COMPUTER ARCHITECTURE AND ORGANISATION

VI C.S.

COMPUTER DATA REPRESENTATION 1

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 ***Why we use Cil and Cir notations?***

Ans. We use the notation Cil and Cir for circular shift left micro-operation and circular shift right micro-operation.

Q.2 ***What is MIMD organization?***

Ans. It refers to a computer system capable of processing several programs at the same time.

Q.3 ***What is register mode?***

Ans. In this mode, the operands are in register that reside within the CPU. The particular register is selected from a register field in the introduction.

Q.4 ***Write the types of shift-micro operations?***

Ans.

- Logical Shift
- Circular Shift
- Arithmetic Shift

Q.5 ***What is the relationship between timing and control signal?***

Ans. Timing and Control Signal : All sequential circuits in the basic computer CPU are driven by a master clock, with the exception of the INPR register.

At each clock pulse, the control unit sends control signals to control inputs of the bus, the registers, and the ALU.

Control unit design and implementation can be done by two general methods:

- A *hardwired* control unit is designed from scratch using traditional digital logic design techniques to produce a minimal, optimized circuit. In other words, the control unit is like an ASIC (application-specific integrated circuit).
- A *microprogrammed* control unit is built from some sort of ROM. The desired control signals are simply stored in the ROM, and retrieved in sequence to drive the microoperations needed by a particular instruction.

PART-B

Q.6 ***Explain and draw a diagram of a bus system that use multiplex k, register of n bits each to produce an n-line common bus.*** *[R.T.U. 2016]*

Ans. Bus System : A bus structure consists of a set of common lines, one for each bit of register, through which binary information is transferred one at a time. The common bus system can be constructed with multiplexers. Multiplexers select the source register whose binary information is then placed into bus.

In general a bus system will multiplex k register of n bits each to produce an n-line common bus. The number of multiplexers needed to construct the bus is equal to number of bits each register. (n bit register uses n multiplexer). The size of each multiplexer must be $k \times 1$.

Example : A digital circuit has a common bus system for 16 register of 32 bits each. Then the number of selection input in each multiplexer is $5(32 = 2^5)$. Size of

Output Control Input During Interval T	Register Select Control Inputs During an Interval T			r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
OUT	RS_2	RS_1	RS_0								
1	0	0	0								Bus $\leftarrow r_0$
1	0	0	1							Bus $\leftarrow r_1$	
1	0	1	0						Bus $\leftarrow r_2$		
1	0	1	1					Bus $\leftarrow r_3$			
1	1	0	0				Bus $\leftarrow r_4$				
1	1	0	1			Bus $\leftarrow r_5$					
1	1	1	0		Bus $\leftarrow r_6$						
1	1	1	1	Bus $\leftarrow r_7$							

Fig. 1 : Bus transfer micro-operation from register to processor internal bus

Input Control Input During Interval T	Register Select Control Inputs During an Interval T			r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
OUT	RS_2	RS_1	RS_0								
1	0	0	0								Bus $\rightarrow r_0$
1	0	0	1							Bus $\rightarrow r_1$	
1	0	1	0						Bus $\rightarrow r_2$		
1	0	1	1					Bus $\rightarrow r_3$			
1	1	0	0				Bus $\rightarrow r_4$				
1	1	0	1			Bus $\rightarrow r_5$					
1	1	1	0		Bus $\rightarrow r_6$						
1	1	1	1	Bus $\rightarrow r_7$							

Fig. 2 : Bus transfer micro-operation from processor internal bus to register

When neither IN nor OUT is enables then the register is in disconnected state (tristate) with the bus.

Q.9 ***What do you understand by fixed point number representation?***

Ans. Fixed Point Number Representation : The shifting process above is the key to understand fixed point number representation. To represent a real number in computers (or any hardware in general), we can define a fixed point number type simply by implicitly *fixing* the binary point to be at some position of a numeral. We will then simply adhere to this implicit convention when we represent numbers.

To define a fixed point type conceptually, all we need are two parameters:

- width of the number representation, and
- binary point position within the number

We can use the notation fixed<w, b>, where w denotes the number of bits used as a whole (the Width of a number), and b denotes the position of binary point counting from the least significant bit (counting from 0).

For example, fixed<8,3> denotes a 8-bit fixed point number, of which 3 right most bits are fractional. Therefore, the bit pattern:

0	0	0	1	0	1	1	0

represents a real number:

$$00010.110_2$$
$$= 1 * 2^1 + 1 * 2^{-1} + 1 * 2^{-2}$$
$$= 2 + 0.5 + 0.25$$
$$= 2.75$$

Note that on a computer, a bit pattern can represents anything. Therefore the same bit pattern, if we "cast" it to another type, such as a fixed<8,5> type, will represents the number:

$$000.10110_2$$
$$= 1 * 2^{-1} + 1 * 2^{-3} + 1 * 2^{-4}$$
$$= 0.5 + 0.125 + 0.0625$$
$$= 0.6875$$

If we treat this bit pattern as integer, it represents the number:

$$10110_2$$
$$= 1 * 2^4 + 1 * 2^2 + 1 * 2^1$$
$$= 16 + 4 + 2$$
$$= 22$$

Q.10 ***What is the method to represent multiple instruction in register transfer?***

Ans. Register Transfer

- Register names use capital letters possibly followed by numbers, e.g. PC, R0, R1, ...
- Individual bits are numbered according to the power of two when they contain an unsigned binary integer. i.e. the rightmost bit is bit 0, next is bit 1, etc. Bits are numbered this way regardless of whether the register contains an unsigned binary integer, because it's an easy convention.
- Registers drawn as a box with the name inside. Bit numbers are written above.
- 16-bit registers may be divided into low and high bytes in both diagrams and in writing, e.g. PC(8-15) = PC(H), PC(0-7) = PC(L).

- Instruction cycle takes less time because it saves time in instruction fetching from memory.

Disadvantages

- When complex expressions are computed, program size increases due to the usage of many short instructions to execute it. Thus memory size increases.
- As the number of ipstructions increases for a program, the execution time increases.

Q.15 ***Explain the register transfer language.***

Ans. Register Transfer Language : The symbolic notation used to describe the microoperation transfers among registers is called a register transfer language.

- The term "register transfer" implies the availability of hardware logic circuits that can perform a stated microoperation and transfer the result of the operation to the same or another register.
- The word "language" is borrowed from programmers, who apply this term to programming languages.
- A register transfer language is a system for expressing in symbolic form the microoperation sequences among the registers of a digital module.
- It is a convenient tool for describing the internal organization of digital computers in concise and precise manner.
- It can also be used to facilitate the design process of digital systems.
- Information transfer from one register to another is designated in symbolic form by means of a replacement operator.
- The statement below denotes a transfer of the content of register R_1 into register R_2.

  $R_2 \leftarrow R_1$
- A statement that specifies a register transfer implies that circuits are available from the outputs of the destination register has a parallel load capability.
- Every statement written in a register transfer notation implies a hardware construction for implementing the transfer.

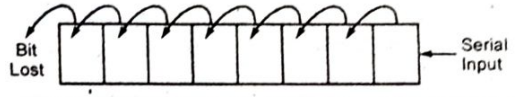
PART-C

Q.16 ***Explain the arithmetic micro-operation in register transfer language.*** ***[R.T.U. 2017]***

OR

Write 3 different types of shift micro-operations in Register Transfer Language. ***[R.T.U. 2014]***

Ans. Shift Micro-operations : Shift micro-operations are used to transfer the data serially. These are used in combination with arithmetic logic and other data processing operations. This operation shift the contents of register in left or right direction. At the same time when the bits are shifted the last or first flip-flop receives the binary information from the serial input. During shift left operation the serial input transfers a bit into the last flip-flop at the right most position.



The rest of the bits are shifted towards left by one bit position.

Similarly during a shift right operation the serial input transfer a bit into the first flip-flop at the left most position.

The rest of the bits are shifted towards right by one bit position.

There are three types of shift micro-operation. Each micro-operation can be done in left or in right direction.

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

These are shown in Table.

Table : Shifted Micro-operations

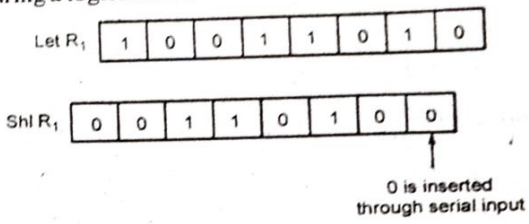
S.No.	Symbolic Designation	Description
1.	R ← Shl R	Shift left register R
2.	R ← Shr R	Shift right register R
3.	R ← Cil R	Circular Shift left register R
4.	R ← Cir R	Circular Shift right register R
5.	R ← ashl R	Arithmetic Shift left R
6.	R ← ashr R	Arithmetic Shift right R

(i) Logical Shift: A logical shift transfers 0 through the serial input. Shl denotes logical shift left operation while Shr denotes logical shift right operation.

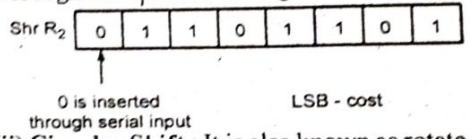
For example: $R_1 \leftarrow$ Shl R_1

$R_2 \leftarrow$ Shr R_2

The two micro-operations that specify a 1 bit shift to the left of the content of register R_1 and a 1 bit shift to the right of the content of register R_2. The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift.

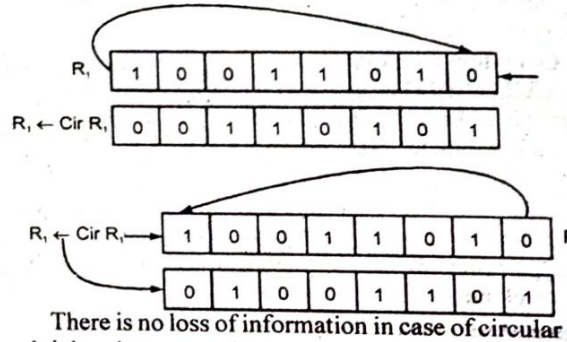


MSB of register R_1 is lost during this micro-operation.



(ii) Circular Shift : It is also known as rotate micro-operation. It circulates the bits of the register around both the ends without any loss of information. This is done by connecting the serial output of the shift register to the serial input. We use the notation Cil and Cir for circular shift left micro-operation and circular shift right micro-operation.

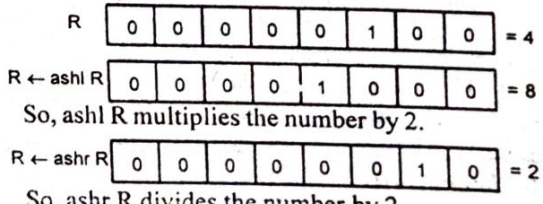
For example:



There is no loss of information in case of circular left and right micro-operation.

(iii) Arithmetic Shift : An arithmetic shift micro-operation shifts a signed binary number to the left or right direction. An arithmetic shift left multiplies a signed binary number by 2 and arithmetic shift right divides a signed binary number by 2.

For example :

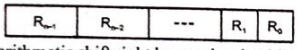


So, ashl R multiplies the number by 2.

So, ashr R divides the number by 2.

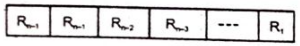
Arithmetic shifts must leave the sign bit unchanged because while dividing or multiplying the number by 2 sign of the number remains same.

The left most bit of the number holds the sign and the remaining bits represent the actual number. The sign bit is 0 for positive numbers and 1 for negative numbers in all representations.



The arithmetic shift right leaves the sign bit unchanged and shifts the number to the right. Thus R_{n-1} will remain same and R_{n-2} receives the bits from R_{n-1} and so on for the other bits in the register. The right most bit R_0 is lost.

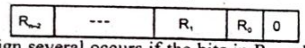
$R \leftarrow$ ashr R



$R_0 \leftarrow$ Lost

The arithmetic shift left inserts a 0 into R_0 and shifts all other bits to the left. The initial bit of R_{n-1} is lost and replaced by the bits from R_{n-2}

$R \leftarrow$ ashl R



A sign several occurs if the bits in R_{n-1} changes in value after shift. This happens if the multiplication by 2 causes an overflow.

Overflow : An overflow occurs after an arithmetic shift left if initially before the shift micro-operation R_{n-1} is not equals to R_{n-2}. An overflow flip-flop can be used to detect an arithmetic overflow.

$$V_{overflow} = R_{n-1} \oplus R_{n-2}$$

If $V_{overflow} = 0$ there is no overflow bit, if $V_{overflow}=1$, there is an overflow and a sign reversal after the shift. $V_{overflow}$ is transferred into the overflow flip-flop with the same clock pulse that shifts the register.

Q.17 ***(a) If a computer has 128 operation codes and 512 K addresses, how many bits would be required for***

(i) Single address instruction

(ii) Two address instruction

(b) What is instruction? What are different parts of an instruction? Explain the significance of each part of an instruction with an example.

(c) What do you mean by instruction set completeness? ***[R.T.U. Dec. 2013]***

Ans. (a) Bit required for opcodes

$= 7$ as $2^7 = 128$

For one address bits required

$= 19$ as $512\ K = 2^9 \times 2^{10} = 2^{19}$

An instruction consists of opcode + address thus

(i) Number of bits required for single address instruction

$= 7 + 19 = 26$ bits

(ii) Number of bits required for two address instruction

$= 7 + 19 + 19 = 45$ bits.

Ans. (b) Instruction : A computer instruction is a binary code that specifies a sequence of micro operations for the computer. Instruction codes together with data are stored in memory. The computer reads each instruction from memory and places it in a control register. The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of micro operations. Every computer has its own unique instruction set. The ability to store and execute instructions, the stored program concept, is the most important property of a general-purpose computer.

Different Parts of an Instruction

An instruction code is a group of bits that instruct the computer to perform a specific operation. The instruction has four parts : indirect bit, an operation code, a register code part and an address part. The most basic part of an instruction code is its operation part. The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement. The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer. The operation code must consist of at least n bits for a given 2^n (or less) distrinct operations. As an illustration, consider a computer with 64 distinct operations, one of them being an ADD operation. The operation code consists of six bits, with a bit configuration 110010 assigned to the ADD operation. When this operation code is decoded in the control unit, the computer issues control signals to read an operand from memory and add the operand to a processor register.

At this point we must recognize the relationship between a computer operation and a micro operation. An operation is part of an instruction stored in computer memory. It is a binary code that tells the computer to perform a specific operation. The control unit receives the instruction from memory and interprets the operation code bits. It then issues a sequence of control signals to initiate micro operations in internal computer registers. For every operation code, the control issues a sequence of micro operations needed for the hardware implementation of the specified operation. For this reason, an operation code is sometimes called a macro operation because it specifies a set of micro operations.

The operation part of an instruction code specifies the operation to be performed. This operation must be performed on some data stored in processor registers or in memory. An instruction code must therefore specify not only the operation but also the registers or the memory words where the operand are to be found, as well as the register or memory word where the result is to be stored. Memory words can be specified in instruction codes by their address. Processor registers can be specified by assigning to the instruction another binary code of k bits that specifies one of 2^k registers. There are many variations for arranging the binary code of instructions, and each computer has its own particular instruction code format.

For example, suppose an operand stored in address P in memory is to be added to one stored in Q and the result is to be stored in a address R, and the address of the next instruction to be executed is S, then an instruction would be of the type :

ADD	P	Q	R	S
What task? (Operation Code)	Address of first operand	Address of Second Operand	Address of result	Address where next instruction would be found

The instruciton given above is known as a four address instruciton. If a memory has 256K addresses then each address would be 18 bits long. The last address is not required as such. Ignoring the last address we have total 3 address instruction. Assuming 7 bits for opcode the length of instruction will be $7 + 18 + 18 + 18 = 61$ bits.

Ans. (c) Instruction : Before investigating the operations performed by the instructions, let us discuss the type of instructions that must be included in a computer. A computer should have a set of instructions so that the user can construct machine language programs to evaluate any function that is known to be computable. The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories:

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

Arithmetic, logical, and shift instructions provide computational capabilities for processing the type of data that the user may wish to employ. The bulk of the binary information in a digital computer is stored in memory, but all computations are done in processor registers. Therefore, the user must have the capability of moving information between these two units. Decisions making capabilities are an important aspect of digital computers. For example, two numbers can be compared, and if the first is greater than the second, it may be necessary to proceed d ;rently than if the second is greater than the first. Program control insturctions such as branch instructions are used to change the sequence in which the program is executed. Inputt and output instructions are needed for communication between the computer and the user. Programs and data must be transferred into memory and results of computations must be transferred back to the user.

Table : Basic computer instructions

Symbol	Hexadecim code I = 0	I = 1	Description
AND	0xxx	8xxx	AND memory word to AG
ADD	1xxx	9xxx	Add memory word to AG
LDA	2xxx	Axxx	Load memory word to AG
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AG
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instruction if AC positive
SNA	7008		Skip next instruction if AC negative
SZA	7004		Skip next instruction if AC zero
SZE	7002		Skip next instruction if E is 0
HLT	7001		Half computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F2		Skip on input flag
SKO	F1..		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

Interrupt handling

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

time 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.

Q.18 ***(a) List atleast five essential functional blocks that any computer should possess. Describe briefly the role of each block.***

(b) Define three state bus buffers. What do you understand by high impedance state in three state buffers? ***[R.T.U. 2013]***

OR

Draw and explain a graphical representation and block diagram for three state bus buffers.

[R.T.U. 2011, Raj. Univ. 2007, 2005]

Ans.(a) The five essential functions of a digital computer can be diagramed in block form (fig.1). The input devices are keyboards, punched card readers, punched tape readers, magnetic tape readers, print readers, and analog-to-digital converters. They read or accept data in original form and, if necessary, convert it to binary digital form (fig. 2). One of the problems concerning the design of the input equipment is the great difference in speed between some of the input devices and the much faster electronic circuits in the control, arithmetic, and memory sections.

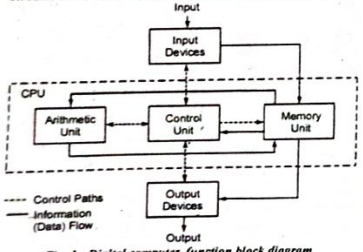


Fig. 1 : Digital computer, function block diagram

One solution is the use of high speed tape reading and tape writing equipment. For example, in a computer where punched cards are the primary input medium, it is uneconomical to tie up the entire machine during the relatively slow process of reading cards. It is better to read the cards in a separate machine and write the information on magnetic tape while the computer is otherwise employed. Then, the input information, now on magnetic tape, can be written into the computer much faster than from the cards.

Fig. 2 : Input devices block

The arithmetic unit (fig. 3) of the computer is limited in its mathematical ability. No problem can be solved unless it can be reduced to the simple arithmetic operations of addition, subtractions, multiplication, or division. The computer cannot be given entire equations and commanded to produce a solution. Instead, the equation must be reduced to its simplest elements. This matter of reducing a problem to its simplest elements in a sequence and form that will allow its solution by a particular computer is called computer programming. The arithmetic unit includes, as a minimum, the ability to count and add. Fortunately, the other arithmetic operations can be reduced to counting and adding.

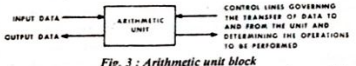


Fig. 3 : Arithmetic unit block

The memory unit (fig. 4) holds information until it is needed in the computing process. The results may be kept in memory until needed in further computation or removed as part of the solution of the problem. Examples of memory devices are magnetic cores, drums, disks, tapes, wires, cathode ray tubes, flip-flop circuits, and delay lines. The smallest element of information stored in a memory device is called a bit. A single bit can represent only the binary digit 0 or the binary digit 1, but bits can be combined to represent "words" of numerical quantities, algebraic signs, operation commands, or any other information. The state of a particular bit location in memory is either 0 or 1, depending on whether it is off or on, open or closed, at a low or high voltage level, in one direction or the other of magnetic saturation, or any other valid two-valued scheme of representation. In most computers, words are of the same length-10 to 50 bits per word being typical. However, modern computers can handle words of different length. The memory unit may be static or dynamic in operation. In a static unit, the binary bits of each word are assigned a set of locations with arbitrary numbering. In a dynamic unit, each word exists as a timed sequence of electrical or mechanical pulses that circulate about a closed loop. In such a loop, the memory location becomes essentially the time, with respect

to a given reference time, at which the first bit of the word passes a specified point in the circulation loop. If a number of loops are used, a location must be specified, both in positions and time. In the memory loops, acoustical delay lines may be used as memory elements. In both static and dynamic memory units, the location of each word in a unit is given by a number known as the address of that word. Memory addresses are frequently expressed as a two-part number, containing a channel number and a sector number, or their equivalents.

Fig. 4 : Memory unit block

The control unit (fig. 5), as its name implies, controls the routing and disposition of data, operational instructions, and the sequence of operations. In the same way that a great degree of variation occurs in the design of radar synchronizing or indicating systems, a similar wide variation exists in the design of the control unit. The control unit is composed of the same kind of logical devices and circuits that are used in the arithmetic unit. It is hard to tell by looking whether such a device or circuit is in the control unit or the arithmetic unit, because the basic operations of computing and control are so closely related.

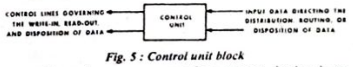


Fig. 5 : Control unit block

The primary purpose of an output device is to furnish or record solutions in a usable form. Examples of output devices (fig. 6) are electric typewriters, lighted numerals, paper tape, card punches, printers, and digital-to-analog converters.

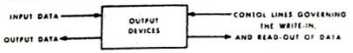


Fig. 6 : Output devices block

Ans.(b) Three-State Bus Buffers

A bus system can be constructed with three-state gates instead of multiplexers. A three-state gate is a digital circuit that exhibits three states. Two of the states are signals equivalent to logic 1 and 0 as in a conventional gate. The third state is a high-impedance state. The high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have a logic significance. Three-state gates may perform any conventional logic, such as AND or NAND. However, the one most commonly used in the design of a bus system is the buffer gate.

The graphic symbol of a three-state buffer gate is shown is fig. 1. It is distinguished from a normal buffer by having both a normal input and a control input. The control input determines the output state. When the control input is equal to 1, the output is enabled and the gate behaves like any conventional buffer, with the output equal to the normal input. When the control input is 0, the output is disabled and the gate goes to a high-impedance state, regardless of the value in the normal input. The high-impedance state of a three-state gate provides a special feature not available in other gates. Because of this feature, a large number of three-state gate outputs can be connected with wires to form a common bus line without endangering loading effects.

The construction of a bus system with three-state buffers is demon-strated in fig.2. The outputs of four buffers are connected together to form a single bus line. (It must be realized that this type of connection cannot be done with gates that do not have three-state outputs). The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line. No more than one buffer may be in the active state at any given time. The connected buffers must be controlled so that only one three-state buffer has access to the bus line while all other buffers are maintained in a high-impedance state.

One way to ensure that no more than one control input is active at any given time is to use a decoder, as shown in the diagram. When the enable input of the decoder is 0, all of its four output are 0, and the bus line is in a high-impedance state because all four buffers are disabled. When the enable input is active, one of the three-state buffers will be active, depending on the binary value in the select inputs of the decoder.

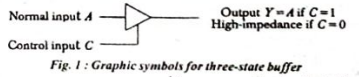


Fig. 1 : Graphic symbols for three-state buffer

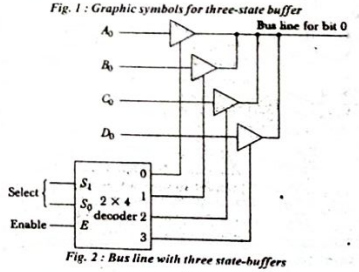


Fig. 2 : Bus line with three state-buffers

To construct a common bus for four registers of n bits each using three-state buffers, we need n circuits with four buffers in each as shown in fig. Each group of four buffers receives one significant bit from the four registers. Each common output produces one of the lines for the common bus for a total of n lines. Only one decoder is necessary to select between the four registers.

Q.19 ***Explain basic data representation method of computer? What is the difference between 1's and 2's complement?***

Ans. Data Representation : Computer memory stores only sequences of 0's and 1's. In order for data to be meaningful we must agree how to encode that data with 0's and 1's. Unfortunately, there are several systems that are used.

Unsigned Integers

- Simply use binary
- Using N bits, the integers $0..2^{(N)-1}$ can be represented.

  Examples : 17

Signed Magnitude

- Use the first bit for a sign (0 for +, 1 for -)
- Use the remaining N-1 bits for the magnitude (absolute value), in binary
- For N bits, $-(2^{(N-1)}-1)$ to $+(2^{(N-1)}-1)$ can be represented.
- Simple, but arithmetic circuits become awkward
- Two different zeroes are created (+0 is 000..00 and -0 is 100..00)
- Arithmetic is awkward eg. add 0101 with 1110. Since signs are different, determine the larger whose sign will be kept for answer, subtract larger from smaller. Not pretty

  Examples: +15, –6

Excess-N

- To represent X, use the binary code for X+M
- Excess-$2^{(N-1)}$ is normal, but any value of M can be used

The 4-bit codes for signed integers in these formats:

Value	Signed Magnitude	Excess-7	Excess-8
–9	----	----	----
–8	----	----	0000
–7	1111	0000	0001
–6	1110	0001	0010
–5	1101	0010	0011
–4	1100	0011	0100
–3	1011	0100	0101
–2	1010	0101	0110

+7	0111	1110	1111	----	0111	0111
+8	----	1111	----	----	----	----

Q.20 *How to represent floating point and IEEE floating point number?*

Ans. Floating-Point Representation : This representation does not reserve a specific number of bits for the integer part or the fractional part. Instead it reserves a certain number of bits for the number (called the mantissa or significand) and a certain number of bits to say where within that number the decimal place sits (called the exponent).

The floating number representation of a number has two part: the first part represents a signed fixed point number called mantissa. The second part of designates the position of the decimal (or binary) point and is called the exponent. The fixed point mantissa may be fraction or an integer. Floating point is always interpreted to represent a number in the following form: Mxr^e.

Only the mantissa m and the exponent e are physically represented in the register (including their sign). A floating-point binary number is represented in a similar manner except that is uses base 2 for the exponent. A floating-point number is said to be normalized if the most significant digit of the mantissa is 1.

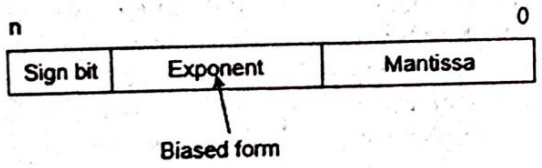


Fig.

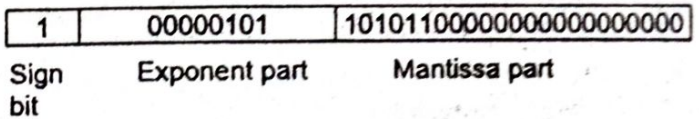
So, actual number is $(-1)^s(1+m)\times 2^{(e-Bias)}$, where *s* is the sign bit, *m* is the mantissa, *e* is the exponent value, and *Bias* is the bias number.

Note that signed integers and exponent are represented by either sign representation, or 1's complement representation, or 2's complement representation.

The floating point representation is more flexible. Any non-zero number can be represented in the normalized form of $\pm(1.b_1b_2b_3\ldots)_2\times 2^n$. This is normalized form of a number x.

Example : Suppose number is using 32-bit format: the 1 bit sign bit, 8 bits for signed exponent, and 23 bits for the fractional part. The leading bit 1 is not stored (as it is always 1 for a normalized number) and is referred to as a "*hidden bit*".

Then -53.5 is normalized as $-53.5=(-110101.1)_2=(-1.101011)\times 2^5$, which is represented as following :



Where 00000101 is the 8-bit binary value of exponent value +5.

Note that 8-bit exponent field is used to store integer exponents $-126 \le n \le 127$.

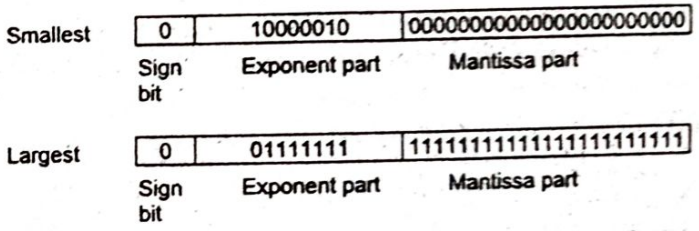
The smallest normalized positive number that fits into 32 bits is

$$(1.00000000000000000000000)_2\times 2^{-126} = 2^{-126} \approx 1.18\times 10^{-38},$$

and largest normalized positive number that fits into 32 bits is

$$(1.11111111111111111111111)_2\times 2^{127} = (2^{24}-1)\times 2^{104} \approx 3.40\times 10^{38}.$$

These numbers are represented as following :



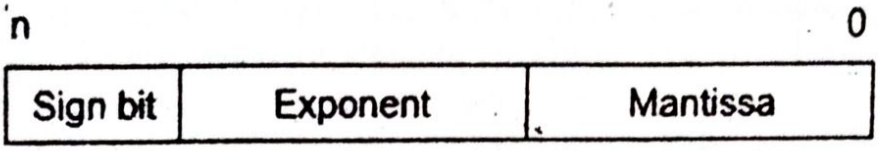
The precision of a floating-point format is the number of positions reserved for binary digits plus one (for the hidden bit). In the examples considered here the precision is 23 + 1 = 24.

The gap between 1 and the next normalized floating-point number is known as machine epsilon. the gap is $(1+2^{-23})-1=2^{-23}$, but this is same as the smallest

positive floating-point nu.nber because of non-uniform spacing unlike in the fixed-point scenario.

Note that non-terminating binary numbers can be represented in floating point representation, e.g., 1/3 = $(0.010101 \ldots)_2$ cannot be a floating-point number as its binary representation is non-terminating.

IEEE Floating point Number Representation : IEEE (Institute of Electrical and Electronics Engineers) has standardized Floating-Point Representation as :



So, actual number is $(-1)^s(1 + m) \times 2^{(e\text{-Bias})}$, where *s* is the sign bit, *m* is the mantissa, *e* is the exponent value, and *Bias* is the bias number. The sign bit is 0 for positive number and 1 for negative number. Exponents are represented by or two's complement representation.

According to IEEE 754 standard, the floating-point number is represented in following ways:

- **Half Precision (16 bit):** 1 sign bit, 5 bit exponent, and 10 bit mantissa
- **Single Precision (32 bit):** 1 sign bit, 8 bit exponent, and 23 bit mantissa
- **Double Precision (64 bit):** 1 sign bit, 11 bit exponent, and 52 bit mantissa
- **Quadruple Precision (128 bit):** 1 sign bit, 15 bit exponent, and 112 bit mantissa

Special Value Representation:

There are some special values depends upon different values of the exponent and mantissa in the IEEE 754 standard.

- All the exponent bits 0 with all mantissa bits 0 represents 0. If sign bit is 0, then +0, else -0.
- All the exponent bits 1 with all mantissa bits 0 represents infinity. If sign bit is 0, then $+\infty$, else $-\infty$.
- All the exponent bits 0 and mantissa bits non-zero represents de-normalized number.
- All the exponent bits 1 and mantissa bits non-zero represents error.

to the indirect phase to read the effective address at location 135, where it finds the previously saved address 21. When the BUN instruction is executed, the effective address 21 is transferred to PC . The next instruction cycle finds PC with the value 21, so control continues to execute the instruction at the return address.

Q.22 ***Explain the register transfer in detail with block diagram and timing diagram.***

Ans. Register Transfer : Information transfer from one register to another is designated in symbolic form by means of a replacement operator is known as register transfer.

$$R_2 \leftarrow R_1$$

Denotes a transfer of the content of register R_1 into register R_2.

Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register.

For example:

MAR	Holds address of memory unit
PC	Program Counter
IR	Instruction Register
R_1	Processor Register

Below fig. 1shows the representation of registers in block diagram form.

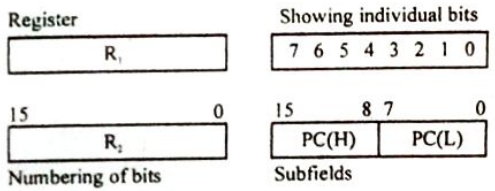


Fig. 1: Block diagram of register

The most common way to represent a register is by a rectangular box with the name of the register inside, as shown in fig.

Bits 0 through 7 are assigned the symbol L (for low byte) and bits 8 through 15 are assigned the symbol H (for high byte). The name of the 16-bit register is PC. The symbol PC (0-7) or PC (L) refers to the low-order byte and PC (8-15) or PC (H) to the high-order byte.

The statement that specifies a register transfer implies that circuits are available from the outputs of the source register to the inputs of the destination register and that the destination register has a parallel load capability.

Register Transfer with Control Function:

If we want the transfer to occur only under a predetermined control condition. This can be shown by means of an if-then statement.

If (P = 1) then ($R_2 \leftarrow R_1$)

Where, P is a control signal.

It is sometimes convenient to separate the control variables from the register transfer operation control function by specifying a control function.

A control function is a boolean variable that is equal to 1 or 0. The control function is included in the statement as follows:

$$P : R_2 \leftarrow R_1$$

The control condition is terminated with a colon. It symbolizes the requirement that the transfer operation be executed by the hardware only if P = 1.

Every statement written in a register transfer notation implies a hardware construction for implementing the transfer. Below fig. 2 shows the block diagram that depicts the transfer from R_1 to R_2.

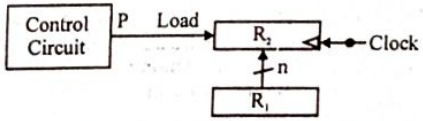


Fig. 2: Transfer from R_1 to R_2 when P = 1

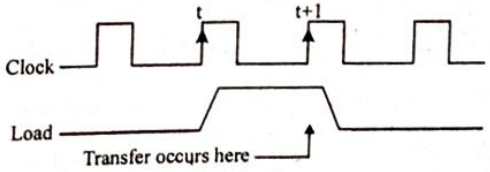


Fig. 3: Timing diagram

The n outputs of register R_1 are connected to the n inputs of register R_2. The letter n will be used to indicate any number of bits for the register. In the timing diagram, P is activated in the control section by the rising edge of a clock pulse at time t. The next positive transition of the clock at time t + 1 finds the load input active and the data inputs of R_2 are then loaded into the register in parallel. P may go back to 0 at time t + 1; otherwise, the transfer will occur with every clock pulse transition while P remains active. The basic symbols of the register transfer notation are listed in table below:

Table : Basic Symbols for Register Transfers

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	MAR, R_2
Parentheses ()	Denotes a part of a register	R_2(0-7), R_2(L)
Arrow	Denotes transfer of information	$R_2 \leftarrow R_1$
Comma,	Separates two micro operations	$R_2 \leftarrow R_1$, $R_1 \leftarrow R_2$

Registers are denoted by capital letters, and numerals may follow the letters. Parentheses are used to denote a part of a register by specifying the range of bits or by giving a symbol name to a portion of a register. The arrow denotes a transfer of information and the direction of transfer. A comma is used to separate two or more operations that are executed at the same time. The statement below, denotes an operation that exchanges the contents of two registers during one common clock pulse provided that T = 1.

$$T : R_2 \leftarrow R_1,\ R_1 \leftarrow R_2$$

This simultaneous operation is possible with registers that have edge-triggered flip-flops.

□□□

PROGRAMMING THE BASIC COMPUTER 2

IMPORTANT QUESTIONS

PART-A

Q.1 ***What do you mean by machine language?***

Ans. Machine Language : It referred to as machine or object code, machine language is a collection of binary digits or bits that the computer reads and interprets.

Q.2 ***What does mnemonic mean?***

Ans. Mnemonic : Mnemonic is a term, symbol or name used to define or specify a computing functions. Assembly languages also uses a mnemonic to represent machine operation, or opcode.

Q.3 ***Write the use of assembly language.***

Ans. Assembly Language : Assembly languages are used for real time systems and microprocessor based applications/devices.

Q.4 ***What is an assembler?***

Ans. Assembler : An assembler is a program that converts assembly language into machine code. It takes the basic commands and operations from assembly code and converts them into binary code that can be recognized by a specific type of processor.

Assemblers are similar to compilers in that they produce executable code. However, assemblers are more simplistic since they only convert low-level code (assembly language) to machine code. Since each assembly language is designed for a specific processor, assembling a program is performed using a simple one-to-one mapping from assembly code to machine code. Compilers, on the other hand, must convert generic high-level source code into machine code for a specific processor.

Most programs are written in high-level programming languages and are compiled directly to machine code using a compiler. However, in some cases, assembly code may be used to customize functions and ensure they perform in a specific way. Therefore, IDEs often include assemblers so they can build programs from both high and low-level languages.

PART-B

Q.5 ***What is loop? Explain types of loops.***

Ans. Loops : Loops are among the most basic and powerful programming concepts. A loop in a computer program is an instruction that repeats until a specified condition is reached. In a loop structure, the loop asks a question. If the answer requires action, it is executed. The same question is asked again and again until no further action is required. Each time the question is asked is called iteration.

Types of Loops

- A **for** loop is a loop that runs for a preset number of times.
- A **while** loop is a loop that is repeated as long as an expression is true. An expression is a statement that has a value.
- A **do while** loop or **repeat until** loop repeats until an expression becomes false.
- An **infinite** or **endless** loop is a loop that repeats indefinitely because it has no terminating condition, the exit condition is never met or the loop is instructed to start over from the beginning. Although

Risk of errors	The risk of errors existing in the syntax of machine language is high.	The risk of errors existing in assembly language is comparatively low.
Memorization	Binary codes cannot be memorized.	It is possible to memorize the commands given in assembly languages.
Compiler	No compiler is necessary for executing commands.	A compiler, also known as an assembler, is needed for the proper execution of assembly language commands.

Q.9 ***What is the difference between control memory and main memory?***

Ans. Control memory is memory inside the CPU or other control unit. We cannot see it on the motherboard, or even by looking at the CPU or control chip. This memory holds microinstructions. If the memory can be written to, it is called the Writeable Control Store. This memory hold the steps or inner-workings of the CPU itself. These days, RISC based CPUs do not use microinstructions, because by hardwiring the instructions they can get faster execution.

Main memory is outside the CPU. We can see it plugged into the motherboard. Main memory is accessed through the MMU and the cache of the CPU. Access to control memory is only internal to the chip itself.

Introduction of Control Unit and its Design

Control Unit is the part of the computer's central processing unit (CPU), which directs the operation of the processor. It was included as part of the Von Neumann Architecture by John Von Neumann. It is the responsibility of the Control Unit to tell the computer's memory, arithmetic/logic unit and input and output devices how to respond to the instructions that have been sent to the processor. It fetches internal instructions of the programs from the main memory to the processor instruction register, and based on this register contents, the control unit generates a control signal that supervises the execution of these instructions.

A control unit works by receiving input information to which it converts into control signals, which are then sent to the central processor. The computer's processor then tells the attached hardware what operations to perform. The functions that a control unit performs are dependent on the type of CPU because the architecture of CPU varies from manufacturer to manufacturer. Examples of devices that require a CU are:

- Control Processing Units(CPUs)
- Graphics Processing Units(GPUs)

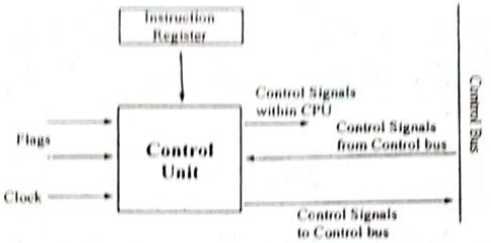


Fig. : Block diagram of the Control Unit

Functions of the Control Unit

1. It coordinates the sequence of data movements into, out of, and between a processor's many sub-units.
2. It interprets instructions.
3. It controls data flow inside the processor.
4. It receives external instructions or commands to which it converts to sequence of control signals.
5. It controls many execution units(i.e. ALU, data buffers and registers) contained within a CPU.
6. It also handles multiple .asks, such as fetching, decoding, execution handling and storing results.

Main Memory : Main memory refers to physical memory that is internal to the computer. The word *main* is used to distinguish it from external mass storage devices such as disk drives. Other terms used to mean main memory include *RAM* and primary storage.

The computer can manipulate only data that is in main memory. Therefore, every program user execute every file access must be copied from a storage device into main memory. The amount of main memory on a computer is crucial because it determines how many programs can be executed at one time and how much data can be readily available to a program.

Because computers often have too little main memory to hold all the data they need, computer engineers invented a technique called *swapping*, in which portions of data are copied into main memory as they are needed.

Swapping occurs when there is no room in memory for needed data. When one portion of data is copied into memory, an equal-sized portion is copied (swapped) out to make room.

Q.10 ***What do you understand by design of control unit? What is the difference between Hardwired Control and Micro programmed Control unit?***

Ans. Design of Control Unit : Control unit generates timing and control signals for the operations of the computer. The control unit communicates with ALU and main memory. It also controls the transmission between processor, memory and the various peripherals. It also instructs the ALU which operation has to be performed on data.

Control unit can be designed by two methodswhich are given below :

Hardwired Control Unit : It is implemented with the help of gates, flip flops, decoders etc. in the hardware. The inputs to control unit are the instruction register, flags, timing signals etc. This organization can be very complicated if we have to make the control unit large.

If the design has to be modified or changed, all the combinational circuits have to be modified which is a very difficul. task.

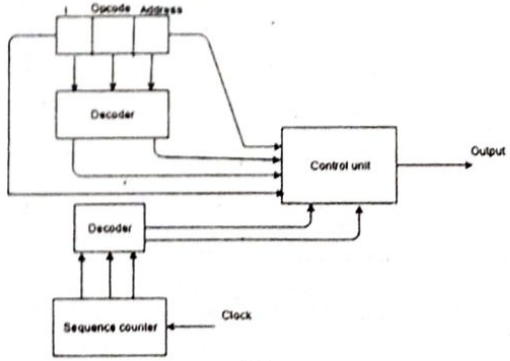


Fig.

Microprogrammed Control Unit : It is implemented by using programming approach. A sequence of micro operations is carried out by executing a program consisting of micro-instructions. In this organization any modifications or changes can be done by updating the micro program in the control memory by the programmer.

PROCESSING UNIT 3

ARS QUESTIONS

Ans. In computing, a vector processor or array processor is a central processing unit (CPU) that implements an instruction set containing instructions that operate on one-dimensional arrays of data called vectors.

Vector and array processing are essentially the same with small differences.

An array is made up of indexed collections of information called indices. Though an array can, in rare cases, have only one index collection, a vector is technically indicative of an array with at least two indices. Vectors are sometimes referred to as "blocks" of computer data.

Vector and array processing technology are most often seen in high-traffic servers.

PART-B

Q.6 ***Explain the Flynn's classification of computer.*** ***[R.T.U. 2017]***

OR

Explain Flynn's classification with suitable examples. ***[R.T.U. 2015]***

OR

Write down the Flynn's classification of computer. ***[R.T.U. 2016]***

OR

Explain Flynn classification of camputer architecture based on streams. ***[R.T.U. 2014]***

Ans. Flynn's Classification : M.J. Flynn considers the organization of a computer system by the number of instructions and data items that are manipulated simultaneously. The normal operation of a computer is to

fetch instructions from memory and execute them in the processor. The sequence of instructions read from memory constitutes an instruction stream. The operations performed on the data in the processor constitutes a data stream. Parallel processing may occur in the instruction stream, in the data stream, or in both. Flynn's classification divides computers into four major groups as follows :

(i) Single instruction stream, single data stream (SISD)

(ii) Single instruction stream, multiple data stream (SIMD)

(iii) Multiple instruction stream, single data stream (MISD)

(iv) Multiple instruction stream, multiple data stream (MIMD)

(i) **SISD** represents the organization of a single computer containing a control unit, a processor unit, and a memory unit. Instructions are executed sequentially and the system may or may not have internal parallel processing capabilities. Parallel processing in the case may be achieved by means of multiple functional units or by pipeline processing.

(ii) **SIMD** represents an organization that includes many processing units under the supervision of a common control unit. All processors receive the same instruction from the control unit but operate ou different items of data. The shared memory unit must contain multiple modules so that it can communicate with all the processors simultaneously.

(iii) **MISD** structure is only of theoretical interest since no practical system has been constructed using this organization.

(iv) **MIMD** organization refers to a computer system capable of processing several programs at the same time. Most multiprocessor and multicomputer systems can be classified in this category.

Flynn's classification depends on the distinction between the performance of the control unit and the data-processing unit. It emphasizes the behavioral characteristics of the computer system rather than its operational and structural interconnections.

Q.7 ***Draw and explain the organization of a CPU showing the connections between the register to a common bus.*** ***[R.T.U. 2017, 2016]***

Ans.

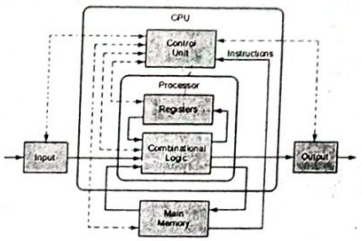


Fig. 1 : Block diagram of a single processor-cpu computer. Black Lines (⟶) indicates data flow, red lines (- - ⟶) indicates instruction flow and arrow indicates direction of flow.

From above diagram, it is clear that CPU consists of control unit, processor and bus carrying data and instructions.

1. Control Unit: The control unit of the CPU contains circuitry that uses electrical signals to direct the entire computer system to carry out stored program instructions. The control unit does not execute program instructions; rather, it directs other parts of the system to do so. The control unit communicates with both the ALU and memory.

2. Arithmetic Logic Unit: The arithmetic logic unit (ALU) is a digital circuit within the processor that performs integer arithmetic and bitwise logic operations. The inputs to the ALU are the data words to be operated on, status information from previous operations, and a code from the control unit indicating which operation to perform.

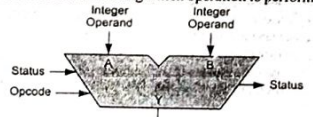


Fig. 2 : Symbolic representation of an ALU and its input and output signal

Depending on the instruction being executed, the operands may come from internal CPU registers or external memory, or they may be constants generated by the ALU itself. When all input signals have settled and

propagated through the ALU circuitry, the result of the performed operation appears at the ALU's outputs. The result consists of both a data word, which may be stored in a register or memory, and status information that is typically stored in a special, internal CPU register reserved for this purpose.

3. Memory Management Unit: Most high-end microprocessors have a memory management unit, translating logical addresses into physical RAM addresses, providing memory protection and paging abilities, useful for virtual memory. Simpler processors, especially microcontrollers, usually do not include an MMU.

Hardwired into a CPU's circuitry is a set of basic operations it can perform, called an instruction set. Such operations may involve, for example, adding or subtracting two numbers, comparing two numbers, or jumping to a different part of a program. Each basic operation is represented by a particular combination of bits, known as the machine language opcodes, while executing instructions in a machine language program, the CPU decides which operation to perform by "decoding" the opcodes. A complete machine language instruction consists of an opcodes and, in many cases, additional bits that specify arguments for the operation (for example, the numbers to be summed in the case of an addition operation). Going up the complexity scale, a machine language program is a collection of machine language instructions that the CPU executes.

The actual mathematical operation for each instruction is performed by a combinational logic circuit within the CPU's processor known as the arithmetic logic unit or ALU. In general, a CPU executes an instruction by fetching it from memory, using its ALU to perform an operation, and then storing the result to memory. Beside the instructions for integer mathematics and logic operations, various other machine instructions exist, such as those for loading data from memory and storing it back, branching operations, and mathematical operations on floating-point numbers performed by the CPU's floating-point unit.

Q.8 ***Explain speed up, efficiency and throughput in pipelining.*** ***[R.T.U. 2017, 2014]***

Ans. Speedup and Efficiency : Let us consider a space-time diagram in a pipeline processor. Fig.1 shows the space-time diagram of a four stage pipeline processor.

As shown in the fig.2 once the pipe is filled up, it outputs one result per clock period independent of the number of stages in the pipe. Ideally, a pipeline with k stages can process n tasks in $T_k = k + (n - 1)$ clock cycles, where k cycles are needed to complete the execution of the first task and the remaining n – 1 tasks require n – 1 cycles. Thus, the total time required is

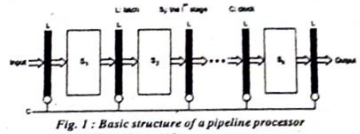


Fig. 1 : Basic structure of a pipeline processor

$$T_k = [k + (n - 1)]t$$

where t is the clock period.

The space-time diagram shown in fig.2 has four stages and five tasks. Therefore, the ideal total time required is

$T_k = [4 + (5 - 1)]t = 8$ clock cycles.

The amount of time required to perform same number of tasks in a non-pipeline processor can be given as

$$T_1 = nkt$$

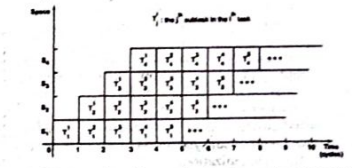


Fig. 2 : The space-time diagram showing the overlapped operations

Speedup Factor : The speedup factor of a k-stage pipeline over an equivalent non-pipelined processor is defined as

$$S_k = \frac{T_1}{T_k} = \frac{nkt}{[k+(n-1)]t}$$

$$= \frac{nk}{k+(n-1)}$$

For example, if pipeline has four stages and five tasks, its speedup factor is

$$S_4 = \frac{4\times5}{4+(5-1)} = \frac{20}{8}$$

$$= 2.5$$

It can be noticed that maximum speedup is $S_k \to k$ as $n \to \infty$. However, this maximum speed is very difficult

to achieve because of data dependencies between successive task (instructions), program branches, interrupts and other factors.

Efficiency: The efficiency of a pipeline is defined as a ratio of speedup factor and the number of stages in the pipeline. The efficiency of k-stage pipeline is given as

$$E_k = \frac{S_k}{k} = \frac{nk}{k+(n-1)} \Big/ k$$

$$= \frac{n}{k+(n-1)}$$

For example, if the speedup factor is 2.5 and the number of stages are four, the efficiency can be given as

$$E_k = \frac{2.5}{4} = 0.625$$

It can be noticed that the efficiency approaches to unity when $n \to \infty$. When $n = 1$, efficiency is minimum. It is 1/k.

Throughput : The pipeline throughput H_k is defined as the number of results (tasks) that can be completed by a pipeline per unit time. It is given as

$$H_k = \frac{n}{[k+(n-1)]t}$$

$$= \frac{E_k}{t} \qquad \because E_k = \frac{n}{[k+(n-1)]}$$

$$= E_k f \qquad \because f = 1/t$$

In ideal case, $H_k = 1/t = f$ when $E_k \to 1$. This means that the maximum throughput of a pipeline is equal to its frequency, which corresponds to one output result per clock period. The overall throughput of pipeline is always less than f. This is because usually $E_k < 1$.

Speedup is a process for increasing the performance between two systems processing the same problem. More technically, it is the improvement in speed of execution of a task executed on two similar architectures with different resources. The notion of speedup was established by Amdahl's law.

Efficiency is a metric of the utilization of the resources of the improved system. It is defined as:

Efficiency = Speedup / Number of Processors

Its value is typically between 0 and 1. Programs with linear speedup and programs running on a single processor have an efficiency of 1, while many difficult-to-parallelize programs have efficiency tending towards 0 as the number of processors increase.

Resources that are likely to be constrained ar hence form rate-limiting features of a given process a generically called **"bottlenecks"**. A bottleneck occu when the capacity of a processor is severely limited by single component. The bottleneck has lowest throughp of all parts of the transaction path.

Q.9 ***What does pipeline, vector and array processo mean in parallel processing?*** *[R.T.U. 201*

Ans. Parallel processing : In computers, paralle processing is the processing of program instructions b dividing them into multiple processors with the objectiv of running a program in less time. It is the simultaneou use of more than one CPU to execute a program. Multipl outputs are computed in parallel in a clock period th effective sampling speed is increased by the level o parallelism.

Pipeline processor: It is a set of data processing elements connected in series, where output of one elemen is the input for next element pipelining involves reusing hardware optimally based on dataflow. Pipelining is categorized in

1. **Linear pipelines:** A linear pipeline processor is a series of processing stages and memory access.
2. **Non-linear pipelines:** Also, called dynamic pipeline can be configured to perform various functions at different times. There is also feed-forward and feed-back connection. It also allows very long instruction word.

Vector Processor: In computing a vector processor is a central processing unit that implements an instruction set containing instruction that operate on 1-D arrays of data called vectors. There is a class of computational problems that are beyond the capabilities of the conventional computers. A computer capable of vector processing eliminates the overhead associated with the time it takes to fetch and execute the instructions in the program loop.

Array Processor: An array processor is a processor that performs the computations on large arrays of data.

There are two different types of array processor :

1. **Attached array processor:** Its purpose is to enhance the performance of the computer by providing vector processing. It achieves high performance my means of parallel processing with multiple functional units.
2. **SIMD array processor:** It is processor, which consists of multiple processing unit operating in

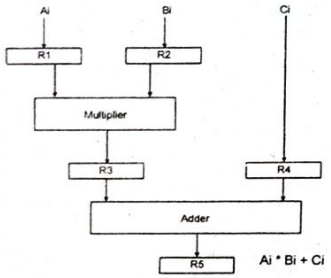
Another important aspect is that an arithmetic pipeline may be nonlinear. The "stages" in this type of pipeline are associated with key processing components such as adders, shifters, etc. Instead of a steady progression through a fixed sequence of stages, a task in a nonlinear pipeline may use more than one stage at a time, or may return to the same stage at several points in processing.

Example of an arithmetic pipeline

Suppose that we want to perform the combined multiply and add operations with a stream of numbers.

Ai * Bi + Ci for i = 1,2, 3, ..., 7

It can be implemented with the following arithmetic pipeline:



The pipeline would work as follows:

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A1	B1	-	-	-
2	A2	B2	A1*B1	C1	-
3	A3	B3	A2*B2	C2	A1*B1+C1
4	A4	B4	A3*B3	C3	A2*B2+C2
5	A5	B5	A4*B4	C4	A3*B3+C3
6	A6	B6	A5*B5	C5	A4*B4+C4
7	A7	B7	A6*B6	C6	A5*B5+C5
8	-	-	A7*B7	C7	A6*B6+C6
9	-	-	-	-	A7*B7+C7

Q.12 *Explain stack organization of Central Processing Unit.* *[R.T.U. 2015]*

Ans. A CPU stack is a register stack. Consider the organization of a 64-word register stack in a CPU as illustrated in the figure below:

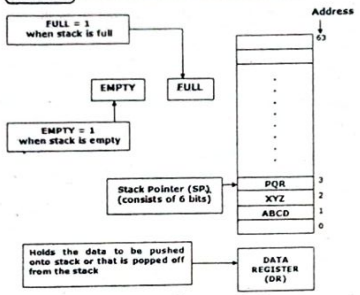


Fig.

The four separate registers used in the stack organization are :

1. *Stack Pointer Register (SP),* contains a value in binary each of 6 bits, which is the address of the top of the stack. Here, the stack pointer SP contains 6 bits because $2^6 = 64$, and SP cannot contain a value greater than 111111 i.e. value 63. When SP contains the address 63 and it is incremented by 1, then SP = 111111 + 1 = 1000000; the most significant 1 will be truncated as the carry and SP will be left with the value 0.
   Similarly when SP will contain the value 0 and it is decremented by 1, then the result will be 111111.
2. *FULL register,* which can store 1 bit information. It is set to 1 when the stack is full.
3. *EMPTY register,* which can store 1 bit information. It is set to 1 when stack is empty.
4. *Data Register (DR),* which holds the data to be written into or to be read from the stack.

Following are the micro-operations associated with the stack :

Initialization Conditions:-

SP β 0 // set stack pointer to 0
EMPTY β 1 // set EMPTY to 1 as initially stack is empty
FULL β 0 // clear FULL to 0 as stack contains no element

PUSH Operation:

SP β SP + 1 // Stack Pointer is incremented
M[SP] β DR // Data from DR register is written at top of stack
If (SP==0) then (FULL β 1) // Checking if Stack is FULL; Stack becomes 0 after 63
EMPTY β 0 // Mark that stack is not empty

Here, stack pointer is incremented so that it can point to the address of the next higher word. M[SP] refers to the memory location addressed by the value in SP. So the statement M[SP] β DR writes the data from the data register DR to the top of stack.

As initially the value of SP is 0, so the first word is written into stack after incrementing SP at location 1 and the last word will be written at address 0. Thus the stack will become full when SP contains the address 0, and then the register FULL will be set to 1.

When PUSH operation is performed, the stack doesn't remain empty and hence the register EMPTY is cleared to 0.

POP Operation :

DR β M[SP] // Read an item from top of stack
SP β SP-1 // Decrement stack pointer
If (SP==0) then (EMPTY β 1) // Check if stack is empty
FULL β 0 // Mark the stack as not full

For the POP operation, the data stored at the top of the stack is read into the data register DR and the stack pointer is then decremented to point to the lower address word. The data which is read is not actually deleted from top of the stack, it remains there, but after the next PUSH operation, it is overwritten. If the value of stack pointer SP becomes 0, then it means that the stack is empty and hence the register EMPTY is set to 1. Also after a POP operation the value of FULL register is cleared to 0 as the stack is not full and can accommodate more data items.

Q.13 *What are the different conflicts that will arise in pipeline? How do you remove the conflict? Describe.* *[R.T.U. Dec. 2013]*

Ans. Pipeline Conflicts : In general, there are three major difficulties that cause the instruction pipeline to derive from its normal operation. These pipeline conflicts are :

1. Resource conflicts
2. Data dependency conflicts
3. Branch difficulties

1. Resource Conflicts : Resource conflicts caused by access to memory by two segments at the same time.

2. Data Dependency Conflicts : Data dependency conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.

3. Branch Difficulties : Branch difficulties arise from branch and other instructions that change the value of PC.

Removing Pipeline Conflicts

1. Resource Conflicts : Most of these conflicts can be resolved by using separate instruction and data memories.

2. Data Dependency Conflicts : Pipelined computers deal with such conflicts in a variety of ways :

(i) Hardware Interlocks : The most straight forward method is to insert *hardware interlocks*. An interlock is a circuit that detects instructions whose source operands are destinations of instructions farther up in the pipeline. Detection of this situation causes the instruction whose source is not available to be delayed by enough clock cycles to resolve the conflict. This approach maintains the program sequence by using hardware to insert the required delays.

(ii) Operand Forwarding : Another technique called *operand forwarding* uses special hardware to detect a conflict and then avoid it by routing the data through special paths between pipeline segments. For example, instead of transferring an ALU result into a destination register, the hardware checks the destination operand, and if it is needed as a source in the next instruction, it passes the result directly into the ALU input, by passing the register file. This method requires additional hardware paths through multiplexers as well as the circuit that detects the conflict.

(iii) Delayed Load : A procedure employed in some computers is to give the responsibility for solving data conflicts problems to the compiler that translates the high-level programming language into a machine language program. The compiler for such computers is designed to detect a data conflict and reorder the instructions as necessary to delay the loading of the conflicting data by inserting no-operation instructions. This method is referred to as *delayed load*.

3. Branch Conflicts : Pipelined computers employ various hardware technique to minimize the performance degradation caused by instruction branching.

(i) Prefetch Target Instruction : One way of handling a conditional branch is to prefetch the target instruction in addition to the instruction following the branch. Both are saved until the branch is executed. If the branch condition is successful, the pipeline continues from the branch target instruction. An extension of this procedure is to continue fetching instructions from both places until the branch decision is made. At that time, control chooses the instruction stream of the correct program flow.

(ii) Branch Target Buffer : Another possibility is the use of a *branch target buffer* or BTB. The BTB is an associative memory included in the fetch segment of the pipeline. Each entry in the BTB consists of the address of a previously executed branch instruction and the target instruction for that branch. It also stores the next few instructions after the branch target instruction. When the pipeline decodes a branch instruction, it searches the associative memory BTB for the address of the instruction. If it is in the BTB, the instruction is available directly and prefetch continues from the new path. If the instruction is not in the BTB, the pipeline shifts to a new instruction stream and stores the target instruction in the BTB. The advantage of this scheme is that branch instructions that have occurred previously are readily available in the pipeline without interruption.

(iii) Loop Buffer : A variation of the BTB is the *loop buffer*. This is a small very high speed register file maintained by the instruction fetch segment of the pipeline. When a program loop is detected in the program, it is stored in the loop buffer in its entirety, including all branches. The program loop can be executed directly without having to access memory until the loop mode is removed by the final branching out.

(iv) Branch Prediction : Another procedure that some computers use is branch prediction pipeline with branch prediction and uses some additional logic to guess the arrival of a conditional branch instruction before it is executed. The pipeline begins prefetching the instruction stream from the predicted path. A correct prediction eliminates the wasted time caused by branch penalties.

(v) Delayed Branch : A procedure employed in most RISC processors is the *delayed branch*. 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. An example of delayed branch is the insertion of a no-operation instruction after a branch instruction. This causes the computer to fetch the target instruction during the execution of the no-operation instruction, allowing a continuous flow of the pipeline.

Q.14 *What do you mean by parallel processing? Write the Flynn's classification of parallel processing.* *[R.T.U. 2013]*

Ans. Parallel Processing is a term used to denote a large class of techniques that are used to provide

Shift Register: It operates in serial fashion 1 bit at a time while register with parallel load operates with all the bits of the words simultaneously.

Parallel processing at a higher level of complexity can be achieved by having a multiplicity of functional unit that performs identical or different operations simultaneously.

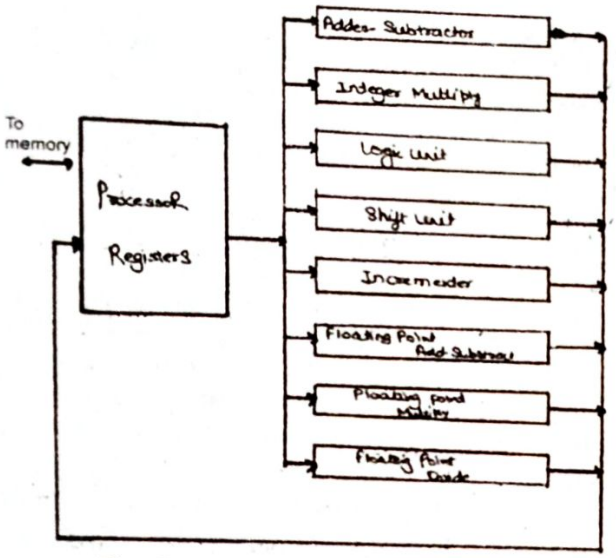


Fig. : Process with Multiple Functional Unit

Parallel Processing Unit is established by distinguish the data among the multiple functional units.

For example

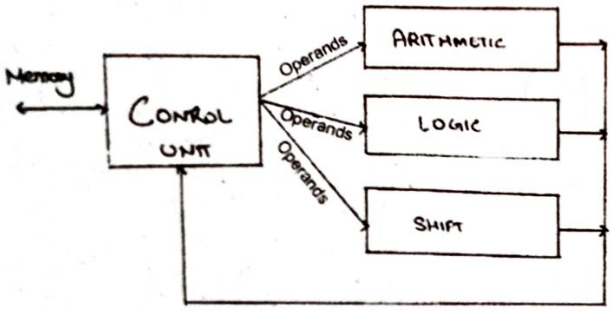


Fig. : Block diagram of Process units

PART-C

Q.17 ***What is addressing mode? Explain the direct and indirect register addressing modes with suitable examples.*** *[R.T.U. 2017]*

OR

Explain direct and indirect register addressing modes with suitable examples. *[R.T.U. 2015]*

OR

What is addressing mode? Explain different addressing modes with suitable examples. *[R.T.U. Dec. 2013]*

Ans. Addressing Modes : The term addressing modes refers to the way in which the operand of an instruction is specified. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually executed.

The operation field of an instruction specifies the operation to be performed. This operation will be executed on some data which is stored in computer registers or the main memory. The way any operand is selected during the program execution is dependent on the addressing mode of the instruction. The purpose of using addressing modes is as follows:

(i) To give the programming versatility to the user.

(ii) To reduce the number of bits in addressing field of instruction.

Types of Addressing Modes : There are different types of addressing modes:

1. Implied Mode : In this mode the operands are specified implicitly in the definition of the instruction. In fact, all register reference instructions that use an accumulator are implied-mode instructions. Zero-address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.

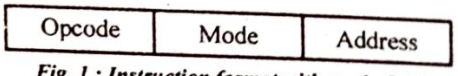


Fig. 1 : Instruction format with mode field

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.

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 introduction. 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 operand itself. Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction. A reference of 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 from 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 in direct 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 spread 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 compution 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.

A few addressing modes require that the address field of the instruction can be added to the content of a specific register in the CPU. The effective address in these modes is obtained from the following computation:

Effective address = Address part of instruction + Content of CPU register.

The CPU register used in the computation may be the program counter, an index register or a base register. In either case we have a different addressing mode which is used for a different application.

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.

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 special CPU register that contains an index value. The address field of the instruction defines the beginning address 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. If an index-type instruction does not include an address field in its format, the instruction converts to the register indirect mode of operation.

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 related to the address part of the instruction. A base register is assumed to hold a base address and the address field 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 instructions must reflect this change of position. With a base register, the displacement values of instructions do not have to change. Only the value of the base register

requires updating to reflect the beginning of a new memory segment.

Numerical Example

To show the differences between the various modes, we will show the effect of the addressing modes on the instruction defined in Fig. 2. The two-word instruction at address 200 and 201 is a "load to AC" instruction with an address field equal to 500. The first word of the instruction specifies the operation code and mode and the second word specifies the address part. PC has the value 200 for fetching this instruction. The content of processor register R1 is 400 and the content of an index register XR is 100. AC receives the operand after the instruction is executed. The figure lists a few pertinent addresses and shows the memory content at each of these addresses.

The mode field of the instruction can specify any one of a number of modes. For each possible mode we calculate the effective address and the operand that must be loaded into AC. In the direct address mode, the effective address is the address part of the instruction 500 and the operand to be loaded into AC is 800.

In the immediate mode the second word of the instruction is taken as the operand rather than an address, so 500 is loaded into AC. (The effective address in this case is 201.) In the indirect mode the effective address is stored in memory at address 500. Therefore, the effective address is 800 and the operand is 300. In the relative mode the effective address is 500 + 202 = 702 and the operand is 325. (Note that the value in PC after the fetch phase and during the execute phase is 202.)

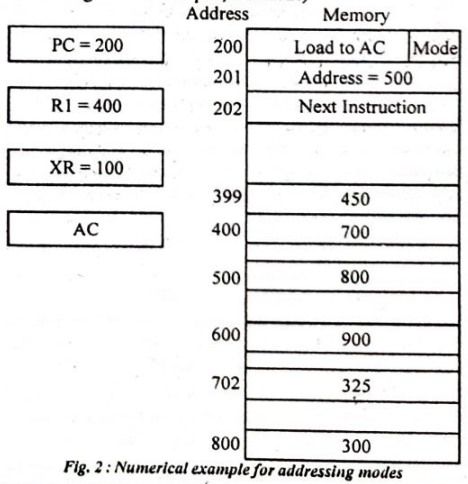


Fig. 2 : Numerical example for addressing modes

In the index mode the effective address is XR + 500 = 100 + 500 = 600 and the operand is 900. In the register mode the operand is in R1 and 400 is loaded into AC. (There is no effective address in this case.) In the register indirect mode the effective address is 400, equal to the content of R1 and the operand loaded into AC is 700. The autoincrement mode is the same as the register indirect mode except that R1 is incremented to 401 after the execution of the instruction. The autodecrement mode decrements R1 to 399 prior to the execution of the instruction. The operand loaded into AC is now 450. Table shows the values of the effective address and the operand loaded into AC for the nine addressing modes.

Table : Tabular list of numerical example

Addressing Mode	Effective Address	Content of AC
Direct address	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	325
Indexed address	600	900
Register	–	400
Register indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

E.g. of direct addressing mode:

LDA addr. This is an ALP statement ADDR in operand field is a symbolic name given to 16-bit address. The systematic name given to 16-bit address symbolic name can be chosen by the user either reference to context LDA is the mnemonic for LOAD accumulator direct. The instruction format for this is as shown.

Opcode	N
$<B_2>$	N + 1
$<B_3>$	N + 2

Where the memory location 'N' contains the opcode namely 00 110 010 = $(3A)_4$ followed by a lower order 8-bits of address $<B_2>$ and higher order 8-bit of address $<B_3>$ at memory location NH&N+2 respectively. The meaning of instruction is load the accumulator from the memory location whose address is available directly, in the instruction itself. The macro RTL implemented is,

$$(A) \leftarrow M(B_3, B_2)$$

This is symbolic representation, Only one operand is involved in the instruction e execution.

Example of indirect addressing mode:

Example: ADD (A), R0

(address A is embedded in the instruction code and (A) is the operand address = pointer variable)

Q.18 ***What is the advantage of pipelining? Explain instruction pipeline in detail.*** *[R.T.U. 2015]*

OR

Why do we require instruction pipelining ? Explain its working procedure. Discuss the pipeline performance measures.

[R.T.U. 2017, Dec. 2013]

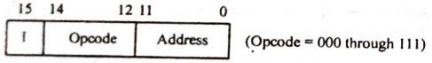
Ans. Pipelining : Pipelining is a technique of decomposing a sequential process into suboperations with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments. A pipeline can be visualized as a collection of processing segments through which binary information flows.

Each segment performs partial processing dictated by the way the task is in partitioned. The result obtained from the computation in each segment is transferred to the next segment in the pipeline. The final result is obtained after the data have passed through all segments. The name "pipeline" implies a flow of information analogous to an industrial assembly line. It is characteristic of pipelines that several computations can be progress in distinct segments at the same time.

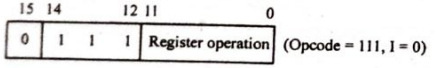
Instruction Format : The basic computer has three instruction code formats. Each format has 16 bits. The operation code (opcode) is a part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered.

Types of Instruction

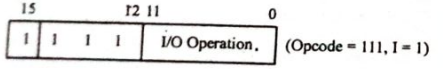
1. Memory-reference instruction
2. Register-reference instruction
3. Input/output instruction



(a) Memory-reference instruction



(b) Register-reference instruction



(c) Input-output instruction

Fig. 1 : Instruction format

A memory-reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I.I is equal to 0 for direct address and to 1 for indirect address.

The register-reference instructions are recognized by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction. A register reference instruction specifies an operation on or a test of the AC register. An operand fron. memory is not needed therefore, the other 12 bits are used to specify the operation or test to be executed. Similarly, an input-output instruction does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type of input-output operation or test performed.

The type of instruction is recognized by the computer control from the four bits in the positions 12 through 15 of the instruction. If the three opcode bits in the positions 12 through 14 are not equal to 111, the instruction in a memory-reference type and the bit in position is 15 taken as the addressing mode I.

If the 3-bit opcode is equal to 111, control then inspects the bit in position 15 If the bit is 0, the instruction is a register-reference type. If the bit is 1, the instruction is an input-output type. Note that the bit in position 15 of the instruction code is designated by the symbol I but is not used as a mode bit when the operation code is equal to 111.

Only three bits of the instruction are used for the operation code. It may seem that computer is restricted to a maximum of eight distinct operations. However, since the register-reference and input-output instructions use the remaining 12 bits as a part of the operation code, the total number of instructions can exceed eight. In fact, the total number of instructions choosen for basic computer are 25.

Instructions Formats

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 Rl, A, B  R1←M[A] + M[B]
ADD R2, C, D R5←M[C] + M[D]
MUL X, Rl, R2 M[X]←R1*R2
```

It is assumed that the computer has two processor registers, Rl 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. 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←M[A]
ADD	Rl, B	R1←R1 + M[B]
MOV	R2, C	R2←M[C]
ADD	R2, D	R2←P2+M[D]
MUL	Rl,R2	R1←R1*R2
MOV	X,R1	M[X]←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 register and assume that the AC contains the result of all operations. The program to evaluate X = (A + B) * (C + D) is

LOAD	A	AC←M[A]
ADD	B	AC←A[C] + M[B]
STORE	T	M[T]←AC
LOAD	C	AC←M[C]
ADD	D	RC←AC + M[D]
MUL	T	AC←AC*M[T]
STORE	X	M[X]←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 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←A
PUSH	B	TOS←B
ADD		TOS←(A + B)
PUSH	C	TOS←C
PUSH	D	TOS←D
ADD		TOS←(C + D)
MUL		TOS←(C + D)*(A + B)
POP	X	M[X]←TOS

RISC Instruction : The instruction set of a typical RISC processor is restricted to the use of load and store instructions communicate between memory and CPU. All other are executed within the registers of the CPU without referring to memory. A program for RISC type CPU consists of LOAD and STORE instructions. That have one memory and address registers with all three specifying processor registor.

LOAD	R1, A	R1 ← M[A]
LOAD	R2, B	R2 ← M[B]
LOAD	R3, C	R3 ← M[C]
LOAD	R4, D	R4 ← M[D]
ADD	R1, R2, R3	R1 ← R2 + R3
ADD	R3, R3, R4	R3 ← R3 + R4
MUL	R1, R1, R3	R1 ← R1 + R3
STORE	X, R1	M[X] ← R1

Instruction Pipeline : Pipeline processing can occur not only in the data stream but in the instruction stream as well. 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. One possible digression associated with such a scheme is that an instruction may cause a branch out of sequence. In that case the pipeline must be emptied and all the instructions that have been read from memory after the branch instruction must be discarded.

Consider a computer with an instruction fetch unit and an instruction execution unit designed to provide a two-segment pipeline. The instruction fetch segment can be implemented by means of a first-in, first-out (FIFO) buffer. This is a type of unit that forms a queue rather than a stack. Whenever the execution unit is not using memory, the control increments the program counter and uses its address value to read consecutive instructions from memory. The instructions are inserted into the FIFO buffer so that they can be executed on a first-in, first-out basis. Thus an instruction stream can be placed in a queue, waiting for decoding and processing by the execution

segment. The instruction stream queuing mechanism provides an efficient way for reducing the average access time to memory for reading instructions. Whenever there is space in the FIFO buffer, the control unit initiates the next instruction fetch phase. The buffer acts as a queue from which control then extracts the instructions for the execution unit.

Computers with complex instructions require other phases in addition to the fetch and execute to process an instruction completely. In the most general case, 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 result in the proper place

There are certain difficulties that will prevent the instruction pipeline from operating at its maximum rate. Different segments may take different times to operate the incoming information. Some segments are skipped for certain operations. For example, a register transfer mode instruction does not need an effective address calculation. Two or more segments may require memory access at the same time, causing one segment to wait until another is finished with the memory. Memory access conflicts are sometimes resolved by using two memory buses for accessing instruction and data in separate modules. In this way, an instruction word and a data word can be read simultaneously from two different modules.

The design of an instruction pipeline will be most efficient if the instruction cycle is divided into segments of equal duration. The time that each step takes to fulfill its function depends on the instruction and the way it is executed.

Four-Segment Instruction Pipeline : Assume that the decoding of the instruction can be combined with the calculation of the effective address into one segment.

Assume further that most of the instructions place the result into a processor register so that the instruction execution and storing of the result can be combined into one segment. This reduces the instruction pipeline into four segments.

Fig. 2 shows how the instruction cycle in the CPU can be processed with a four-segment pipeline. While an instruction is being executed in segment 4, the next instruction in sequence is busy in fetching an operand from memory in segment 3. The effective address may be calculated in a separate arithmetic circuit for the third instruction and whenever the memory is available, the fourth and all subsequent instructions can be fetched and placed in an instruction FIFO. Thus, up to four suboperations in the instruction cycle can overlap and up to four different instructions can be in progress of being processed at the same time.

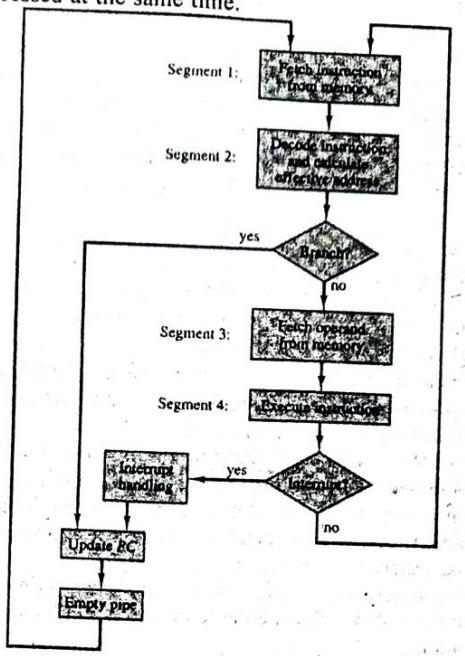


Fig. 2 : Four-segment CPU pipeline

Once in a while, an instruction in the sequence may be a program control type that causes a branch out of normal sequence. In that case, the pending operations in the last two segments are completed and all information stored in the instruction buffer is deleted. The pipeline then restarts from the new address stored in the program counter. Similarly, an interrupt request, when acknowledged, will cause the pipeline to empty and start again from a new address value.

Table : Timing of instruction pipeline

Step :		1	2	3	4	5	6	1	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

Table shows the operation of the instruction pipeline. The time in the horizontal axis is divided into steps of equal duration. The four segments are represented in the diagram with an abbreviated symbol.

1. FI is the segment that fetches an instruction.
2. DA is the segment that decodes the instruction and calculates the effective address.
3. FO is the segment that fetches the operand.
4. 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. In the absence of a branch instruction, each segment operates on different instructions. Thus, in step 4, instruction 1 is being executed in segment EX; the operand for instruction 2 is being fetched in segment FO; instruction 3 is being decoded in segment DA and instruction 4 is being fetched from memory in segment FI.

Assume now 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. The pipeline then continues until a new branch instruction is encountered.

Another delay may occur in the pipeline if the EX segment needs to store the result of the operation in the data memory while the FO segment needs to fetch an operand. In that case, segment FO must wait until segment EX has finished its operation.

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

1. **Resource conflicts** caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.
2. **Data dependency conflicts** arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
3. **Branch difficulties arise** from branch and other instructions that change the value of PC.

Advantages of Pipelining

- The cycle time of the processor is reduced, thus increasing instruction issue-rate in most cases.
- Some combinational circuits such as address or multipliers can be made faster by adding more circuitry. If pipelining is used instead, it can save circuitry.

Q.19 ***Explain the differences between RISC and CISC computers.*** ***[R.T.U. 2017]***

OR

Is there any difference between RISC and CISC computers? Explain. ***[R.T.U. 2015]***

OR

Give the difference between RISC and CISC processor. Describe in detail. ***[R.T.U. Dec. 2013]***

As sign of A = + ve set Q[0] = 1	00000011001	00000010011	10000000101	1
Step-10				
LS(AQ)	00000011001	00000100111	000000101–	
A = A – M	00000011001	00000001110	000000101–	
As sign of A = +ve set Q[0] = 1	00000011001	00000001110	0000001011	0

From the above result, we see that quotient = Q = 1011 = 11 and remainder = A = 1110 = 14.

Q.8 ***Draw and explain flow chart for addition and subtraction of floating points numbers.***

[R.T.U. 2017, 2016]

Ans. Addition and Subtraction :

During addition or subtraction the two floating-point operands are in AC and BR. The sum or difference is formed in the AC. The algorithm can be divided into four consecutive parts :

1. Check for zeros
2. Align the mantissas
3. Add or subtract the mantissas
4. Normalize the result

A floating point number that is zero cannot be normalized. We check for zeros at the beginning and terminate the process if necessary. The alignment of the mantissas must be carried out prior to their operation. After the mantissas are added the flowchart for adding and subtracting two floating point binary numbers is shown in figure. If BR is equal to zero, the operation is terminated, with the value in the AC being the result. If AC is equal to zero, we transfer the content of BR in AC and also complement its sign if the number are to be subtracted. If neither number is equal to zero we proceed to align the mantissas.

The magnitude comparator attached to exponent a and b provides three outputs that indicate their relative magnitude. If the two exponent are equal then the operation will be performed. if the exponents are not equal,

the mantissa having the smaller exponent is shifted to the right and its exponent incremented. The addition and subtraction of the two mantissas is identical to the fixed point addition and subtraction but if overflow occurs when magnitudes are added, it is transferred into flip-flop E. If E = 1 the bit is transferred into A_1 and all other bits of A are shifted right. The exponent must be incremented to maintain the correct number. If the magnitudes were subtracted, the result may be zero or may have an underflow. If the mantissa is zero, the entire floating point number in the AC is made zero. The mantissa has an underflow if the most significant bit in position A_1 is 0. In this case, the mantissa is shifted left and the exponent decremented. The bit in A_1 is checked again and the process.

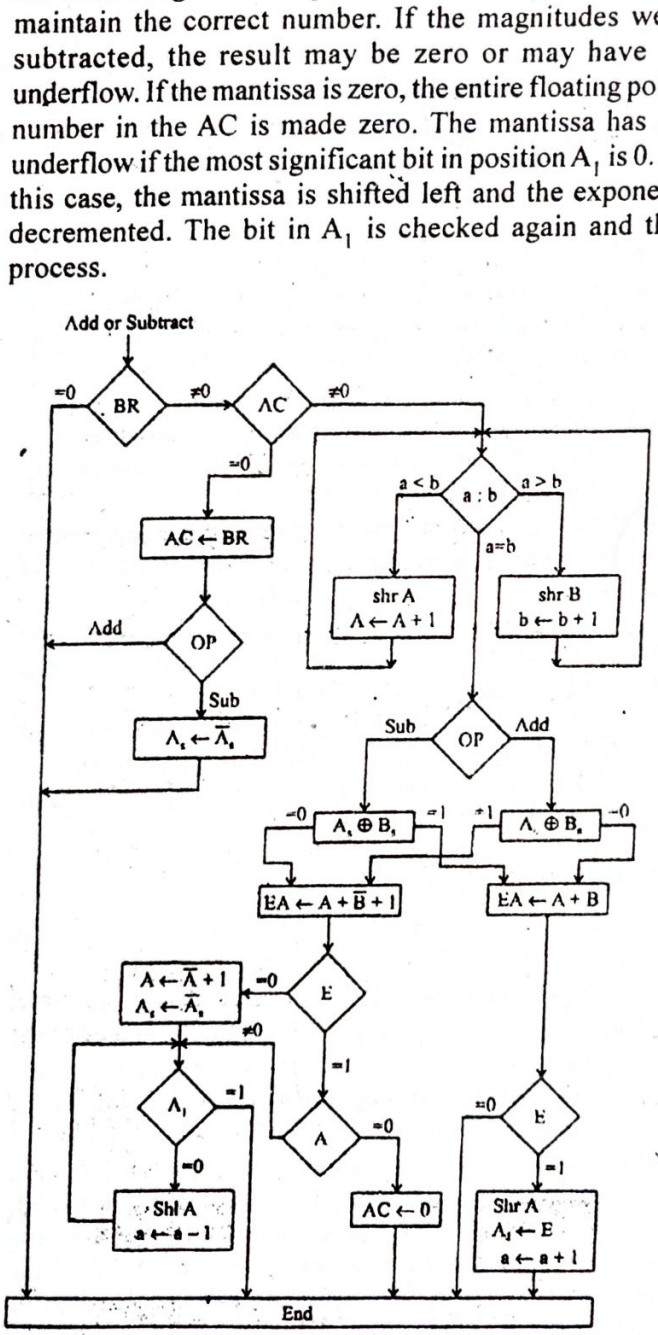


Fig. : Addition and subtraction of floating point number

Q.9 ***Write short note on priority interrupt.***

[R.T.U. 2017]

Ans. Data transfer between the CPU and I/O c initiated by the CPU. However, the CPU cannot transfer unless the device is ready to communic the CPU. There is also a possibility that several will requires service simultaneously. In this case th must also decide which device will get the servic

A priority interrupt is a system that estab priority over the various sources to determine condition is to be serviced first when two or more r arrive simultaneously. The system may also de which conditions are permitted to interrupt the co while another interrupt is being serviced. Higher p interrupt levels are assigned to request which if c or interrupted, could have serious consequences. D with high speed transfer such as magnetic disks ar high priority and slow devices such as keyboard re low priority. When two devices interrupt the com at the same time, the computer services the devic higher priority first.

Establishing the priority of simultaneous inte can be done by software or hardware. The sof priority interrupt identifies the highest priority sou software means such as polling. In polling method is one common branch address for all interrupts programs that take care of interrupts begins at the b address and polls the interrupt sources in sequence order in which they are tested determines the prior each interrupt. The highest priority is tested first a the interrupt signal is on, control branch provide the se to that source, otherwise the next lower priority sou tested and so on. The disadvantage of the software me is that if there are many interrupts, the time requir poll them can exceed the time available to service th device. The hardware priority interrupt unit functio an overall manager in an interrupt system environme accept interrupt request from many sources, determ which of the incoming request has the highest priority issues interrupt request.

Q.10 ***Write short note on IOP processor.***

[R.T.U. 2017, 2

Ans. Instead of having each interface communicate v the CPU, a computer may incorporate one or more exter processors and assign them the task of communicat directly with all I/O devices. An input–output proces (IOP) may be classified as a processor with direct mem acces capability that communicates with I/O devices. this configuration, the computer system can be divid into a memory unit, and a number of processors compris of the CPU and one or more IOPs. Each IOP takes c

data words from many different sources. For example, it may be necessary to take four bytes from an input device and pack them into 32-bit word before the transfer to memory. Data are gathered in the IOP at the device rate and bit capacity while the CPU is executing its own program. After the input data are assembled into a memory word, they are transferred from IOP directly into memory by "stealing" one memory cycle from the CPU. Similarly, an output device at the device rate and bit capacity.

The communication between the IOP and the devices attached to it is similar to the program control method of transfer. Communication with the memory is similar to the direct memory access method. The way by which the CPU and IOP communicate depends on the level of sophistication included in the system. In very large scale computers, each processor is independent of all others and any one processor can initiate an operation. In most computer system, the CPU is the master while the IOP is slave processor. The CPU is assigned the task of initiating all operations, but I/O instructions are executed in the IOP. CPU instruction provide operation to start an I/O transfer and also to test I/O status conditions needed for making decisions on various I/O activities. The IOP, in turn typically asks for CPU attention by means of an interrupt. It also responds to CPU requests by placing a status word in a prescribed location in memory to be examined later by a CPU program. When an I/O operation is desired the CPU informs the IOP where to find the I/O program and then leaves the transfer detail to the IOP.

Q.11 ***List various commands that an interface may receive from control line of the bus. Explain the process of handling an interrupt that occurs during the execution of a program, with the help of an example.*** ***[R.T.U. 2016]***

Ans. List of commands line: There are 4 types of commands that an interface may receive:

1. Control command: This command is issued to activate the peripheral and to inform it what to do? E.g., magnetic tape unit may be instructed to backspace tape by one record.

2. Status command: This command is used to check the various status conditions of the interface before a transfer is initiated.

3. Data Input command: This command causes the interface to read the data from the peripheral and places it into the interface buffer. Processor checks if data are available using status command and then issues a data input command. The interface places the data on data lines, where the processor accepts them.

4. Data Output command: This command causes the interface to read the data from the bus and saves it into the interface buffer.

Handling an interrupt that occurs during the execution of a program: On the occurrence of an interrupt, an interrupt request (in the form of a signal) is issued to the CPU. The CPU on receipt of interrupt request suspends the operation of the currently executing program, saves the context of the currently executing program and starts executing the program which services that interrupt request. This program is also known as interrupt handler. After the interrupting condition / device have been serviced, the execution of original program is resumed.

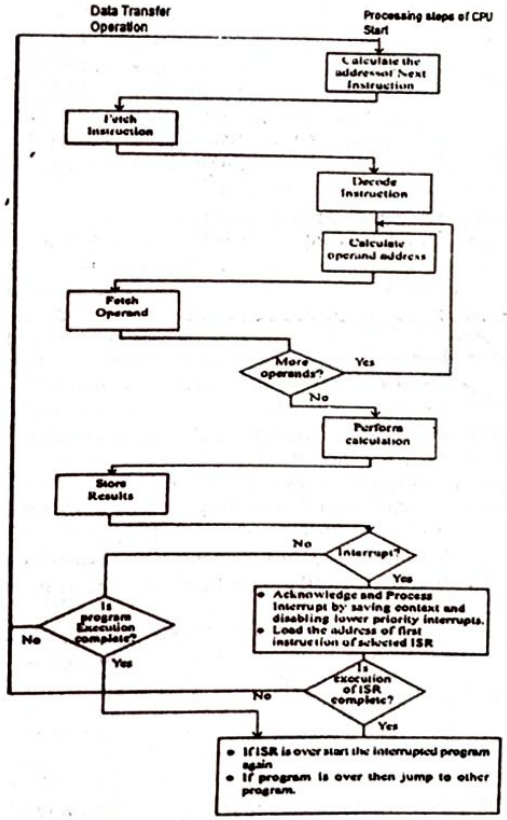


Fig. : Instruction Cycle with Interrupt Cycle

Thus, an interrupt can be considered as the interruption of the execution of an ongoing user program. The execution of user program resumes as soon as the interrupt processing is completed. Therefore, the user program does not contain any code for interrupt handling. This job is to be performed by the processor and the operating system, which in turn is also responsible for suspending the execution of the user program, and later after interrupt handling, resumes the user program from the point of interruption.

But when can a user program execution are interrupted? It will not be desirable to interrupt a program while an instruction is getting executed and is in a state like instruction decode. The most desirable place for program interruption would be when it has completed the previous instruction and is about to start a new instruction. Figure shows instruction execution cycle with interrupt cycle, where the interrupt condition is acknowledged. Note that even interrupt service routine is also a program and after acknowledging interrupt the next instruction executed through instruction cycle is the first instruction of interrupt servicing routine.

In the interrupt cycle, the responsibility of the CPU/ Processor is to check whether any interrupts have occurred checking the presence of the interrupt signal. In case no interrupt needs service, the processor proceeds to the next instruction of the current program.

In case an interrupt needs servicing then the interrupt is processed as per the following.

1. Suspend the execution of current program and save its context.
2. Set the Program counter to the starting address of the interrupt service routine of the interrupt acknowledged.
3. The processor then executes theinstructions in the interrupt-servicing program. The interrupt servicing programs are normally part of the operating system.
4. After completing the interrupt servicing program the CPU can resume the program has suspended in step 1 above.

Q.12 ***A DMA controller transfer 16-bits words to memory using cycle stealing. The words are assembled from a device that transmits character at a rate of 2400 characters per second. The CPU be is fetching and executing instruction at an average rate of 1 million instructions per second. By how much the CPU be slowed down because of the DMA transfer.*** ***[R.T.U. 2016]***

Ans. Hence, In 1 cycle stolen from the CPU, DMA controller sends a word to memory. Derive transmits 2400 characters per sec where 1 character = 8 bit ASCII.

(A) DMA controller is a special processor only used for DMA which acts as a intermediate between memory and I/O device.

Bit Pairs Q_0 Q_{-1}	
0 0	ASR
1 1	ASR
1 0	–, ASR
0 1	+, ASR

Step-1 given n = 5

	A	Q	Q_{-1}	initially value	
	00000	11011	0		
			(initially always 0)		
Step-2	01011	11011	0	A←A–M	00000
	00101	11101	1	ASR	10101
				$n = 4$	01011
Step-3	00010	11110	1	ASR	
				$n = 3$	
Step-4	10111	11110	1	A←A+M	00010
	11011	11111	0	ASR	10101
				$n = 2$	10111
Step-5	00110	11111	0	A←A–M	11011
	00011	01111	1	ASR	10101
				$n = 1$	00110
Step-6	00001	10111	1	ASR	
				$n = 0$	

Now when n will be 0 then stop algorithm and return AQ.

So final result AQ = 0000110111

Q.18 ***What are different type of DMA transfer? Explain.*** *[R.T.U. Dec. 2013]*

Ans. Different type of DMA Transfer

The two types of DMA transfers are flyby DMA transfers and fetch-and-deposit DMA transfers. The three common transfer modes are single, block, and demand transfer modes. These DMA transfer types and modes are described in the following paragraphs.

1. The fastest DMA transfer type is referred to as a single-cycle, single-address, or flyby transfer. In a flyby DMA transfer, a single bus operation is used to accomplish the transfer, with data read from the source and written to the destination simultaneously. In flyby operation, the device requesting service asserts a DMA request on the appropriate channel request line of the DMA controller. The DMA controller responds by gaining control of the system bus from the CPU and then issuing the pre-programmed memory address. Simultaneously, the DMA controller sends a DMA acknowledge signal to the requesting device. This signal alerts the requesting device to drive the data onto the system data bus or to latch the data from the system bus, depending on the direction of the transfer.

In other words, a flyby DMA transfer looks like a memory read or write cycle with the DMA controller supplying the address and the I/O device reading or writing the data. Because flyby DMA transfers involve a single memory cycle per data transfer, these transfers are very efficient; however, memory-to-memory transfers are not possible in this mode.

2. The second type of DMA transfer is referred to as a dual-cycle, dual-address, flow-through, or fetch-and-deposit DMA transfer. As these names imply, this type of transfer involves two memory or I/O cycles. The data being tansferred is first read from the I/O device or memory into a temporary data register internal to the DMA controller. The data is then written to the memory of I/O device in the next cycle. Although inefficient because the DMA controller performs two cycles and thus retains the system bus longer, this type of transfer is useful for interfacing devices with different data bus sizes.

For example, a DMA controller can perform two 16-bit read operations from one location followed by a 32-bit write operation to another location. A DMA controller supporting this type of transfer has two address registers per channel (source address and destination address) and bus-size registers, in addition to the usual transfer count and control registers. Unlike the flyby operation, this type of DMA transfer is suitable for both memory-to-memory and I/O transfers.

DMA Transfer Modes

In addition to DMA transfer types, DMA controllers have one or more DMA transfer modes. Single, block and demand are the most common transfer modes.

1. Single transfer mode transfers one data value for each DMA request assertion. This mode is the slowest method of transfer because it requires the DMA controller to arbitrate for the system bus with each transfer. This arbitration is not a major problem on a lightly loaded bus, but it can lead to latency problems when multiple devices are using the bus.

2. Block and demand transfer modes increase system throughput by allowing the DMA controller to perform multiple DMA transfers when the DMA controller has gained the bus. For block mode transfers, the DMA controller performs the entire DMA sequence as specified by the transfer count register at the fastest possible rate in response to a single DMA request from the I/O device.

3. For demand mode transfers, the DMA controller performs DMA transfers at the fastest possible rate as long as the I/O device asserts its DMA request. When the I/O device unasserts this DMA request, transfers are held off.

Similarly, C_4, C_5, C_6, C_7 and C_8 can be expanded to remove the recursion.

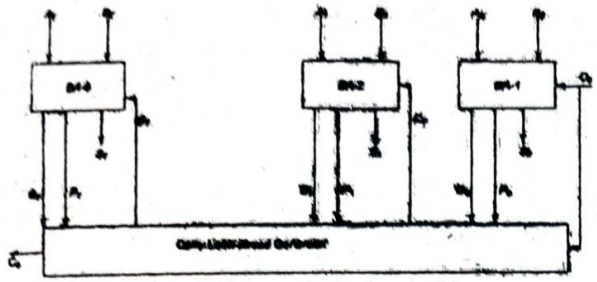


Fig. 2 : 8-bit carry look-ahead adder

The equations (2), (3), (4) and others, if derived, suggest that C_1, C_2, ..., C_8 can be generated directly from C_0. In other words, these eight carries depend only on the initial carry C_0. For this reason, these equations are called carry look-ahead equations. An 8-bit carry look-ahead adder (CLA) is shown in fig.

Total time needed to perform one addition : The maximum delay of the CLA is 6 × δ (for G_i and P_i generation, delay = for C_i generation, dely = and lastly another 3 for sum bit S_i) where is the average gate delay. The same holds good for any number fo bits because the adder delay does not depend on size of number (n). It depends on the number of levels of gates used to generate the sum and the carry bits.

Q.20 ***What is serial adder? Discuss it briefly with diagram.*** *[R.T.U. Dec. 2013]*

Ans. Serial Adder : If speed is not of great importance, a cost-effective option is to use a serial adder. This is the one which would accept bit by bit input of the n-bit numbers and there is a bit by bit output of the n-bit Sum.

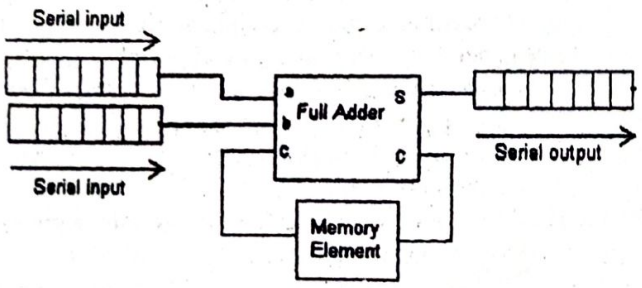


Fig. : The circuit for serial addition

i.e. In Serial adder, Bits are added a pair at a time (in one clock cycle)

A = an–1 an–2 ... a0, B=bn–1 bn–2 .... b0

In this adder we would be required one full adder and a memory element. Hence we see we require lesser hardware. The circuit for serial addition is as follows :

Example : Decimal

5 + 9 = 14

X = 5, Y = 9 , Sum = 14

Binary

0101 + 1001 = 1110

Addition of each step

Inputs			Outputs	
Cin	X	Y	Sum	Cout
0	1	1	0	1
1	0	0	1	0
0	1	0	1	0
0	0	1	1	0

addition starts from lowest.

Result = 1110 or 14

Q.21 ***Explain floating point addition and subtraction with suitable example.*** *[R.T.U. Dec. 2013]*

Ans. Floating Point Addition and Subtraction

For addition and subtraction, it is necessary to ensure that both operands have the same exponent value. This may require shifting the radix point on one of the operands to achieve alignment radix point on one of the operands to achieve alignment so due to alignment needs addition and subtraction are more complex than multiplication and division. In floating point arithmetic.

There are four basic phases of the algorithm for addition and subtraction.

(a) Check for zeros

(b) Align the significants.

(c) Add of subtract the significants.

(d) Normalize the result

Floating point Numbers	Addition and subtraction operation
$X = X_S XB^{X_E}$ $Y = Y_S XB^{Y_E}$	$X+Y = (X_S XB^{X_E - Y_E} + Y_S)XB^{Y_E}$ $X-Y = (X_S XB^{X_E - Y_E} - Y_S)XB^{Y_E}$ } $X_E \le Y_E$

Examples :

$X = 0.3 \times 10^2 = 30$

$Y = 0.2 \times 10^3 = 200$

$X + Y = (0.3 \times 10^{2-3} + 0.2) \times 10^3$

$= 0.23 \times 10^3 = 230$

$X - Y = (0.3 \times 10^{2-3} - 0.2) \times 10^3$

$= (-0.17) \times 10^3 = -170$

Memory mapped I/O

Advantages

(i) No opcodes or processor circuits are used up for I/O instructions.

(ii) All memory reference instructions, not just loads and stores, may be-used to manipulate I/O ports.

(iii) The number of available I/O ports addresses is virtually unlimited.

(iv) The hardware bus structure is simplified.

Disadvantages

(i) Part of the memory address space is used up.

(ii) Interfaces may need more circuitry to recognize longer addresses.

(iii) Memory reference instructions may be longer or slower than optimized I/O instructions.

Isolated I/O

Advantages

(i) The complete 1 Mbyte memory address space is available for use with memory.

(ii) Special instructions have been provided in the instruction set of the 8088/8086 to perform isolated I/O operations. These instructions have been tailored to maximize I/O performance.

(iii) I/O device addressed can be short.

(iv) Memory and I/O design can be separated.

(v) Programs are clears because I/O transfers are distinguished from other operations.

Disadvantages

(i) All input and output data transfers must take place between the AL or AX register and the I/O part.

(ii) Extra decoding and extra instructions.

Q.25 *Write short note on CPU-IOP Communication.*

Ans.CPU-IOP Communication: The communication between CPU and IOP may take different forms, depending on the particular computer considered. In most cases the memory unit acts as a message center where each processor leaves information for the other. To appreciate the operation of a typical IOP, we will illustrate by a specific example the method by which the CPU and IOP communicate. This is a simplified example that omits many operating details in order to provide an overview of basic concepts.

The sequence of operations may be carried out as shown in the flowchart of Fig. The CPU sends an instruction to test the IOP path.

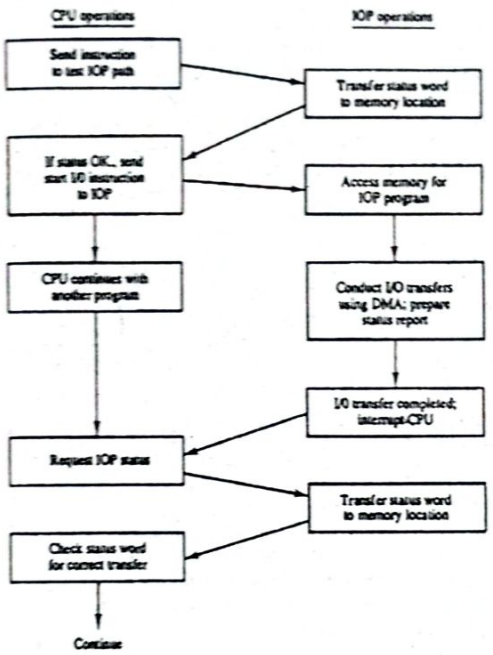


Fig. : CPU-IOP Communication

The IOP responds by inserting a status word in memory for the CPU to check. The bits of the status word indicate the condition of the IOP and I/O device, such as IOP overload condition, device busy with another transfer, or device ready for I/O transfer. The CPU refers to the status word in memory to decide what to do next. If all is in order, the CPU sends the instruction to start I/O transfer. The memory address received with this instruction tells the IOP where to find its program.

The CPU can now continue with another program while the IOP is busy with the I/O program. Both programs refer to memory by means of DMA transfer. When the IOP terminates the execution of its program, it sends an interrupt request to the CPU. The CPU responds to the interrupt by issuing an instruction to read the status from the IOP. The IOP responds by placing the contents of its status report into a specified memory location. The status word indicates whether the transfer has been completed or if any errors occurred during the transfer. From inspection of the bits in the status word, the CPU determines if the I/O operation was completed satisfactorily without errors.

The IOP takes care of all data transfers between several I/O units and the memory while the CPU is processing another program. The IOP and CPU are competing for the use of memory, so the number of devices that can be in operation is limited by the access time of the memory. It is not possible to saturate the memory by I/O devices in most systems, as the speed of most devices is much slower than the CPU. However, some very fast units, such as magnetic disks, can use an appreciable number of the available memory cycles. In that case, the speed of the CPU may deteriorate because it will often have to wait for the IOP to conduct memory transfers.

Q.26 *Explain character oriented protocol.*

Ans.Character-Oriented Protocol: The character-oriented protocol is based on the binary code of a character set. The code most commonly used is ASCII (American Standard Code for Information Interchange). It is a 7-bit code with an eighth bit used for parity. The code has 128 characters, of which 95 are graphic characters and 33 are control characters. The graphic characters include the upper and lowercase letters, the ten numerals, and a variety of special symbols. The control characters are used for the purpose of routing data, arranging the test in a desired format, and for the layout of the printed page. The characters that control the transmission are called communication control characters. These characters are listed in table. Each character has a 7-bit code and is referred to by a three-letter symbol. The role of each character in the control of data transmission is stated briefly in the function column of the table.

The SYN character serves as synchronizing agent between the transmitter and receiver. When the 7-bit ASCII code is used with an odd-parity bit in the most significant position, the assigned SYN character has the 8-bit code 00010110 which has the property that, upon circular shifting, it repeats itself only after a full 8-bit cycle. When the transmitter starts sending 8-bit characters, it sends a few characters first and then sends the actual message. The initial continuous string of bits accepted by the receiver is checked for a SYN character. In other words, with each clock pulse, the receiver checks the last eight bits received.

If they do not match the bits of the SYN character, the receiver accepts the next bit, rejects the previous high-order bit, and again checks the last eight bits received for a SYN character. This is repeated after each clock pulse and bit received until a SYN character is recognized. Once a SYN character is detected, the receiver has framed a character. From here on the receiver counts every eight bits and accepts them as a single character. Usually, the

receiver checks two consecutive SYN characters to remove any doubt that the first did not occur as a result of a noise signal on the line. Moreover, when the transmitter is idle and does not have any message characters to send, it sends a continuous string of SYN characters. The receiver recognizes these characters as a condition for synchronizing the line and goes into a synchronous idle state. In this state, the two units maintain bit and character synchronism even though no meaningful information is communicated.

Table : ASCII Communication Control Characters

Code	Symbol	Meaning	Function
0010110	SYN	Synchronous idle	Establishes synchronism
0000001	SOH	Start of heading	Heading of block message
0000010	STX	Start of text	Precedes block of text
0000011	ETX	End of text	Terminates block of text
0000100	EOT	End of transmission	Concludes transmission
0000110	ACK	Acknowledge	Affirmative acknowledgement
0010101	NAK	Negative acknowledge	Negative acknowledgement
0000101	ENQ	Inquiry	Inquire if terminal is on
0010111	ETB	End of transmission block	End of block of data
0010000	DLE	Data link escape	Special control character

Messages are transmitted through the data link with an established format consisting of a header field, a text field, and an error-checking field. A typical message format for a character-oriented protocol is shown in Fig. The two SYN characters assure proper synchronization at the start of the message. Following the SYN characters is the header, which starts with an SOH (start of heading) character. The header consists of address and control information. The STX character terminates the header and signifies the beginning of the text transmission. The text portion of the message is variable in length and may contain any ASCII characters except the communication control characters. The text field is terminated with the ETX character. The last field is a block check character (BCC) used for error checking. It is usually either a longitudinal redundancy check (LRC) or a cyclic redundancy check (CRC).

The receiver accepts the message and calculates its own BCC. If the BCC transmitted does not agree with the BCC calculated by the receiver, the receiver responds with a negative acknowledge (NAK) character. The message is then retransmitted and checked again. Retransmission will be typically attempted several times before it is assumed that the line is faulty. When the transmitted BCC matches the one calculated by the receiver, the response is a positive acknowledgment using the ACK character.

Transmission Example: In order to appreciate the function of a data communication processor, let us illustrate by a specific example the method by which a terminal and the processor communicate. The communication with the memory unit and CPU is similar to any I/O processor.

SYN	SYN	SOH	Header	STX	Text	ETX	BCC

Fig. : Typical message format for character-oriented protocol

Q.27 ***Explain Decimal arithmetic unit.***

Ans.Decimal Arithmetic Unit: The user of a computer prepares data with decimal numbers and receives results in decimal form. A CPU with an arithmetic logic unit can perform arithmetic micro-operations with binary data. To perform arithmetic operations with decimal data, it is necessary to convert the input decimal numbers to binary, to perform all calculations with binary numbers, and to convert the results into decimal. This may be an efficient method in applications requiring a large number of calculations and a relatively smaller amount of input and output data.

When the application calls for a large amount of input-output and a relatively smaller number of arithmetic calculations, it becomes convenient to do the internal arithmetic directly with the decimal numbers. Computers capable of performing decimal arithmetic must store the decimal data in binary-coded form. The decimal numbers are then applied to a decimal arithmetic unit capable of executing decimal arithmetic microoperations.

Electronic calculators invariably use an internal decimal arithmetic unit since inputs and outputs are frequent. There does not seem to be a reason for converting the keyboard input numbers to binary and again converting the displayed results to decimal, since this process requires special circuits and also takes a longer time to execute. Many computers have hardware for arithmetic calculations with both binary and decimal data. Users can specify by programmed instructions whether they want the computer to perform calculations with binary or decimal data.

In view of higher data transfer rate, parallel data transfer scheme has the inherent advantage over the serial one. However, it requires larger bus width and thus the associated hardware cost is higher. Serial communication is usually selected for data transfer over larger distance involving devices having low data transfer rates. For example, VDU terminals usually access the computer in a serial data transfer mode. By contrast, a disk drive having much higher data transfer rate is invariably connected in parallel data transfer mode. In general, the cost of bus structure along with the data transfer rate of the IO device dictate the selection of serial or parallel data transfer format in an IO subsystem.

Q.30 ***Describe the data transfer method using DMA.*** ***[R.T.U. 2017]***

OR

Write short note on DMA. ***[R.T.U. 2014]***

Ans. DMA : In Direct Memory Access (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 the proceeds to execute other tasks. When the transfer is made, the DMA requests memory cycles through the memory bus. When the request is granted by the memory controller, the DMA transfers the data directly into the memory. The CPU merely delays its memory access operation to allow the direct memory I/O transfer. Since peripheral speed is usually slower than processor speed, I/O memory transfers are infrequent compared to processor access to memory.

Working of DMA : The position of the DMA controller among the other components in a computer system is illustrated in fig. The CPU communicates with the DMA through the address and data buses as with any interface unit. The DMA has its own address, which activates the DS and RS lines. The CPU initializes the DMA through the data bus. Once the DMA receives the start control command, it can start the transfer between the peripheral device and the memory.

When the peripheral device sends a DMA request, the DMA controller activates the BR line, informing the CPU to relinquish the buses. The CPU responds with its BG line, informing the DMA that its buses are disabled. The DMA then puts the current value of its address register into the address but initiates the RD or WR signal, and sends a DMA acknowledge to the peripheral device. Note that the RD and WR lines in the DMA controller are bidirectional. The direction of transfer depends on the status of the BG line. When BG = 0, the RD and WR are input lines allowing the CPU to communicate with the internal DMA registers. When BG = 1, the RD and WR are output lines from the DMA controller to the random-access memory to specify the read or write operation for the data.

When the peripheral device receives a DMA acknowledge, it puts a word in the data bus (for write) or receives a word from the data bus (for read). Thus, the DMA controls the read or write operations and supplies the address for the memory. The peripheral unit can then communicate with memory through the data bus for direct transfer between the two units while the CPU is momentarily disabled.

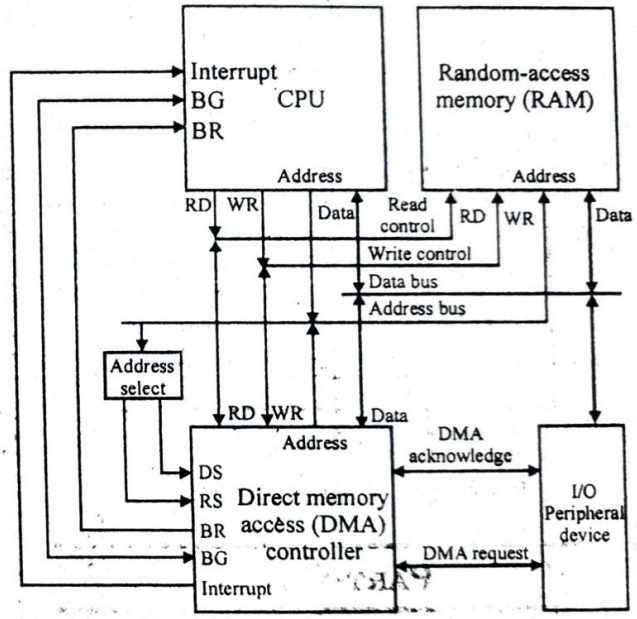


Fig. : DMA transfer in a computer system

For each word that is transferred, the DMA increments its address register and decrements its word count register. If the word count does not reach zero, the DMA checks the request line coming from the peripheral. For a high-speed device, the line will be active as soon as the previous transfer is completed. A second transfer is then initiated, and the process continues until the entire block is transferred. If the peripheral speed is slower, the DMA request line may come somewhat later. In this case the DMA disables the bus request line so that the CPU can continue to execute its program. When the peripheral requests a transfer, the DMA requests the buses again.

If the word count register reaches zero, the DMA stops any further transfer and removes its bus request. It also informs the CPU of the termination by means of an interrupt. When the CPU responds to the interrupt, it reads the content of the word count register. The zero value of

this register indicates that all the words were transferred successfully. The CPU can read this register at any time to check the number of words already transferred.

A DMA controller may have more than one channel. In this case, each channel has a request and acknowledge pair of control signals which are connected to separate peripheral devices. Each channel also has its own address register and word count register within the DMA controller. A priority among the channels may be established so that channels with high priority are serviced before channels with lower priority.

DMA transfer is very useful in many applications. It is used for fast transfer of information between magnetic disks and memory. It is also useful for updating the display in an interactive terminal. Typically, an image of the screen display of the terminal is kept in memory which can be updated under program control. The contents of the memory can be transferred to the screen periodically by means of DMA transfer.

Q.31 *(a) What is an interrupt service subroutine? How can the interrupt priority be resolved?*

(b) Explain in short programmed I/O and interrupt initiated I/O.

(c) What do you mean by synchronous and asynchronous data transfer? Explain hand shaking method asynchronous data transfer?

[R.T.U. 2016]

Ans.(a) Interrupt Service Subroutine : An interrupt

modem from the signal transitions that occur in the received data. Any frequency shift that may occur between the transmitter and receiver clocks is continuously adjusted by maintaining the receiver clock at the frequency of the incoming bit stream. The modem transfers the received data together with the clock to the interface unit. The interface or terminal on the transmitter side also uses the clock information from its modem. In this way, the same bit rate is maintained in both transmitter and receiver.

Contrary to asynchronous transmission, where each character can be sent separately with its own start and stop bits, synchronous transmission must send a continuous message in order to maintain synchronism. The message consists of a group of bits transmitted sequentially as a block of data. The entire block is transmitted with special control characters at the beginning and end of the block. The control characters at the beginning of the block supply the information needed to separate the incoming bits into individual characters.

One of the functions of the data communication processor is to check for transmission errors. An error can be detected by checking the parity in each information. The purpose of a data link protocol is to establish and terminate a connection between two stations, to identify the sender and receiver, to ensure that all messages are passed correctly without errors, and to handle all control functions involved in a sequence of data transfers. Protocols are divided into two major categories according to the message-framing technique used. These are character-oriented protocol and bit-oriented protocol.

Q.37 ***Write short note on the following.***
(a) Data transparency
(b) Bit oriented protocol

Ans.(a) Data transparency : The character-oriented protocol was originally developed to communicate with keyboard, printer, and display devices that use alphanumeric characters exclusively. As the data communication field expanded, it became necessary to transmit binary information which is not ASCII text. This happens, for example, when two remote computers send programs and data to each other over a communication channel. An arbitrary bit pattern in the text message becomes a problem in the character-oriented protocol. This is because any 8-bit pattern belonging to a communication control character will be interpreted erroneously by the receiver. For example, if the binary data in the text portion of the message has the 8-bit pattern 10000011, the receiver will interpret this as an ETX character and assume that it reached the end of the text field. When the text portion of the message is variable in length and contains bits that are to be treated without reference to any particular code, it is said to contain transparent data. This feature requires that the character recognition logic of the receiver be turned off so that data patterns in the text field are not accidentally interpreted as communication control information.

Data transparency is achieved in character-oriented protocols by inserting a DLE (data link escape) character before each communication control character. Thus, the start of heading is detected from the double character DLE SOH, and the text field is terminated with the double character DLE ETX. If the DLE bit pattern 00010000 occurs in the text portion of the message, the transmitter inserts another DLE bit pattern following it. The receiver removes all DLE characters and then checks the next 8-bit pattern. If it is another DLE bit pattern, the receiver considers it as part of the text and continues to receive text. Otherwise, the receiver takes the following 8-bit pattern to be a communication control character.

The achievement of data transparency by means of the DLE character is inefficient and somewhat complicated to implement. Therefore, other protocols have been developed to make the transmission of transparent data more efficient. One protocol used by Digital Equipment Corporation employs a byte count field that gives the number of bytes in the message that follows. The receiver must then count the number of bytes received to reach the end of the text field. The protocol that has been mostly used to solve the transparency problem and other problems associated with the character-oriented protocol is the bit-oriented protocol.

Ans.(b) Bit-Oriented Protocol: The bit-oriented protocol does not use characters in its control field and is independent of any particular code. It allows the transmission of serial bit stream of any length without the implication of character boundaries. Messages are organized in a specific format called a frame. In addition to the information field, a frame contains address, control, and error-checking fields. The frame boundaries are determined from a special 8-bit number called a flag. Examples of bit-oriented protocols are SDLC (synchronous data link control) used by IBM, HDLC (high-level data link control) adopted by the International Standards Organization, and ADCCP (advanced data communication control procedure) adopted by the American National Standards Institute.

Any data communication link involves at least two participating stations. The station that has responsibility for the data link and issues the commands to control the link is called the primary station. The other station is a secondary station. Bit-oriented protocols assume the presence of one primary station and one or more secondary stations. All communication on the data link is from the primary station to one or more secondary stations, or from a secondary station to the primary station.

The frame format for the bit-oriented protocol is shown in Fig.1. A frame starts with the 8-bit flag 01111110 followed by an address and control sequence. The information field is not restricted in format or content and can be of any length. The frame check field is a CRC (cyclic redundancy check) sequence used for detecting errors in transmission. The ending flag indicates to the receiving station that the 16 bits just received constitute the CRC bits. The ending frame can be followed by another frame, another flag, or a sequence of consecutive 1's. When two frames follow each other, the intervening flag is simultaneously the ending flag of the first frame and the beginning flag of the next frame. If no information is exchanged, the transmitter sends a series of flags to keep the line in the active state. The line is said to be in the idle state with the occurrence of 15 or more consecutive 1's. Frames with certain control messages are sent without an information field. A frame must have a minimum of 32 bits between two flags to accommodate the address, control, and frame check fields. The maximum length depends on the condition of the communication channel and its ability to transmit long messages error-free.

To prevent a flag from occurring in the middle of a frame, the bit-oriented protocol uses a method called zero insertion.

Flag 01111110	Address 8 bits	Control 8 bits	Information any number of bits	Frame check 16 bits	Flag 01111110

Fig.1: Frame format for bit-oriented protocol.

This requires that a 0 be inserted by the transmitting station after any succession of five continuous 1's. The receiver always removes a 0 that follows a succession of five 1's. Thus the bit pattern 01111111 is transmitted as 011111101 and restored by the receiver to its original value by removal of the following the five 1's. As a consequence, no pattern of 01111110 is ever transmitted between the beginning and ending flags.

Following the flag is the address field, which is used by the primary station to designate the secondary station address. When a secondary station transmits a frame, the address tells the primary station which secondary station originated the frame. An address field of eight bits can specify up to 256 addresses. Some bit-oriented protocols permit the use of an extended address field. To do this, the least significant bit of an address byte is set to 0 if another address byte follows A 1 in the least significant bit of a byte is used to recognize the last address byte.

Following the address field is the control field. The control field comes in three different formats, as shown in Fig.2: The information transfer format is used for ordinary data transmission. Each frame transmitted in this format contains send and receive counts. A station that transmits sequenced frames counts and numbers each frame. This count is given by the send count N_s. A station receiving sequenced frames counts each error-free frame that it receives. This count is given by the receive count N_r. The N_r count advances when a frame is checked and found to be without errors. The receiver confirms accepted numbered information frames by returning its N_r count to the transmitting station.

The P/F bit is used by the primary station to poll a secondary station to request that it initiate transmission.

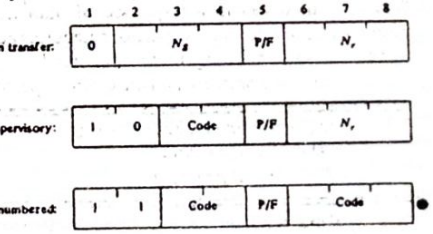


Fig.2 : Control field format in bit-oriented protocol.

It is used by the secondary station to indicate the final transmitted frame. Thus the P/F field is called P (poll) when the primary station is transmitting but is designated as F (final) when a secondary station is transmitting. Each frame sent to the secondary station from the primary station has a P bit set to 0. When the primary station is finished and ready for the secondary station to respond, the P bit is set to 1. The secondary station then responds with a number of frames in which the F bit is set to 0. When the secondary station sends the last frame, it sets

the F bit to 1. Therefore, the P/F bit is used to determine when data transmission from a station is finished.

The supervisory format of the control field is recognized from the first two bits being 1 and 0. The next two bits indicate the type of command. This follows by a P/F bit and a receive sequence frame count. The frames of the supervisory format do not carry an information field. They are used to assist in the transfer of information in that they confirm the acceptance of preceding frames carrying information, convey ready or busy conditions, and report frame numbering errors.

The unnumbered format is recognized from the first two bits being 11. The five code bits available in this format can specify up to 32 commands and responses. The primary station uses the control field to specify a command for a secondary station. The secondary station uses the control field to transmit a response to the primary station. Unnumbered-format frames are employed for initialization of link functions, reporting procedural errors, placing stations in a disconnected mode, and other data link control operations.

Q.38 *Explain BCD adder with block diagram.*

Ans. BCD Adder: Consider the arithmetic addition of two decimal digits in BCD, together with a possible carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than 9 + 9 +1 = 19, the 1 in the sum being an input-carry. Suppose that we apply two BCD digits to a 4-bit binary adder. The adder will form the sum in binary and produce a result that may range from 0 to 19. These binary numbers are listed in Table and are labeled by symbols K, Z_8, Z_4, Z_2, and Z_1. K is the carry and the subscripts under the letter Z represent the weights 8, 4, 2, and 1 that can be assigned to the four bits in the BCD code. The first column in the table lists the binary sums as they appear in the outputs of a 4-bit binary adder. The output sum of two decimal numbers must be represented in BCD and should appear in the form listed in the second column of the table. The problem is to find a simple rule by which the binary number in the first column can be converted to the correct BCD digit representation of the number in the second column.

In examining the contents of the table, it is apparent that when the binary sum is equal to or less than 1001, the corresponding BCD number is identical and therefore no conversion is needed.

Table : Derivation of BCD Adder

Binary Sum					BCD Sum					
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	Decimal
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

When the binary sum is greater than 1001, we obtain a nonvalid BCD representation. The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation and also produces an output-carry as required.

One method of adding decimal numbers in BCD would be to employ one 4-bit binary adder and perform the arithmetic operation one digit at a time. The low-order pair of BCD digits is first added to produce a binary sum. If the result is equal or greater than 1010, it is corrected by adding 0110 to the binary sum. This second operation will automatically produce an output-carry for the next pair of significant digits. The next higher-order pair of digits, together with the input-carry, is then added to produce their binary sum. If this result is equal to or greater than 1010, it is corrected by adding 0110. The procedure is repeated until all decimal digits are added.

The logic circuit that detects the necessary correction can be derived from the table entries. It is obvious that a correction is needed when the binary sum has an output carry K = 1. The other six combinations from 1010 to 1111 that need a correction have a 1 in position Z_8. To distinguish them from binary 1000 and 1001 which also have a 1 in position Z_8, we specify further that either Z_4 or Z_2 must have a 1. The condition for a correction

and an output-carry can be expressed by the Boolean function

$$C = K + Z_8Z_4 + Z_8Z_2$$

When C = 1, it is necessary to add 0110 to the binary sum and provide an output-carry for the next stage. A BCD adder is a circuit that adds two BCD digits in parallel and produces a sum digit also in BCD. A BCD adder must include the correction logic in its internal construction. To add 0110 to the binary sum, we use a second 4-bit binary adder as shown in Fig. The two decimal digits, together with the input-carry, are first added in the top 4-bit binary adder to produce the binary sum. When the output-carry is equal to 0, nothing is added to the binary sum. When it is equal to 1, binary 0110 is added to the binary sum through the bottom 4-bit binary adder. The output-carry generated from the bottom binary adder may be ignored, since it supplies information already available in the output-carry terminal.

A decimal parallel-adder that adds n decimal digits needs n BCD adder stages with the output-carry from one stage connected to the input-carry of the next-higher-order stage. To achieve shorter propagation delays, BCD adders include the necessary circuits for carry look-ahead. Furthermore, the adder circuit for the correction does not need all four full-adders, and this circuit can be optimized.

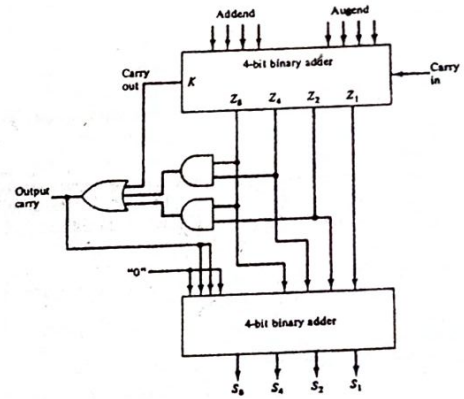


Fig. : Block diagram of BCD adder.

□□□

PART-B

Q.7 *Explain how virtual address is translated into real address in segmented memory system.* *[R.T.U. 2017]*

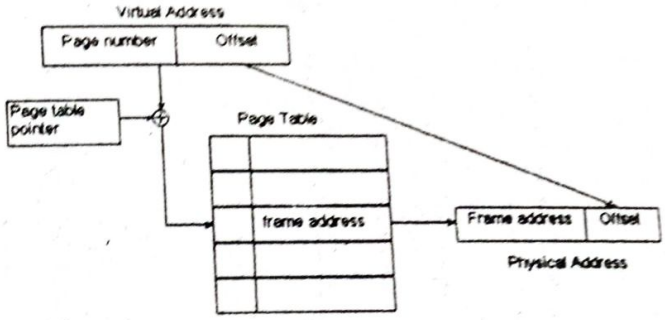
Ans. In a virtual memory system, the program memory is divided into fixed sized pages and allocated in fixed sized physical memory frames. The pages do not have to be contiguous in memory. A page table keeps track of where each page is located in physical memory. This allows the operating system to load a program of any size into any available frames. Only the currently used pages need to be loaded. Unused pages can remain on disk until they are referenced. This allows many large programs to be executed on a relatively small memory system. A resident flag in the page table indicates whether or not the page is in memory. The page table also includes several other flags to keep track of memory usage. A use flag is set whenever the page is referenced. A dirty bit is set whenever the page is changed to inform the operating system that the page in memory is different than the page on disk.

The addresses that appear in programs are the virtual addresses or program addresses. For every memory access, either to fetch an instruction or data, the CPU must translate the virtual address to a real physical address. A virtual memory address can be considered to be composed of two parts: a page number and an offset into the page. The page number determines which page contains the information and the offset specifies which byte within the page. The size of the offset field is the log base 2 of the size of a page.

The virtual addresses can be represented as

13 bits	10 bits
page number	offset

To convert a virtual address into a physical address, the CPU uses the page number as an index into the page table. If the page is resident, the physical frame address in the page table is concatenated in front of the offset to create the physical address.



Q.8 *Briefly compare the mapping procedure used in cache memory organization.* *[R.T.U. 2017]*

Ans. The different Cache mapping techniques are as follows:-

(i) Direct Mapping

(ii) Associative Mapping

(iii) Set Associative Mapping

Consider a cache consisting of 128 blocks of 16 words each, for total of 2048(2K) works and assume that the main memory is addressable by 16 bit address. Main memory is 64K which will be viewed as 4K blocks of 16 works each.

(i) Direct Mapping

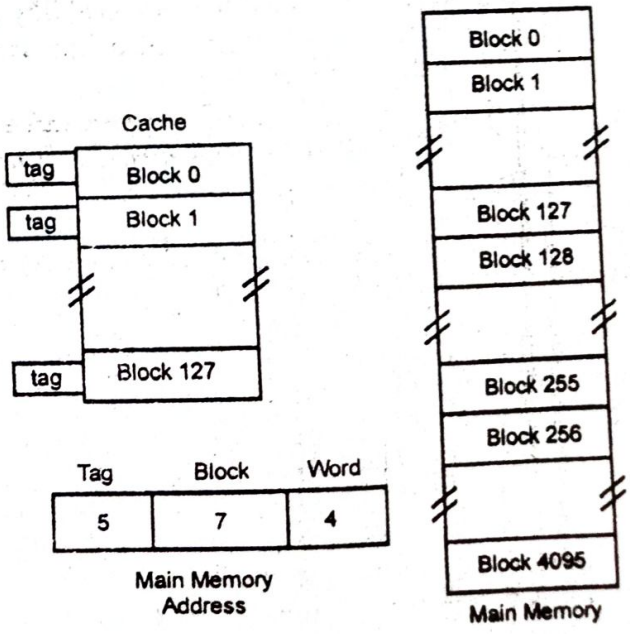


Fig. 1 : Direct Mapping

i. The simplest way to determine cache locations in which store memory blocks is direct mapping technique.

ii. In this block J of the main memory maps on to block J modulo 128 of the cache. Thus main memory blocks 0, 128, 256....is loaded into cache is stored at block 0. Block 1, 129, 257....are stored at block 1 and so on.

iii. Placement of a block in the cache is determined from memory address. Memory address is divided into 3 fields, the lower 4-bits selects one of the 16 words in a block.

iv. When new block enters the cache, the 7-bit cache block field determines the cache positions in which this block must be stored.

v. The higher order 5-bits of the memory address of the block are stored in 5 tag bits associated with its location in cache. They identify which of the 32 blocks that are mapped into this cache position are currently resident in the cache.

(ii) Associative Mapping

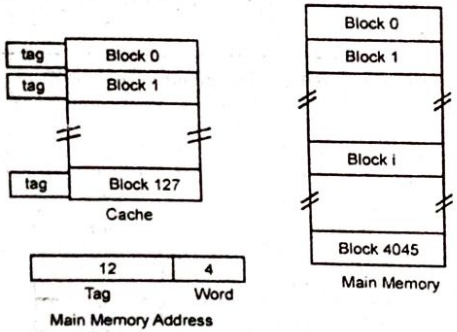


Fig. 2 : Associative Mapping

i. This is more flexible mapping method, in which main memory block can be placed into any cache block position.

ii. In this, 12 tag bits are required to identify a memory block when it is resident in the cache.

iii. The tag bits of an address received from the processor are compared to the tag bits of each block of the cache to see, if the desired block is present. This is known as Associative Mapping technique.

iv. Cost of an associated mapped cache is higher than the cost of direct-mapped because of the need to search all 128 tag patterns to determine whether a block is in cache. This is known as associative search.

(iii) Set-Associated Mapping

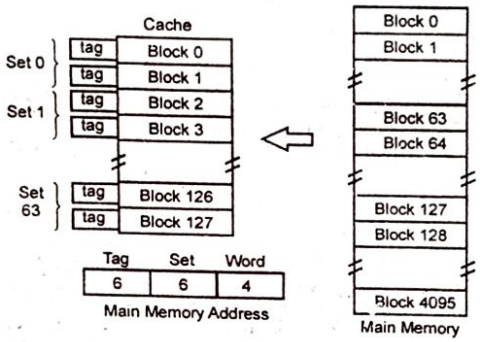


Fig. 3 : Set-Associated Mapping

i. It is the combination of direct and associative mapping technique.

ii. Cache blocks are grouped into sets and mapping allow block of main memory reside into any block of a specific set. Hence contention problem of direct mapping is simplified and at the same time, hardware cost is reduced by decreasing the size of associative search.

iii. For a cache with two blocks per set. In this case, memory block 0, 64, 128.....4032 map into cache set 0 and they can occupy any two block within this set.

iv. Having 64 sets means that the 6 bit set field of the address determines which set of the cache might contain the desired block. The tag bits of address must be associatively compared to the tags of the two blocks of the set to check if desired block is present. This is also called two-way associative search.

Q.9 ***Explain the role of virtual Memory.*** *[R.T.U. 2016]*

Ans. Virtual Memory : In a memory hierarchy system, programs and data are first stored in auxiliary memory.

Portions of a program of data are brought into main memory as they are needed by the CPU. Virtual memory is a concept used in the some large computer system that permit the user to construct programs as though a large memory space were available, equal to the totality of auxiliary memory. Each address that is referenced by the CPU goes through an address mapping from the so-called virtual address to a physical address in main memory. Virtual memory is used to give programmers the illusion that they have a very large memory at their disposal, even though the computer actually has a relatively small main memory. A virtual memory system provides a mechanism for translating program-generated addresses into correct main memory locations. This is done dynamically, while programs are being executed in the CPU. The translation or mapping is handled automatically by the hardware by means of a mapping table.

Q.10 ***Draw and explain the memory hierarchy in a digital computer. What are the advantages of cache memory over main memory?*** *[R.T.U. 2016]*

Ans. Memory Hierarchy : The total capacity of a computer can be visualized as being a hierarchy to components. The memory hierarchy system consists of all storage devices employed in a computer system from the auxiliary memory to main memory, to an even smaller and faster cache memory accessible to the high speed processing logic.

The memory which resides at the top level of the memory hierarchy is a very high speed memory called a cache memory. CPU logic is usually faster than main memory access time. With the result that processing speed is limited primarily by the speed of the main memory. A technique used to compensate for the mismatch in operating speeds is to employ an extremely fast small cache between the CPU and main memory whose access time is close to processor logic clock cycle time. The cache is used for storing segments of programs currently being executed in the CPU and temporary data frequency needed in the present calculations. By making programs and data available at a rapid rate, it is possible to increase the performance rate of the computer.

The main memory resides at the second highest level of the memory hierarchy. It occupies a central position by being able to communicate directly with the CPU and with auxiliary memory device through an I/O processor. Only programs and data currently needed by the processor reside in main memory.

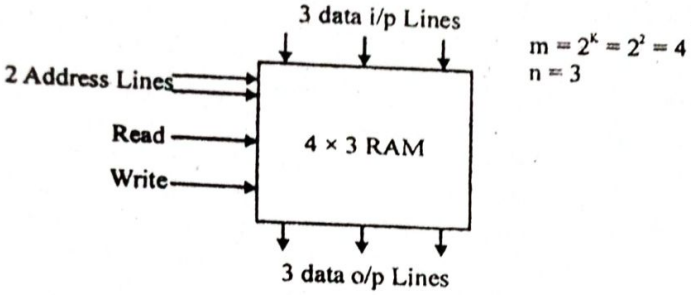


Fig. 1 : A block diagram of a 4×3 RAM

Binary Cell : It consists of a strong element to store a single bit of an information. On the basis of a construction of a binary cell, RAM is of two types static, RAM and dynamic RAM. In static RAM, the storing element used in a binary cell is flip flop and in dynamic RAM, the storing element is capacitor.

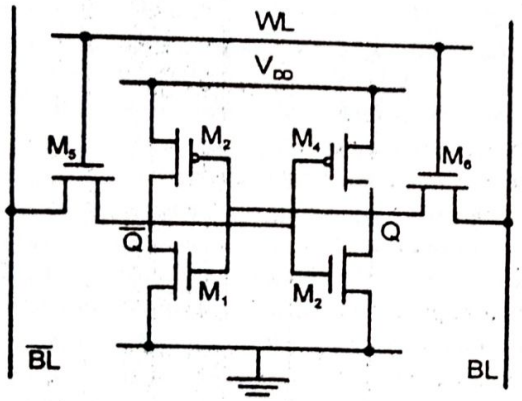


Fig. 2 : SRAM binary cell (6 transistors)

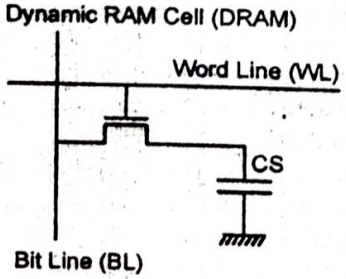


Fig. 3 : DRAM binary cell (one capacitor and one transistor)

Q.12 ***Design a 16 by 4 RAM.***

[R.T.U. 2014]

Ans. Design 16 × 4 RAM

16×4 RAM means, a RAM consists of 16 words of 4 bit each.

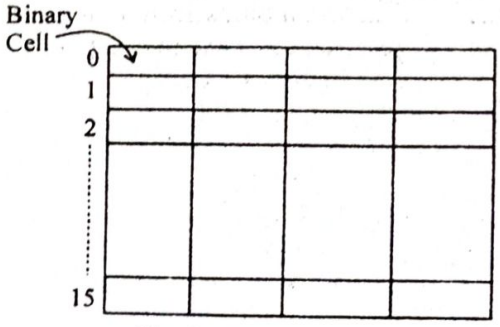


Fig. 1 : 16×4 RAM block diagram

To design a 16 × 4 RAM using 4 × 4 RAM chips. The address lines are of 4 bit A_3, A_2, A_1, A_0.

A_3, A_2 are used to select memory chip and A_1, A_0 are used to select word in a chip.

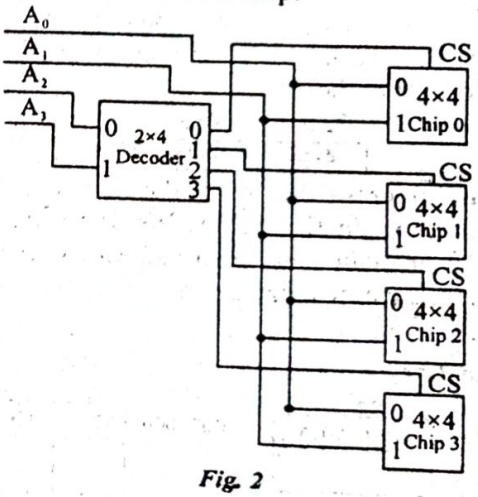


Fig. 2

A_3	A_2	A_1	A_0	
0	0	0	0	Chip 0
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	Chip1
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	Chip 2
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	Chip 3
1	1	0	1	
1	1	1	0	
1	1	1	1	

Q.13 ***What is Cache mapping? Explain directmapping for 256×8 RAM and 64×8 Cache memory.***

[R.T.U. Dec. 2013]

Snoopy Protocols: Snoopy protocols distribute the responsibility for maintaining cache coherence among all of the cache controllers in a multiprocessor system. A cache must recognize when a line that it holds is shared with other caches. When an update action is performed on a shared cache line, it must be announced to all other caches by a broadcast mechanism. Each cache controller is able to "snoop" on the network to observed these broadcasted notification and react accordingly. Snoopy protocols are ideally suited to a bus-based multiprocessor, because the shared bus provides a simple means for broadcasting and snooping. With a write-invalidate protocol, there can be multiple readers but only one write at a time. Initially, a line may be shared among several caches for reading purposes.

When one of the caches wants to perform a write to the line it first issues a notice that invalidates that tine in the other caches, making the line exclusive to the writing cache. Once the line is exclusive, the owning processor can make local writes until some other processor requires the same line. With a write update protocol, there can be multiple writers as well as multiple readers. When a processors wishes to update a shared line, the word to be updated is distributed to all others, and caches containing that line can update it.

Q.20 ***What are the 3 different cache memory schemes? Explain in detail with suitable examples.*** *[R.T.U. 2015]*

Ans. The 3 different cache memory schemes are -

1. Fully-Associative
2. Direct Mapped / 1-Way Set Associative
3. Set Associative

1. Fully Associative

Fig. 1 shows an example of a Fully Associative cache.

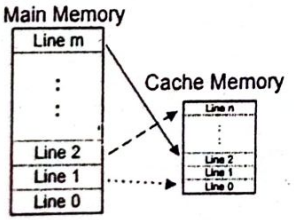


Fig. 1

This organizational scheme allows any line in main memory to be stored at any location in the cache. Fully-Associative cache does not use cache pages, but only uses lines. Main memory and cache memory are both divided into lines of equal size.

For example the fig. 1 shows that Line 1 of main memory is stored in Line 0 of cache. However, this is not the only possibility, Line 1 could have been stored anywhere within the cache. Any cache line may store any memory line, hence it is named, Fully Associative.

A Fully Associative scheme provides the best performance because any memory location can be stored at any cache location. The disadvantage is the complexity of implementing this scheme. The complexity comes from having to determine if the requested data is present in cache. In order to meet the timing requirements, the current address must be compared with all the addresses present in the TRAM. This requires a very large number of comparators that increase the complexity and cost of implementing large caches. Therefore, this type of cache is usually only used for small caches, typically less than 4K.

2. Direct Mapped / 1-Way Set Associative

Direct Mapped cache is also referred to as 1-Way set associative cache. Fig. 2 shows an example of a direct map scheme.

In this scheme, main memory is divided into cache pages. The size of each page is equal to the size of the cache. Unlike the fully associative cache, the direct mapped cache may only store a specific line of memory within the same line of cache. For example, Line 0 of any page in memory must be stored in Line 0 of cache memory. Therefore if Line 0 of Page 0 is stored within the cache and Line 0 of Page 1 is requested, then Line 0 of Page 0 will be replaced with Line 0 of Page 1.

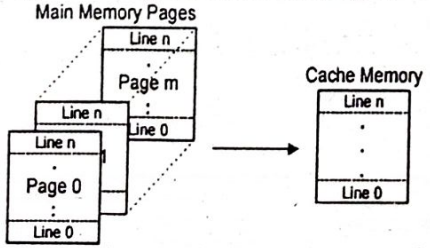


Fig. 2 : A Direct Map Scheme

This scheme directly maps a memory line into an equivalent cache line, hence it is named Direct Mapped cache. A Direct Mapped cache scheme is the least complex of all three caching schemes. Direct Mapped cache only requires that the current requested address be compared with only one cache address. Since this implementation is less complex, it is far less expensive than the other caching schemes. The disadvantage is that Direct Mapped cache is far less flexible making the performance much slower, especially when jumping between cache pages.

3. Set Associative

A Set Associative cache scheme is a combination of Fully Associative and Direct Mapped caching schemes. A set associate scheme works by dividing the cache SRAM into equal sections (2 or 4 sections typically) called cache ways. The cache page size is equal to the size of the cache way. Each cache way is treated like a small direct mapped cache.

To make the explanation clear, lets look at a specific example. Fig. 3 shows a diagram of a 2-Way Set-Associative cache scheme.

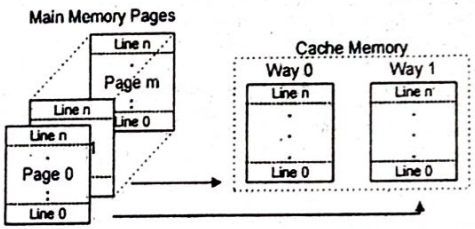


Fig. 3 : 2-Way Set Associative Cache Scheme

In this scheme, two lines of memory may be stored at any time. This helps to reduce the number of times the cache line data is written-over.

Q.21 ***Discuss the general characteristics of memory system. What is the use of virtual memory and discuss its concept.*** *[R.T.U. Dec. 2013]*

Ans. General Characteristics of Computer Memory Systems

Location
- Internal (e.g. processor registers, main memory, cache)
- External (e.g. optical disks, magnetic disks, tapes)

Capacity
- Number of words
- Number of bytes

Unit of Transfer
- Word
- Block

Access Method
- Sequential
- Direct
- Random
- Associative

Performance
- Access time
- Cycle time
- Transfer rate

Physical Type
- Semiconductor
- Magnetic
- Optical
- Magneto-optical

Physical Characteristics
- Volatile / non volatile
- Erasable / nonerasable

Organization
- Memory modules

Virtual Memory and its Uses : *Refer to Q.9.*

Address Mapping Using Pages : The table implementation of the address mapping is simplified if the information in the address space and the memory space are each divided into group of fixed size. The physical memory is broken down into groups of equal size called blocks, which may range form 64 to 4096 words each. The term page refers to groups of address space of the same size. For example, if a page or block consists if 1K words, then using the previous example, address space is divided into 1024 pages and main memory is divided into 32 blocks. Although both a page and a block are split into groups of 1K words, a page refers to the organization of address space, while a block refers to the organization of memory space. The programs are also considered to be split into page. Portions of programs are moved from auxiliary memory to main memory in records equal to the size of a page. The term "page frame" is sometimes used to denote a block.

Consider a computer with an address space of 8K and a memory space of 4K. If we split each into groups of 1K words we obtain eight pages and four blocks as shown in fig.1. At any given time, up to four pages of address space may reside in main memory in any one of the four blocks.

The mapping from address space to memory space is facilitated if each virtual address is considered to be represented by two numbers : A page number address and a line within the page. In a computer with 2^p words per page, p bits are used to specify a line address and the

remaining high-order bits of the virtual address specify the page number. In the example, of fig.1 a virtual address has 13 bits. Since each page consists of $2^{10} = 1024$ words, the high-order three bits of a virtual address will specify one of the eight pages and the low-order 10 bits give the line address within the page. Note that the line address in address space and memory space is the same; the only mapping required is from number to a block number.

The organization of the memory mapping table in a paged system is shown in fig.2. The memory-page table consists of eight words, one for each page. The address in the page table denotes the page number and the content of the word gives the block number where that page is stored in main memory. The table shown that pages 1,2,5, and 6 are now available in main memory in blocks 3,0,1 and 2, respectively. A presence bit in each location indicates whether the page has been transferred form auxiliary memory into main memory. A 0 in the presence bit indicates that this page is not available in main memory. The CPU references a word in memory with a virtual address of 13 bits. The three high-order bits of the virtual address specify a page number and also an address for the memory-page table. The content of the world page table at the page number address is read out into the memory table buffer register. If the presence bit is a 1, the block number thus read is transferred to the high-order bits of the main memory address register. The line number from the virtual address is transferred into the 10 low-order bits of the memory address register. A read signal to main memory transfers the content of the word to the main memory buffer register ready to be used by the CPU. If the presence bit in the word read from the page table is 0, it signifies that the content of the word referenced by the virtual address does not reside in main memory. A call to the operating system is then generated to fetch the required page down form auxiliary memory and place it into main memory before resuming computation.

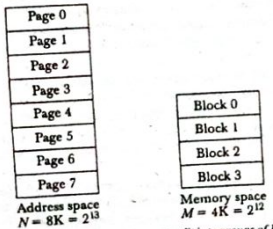


Fig.1 : Address space and memory space split into groups of 1K words

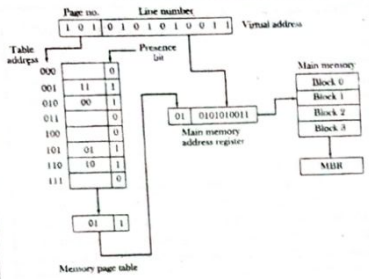


Fig. 2 : Memory table in a paged system

Address Space and Memory Space : An address used by a programmer will be called a virtual address and the set of address space of such addresses is called the address space. An address in main memory is called a memory space location or physical address. The set of such locations is called the memory space. Thus the address space is the set of addresses generated by programs as they reference instructions and data; the memory space consists of the actual main memory locations directly addressable for processing. In most computers the address and memory spaces are identical. The address space is allowed to be larger than the memory space in computers with virtual memory.

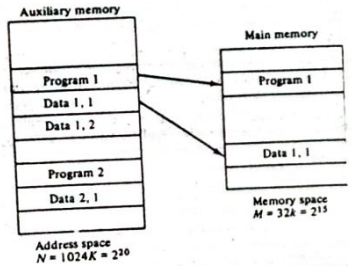


Fig. 3: Relation between address and memory space in a virtual memory system

As an illustration, consider a computer with a main-memory capacity of 32K words (K = 1024). Fifteen bits

are needed to specify a physical address in memory since $32K = 2^{15}$. Suppose that the computer has available auxiliary memory for storing $2^{20} = 1024K$ words. Thus auxiliary memory has a capacity for storing information equivalent to the capacity of 32 main memories. Denoting the address space by N and the memory space by M, we then have for this example N = 1024K and M = 32K.

In a multiprogram computer system, programs and data are transferred to and from auxiliary memory and main memory based on demands imposed by the CPU. Suppose that program 1 is currently being executed in the CPU. Program 1 and a portion of its associated data are moved from auxiliary memory into main memory as shown in Fig. Portions of programs and data need not be in contiguous locations in memory since information is being moved in and out and empty spaces may be available in scattered locations in memory.

In a virtual memory system, programmers are told that they have the total address space at their disposal. Moreover, the address field of the instruction code has a sufficient number of bits to specify all virtual addresses. In example, the address field of an instruction code will consist of 20 bits but physical memory addresses must be specified with only 15 bits. Thus CPU will reference instructions and data with a 20-bit address, but the information at this address must be taken from physical memory because access to auxiliary storage for individual words will be prohibitively long. (For efficient transfers, auxiliary storage moves an entire record to the main memory.) A table is then needed, to map a virtual address of 20 bits to a physical address of 15 bits. The mapping is a dynamic operation, which means that every address is translated immediately as a word is referenced by CPU.

The mapping table may be stored in a separate memory as shown in fig. in main memory. In the first case, an additional memory unit is required as well as one extra memory access time. In the second case, the table takes space from main memory and two accesses to memory are required with the program running at half speed.

Q.22 ***Explain the concept of time-shared common bus.***

Ans. Time-Shared Common Bus: A common-bus multiprocessor system consists of a number of processors connected through a common path to a memory unit. A time-shared common bus for five processors is shown in Fig.1. Only one processor can communicate with the memory or another processor at any given time.

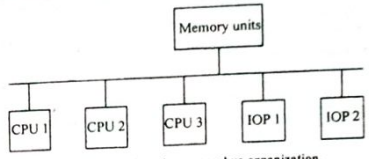


Fig. 1 : Time-shared common bus organization.

Transfer operations are conducted by the processor that is in control of the bus at the time. Any other processor wishing to initiate a transfer must first determine the availability status of the bus, and only after the bus becomes available can the processor address the destination unit to initiate the transfer. A command is issued to inform the destination unit what operation is to be performed. The receiving unit recognizes its address in the bus and responds to the control signals from the sender, after which the transfer is initiated. The system may exhibit transfer conflicts since one common bus is shared by all processors. These conflicts must be resolved by incorporating a bus controller that establishes priorities among the requesting units.

A single common-bus system is restricted to one transfer at a time. This means that when one processor is communicating with the memory, all other processors are either busy with internal operations or must be idle waiting for the bus. As a consequence, the total overall transfer rate within the system is limited by the speed of the single path. The processors in the system can be kept busy more often through the implementation of two or more independent buses to permit multiple simultaneous bus transfers. However, this increases the system cost and complexity.

A more economical implementation of a dual bus structure is depicted in Fig.2. Here we have a number of local buses each connected to its own local memory and to one or more processors. Each local bus may be connected to a CPU, an IOP, or any combination of processors. A system bus controller links each local bus to a common system bus. The I/O devices connected to the local IOP, as well as the local memory, are available to the local processor. The memory connected to the common system bus is shared by all processors. If an IOP is connected directly to the system bus, the I/O devices attached to it may be made available to all processors. Only one processor can communicate with the shared memory and other common resources through the system

bus at any given time. The other processors are kept busy communicating with their local memory and I/O devices.

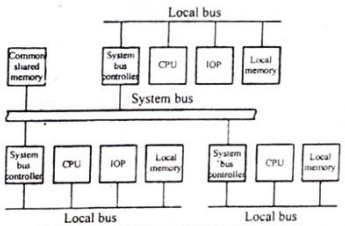


Fig. 2 : System Bus Structure for Multiprocessor.

Part of the local memory may be designed as a cache memory attached to the CPU. In this way, the average access time of the local memory can be made to approach the cycle time of the CPU to which it is attached.

Q.23 ***Write note on the following:***
(a) Crossbar switch
(b) Hypercube Interconnection

Ans.(a) Crossbar switch: The crossbar switch organization consists of a number of crosspoints that are be placed at intersections between processor buses and memory module paths. Fig.1 shows a crossbar switch interconnection between four CPUs and four memory modules. The small square in each crosspoint is a switch that determines the path from a processor to a memory module. Each switch point has control logic to set up the transfer path between a processor and memory. It examines the address that is placed in the bus to determine whether its particular module is being addressed. It also resolves multiple requests for access to the same memory module on a predetermined priority basis.

Fig.2 shows the functional design of a crossbar switch connected to one memory module. The circuit consists of multiplexers that select the data, address, and control from one CPU for communication with the memory module.

Priority levels are established by the arbitration logic to select one CPU when two or more CPUs attempt to access the same memory. The multiplexer are controlled with the binary code that is generated by a priority encoder within the arbitration logic.

A crossbar switch organization supports simultaneous transfers from memory modules because there is a separate path associated with each module.