peripheral device.
- For each word that is transferred the DMA increments its address register and decrements its word count register.
- IF the word count does not reach zero the DMA checks the request line coming from the peripheral.



#### 4. PRIORITY INTERRUPT:

Data transfer between the CPU and an IO device is initiated by the CPU. The CPU cannot start the transfer unless the device is ready to communicate with the CPU. The readiness to the device can be determined from an interrupt signal. The CPU responds to the interrupt request by storing the return address from PC into a memory stack and then the program branches to a service routine that processes the required transfer.

In a typical application, a number of IO devices are attached to the computer, with each device being able to originate an interrupt request. The first task of the interrupt system is to identify the source of the interrupt. There is also the possibility that several sources will request service simultaneously. In this case the system must also decide which device to service first.

- Devices with high speed transfer like magnetic disks are given high priority.
- Devices like keyboards are given low priority.

- Polling procedure is used to identify the highest priority source by software means.
- Highest priority source is tested first, if the interrupt signal is ON, otherwise next lower priority source is tested.

Disadvantage is if there are many interrupts, the time required to poll them can exceed the time available to service the IO device. So hardware priority interrupt unit can be used.

**a) Daisy Chaining priority:**

- It consists of serial connections of all devices that request an interrupt,
- Device with the highest priority is placed in the first position.
- Lowest priority is placed in the last position.
- CPU responds by enabling the interrupt acknowledge line. Signal received by 1 at its Priority in(Pi). Ack signal passes on to the next device by Priority Out (Po). If pending interrupt, it blocks the acknowledge signal from next device by placing a 0 in the Po output.
- It uses a register whose bits are set separately by the interrupt signal.
- A high priority device interrupt the CPU while a lower priority device is being serviced.
- It consist of an interrupt register, individual bits are set by external conditions and cleared by program instructions.

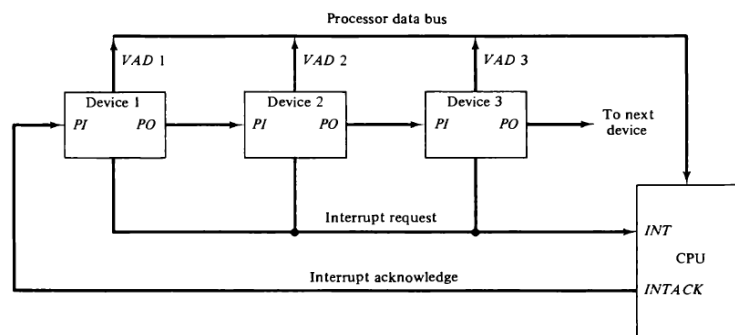


Figure 11-12 Daisy-chain priority interrupt.

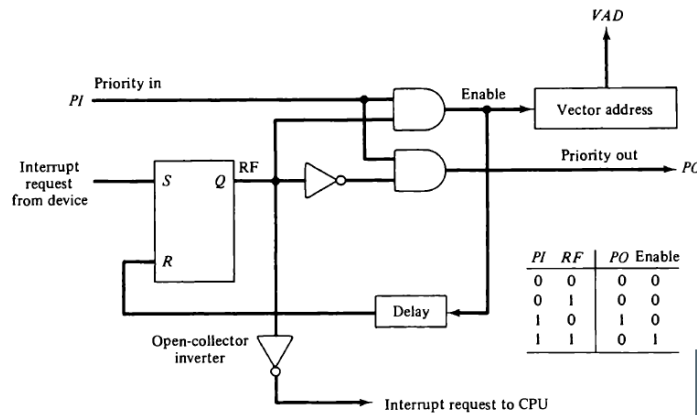


Figure 11-13 One stage of the daisy-chain priority arrangement.

Activ

### b) Parallel Priority Interrupt:

- Magnetic disk has highest priority.
- Printer has next priority followed by character reader and a keyboard.
- Mask register has same number of bits as the interrupt register.
- By program instructions, possible to set or reset any bit in the mask register.
- Interrupt bit and mask bit are applied to an AND gate to produce four inputs to the priority encoder.
- IST(Interrupt Status) and IEN(Interrupt Enable) provide a common interrupt signal for the CPU.

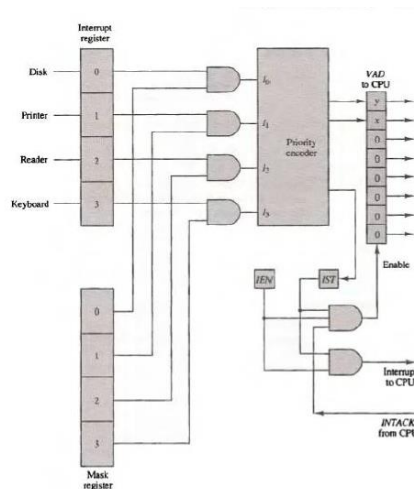
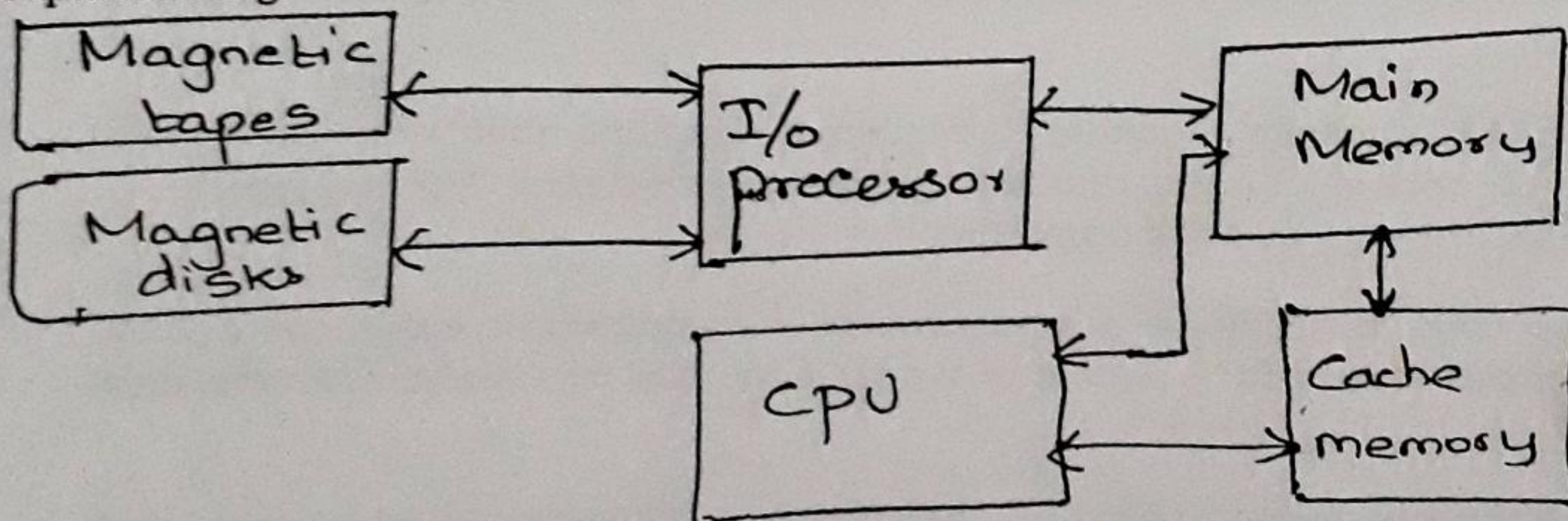


Figure 11-14 Priority interrupt hardware.



## 1. Memory Hierarchy

- The memory unit that communicates directly with the CPU is called the **main memory**.
- Devices that provide the backup storage are called <sup>Secondary</sup> **auxiliary memory**.
- The total memory capacity of a computer can be visualized as being a hierarchy of components.
- 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 speed processing logic.  
*auxiliary → main → cache*
- The **main memory** occupies a central position by being able to communicate with directly the CPU and with auxiliary memory devices through an I/O processor.
- When programs not residing in main memory are needed by the CPU, they are brought in from **auxiliary memory**.
- 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. Fig.



- 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.
- The I/O processor manages data transfers between auxiliary memory and main memory, the cache organization is connected with the transfer of information between main memory and CPU.
- The **auxiliary memory** has a large storage capacity, is relatively inexpensive, but has low access speed compared to main memory.
- The cache memory is very small, relatively expensive, and has very high access speed.
- Auxiliary and cache memories are used for different purposes. The cache holds those parts of the program and data that are most heavily used, while the auxiliary memory holds those parts that are not presently used by the CPU.
- Moreover, the CPU has direct access to both cache and main memory but not to auxiliary memory.
- Auxiliary memory average access time is usually 1000 times that of main memory.



- Block size in auxiliary memory typically ranges from 256 to 2048 words, while cache block size is typically from 1 to 16 words.
- Many operating systems are designed to enable the CPU to process a number of independent programs concurrently. This concept, called **multiprogramming**, refers to the existence of two or more programs in different parts of the memory hierarchy at the same time.
- The part of the computer system that supervises the flow of information between auxiliary memory and main memory is called the **memory management system**.

## 2. Main Memory

- The 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 semiconductor integrated circuits.
- Integrated circuit **RAM** chips are available in two possible operating modes, static and dynamic.
- The **static RAM** consists essentially of internal flip-flops that store the binary information of electric charges that are applied to capacitors. The capacitors are provided inside the chip by MOS transistors.
- 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.
- Originally, **RAM** was used to refer to a random-access memory, but now it is used to designate a read/write memory to distinguish it from a read-only memory, although ROM is also random access that are subject to change.
- **ROM** is used for storing 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**.
- The bootstrap loader is a program whose function is to start the computer software operating when power is turned on.
- Since RAM is volatile, its contents are destroyed when power is turned off.
- The contents of ROM remain unchanged after power is turned off and on again.

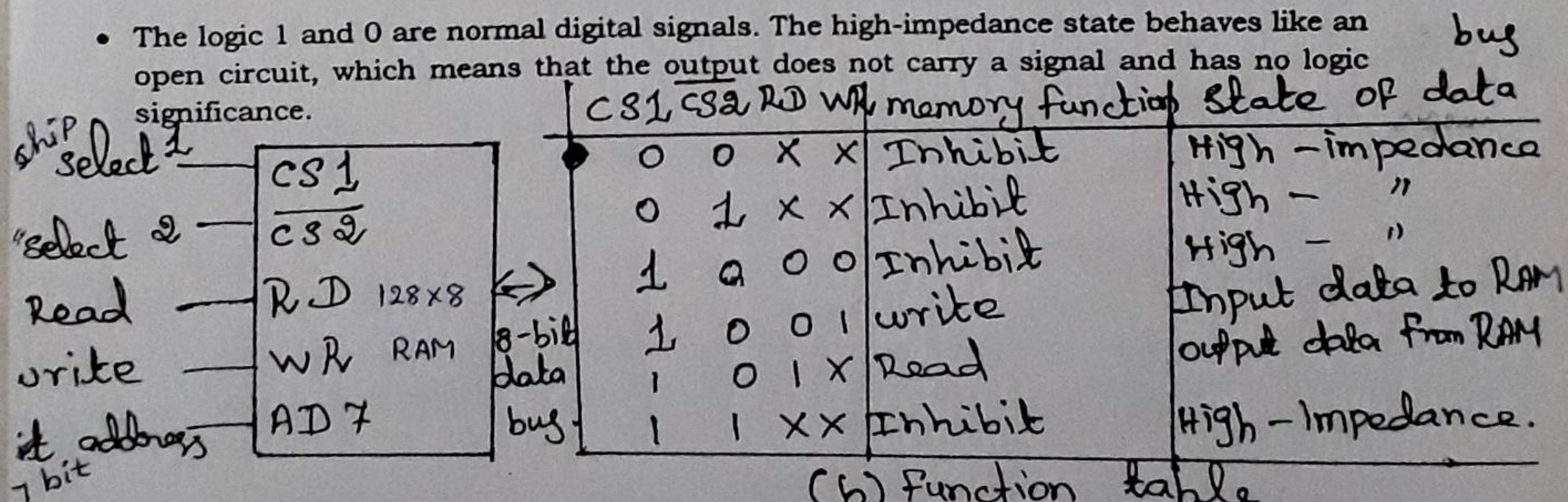


- The startup of a computer consists of Turning the power on and starting the execution of an initial program.
- Thus when power is turned on, the hardware of the computer sets the program counter to the first address of the bootstrap loader.
- The **bootstrap program** loads a portion of the operating system from disk to main memory and control is then transferred to the operating system, which prepares the computer for general use.

#### a) RAM and ROM chips

- A RAM chip is better suited for communication with the CPU if it has one or more control inputs that select the chip only when needed.
- Another common feature is a bidirectional data bus that allows the transfer of data either from memory to CPU during a read operation or from CPU to memory during a write operation.
- A bidirectional bus can be placed in one of **three-state buffers**. A three-state buffer output can be placed in one three possible states: a signal equivalent to logic 1, a signal equivalent to logic 0, or a high-impedance state.

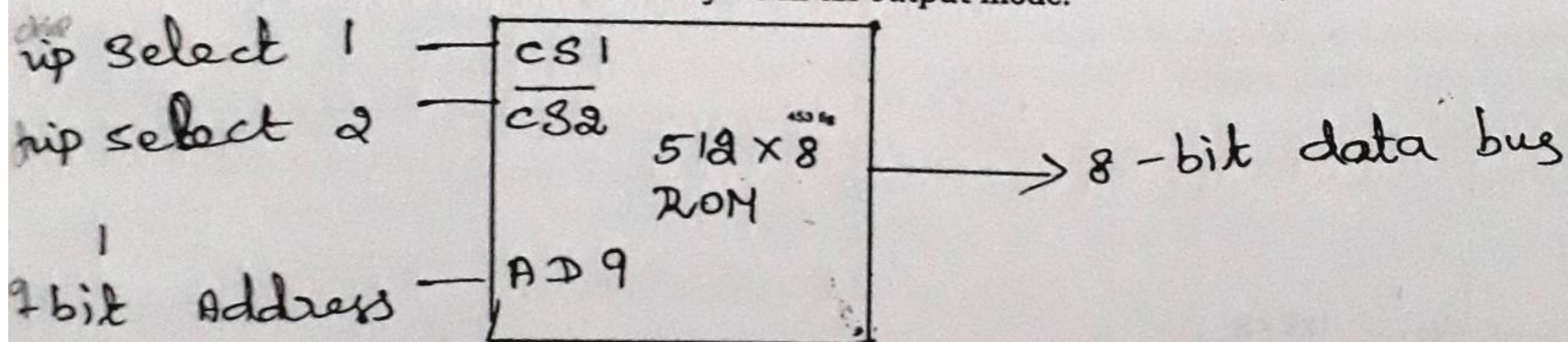
- The logic 1 and 0 are normal digital signals. The high-impedance state behaves like an open circuit, which means that the output does not carry a signal and has no logic significance.



- The block diagram of a RAM chip is shown above. The capacity of the 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 for enabling the chip only when it is selected by the microprocessor.
- The availability of more than one control input to select the chip facilitates the decoding of the address lines when multiple chips are used in the microcomputer.
- The read and write inputs are sometimes combined into one line labeled R/W.



- When the chip is selected two binary states in this line specify the two operations of read or write.
- The function table listed above specifies the operation of the RAM chip.
- The unit is an 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 operations as well as the bus buffers associated with the bidirectional data bus.
- 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. For the same-size chip, it is possible to have more bits of ROM than of RAM, because the internal binary cells in ROM occupy less space than in RAM.
- For this reason, the diagram specifies a 512-bytes ROM, while the RAM has only 128 bytes. The nine address lines in the ROM chip specify any one of the 512 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. There is no need for a read or write control because the unit can only read.
- Thus when the chip is enabled by the two select by the address lines appears on the data bus.



## b)Memory Address Map:

- The designer of a computer system must calculate the amount of memory required for the particular application and assign it to either RAM or ROM.
- 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 specifies the memory address assigned to each chip.
- The table, called a **memory address map**, is a pictorial representation of assigned address space for each chip in the system.
- To demonstrate with a particular example, assume that a computer system needs 512 bytes of RAM and 512 bytes of ROM.

- The RAM and ROM chips to be used are specified in fig. The memory address map for this configuration is shown in Table.

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 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. Although there are 16 lines in the address bus, the table shows only 10 lines because the other 6 are not used in this example and are assumed to be zero.
- The small x's under the address bus lines designated those lines that must be connected to the address inputs in each chip.
- The RAM chips have 128 bytes and need 7 address lines. The ROM chip has 512 bytes and needs 9 address lines.
- The x's are always assigned to the low-order bus lines: lines 1 through 7 for the RAM and lines 1 through 9 for the ROM. It is now necessary to distinguish between four RAM chips by assigning to each a different address.
- For this particular example we choose bus lines 8 and 9 to represent four distinct binary combinations. Note that any other pair of unused bus lines can be chosen for this purpose.



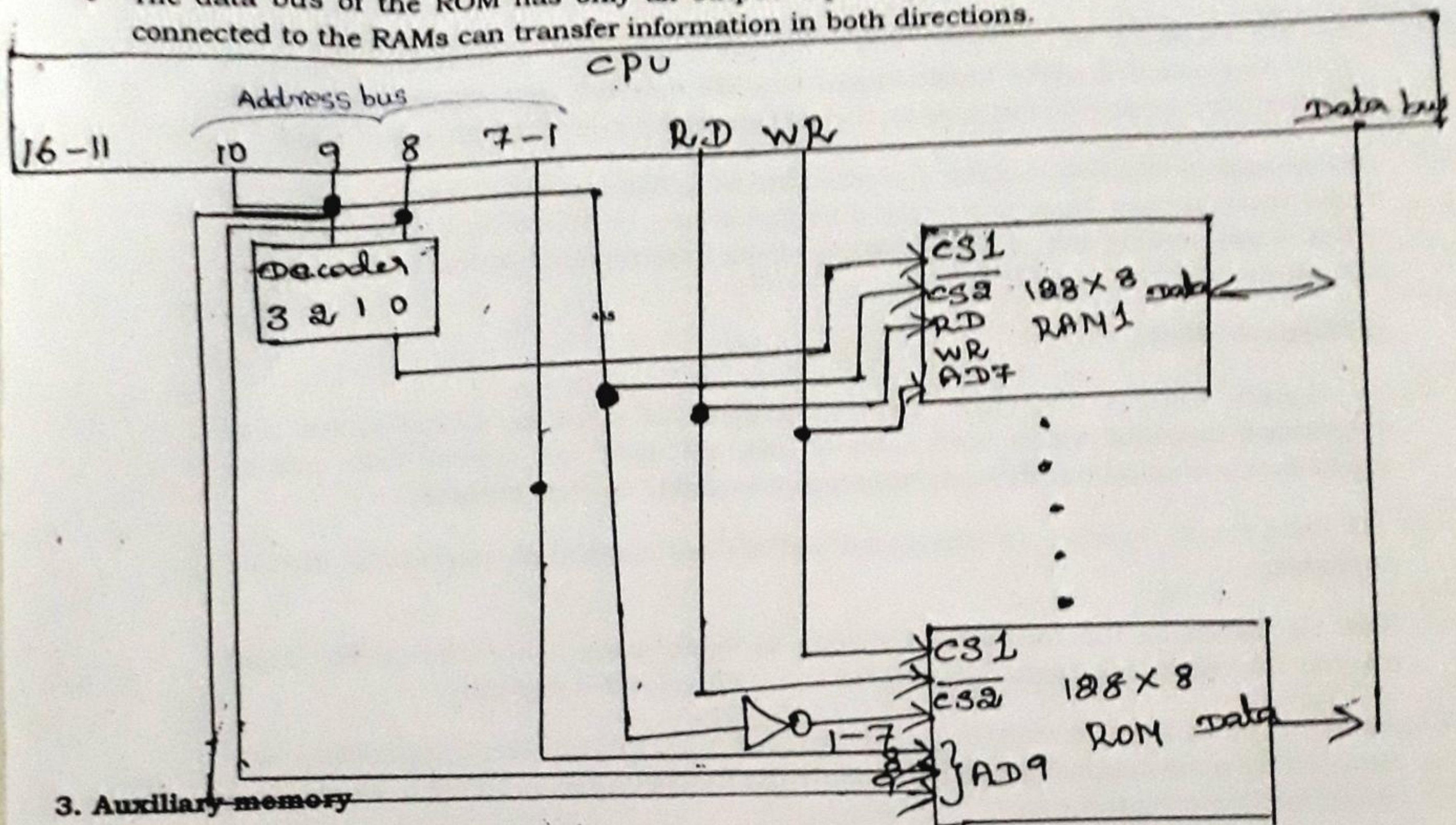
- The table clearly shows that the nine low-order bus lines constitute a memory space for RAM equal to  $2^9=512$  bytes. The distinction between a RAM and ROM address is done with another bus line.
- Here we chose line 10 for this purpose. When line 10 is 0, the CPU selects a RAM, and when this line is equal to 1, selects the ROM. The equivalent hexadecimal address for each chip obtained from the information under the address bus assignment.
- The address bus lines are subdivided into **groups of four bits** each so that each group can be represented with a hexadecimal digit.
- The first hexadecimal digit represents lines 13 to 16 and is always 0. The next hexadecimal digit represents lines 9 to 12, but lines 11 and 12 are always 0.
- The range of hexadecimal address for each component is determined from the x's associated with it. These x's represent a binary number that can range from an all -0's to an all -1's value.

### c) 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.
- The connection of memory chips to the CPU is shown in fig. This configuration gives a memory capacity of 512 bytes of RAM and 512 bytes of ROM.
- It implements the memory map of table. 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  $2 \times 4$  decoder whose outputs go to the CS1 inputs in each RAM chip.
- Thus, when 01, the second RAM chip is selected, 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.
- The other chip select input in the ROM is connected to the RD control line for the ROM chip select to be enabled only during a read operation.
- Address bus lines 1 to 9 are applied to the input address of ROM without going through the decoder. This assigns address 0 to 511 to RAM and 512 to 1023 to ROM.



- 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.



### 3. Auxiliary memory

- The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. Other components used, but not as frequently, are magnetic drums, magnetic bubble memory, and optical disks.
- The important characteristics of any device are its access mode, access time, transfer rate, capacity, and cost.
- The average time required to reach a storage location in memory and obtain its contents is called the **access time**. The access time consists of a **seek time** required to position the read-write head to a location and a **transfer time** required to transfer data to or from the device, because the seek time is usually much longer than the transfer time, auxiliary storage is organized in records or blocks.
- A record is a specified number of characters or words. Reading or Writing is always done on entire records. The **transfer rate** is the number of characters or words that the device can transfer per second, after it has been positioned at the beginning of the record.
- Magnetic drums and disks are quite similar in operation. Both consist of high-speed rotating surfaces coated with a magnetic recording medium. The rotating surface of the drum is a **cylinder** and that of the disk, a round flat plate. The



recording surface rotates at uniform speed and is not started or stopped during access operations.

- Bits are recorded as magnetic spots on the surface as it passes a stationary mechanism called a **write head**. Stored bits are detected by a change in magnetic field produced by a recorded spot on the surface as it passes through a **read head**.
- The amount of surface available for recording in a disk is greater than in a Drum of equal physical size. Therefore, more information can be stored on a disk than on a drum of comparable size. For this reason, disks have replaced drums in more recent computers.

#### **a) Magnetic disks**

- A Magnetic disk is a circular plate constructed of metal or plastic coated with magnetized material. Often both sides of disk are used and several disks may be stacked on one spindle with read/write heads available on each surface.
- All disks rotate together at high speed and are not stopped or started for accesses purposes.
- Bits are stored in the magnetized surface in spots along concentric circles called **tracks**. The tracks are commonly divided into sections called **sectors**.
- Some units use a single read/write head for each disk surface. The track address bits are used by a mechanical assembly to move the head into the specified track position before reading or writing.
- Permanent timing tracks are used in disks to synchronize the bits and recognize the sectors.
- Disks may have multiple heads and simultaneous transfer of bits from several tracks at the same time.
- If bits are recorded with equal density, some tracks will contain more recorded bits than others.
- Disks are permanently attached to a unit assembly and cannot be removed by the occasional user called hard disks. A disk drive with removable disks is called a floppy disk.
- There are two sizes commonly used with diameters of 5.25 and 3.5 inches.

#### **b) Magnetic Tape:**

- A magnetic tape consists of the electrical, mechanical and electronic components to provide the parts and control mechanism for a magnetic tape unit.
  - The tape itself is a strip of plastic coated with a magnetic recording medium. Bits are
-



- recorded as magnetic spots on the tape along several tracks.
- Magnetic tape units can be stopped, started to move forward or in reverse or can be rewound.
- Information is recorded in blocks referred to as records. Gaps of unrecorded tape are inserted between records where the tape can be stopped.
- The tape starts moving while in a gap and attains its constant speed by the time it reaches the next record.
- Each record on tape has an identification bit pattern at the beginning and end. By reading the bit pattern at the beginning, the tab control identifies the record number.
- By reading the bit pattern at the end of the record, the control recognizes the beginning of the gap.
- A tape unit is addressed by specifying the record number and the number of characters in the record. Records may be of fixed or variable length.

#### 4. Associative Memory 5m

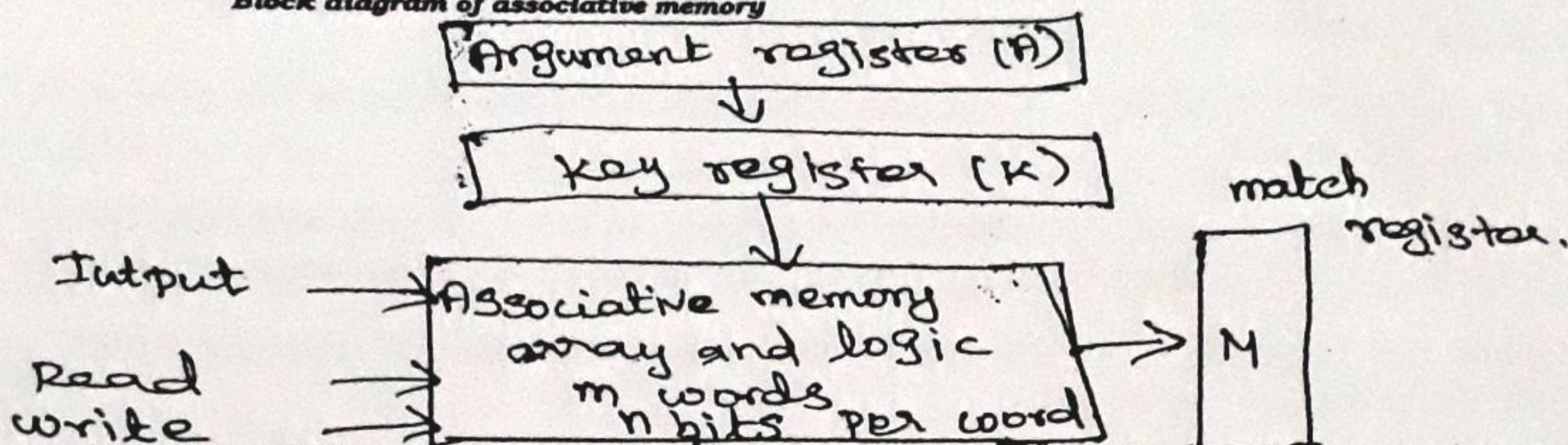
- Many data-processing applications require the search of items in a table stored in memory. An assembler program searches the symbol address table in order to extract the symbol's binary equivalent.
- An account number may be searched in a file to determine the holder's name and account status. The established way to search a table is to store all items where they can be addressed in sequence.
- The search procedure is a strategy for choosing a sequence of addresses, reading the content of memory at each address, and comparing the information read with the item being searched until a match occurs.
- The number of accesses to memory depends on the location of the item and the efficiency of the search algorithm. Many search algorithms have been developed to minimize the number of accesses while searching for an item in a random or sequential access 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)**.
- This type of memory is accessed simultaneously and in parallel on the basis of the data content rather than by specific address or location. When a word is written in an associative memory, no address is given.
- The memory is capable of finding an empty unused location to store the word. When a word is to be read from an associative memory, the content of word, or part of the word, is specified.
- The memory locates all words which match the specified content and marks them for reading.



### a) Hardware Organization

- The block diagram of an associative memory is shown in fig. 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.

Block diagram of associative memory

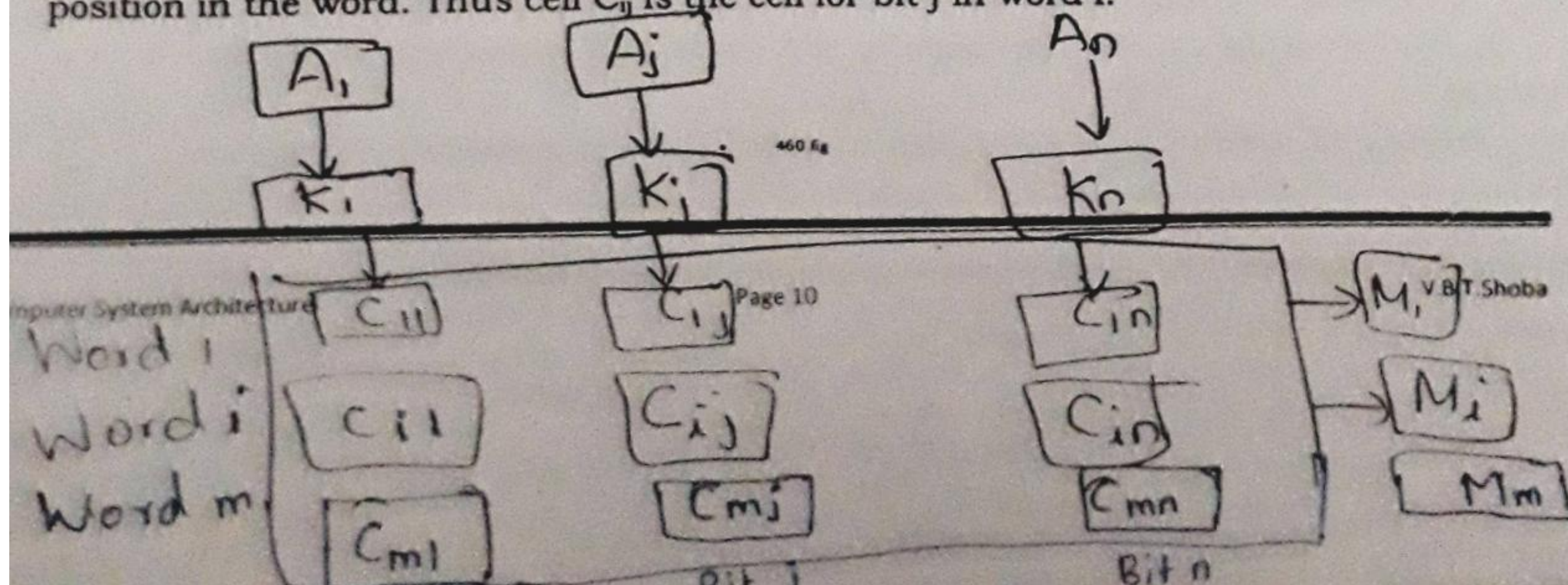


- The match register  $M$  has  $m$  bits, one for each memory word. Each word in memory is compared in parallel with the content of the argument register. The words that match the bits of the argument register set a corresponding bit in the match register.
- After the matching process, those bits in the match register that can be set 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.
- The key register provides a mask for choosing a particular field or key in the argument word. The entire argument is compared with each memory word if the key register contains all 1's.
- Otherwise, only those bits in the argument that have 1's in their corresponding position of the key register are compared. Thus the key provides a mask identifying piece of information which specifies how the reference to memory is made.
- To illustrate with the numerical example, suppose that the argument register  $A$  and the key have the bit configuration as given below; Only the three leftmost bits of  $A$  are compared with memory words because  $K$  has 1's in these positions.

$A$                     101 111100  
 $K$                     111 000000  
 Word 1            100 111100 no match  
 Word 2            101 000001 match

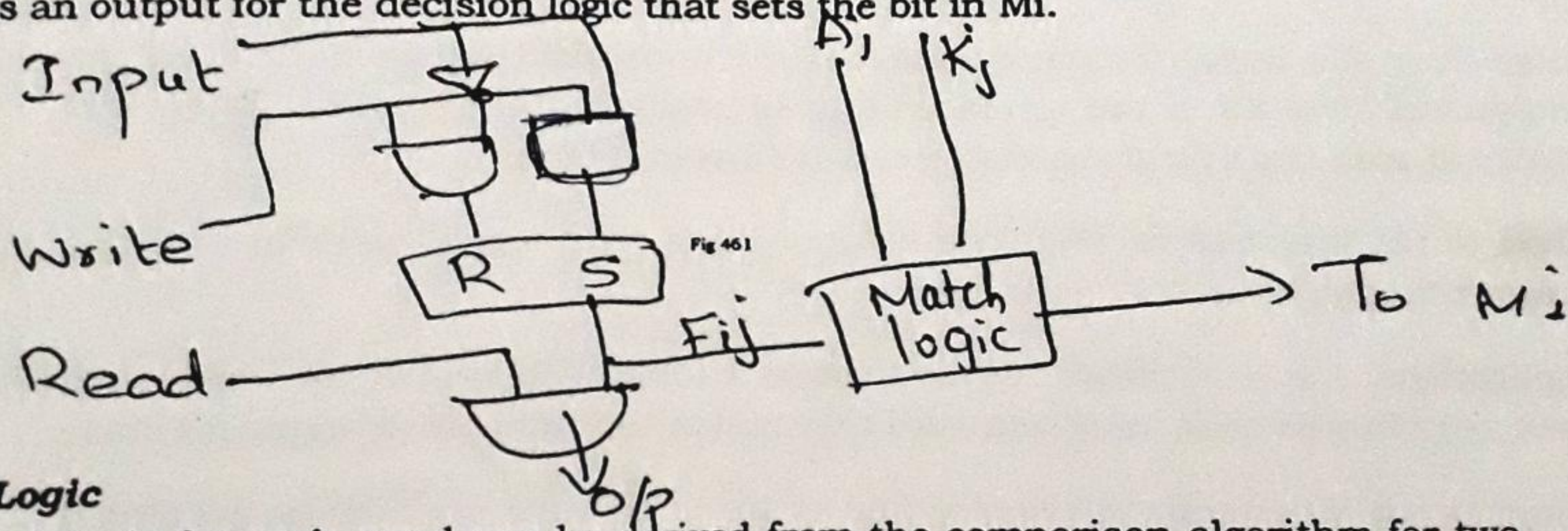
Word 2 matches the unmasked argument field because the three leftmost bits of the argument and the word are equal.

The relation between the memory array and the external registers in an associative memory shown in fig. The cells in the array are marked by the letter  $C$  with two subscripts. The first subscript gives the word number and the second specifies the bit position in the word. Thus cell  $C_{ij}$  is the cell for bit  $j$  in word  $i$ .





- A bit  $A_j$  in the argument register is compared with all the bits in column  $j$  of the array provided that  $K_j=1$ . This is done for all columns  $j=1,2,\dots,n$ .
- If a match occurs between all the unmasked bits of the argument and the bits in word  $i$ , the corresponding bit  $M_i$  in the match register is set to 1. If one or more unmasked bits of the argument and the word do not match,  $M_i$  is cleared to 0.
- The internal organization of a typical cell  $C_{ij}$  is shown in fig below. It consists of a flip-flop storage element  $F_{ij}$  and the circuits for reading, writing and matching the cell. The input bit is transferred in to 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 the corresponding unmasked bit of the argument and provides an output for the decision logic that sets the bit in  $M_i$ .



#### b) Match Logic

- The match logic for each word can be derived from the comparison algorithm for two binary numbers. First, we neglect the key bits and compare the argument in  $A$  with the bit stored in the words.
- Word  $i$  is equal to the argument in  $A$  if  $A_j=F_{ij}$  for  $j=1,2,\dots,n$ . Two bits are equal if they are both 1 or both 0. The equality of two bits can be expressed logically by the Boolean function

$$X_j = A_j F_{ij} + A_j' F_{ij}'$$

Where  $X_j=1$  if the pair of bits in position  $j$  are equal; otherwise,  $X_j=0$ .

- For a word  $i$  to be equal to the argument in  $A$  we must have all  $X_j$  variables equal to 1. This is the condition for setting the corresponding match bit  $M_i$  to 1. The Boolean function for this condition is

$$M_i = X_1 X_2 X_3 \dots X_n$$

And constitutes the AND operation of all pairs of matched bits in a word.

- We now include the key bits  $K_j$  in the comparison logic. The requirement is that if  $K_j=0$ , the corresponding bits of  $A_j$  and  $F_{ij}$  need no comparison. Only when  $K_j=1$  must they be compared. This requirement is achieved by ORing each term with  $K_j'$ , thus:

$$X_j = K_j' + \{X_j \text{ if } K_j=1, 1 \text{ if } K_j=0\}$$



- When  $K_j=1$ , we have  $K'_j=0$  and  $x_j+0=x_j$ . When  $K_j=0$ , then  $K'_j=1$  and  $x_j+1=1$ . A term  $(x_j+K'_j)$  will be in the 1 state, if its pair of bits is not compared. This is necessary because each term is ANDed with all other terms so that an output of 1 will have no effect. The comparison of the bits has an effect only when  $K_j=1$ .
- The match logic for word  $i$  in an associative memory can now be expressed by the following Boolean function:
  - $M_i=(x_1+K'_1)(x_2+K'_2)(x_3+K'_3).....(x_n+K'_n)$
- Each term in the expression will be equal to 1 if its corresponding  $K_j=0$ . If  $K_j=1$ , the term will be either 0 or 1 depending on the value of  $x_j$ . A match will occur and  $M_i$  will be equal to 1 if all terms are equal to 1.
- If we substitute the original definition of  $x_j$  the Boolean function above can be expressed as follows:  $M=\prod(A_jF_{ij}+A'_jF'_{ij}+K'_j)$ 
  - Where  $\prod$  is a product symbol designating the AND operation of all  $n$  terms. We need  $m$  such functions, one for each word  $i=1,2,3.....,m$ .

### c) 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. It is then necessary to scan the bits of the match register one at a time.

The matched words are read in sequence by applying a read signal to each word line whose corresponding  $M_i$  bit is a 1.

In most applications, the associative memory stores a table with no two identical items under a given key. In this case, only one word may match the unmasked argument field.

By connecting output  $M_i$  directly to the read line in the same word position, the content of the matched word will be presented automatically at the output lines and no special read command signal is needed.

Furthermore if we exclude words having zero content, an all-zero output will indicate that no match occurred and that the searched item is not available in memory.

### d) Write operation

- An associative memory must have a write capability for storing the information to be searched. Writing in an associative memory can take different forms, depending on the application. If the entire memory is loaded with new information at once prior to a search operation then the writing can be done by addressing each location in sequence.
- This will make the device a random access memory for writing and a content addressable memory for reading. The advantage here is that the address for input can be decoded as in a random-access memory. Thus instead of having  $m$  address lines can be reduced by the decoder to  $d$  lines, where  $m=2^d$ .



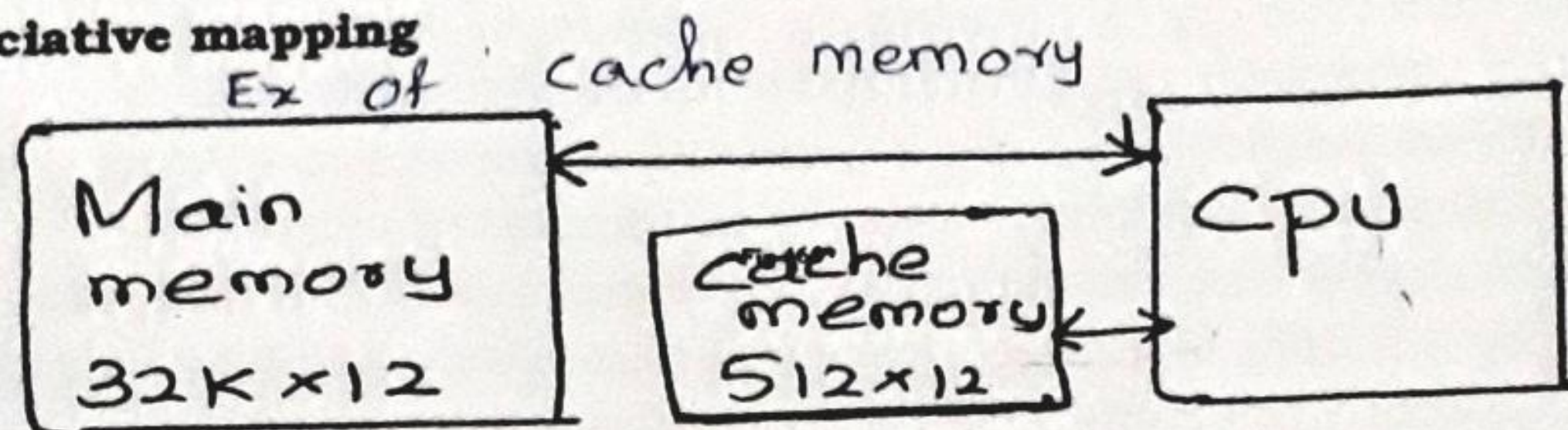
- If unwanted words have to be deleted and new words inserted one at a time, there is a need for a special register to distinguish between active and inactive words. This register, sometimes called a **tag register**, would have as many bits as there are words in the memory.
- For every active word stored in memory, the corresponding bit in the tag register is set to 1. A word is deleted from memory by clearing its tag bit to 0. Words are stored in memory by scanning the tag register until the first 0 bit is encountered.
- This gives the first available inactive word and a position for writing a new word. After the new word is stored in memory it is made active by setting its tag bit to 1.
- An unwanted word when deleted from memory can be cleared to all 0's if this value is used to specify an empty location. Moreover, the words that have a tag bit of 0 must be masked with the argument word so that only active words are compared.)

### 5. Cache Memory: 5m

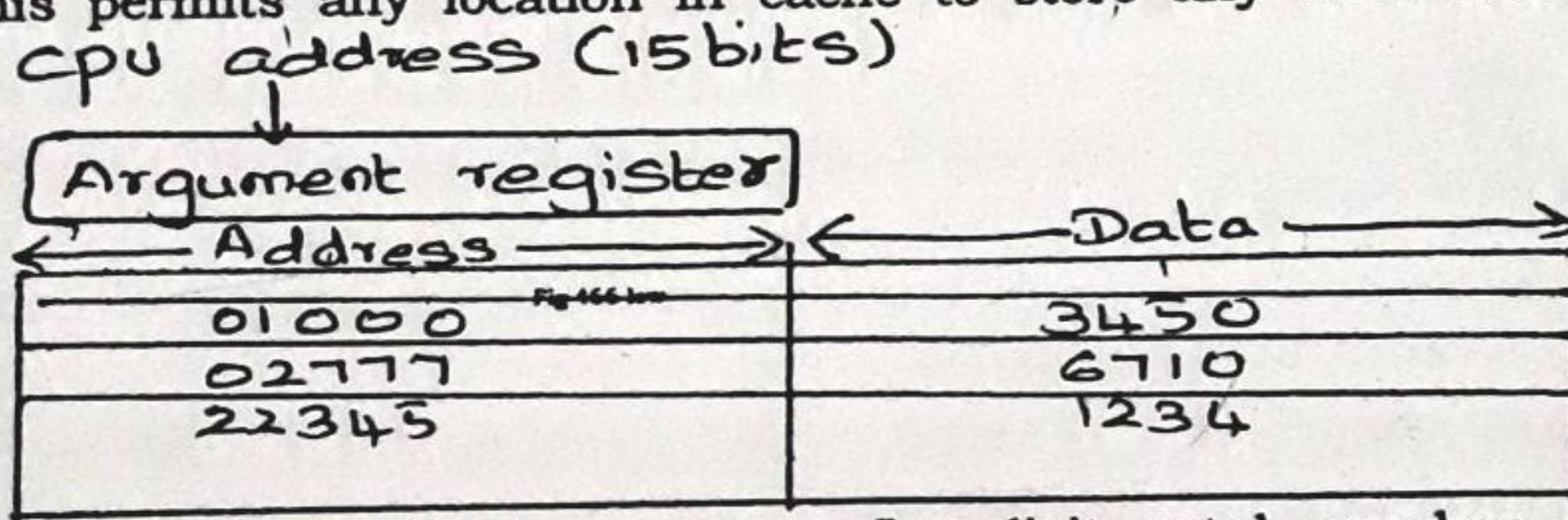
- Analysis of a large number of typical programs show that the references to memory at any given interval of time tend to be confined within a few localized areas in memory. This phenomenon is known as the property of **locality of reference**.
- When a program loop is executed, the CPU repeatedly refers to the set of instructions are fetched from memory. Thus loops and subroutines tend to localize the references to memory for fetching instructions.
- Iterative procedures refer to common memory locations and array of numbers is confined within a local portion of memory.
- 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 cache memory.
- It is placed between the CPU and main memory. The cache memory access time is less than the access time of main memory by a factor of 5 to 10. The cache is the fastest component in the memory hierarchy.
- Although the cache is only a small fraction of the size of main memory.
- The **basic operation** is when the CPU needs to access memory, the cache is examined. If the word is found in the cache, it is read from the fast memory. A block of words containing the one just accessed is then transferred from main memory to cache memory.
- The performance of cache memory is frequently measured in terms of a quantity called **hit ratio**. When the CPU refers to memory and finds the word in cache it is said to produce a **hit**. If the word is not found in cache, it is in main memory and it counts as a **miss**.
- The ratio of the number of hits divided by the total CPU references to memory is the **hit ratio**. Hit ratios of 0.9 and higher have been reported.
- The average memory access time of a computer system can be improved by the use of cache. A computer with cache access time of 100ns, a main memory access time of 1000ns and a hit ratio of 0.9 produce an average access time of 200ns.



- The basic characteristic of cache memory is its fast access time. The transformation of data from main memory to cache memory is referred to as a **mapping process**.
- Three types of mapping procedures are
  1. Associate mapping
  2. Direct mapping
  3. Set associative mapping



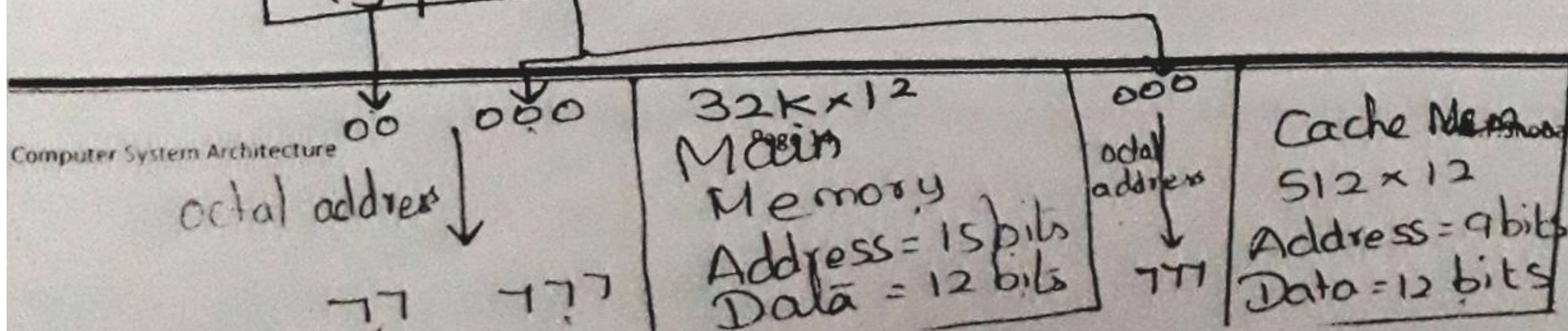
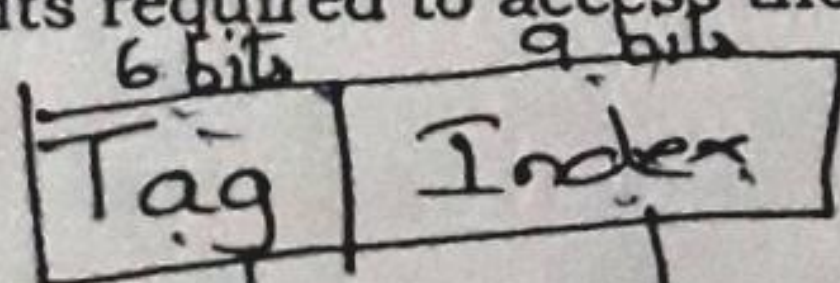
- The main memory can store 32K words of 12 bits each. The cache is capable of storing 512 of these words at any given time. For every word stored in cache, there is a duplicate copy in main memory.
- Associate mapping:** The associative memory stores both the address and data of the memory word. This permits any location in cache to store any word from main memory.



- The address value of 15 bits is shown as a five digit octal number and its corresponding 12-bit word 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 the 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.
- If the cache is full, an address-data pair must be displaced to make room for a pair is replaced is determined. To replace cells of the cache in round robin order whenever a new word is requested from main memory.

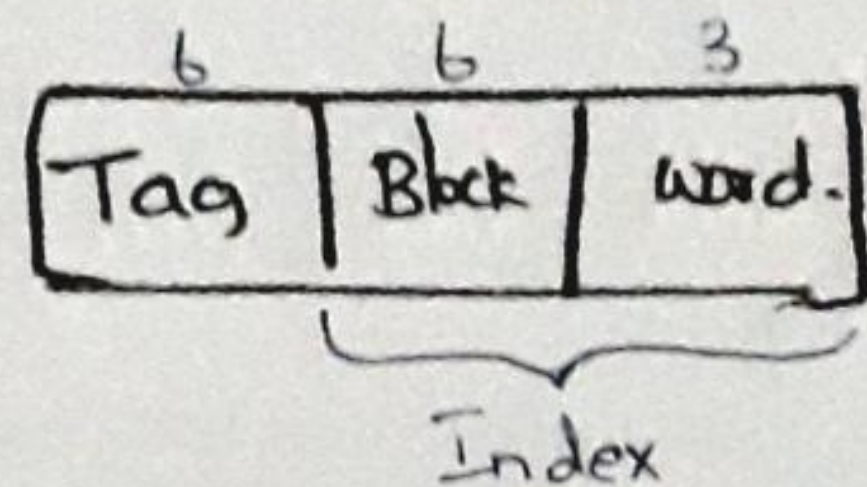
- Direct Mapping:** Associative memories are expensive compared to random access <sup>RAM</sup> memories because of the added logic associated with each cell. The CPU address of 15 bits is divided into **two** fields.

- The nine least significant bits constitute the **index** field and the remaining six bits form the **tag** field. The number of bits in the index field is equal to the number of address bits required to access the cache memory.





Mem. Add	Memory data	Index	Tag	Data
00000	1220	000	01	3450
00111	2340	001	01	6578
01000	3450	010		
02111	6710	011		
		110	02	
		111	02	6710



- 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 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 required word is read from main memory.
  - The **disadvantage** of direct mapping is that the hit ratio can drop considerably if two or more words whose addresses have the same index but different tags are accessed repeatedly.
  - The index field is now divided into two parts: the **block field** and the **word field**. In a 512 word cache there are 64 blocks of 8 words each, since  $64 \times 8 = 512$ . The block number is specified with a 6 bit field and the word within the block is specified with a 3 bit field.
  - The tag field stored within the cache is common to all eight words of the same block. Every time a **miss** occurs, an entire block of eight words must be transferred from main memory to cache memory.
- c) Set-Associative Mapping:** It is an improvement over the direct mapping organization in that each word of cache can store two or more words of memory under the same index addresses. Each data word is stored together with its tag and the number of tag data items in one word of cache is said to form a set.
- Each index address refers to two data words and their associated tags. Each tag requires six bits and each data word has 12 bits, so the word length is  $2(6+12)=36$  bits. An index address of nine bits can accommodate 512 words. Thus the size of cache memory is  $512 \times 36$ .

Index	Tag 6bits	Data 12bits	Tag	Data
000	01	3450	02	5670
777	02	6710	00	2345

- The words stored at addresses 01000 and 02000 of main memory are stored in cache memory at index address 000. Similarly the words at addresses 02777 and 00777 are stored in cache at index address 777.
- When the CPU generates a memory request, the index value 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.
- When a miss occurs in a set-associative cache and the set is **full**, it is necessary to replace one of the tag data items with a new value. The most common replacement algorithms used are: random replacement, first in first out, Least Recently Used.



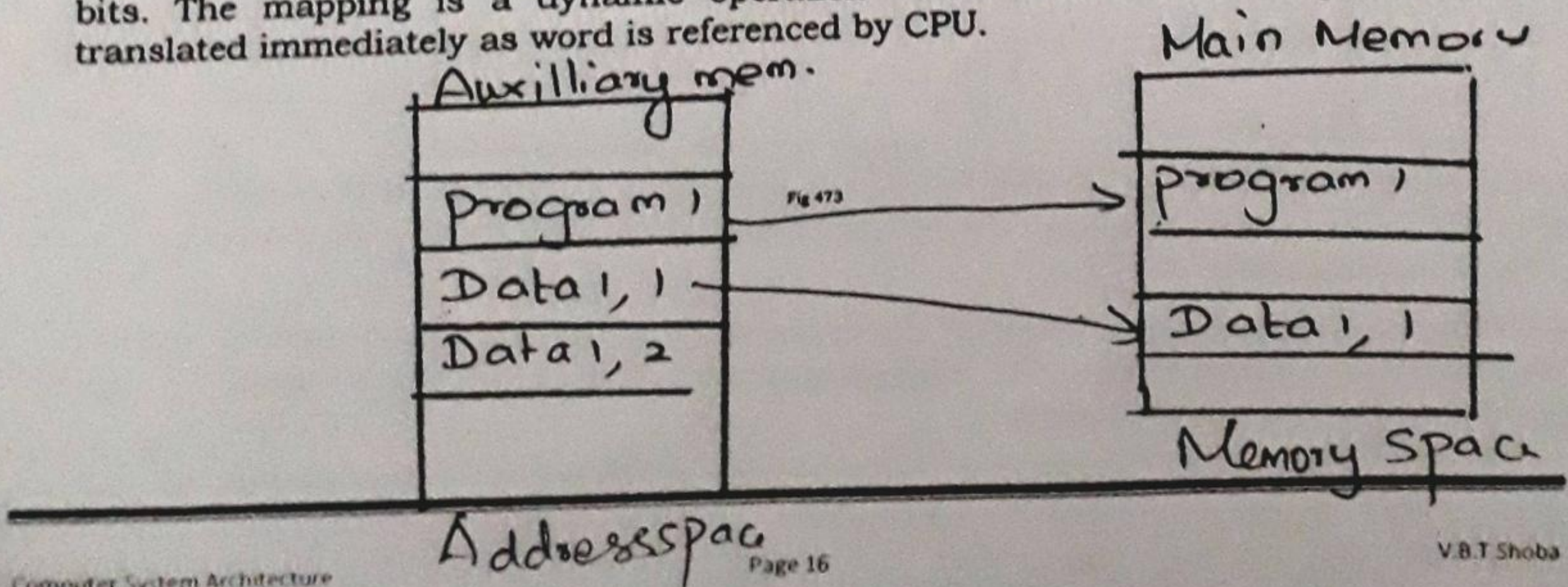
- Both FIFO and LRU can be implemented by adding a few extra bits in each word of cache.
- When the CPU finds a word in cache during a read operation, the main memory is not involved in the transfer. The simplest and most commonly used procedure is to update main memory with every memory write operation, with cache memory being updated in parallel if it contains the word at the specified address. This is called the **write-through method**.
- The method in which only the cache location is updated during a write operation. The location is then marked by a flag so that later when the word is removed from the cache it is copied into main memory is called **write-back method**.
- The cache is initialized when power is applied to the computer or when the main memory is loaded with a complete set of programs from auxiliary memory. After initialization the cache is considered to be empty. It is customary to include with each word in cache a valid bit to indicate whether or not the word contains valid data.)

6. **Virtual Memory:** is a concept used in some large computer systems that permit the user to construct programs as though a large memory space were available, equal to the totality of auxiliary memory.

- Each address that is referenced by the CPU goes through an address mapping from the so called virtual address to a physical address in main memory.
- A virtual memory system provides a mechanism for translating program generated addresses into correct main memory locations.

a) **Address space and Memory space:** An address used by a programmer will be called a virtual address and the set of such addresses the address space. An address in main memory is called a location or physical address.

- The set of such locations is called the memory space. The address space is allowed to be larger than the memory space in computers with virtual memory
- In a virtual memory system, the address field of the instruction code has a sufficient number of bits to specify all virtual addresses. In our example, the address field of an instruction code will consist of 20 bits but physical memory addresses must be specified with only 15 bits.
- A memory table is used to map a virtual address of 20 bits to a physical address of 15 bits. The mapping is a dynamic operation which means that every address is translated immediately as word is referenced by CPU.

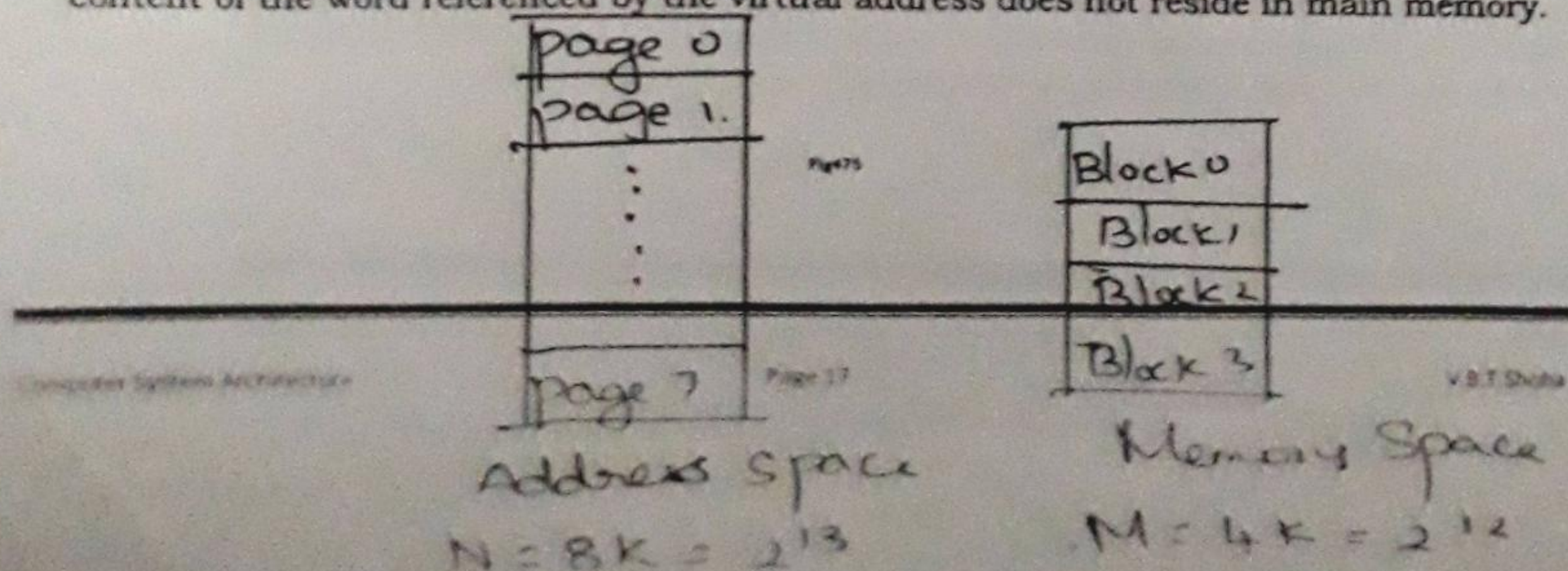




**b) Address Mapping using Pages:** The physical memory is broken down into groups of equal size called blocks, which may range from 64 to 4096 words each. The term page refers to groups of address space of the same size.

For example, if a page or block consists of 1K words, then, using the previous example, address space is divided into 1024 pages and main memory is divided into 1024 pages and main memory is divided into 32 blocks.

- Although both a page and a block are split into groups of 1K words, a page refers to the organization of address space, while a block refers to the organization of memory space.
- The programs also considered to be split into pages. Portions of programs are moved from auxiliary memory to main memory in records equal to the size of a page. The term "page frame" is sometimes used to denote a block.
- Consider a computer with an address space of 8K and a memory space of 4K. If we split each into groups of 1K words we obtain eight pages and four blocks as shown in Fig. At any given time, up to four pages of address space may reside in main memory in any one of the four blocks.
- The mapping from address space to memory space is facilitated if each virtual address is considered to be represented by two numbers: a page number address and a line within the page. In a computer with  $2^p$  words per page,  $p$  bits are used to specify a line address and the remaining high-order bits of the virtual address specify the page number. In the example of fig. a virtual address has 13 bits.
- Since each page consists of  $2^{10}=1024$  words, the High-order three bits of a virtual address will specify one of the eight pages and the low-order 10 bits give the line address within the page. Note that the line address in address space and memory space is the same; the only mapping required is from a page number to a block number.
- The memory-page table consists of eight words, one for each page. The address in the page table denotes the page number and the content of the word gives the block number where that page is stored in main memory. The table shows that pages 1, 2, 5, and 6 are now available in main memory in blocks 3, 0, 1, and 2, respectively.
- A presence bit in each location indicates whether the page has been transferred from auxiliary memory into main memory. A 1 in the presence bit indicates that this page is not available in main memory.
- The CPU reference a word in memory with a virtual address of 13 bits. The three high-order bits of the virtual address specify a page number and also an address for the memory-page table. The content of memory transfers the content of the word to the main memory buffer register ready to be used by the CPU.
- If the presence bit in the word read from the page table is 0, it signifies that the content of the word referenced by the virtual address does not reside in main memory.





**c) Associative Memory page table:** A random access memory page table is inefficient with respect to storage utilization. A system with  $n$  pages and  $m$  blocks would require a memory page table of  $n$  locations of which up to  $m$  blocks would require a memory page table of  $n$  locations of which up to  $m$  blocks will be marked with block numbers and all others will be empty.

This method can be implemented by means of an associative memory with each word in memory containing a page number together with its corresponding block number. The page field in each word is compared with the page number in the virtual address. If a match occurs, the word is read from memory and its corresponding block number is extracted.

Each entry in the associative memory array consists of two fields. The first three bits specify a field for storing the page number. The last two bits constitute a field for storing the block number. The virtual address is placed in the argument register. The page number bits in the argument register are compared with all page numbers in the page field of the associative memory.

If the page number is found, the 5bit word is read out from memory. The corresponding block number is transferred to the main memory address register. If no match occurs, a call to the operating system is generated to bring the required page from auxiliary memory.

**d) Page Replacement:** A virtual memory system is a combination of hardware and software techniques. The memory management software system handles all the software operations for the efficient utilization of memory space. It must decide

1. Which page in main memory ought to be removed to make room for a new page.
2. When a new page is to be transferred from auxiliary memory to main memory .
3. Where the page is to be placed in main memory.

When a program starts execution, one or more pages are transferred into main memory and the page table is set to indicate their position. The program is executed from main memory and the page table is set to indicate their position. The program is executed from main memory until it attempts to reference a page that is still in auxiliary memory. This condition is called page fault.

When page fault occurs, the execution of the present program is suspended until the required page is brought into main memory. The new page is then transferred from auxiliary memory to main memory. If main memory is full, it would be necessary to remove a page from a memory block to make room for the new page.

Two of the most common replacement algorithms used are the first-in, first-out(FIFO) and the least recently used(LRU) .The FIFO algorithm selects for replacement the page that has been in memory has no more empty blocks. When a new page must be loaded, the page least recently brought in is removed. The page to be removed is easily determined because its identification number is at the top of the FIFO stack.

The LRU policy is more difficult to implement. The least recently used page is the page with the highest count. The counter are called aging registers, as their count indicates their age, that is, how long ago their associated pages have been referenced.