





# The floating-point representation of a number has two parts:-

(i) Mantissa :-

Signed, fixed point number

(ii) Exponent :-

Designates the position of the decimal (or binary) point.

E.g:- + 6132.789 is represented in floating-point as :-

Fraction	Exponent
+0.6132789	+04

The value of the exponent indicates that the actual position of the decimal point is four positions to right of the indicated decimal point in fraction.

E.g:- The binary number +1001.11 is represented with an 8-bit fraction and 6-bit exponent as follows:-

Fraction	Exponent
01001110	000100

sign-bit

The fraction has a 0 in the leftmost position to denote positive.

The binary point of the fraction follows the sign bit but is not shown in the register.



### Normalized Floating Point Number :-

A floating-point number is said to be normalized if the most significant digit of the mantissa is non-zero.

E.g.:- The decimal number 350 is normalized but 000350 is not.

# Normalized numbers provide the maximum possible precision for the floating point number.

# A zero can not be normalized because it does not have a non-zero digit. It is usually represented in floating-point by all 0's in the mantissa and exponent.

[www.ankurgupta.net](http://www.ankurgupta.net)

[www.ankurgupta.net](http://www.ankurgupta.net)



Micro-operation :-

The operations executed on data stored in registers are called micro operations.

A micro-operation is an elementary operation.

E.g.:- shift, count, clear and load.

Register Transfer Language :-

The symbolic notation used to describe the microoperation transfers among registers is called a Register Transfer Language (RTL).

[www.ankurgupta.net](http://www.ankurgupta.net)

# The internal hardware organization of a digital computer is best defined by specifying:-

- (i) The set of registers it contains and their function.
- (ii) The sequence of micro-operations performed on the binary information stored in the registers.
- (iii) The control that initiates the sequence of micro-operations.



Registers:-

(i) Memory Address Register (MAR):-  
Holds an address of the memory unit.

(ii) PC:-  
Program Counter

(iii) IR:-  
Instruction Register

(iv) R<sub>i</sub>:-  
Processor Register.

→ Computer registers are designated by capital letters.

Register Transfer:-

$R_2 \leftarrow R_1$

Control Function:-

$P: R_2 \leftarrow R_1$

It represents the transfer operation to be executed by the hardware only if  $P = 1$ .

→ A control function is a boolean variable that is equal to 1 or 0.

Note:- The clock is not included as a variable in the register transfer statements. It is assumed that all transfers occur during a clock edge transition.

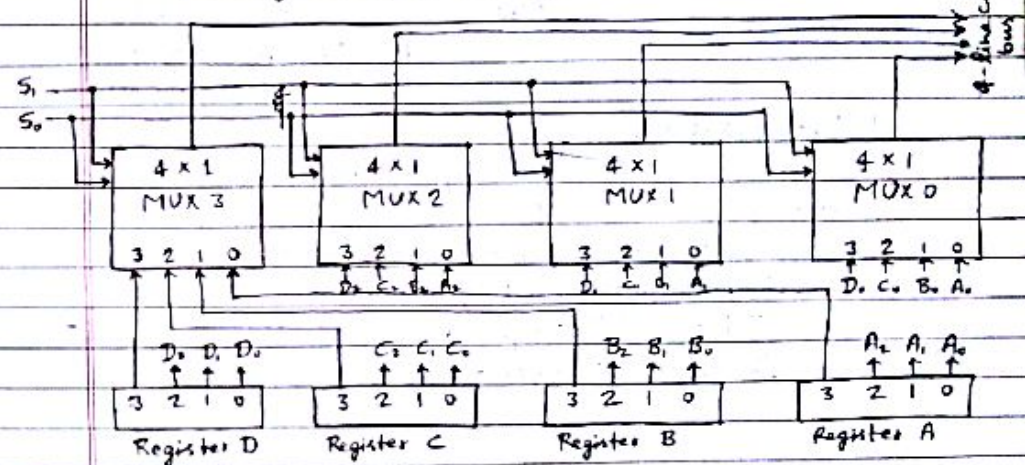
Bus and Memory Transfers:-

Common Bus:-

It consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time.

Control signals determine which register is selected by the bus during each particular register transfer.

\*\*\* Bus System for 4 Registers:-



\*\*\* → In general, a bus system will multiplex  $k$ -registers of  $n$ -bits each to produce an  $n$ -line common bus.

\*\*\* → The number of multiplexers needed to construct the bus is equal to  $n$ , the number of bits in each register.

\*\*\* → The size of each multiplexer must be  $k \times 1$ , since it multiplexes  $k$  data lines.



### Bus Transfer :-

$$\text{BUS} \leftarrow R_2, R_1 \leftarrow \text{BUS} \\ \Rightarrow R_1 \leftarrow R_2$$

### Memory Transfer :-

$$\text{Read : } DR \leftarrow M[AR]$$

$$\text{Write : } M[AR] \leftarrow R_1$$

where, DR  $\Rightarrow$  Data Register  
AR  $\Rightarrow$  Address Register

### Micro-operations :-

The microoperations are classified into four categories :-

- (i) Register Transfer Microoperations
- (ii) Arithmetic Microoperations
- (iii) Logic Microoperations
- (iv) Shift Microoperations

### Arithmetic Microoperations :-

$$(i) \text{ Addition :- } R_3 \leftarrow R_1 + R_2$$

$$(ii) \text{ Subtraction :- } R_3 \leftarrow R_1 - R_2 \\ \Rightarrow R_3 \leftarrow R_1 + \bar{R}_2 + 1$$

$$(iii) \text{ 1's complement :- } R_2 \leftarrow \bar{R}_2$$

$$(iv) \text{ 2's Complement :- } R_2 \leftarrow \bar{R}_2 + 1$$

$$(v) \text{ Increment :- } R_1 \leftarrow R_1 + 1$$

$$(vi) \text{ Decrement :- } R_1 \leftarrow R_1 - 1$$

(vii) Arithmetic Shift :- Explained later with shift microops

$\rightarrow$  The increment and decrement microoperations are implemented with a binary updown counter.



Logic Microoperations These operations consider each bit of the register separately and treat them as binary variables.

(iv) Complement :-  $R_1 \leftarrow \bar{R}_1$

→ The 'if' between P and Q

When it occurs in a control function, it will denote an OR operation.

$P + Q: R1 \leftarrow R2 + R3, R4 \leftarrow R5 \vee R6$

There are 16 different logic operations that can be performed with two binary variables.

[illegible]

There are three types of shifts:-

- (i) Logical Shift
- (ii) Circular Shift
- (iii) Arithmetic Shift

The serial input transfers a bit into the rightmost position.

The serial input transfers a bit into the leftmost position.

A logical shift is one that transfers 0 through the serial input.

shl and shr symbols are used for logical left shift and right shift microoperations.

E.g.:-  $R1 \leftarrow shr\ R1$   
 $R2 \leftarrow shl\ R2$

Here the serial output of the shift register is connected to its serial input.  $c_{il}$  and  $c_{ir}$  symbols are used for circular shift left and shift right microoperations.

E.g:-  $R1 \leftarrow \text{cir } R1$   
 $R2 \leftarrow \text{cir } R2$



### ## (iii) Arithmetic Shift:-

→ An arithmetic shift-left (ashl) multiplies a signed binary number by 2.

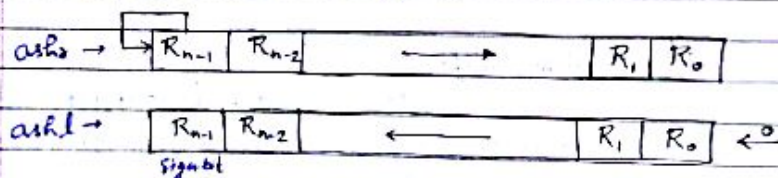
→ An arithmetic shift-right (ashr) divides the number by 2.

→ Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by 2.

→ The ashr leaves the sign bit unchanged and shifts the number (including the sign-bit) to the right.

→ The ashl inserts a 0 from right and shifts all other bits to the left.

If the multiplication by 2 causes an overflow, the sign-bit in the register changes from 0 to 1 or 1 to 0.



### Checking for overflow:-

If  $V_s = 0$ :-

There is no overflow.

If  $V_s = 1$ :-

There is an overflow.

$$V_s = R_{n-1} \oplus R_{n-2}$$

### Arithmetic Logic Shift Unit (ALU):-

To perform a microoperation in ALU, the contents of specified registers are placed in the inputs of the common ALU. The ALU performs an operation and the result of the operation is then transferred to a destination register.

# The ALU is a combinational circuit so that the entire operation can be performed during one clock pulse period.



## Problems

classmate

Date \_\_\_\_\_  
Page \_\_\_\_\_

Q:- If  $(P=1)$  then  $(R1 \leftarrow R2)$   
else if  $(Q=1)$  then  $(R1 \leftarrow R3)$

Represent above statements using control functions:-

Sol:-

P:  $R1 \leftarrow R2$

PQ:  $R1 \leftarrow R3$

Q:- A digital computer has a common bus system for 16 registers of 32-bits each. The bus is constructed with multiplexers.

(a) Each multiplexer will have 4 selection lines to select one of 16 registers.

(b) The size of multiplexers required is  $16 \times 1$ .

(c) 32 multiplexers will be required, one for each bit of the registers.

[www.ankurgupta.net](http://www.ankurgupta.net)

Q:-  $A = 10011100$ . Find  $\text{ashr}$  and  $\text{ashl}$ .

Sol:-

$\text{ashr } A \Rightarrow 11001110$

$\text{ashl } A \Rightarrow 00111000 \rightarrow \text{Overflow, because a negative number changed to positive.}$



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 bits of the instruction code are usually divided into three groups:-

- (i) An operation code field that specifies the operation to be performed.
- (ii) An address field that designates a memory address or a processor register.
- (iii) A mode field that specifies the way the operand or the effective address is determined.

Operation Code	Address Field	Mode
----------------	---------------	------

# A register address is a binary number of  $k$ -bits that define one of  $2^k$  registers in the CPU.

# The instruction code is also called as control word.

# An operation code is sometimes called a macrooperation because it specifies a set of micro-operations.



Types of CPU Organizations:-

The number of address fields in the instruction format of a computer depends on the internal organization of its registers.

Most of the computers fall into one of the three types of CPU organizations:-

- (i) Single Accumulator organization,
- (ii) General Register Organization,
- (iii) Stack Organization

(i) Single Accumulator Organization:-

All operations are performed with an implied accumulator register.

The instruction format in this type of computer uses one address field.

E.g.:- ADD X

$\Rightarrow AC \leftarrow AC + M[X]$

AC  $\rightarrow$  Accumulator Register.

(ii) General Register Organization:-

The instruction format in this type of computer uses either three or two address fields.

E.g.:- (i) ADD R1, R2, R3  $\Rightarrow R1 \leftarrow R2 + R3$

(ii) ADD R1, R2  $\Rightarrow R1 \leftarrow R1 + R2$

(iii) ADD R1, X  $\Rightarrow R1 \leftarrow R1 + M[X]$

(iv) MOV R1, R2  $\Rightarrow R1 \leftarrow R2$

(iii) Stack Organization:-

Computers with stack organization have PUSH and POP instructions which require one address field.

E.g.:- PUSH X

Operation-type instructions do not need an address field. \* (Zero address instruction).

E.g.:- ADD

# Single Accumulator Organization uses One-Address instructions.

# General Register Organization uses three and two-address instructions.

# Stack Organization uses zero and one-address instructions.

E.g.:- In a general register's CPU organization, there are eight general-purpose registers and ALU can perform 32-different operations.

The number of selection lines of each multiplexer for selecting the operand is :- 3 ( $2^3 = 8$ )

The number of bits in operation code = 5 ( $2^5 = 32$ )

The length of the control word =  $5 + 3 + 3 + 3 = 14$   
(for 3-address instruction)



## CISC :-

A computer with a large number of instructions is classified as a complex instruction set computer.

The essential goal of a CISC architecture is to attempt to provide a single machine instruction for each statement that is written in a high-level language.

The major characteristics of CISC architecture are:-

- (1) A large number of instructions - typically from 100 to 250 instructions.
- (2) Some instructions that perform specialized tasks and are used infrequently.
- (3) A large variety of addressing modes - typically from 5 to 20 different modes.
- (4) Variable-length instruction formats.
- (5) Instructions that manipulate operands in memory.

## RISC :-

The Reduced Instruction Set Computer is a computer containing fewer instructions with simple constructs that can be executed much faster within the CPU without having to use memory as often.

The major characteristics of a RISC processor are:-

- (1) Relatively few instructions.
- (2) Relatively few addressing modes.
- \* (3) Memory access limited to load and store instructions.
- (4) All operations done within the registers of the CPU.
- (5) Fixed-length, easily decoded instruction format.
- (6) Single-cycle instruction execution.
- (7) Hardwired rather than microprogrammed control.
- (8) Delayed Load → When there is a data dependency conflict in instruction pipeline.
- (9) Delayed Branch → When there is a branch instruction in instruction pipeline.



## Addressing Modes :-

The way the operands are chosen during program execution is dependent on the addressing mode of the instruction.

# The addressing mode may reduce the number of bits in the addressing field of the instruction.

Different types of addressing modes are given below :-

### (1) Implied Mode :-

In this mode the operands are specified implicitly in the definition of the instruction.

→ All register reference instructions that use only an accumulator are implied-mode instructions.

→ Zero-address instructions in a stack-organized computer are implied-mode instructions.

### (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.

E.g.:-  $LD \#NBR \Rightarrow AC \leftarrow NBR$

### (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.

~~E.g.:-~~ E.g.:-  $LD R1 \Rightarrow AC \leftarrow R1$

### (4) Register Indirect Mode :-

In this mode the instruction specifies a register that reside within the CPU whose contents give the address of the operand in memory.

E.g.:-  $LD (R1) \Rightarrow AC \leftarrow M[R1]$

### (5) Auto-increment or Autodecrement 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 the memory.

E.g.:-  $LD (R1) + \Rightarrow AC \leftarrow M[R1], R1 \leftarrow R1 + 1.$

### # Effective Address :-

The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode.



### (6) Direct Address Mode:-

In this mode the effective address is equal to the address part of the instruction.

E.g.:- LD ADR  $\Rightarrow AC \leftarrow M[ADR]$

### (7) Indirect Address Mode:-

In this mode the address field of the instruction gives the address where the effective address is stored in memory.

E.g.:- LD @ADR  ~~$\Rightarrow AC \leftarrow M[ADR]$~~   
 $\Rightarrow AC \leftarrow M[M[ADR]]$

### (8) Relative Address Mode:-

In this mode, the content of the program counter (PC) is added to the address part of the instruction in order to obtain the effective address.

E.g.:- LD \$ADR  $\Rightarrow AC \leftarrow M[PC + ADR]$

# Relative addressing is often used with branch-type instructions.

### (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 address field of the instruction defines the beginning address of a data array in memory.

E.g.:- LD ADR(X)  $\Rightarrow AC \leftarrow M[ADR + X]$

### (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.

E.g.:-

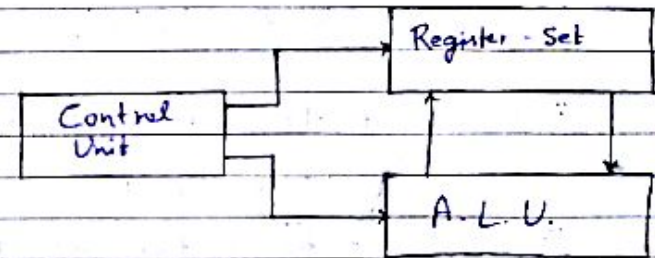
# The base register addressing mode is used for relocation of programs in memory.



## Central Processing Unit

# CPU is made up of three major parts:-

- (i) Register Set
- (ii) Arithmetic - logic unit (ALU)
- (iii) Control Unit (CU)



Some Fundamental Concepts:-

→ The instructions constituting a program to be executed are loaded in sequential locations in the main memory.

→ Instructions are fetched from successive memory locations until a branch or a jump instruction is executed.

→ A dedicated CPU register, referred to as the program counter (PC) contains the address of the memory location containing the next instruction.

→ After fetching an instruction, the contents of the PC are updated to point to the next instruction in sequence.



## Steps to execute an instruction:-

(i) Fetch the contents of the memory location pointed to by PC (program counter) and load them into IR (instruction register).

$$IR \leftarrow M[PC]$$

(ii) Increment the contents of PC by 1.

$$PC \leftarrow PC + 1$$

(iii) Carry out the actions specified by the instructions in the IR.

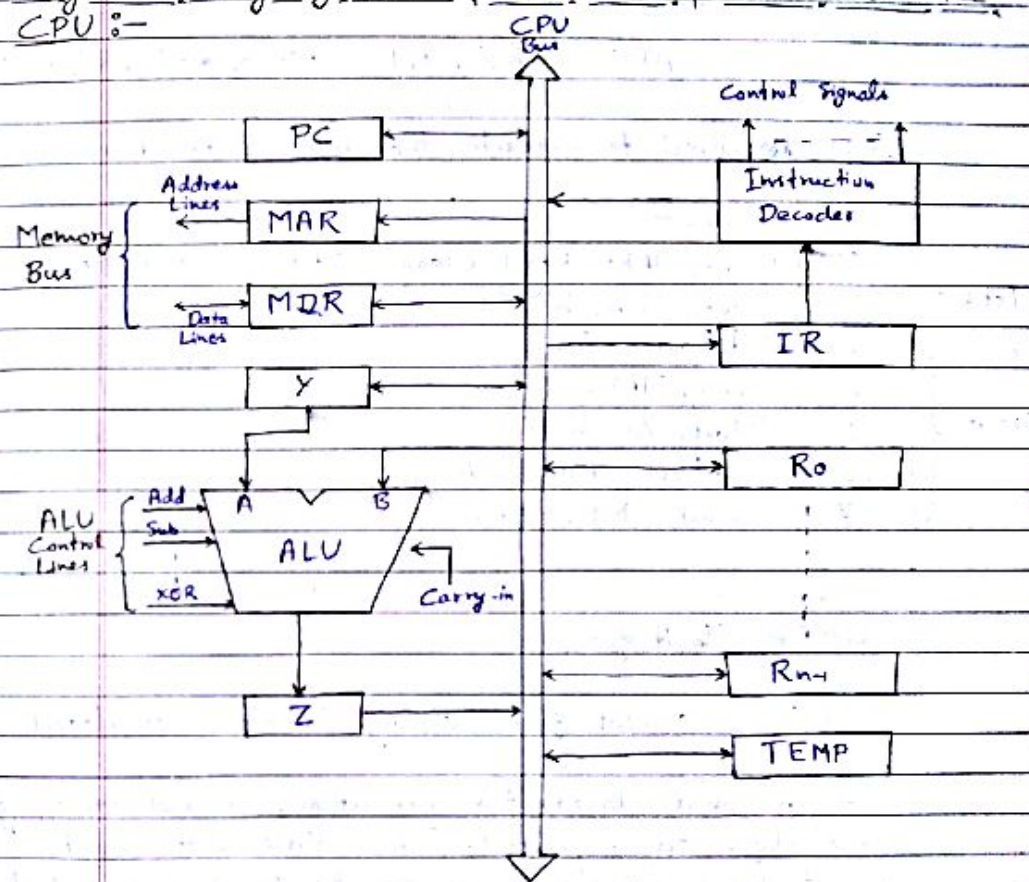
# Step 1 and 2 are usually referred to as the fetch phase.

# Step 3 constitutes the execution phase.

→ In cases where an instruction occupies more than one word, steps 1 and 2 must be repeated as many times as necessary to fetch the complete instruction.

→ The width of the data bus between the CPU and the main memory determines the number of bytes that can be transferred in one access. This width is likely to be 2, 4, or 8 bytes, so the PC must be incremented by 2, 4, or 8 after each fetch step.

## Single-bus organization of the data paths inside the CPU:-



# Here the single common bus is internal to the CPU, and should not be confused with the external bus.

# The ALU and the registers are used for storing and manipulating data and referred to as datapath.



## Execution of a complete instruction:-

ADD (R3), R1  $\Rightarrow R_1 \leftarrow M[R_3] + R_1$

Steps required to execute the above instruction are:-

	Step	Action
Fetch	1	PC <sub>out</sub> , MAR <sub>in</sub> , Read, Clear Y, Set Carry-in to ALU, Add, Z <sub>in</sub>
	2	Z <sub>out</sub> , PC <sub>in</sub> , WMFC
	3	MDR <sub>out</sub> , IR <sub>in</sub>
Execute	4	R3 <sub>out</sub> , MAR <sub>in</sub> , Read
	5	R1 <sub>out</sub> , Y <sub>in</sub> , WMFC
	6	MDR <sub>out</sub> , Add, Z <sub>in</sub>
	7	Z <sub>out</sub> , R1 <sub>in</sub> , End.

~~WMFC~~ ~~Wait for~~

WMFC  $\rightarrow$  Wait for Memory Function Completed.

# For above instruction execution, 7 non-overlapping time-slots are required. Each time slot must be at least long enough for the functions specified in the corresponding to be completed.

# Since a memory-access is slow, while a memory access is taking place, the CPU can perform other internal operations.

## CPU Control Design:-

There are two major types of control organization:-

- Hardwired Control
- Microprogrammed Control.

### (i) Hardwired Control Organization:-

In the hardwired organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits.

It has the advantage that it can be optimized to produce a fast mode of operation.

It requires changes in the wiring among the various components if the design has to be modified or changed.

### (ii) Micro-programmed Control Organization:-

In the micro-programmed organization, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of micro-operations.

In the micro-programmed control, any required changes or modifications can be done by updating the microprogram in control memory.



Micro-programmed Control:-

In micro-programmed control, the control signals are generated by a program similar to machine language programs.

Control Word:-

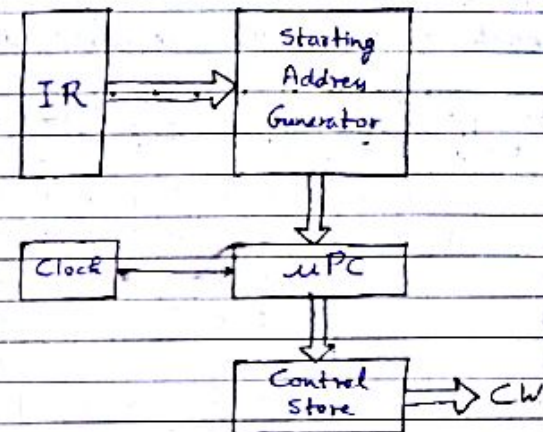
A control word (CW) is a word whose individual bits represent the various control signals.

# A sequence of CWs corresponding to the control sequence of a machine instruction constitutes the micro-routine.

# The individual CWs in this micro-routine are referred to as micro-instructions.

# The micro-routines for all instructions in the instruction set of a computer are stored in a special memory called the control store.

# The control unit can generate the control signals for any instruction by sequentially reading the CWs of the corresponding micro-routine from the control store.

Basic organization of a microprogrammed control unit:-

# The microprogram counter (μPC) is used to read the control words sequentially from the control store.

# Every time a new instruction is loaded into the IR, the output of the block labeled "starting address generator" is loaded into the μPC. The μPC is then automatically incremented by the clock, causing successive microinstructions to be read from the control store.

The μPC does not increment in the following situations:-

- (i) When an End microinstruction is encountered.
- (ii) When a new instruction is loaded into IR.
- (iii) When a branch microinstruction is encountered and the branch condition is satisfied.



Microinstructions :-

A straightforward way to structure microinstructions is to assign one bit position to each control signal required in the CPU.

Micro-instruction	PC <sub>in</sub>	PC <sub>out</sub>	MAR <sub>in</sub>	Read	MDR <sub>in</sub>	IR <sub>in</sub>	Y <sub>in</sub>	Clear	Count <sub>in</sub>	Add	Z <sub>in</sub>	Z <sub>out</sub>	R <sub>out</sub>	R <sub>in</sub>	R <sub>out</sub>	WMFC	End
1	0	1	1	1	0	0	0	1	1	1	1	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
4	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0
5	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0
6	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1

# This scheme has one serious drawback - assigning individual bits to each control signal results in long microinstructions.

# The above scheme is called a horizontal organization.

# The length of the micro-instructions can be reduced easily, because:-

- Most signals are not needed simultaneously.
- Many signals are mutually exclusive.

Vertical Microinstructions :-

Highly encoded schemes that use compact codes to specify only a small number of control functions in each microinstruction are referred to as a vertical organization.

F1 (4-bits)	F2 (3-bits)	F3 (3-bits)	F4 (4-bits) - -
0000: No transfer	000: No transfer	000: No transfer	0000: Add
0001: PC <sub>out</sub>	001: PC <sub>in</sub>	001: MAR <sub>in</sub>	0001: Sub
0010: MDR <sub>out</sub>	010: IR <sub>in</sub>	010: MDR <sub>in</sub>	
0011: Z <sub>out</sub>	011: Z <sub>in</sub>	011: TEMP <sub>in</sub>	
0100: R0 <sub>out</sub>	100: R0 <sub>in</sub>	100: Y <sub>in</sub>	
0101: R1 <sub>out</sub>	101: R1 <sub>in</sub>		
0110: R2 <sub>out</sub>	110: R2 <sub>in</sub>		
0111: R3 <sub>out</sub>	111: R3 <sub>in</sub>		
1010: Temp <sub>out</sub>			
1011: Address <sub>out</sub>			
			1111: XOR 16-ALL functions

# Grouping control signals into fields require a little more hardware, because decoding circuits must be used to decode the bit patterns of each field into individual control signals.



### Prefetching Microinstructions :-

→ The major drawback of microprogrammed control is that it is slow.

→ Fetching a microinstruction from the control store takes considerably longer than generating control signals using hardwired circuits.

→ The performance of the microprogrammed control can be improved by prefetching the next micro-instruction while the current one is being executed.

→ Prefetching microinstructions presents some organizational difficulties because sometimes the status flags and the results of the currently executed microinstructions are needed to determine the address of the next microinstruction.

### Instruction Pipeline :-

An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments.

This causes the instruction fetch and execute phases to overlap and perform simultaneous operations.

# If an instruction causes a branch out of sequence, then the pipeline must be emptied and all the instructions that have been read from memory after the branch instruction must be discarded.

### Instruction Cycle :-

Generally the computer needs to process each instruction with the following sequence of steps :-

- (1) Fetch the instruction from memory.
- (2) Decode the instruction.
- (3) Calculate the effective address.
- (4) Fetch the operands from memory.
- (5) Execute the instruction.
- (6) Store the results in the proper place.



### Four-Segment Instruction Pipeline:-

Suppose the instruction cycle is divided into four segments as given below:-

- (i) FI is the segment that fetches an instruction
- (ii) DA is the segment that decodes the instruction and calculates the effective address
- (iii) FO is the segment that fetches the operand.
- (iv) EX is the segment that executes the instruction.

# It is assumed that the processor has separate instruction and data memories so that the operation in FI and FO can proceed at the same time.

Step →	1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction: 1	FI	DA	FO	EX									
2		FI	DA	FO	EX								
(Branch) 3			FI	DA	FO	EX							
4				FI	-	-	FI	DA	FO	EX			
5					-	-	-	FI	DA	FO	EX		
6									FI	DA	FO	EX	
7										FI	DA	FO	EX

(Timing of instruction pipeline)

# Assume that instruction 3 is a branch instruction. As soon as this instruction is decoded in segment DA in step 4, the transfer from FI to DA of the other instructions is halted until the branch instruction is executed in step-6.

# If the branch is taken, a new instruction is fetched in step 7.

# If the branch is not taken, the instruction fetched previously in step-4 can be used.

### Pipeline Conflicts:-

In general, there are three major difficulties that cause the instruction pipeline to deviate from its normal operation:-

- (i) Resource Conflicts:- Caused by access to memory by two segments at the same time.
- (ii) Data dependency Conflicts:- arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
- (iii) Branch difficulties arise from branch and other instructions that change the value of PC.



Hazards:-

There are situations, called hazards, that prevent the next instruction in the instruction stream from being executing during its designated clock cycle. Hazards reduce the performance from the ideal speedup gained by the pipeline.

There are three classes of hazards:-

(i) Structural Hazards:-

They arise from resource conflicts when the hardware can not support all possible combinations of instructions in simultaneous overlapped execution.

(ii) Data Hazards:-

They arise when an instruction depends on the result of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline.

(iii) Control Hazards:-

They arise from the pipelining of branches and other instructions that change the PC.

Hazards in pipeline can make it necessary to stall the pipeline.

# It is possible to overcome the problem of structural hazards, but the designers allow structural hazards to reduce hardware cost.

Resolving Data Dependency Conflicts or Data Hazards:-(i) Hardware Interlocks:-

An interlock is a circuit that detects instructions whose source operands are destinations of instructions further up in pipeline. After detection of this situation, we delay the instruction by enough clock cycles to resolve the conflict.

(ii) Operand Forwarding:-

Avoid the conflict by routing the data through special paths between pipeline segments.

(iii) Delayed Load:-

Delay the loading of the conflicting data by inserting no-operation instructions (in compiler).

Handling of Branch Instructions or Control Hazard:-(i) Prefetch Target Instruction:-

Prefetch the target instructions in addition to the instruction following the branch. Both are saved until the branch is executed.

(ii) Branch Prediction:-

Some additional logic is used to guess the outcome of a conditional branch instruction before its executed. The pipeline then begins prefetching the instruction stream from the predicted path.

(iii) Delayed Branch:-

→ Used in most RISC processors.

→ In this procedure, the compiler detects the branch instructions and rearranges the machine language code sequence by inserting useful instructions that keep the pipeline operating without interruptions.



Synchronous Pipelines:-

These are clocked synchronously. The time between each clock signal is set to be greater than the longest Delay between pipeline stages.

Asynchronous Pipelines:-

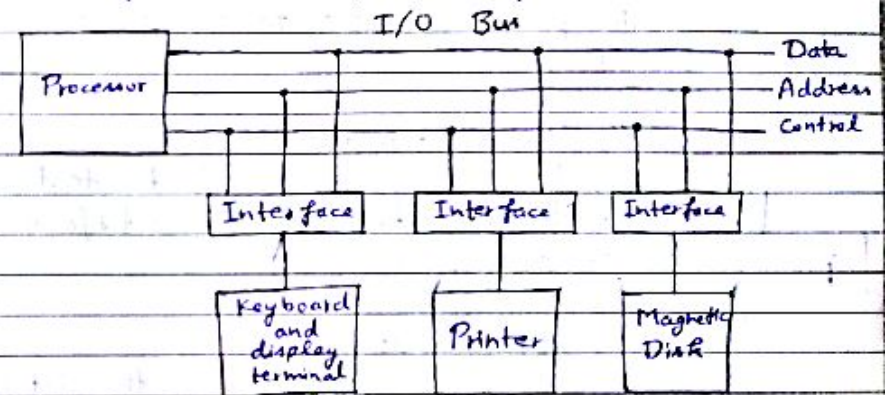
They use request/acknowledge system, wherein each stage can detect when it's finished.

www.ankurgupta.net

Input - Output OrganizationI/O Interface:-

I/O interfaces connects the peripherals to the CPU.

I/O interfaces supervise and synchronize all input and output transfers.



The I/O bus consists of :-

- (i) Data Lines
- (ii) Address Lines
- (iii) Control Lines

→ Each interface decodes the address and control received from the I/O bus, interprets them for the peripheral, and provides signals for the peripheral controllers.

→ Each peripheral has its own controller that operates the particular electromechanical device.

→ Address lines are used to select a peripheral.

→ Control lines are used to send commands to an interface.



There are four types of commands that an interface may receive:-

(i) Control Command:-

Used to activate the peripheral and to inform it what to do.

(ii) Status Command:-

Used to test various status conditions in the interface and the peripheral.

(iii) Output Data:-

It causes the interface to respond by transferring data from bus into one of its registers.

(iv) Input Data:-

The interface receives an item of data from the peripheral and places on the data lines.

I/O versus Memory Bus:-

The memory bus contains data, address, and read/write control lines.

There are three ways that computer buses can be used to communicate with memory and I/O:-

(i) Use two separate buses, one for memory and the other for I/O. (Using IOP - Input Output Processor).

(ii) Use one common bus for both memory and I/O, but have separate control lines for each. (Isolated I/O).

(iii) Use one common bus for memory and I/O with common control lines (memory-mapped).

# In isolated I/O, the I/O read and I/O write control lines are enabled during an I/O transfer. The memory read and memory write control lines are enabled during a memory transfer.

# In memory mapped I/O, a segment of the total address space is reserved for interface registers.



## Program Interrupt :-

Program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request.

Control returns to the original program after the service program is executed.

There are two ways that the processor chooses the branch address of the service routine after interrupt :-

### (i) Non- vectored interrupt :-

In a non-vectored interrupt, the branch address is assigned to a fixed location in memory.

### (ii) Vectored interrupt :-

In a vectored interrupt, the source that interrupts supplies the branch information to the computer. This information is called the interrupt vector.

## Interrupt - Request Line :-

The CPU has a wire called the interrupt request line that the CPU senses after executing every instruction.

## Types of Interrupts :-

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

- (i) External Interrupts
- (ii) Internal Interrupts
- (iii) Software Interrupts

### (i) External Interrupts :-

External interrupts come from external sources like I/O devices, timing device etc.

### (ii) Internal Interrupts :-

Internal interrupts arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called traps. Examples are :- Register overflow, attempt to divide by zero etc.

### (iii) Software Interrupts :-

Software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call.



Priority Interrupt :-

A priority interrupt is a system that establishes a priority over the various sources to determine which condition is to be served first when two or more requests arrive simultaneously.

The system may also determine which conditions are permitted to interrupt the computer while another interrupt is being serviced.

Establishing the priority of simultaneous interrupts :-

It can be done by software or hardware. There are three methods to establish priority :-

- (i) Polling
- (ii) Daisy-Chaining Priority
- (iii) Parallel-Priority Interrupt

(i) Polling :-

A polling procedure is used to identify the highest-priority source by software means.

(ii) Daisy-Chaining Priority :-

It is used to identify the highest-priority source by hardware means.

It consists of a serial connection of all devices that request an interrupt.

(iii) Parallel Priority Interrupt :-

This method uses a register whose bits are set separately by the interrupt signal from each device. Priority is established according to the position of the bits in the register.

Maskable Interrupt :-

A hardware interrupt that can be ignored by setting a bit in an interrupt mask register's bit mask.

Non-maskable Interrupt :-

It is a hardware interrupt that lacks an associated bit-mask, so that it can never be ignored. It is often used for timers.

Inter-Processor Interrupt :-

It is generated by one processor to interrupt another processor in a multiprocessor system.

Spurious Interrupt :-

It is a hardware interrupt that is unwanted.



## Modes of Data Transfer:-

Data transfer between the CPU and I/O devices may be handled in a variety of modes. Some modes use the 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 the three possible modes:-

- (i) Programmed I/O
- (ii) Interrupt-initiated I/O
- (iii) Direct memory access (DMA)

### (i) Programmed I/O:-

In programmed I/O, each data item transfer is initiated by an instruction in the program.

Once a data transfer is initiated, the CPU is required to monitor the interface to see when a transfer can again be made.

In the programmed I/O method, the CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer.

Usually the transfer is to and from a CPU register and peripheral. Other instructions are needed to transfer the data to and from CPU and memory.

Here the I/O device does not have direct access to memory.

### (ii) Interrupt-Initiated I/O:-

An alternative to the CPU constantly monitoring the flag is to let the interface inform the computer when it is ready to transfer data. This mode of transfer uses the interrupt facility. In the meantime the CPU can proceed to execute another program. The interface meanwhile keeps monitoring the device.

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

### (iii) Direct Memory Access (DMA):-

In DMA, the interface transfers data into and out of the memory unit through the memory bus.

The CPU initiates the transfer by supplying the interface with the starting address and the number of words needed to be transferred and then proceeds to execute other tasks.

DMA is a feature of modern computers. A simple DMA Controller is a standard component in PCs.



## Direct Memory Access :-

During DMA transfer, the CPU is idle and has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.

### Bus Request :-

The bus request (BR) input is used by the DMA controller to request the CPU to relinquish control of buses.

### Bus Grant :-

The CPU activates the bus grant (BG) output to inform the external DMA that the buses are in high-impedance state.

### Burst Transfer :-

A block sequence consisting of a number of memory words is transferred in a continuous burst while the DMA controller is the master of the memory buses.

### Cycle Stealing :-

It allows the DMA controller to transfer one data word at a time, after which it must return control of the buses to the CPU.

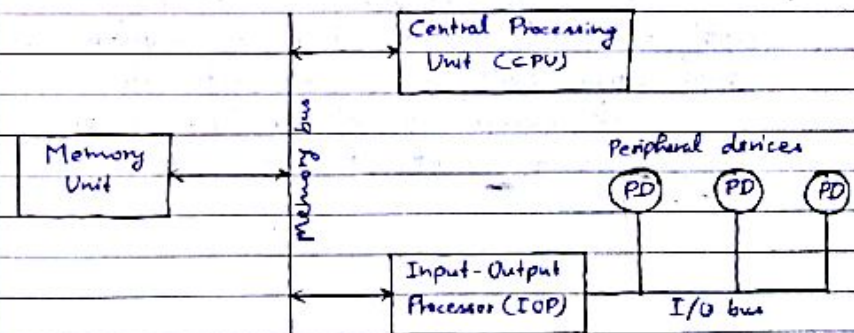
## Input-Output Processor (IOP) :-

An IOP is a processor with direct memory access capability that communicates with I/O devices.

The IOP is similar to a CPU except that it is designed to handle the details of I/O processing.

Unlike the DMA controller that must be set up entirely by the CPU, the IOP can fetch and execute its own instructions. In addition, the IOP can perform other processing tasks, such as arithmetic, logic, branching and code translation.

The memory unit occupies a central position and can communicate with each processor by means of direct memory access.



The CPU is usually assigned the task of initiating the I/O program. From then on the IOP operates independent of the CPU.



## Blocking and Nonblocking I/O:-

When an application issues a blocking system call, the execution of the application is suspended. The application is moved from the running queue to the waiting queue. After the system call completes, the application is moved back to the running queue.

A nonblocking call does not halt the execution of the application for an extended time. Instead, it returns quickly, with a return value that indicates how many bytes were transferred.

## Asynchronous I/O:-

An alternative to a nonblocking system call is an asynchronous system call.

An asynchronous call returns immediately, without waiting for the I/O to complete. The application continues to execute its code.

The completion of the I/O at some future time is communicated to the application through interrupt.

## Buffering:-

A buffer is a memory area that stores data while they are transferred between two devices or between a device and ~~and~~ an application.

## Spooling:-

A spool is a buffer that holds output for a device, such as a printer, that can not accept interleaved data streams.

## Expansion Bus & PCI Bus:-

An expansion bus connects the internal hardware of the computer system (including the CPU and RAM) and peripheral devices.

or  
Expansion Bus connects slow devices to processors.

PCI Bus connects fast devices to processors.

## Polling:-

Pooling is also referred as busy-waiting. In this situation, when an I/O operation is required, the computer does nothing other than check the status of the I/O device until it is ready, at which point device is accessed.

It is sometimes also referred as software driven I/O.



## Data Hazard Classification

# The hazards are named by the ordering in the program that must be preserved by the pipeline.

Consider two instructions  $i$  and  $j$ , with  $i$  occurring before  $j$ . The possible data hazards are:-

(i) RAW (Read After Write):-

$j$  tries to read a source before  $i$  writes it.

→ Operand Forwarding is used to overcome it.

(ii) WAW (Write After Write):-

$j$  tries to write an operand before it is written by  $i$ .

→ It is present only in pipelines that write in more than one pipe stages or allow an instruction to proceed even when a previous instruction is stalled.

(iii) WAR (Write After Read):-

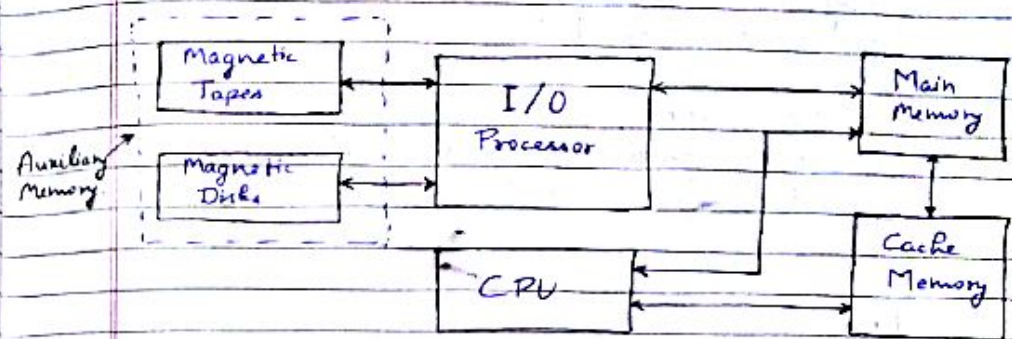
$j$  tries to write a destination before it's read by  $i$ .

→ It occurs when there are some instructions that write results early in the instruction pipeline and other instructions that read a source late in the pipeline.



Raster Displays:-

The scanning process sweeps the beam from left to right across the screen. At the end of the scanline the beam is moved to the start of the next line (horizontal retrace). After the last scanline, the beam is returned to start position (vertical retrace).

Memory ManagementMemory Hierarchy in a Computer System:-

# The main memory occupies a central position by being able to communicate directly with the CPU and with auxiliary memory devices through an I/O processor.

# The typical access time ratio between cache and main memory is about 1 to 7. Auxiliary memory average access time is usually 1000 times that of main memory.

Protection of Memory Space:-

We need to make sure that each process has a separate memory space. To do this, we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses. We can provide this protection by using two registers:

- (i) Base Register holds the smallest legal physical memory address.
- (ii) Limit Register specifies the size of the range.



The base and limit registers can be loaded only by the operating system, which uses a special privileged instruction.

The operating system, executing in kernel mode, is given unrestricted access to both operating system and user's memory.

### Static RAM:-

It consists of internal flip-flops that store the binary information. The stored information remains valid as long as power is applied to the unit. The Static RAM is easier to use and has shorter read and write cycles.

### Dynamic RAM:-

It stores the binary information in the form of electric charge that is applied to capacitors. The capacitors are provided inside the chip by MOS transistors.

The stored charge on the capacitors tend to discharge with time and the capacitors must be periodically recharged by refreshing the dynamic memory.

The Dynamic RAM offers reduced power consumption and large storage capacity in a single memory chip.

### ROM:-

ROM is used for storing programs that are permanently resident in the computer and for tables of constants that do not change in value once the production of the computer is completed. ROM is also Random Access.

### Bootstrap Loader:-

The bootstrap loader is a program whose function is to start the computer software operating when power is turned on.

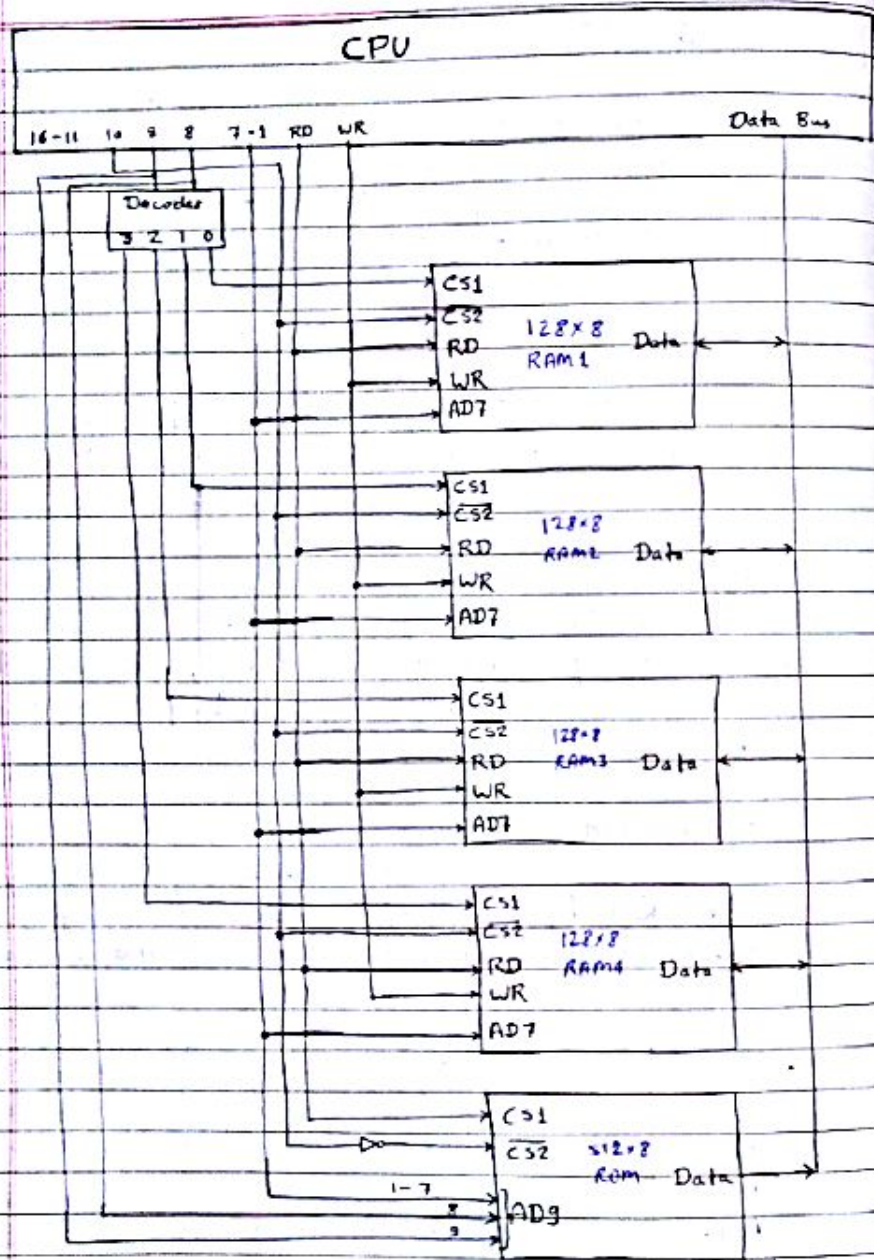
ROM portion of main memory is used for storing a bootstrap loader.

When power is turned on, the hardware of the computer sets the program counter (PC) to the first address of the bootstrap loader. The bootstrap program loads a portion of the operating system from disk to main memory and control is then transferred to the operating system, which prepares the computer for general use.



[illegible]





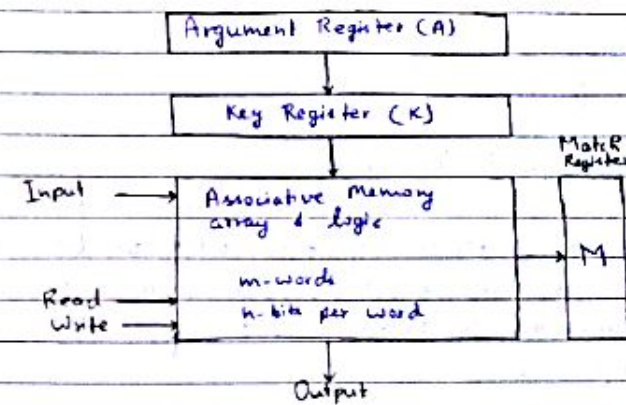
(Memory connections to the CPU)

## Associative 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.

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.



(Block diagram of associative memory)

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.



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.

Example:-

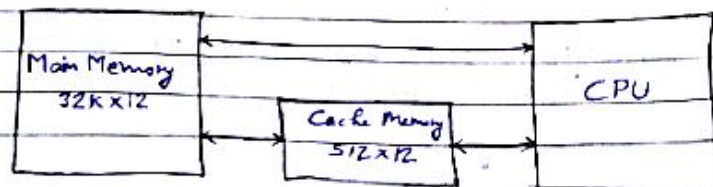
A	101 1111 00	
K	111 0000 00	
Word 1	100 1111 00	No match
Word 2	101 0000 01	Match.

## Cache Memory:-

### Locality of Reference:-

The references to memory at any given interval of time tend to be confined within a few localized area in memory. This phenomenon is known as the property of locality of reference.

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.



## 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 miss) is the hit ratio.

Example:- A computer with cache access time of 100ns, a main memory access time of 1000ns, and a hit ratio of 0.9 produces an average access time of 200 ns.

## Mapping:-

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:-

- (i) Associative Mapping
- (ii) Direct Mapping
- (iii) Set-associative Mapping



### (i) Associative Mapping:-

The associative memory stores both the address and content (data) of the memory word. This permits any location in cache to store any word from main memory.

CPU address (15-bits)

Argument Register

Address	Data
01000	3450
02777	6710
22345	1234

[Associative mapping cache (all numbers in octal)]

A CPU address of 15 bits is placed in the argument register and the associative memory is searched for a matching address. If the address is found, the corresponding 12-bit data is read and sent to the CPU. If no match occurs, the main memory is accessed for the word. The address-data pair is then transferred to the associative cache memory.

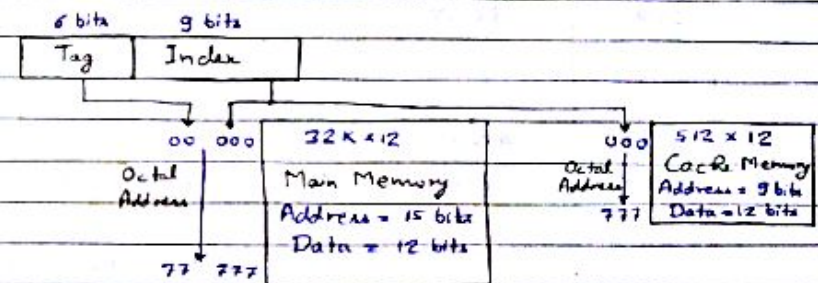
### (ii) Direct Mapping:-

In the general case, there are  $2^k$  words in cache memory and  $2^n$  words in main memory. The  $n$ -bit memory address is divided into two fields:-

(i)  $k$ -bits for the index field.

(ii)  $(n-k)$ -bits for the tag field.

The direct mapping cache organization uses the  $n$ -bit address to access the main memory and the  $k$ -bit index to access the cache.



Each word in cache consists of the data word and its associated tag. When a new word is first brought into the cache, the tag bits are stored alongside the data bits. When the CPU generates a memory request, the index field is used for the address to access the cache. The tag field of the CPU address is compared with the tag in the word read from the cache.

If the two tags match, there is a hit, otherwise there is a miss.



The disadvantage of direct mapping is that the hit ratio can drop considerably if two or more words whose addresses have the same index but different tags are accessed repeatedly.

Direct mapping cache with block size of 1 words:-

Memory Address	Memory Data	Index Address	Tag	Data
00000	1220	000	00	1220
00777	2340			
01000	3450			
01777	4560			
02000	5670	777	02	6710
02777	6710			

(b) Cache Memory

(b) Cache Memory

(a) Main Memory

Direct mapping cache with block size of 8 words:-

The index field is now divided into two parts :-

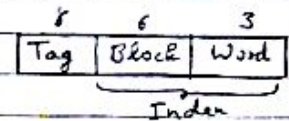
- (i) Block field, (ii) Word Field.

In a 512-word cache, there are 64 blocks of 8 words each. The block number is specified with a 6-bit field and the word within a block is specified with a 3-bit field.

The tag field stored within the cache is common to all eight words of the same block.

Every time a miss occurs, an entire block of eight words must be transferred from main memory to cache memory.

	Index	Tag	Data
	000	01	3456
Block 0	007	01	6578
	010		
Block 1	017		
Block 63	770	02	
	777	02	6710



(Direct mapping cache with block size of 8 words)



(iii) Set-Associative Mapping:-

The set-associative mapping is an improvement over the direct-mapping in that each word of cache can store two or more words of memory under the same index address.

Index	Tag	Data	Tag	Data
000	01	3450	02	5670
001				
777	02	6710	00	2340

(Two-way set-associative mapping cache)

Each data word is stored together with its tag and the number of tag-data items in one word of cache is said to form a set.

In the above diagram, each index address refers to two data words and their associative tags. Each tag requires six bits and each data word has 12 bits, so the word length is  $2(6+12) = 36$  bits. An index address of 9-bits can accommodate 512 words. Thus the size of cache memory is  $512 \times 36$  and it can accommodate 1024 words of main memory.

When the CPU generates a memory request, the index value of the address is used to access the cache. The tag field of the CPU address is then compared with both tags in the cache to determine if a match occurs. The comparison logic is done by an associative search of the tags.

Writing into Cache:-

There are two ways:-

- Write-through:- With every memory write operation, update main memory, with cache memory being updated in parallel if it contains the word at the specified address.
- Write-back:- In this method only the cache location is updated during a write operation. The location is then marked by a flag so that later when the word is removed from the cache, it is copied into main memory.

Cache Initialization:-

Each word in cache includes a valid bit to indicate whether or not the word contains valid data.

The cache is initialized by clearing all the valid bits to 0. The valid bit of a particular cache word is set to 1 the first time this word is loaded from main memory and stays set unless the cache has to be initialized again.

Cache Coherence:-

In a shared memory multiprocessor system, with multiple processors having separate caches, it is necessary to keep the caches in coherence by ensuring that any shared operand that is changed in any cache is changed throughout the entire system.



~~Cache Coherence~~Memory Binding :-

The binding of instructions and data to memory addresses can be done at any step along the way :-

(i) Compile time :-

If we know at compile-time where the process will reside in memory, then absolute code can be generated.

(ii) Load time :-

In this case, compiler generates relocatable code and final binding is delayed until load time.

(iii) Execution time :-

If the process can be moved during its execution from one memory segment to another, then binding must be delayed until runtime.

Dynamic Loading :-

With dynamic loading, a routine is not loaded until it is called. All routines are kept on disk in a relocatable load format.

✱✱ Dynamic loading does not require special support from the operating system. It is the responsibility of the users to design their programs to take advantage of such a method.

Dynamic Linking :-

Some operating systems support only static linking, in which system libraries are treated like any other object module and are combined by the loader into the binary program image.

The concept of dynamic linking is similar to that of dynamic loading. Here, though, linking, rather than loading, is postponed until execution time. This feature is usually used with system libraries.

With dynamic linking, a stub is included in the image for each library-routine reference. The stub is a small piece of code that indicates how to locate the appropriate memory-resident library routine or how to load the library if the routine is not already present.

Under this scheme, all processes that use a language library execute only one copy of the library code.

✱✱ Unlike dynamic loading, dynamic linking generally requires help from the operating system, because processes in memory are protected from one another.

Virtual Memory :-

Virtual memory is a concept that permits the user to construct programs as though a large memory space were available, equal to the totality of auxiliary memory.



## Logical Versus Physical Address Space:-

An address generated by the CPU is commonly referred to as a logical address or virtual address, and the set of such addresses is referred to as address space or logical address space or virtual address space.

An address in main memory is referred to as a physical address, and the set of such addresses is referred to as physical address space or memory space.

The compile-time and load-time address-binding methods generate identical logical and physical addresses.

In the execution-time address-binding scheme, the logical and physical address spaces differ.

The run-time mapping from virtual to physical addresses is done by a hardware device called the memory-management unit (MMU).

## Contiguous Memory Allocation:-

For a memory request of size  $n$ , there are three methods that are used to satisfy the request from a list of free holes:-

(i) First Fit:-

Allocate the first hole that is big enough.

(ii) Best Fit:-

Allocate the smallest hole that is big enough.

(iii) Worst Fit:-

Allocate the largest hole.

Both the first-fit and best-fit strategies for memory allocation suffer from external fragmentation.

External fragmentation exists when there is enough total memory space to satisfy a request, but the available spaces are not contiguous; storage is fragmented into a large number of small holes.

One solution to the problem of external fragmentation is compaction. Compaction is possible only if relocation is dynamic and is done at execution time.

Another possible solution to the external fragmentation problem is to permit the logical address space of the processes to be noncontiguous, thus allowing a process to be allocated physical memory wherever the latter is available.

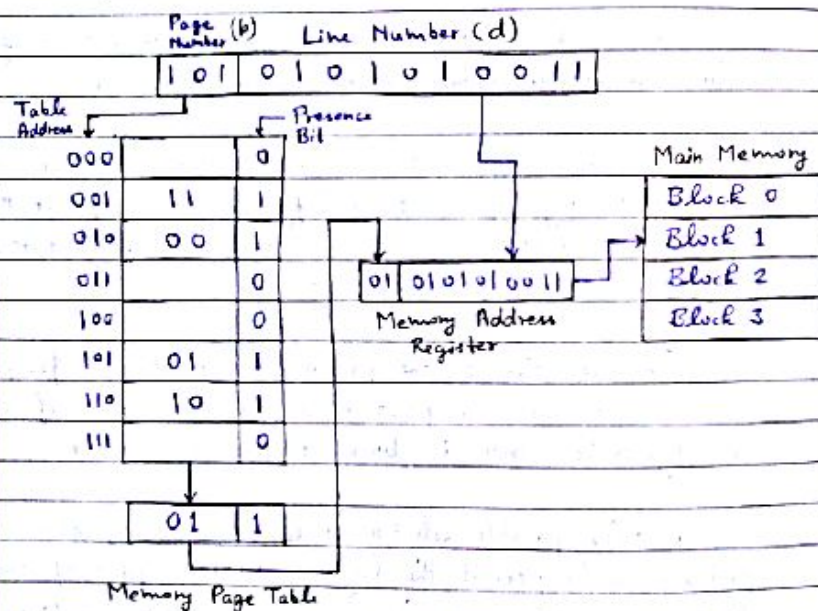


## Paging:-

Paging is a memory-management scheme that permits the physical address space of a process to be non-contiguous.

The basic method for implementing paging involves breaking physical memory into blocks of fixed size called frames or page-frames or blocks, and breaking logical memory into blocks of the same size called pages.

### Mapping from address space to memory space:-



A presence bit in each location indicates whether the page has been transferred from auxiliary memory into main memory. A 0 in the presence bit indicates that this page is not available in main memory.

When we use a paging scheme, we have no external fragmentation, however we may have some internal fragmentation.

If the memory allocated to a process is larger than the requested memory. The difference between these two is internal fragmentation - memory that is internal to a partition but is not being used.

### Translation Look-aside Buffer (TLB):-

In paging, two memory accesses are needed to access some data (one for page table entry and other for the data). Thus memory access is slowed by a factor of 2.

The solution to this problem is to use a special, small, fast-lookup hardware cache called a translation look-aside buffer (TLB). The TLB is associative, high speed memory.

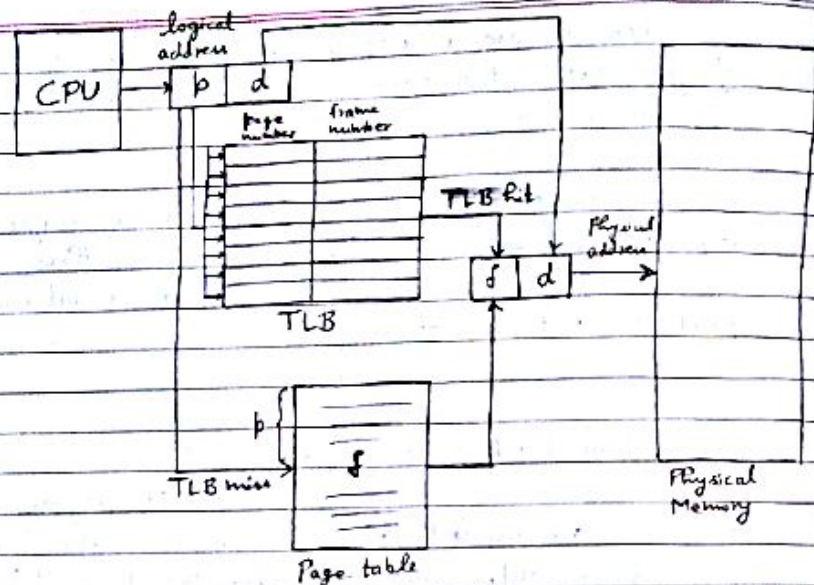
The TLB is used with page tables in the following way:-

The TLB contains only a few of the page-table entries. When a logical address is generated by the CPU, its page number is presented to the TLB.

If the page number is found, its frame no. is immediately available and is used to access memory.

If the page number is not in TLB (known as TLB miss), a memory reference to the page table must be made. In addition, we add the page number and frame number to the TLB, so that they will be found quickly on the next reference.





The percentage of times that a particular page number is found in the TLB is called the hit ratio.

Example:- Suppose a TLB has 80% hit ratio. If it takes 20 ns to search the TLB and 100 ns to access memory, then:-

$$\begin{aligned} \text{Effective Access Time} &= 0.80(20 + 100) \\ &\quad + 0.20(20 + 100 + 100) \\ &= 140 \text{ ns} \end{aligned}$$

Protection Bit:-

These bits are kept in page tables. One bit can define a page to be read-write or read-only.

Valid-invalid bit:-

This bit is also kept in page table. When this bit is set to valid, the associated page is in the process's logical address space and is thus a legal page.

Demand Paging:-

With demand-paged virtual memory, pages are only loaded when they are demanded during program execution; pages that are never accessed are thus never loaded into physical memory.

Page Replacement:-

# ~~The modify~~

Modify Bit or Dirty Bit:-

The modify bit for a page is set by the hardware whenever any word or byte in the page is written into, indicating that the page has been modified.

[Some remaining part of paging in different register]



[www.ankurgupta.net](http://www.ankurgupta.net)

## [Remaining Part of Page Table]

### Shared Pages:-

An advantage of paging is the possibility of sharing common code.

Consider a system that supports 40 users, each of whom executes a text editor. Here only one copy of the editor need to be kept in physical memory. Each user's page table maps onto the same physical copy of the editor, but data pages are mapped to different frames.

### Hierarchical Paging or Multilevel Page Tables:-

#### Problem with single level page table:-

Consider a system with a 32-bit logical address space. If the page size in such a system is 4KB ( $2^{12}$ ), then a page table may consist of  $2^{32}/2^{12} = 2^{20}$  entries. Assuming that each entry consists of 4 bytes, each process may need up to 4MB of physical address space for the page table alone.

So, the problem is that we can't hold all of the page tables in memory.

Solution:- The solution to this problem is that page the page tables. It allows portions of the page tables to be kept in memory at one time.

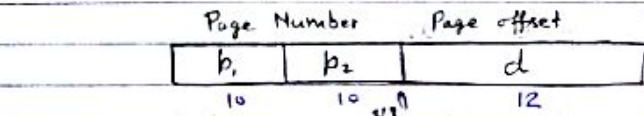


Consider the previous example of a 32-bit machine with a page size of 4KB. The logical address is divided into a page number consisting of 20-bits and a page offset consisting of 12-bits.

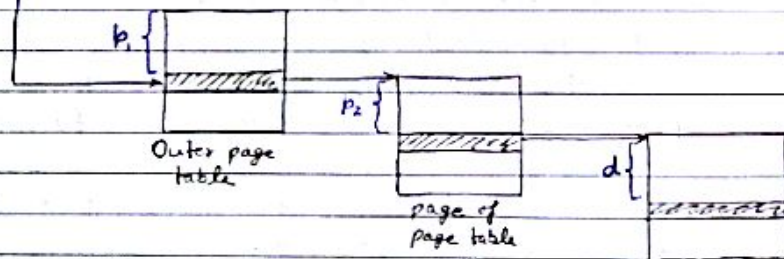
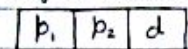
A page can have  $4KB/4B = 1K = 2^{10}$  entries of a page table.

Because we page the page table, the page number is further divided into a 10-bit page number and a 10-bit page offset.

Thus, a logical address is as follows:-



logical address



Because address translation works from the outer page table inward, this scheme is also known as a forward-mapped page table.

For 64-bit architectures, hierarchical page tables are generally considered inappropriate, because it will require 6-levels.

6 levels  $\rightarrow$  6 memory accesses  $\rightarrow$  slow.

### Hashed Page Tables:-

A common approach for handling address spaces larger than 32-bits is to use a hashed page table, with the hash value being the virtual page number. Each entry in the hash table contains a linked list of elements that hash to the same location.

Each element consists of three fields:-

- (i) Virtual page number
- (ii) Value of the mapped page frame.
- (iii) A pointer to the next element in the linked list.

### Inverted Page Tables:-

Usually, each process has an associated page table. The page table has one entry for each virtual address (each page).

The drawback of this method is that each page table may consist of millions of entries. These tables may consume large amounts of memory. To solve this problem, we use an inverted page table.

An inverted page table has one entry for each real page (or frame) of memory. There is only one page table in the system, and it has only one entry for each page of physical memory.

Each page table entry contains:-

- (i) Process ID
- (ii) Virtual Page Number

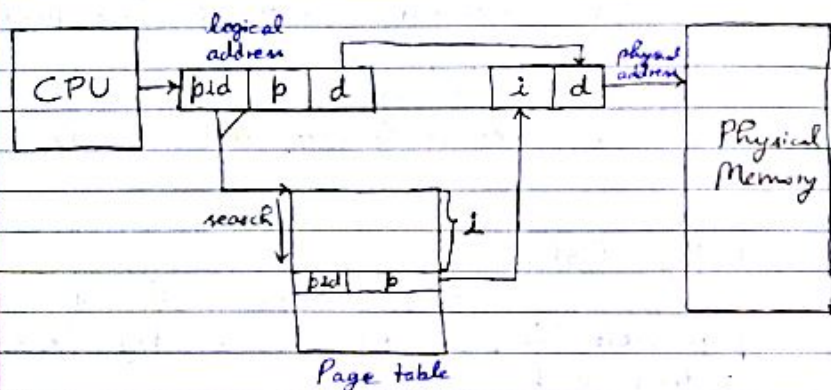
The table is indexed by physical page number.



### Accessing Inverted Page Tables:-

For each entry in the inverted page table, compare process ID and virtual page number in entry to the requested process ID and virtual page number.

Because of linear search its very slow.



### Hashed Inverted Page Tables:-

→ Linear inverted page tables require too many memory accesses.

→ Keep another level before actual inverted page table (hash anchor table):-

(i) Contains a mapping of process ID and virtual page number to page table entries.

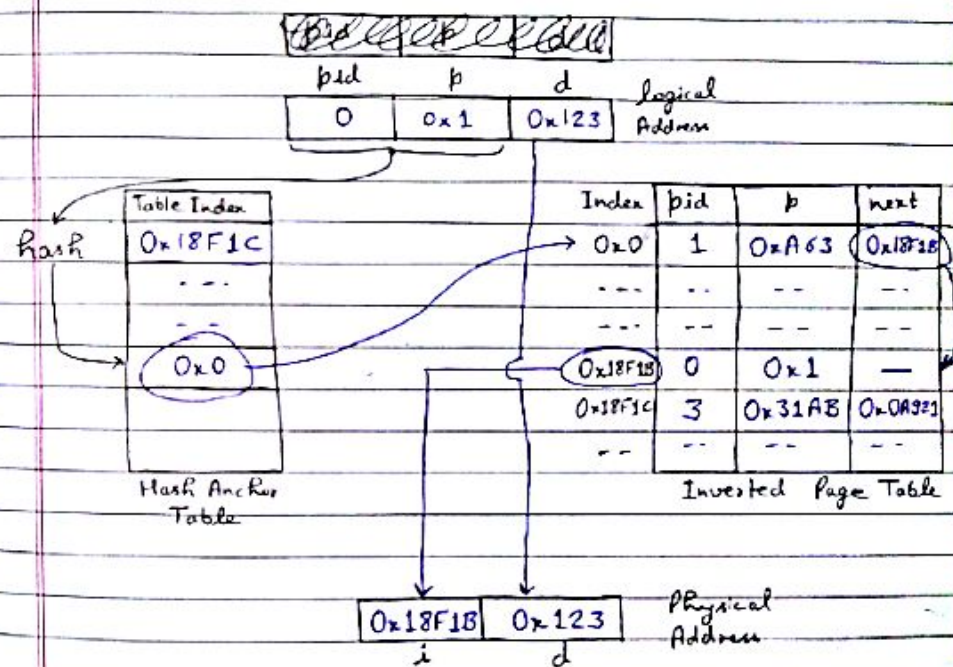
(ii) Use separate chaining for collisions.

### Accessing Hashed Inverted Page Table:-

(1) Lookup in hash anchor table for page table entry. Compare process ID and virtual page number.

(2) If match, then found.

(3) If not match, check the next pointer for another page table entry and check again.





## Page Replacement Algorithms:-

### (i) FIFO Page Replacement:-

A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen.

FIFO page replacement algorithm suffers from Belady's anomaly.

### Belady's Anomaly:-

For some page replacement algorithms, the page-fault rate may increase as the number of allocated frames increases.

### (ii) Optimal Page Replacement:-

Replace the page that will not be used for the longest period of time.

An optimal page replacement algorithm has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly.

### (iii) LRU Page Replacement:-

LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses the page that has not been used for the longest period of time.

This algorithm does not suffer from Belady's Anomaly.

## Thrashing:-

A process is thrashing if it is spending more time paging than executing.

## Local vs. Global Page Replacement:-

When a process incurs a page fault, a local page replacement algorithm selects for replacement some page that belongs to that same process.

A global replacement algorithm is free to select any page in memory.



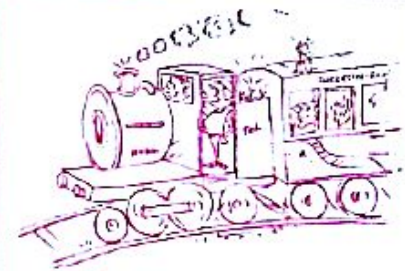
## Byte Addressing vs. Word Addressing:-

Byte Addressing (1 word = 4 bytes)	Word Addressing (1 word = 4 bytes)
<div> <div>0</div> <div>1</div> <div>2</div> <div>3</div> <div>4</div> <div>5</div> <div>6</div> <div>7</div> <div>⋮</div> <div>⋮</div> </div>	<div> <div>0</div> <div>1</div> <div>⋮</div> </div>
<div> <div>1 word</div> </div>	<div> <div>1 word</div> </div>
Address	Address

In the above addressing scheme, the first word starts at address 0, and the second word starts at address 4.

In the above addressing scheme, all bytes of first word are located in address 0, and all bytes of the second word are located in address 1.

## Laugh out Loud!



'El Pueblo de Nuestra Señora la Reina de los Angeles del Río de Porciúncula' is the full name of the American city Los Angeles.

In Turkestan, in 1919, a train was powered by almost 9000 tonnes of dried fish!



The cuddly animal koala bear, as it is often called, is not a bear at all; it is a marsupial! It sleeps for up to 18 hours daily!

Newton served as a member of the Parliament of England for one year. During all the lengthy proceedings, he spoke only once: he asked the person next to him to close an open window!

