

UNIT I

CPU Organisation

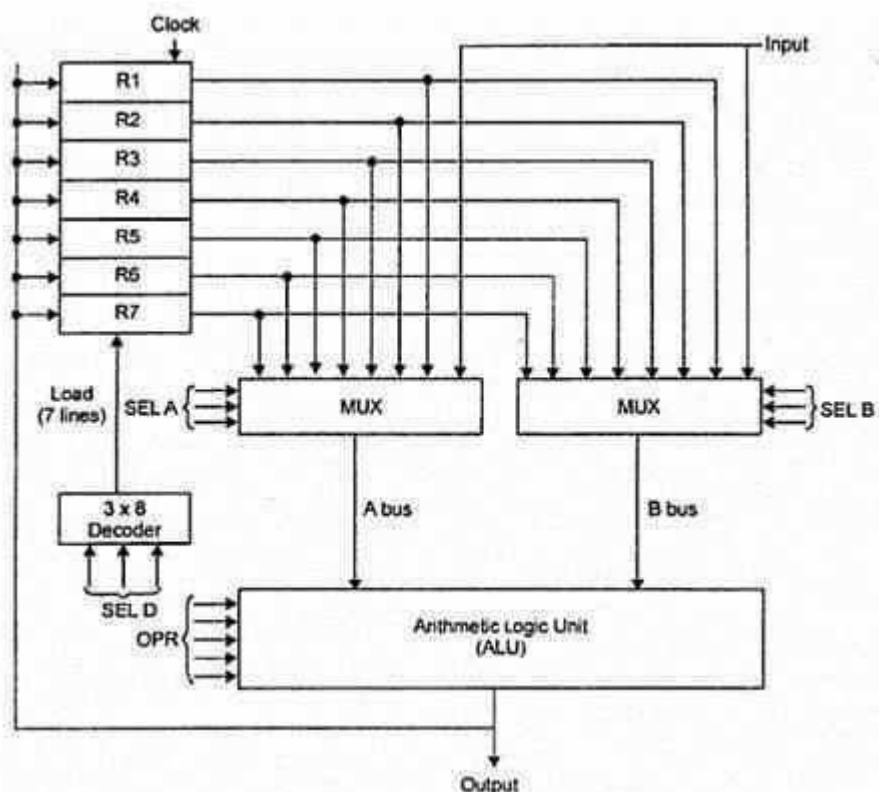
A central processing unit (CPU) is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.

A register may hold an instruction, a storage address, or any kind of data (such as a bit sequence or individual characters). Some instructions specify registers as part of the instruction. For example, an instruction may specify that the contents of two defined registers be added together and then placed in a specified register.

General Register organization

Generally CPU has seven general registers. Register organization show how registers are selected and how data flow between register and ALU. A decoder is used to select a particular register. 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 the input data for the particular bus. The A and B buses form the two inputs of an ALU. The operation select lines decide the micro operation to be performed by ALU. The result of the micro operation is available at the output bus. The output bus connected to the inputs of all registers, thus by selecting a destination register it is possible to store the result in it.

A bus organization for seven CPU registers



Example : To perform the operation **$R3 = R1+R2$** we have to provide following binary selection variable to the select inputs.

- 1. **SEL A: 001** -To place the contents of R1 into bus A.
- 2. **SEL B: 010** - to place the contents of R2 into bus B
- 3. **SEL OPR: 10010** – to perform the arithmetic addition A+B
- 4. **SEL REG or SEL D: 011** – to place the result available on output bus in R3.

Register and multiplexer input selection code

Binary	SELA	SELB	SELD or SELREG
000	Input	Input	---
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

Operation with symbol

Operation code	selection	Operation	symbol
0000		Transfer A	TSFA
0001		Increment A	INC A
0010		A+B	ADD
0011		A-B	SUB
0100		Decrement A	DEC
0101		A AND B	AND
0110		A OR B	OR
0111		A XOR B	XOR
1000		Complement A	COMA
1001		Shift right A	SHR
1010		Shift left A	SHL

What is CONTROL WORD?

- The combined value of a binary selection inputs specifies the control word.
- It consists of four fields SELA, SELB, and SELD or SELREG contains three bit each and SELOPR field contains four bits thus the total bits in the control word are 13-bits.

SEL A	SELB	SELREG OR SELD	SELOPR
-------	------	----------------	--------

FORMAT OF CONTROL WORD

1. The three bit of SELA select a source registers of the input of the ALU.
2. The three bits of SELB select a source registers of the b input of the ALU.
3. The three bits of SELED or SELREG select a destination register using the decoder.
4. The four bits of SELOPR select the operation to be performed by ALU.

CONTROL WORD FOR OPERATION $R2 = R1 + R3$

SEL D OR			
SEL A	SEL B	SEL REG	SELOPR
001	011	010	0010

STACK ORGANIZATION

Stack is a storage structure that stores information in such a way that the last item stored is the first item retrieved. It is based on the principle of LIFO (Last-in-first-out). The stack in digital computers is a group of memory locations with a register that holds the address of top of element. This register that holds the address of top of element of the stack is called **Stack Pointer**.

Stack Operations

The two operations of a stack are:

1. **Push:** Inserts an item on top of stack.
2. **Pop:** Deletes an item from top of stack.

Implementation of Stack

In digital computers, stack can be implemented in two ways:

1. Register Stack
2. Memory Stack

Register Stack

A stack can be organized as a collection of finite number of registers that are used to store temporary information during the execution of a program. The stack pointer (SP) is a register that holds the address of top of element of the stack.

Memory Stack

A stack can be implemented in a random access memory (RAM) attached to a CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. The starting memory location of the stack is specified by the processor register as *stack pointer*.

Register stack

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

In a 64-word stack, the stack pointer contains 6 bits because $2^6 = 64$. since SP has only six bits, it cannot exceed a number greater than 63 (111111 in binary). When 63 is incremented by 1, the result is 0 since $111111 + 1 = 1000000$ in binary, but SP can accommodate only the six least significant bits. Similarly, when 000000 is decremented by 1, the result is 111111. The one bit register Full is set to 1 when the stack is full, and the one-bit register Empty is set to 1 when the stack is empty of items. DR is the data register that holds the binary data to be written in to or read out of the stack.

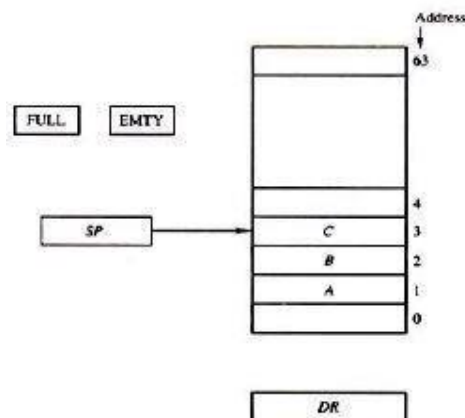


Figure 3: Block Diagram Of A 64-Word Stack

Initially, SP is cleared to 0, Empty is set to 1, and Full is cleared to 0, so that SP points to the word at address 0 and the stack is marked empty and not full. if the stack is not full, a new item is inserted with a push operation. the push operation is implemented with the following sequence of micro-operation.

$SP \leftarrow SP + 1$ (Increment stack pointer)
 $M(SP) \leftarrow DR$ (Write item on top of the stack) if (sp=0) then (Full $\leftarrow 1$) (Check if stack is full)
Empty $\leftarrow 0$ (Marked the stack not empty)

The stack pointer is incremented so that it points to the address of the next-higher word. A memory write operation inserts the word from DR into the top of the stack. Note that SP holds the address of the top of the stack and that $M(SP)$ denotes the memory word specified by the address presently available in SP, the first item stored in the stack is at address 1. The last item is stored at address 0, if SP reaches 0, the stack is full of item, so Full is set to 1. This condition is reached if the top item prior to the last push was in location 63 and after increment SP, the last item stored in location 0. Once an item is stored in location 0, there are no more

empty register in the stack. If an item is written in the stack, Obviously the stack can not be empty, so EMPT is cleared to 0.

```

DR ← M[SP]           Read item from the top of stack
SP ← SP-1             Decrement stack
Pointer if ( SP=0) then (Emty ← 1) Check if
stack is empty FULL ← 0 Mark the stack not
full

```

The top item is read from the stack into DR. The stack pointer is then decremented. if its value reaches zero, the stack is empty, so Emty is set to 1. This condition is reached if the item read was in location 1. once this item is read out , SP is decremented and reaches the value 0, which is the initial value of SP. Note that if a pop operation reads the item from location 0 and then SP is decremented, SP changes to 111111, which is equal to decimal 63. In this configuration, the word in address 0 receives the last item in the stack. Note also that an erroneous operation will result if the stack is pushed when FULL=1 or popped when EMTY =1.

Memory stack

A stack can exist as a stand-alone unit as in figure 4 or can be implemented in a random access memory attached to CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. Figure shows a portion of computer memory partitioned in to three segment program, data and stack. The program counter PC points at the address of the next instruction in the program. The address register AR points at an array of data. The stack pointer SP points at the top of the stack. The three register are connected to a common address bus, and either one can provide an address for memory. PC is used during the fetch phase to read an instruction. AR is used during the execute phase to read an operand. SP is used to push or POP items into or from the stack.

As show in figure :4 the initial value of SP is 4001 and the stack grows with decreasing addresses. Thus the first item stored in the stack is at address 4000, the second item is stored at address 3999, and the last address hat can be used for the stack is 3000. No previous are available for stack limit checks. We assume that the items in the stack communicate with a data register DR. A new item is inserted with the push operation as follows.

```

SP ← SP-
1   M[SP]
← DR

```

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

```

DR ← M[SP]
SP ← SP + 1

```

The top item is read from the stack in to DR. The stack pointer is then incremented to point at the next item in the stack.

Most computer do not provide hardware to check for stack overflow (FULL) or underflow (Empty). The stack limit can be checked by using two processor register :one to hold upper limit and other hold the lower limit. after the pop or push operation SP is compared with lower or upper limit register.

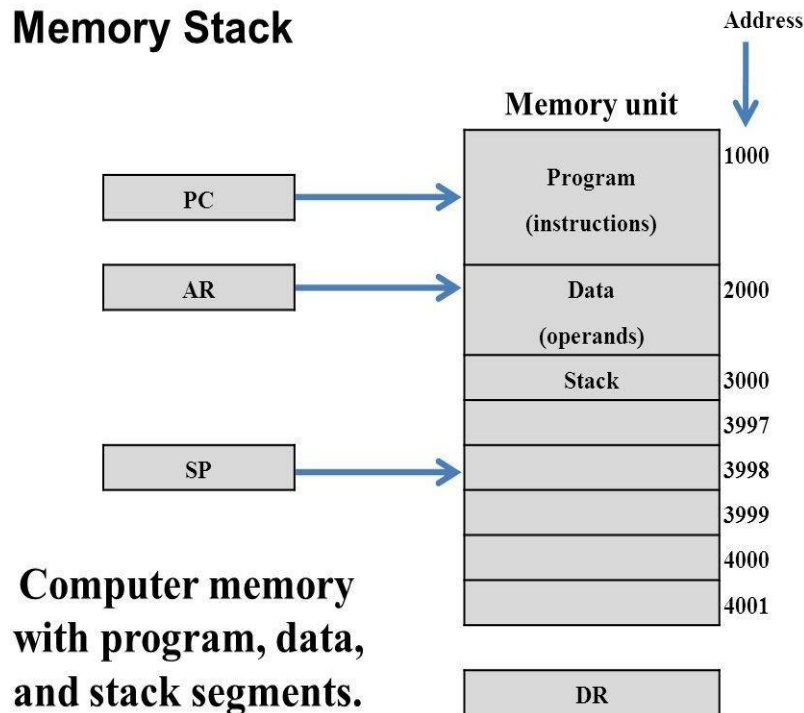


Figure 4: computer memory with program, data and stack segments

Instruction Formats:

A computer will usually have a variety of instruction code formats. It is the function of the control unit within the CPU to interpret each instruction code and provide the necessary control functions needed to process the instruction.

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

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

Other special fields are sometimes employed under certain circumstances, as for example a field that gives the number of shifts in a shift-type instruction.

The operation code field of an instruction is a group of bits that define various processor operations, such as add, subtract, complement, and shift. The bits that define the mode field of an instruction code specify a variety of alternatives for choosing the operands from the given address.

Operations specified by computer instructions are executed on some data stored in memory or processor registers. Operands residing in processor registers are specified with a register address. A register address is a binary

number of k bits that defines one of 2^k registers in the CPU. Thus a CPU with 16 processor registers R0 through R15 will have a register address field of four bits. The binary number 0101, for example, will designate register R5.

Computers may have instructions of several different lengths containing varying number of addresses. The number of address fields in the instruction format of a computer depends on the internal organization of its registers. Most computers fall into one of three types of CPU organizations:

- 1 Single accumulator organization.
- 2 General register organization.
- 3 Stack organization.

All operations are performed with an implied accumulator register. The instruction format in this type of computer uses one address field. For example, the instruction that specifies an arithmetic addition is defined by an assembly language instruction as `ADD`.

Where X is the address of the operand. The `ADD` instruction in this case results in the operation $AC \leftarrow AC + M[X]$. AC is the accumulator register and $M[X]$ symbolizes the memory word located at address X .

An example of a general register type of organization was presented in Fig. 7.1. The instruction format in this type of computer needs three register address fields. Thus the instruction for an arithmetic addition may be written in an assembly language as

`ADD            R1, R2, R3`

To denote the operation $R1 \leftarrow R2 + R3$. The number of address fields in the instruction can be reduced from three to two if the destination register is the same as one of the source registers. Thus the instruction

`ADD            R1, R2`

Would denote the operation $R1 \leftarrow R1 + R2$. Only register addresses for $R1$ and $R2$ need be specified in this instruction.

Computers with multiple processor registers use the move instruction with a mnemonic `MOV` to symbolize a transfer instruction. Thus the instruction

`MOV           R1, R2`

Denotes the transfer $R1 \leftarrow R2$ (or $R2 \leftarrow R1$, depending on the particular computer). Thus transfer-type instructions need two address fields to specify the source and the destination.

General register-type computers employ two or three address fields in their instruction format. Each address field may specify a processor register or a

memory word. An instruction symbolized by `ADD R1, X`

Would specify the operation $R1 \leftarrow R + M[X]$. It has two address fields, one for register R1 and the other for the memory address X.

The stack-organized CPU was presented in Fig. 8-4. Computers with stack organization would have PUSH and POP instructions which require an address field. Thus the instruction

`PUSH            X`

Will push the word at address X to the top of the stack. The stack pointer is updated automatically. Operation-type instructions do not need an address field in stack-organized computers. This is because the operation is performed on the two items that are on top of the stack. The instruction ADD in a stack computer consists of an operation code only with no address field. This operation has the effect of popping the two top numbers from the stack, adding the numbers, and pushing the sum into the stack. There is no need to specify operands with an address field since all operands are implied to be in the stack.

To illustrate the influence of the number of addresses on computer programs, we will evaluate the arithmetic statement $X = (A + B) * (C + D)$.

Using zero, one, two, or three address instruction. We will use the symbols ADD, SUB, MUL, and DIV for the four arithmetic operations; MOV for the transfer-type operation; and LOAD and STORE for transfers to and from memory and AC register. We will assume that the operands are in memory addresses A, B, C, and D, and the result must be stored in memory at address X.

Three-Address Instructions

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

```
ADD  R1, A, B   R1 ← M[A] + M[B]
ADD  R2, C, D   R2 ← M[C] + M[D]
MUL  X, R1, R2  M[X] ← R1 * R2
```

It is assumed that the computer has two processor registers, R1 and R2. The symbol M[A] denotes the operand at memory address symbolized by A.

The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses. An example

of a commercial computer that uses three-address instructions is the Cyber 170. The instruction formats in the Cyber computer are restricted to either three register address fields or two register address fields and one memory address field.

Two-Address Instructions

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

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

One-Address Instructions

One-address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication and division there is a need for a second register. However, here we will neglect the second and assume that the AC contains the result of all operations. The program to evaluate $X = (A + B) * (C + D)$ is

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 a memory operand. T is the address of a temporary memory location required for storing the intermediate result.

Zero-Address Instructions

A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack. The following program shows how $X = (A + B)$

* (C + D) will be written for a stack organized computer. (TOS stands for top of stack)

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

To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse Polish notation. The name “zero-address” is given to this type of computer because of the absence of an address field in the computational instructions.

Instruction Codes

A set of instructions that specify the operations, operands, and the sequence by which processing has to occur. An instruction code is a group of bits that tells the computer to perform a specific operation part.

Format of Instruction

The format of an instruction is depicted in a rectangular box symbolizing the bits of an instruction. Basic fields of an instruction format are given below:

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

Computers may have instructions of different lengths containing varying number of addresses. The number of address field in the instruction format depends upon the internal organization of its registers.

Addressing modes

The operation field of an instruction specifies the operation to be performed. This operation must be executed on some data stored in computer registers or memory words. The way the operands are chosen during program execution is dependent on the addressing mode of the instruction. The addressing mode of the instruction specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.

Although most addressing modes modify the address field of the instruction, there are two modes that need no address field at all. These are the implied and immediate modes.

1 Implied Mode: In this mode the operands are specified implicitly in the definition of the instruction. **For example, the instruction “complement accumulator” is an implied-mode**

instruction because the operand in the accumulator register is implied in the definition of the instruction. In fact, all register reference instructions that use an accumulator are implied- mode instructions. Instruction format with mode field Zero-address instructions in a stack-

organized computer are implied-mode instructions since the operands are implied to be on top of the stack.

2 Immediate Mode: In this mode the operand is specified in the instruction itself. In other words, an immediate mode instruction has an operand field rather than an address field. The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction. Immediate-mode instructions are useful for initializing registers to a constant value. It was mentioned that the address field of an instruction may specify either a memory word or a processor register. When the address field specifies a processor register, the instruction is said to be in the register mode.

3 Register Mode: In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction. A k-bit field can specify any one of 2^k registers.

4 Register Indirect Mode: In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory. In other words, the selected register contains the address of the operand rather than the Op code Mode Address operand itself. Before using a register indirect mode instruction, the programmer must ensure that the memory address for the operand is placed in the processor register with a previous instruction. A reference to the register is then equivalent to specifying a memory address. The advantage of a register indirect mode instruction is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

5 Auto increment or Auto decrement Mode: This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory. When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table. This can be achieved by using the increment or decrement

instruction. However, because it is such a common requirement, some computers incorporate a special mode that automatically increments or decrements the content of the register after data access. The address field of an instruction is used by the control unit in the CPU to obtain the operand from memory. Sometimes the value given in the address field is the address of the operand, but sometimes it

is just an address from which the address of the operand is calculated. To differentiate among the various addressing modes it is necessary to distinguish between the address part of the instruction and the effective address used by the control when executing the instruction. The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode. The effective address is the address of the operand in a computational-type instruction. It is the address where control branches in response to a branch-type instruction.

6 Direct Address Mode: In this mode the effective address is equal to the address part of the instruction. The operand resides in memory and its address is given directly by the address field of the instruction. In a branch-type instruction the address field specifies the actual branch address.

7 Indirect Address Mode: In this mode the address field of the instruction gives the address where the effective address is stored in memory. Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.

8 Relative Address Mode: In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address. The address part of the **instruction is usually a signed number (in 2's complement representation) which can be either** positive or negative. When this number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction. To clarify with an example, assume that the program counter contains the number 825 and the address part of the instruction contains the number 24. The instruction at location 825 is read from memory during the fetch phase and the program counter is then incremented by one to $826 + 24 = 850$. This is 24 memory locations forward from the address of the next instruction. Relative addressing is often used with branch-type instructions when the branch address is in the area surrounding the instruction word itself. It results in a shorter address field in the instruction format since the relative address can be specified with a smaller number of bits compared to the number of bits required to designate the entire memory

address.

9 Indexed Addressing Mode: In this mode the content of an index register is added to the address part of the instruction to obtain the effective address. The index register is a special CPU register that contains an index value. The address field of the instruction defines the beginning address of a data array in memory. Each operand in the array is stored in memory

relative to the beginning address. The distance between the beginning address and the address of the operand is the index value stored in the index register. Any operand in the array can be accessed with the same instruction provided that the index register contains the correct index value. The index register can be incremented to facilitate access to consecutive operands. Note that if an index-type instruction does not include an address field in its format, the instruction converts to the register indirect mode of operation. Some computers dedicate one CPU register to function solely as an index register. This register is involved implicitly when the index-mode instruction is used. In computers with many processor registers, any one of the CPU registers can contain the index number. In such a case the register must be specified explicitly in a register field within the instruction format.

10 Base Register Addressing Mode: In this mode the content of a base register is added to the address part of the instruction to obtain the effective address. This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register. The difference between the two modes is in the way they are used rather than in the way that they are computed. An index register is assumed to hold an index number that is relative to the address part of the instruction. A base register is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address. The base register addressing mode is used in computers to facilitate the relocation of programs in memory. When programs and data are moved from one segment of memory to another, as required in multiprogramming systems, the address values of the base register require updating to reflect the beginning of a new memory segment.

One mark questions

- Q1. Define CPU
- Q2. Define Control Word
- Q3. List two stack operations
- Q4. Define Stack
- Q5. List two ways of implementing stack
- Q6. List three types of CPU Organisations.

Short answer questions

- Q7. Briefly explain register stack
- Q8. Briefly explain memory stack
- Q9. Explain control word with example
- Q10. Explain format of instruction briefly.

Long answer questions

- Q11. Explain general register organization in detail.
- Q12. Explain stack organization in detail.
- Q13. Explain various types of instructions based on number of operands/addresses
- Q14. Explain the various addressing modes in detail.

UNIT II

MEMORY ORGANISATION

The memory unit is an essential component in any digital computer since it is needed for storing programs and Data. There is just not enough space in one memory unit to accommodate all the programs used in a typical computer. Moreover, most computer users accumulate and continue to accumulate large amounts of data-processing software. Not all accumulated information is needed by the processor at the same time. Therefore, it is more economical to use low-cost storage devices to serve as a backup for storing the information that is not currently used by the CPU. The memory unit that communicates directly with the CPU is called the main memory. Devices that provide backup storage are called auxiliary memory. The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. They are used for storing system programs, large data files, and other backup information. Only programs and data currently needed by the processor reside in main memory. All other information is stored in auxiliary Memory and transferred to main memory when needed.

Memory Hierarchy

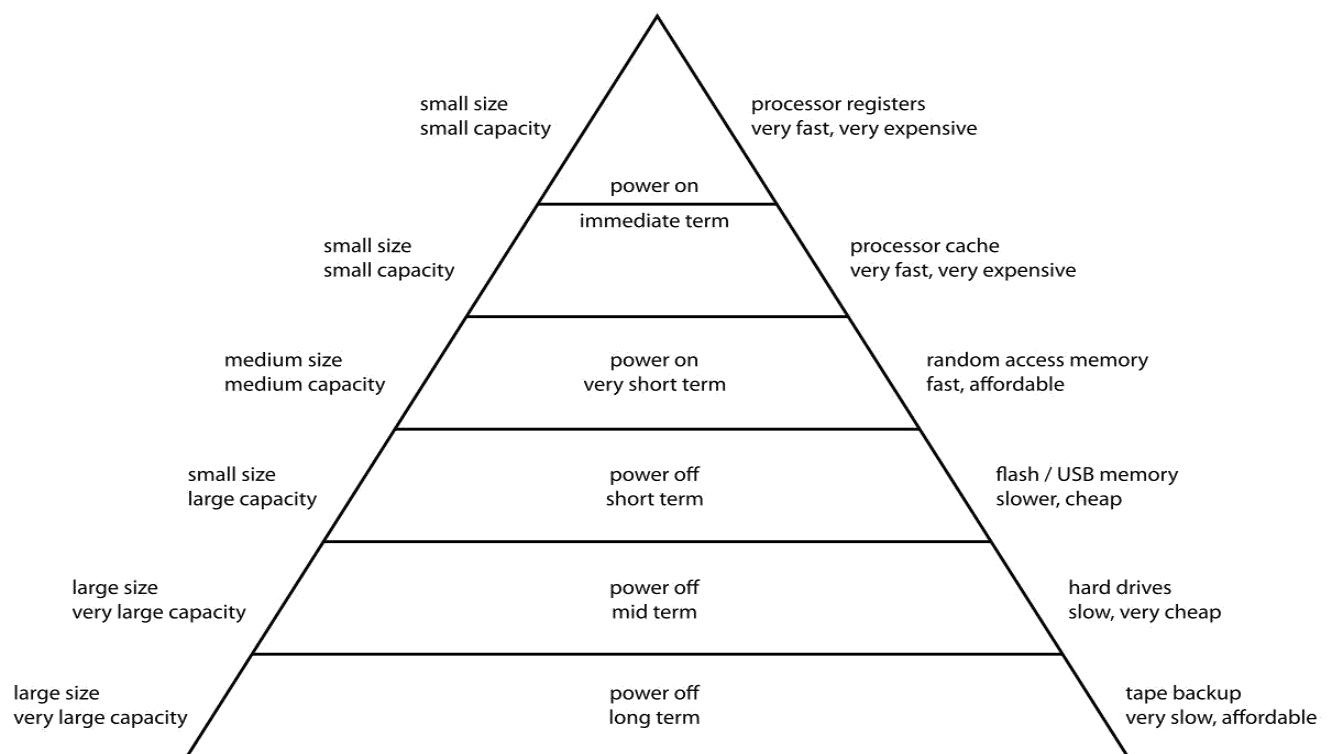
Memory is categorized into volatile and nonvolatile memories, with the former requiring constant power ON of the system to maintain data storage. Furthermore, a typical computer system provides a hierarchy of different times of memories for data storage.

Different levels of the memory hierarchy

- **Cache (MB):** Cache is the fastest accessible memory of a computer system. Its access speed is in the order of a few nanoseconds. It is volatile and expensive, so the typical cache size is in the order of megabytes.
- **Main memory (GB):** Main memory is arguably the most used memory. When discussing computer algorithms such as quick sort, balanced binary sorted trees, or fast Fourier transform, one typically assumes that the algorithm operates on data stored in the main memory. The main memory is reasonably fast, with access speed around 100 nanoseconds. It also offers larger capacity at a lower cost. Typical main memory is in the order of 10 GB. However, the main memory is volatile.

- Secondary storage (TB): Secondary storage refers to nonvolatile data storage units that are external to the computer system. Hard drives and solid state drives are examples of secondary storage. They offer very large storage capacity in the order of terabytes at very low cost. Therefore, database servers typically have an array of secondary storage devices with data stored distributed and redundantly across these devices. Despite the continuous improvements in access speed of hard drives, secondary storage devices are several magnitudes slower than main memory. Modern hard drives have access speed in the order of a few milliseconds.
- Tertiary storage (PB): Tertiary storage refers storage designed for the purpose data backup. Examples of tertiary storage devices are tape drives are robotic driven disk arrays. They are capable of peta byte range storage, but have very slow access speed with data access latency in seconds or minutes.

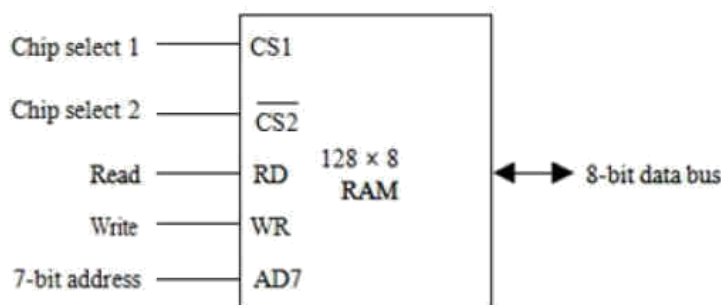
Computer Memory Hierarchy



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 constructed with three-state buffers. A three-state buffer output can be placed in one of 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 in Fig. The capacity of the memory is 128 words of eight bits (one byte) per word.



(a) Block diagram

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

(b) Function table

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, the two binary states in this line specify the two operations or read or write.

The function table listed in Fig. (b) Specifies the operation of the RAM chip. The unit is in operation only when CS1 = 1 and CS2 = 0. The bar on top of the second select variable indicates that this input is enabled when it is equal to 0. If the chip select inputs are not enabled, or if they are enabled but the read but the read or write inputs are not enabled, the memory is inhibited and its data bus is in a high-impedance

state. When $CS1 = 1$ and $CS2 = 0$, the memory can be placed in a write or read mode. When the WR input is enabled, the memory stores a byte from the data bus into a location specified by the address input lines. When the RD input is enabled, the content of the selected byte is placed into the data bus. The RD and WR signals control the memory operation as well as the bus buffers associated with the bidirectional data bus.

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 below Fig. For the same-size chip, it is possible to have more bits of ROM occupy less space than in RAM. For this reason, the diagram specifies a 512-byte 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 inputs, the byte selected by the address lines appears on the data bus.

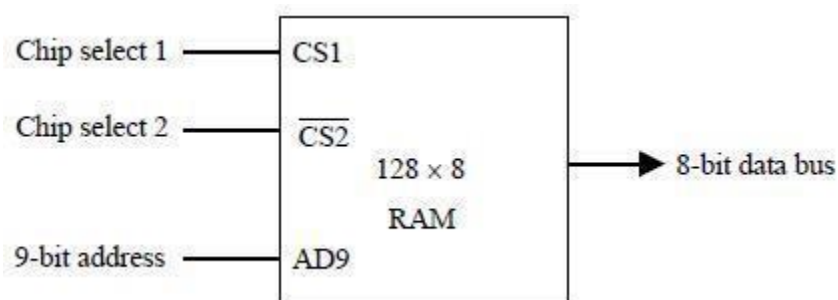


Figure-Typical ROM chip.

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 Typical RAM chip and Typical ROM chip. The memory address map for this configuration is shown in Table Below. 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 designate those lines that must be connected to the address inputs in each chip. The RAM chips have 128 bytes and need seven 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 from RAM equal to 512 bytes. The distinction between a RAM and ROM address is done with another bus line. Here we choose line 10 for this purpose. When line 10 is 0, the CPU selects a RAM, and when this line is equal to 1, it selects the ROM. The equivalent hexadecimal address for each chip is obtained from the information under the address bus assignment.

TABLE-Memory Address Map for Micro computer

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 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 addresses for each component is determined from the x's associated with it. This x's represent a binary number that can range from an all-0's to an all-1's value.

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 below Fig. This configuration gives a memory capacity of 512 bytes of RAM and 512 bytes of ROM. It implements the memory map of Table above. 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 2×4 decoder whose outputs go to the SCI input in each RAM chip. Thus, when address lines 8 and 9 are equal to 00, the first RAM chip is selected. When 01, the second RAM chip is 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 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 addresses 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.

The example just shown gives an indication of the interconnection complexity that can exist between memory chips and the CPU. The more chips that are connected, the more external decoders are required for selection among the chips. The designer must establish a memory map that assigns addresses to the various chips from which the required connections are determined.

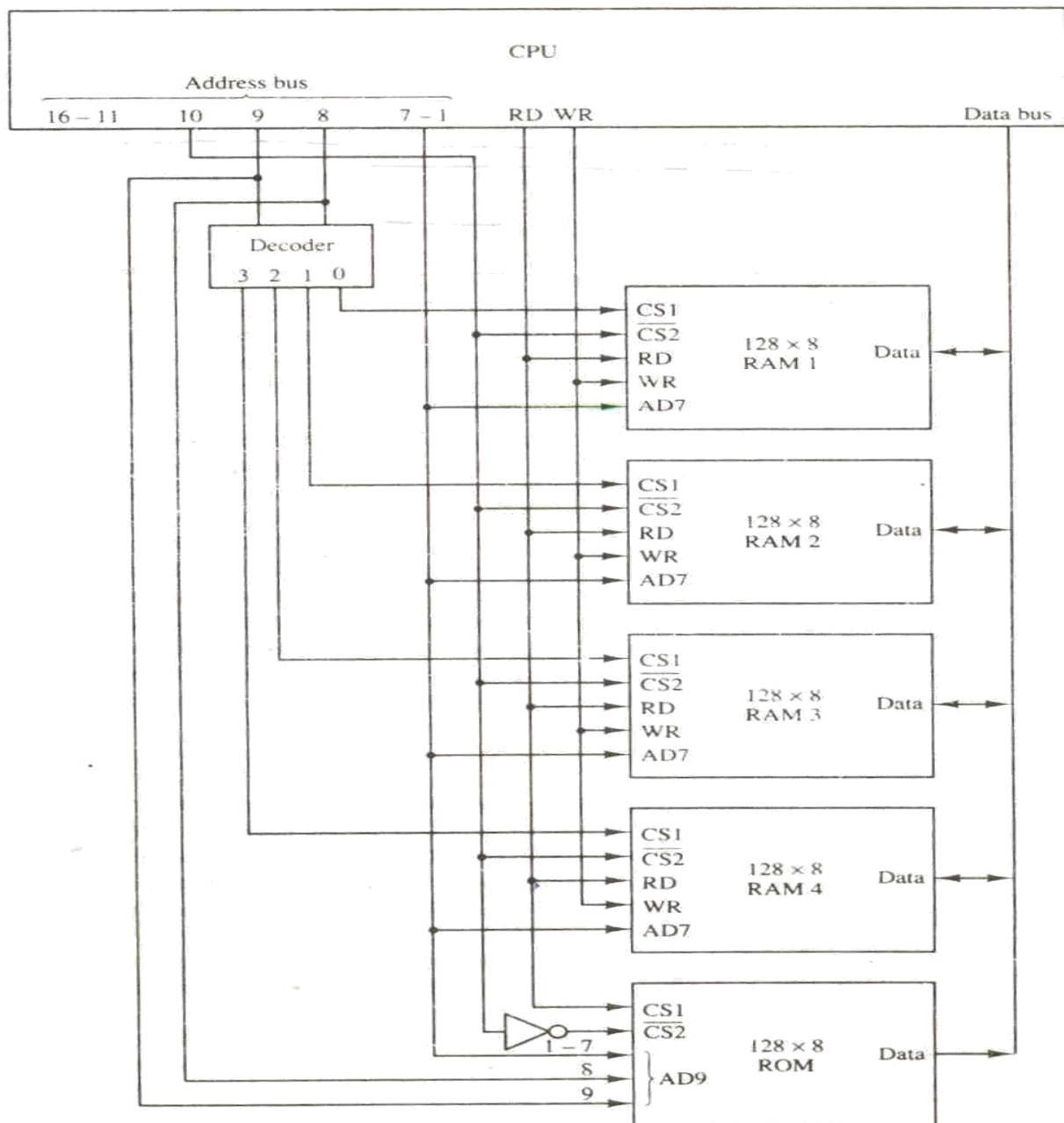


Figure -Memory connection to the CPU.

Auxiliary memory

(also referred to as secondary storage) is the non-volatile memory lowest-cost, highest- capacity, and slowest-access storage in a computer system. It is where programs and data kept for long-term storage or when not in immediate use.

Auxiliary memory may also refer to as auxiliary storage, secondary storage, secondary memory, external storage or external memory. Auxiliary memory is not directly accessible by the CPU; instead, it stores noncritical system data like large data files, documents, programs and other back up information that supplied to primary memory from auxiliary memory over a high-bandwidth channel, which will use whenever necessary. Auxiliary memory holds data for future use, and that retains information even the power fails.

A magnetic disk is a storage device that uses a magnetization process to write, rewrite and access data. It is covered with a magnetic coating and stores data in the form of tracks, spots and sectors. Hard disks, zip disks and floppy disks are common examples of magnetic disks. A magnetic disk primarily consists of a rotating magnetic surface and a mechanical arm that moves over it. The mechanical arm is used to read from and write to the disk. The data on a magnetic disk is read and written using a magnetization process. Data is organized on the disk in the form of tracks and sectors, where tracks are the circular divisions of the disk. Tracks are further divided into sectors that contain blocks of data. All read and write operations on the magnetic disk are performed on the sectors.

Magnetic tape is one of the oldest technologies for electronic data storage. Tape has largely been displaced as a primary and backup storage medium, but it remains well-suited for archiving because of its high capacity, low cost and long durability. It is a linear recording system that is not good for random access. If the tape is part of a library, robotic selection and loading of the right cartridge into a tape drive adds more latency. In an archive, such latencies are not an issue. With tape archiving, there is no online copy for quick retrieval, as everything is vaulted for the long term. While tape can't compete with other media in terms of random access, there are still industries where magnetic tape storage is the preferred technology:

1. Many motion picture production companies record their shoots to tape after experiencing costly failures with both disk and flash.
2. Scientific experiments that produce mass quantities of data in a few microseconds leverage tape's capacity and write speeds.
3. The oil and gas industry has used tape for years to capture, transport and store

valuable data. Because oil exploration occurs outside the data center, tape is a good medium for transporting data back from the field.

Tape is often paired with object storage to address the need for lower-latency file access. Sometimes, it is entirely replaced by object storage.

How magnetic tape works

Data bits -- magnetic states representing on and off -- are recorded to a particulate medium bonded to a substrate of Mylar plastic. Improvements in track-following technology and giant magnet resistive read/write heads have increased the number of tracks that can be recorded on a tape.



CACHE MEMORY

Analysis of a large number of typical programs has shown that the references, to memory at any given interval of time tend to be confined within a few localized areas in memory. The phenomenon is known as the property of locality of reference. The reason for this property may be understood considering that a typical computer

program flows in a straight-line fashion with program loops and subroutine calls encountered frequently. When a program loop is executed, the CPU repeatedly refers to the set of instructions in memory that constitute the loop. Every time a given subroutine is called, its set of instructions is fetched from memory. Thus loops and subroutines tend to localize the references to memory for fetching instructions. To a lesser degree, memory references to data also tend to be localized. Table-lookup procedures repeatedly refer to that portion in memory where the table is stored. Iterative procedures refer to common memory locations and array of numbers are confined within a local portion of memory. The result of all these observations is the locality of reference property, which states that over a short interval of time, the addresses generated by a typical program refer to a few localized areas of memory repeatedly, while the remainder of memory is accessed relatively infrequently. If the active portions of the program and data are placed in a fast small memory, the average memory access time can be reduced, thus reducing the total execution time of the program. Such a fast small memory is referred to as a cache memory. It is placed between the CPU and main memory as illustrated in below Fig. 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 and approaches the speed of CPU components. The fundamental idea of cache organization is that by keeping the most frequently accessed instructions and data in the fast cache memory, the average memory access time will approach the access time of the cache. Although the cache is only a small fraction of the size of main memory, a large fraction of memory requests will be found in the fast cache memory because of the locality of reference property of programs.

The basic operation of the cache is as follows. 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. If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word. A block of words containing the one just accessed is then transferred from main memory to cache memory. The block size may vary from one word (the one just accessed) to about 16 words adjacent to the one just accessed. In this manner, some data are transferred to cache so that future references to memory find the required words in the fast 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 (hits plus misses) is the hit ratio. The hit ratio is best measured experimentally by running representative programs in the computer and measuring the number of hits and misses during a given interval of time. Hit ratios of 0.9 and higher have been reported. This high ratio verifies the validity of the locality of reference property.

The average memory access time of a computer system can be improved considerably by use of a cache.

If the hit ratio is high enough so that most of the time the CPU accesses the cache instead of main memory, the average access time is closer to the access time of the fast cache memory. For example, a computer with cache access time of 100 ns, a main memory access time of 1000 ns, and a hit ratio of 0.9 produces an average access time of 200 ns. This is a considerable improvement over a similar computer without a cache memory, whose access time is 1000 ns. The basic characteristic of cache memory is its fast access time. Therefore, very little or no time must be wasted when searching for words in the cache. The transformation of data from main memory to cache memory is referred to as a mapping process. Three types of mapping procedures are of practical interest when considering the organization of cache memory:

1. Associative mapping
2. Direct mapping
3. Set-associative mapping

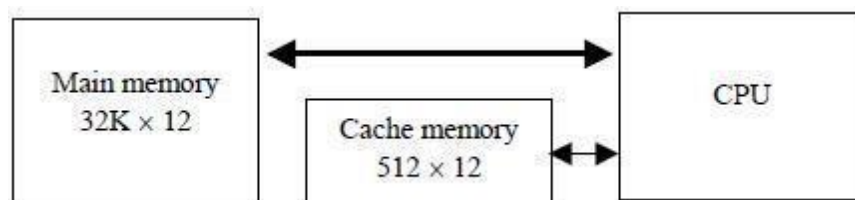


Figure - Example of cache memory

ASSOCIATIVE MEMORY

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 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 the word, or part of the word, is specified. The memory locates all words which match the specified content and marks them for reading. Because of its organization, the associative memory is uniquely suited to do parallel searches by data association. Moreover, searches can be done on an entire word or on a specific field within a word. An associative memory is more expensive than a random access memory because each cell must have storage capability as well as logic circuits for matching its content with an external argument. For this reason, associative memories are used in applications where the search time is very critical and must be very short.

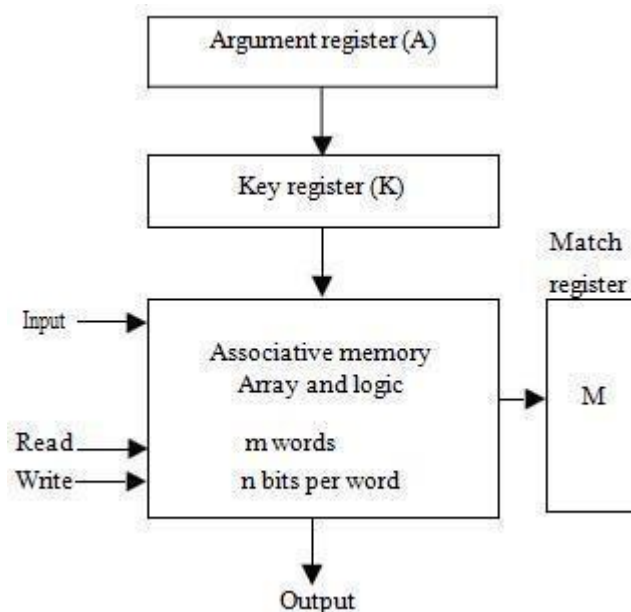
The block diagram of an associative memory is shown in below Fig. It consists of a

memory array and logic from words with n bits per word. The argument register A and key register K each have n bits, one for each bit of a word. The match register M has m bits, one for each memory word. Each word in memory is compared in parallel with the content of the argument register. The words that match the bits of the argument register set a corresponding bit in the match register. After the matching process, those bits in the match register that have been set 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 or identifying piece of information which specifies how the reference to memory is made.

Figure- Block diagram of associative memory



To illustrate with a numerical example, suppose that the argument register A and the key register K have the bit configuration shown 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

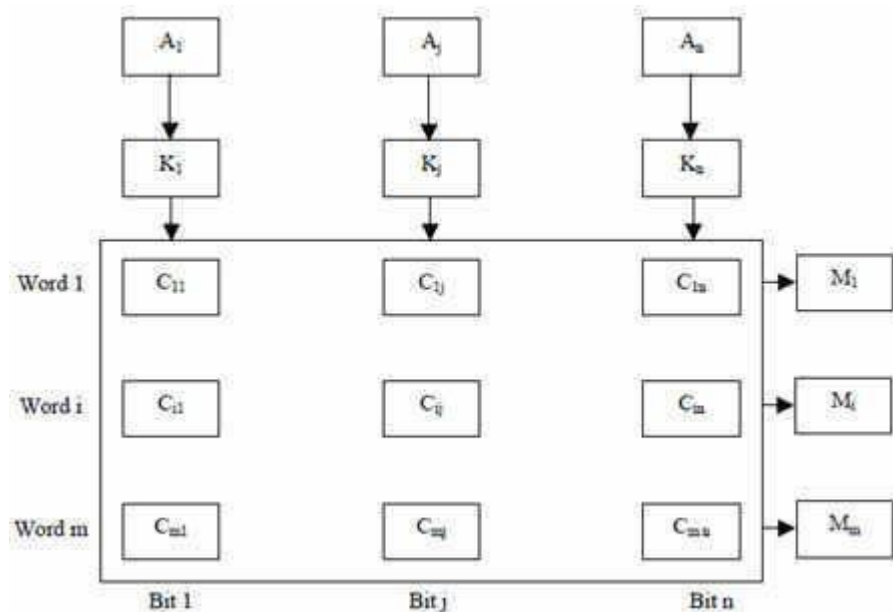
Word 2 101 000001 match

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 external registers in an associative memory is shown in below 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

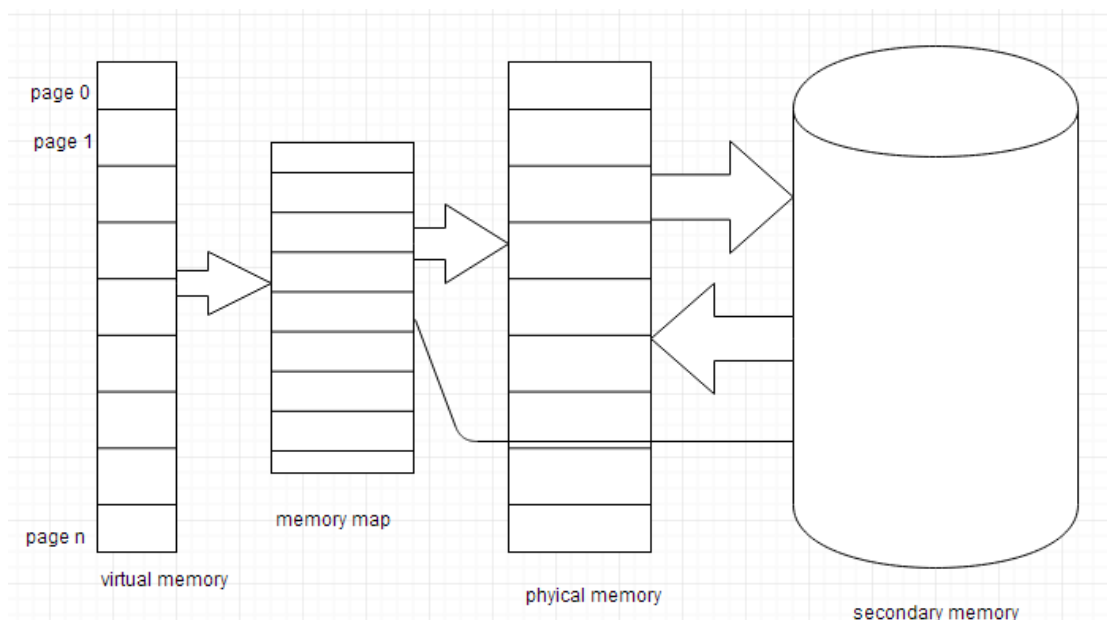
argument and the bits in word i , the corresponding bit M_i in the match register is set to 1. If one or more unmasked bits of the argument and the word do not match, M_i is cleared to 0.

Figure -Associative memory of m word, n cells per word



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 into the storage cell during a write operation. The bit stored is read out during a read operation. The match logic compares the content of the storage cell with the corresponding unmasked bit of the argument and provides an output for the decision logic that sets the bit in M_i .

Virtual memory is the separation of logical memory from physical memory. This



separation provides large virtual memory for programmers when only small physical memory is available. Virtual memory is used to give programmers the illusion that they have a very large memory even though the computer has a small main memory. It makes the task of programming easier because the programmer no longer needs to worry about the amount of physical memory available.

Address mapping using pages The table implementation of the address mapping is simplified if the information in the address space. And the memory space is each divided into groups of fixed size. Moreover, 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. Also, 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 the figure. At any given time, up to four pages of address space may reside in main memory in any one of the four blocks.

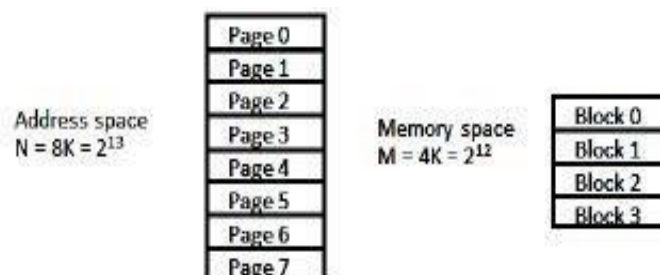
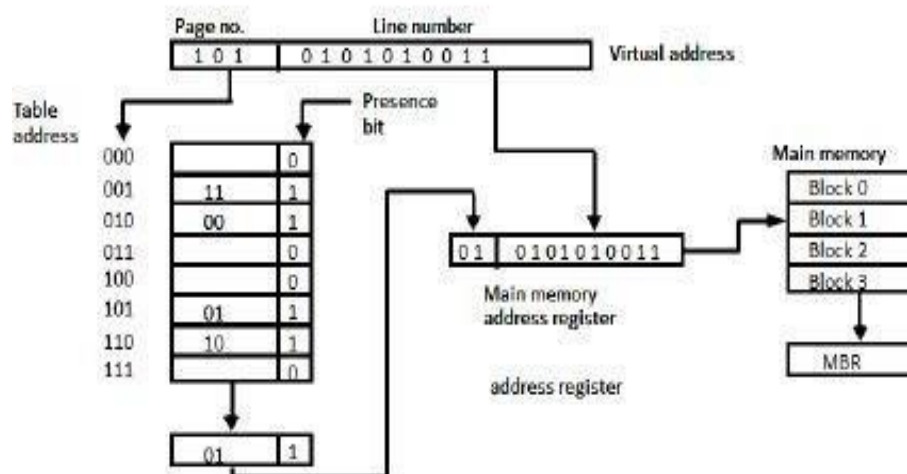


Figure Address and Memory space split into group of 1K words



The advantage of virtual memory is that processes can be using more memory than exists in the machine; when memory is accessed that is not present (a **page fault**), it must be paged in (sometimes referred to as being "swapped in", although some people reserve "swapped in" to refer to bringing in an entire address space).

Swapping in pages is very expensive (it requires using the disk), so we'd like to avoid page faults as much as possible. The algorithm that we use to choose which pages to evict to make space for the new page can have a large impact on the number of page faults that occur.

Memory Management The task of the memory manager and memory management are to ensure that all processes are always able to access their memory. To accomplish this task requires careful integration between the computer's hardware and the operating system.

When several processes with dynamic memory needs run on the computer at the same time, it is necessary to reference data with both a logical address and a physical address. The hardware is responsible for translating the logical addresses into physical addresses in real time. While the operating system is responsible to:

1. ensure that the requested data is in physical memory when needed
2. Program the hardware to perform the address translations.

The Paged Memory Management scheme gives rise to the notion of demand paging using virtual memory. The Virtual Memory Management system maintains a copy of the memory for all programs on secondary storage, such as a hard drive. In fact, many pages for a process may only reside in virtual memory. Loading only the page frames that are needed to run a program can make it faster to load a program. Some memory frames for a program may never be needed while the program runs.

If a current copy of a frame of physical memory is held on disk in virtual memory, then that frame may be removed from physical memory to

free up needed memory. When a process references memory that is not loaded in physical memory, the Memory Management Unit of the CPU issues a page fault trap.

One mark questions

- Q1. Define memory
- Q2. Define memory address map
- Q3. List two secondary memories
- Q4. Define RAM
- Q5. Define Cache
- Q6. List three mapping procedures of cache memory.
- Q7. Give the full form of ROM

Short answer questions

- Q8. Briefly explain memory hierarchy.
- Q9. Explain memory connections to CPU
- Q10. What is the difference between magnetic disk and tape..

Long answer questions

- Q11. Explain Virtual memory in detail.
- Q12. Explain Associative memory in detail.
- Q13. Explain memory address map using an example.
- Q14. Explain the concept of paging in detail.

UNIT III

I/O ORGANIZATION

BIOS (Basic Input Output System) is a fundamental component in all computers. Now we will discuss about what is BIOS (Basic Input Output System), its functions, What does it do, configuring Basic I/O System and its limitations.

Basic Input Output System is abbreviated as **BIOS** and was first developed by Gary Kildall, an American scientist. Every computer has a special chip on the Motherboard called as BIOS. All the personal computer comes preinstalled with BIOS. Once a computer is turned on, it needs to perform certain operations. The basic function of Basic I/O System is to govern the early stages of the startup process. It enables the operating system to communicate with the hardware of the computer. Basic I/O System provides the hardware initialization during the booting process.

To enter the BIOS Setup Screen also called as CMOS Setup, simply power up the computer and while the BIOS is performing *Power On Self Test (POST)* press the BIOS key which could be F2, F10, Del, F12 etc. This depends on the manufacturer. The instructions to load a basic computer hardware is included in BIOS. The “Power ON Self Test” (POST) helps to verify that boot up requirements are in place. If the computer is unable to pass the POST test, it is indicated through a series of beeps, informing the user about the malfunctioning of the computer.

However nowadays, BIOS has slowly been replaced by **UEFI (Unified Extensible Firmware Interface)** in new computers which offers benefits like a better user interface and a built in Pre-OS platform for accessing the web. If we want to add additional memory to the chip, it is only possible through physical removal and replacing it with more advanced BIOS. The data on the chip can be updated using specially designed software, which could fix problems or add additional features to the motherboard.

Major manufactures of Basic I/O System include:

- American Megatrends Inc (AMI)
- Ali
- Winbond

Functions of BIOS

There are four main functions of Basic I/O System. They are:

POST

Testing of computer hardware and also making sure that no error is there while loading the operating system.

Bootstrap Loader

Once the capable operating system is loaded, the BIOS passes the control to the operating system.

BIOS Drivers

Low level drivers that give computer basic control over computer's basic hardware.

BIOS Setup

Hardware settings can be configured using this setup including system settings such as system password, time and date.

What BIOS does on My Computer

There are various functions that Basic I/O System performs but its most important function is to load the operating system. The first instruction that a microprocessor needs, once the PC is switched on is provided by Basic I/O System. The usual sequence it follows are-

- Check the CMOS setup for custom settings
- Load the interrupt handlers and device drivers.
- Initialize registers and power management.
- Performs "Power On Self Test" (POST)
- Display system settings.
- Determine which devices are bootable.
- Initiate the bootstrap sequence.

The moment the computer is turned on, BIOS checks for RAM stored on Complementary Metal Oxide Semiconductor (CMOS). It provides detailed information pertaining to the system like current Date and Time of the system clock, Boot Sequence, types of Disk Drives. It uses this information to modify or supplement its default programming.

The Operating System (OS) cannot load without a BIOS. OS is stored in hard disk and BIOS loads the drivers for the hard disks and also loads some segments of OS like MBR (Master Boot Record), FAT (File Allocation Table), GPT (GUID Partition Table) into the memory. Once the drivers are loaded, OS loads and controls computer.

Boot Screen

The first thing that appears on the screen as soon as the computer is turned on is the display text describing things like the amount of memory installed on the computer, type of hard disk & so on during the booting process. Once CMOS is set up & interrupt handlers are checked, it's time to check whether video card is operational, it initializes the memory & graphic processor on the card. Next it displays some details about the system which includes information about:

- CPU Specifications
- The Floppy Drive & Hard Drive
- Memory Information
- BIOS Version
- Chipset, Motherboard Model Information
- Storage Device Information

Other special drivers, such as Small System Computers Interface (SCSI) are loaded into the adapter & the information is displayed. It, then looks for sequence of storage devices identified as boot devices. Launching the operating system is called booting. If the first device on the list is not located, it checks for the next device on the list. If proper files are not located, the start-up process will be halted.

Configuring BIOS (Basic Input Output System)

There are different options for its configuration, they vary from one BIOS manufacturer to another. In order to configure BIOS, you must first enter the set-up window. Once that is done, you can enter BIOS configuration window, where we get to see different sections listed at the top of the screen.

The options that appear in all Basic I/O System configuration windows are as following:

- System Time and Date
- Boot Sequence
- Plug and Play
- Mouse and Keyboard

- Drive Configuration
 - Memory
 - Security
 - Power Management
- System Time and Date**

To set date and time for the system.

Boot Sequence

It refers to the order of devices that a computer search for bootable data.

Plug and Play

This feature is used to detect auto-connected devices. It should be set to 'YES', if your operating system and computer supports it.

Mouse and Keyboard

Enable 'Numlock' on startup, enable the keyboard and auto detect menus.

Drive Configuration

Configures hard drive, CD/DVD ROM

Memory

Direct the BIOS to shadow to a specific memory address.

Security

Setting up the password for accessing the computer.

Power Management

Select the amount of time for standby and suspend.

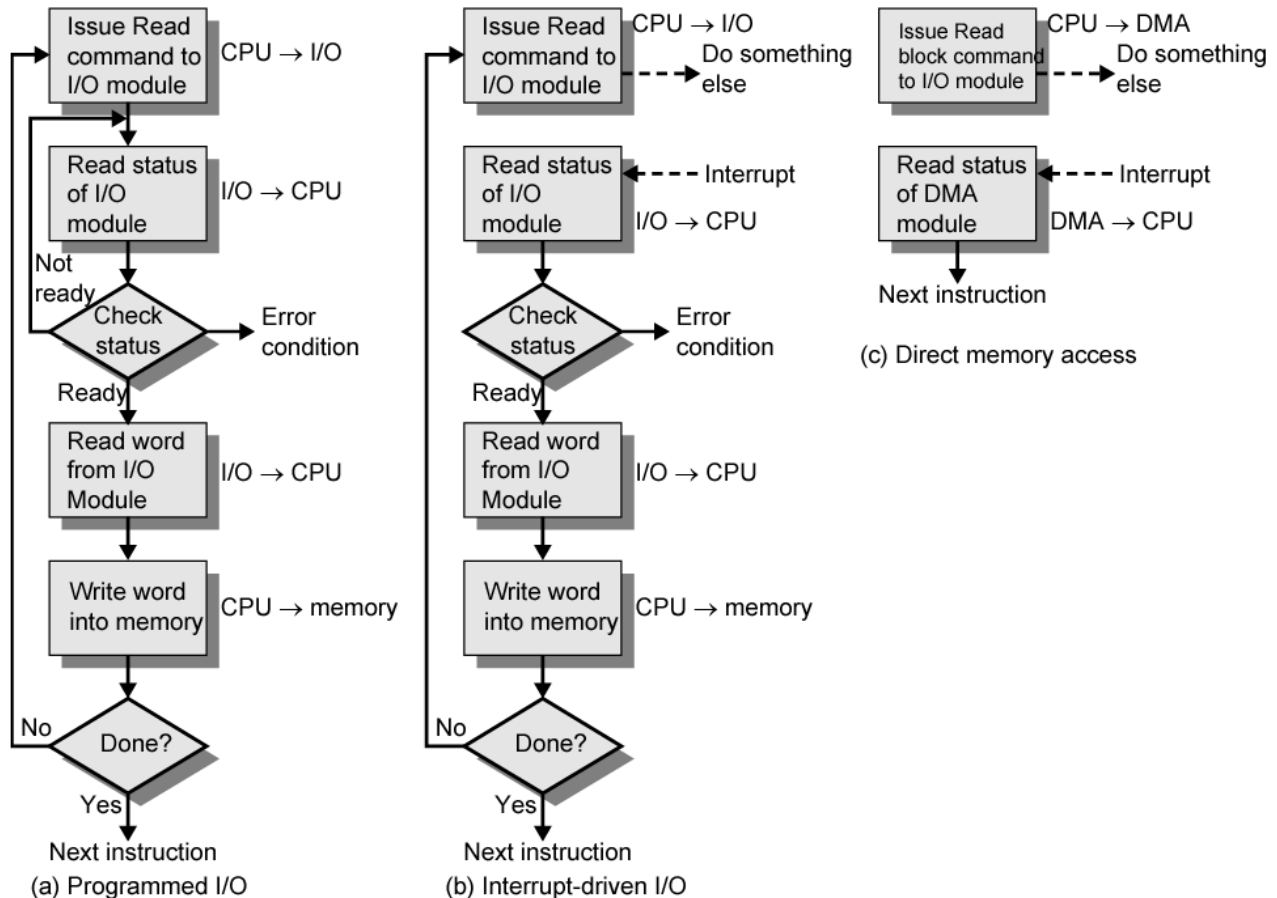
Modes of Data transfer

Data Transfer between the central computer and I/O devices may be handled in a variety of modes.

Some modes use CPU as an intermediate path, others transfer the data directly to and from the memory unit.

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

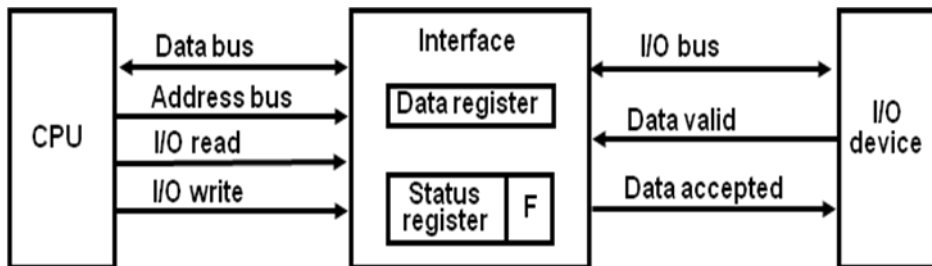
- Programmed I/O
- Interrupt Driven I/O
- Direct Memory Access (DMA)



Programmed I/O

- Programmed I/O operations are the result of I/O instructions written in the computer program.
- In programmed I/O, each data transfer is initiated by the instructions in the CPU and hence the CPU is in the continuous monitoring of the interface.
- Input instruction is used to transfer data from I/O device to CPU, store instruction is used to transfer data from CPU to memory and output instruction is used to transfer data from CPU to I/O device.
- This technique is generally used in very slow speed computer and is not an efficient method if the speed of the CPU and I/O is different.

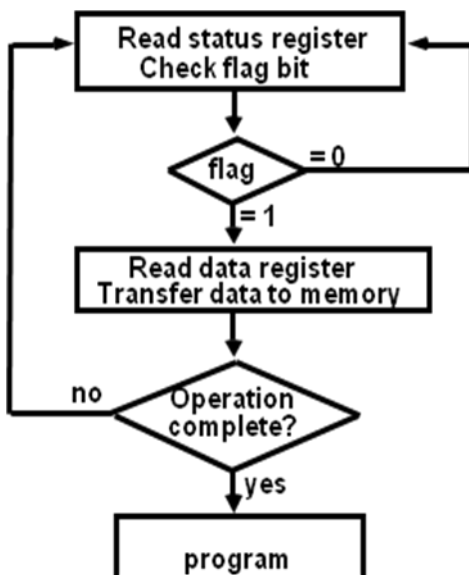
- Fig: Data transfer from I/O device to CPU



I/O device places the data on the I/O bus and enables its data valid signal

- The interface accepts the data in the data register and sets the F bit of status register and also enables the data accepted signal.
- Data valid line is disabled by I/O device.
- CPU is in a continuous monitoring of the interface in which it checks the F bit of the status register.
- If it is set i.e. 1, then the CPU reads the data from data register and sets F bit to zero
- If it is reset i.e. 0, then the CPU remains monitoring the interface.
- Interface disables the data accepted signal and the system goes to initial state where next item of data is placed on the data bus.

Fig: Flowchart for CPU program to input data



Characteristics:

- Continuous CPU involvement

- CPU slowed down to I/O speed
- Simple
- Least hardware

Polling, or polled operation, in computer science, refers to actively sampling the status of an external device by a client program as a synchronous activity. Polling is most often used in terms of input/output (I/O), and is also referred to as **polled I/O or software driven I/O. Computer**

Interrupt-driven I/O

- Polling takes valuable CPU time
- Open communication only when some data has to be passed -> *Interrupt*.
- I/O interface, instead of the CPU, monitors the I/O device
- When the interface determines that the I/O device is ready for data transfer, it generates an *Interrupt Request* to the CPU
- Upon detecting an interrupt, CPU stops momentarily the task it is doing, branches to the service routine to process the data transfer, and then returns to the task it was performing

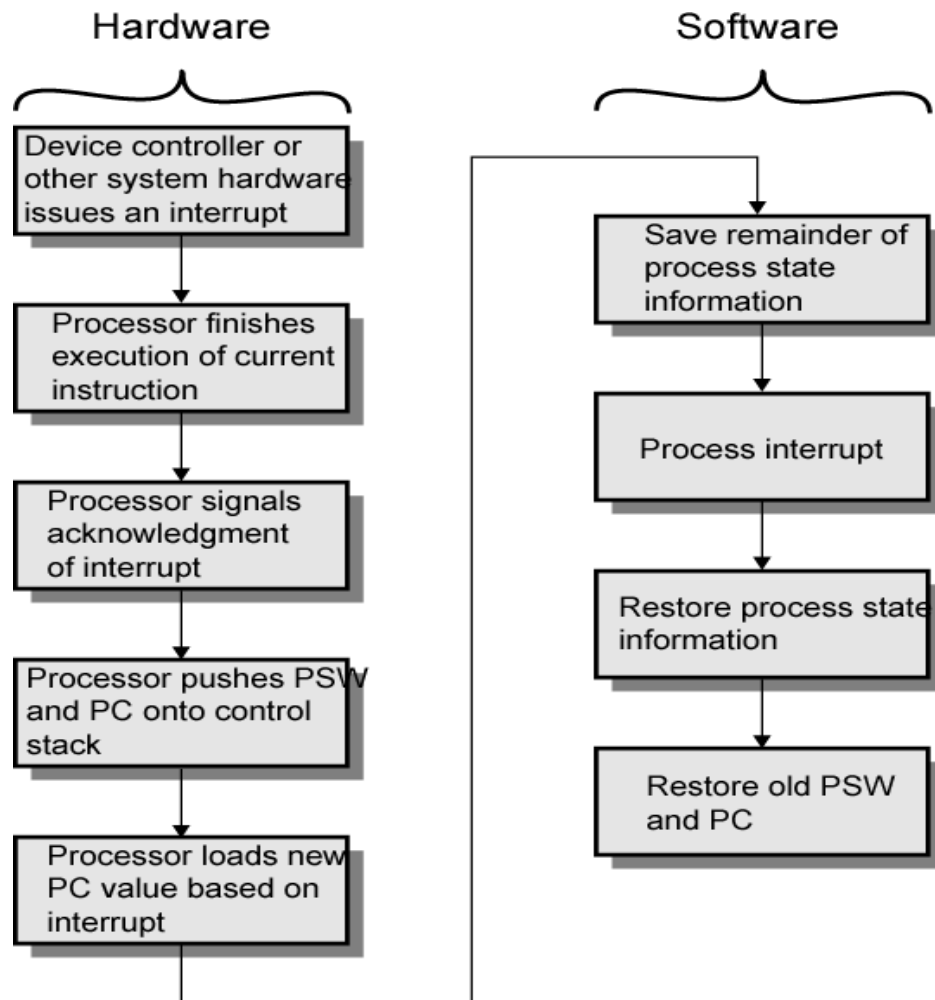
The problem with programmed I/O is that the processor has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data. The processor, while waiting, must repeatedly interrogate the status of the I/O module. As a result, the level of the performance of the entire system is severely degraded. An alternative is for the processor to issue an I/O command to a module and then go on to do some other useful work. The I/O module will then interrupt the processor to request service when it is ready to exchange data with processor. The processor then executes the data transfer, and then resumes its former processing. The interrupt can be initiated either by software or by hardware.

Interrupt Driven I/O basic operation

- CPU issues read command
- I/O module gets data from peripheral whilst CPU does other work
- I/O module interrupts CPU
- CPU requests data
- I/O module transfers data

Interrupt Processing from CPU viewpoint

- Issue read command
- Do other work
- Check for interrupt at end of each instruction cycle
- If interrupted:-
 - o Save context (registers)
 - o Process interrupt
 - o Fetch data & store

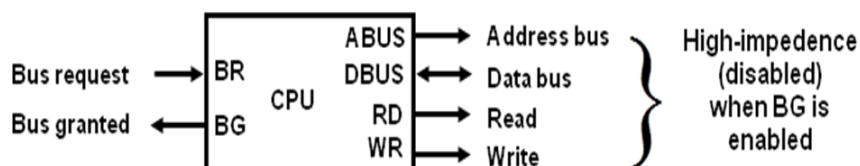


Direct Memory access

- Large blocks of data transferred at a high speed to or from high speed devices, magnetic drums, disks, tapes, etc.
- DMA controller Interface that provides I/O transfer of data directly to and from the memory and the I/O device
- CPU initializes the DMA controller by sending a memory address and the number of words to be transferred
- Actual transfer of data is done directly between the device and memory through DMA controller -> Freeing CPU for other tasks

The transfer of data between the peripheral and memory without the interaction of CPU and letting the peripheral device manage the memory bus directly is termed as Direct Memory Access (DMA).

Fig: CPU bus signal for DMA transfer



The two control signals Bus Request and Bus Grant are used to facilitate the DMA transfer. The bus request input is used by the DMA controller to request the CPU for the control of the buses. When BR signal is high, the CPU terminates the execution of the current instructions and then places the address, data, read and write lines to the high impedance state and sends the bus grant signal. The DMA controller now takes the control of the buses and transfers the data directly between memory and I/O without processor interaction. When the transfer is completed, the bus request signal is made low by DMA. In response to which CPU disables the bus grant and again CPU takes the control of address, data, read and write lines.

The transfer of data between the memory and I/O of course facilitates in two ways which are DMA Burst and Cycle Stealing.

DMA Burst: The block of data consisting a number of memory words is transferred at a time.

Cycle Stealing: DMA transfers one data word at a time after which it must return control of the buses to the CPU.

- CPU is usually much faster than I/O (DMA), thus CPU uses the most of the memory cycles
- DMA Controller steals the memory cycles from CPU
- For those stolen cycles, CPU remains idle
- For those slow CPU, DMA Controller may steal most of the memory cycles which may cause CPU remain idle long time

DMA Controller : The DMA controller communicates with the CPU through the data bus and control lines. DMA select signal is used for selecting the controller, the register select is for selecting the register. When the bus grant signal is zero, the CPU communicates through the data bus to read or write into the DMA register. When bus grant is one, the DMA controller takes the control of buses and transfers the data between the memory and I/O.

DMA Transfer

- DMA request signal is given from I/O device to DMA controller.
- DMA sends the bus request signal to CPU in response to which CPU disables its current instructions and initialize the DMA by sending the following information.
- The starting address of the memory block where the data are available (for read) and where data to be stored (for write)
- The word count which is the number of words in the memory block
- Control to specify the mode of transfer
- Sends a bus grant as 1 so that DMA controller can take the control of the buses
- DMA sends the DMA acknowledge signal in response to which peripheral device puts the words in the data bus (for write) or receives a word from the data bus (for read).

Fig: DMA transfer in a computer system

Q10. What is the difference between programmed I/O and DMA.

Long answer questions

Q11. Explain Configuration of system in detail.

Q12. Explain various functions of BIOS in detail.

Q13. Explain DMA data transfer operation.

Q14. Explain the various modes of data transfer.

UNIT IV

ARCHITECTURE OF MULTIPROCESSOR SYSTEM

***Parallel processing** is a computing technique when multiple streams of calculations or data processing tasks co-occur through numerous central processing units (CPUs) working concurrently. This article explains how parallel processing works and examples of its application in real-world use cases.*

Parallel processing uses two or more processors or CPUs simultaneously to handle various components of a single activity. Systems can slash a program's execution time by dividing a task's many parts among several processors. Multi-core processors, frequently found in modern computers, and any system with more than one CPU are capable of performing parallel processing.

For improved speed, lower power consumption, and more effective handling of several activities, multi-core processors are integrated circuit (IC) chips with two or more CPUs. Most computers can have two to four cores, while others can have up to twelve. Complex operations and computations are frequently completed in parallel processing.

At the most fundamental level, the way registers are used distinguishes between parallel and serial operations. Shift registers operate serially, processing each bit one at a time, whereas registers with parallel loading process each bit of the word simultaneously. It is possible to manage parallel processing at a higher level of complexity by using a variety of functional units that perform the same or different activities simultaneously.

Types of Parallel Processing

Single Instruction, Single Data (SISD)

In the type of computing called Single Instruction, Single Data (SISD), a single processor is responsible for simultaneously managing a single algorithm as a single data source. A computer organization having a control unit, a processing unit, and a memory unit is represented by SISD. It is similar to the current serial computer. Instructions are carried out sequentially by SISD, which may or may not be capable of parallel processing, depending on its configuration.

Sequentially carried-out instructions may cross over throughout their execution phases. There may be more than one functional unit inside a SISD computer. However, one control unit is in charge of all functional units. Such systems allow for pipeline processing or using numerous functional units to achieve parallel processing.

Multiple Instruction, Single Data (MISD)

Multiple processors are standard in computers that use the Multiple Instruction, Single Data (MISD) instruction set. While using several algorithms, all processors share the same input data. MISD computers can simultaneously perform many operations on the same batch of data. As expected, the number of operations is impacted by the number of processors available.

The MISD structure consists of many processing units, each operating under its instructions and over a comparable data flow. One processor's output becomes the input for the following processor. This organization's debut garnered little notice and wasn't used in architecture.

Single Instruction, Multiple Data (SIMD)

Computers that use the Single Instruction, Multiple Data (SIMD) architecture have multiple processors that carry out identical instructions. However, each processor supplies the instructions with its unique collection of data. SIMD computers apply the same algorithm to several data sets. The SIMD architecture has numerous processing components.

All of these components fall under the supervision of a single control unit. While processing numerous pieces of data, each processor receives the same instruction from the control unit. Multiple modules included in the shared subsystem aid in simultaneous communication with every CPU. This is further separated into organizations that use bit-slice and word-slice modes.

Multiple Instruction, Multiple Data (MIMD)

Multiple Instruction, Multiple Data, or MIMD, computers are characterized by the presence of multiple processors, each capable of independently accepting its instruction stream. These kinds of computers have many processors. Additionally, each CPU draws data from a different data stream. A MIMD computer is capable of running many tasks simultaneously.

Although MIMD computers are more adaptable than SIMD or MIMD computers, developing the sophisticated algorithms that power these machines is more challenging. Since all memory flows are changed from the shared data area transmitted by all processors, a MIMD computer organization incorporates interactions between the multiprocessors.

The multiple SISD operation is equivalent to a collection of separate SISD systems if the many data streams come from various shared memories.

Single Program, Multiple Data (SPMD)

SPMD systems, which stand for Single Program, Multiple Data, are a subset of MIMD. Although an SPMD computer is constructed similarly to a MIMD, each of its processors is responsible for carrying out the same instructions. SPMD is a message passing programming used in distributed memory computer systems. A group of separate computers, collectively called nodes, make up a distributed memory computer.

Each node launches its application and uses send/receive routines to send and receive messages when interacting with other nodes. Systems can also use messages to provide barrier synchronization. It is possible to transfer the messages via a wide range of communication techniques, such as [transmission control protocol \(TCP/IP\)](#) over Ethernet and specialized high-speed interconnects, such as Supercomputer Interconnect and Myrinet.

Massively Parallel Processing (MPP)

A storage structure called Massively Parallel Processing (MPP) is made to manage the coordinated execution of program operations by numerous processors. With each CPU using its operating system and memory, this coordinated processing can be applied to different program sections.

As a result, MPP databases can handle enormous amounts of data and deliver analyses based on large datasets considerably faster. MPP processors typically communicate through a messaging interface and can have up to 200 or more processors working on an application. It functions by enabling the transmission of messages between processes via a set of corresponding data links.

The most common types of computers used in parallel processing systems are SIMD and MIMD. Although SISD computers can't run in parallel on their own, a cluster can be

created by connecting many of them. In a more extensive parallel system, the CPU of each computer can function as a processor. The computers work as a single supercomputer when used collectively. [Grid computing](#) is the name of this method.

Characteristics of multiprocessors

- A multiprocessor system is an interconnection of two or more CPUs with memory and input-output equipment.
- The term “processor” in multiprocessor can mean either a central processing unit (CPU) or an input-output processor (IOP).
- Multiprocessors are classified as *multiple instruction stream, multiple data stream*
 - (MIMD) systems
- Multiprocessing improves the reliability of the system.
- The benefit derived from a multiprocessor organization is an improved system performance.
 - Multiple independent jobs can be made to operate in parallel.
 - A single job can be partitioned into multiple parallel tasks.
- Multiprocessing can improve performance by decomposing a program into parallel executable tasks.
 - The user can explicitly declare that certain tasks of the program be executed in parallel.
 - This must be done prior to loading the program by specifying the parallel executable segments.
 -

The other is to provide a compiler with multiprocessor software that can automatically detect parallelism in a user's program.

Multiprocessor are classified by the way their memory is organized.

- A multiprocessor system with *common shared memory* is classified as a *shared-memory* or *tightly coupled multiprocessor*.
 - Tolerate a *higher degree* of interaction between tasks.
- Each processor element with its own *private local memory* is classified as a *distributed-memory* or *loosely coupled system*.
 - Are most efficient when the interaction between tasks is *minimal*

Interconnection Structures

- The components that form a multiprocessor system are CPUs, IOPs connected to input-output devices, and a memory unit.
- The interconnection between the components can have different physical configurations, depending on the number of transfer paths that are available
 - Between the processors and memory in a shared memory system
 - Among the processing elements in a loosely coupled system
- There are several physical forms available for establishing an interconnection network.
 - Time-shared common bus

- Multiport memory
- Crossbar switch
- Multistage switching network
- Hypercube system

Time Shared Common Bus

- A common-bus multiprocessor system consists of a number of processors connected through a common path to a memory unit.
- *Disadv.:*
 - Only one processor can communicate with the memory or another processor at any given time.
 - As a consequence, the total overall transfer rate within the system is limited by the speed of the single path
- A more economical implementation of a dual bus structure is depicted in Fig. below.
- Part of the local memory may be designed as a *cache memory* attached to the CPU.

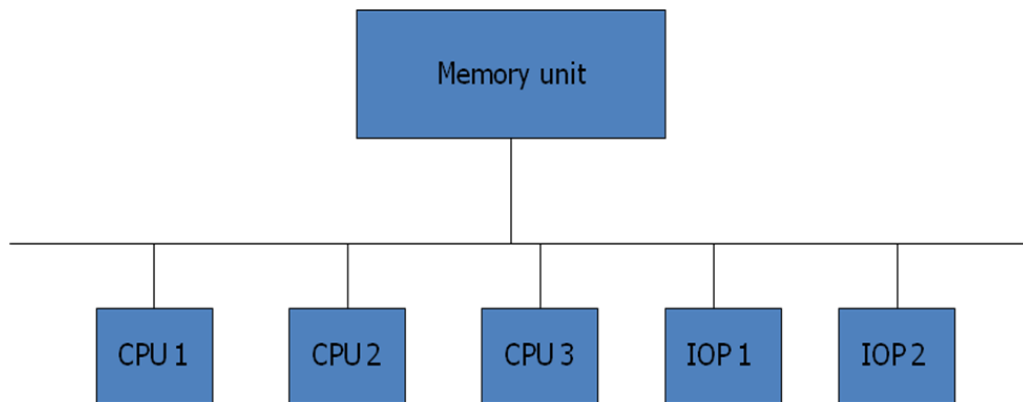


Fig: Time shared common bus organization

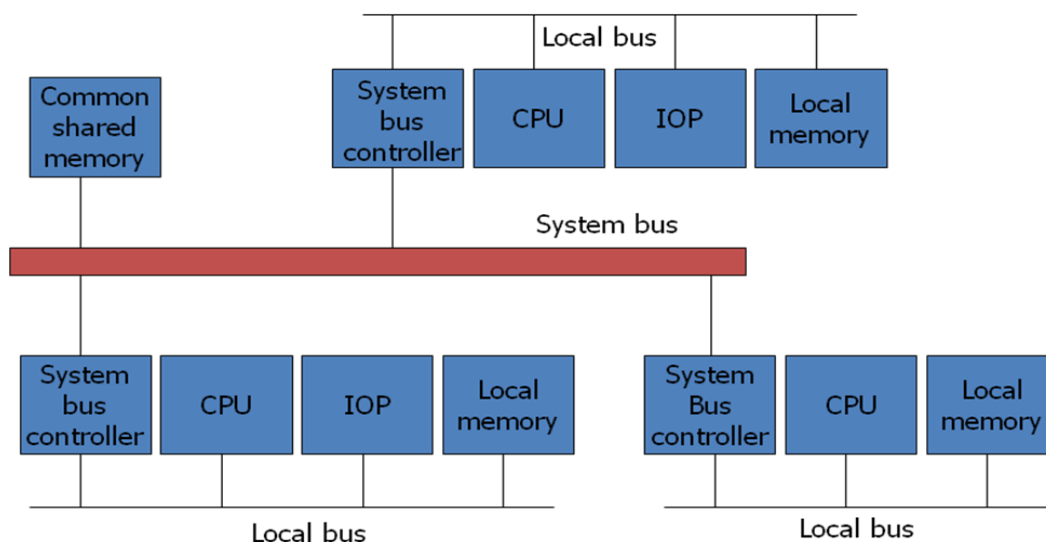


Fig: System bus structure for multiprocessors

Multiport Memory

- A multiport memory system employs separate buses between each memory module and each CPU.
- The module must have internal control logic to determine which port will have access to memory at any given time.
- Memory access conflicts are resolved by assigning fixed priorities to each memory port.
- Adv.:
 - The high transfer rate can be achieved because of the multiple paths.
- Disadv.:
 - It requires expensive memory control logic and a large number of cables and connections

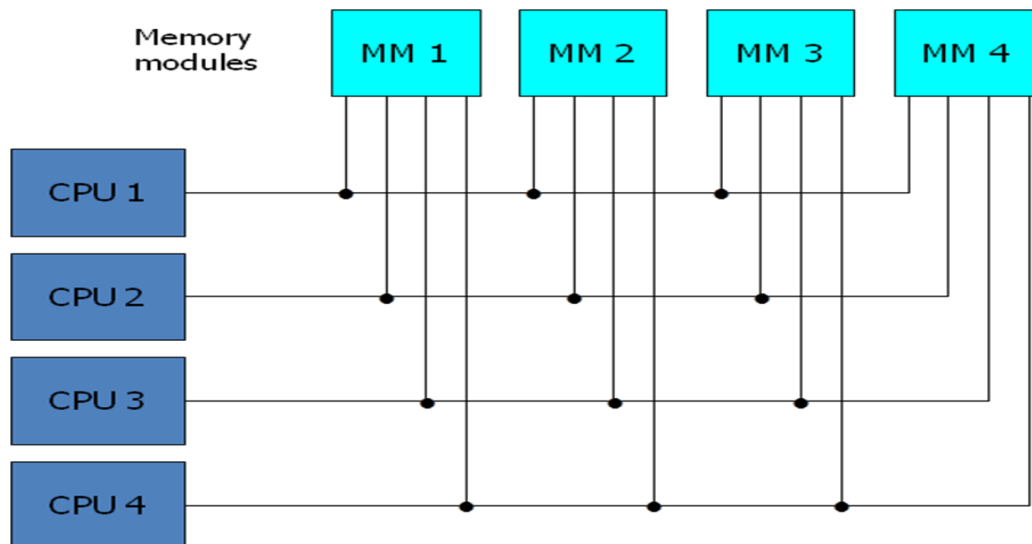


Fig: Multiport memory organization

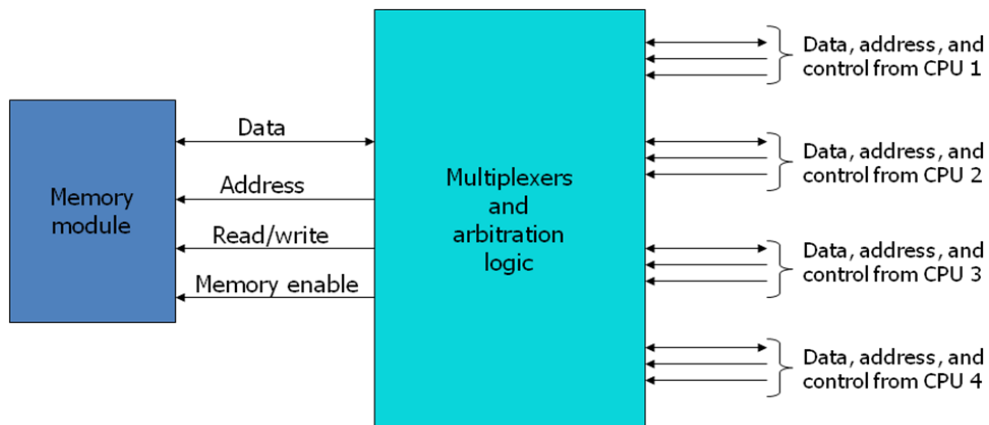
Crossbar Switch

- Consists of a number of *crosspoints* that are placed at intersections between processor buses and memory module paths.
- The small square in each crosspoint is a *switch* that determines the path from a processor to a memory module.
- Adv.:
 - Supports simultaneous transfers from all memory modules
- Disadv.:
 - The hardware required to implement the switch can become quite

large and complex.

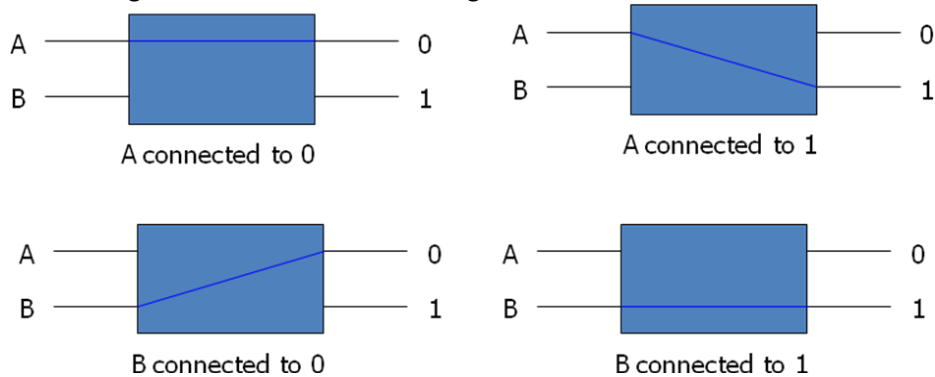
fig. shows the functional design of a crossbar switch connected to one memory module

Fig: Crossbar switch

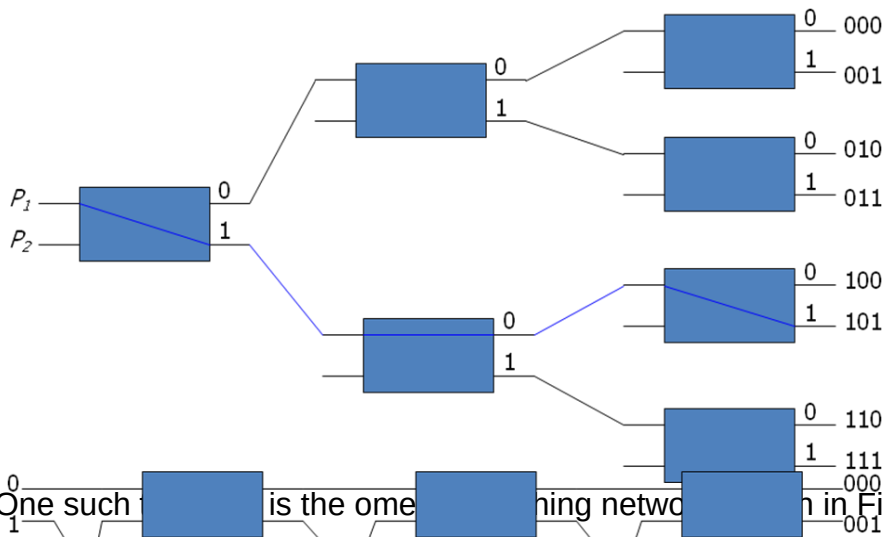


Multistage Switching Network

- The basic component of a multistage network is a two-input, two-output interchangeswitch as shown in Fig. below.



- Using the 2x2 switch as a building block, it is possible to build a multistage network to control the communication between a number of sources and destinations.
 - To see how this is done, consider the binary tree shown in Fig. below.
 - Certain request patterns cannot be satisfied simultaneously. i.e., if $P_1 \square 000\sim 011$, then $P_2 \square 100\sim 111$



- One such network is the omega switching network shown in Fig. below

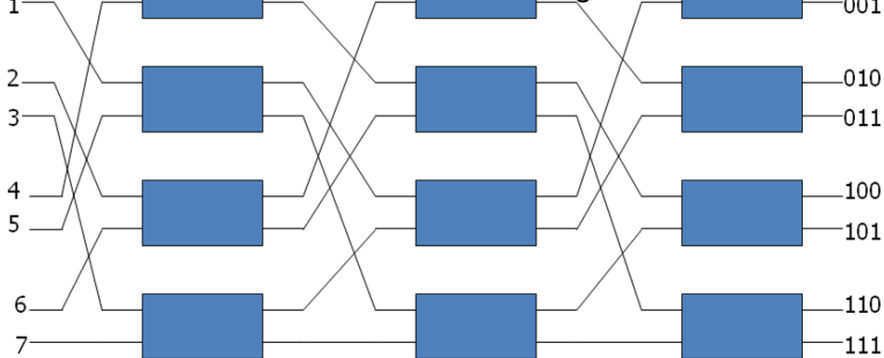


Fig: 8 x 8 Omega Switching Network

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

Hypercube System

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

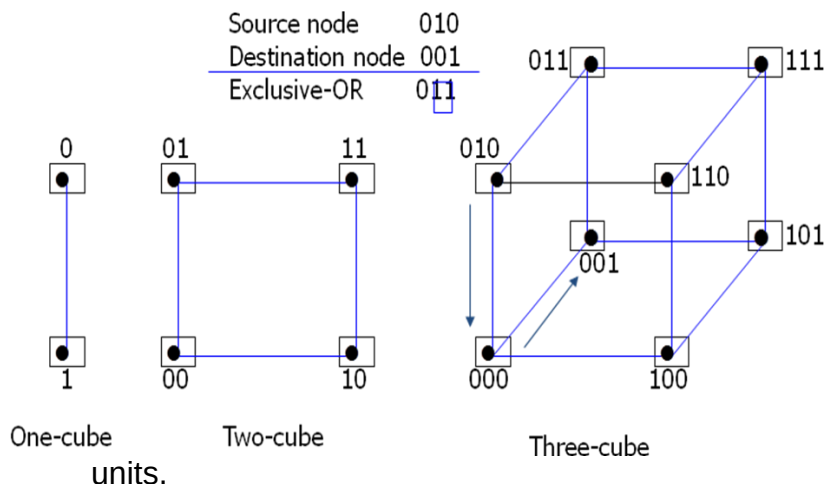


Fig: Hypercube structures for $n=1,2,3$

One mark questions

- Q1. Define Multiprocessor
- Q2. Define MPP
- Q3. Give the full form of MIMD
- Q4. Define SPMD system
- Q5. Define parallel processing
- Q6. Name any two interconnection structures
- Q7. Give the full form of SIMD

Short answer questions

- Q8. Briefly explain types of parallel processing.
- Q9. List the characteristics of multiprocessors
- Q10. Briefly explain multiport memory.

Long answer questions

- Q11. Explain interconnection structures of multiprocessor system in detail.

UNIT V

I/O INTERFACE

The input/output devices are used by a computer system to provide an interface with the external environment i.e., humans and other devices, which are connected to a computer. The major components of a typical microcomputer system are *microprocessor/CPU*, *memory system* and an *I/O interface*. These components are connected through buses. The primary function of the system bus is to transfer control information, addresses, instructions, and data between these components. Figure 5.1 depicts the block diagram of a typical microcomputer system.

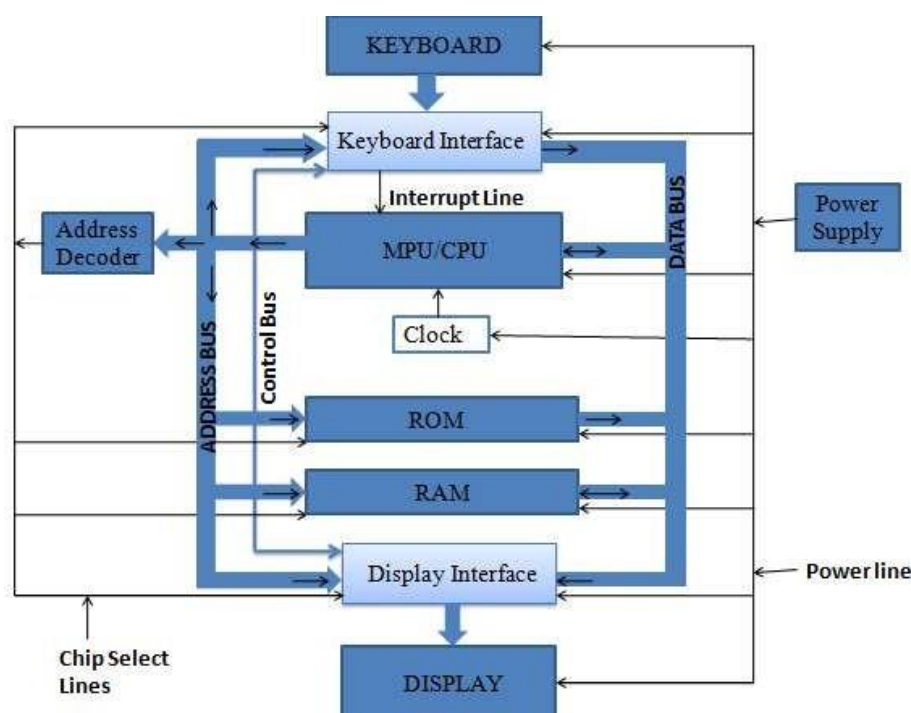


Figure 5.1: A Typical Microcomputer System's Block Diagram

The microcomputer, as shown in Figure 7.1, has a single micro processing unit (MPU). It also has RAM and ROM, which may be constructed by a number of RAM and ROM chips. The diagram also shows two basic interfaces, viz. keyboard interface and display interface, which are connected through the system bus. You may please note that other I/O interface may be connected in a similar way. Please also note that the system bus has been shown in the diagram as three distinct buses, viz. control bus, address bus, and data bus.

An Input/Output (I/O) subsystem includes all the input/output interfaces and connected Input/output devices. The basic objective of an I/O subsystem is to provide an efficient communication medium

between the computer system and the external environment (humans and other devices). An I/O interface is used to connect an external I/O device with the computer system. The I/O interface interchanges control, status and data with the external device. The I/O interfaces can also be used to transfer instruction/data/control within the computer units, including processor registers and memory units. An I/O device attached to the computer is also known as **peripheral or external device**. The external devices may be categorized on the basis of their communication endpoints as:

- **Human readable:** These devices provide information in human readable form. Example- *display terminals, printers, etc.*
- **Machine-readable:** These devices provide information in machine readable form. Example- *magnetic disks, CD-RW, etc.*
- **Communication:** These devices provide information to communication devices such as *MODEM*.

An I/O interface (also known as **I/O Module**) is used to transfer information between internal memory and peripheral devices. Each peripheral device has its own set of characteristics. An I/O interface helps in resolving the differences between the processing unit components and peripherals.

A hardware component between the peripheral devices and CPU is required to control and synchronize all input/output operations in the computer systems. These hardware components, which are used to provide an interface between the peripheral devices and the processor, are known as **interface units**. An I/O interface acts as a bridge between the processor and peripheral devices.

The I/O interface offers an interface which connects the internal components i.e., processor and main memory as well as external components i.e., peripheral or external devices. In addition to data transfer between processor to I/O devices, it also establishes the coordination between them. Moreover, the I/O interface also has components like **buffer system** and **error detection** mechanism to deal with the speed differences between processor and peripheral devices.

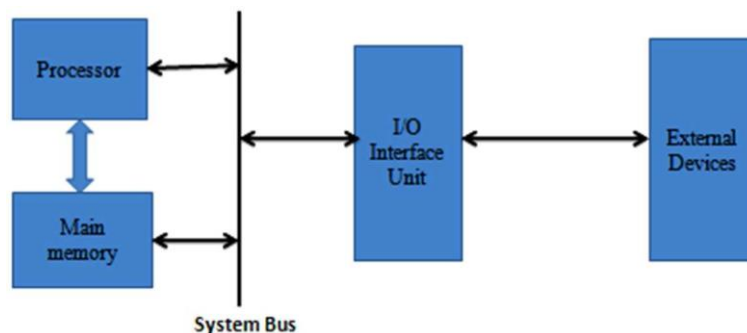


Figure 5.2: A Block Diagram of an I/O Interface connecting with Processor and External Devices

Roles of I/O Interface

The major roles of I/O interfaces are:

1. Communication with the processor

To provide data transfer between processor and I/O interface, the control signals (Read/Write/Status), address of memory locations and actual data are required to exchange. The system bus works as a communication medium in order to facilitate the exchange of information/data between the processor and I/O interface. In system bus, the address bus is used to send addresses; the control bus for control signals whereas the data bus is employed for actual data transfer between processor and I/O

interface.

2. Synchronization of control and timing signals

The I/O interface shares system bus with memory in order to provide data transfer. The timing and control signals are required in order to synchronize the flow of data from peripheral devices to processor/memory and vice versa. For instance-An I/O interface may request for control of data bus for data transfer from a peripheral device to the processor.

3. Communication with the I/O device

To complete an I/O operation, the exchange of data between I/O device and I/O interface is necessary. An I/O operation may require sending status signals, commands, or data.

4. Provision for data buffering

Due to the speed mismatch between the I/O devices and processor, the facility for data storage for a temporary period is required. The I/O interface stores information in registers temporarily which is referred to as data buffering. An I/O operation occurs in short bursts and the data are temporarily stored in a buffer area of I/O interface until the peripheral device/processor is ready to receive the data. Therefore, it is not required to hold the system bus for I/O operation due to slower I/O devices.

5. Error detection mechanism

The I/O interface also provides an in-built mechanism for error detection to check the communication errors along with mechanical errors. The errors are reported to the processor using parity bits and other mechanisms for further actions. Some mechanical errors may occur in devices such as mechanical and electrical failures, printer's paper jam etc.

In Figure 5.3, the internal block diagram of I/O interface unit consisting of different signals for processor as well as for I/O device is shown.

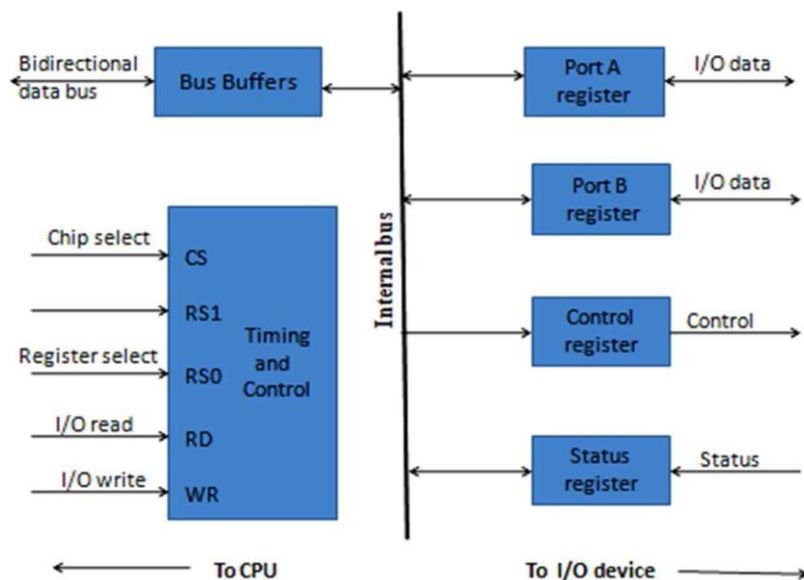
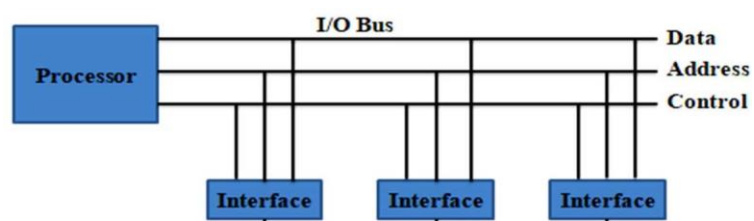


Figure 5.3: I/O Interface Unit's Block Diagram

System Bus and I/O Interface Modules

The control bus, data bus and address bus form a single bus which is known as system bus and it is used to establish the connection between the processor and I/O devices. Figure 7.4 depicts the communication links between the processor and several peripheral devices.



The I/O bus is connected to all peripheral devices through I/O interfaces. In order to communicate with an intended device, the processor sends the address of the device using address bus. Every I/O interface consists of an address decoder to continuously monitor the content of address bus. When an I/O interface observes the address of its own peripheral device, it activates the associated device and the bus; otherwise, the peripheral device is disabled. A control signal (*I/O command*) is also provided simultaneously through the control bus.

Asynchronous Data Transfer

We know that, the internal operations in individual unit of digital system are synchronized by means of clock pulse, means clock pulse is given to all registers within a unit, and all data transfer among internal registers occur simultaneously during occurrence of clock pulse. Now, suppose any two units of digital system are designed independently such as CPU and I/O interface.

And if the registers in the interface (I/O interface) share a common clock with CPU registers, then transfer between the two units is said to be synchronous. But in most cases, the internal timing in each unit is independent from each other in such a way that each uses its own private clock for its internal registers. In that case, the two units are said to be asynchronous to each other, and if data transfer occur between them this data transfer is said to be **Asynchronous Data Transfer**.

But, the Asynchronous Data Transfer between two independent units requires that control signals be transmitted between the communicating units so that the time can be indicated at which they send data.

This asynchronous way of data transfer can be achieved by two methods:

1. One way is by means of strobe pulse which is supplied by one of the units to other unit. When transfer has to occur. This method is known as “**Strobe Control**”.
2. Another method commonly used 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 signal to acknowledge receipt of the data. This method of data transfer between two independent units is said to be “**Handshaking**”.

The strobe pulse and handshaking method of asynchronous data transfer are not

restricted to I/O transfer. In fact, they are used extensively on numerous occasions requiring transfer of data between two independent units. So, here we consider the transmitting unit as source and receiving unit as destination.

As an example: The CPU, is the source during an output or write transfer and is the destination unit during input or read transfer.

And thus, the sequence of control during an asynchronous transfer depends on whether the transfer is initiated by the source or by the destination.

So, while discussing each way of data transfer asynchronously we see the sequence of control in both terms when it is initiated by source or when it is initiated by destination. In this way, each way of data transfer, can be further divided into parts, source initiated and destination initiated.

We can also specify, asynchronous transfer between two independent units by means of a timing diagram that shows the timing relationship that exists between the control and the data buses.

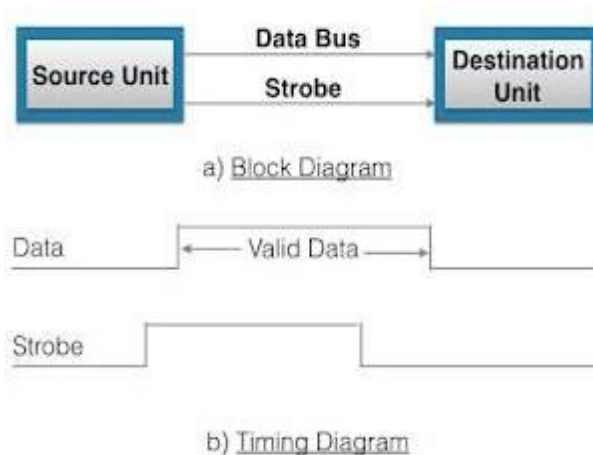
Now, we will discuss each method of asynchronous data transfer in detail one by one.

1. **Strobe Control:**

The Strobe Control method of asynchronous data transfer employs a single control line to time each transfer. This control line is also known as strobe and it may be achieved either by source or destination, depending on which initiates transfer.

Source initiated strobe for data transfer:

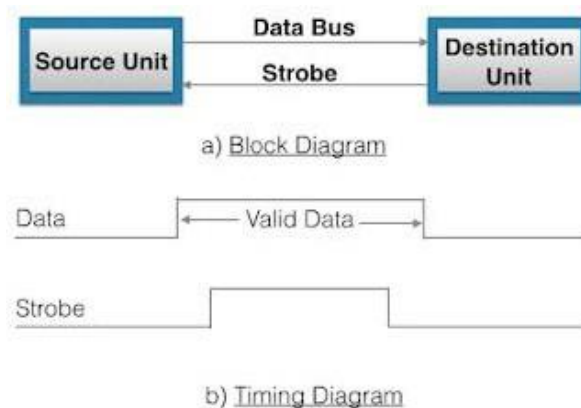
The block diagram and timing diagram of strobe initiated by source unit is shown in figure below:



In block diagram we see that strobe is initiated by source, and as shown in timing diagram, the source unit first places the data on the data bus. After a brief delay to ensure that the data settle to a steady value, the source activates a strobe pulse. The information on data bus and strobe control signal remain in the active state for a sufficient period of time to allow the destination unit to receive the data. Actually, the destination unit, uses a falling edge of strobe control to transfer the contents of data bus to one of its internal registers. The source removes the data from the data bus after it disables its strobe pulse. New valid data will be available only after the strobe is enabled again.

Destination-initiated strobe for data transfer:

The block diagram and timing diagram of strobe initiated by destination is shown in figure below:



In block diagram, we see that, the strobe initiated by destination, and as shown in timing diagram, the destination unit first activates the strobe pulse, informing the source to provide the data. The source unit responds by placing the requested binary information on the data bus. The data must be valid and remain in the bus long enough for the destination unit to accept it. The falling edge of strobe pulse can be used again to trigger a destination register. The destination unit then disables the strobe. And source removes the data from data bus after a per determine time interval.

Now, actually in computer, in the first case means in strobe initiated by source - the strobe may be a memory-write control signal from the CPU to a memory unit. The source, CPU, places the word on the data bus and informs the memory unit, which is the destination, that this is a write operation.

And in the second case i.e, in the strobe initiated by destination - the strobe may be a memory read control from the CPU to a memory unit. The destination, the CPU, initiates the read operation to inform the memory, which is a source unit, to place selected word into the data bus.

2. Handshaking:

The disadvantage of strobe method is that source unit that initiates the transfer has no way of knowing whether the destination has actually received the data that was

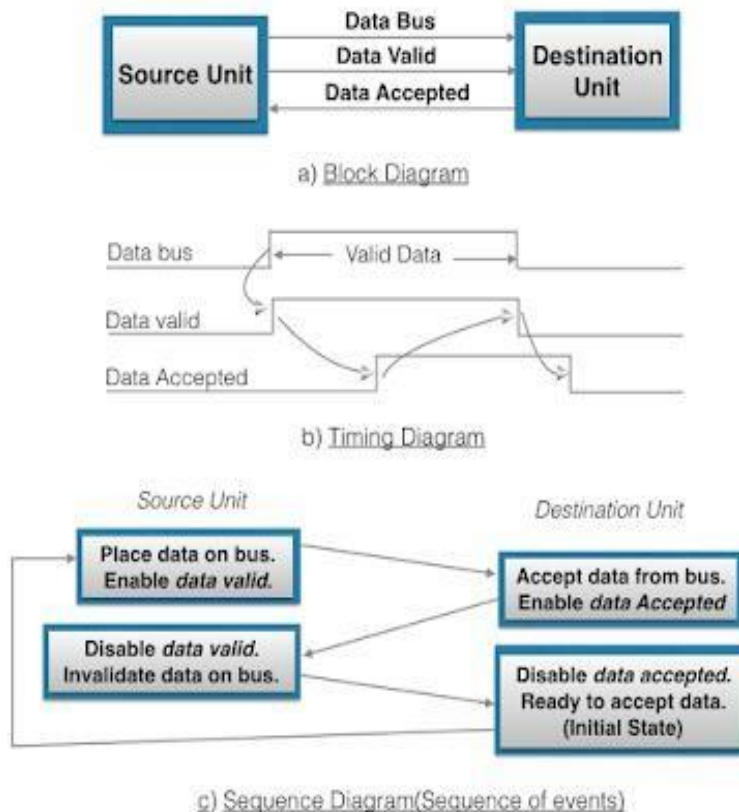
placed in the bus. Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit, has actually placed data on the bus. This problem can be solved by handshaking method.

Hand shaking method introduce a second control signal line that provides a replay to the unit that initiates the transfer.

In it, one control line is in the same direction as the data flow in the bus from the source to destination. It is used by source unit to inform the destination unit whether there are valid data in the bus. The other control line is in the other direction from destination to the source. It is used by the destination unit to inform the source whether it can accept data. And in it also, sequence of control depends on unit that initiate transfer. Means sequence of control depends whether transfer is initiated by source and destination. Sequence of control in both of them are described below:

Source initiated Handshaking:

The source initiated transfer using handshaking lines is shown in figure below:



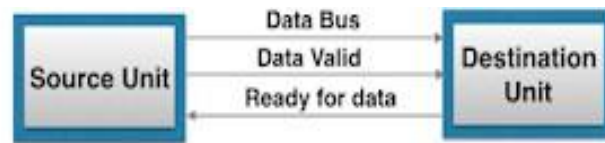
In its block diagram, we see that two handshaking lines are "data valid", which is generated by the source unit, and "data accepted", generated by the destination unit.

The timing diagram shows the timing relationship of exchange of signals between the two units. Means as shown in its timing diagram, the source initiates a transfer by placing data on the bus and enabling its data valid signal. The data accepted signal is then activated by destination unit after it accepts the data from the bus. The source unit then disables its data valid signal which invalidates the data on the bus. After this, the destination unit disables its data accepted signal and the system goes into initial state. The source unit does not send the next data item until after the destination unit shows its readiness to accept new data by disabling the data accepted signal.

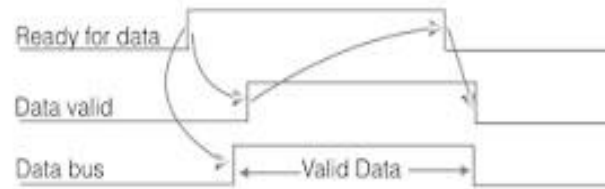
This sequence of events described in its sequence diagram, which shows the above sequence in which the system is present, at any given time.

Destination initiated handshaking:

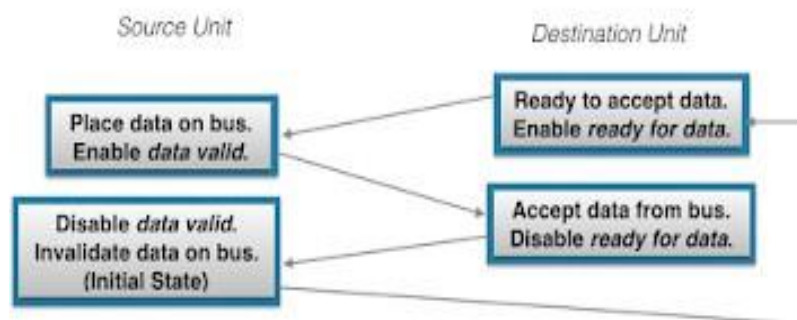
The destination initiated transfer using handshaking lines is shown in figure below:



a) Block Diagram



b) Timing Diagram



c) Sequence Diagram(sequence of events)

In its block diagram, we see that the two handshaking lines are "data valid", generated by the source unit, and "ready for data" generated by destination unit. Note that the name of signal data accepted generated by destination unit has been changed to ready for data to reflect its new meaning.

In it, transfer is initiated by destination, so source unit does not place data on data bus until it receives ready for data signal from destination unit. After that, hand shaking process is some as that of source initiated.

The sequence of event in it are shown in its sequence diagram and timing relationship between signals is shown in its timing diagram.

Thus, here we can say that, sequence of events in both cases would be identical. If we consider ready for data signal as the complement of data accept. means, the only difference between source and destination initiated transfer is in their choice of initial state.

One mark questions

Q1. Define Microprocessor.

Q2. Define I/O interface

Q3. What is a microcomputer

- Q4. Define handshaking.
- Q5. What is asynchronous data transfer?
- Q6. What is synchronous data transfer?

Short answer questions

- Q7. Explain strobe control method of asynchronous data transfer.
- Q8. Briefly explain handshaking.
- Q9. Draw block diagram of I/O interface
- Q10. List the various roles of I/O interfaces.
- Q11. List categories of external devices based on their communication endpoints.

Long answer questions

- Q12. Explain various methods of asynchronous data transfer.
- Q13. Explain I/O interface in detail.
- Q14. Explain various roles of I/O interface in detail.
- Q15. Describe the microcomputer system in detail.