COMPUTER ORGANISATION

SECOND EDITION

Dr.G.S.Bapi Raju.
Prof., Dept. of CSE, GRIET, HYD.

Gokaraju Rangaraju Institute of Engineering and Technology
(Autonomous)
Survey No. 288, Nizampet-Bachupally Road, Bachupally,
Kukatpally, Hyderabad, Telangana 500090.
www.griet.ac.in

First Impression, 2016
Second Impression, 2017

Published by GRIET, Hyderabad

PREFACE

About Computer Organisation:

Computer Architecture and Organization is the study of internal working, structuring and implementation of a computer system. Organization of computer system is the way of practical implementation which results in the realization of architectural specifications of a computer system. As stated by Gordon Moore, the co-founder of Intel, and familiarly known as Moore's law, which states that the number of transistors per square inch on integrated circuits had doubled every year since their invention. Even the processor speeds reaching higher speeds but still, the bottleneck lies with the FSB. Even though more innovations are in the pipeline, the basic computer organisation and the structure might not change drastically in the near future. One more thing to notice is the GT/s is replacing the MHz which refers to the physical clock speed of the GPU, CPU, FSB, Memory Clock or controller, where GT/s is a calculated amount of data based on the bus width and speed of whatever you're wanting to measure.

About the author:

The author holds three post graduation degrees M.Tech in computer science and engineering, M.Sc in Nuclear Physics, MBA in HRM and a Doctoral degree in Nuclear Physics from Andhra University. Also, certifications from renowned institutions which includes IDCPA (International Diploma in Computer Programming and Applications) from NCC (National Computing Centre) U.K and MCSE from Microsoft, CHSM from SISI, ministry of industries, Govt of India etc.

This book originated from a series of lectures on Computer Organisation which had given by the author in the past two decades at both undergraduate and postgraduate level in different engineering colleges.

ACKNOWLEDGEMENTS

I owe much and place my profound gratitude to my students and colleagues for all the discussions and questions they have raised.

I am immensely thankful and gratefully acknowledge the support and encouragement rendered by our Director, Prof. P.S.Raju, Principal Dr. Jandhyala N Murthy and Sr. Administration officer Dr. K.V.S. Raju in bringing out this book.

Dr.G.S.Bapiraju
Prof.,Dept of CSE
GRIET, Hyderabad.

CONTENTS

1. BASIC STRUCTURE OF COMPUTERS

1.1 Types of Computers: There are different types of computers that differ widely in size, cost, computational power and intended use.

i). Desktop computer (personal computer):

This is the most common computer which has found wide use in homes, schools and business offices.

ii). Laptop or notebook computer:

These are a portable and compact version of the personal computer, works on both battery and AC power supply.

iii). Workstation:

These systems are desktop systems but more powerful than normal systems in terms of resolution, graphics, input-output capabilities and computational power. These are widely used in engineering applications like interactive design works.

iv). Enterprise systems and servers:

Enterprise systems or mainframes are used for business data processing in medium to large corporations that require more processing power and storage capacity than workstations.

These are mainly used in server/client technology. where a number of clients communicate with these servers for database access.

v). Supercomputers:

These computers are very powerful in terms of processing as well as the storage capacities.

Used for the large-scale numeric applications like weather forecasting, military applications and aircraft design and simulation etc.

Fig: India's fastest supercomputer developed by ISRO.

SAGA-220 (Supercomputer for Aerospace with GPU Architecture-220 TeraFLOPS)

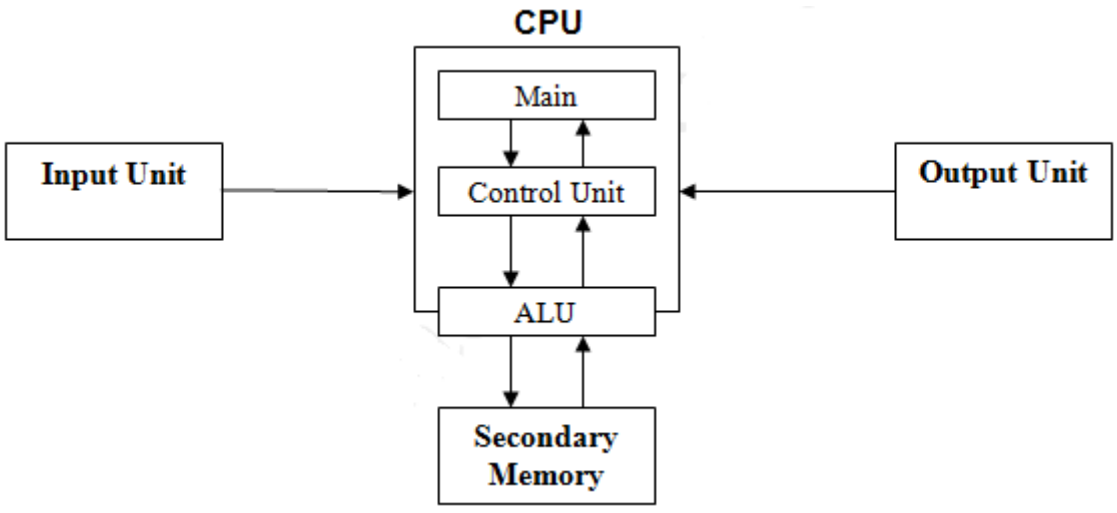
1.2 Functional Units: A Computer consists of five functionally independent main parts:

i). Input Unit: Input units are the devices used to input data or information to the computer. Ex: Keyboard, mouse, joysticks etc.

ii). Memory: Memory unit is used to store programs or data. There are two classes of storage called primary storage and secondary storage memory.

Primary storage memory is a high-speed memory with low memory access time, used to store programs which are connected with the process or execution. Ex: RAM or random access memory.

Secondary storage memory: Secondary memory is slower and cheaper than primary memory. Used mainly to store large amounts of data and programs. Ex: Hard disk, CD/DVD ROM, Tapes etc.



iii). Arithmetic and Logic unit: The arithmetic or logic unit (ALU) is the unit where most computer operations are executed. To do any process for ex: an addition first the operands are brought into the processor and the operation is performed by the ALU. ALU is many times faster than any other device connected to a computer system.

iv). Control units: The purpose of the control unit is to control and coordinate the operations of the memory, ALU, input and output and other units used in a computer.

v). Output: The output device is a device which takes the data from the CPU unit and displays (Monitor) or prints (Printer) or communicates to the outside world.

1.3 Basic operational concepts:

Any activity in a computer is governed by instructions. To perform a given task an appropriate program consisting of a list of instructions is stored in the memory. Individual instructions are brought from the memory into the processor, which executes the specified operations. Data to be used as operands also stored in the memory.

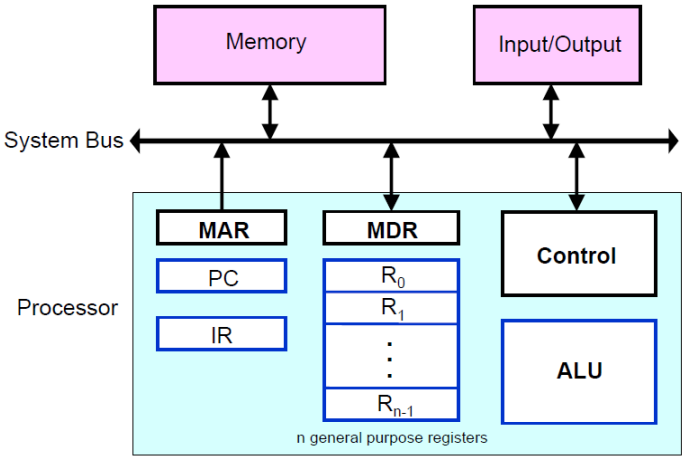
Ex: Load loca, R1
Add R1, R0

The first instruction is fetched from the memory into the processor. Next the operand at loca is fetched and transfers into processor register R1. The second instruction adds the contents of the register R1 and R0 and places the sum into R0.

Apart from ALU and CU the processor unit contains a number of registers used for several different purposes.

1.3.1 The instruction register (IR):

An instruction register is the part of a CPU's control unit that holds the instruction currently being executed or decoded.



1.3.2 The program counter (PC):

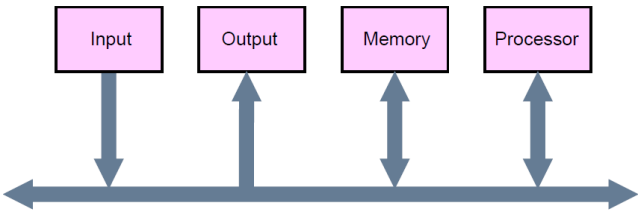
A program counter is a register in a computer processor that contains the address (location) of the instruction being executed at the current time. As each instruction gets fetched, the program counter increases its stored value by 1.

The memory address register (MAR) and memory data register (MDR) facilitate communication with the memory.

1.4 Bus and Bus structures:

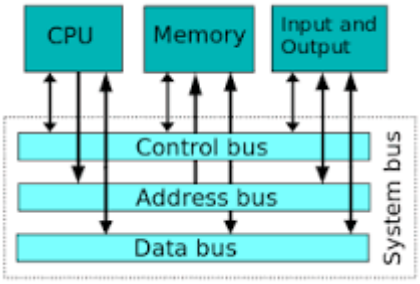
1.4.1 BUS: All parts within a PC are normally connected through copper lines (electronic pathways) known as tracks. A group of these lines or tracks are known as bus. The simplest way to interconnect functional units is to use a single bus.

1.4.2 Single bus structure: Because the bus can be used for only one transfer at a time, only two units can actively use the bus at any given time as shown in the figure.



1.4.3 Multiple bus structure can allow concurrent transfers which is faster and with better performance but at an increased cost.

1.5 Types of Buses:



i). Data Bus: Bus dedicated for the transmission of data signals.. Sometimes same bus can be multiplexed to use for both data and address.

ii). Address Bus: Bus dedicated for the transmission of address signals.

iii). Control Bus: Bus dedicated for the transmission of control signals.

iv). System Bus: Also known as Front side bus (FSB) connects main components in a computer mother board. System bus can be synchronous or asynchronous.

v). Internal Bus: The internal bus connects all the internal components of a computer, such as CPU and memory, to the motherboard.

vi). External Bus: The external bus, or expansion bus, is provided for connected external devices, such as printers, external hard disks, Modems etc.

vii). I/O Bus: I/O bus or Input output bus is meant for communication between a PC and any external device. Normally connects to the processor-memory bus or backplane bus.
Ex: SCSI, PCI,USB, Firewire.

viii). Backplane bus: The backplane is an interconnection structure within the chassis, Used as an intermediary bus connecting I/O busses to the processor-memory bus.

ix). Synchronous bus: Synchronous transmissions are synchronized by an external clock signal. Supports high data transfer rate. Devices (transmitter and receivers) communicating on the bus must synchronized by clock and use same clock rate. Ex: processor-memory buses.

x). Asynchronous bus: For asynchronous data transfer, there is no common clock signal between the sender and receivers. Therefore, the sender and the receiver first need to agree on a data transfer speed. Slower data transfer rate. A Ex: I/O buses.

1.6 Buffer registers: The devices connected to a bus vary widely in their speed of operation. For example keyboards, printers are relatively slow. Storage devices, Processor and memory devices operate very fast. To compensate, a special buffer registers are used to hold the information temporarily during transfers.

1.7 Software: Software is a group of instructions. Depending on their usage they are categorized into different forms.

1.7.1 System software: is a collection of programs that are used to manage the system resources and control the system operation. This includes managing the storage and retrieval of files, running the program's, controlling I/O units Translating programs from source form to machine language, for this compilers are used.

1.8 Performance measures:

Performance depends on the response time, means how quick the program is returning the output. Programs are usually written in high-level language, to translate them to machine language translators are needed. So for best performance it is necessary to design a compiler, the machine instruction set and the hardware in a coordinated way.

i). Cache:

A program will be executed faster if the movement of instructions and data between the main memory and the processor is minimized, this is achieved by using cache.

ii). Processor clock:

Processor circuits are controlled by a timing signal called a clock. The clock defines regular time intervals called clock cycles. Latest processors support Giga-hertz cycles. Where hertz is cycles per second.

iii). Basic performance Equation:

$T = (N * S) / R$ where T = processor time required to execute a program
N = machine language instructions
S = average number of steps needed to execute one machine instruction.
R = clock rate, cycles per second.

iv). Pipelining operation:

The pipelining processor completes the processing of one instruction in each clock cycle which means that the rate of instruction processing is four times that of sequential operation. This is done by overlapping the execution of successive instructions using a technique called pipelining.

v). Superscalar operation:

A higher degree of concurrency can be achieved if multiple instruction pipelines are implemented in the processor. If multiple paths are created, it becomes possible to start the execution of several instructions in every clock cycle. This mode of operation is called superscalar execution.

vi). Instruction set CISC and RISC: Instruction set plays a major role in the performance of a computer system. Two types of instruction sets CISC (Complex instruction set computing) and RISC (reduced instruction set computing) are normally used. In recent years the hybrid – a combination of CISC and RISC is also under development.

RISC: Simple instructions require a small number of basic steps to execute. But a large number of instructions may be needed to perform a given programming task. This could lead to a large value of N and a small value of S.

CISC: On the other hand if individual instructions perform more complex operations fewer instructions will be needed, leading to a lower value of N and a larger value of S.

So the key consideration is the use of pipelining. It is much easier to implement efficient pipelining in processors with simple instruction sets. So the suitability of the instruction set for pipelined execution is an important and often deciding consideration.

vii). Compiler:

A compiler translates a high-level language program into a sequence of machine instructions. An optimising compiler takes advantage of various features of the target processor to reduce the product N x S, which is the total number of clock cycles needed to execute a program.

1.9. Performance Measurement:

i). MIPS:

MIPS stands for 'Million Instructions per Second', The MIPS rating of a CPU refers to how many low-level machine code instructions a processor can execute in one second.

MIPS = (Instruction count / Execution time) * 10^6.

Unfortunately, using this number as a way of measuring processor performance is completely pointless because no two chips use exactly the same kind of instructions, execution method, etc.

ii). MFLOPS:

Another popular alternative to measure execution time is million floating-point operations per second, or MFLOPS (megaflops).

The formula for MFLOPS is simply

MFLOPS = (Number of floating-point operations in a program / Execution time) * 10^6.

iii). SPEC Rating:

The performance measure is the time (T) it takes a computer to execute a given benchmark which is a real-time application program. A non-profit organisation called SPEC (system performance evaluation corporation) selects and publishes representative application programs for different application domains, together with test results for many commercially available computers.

Thus a SPEC rating of 50 means that the computer under test is 50 times fast as the UltraSPARC 10 for this particular benchmark.

$$SPEC\,rating = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test}}$$

$$SPEC\,rating = \left(\prod_{i=1}^{n} SPEC_i\right)^{\frac{1}{n}}$$

Where n = is the number of programs in the suite.

Because the actual execution time is measured, the SPEC rating is a measure of the combined effect of all factors affecting performance, including the compiler, operating system, the processor and the memory of the computer being tested.

1.10 Multiprocessors and Multi computers:

1.10.1 Multiprocessors*:*

Computer systems with a number of processor units are called as multiprocessor systems. These systems either execute a number of different application tasks in parallel or they execute subtasks of a single large task in parallel.

All the processors have access to same memory termed as **shared memory multiprocessors**. Cost is drastically increased because of the need of more complex interconnection networks.

1.10.2 Multi-computers*:*

Instead of using multiprocessors it is also possible to connect a number of individual computer systems as a group to achieve high total computation power. These individual systems have access only to their own memory used in them.

The tasks are processed individually and shared by exchanging messages, referred as **message passing multi-computers**.

1.11 Data Representation:

Data Types:

The data types used in digital computers may be classified as:

i). Numbers used in arithmetic computations,
ii). letters of the alphabet used in data processing.
iii). Other discrete symbols used for specific purposes.

Number systems:

A number system of the *base*, or *radix*, r is a system that uses distinct symbols for r digits.

i). Decimal system: The decimal number system employs the radix 10 system. The 10 symbols are 0,1,2,3,4,5,6,7,8,9.

Ex: The string of digits 125.5 is represented as: $1 \times 10^2 + 2 \times 10^1 + 5 \times 10^0 + 5 \times 10^{-1}$

ii). Binary system: The binary number uses the radix 2. The two digit symbols used are 1 and 0.

Ex: $(101010)_2 = 1 \times 2^{16} + 0 \times 2^8 + 1 \times 2^4 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = (42)_{10}$

iii). Octal system: The octal number uses the radix (octal) 8. The 8 symbols are

0,1,2,3,4,5,6,7.

Ex: The string of digits 125.5 is represented as: $1 \times 8^2 + 2 \times 8^1 + 5 \times 8^0 + 5 \times 8^{-1}$

iv). Hexadecimal system: The hexadecimal number uses the radix 16. The 16 symbols are

0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

Ex: $(F5)_{16} = F \times 16^1 + 5 \times 16^0 = 15 \times 16 + 5 \times 1 = 240 + 1 = (241)_{10}$

Conversions:

1	6	4	6	5	3	Octal
1	1 1 0	1 0 0	1 1 0	1 0 1	0 1 1	Binary
	D	9	A	B		Hexadecimal

1.12 Decimal Representation:

1.12.1 Binary coded decimal: The bit assignment used for the decimal digits is the straight binary assignment listed below:

Decimal Number	Binary coded decimal
0	0000 0000
1	0000 0001
2	0000 0010
3	0000 0011
4	0000 0100
5	0000 0101
.	.
10	0001 0000
20	0010 0000
99	1001 1001
248	0010 0100 1000

1.12.2 Alphanumeric Representation:

An alphanumeric character set is a set of elements that includes the 10 decimal digits, the 26 letters of the alphabet and a number of special characters, such as $, + and =. Such a set contains between 64 and 128 elements.

The standard alphanumeric binary code is the ASCII (American Standard Code for information interchange) uses seven bits to code 128 characters.

Character	Binary code	Character	Binary code	Character	Binary code
A	100 0001	0	011 0000	space	010 0000
B	100 0010	1	011 0001	.	010 1110
		2	011 0010	(	010 1000
Z	101 1010			+	010 1011
		9	011 1001	$	010 0100
				*	010 1010

1.13 Complements:

When the value of the base r is substituted in the name, the two types are referred to as

i). 2's and 1's complement for binary numbers and
ii). 10s and 9s complement for decimal numbers.

i). (r-1)'s complement:

Given a number N in base r having n digits, the (r-1)'s complement of N is defined as

$(r^n - 1) - N$

where N is the given number and " r " is radix, and r^n represents a number that consists of a single 1 followed by " n " 0's.

For decimal numbers :	For binary numbers : radix r = 2, and r-1 = 1 so the 1's complement of N is
Ex: the 9's complement of 546700 is (N=546700, n=6 digits, r=10)	Ex: the 1's complement of 10101 n (no of digits) =5, radix (binary) r = 2, N=10101)
$(r^n - 1) - N$ $= (10^6 - 1) - 546700$ $= 999999 - 546700$ $= 453299$	$(r^n - 1) - N$ $= (2^n - 1) - 10101$ $= (2^5 - 1) - 10101$ $(2^5 = 32 = 100000)$ $= (100000 - 1) - 10101$ $= 11111 - 10101$ $= 01010$

Thus the 1's complement of a binary number is obtained by subtracting each digit from 1.

However the subtraction of a binary digit from 1 causes the bit to change from 0 to 1 or from 1 to 0. Therefore the 1's complement of a binary number is formed by changing 1's to 0's and 0's into 1's.

ii). r's complement:

The r's complement of an n-digit number N in base r is defined as $r^n - N$ for n != 0 and 0 for N=0. Comparing with the (r-1)'s complement, we note that the r's complement is obtained by adding 1 to the (r-1)'s complement since

$r^n - N = [(r^n - 1) - N] + 1$

Thus the 10^{th} complement of the decimal 2389 is 7610 +1 = 7611,
obtained by adding 1 to the 9's complement value.

The 2's complement of binary number 1011 is 0100 + 1 = 0101,
obtained by adding 1 to the 1's complement value.

1.13.1 Subtraction of unsigned numbers:

The subtraction of two n-digit unsigned numbers M-N (N!=0) in base r can be done as follows:

1. Add the minuend M to the r's (2's) complement of the subtrahend N. This performs

$M + (r^n - N) = M - N + r^n$.

2. If M >= N , the sum will produce an end carry r^n which is discarded, and what is left is the result M – N.

3. If M < N the sum does not produce end carry, take the r's complement of the sum and place a negative sign in front.

Ex: 72532(M) – 13250(N) = 59282, The 10's complement of 13250 is 86750. therefore, M = 72532 10's complement of N = + 86750 sum = 159282 discard end carry 1 answer = 59282	Ex: 13250(M) – 72532(N) = - 59282 10's Complement of N (72532) = 99999-72532 = 27467 + 1 M = 13250 10's complement of N = + 27468 sum = 40718 There is no end carry, means answer is negative. Now 10's complement the sum 40718 (99999 – 40718 = 59281 + 1 = 59282) answer = - 59282

Note: The negative answer is recognized by the absence of the end carry and the complemented result.

Binary number subtraction is done in the similar way as we did in the above examples

Ex: Let X=1010100 and Y=1000011 (To perform X –Y, Add value of X to the 2's complement of value of Y) 2's complement of Y = 0111100 + 1 = 0111101 X = 1010100 2's complement of Y = 0111101 Sum = 10010001 Discard end carry 2^7 = - 10000000 Answer X – Y = 0010001	(To perform Y - X, Add value of Y to the 2's complement of value of X) 2's complement of X = 0101011+ 1 = 0101100 Y = 1000011 2's complement of X = 0101100 Sum = 1101111 There is no end carry, so answer is negative 2's complement of sum 1101111 = 0010001 Answer Y - X = 0010001

1.14 Fixed-point representation:

Positive integers, including zero can be represented as unsigned numbers. To represent a signed number, the convention is to make the sign bit (at the left most position) equal to 0 for positive and to 1 for negative. The representation of a binary point in a register is characterized by a position in the register.

There are two ways of specifying the position.

i). By giving it a fixed position or
ii). by employing a floating point representation.

1.14.1 Integer representation: When an integer binary number is positive, the sign is represented by 0 and the magnitude by a positive binary number. When the number is negative the sign is represented by 1 but the rest of the number may be represented in one of three possible ways.

1. Signed magnitude representation
2. Signed 1's complement representation
3. Signed 2's complement representation

Ex:	Positive signed number	+ 14	0 000 1110
	Signed magnitude representation	-14	1 000 1110
	Signed 1's complement representation	-14	1 111 0001
	Signed 2's complement representation	-14	1 111 0010

1.14.2 Arithmetic Addition: The addition of two numbers in the signed magnitude system follows the rules of ordinary arithmetic.

a). If the signs are same we add the two magnitudes and give the sum the common sign
b). If the signs are different we subtract the smaller magnitude from the larger and give the result the sign of the larger magnitude.

Ex: (+25) +(- 37) = - (37 -25) = -12

1.14.3 2's complement addition: The rule for adding numbers in the signed 2's complement system does not require a comparison or subtraction, only addition and complementation.

a). Add two numbers including their signed bits and discard any carry digit
b). Note that negative numbers must initially be in 2's complement and if the sum obtained after the addition is negative, it is in 2's complement form.

+6	0000 0110	-6	1111 1010
+13	0000 1101	+13	0000 1101
+19	**0001 0011**	+7	**0000 0111**
+6	0000 0110	-6	1111 1010
-13	1111 0011	-13	1111 0011
-7	**1111 1001**	-19	**1110 1101**

1.14.4 Arithmetic subtraction:

Subtraction of two signed binary numbers can be done by taking the 2's complement of subtrahend (including the sign bit) and adding to the minuend. Any carry is discarded.

Subtraction operation can be changed to an addition if the sign of the subtrahend is changed.

Ex: $(\pm A) - (+B) = (\pm A) + (-B)$
$(\pm A) - (-B) = (\pm A) + (+B)$

Changing a positive number to negative number or negative number to positive can be done by its 2's complement.

Ex: (-6)-(-13) = +7

First, let the negative numbers be 2's complemented

2's complement of 6 is 0110 → 1001 + 1 → 1010 = 1010
2's complement of 13 is 1101 → 0010 + 1 → 0011 = 0011

Now for subtraction add minuend to the 2's complement of subtrahend

Minuend (6) = 1010
2's complement of subtrahend (13) = 1101 (0011 → 1100 + 1 → 1101)
Sum = 10111 discard the left most 1 so the answer is +7.

1.14.5 Decimal fixed point representation:

The representation of a signed decimal number in BCD is similar to the representation of signed numbers in binary.

The signed magnitude system is difficult to use with computers, so signed complement system either 9's complement or 10 's complement is used. 10's complement is more often used.

Ex: (+375) + (-240) = +135

10's complement of 240 = 999-240 = 759 + 1 = 760

0 375 (0000 0011 0111 0101) $_{BCD}$
+9 760 (1001 1101 0110 0000) $_{BCD}$
0 135 (0000 0001 0011 0101) $_{BCD}$

1.15 Floating Point Representation:

The floating point representation of a number has two parts. The first part represents a signed, fixed-point number called the mantissa. The second part designates the position of the decimal (or binary) point and is called the exponent. The fixed point mantissa may be a fraction or an integer.

Ex: Decimal Number +6132.789
Fraction part +0.6132789 exponent +04
Scientific notation $+0.6132789 \times 10^{+4}$

Floating point is always interpreted to represent a number in the following form:

$m \times r^e$

Ex: Binary number +1001.11
Fractional part 01001110 exponent 000100

The exponent has the equivalent binary number +4. The floating point number is equivalent to

$m \times 2^e = +(.1001110)_2 \times 2^{+4}$

1.15.1 Other Binary & Decimal codes:

Other binary codes		Other decimal codes			
		BCD		Excess -3	Excess – 3
Decimal	Gray code	8421	2421		gray
0	0000	0000	0000	0011	0010
1	0001	0001	0001	0100	0110
2	0011	0010	0010	0101	0111
3	0010	0011	0011	0110	0101
…					
10	1111	1010	0101	0000	0000
11	1110	1011	0110	0001	0001
…					
15	1000	1111	1010	1111	1011

1.15.2 Weighted code:

In a weighted code, the bits are multiplied by the weights indicated and the sum of the weighted bits gives the decimal digit.

For example, the bit combination 1101, when weighted by the respective digits 2421 gives the decimal equivalent 2x1+4x1+2x0+1+1=7. The BCD code can be assigned the weights 8421 and for this reason, it is sometimes called the 8421 code.

1.16 IEEE 754 Standards:

IEEE Standard 754 floating point is the most common representation today for real numbers on computers, including Intel-based PC's, Macintoshes, and most Unix platforms.

Fixed-point has a fixed window of representation, which limits it from representing very large or very small numbers. Also, fixed-point is prone to a loss of precision when two large numbers are divided.

Floating-point representation the most common solution uses scientific notation to encode numbers, with a base number and an exponent.

For example, 123.456 could be represented as 1.23456×10^2

Storage Layout: IEEE floating point numbers have three basic components:

a). The sign,
b). The exponent,
c). The mantissa.

Floating Point Components			
	Sign	**Exponent**	**Fraction**
Single Precision	1 [31]	8 [30–23]	23 [22–00]
Double Precision	1 [63]	11 [62–52]	52 [51–00]

a). The Sign Bit : The sign bit 0 denotes a positive number, and 1 denotes a negative number. Flipping the value of this bit flips the sign of the number.

b). The Exponent: The exponent field needs to represent both positive and negative exponents. To do this, a **bias** is added to the actual exponent in order to get the stored exponent.

For IEEE single-precision floats, this value is 127. Thus, an exponent of zero (o) means that 127 is stored in the exponent field.

A stored value of 200 indicates an exponent of (200–127), or 73. For double precision, the exponent field is 11 bits, and has a bias of 1023.

c). The Mantissa: The mantissa, also known as the significand, represents the precision bits of the number. For example, the number 50 can be represented as any of these:

$.5000 \times 10^2$ or 0.050×10^3 or $5000. \times 10^{-2}$

Normalisation: In order to maximize the quantity of representable number, floating-point numbers are typically stored in normalized form.

A floating-point number is said to be normalized if the most significant digit of the mantissa is **nonzero**. For example, the decimal number 350 is normalized but 00035 is not.

Regardless the position in the mantissa, the **number is normalized only if its leftmost digit is nonzero**. To keep the same value for the floating-point number, the exponent is modified.

1.17 Example: Represent 1259.125 in single precision and double precision format

Convert Decimal number to binary:

a). Integer Part: 1259 = divide the number with 2 until 1 or 0 reminder gets.

100 1110 1011

b) Decimal part: .125 = Multiply the number with 2 until 1 or 0 reminder gets**.**

.001

So binary value of 1259.125 is

100 1110 1011.001

1259		*1024*	*1*		.125		
629	**1**	*512*	*0*		* 2		
314	**1**	*256*	*0*		.250	**0**	
157	**0**	*128*	*1*		* 2		
78	**1**	*64*	*1*		.500	**0**	
39	**0**	*32*	*1*		* 2		
19	**1**	*16*	*0*		1.00	**1**	
9	**1**	*8*	*1*		**0**		
4	**1**	*4*	*0*				
2	**0**	*2*	*1*				
1	**0**	*1*	*1*				
100 1110 1011					0.001		

Normalise the number: $1.00\ 1110\ 1011001 * 2^{10}$

1.17.1 Single Precision:

For a given number S = 0, E = 10, and M = 0 0 1 1 1 0 1 0 1 1 0 0 1
Bias for single precision format is = 127

$$E' = E + 127 = 10 + 127 = 137_{10}$$
$$= 1\ 0\ 0\ 0\ 1\ 0\ 0\ 1_2$$

1 bit 8 bits 23 bits.

Number in double precision format is given by.

Sign	Exponent	Mantissa
0	1000 1001	0011101011001….0

1.17.2 Double Precision:

Given number: S = 0, E = 10, and M = 0 0 1 1 1 0 1 0 1 1 0 0 1
Bias for double precision format is = 1023

$$E' = E + 1023 = 10 + 1023 = 1033_{10}$$
$$= 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1_2$$

1 bit 11 bits 23 bits

Number in double precision format is =

Sign	Exponent	Mantissa
0	10000001001	0011101011001….0

1.18 Bias: The sign bit is removed from being a separate entity and e is treated as unsigned number for this bias is added to e (exponential).

The actual exponent is biased by 127 to get e i.e. the actual value of the exponent is e − 127.

This gives the range: $2^{1-127} = 2^{-126}$ **to** $2^{254-127} = 2^{127}$.

1.18.1 Why Bias:

In IEEE 754 floating point numbers, the exponent is biased. Exponents have to be signed values in order to be able to represent both tiny and huge values, but two's complement, the usual representation for signed values, would make comparison harder.

To solve this problem the exponent is biased before being stored, by adjusting its value to put it within an **unsigned range** suitable for comparison.

* For a single-precision number (32 bits), an exponent in the range −126 .. +127 is biased by subtracting 127 to get a value in the range 1 .. 254 (0 and 255 are not used they have special meanings).

* For a double-precision number (64 bits), an exponent in the range −1022 .. +1023 is biased by subtracting 1023 to get a value in the range 1 .. 2046 (0 and 2047 have special meanings).

1.19 Error Detection codes:

An error detection code is a binary code that detects digital errors during transmission. The most common error detection code used is the parity bit.

1.19.1 Parity bit generation:

A parity bit is an extra bit included with a binary message to make the total number of 1's either odd or even. The message including the parity bit is transmitted to its destination.

i). The p (odd) bit is chosen in such a way as to make the sum of 1's in all bits (including parity bit) odd.

ii). The p (even) bit is chosen in such a way as to make the sum of 1's in all bits (including parity bit) even.

Message	P (odd)	P(even)
000	1	0
001	0	1
010	0	1
011	1	0
100	0	1
101	1	0
110	1	0
111	0	1

1.19.2 Parity checker:

At the receiving all the incoming bits are applied for parity checker that checks the proper parity adopted (odd or even). An error is detected if the checked parity does not conform to the adopted parity.

The parity method detects the presence of one, three or any odd number of errors. An even number of errors is not detected. Exclusive OR logic circuits are normally used for parity generation and checking.

1.20 Exercises:

1. List and explain different types of computers with examples. Also mention their merits and demerits.

2. Give a detailed account of how a digital computer is organized and explain its functions.

3. Draw and explain the block diagram of a digital computer.

4. Explain the use of program counter and instruction register.

5. Explain the key elements of various Bus Structures

6. Explain about various buses such as internal, external, backplane, I/O, system, address, data, synchronous and asynchronous.

7. What are the functions provided by system software?

8. List and explain different performance measures used to represent a computer system performance.

9. a). What are the measures to compare their speeds? b). How do we say one computer is faster than another? c). Explain about MIPS, MFLOPS rating of a processor.

10. Explain about multi processors and multi computers.

11.Give a detailed explanation of data representation in digital Computer with suitable examples.

12. Explain different techniques to represent signed integer numbers with an example.

13. Explain about sign magnitude and 2's complement approaches for representing the fixed point numbers. Explain why 2's complement approach is preferable.

14. Explain the representation of fixed point and floating point numbers with examples.

15. What is the standard representation of floating point numbers?

16. Explain IEEE 754 standard representation for example. b). Represent $(1259.125)_{10}$ in single precision and double precision IEEE 754 standards.

17. What is Bias? Why do they use biased exponent?

18. a) What is meant by odd parity and even parity? Explain them. b) Explain about Error Detection Codes in detail. c). Distinguish between error detection and correction codes.

19. Perform the arithmetic operations in binary using 2's complement representation for negative numbers. i). (+70)+(+80) and (-70)+(-80) ii). (+42) + (-13) and (-42)-(-13).

20. Represent the number (+ 46.5) as a floating point binary number with 24 bits. The normalized Fraction mantissa has 16 bits and exponent has 8 bits.

Multiple choice questions:

1. Co-processor in a CPU is used for
a). I/O operations b) Storage operations c). Math operations d). None
2. Which device gives you faster access……
a) Hard disk b) Compact disk c) SSD d) Floppy Disk
3. Supercomputers use the following operating system ……
a). Windows b). Ubuntu c). Fedora d). None
4. Cache memory belongs to which category ……
a) Rom b) Ram c) NVRAM d) None
5. In odd parity, if the parity bit is 1 the sum of the remaining high bits should be …..……
a) Even b) Odd c) any d) none

Fill in the blanks:

1. The main advantage of laptop computers is …….
2. Are the processors used in desktop and laptop computers are interchangeable …….
3. Which technology is used in enterprise servers……..
4. Which device can be used for both input and output …….
5. SIMs used in mobile belongs to which category of memory……
6. A program counter holds the ……..
7. The full form of MFLOPS is ……
8. The SPEC rating is used to ……..
9. The difference between Multiprocessors and Multicomputers is …….
10. What is the binary value of -5 ……
11. What is the 9's complement of 546700 ……..
12. Which standard is used for floating point operations in PCs ……

This page intentionally left blank.

2. REGISTER TRANSFER LANGUAGE AND MICRO OPERATIONS

2.1 Register Transfer language:

A register transfer language is a **symbolic notation used to describe the micro operations** transfers among registers. It is a convenient tool for describing the internal organisation of digital computers in concise and precise manner. It can also be used to facilitate the design process of digital systems.

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

MAR : Memory address register that holds the address of memory unit.
PC : Program counter
IR : Instruction register
PC(0-7) or PC(L) refers to the lower byte and PC(8-15) or PC(H) refers higher order byte.

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

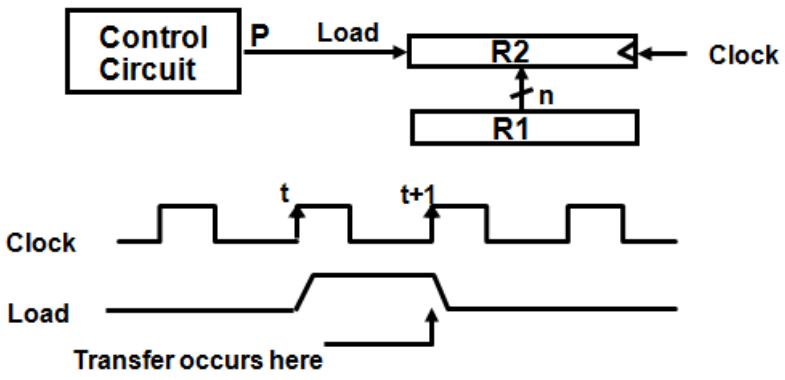
$$R2 \leftarrow R1$$

The above statement denotes a transfer of the content of register1 into register2. It designates the replacement of the content of R2 by the content of R1. There will be no change in R1 content.

2.1.3 Control function: A control function is a Boolean variable that is equal to 1 or 0. 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.

$$P: R2 \leftarrow R1$$

Fig: 1B1 : Transfer from R1 to R2 when P = 1.



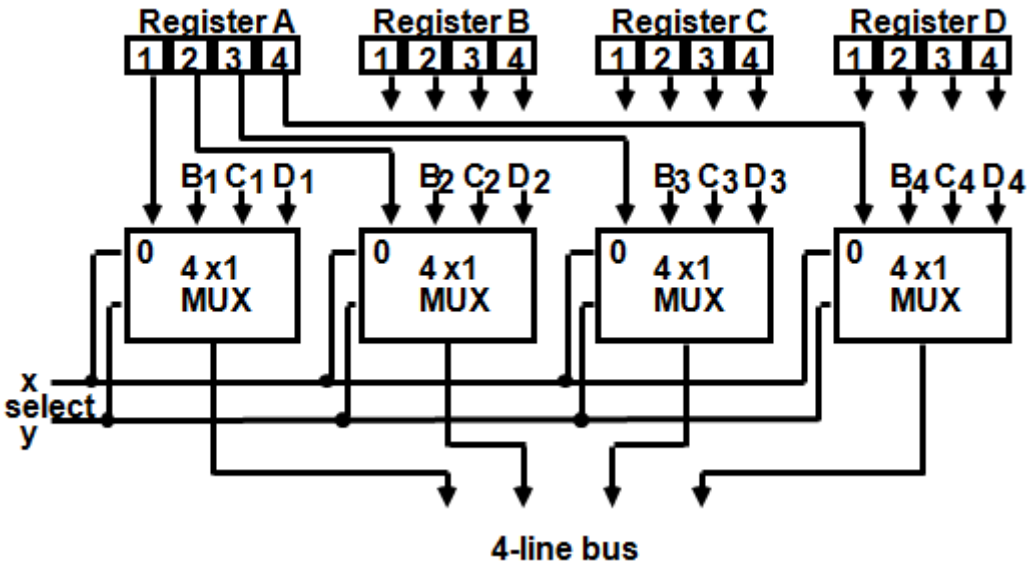
2.2 Bus and Memory Transfers:

2.2.1 Common Bus system: A digital computer has many registers and paths must be provided to transfer information from one register to another. For this, a common bus system is used.

A bus system structure consists of a set of common lines, one for each bit of a register. Separate control lines determine which register is selected by the bus during each particular register transfer. This is normally done by using the multiplexers.

The multiplexers select the source registers whose binary information is then placed on the bus. The below figure illustrates the construction of a bus system for four; 4-bit registers.

Fig 2.1 Bus system for four registers.



Bus Selection: The two selection lines x and y choose the four bits of one register and transfer them into the common bus. For example if the xy = 00 Register A is selected and its outputs are received by the 4 multiplexers.

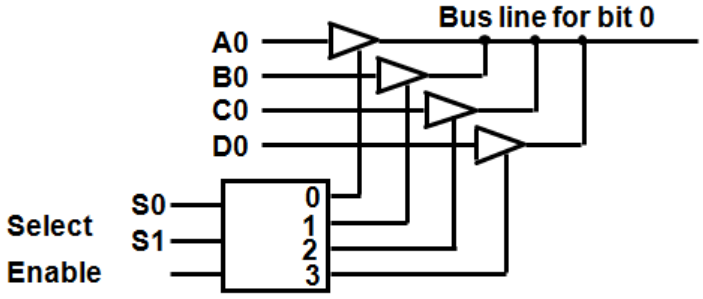
2.2.2 Three state bus buffers: A bus system can be constructed with three state gates instead of multiplexers. A three state gate is a digital circuit that exhibits three states. High, Low and *High impedance* state.

The high impedance state behaves like an open circuit which does not have any logic significance. Three state gates are most commonly used in the design of a bus system is the *buffer gate.*

Fig 2.2 Three state buffer

Bus system: The construction of three state bus system is demonstrated below. The outputs of 4 buffers are connected together to form a single bus line. The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line.

Fig 2.3 Bus line with three state buffers:



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.

Decoder: A decoder is used to ensure that no more than one control input is active at any given time.

If the enable input of the decoder is enabled (High) one of the three state buffers will be active, depending on the selection inputs (S_1S_0).

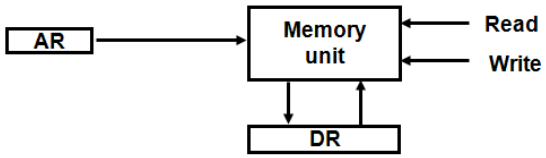
If the enable input is disabled (Low) all of its four outputs are 0, and the bus line is in high impedance state.

2.3 Memory Transfer:

The transfer of information from a memory word to the outside environment is called a read operation. The transfer of new information to be stored into the memory is called a write operation.

A memory word will be symbolized by the letter M. It is necessary to specify the address of M (in square brackets) when writing memory transfer operations as shown in Fig 2.4.

2.3.1 Memory Read: Consider a memory unit that receives the address from an address register (AR) and transfers to a data register (DR). The read operation can be stated as:



Memory *read* micro-op: DR ← M (DR ← M[AR])
Memory *write* micro-op: M ← DR (M[AR] ← DR)

Read: DR ← M[AR]

This causes a transfer of information into DR from the memory word M selected by the address in AR.

2.3.2 Memory Write: The write operation transfers the content of a data register to a memory word M selected by the address. Assume that the input data are in register R1 and the address is in AR. The write operation can be stated symbolically as :

Write: M[AR] ← R1
This causes a transfer of information from R1 into the memory word M selected by the address in AR.

2.4 Micro Operations:

A micro operation is an elementary operation performed with the data stored in registers. They can be classified into four categories:

2.4.1 Register transfer micro operations (transfer from one register to another register).
2.4.2 Arithmetic micro operations (performs arithmetic operations on numeric data stored in registers)
2.4.3 Logic micro operations (perform bit manipulation operations on numeric data stored in registers)
2.4.4 Shift micro operations (perform shift operations on data stored in registers)

Arithmetic micro operations: The basic arithmetic operations are addition, subtraction, increment, decrement and shift.

ADD micro operation: The below statement specifies an add micro operation

R3 ← R1 + R2

It states that the contents of register R1 are added to the contents of register R2 and the sum transferred to register R3.

Subtract micro operation: Subtraction is most often implemented through complementation and addition. Instead of using the minus operator, we can specify the subtraction by the following statement:

R3 ←R1 + R2* + 1 (* complement)

Adding R1 to the 2's complement of R2 is equivalent to R1 – R2.

Increment and Decrement: The Increment and Decrement micro operations are symbolized by plus one and minus one operations respectively. These micro operations are implemented with a combinational circuit.

Multiply and divide micro operations: Multiply and divide micro operations are implemented by means of a combinational circuit. In most computers, Multiplication operation is implemented with a sequence of add and shift micro operations and Division is implemented with a sequence of subtract and shift micro operations.

Arithmetic micro operations: * complement

Symbolic designation	Description
R3 ← R1 + R2	Contents of R1 plus R2 transferred to R3
R3 ← R1 - R2	Contents of R1 minus R2 transferred to R3
R2 ← R2*	Complements the contents of R2 (1's comp.)
R2 ← R2*+ 1	Complements the contents of R2 (2's comp.) negate
R3 ← R1 + R2* + 1	R1 plus the 2's complement of R2 (subtraction)
R1 ← R1 + 1	Increment the contents of R1 by one
R1 ← R1 – 1	Decrement the contents of R1 by one

2.5 Combinational circuits:

2.5.1 Half-Adder:

Half adder is a combinational circuit that performs the arithmetic addition of two bits.

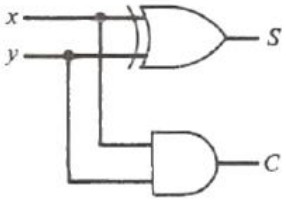
The input variables of a half-adder are called the **augend** and **addend** bits. The output variables are **sum (S) and carry (C)** as illustrated in the below Fig 2.5.

Ex: Let the input variables be *x* and *y*. Addition of two variables 1 + 1 is binary 10, here 0 is the value of sum (S) and 1 is the value of carry (C).

Note: In XOR gate if the all the inputs have same value then the output is zero (0), else one (1)**.**

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(a) Truth table



(b) Logic diagram

2.5.2 Full Adder:

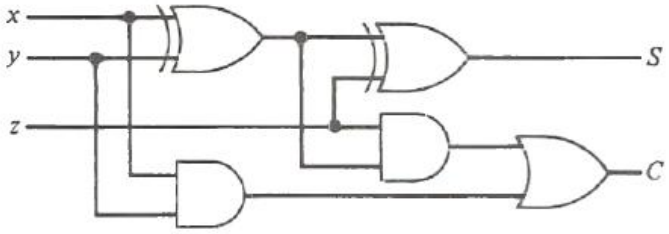
One that performs the addition of three bits (two significant bits and a previous carry) is called a full-adder.

A full-adder is a combinational circuit that forms the arithmetic sum of three input bits. In which two (x and y) are significant bits and the third one (z) is carry bit.

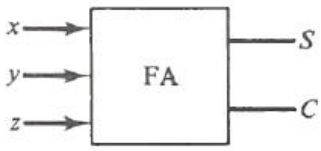
The two outputs S gives the value of the least significant bit of the sum. The binary variable C gives the output carry. Below truth table illustrates all the combinations.

Fig 2.6. Full adder circuit

(a) Logic diagram



(b) Block diagram



Truth Table for Full-Adder

Inputs			Outputs	
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

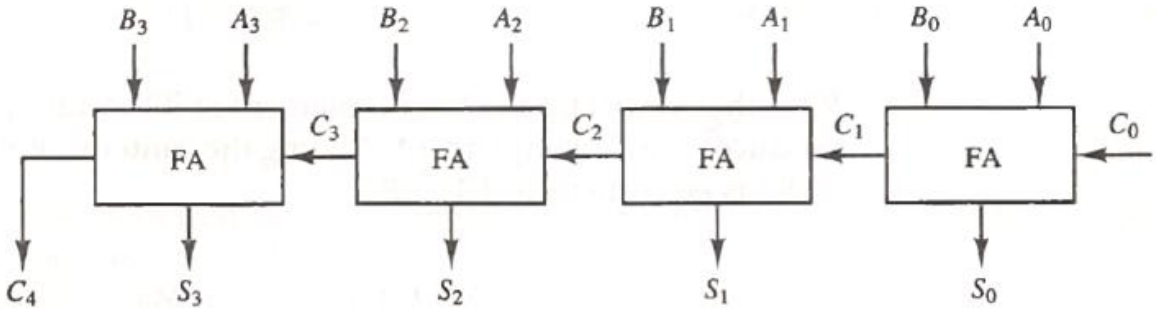
When all input bits are 0, the output is 0. The S output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1. The C output has a carry of 1 if two or three inputs are equal to 1.

2.5.3 Binary Adder:

The digital circuit that generates the arithmetic sum of two binary numbers of any length is called a binary adder.

The binary adder is constructed with **full-adder circuits connected in cascade**, with the output carry from one full-adder connected to the input carry of the next full-adder. Below fig. shows the interconnections of four full-adders (FA) to provide a 4-bit binary adder.

Fig 2.7. 4 bit binary adder.



The augend bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting the low-order bit. The carries are connected in a chain through the full-adders.

The input carry to the binary adder is C0 and the output carry is C4. The S outputs of the full-adders generate the required sum bits.

An n-bit binary adder requires n full-adders. The output carries from each full-adder is connected to the input carry of the next-high-order full-adder.

The n data bits for the A inputs come from one register (such as Rl), and the n data bits for the B inputs come from another register (such as R2).

R3 ← R2 + R1 or R2 ←R2 + R1

The sum can be transferred to a third register or to one of the source registers (Rl or R2), replacing its previous content.

2.5.4 Binary Adder and Subtractor:

The subtraction of binary numbers can be done most conveniently by means of compliments. For Example *A – B* can be done by taking the 2's complement of *B* and adding it to *A.*

The 2's complement can be obtained by taking the l's complement and adding one to the least significant pair of bits.

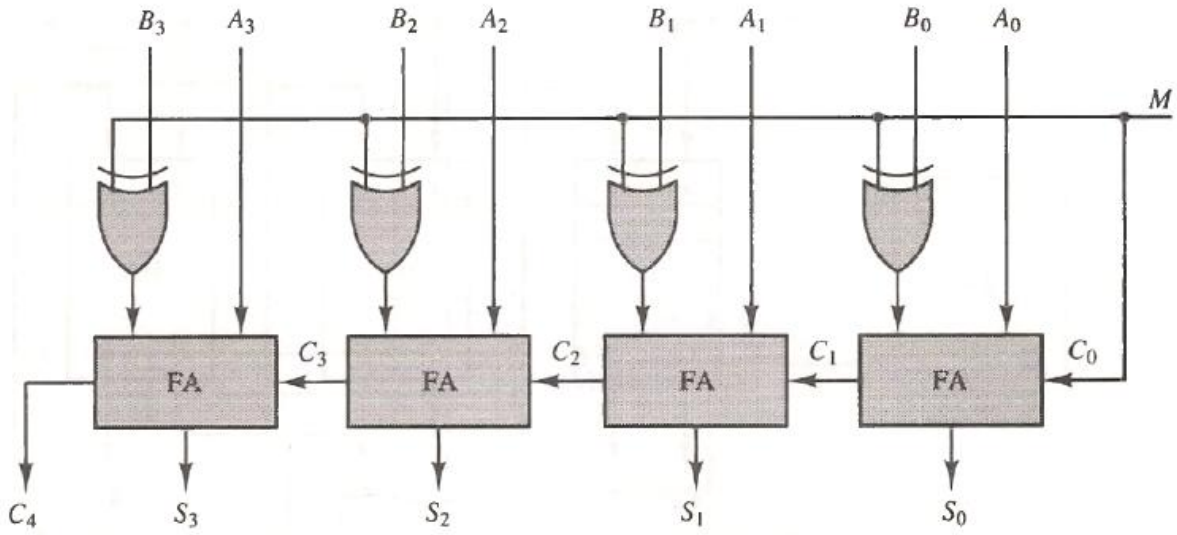
The l's complement can be implemented with inverters and a one can be added to the sum through the input carry.

The addition and subtraction operations can be combined into one common circuit by including an exclusive-OR gate with each full-adder.

The mode input M controls the operation.

When **M = 0 the circuit is an adder** and when
M = 1 the circuit becomes a subtractor.

Fig 2.8. 4 bit adder, subtractor.



Each exclusive OR gate receives input M to one of the inputs of B. When M = 0, we have $B \oplus 0 = B$. The full-adders receive the value of B, the input carry is 0, and the circuit **performs A plus B**.

When M = 1, we have $B \oplus 1 = B'$ and $C_0 = 1$. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation **A** plus the 2's complement of B.

For unsigned numbers, this gives A - B if A > B or the 2's complement of (B - A) if A < B. For signed numbers, the result is A - B provided that there is no overflow.

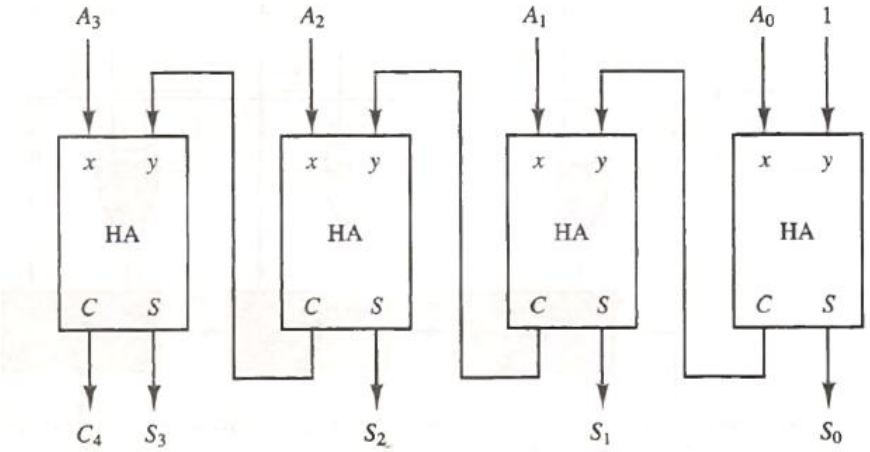
2.5.5 4 bit Binary Incrementer (using half adders):

The increment microoperation adds one to a number in a register. For example, if a 4-bit register has a binary value 0110, it will go to 0111 after it is incremented.

This microoperation is easily implemented with a binary counter (see Fig. 2-10). Every time the count enable is active, the clock pulse transition increments the content of the register by one.

Combinational circuit for 4 bit binary increment microoperation using half-adders:

Fig 2.9. 4 bit binary incrementer.



One of the inputs to the **least significant half-adder** (HA) is connected to logic-1 and the other input is connected to the least significant bit of the number to be incremented.

The output carry from one half-adder is connected to one of the inputs of the next-higher-order half-adder.

The circuit receives the four bits from A0 through A3 adds one to it, and generates the incremented output in S_0 through S_3.

The output carry C_4 will be 1 only after incrementing binary 1111. This also causes outputs S_0 through S_3 to go to 0.

The above circuit can be extended to an *n-bit* binary incrementer by extending the diagram to include *n* half-adders.

The least significant bit must have one input connected to logic-1. The other inputs receive the number to be incremented or the carry from the previous stage.

2.6 Four bit Arithmetic circuit:

The basic component of an arithmetic circuit is the parallel adder. By controlling the data inputs to the adder, it is possible to obtain different types of arithmetic operations.

Fig 2.10. Four bit arithmetic circuit.

The 4 bit arithmetic circuit consists of :

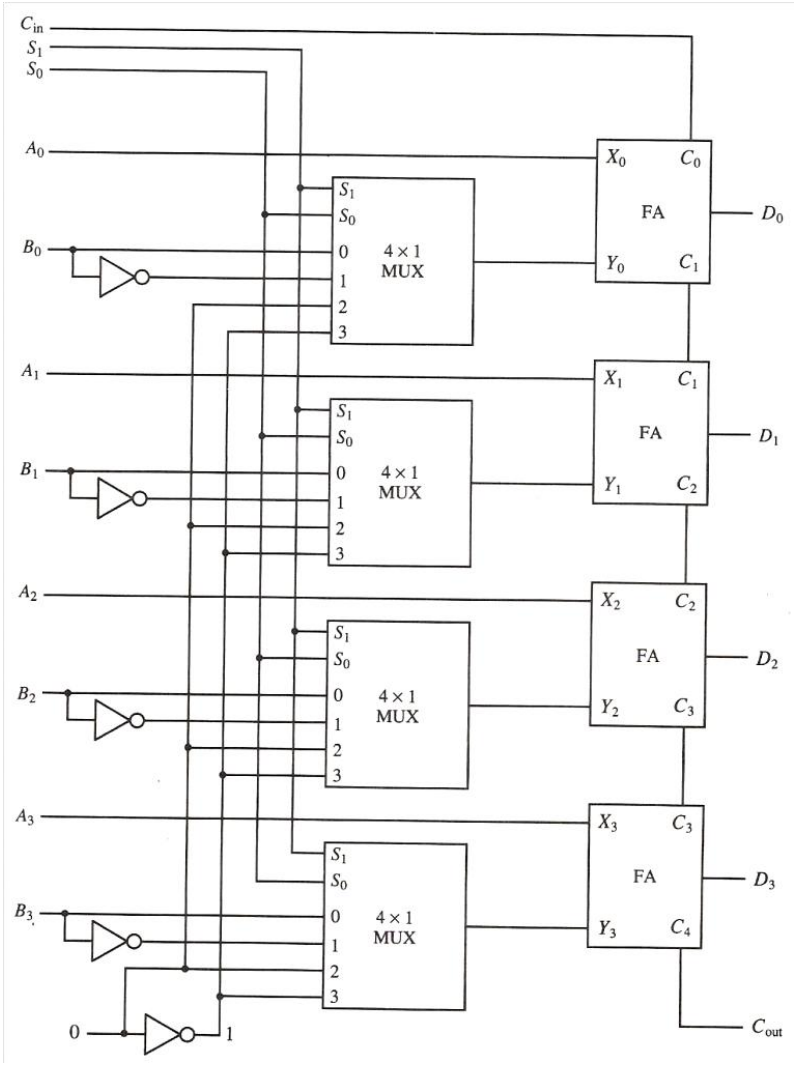
four full adder circuits,
four multiplexers,
four A inputs and
four B inputs and inverted B inputs as shown in the figure.

C_{in} is the carry in signal and

S_1 and S_0 are status signals, which are used to select the multiplexers.

D_0 to D_3 are data outputs and C_{out} is a carry out signal.

The output of the binary adder is calculated from the following arithmetic sum:



$D = A + Y + C_{in}$
4 bit binary circuit

Arithmetic circuit function table:

Select			Input	Output	Micro operations
			Y	$D = A + Y + C_{in}$	
S_i	S_0	C_{in}			
0	0	0	B	D = A +B	Add
0	0	1	B	D = A+B+ 1	Add with carry
0	1	0	B	D = A + B*	Subtract with borrow
0	1	1	B	D = A + B* + 1	Subtract
1	0	0	0	D = A	Transfer A
1	0	1	0	D = A + 1	Increment A
1	1	0	1	D = A – 1	Decrement A
1	1	1	1	D = A	Transfer A

2.7 Logic Micro operations:

Logical micro operations include OR, XOR, AND, NOT, Complement and the combinations of these micro operations. It consider each bit in the register as a separate binary variable. There are 16 different logic operations that can be performed with two binary variables. The 16 Boolean functions of two variables x and y are expressed in Truth tables and algebraic form below.

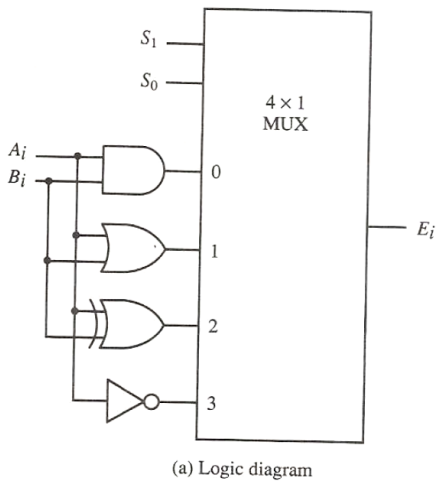
x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Sixteen micro operations:

Boolean functions	Micro operations	Name
$F_0 = 0$	F ←0	Clear
$F_1 = xy$	F ← A^B	AND
$F_2 = xy^*$	F ← A^B*	
$F_3 = x$	F ← A	Transfer A
$F_4 = x^* y$	F ← A* ^B	
$F_5 = y$	F ← B	Transfer B
$F_6 = x \oplus y$	F ← A⊕ B	Exclusive – OR
$F_7 = x + y$	F ← A v B	OR
$F_8 = (x + y)^*$	F ← (A v B)*	NOR
$F_9 = (x \oplus y)^*$	F ← (A ⊕ B)*	Exclusive – NOR
$F_{10} = y^*$	F ← B*	Complement B
$F_{11} = x + y^*$	F ← A v B*	
$F_{12} = x^*$	F ← A*	Complement A
$F_{13} = x^* + y$	F ← A* v B	
$F_{14} = (xy)^*$	F ←(A ^ B)*	NAND
$F_{15} = 1$	F ← all 1's	Set to all 1's

Fig 2.11. Hardware implementation of Logical micro operations:

One stage of logic circuit.



(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

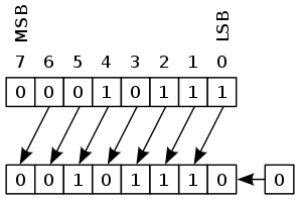
(b) Function table

2.8 Shift Micro operations:

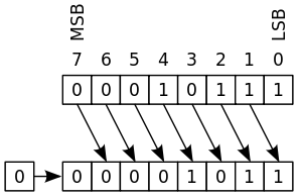
Shift micro operations are used for serial transfer of data. The contents of the register can be shifted to left or right. The information transferred through the serial input determines the type of shift. There are three types of shifts. Logical, circular and arithmetic.

i). Logical Shift:

A logical shift is one that transfers 0 through the serial input.
The shl (shift left) and shr (shift right) in the below fig. demonstrates this.



$R_1 \leftarrow$ shl R_1 (1 bit shift to the left of the content of register R1)



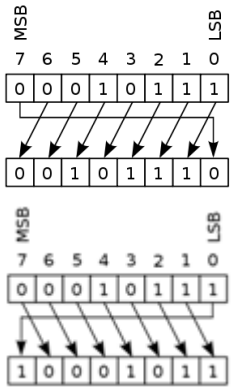
$R_2 \leftarrow$ shr R_2 (1 bit shift to the right of the content of register R1)

Note: both the register symbol must be same on both sides of the arrow.

ii). Circular (or rotate) Shift:

The circular shift also known as a rotate operation circulates the bits of the register around the two ends without loss of information. This is accomplished by connecting the serial output of the shift register to its serial input.

Symbolic designation	Description.
R ←shl R	Shift left register R
R ←shr R	Shift right register R
R ←cil R	Circular Shift left register R
R ←cir R	Circular Shift right register R
R ←ashl R	Arithmetic Shift left register R
R ←ashr R	Arithmetic Shift right register R



iii). Arithmetic shift:

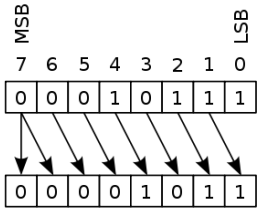
An arithmetic shift is a micro operation that shifts a signed binary number to the left or right. An arithmetic shift-left multiplies a signed binary number by 2. An arithmetic shift right divides the number by 2.

It must leave the sign bit unchanged because the sign of the number remains the same.

Ex: 1100 (12)

Arithmetic shift left **1**100 (12), **1**1000 (24), **1**10000 (48)
Multiply by 2

Arithmetic shift right 1100 (12), 0110 (6), 0011 (3)
Divide by 2



2.9 Hardware implementation of Shift micro operations:

A combinational circuit shifter can be constructed with multiplexers as shown in the Fig 2.12.

The 4-bit shifter has four data inputs, A0 through A3, and four data outputs, H0 through H3.

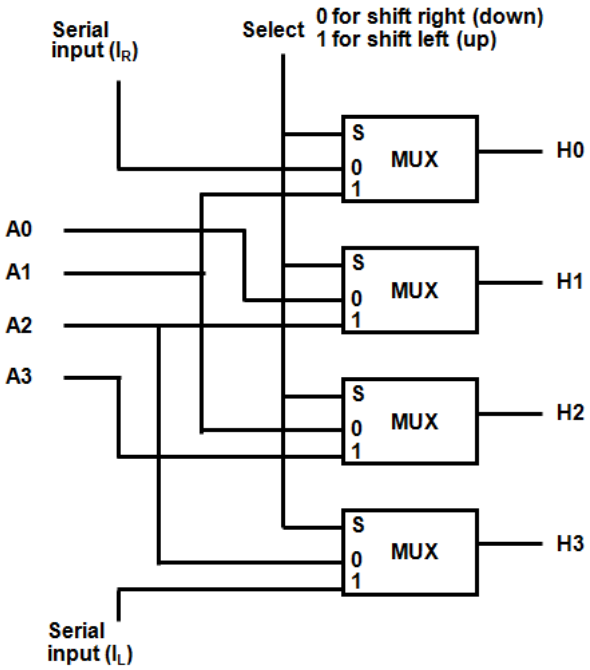
There are two serial inputs, one for shift left (I_L) and the other for shift right (I_L).

When the selection input S = 0, the input data are shifted right (down in the diagram).

When S = 1, the input data are shifted left (up in the diagram).

A shifter with n data inputs and outputs requires n multiplexers.

The two serial inputs can be controlled by another multiplexer to provide the three possible types of shifts.



Function Table				
Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

2.10 Arithmetic Logic Shift Unit:

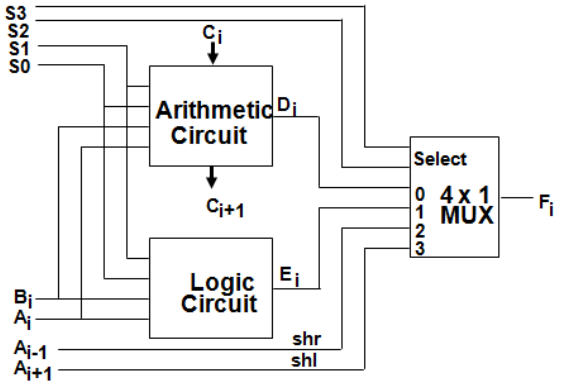
The below arithmetic logic shift circuits are combined into one stage circuit as shown in the below Figure.

The subscript i designates a typical stage. Inputs Ai and Bi are applied to both the arithmetic and logic units. A particular microoperation is selected with inputs S1 and S0.

A 4 x 1 multiplexer at the output chooses between an arithmetic output, and a logic output. The data in the multiplexer are selected with inputs S3 and S2.

The other two data inputs to the multiplexer receive inputs A_{i-1} for the shift-right operation and A_{i+1} for the shift-left operation.

Note that the diagram shows just one typical stage. The output carry C_{i+1} of a given arithmetic stage must be connected to the input carry C_i of the next stage in sequence.



The input carry to the first stage is the input carry C_{in}, which provides a selection variable for the arithmetic operations.

The one stage circuit provides eight arithmetic operation, four logic operations, and two shift operations.

Each operation is selected with the five variables S3, S2, S1, S0, and C_{in}. The input carry C_{in} is used for selecting an arithmetic operation only.

Below table lists the 14 operations of the ALU. The first eight are arithmetic operations and are selected with S3 S2 = 00.

The next four are logic operations and are selected with S3 S2 = 01.

The input carry has no effect during the logic operations and is marked with don't-care x 's.

S3	S2	S1	S0	Cin	Operation	Function
0	0	0	0	0	F = A	Transfer A
0	0	0	0	1	F = A + 1	Increment A
0	0	0	1	0	F = A + B	Addition
0	0	0	1	1	F = A + B + 1	Add with carry
0	0	1	0	0	F = A + B'	Subtract with borrow
0	0	1	0	1	F = A + B'+ 1	Subtraction
0	0	1	1	0	F = A - 1	Decrement A
0	0	1	1	1	F = A	TransferA
0	1	0	0	X	F = A ∧ B	AND
0	1	0	1	X	F = A ∨ B	OR
0	1	1	0	X	F = A ⊕ B	XOR
0	1	1	1	X	F = A'	Complement A
1	0	X	X	X	F = shr A	Shift right A into F
1	1	X	X	X	F = shl A	Shift left A into F

The last two operations are shift operations and are selected with S3 S2 = 1 0 and 11.

The other three selection inputs have no effect on the shift Function table for Arithmetic Logic Shift Unit.

2.11 Exercises:

1). Define Register Transfer Language? Explain register transfer process and control function with suitable examples.
2). Explain the implementation of the common bus using multiplexers.
3). Construct a Common Bus System with Multiplexers for four Registers.
4). What is the use of buffers. Explain about tri-state buffers. Explain about high impedance state.
5). Discuss about micro operations.
6). List and explain arithmetic micro operations.
7). Explain the organization of 4-bit binary adder and 4 bit binary incrementer. Illustrate the functioning of these with examples.
8). Explain the 4-bit arithmetic circuit using multiplexer. Or Explain the organization of 4 bit arithmetic circuit with a schematic. Illustrate its functioning with an example.
9). List and explain logic micro operations.
10). List and explain shift micro operations.

Multiple choice questions:

1. PC(8-15) refers to ……. Byte.
a). Lower order b) middle order c). Higher order d). None
2. R2 ←R1 indicates …….. data from R1 to R2
a) copy b) move c) compare d) none
3. An n bit arithmetic circuit requires …….. number of multiplexers
a). n-1 b). n c). n+1 d). n^2
4. Arithmetic micro operations are performed in ……
a) Cache b) Registers c) Ram d) Rom
5. If you shift R ←cil R on 0001 0111 the obtained value is …..……
a) 11 b) 23 c) 139 d) 146

Fill in the blanks:

1. A CPU register is used to …….
2. …… register holds the address of memory unit.
3. Difference between an arithmetic shift and normal shift is …….
4. Buffer is a device used to …….
5. …….. and …… gates are needed to construct a half adder circuit
6. A binary adder is defined as……..
7. The high impedance state is defined as ……..

3. BASIC COMPUTER ORGANIZATION AND DESIGN

3.1 Instruction Codes:

An instruction code is a group of bits that instruct the computer to perform a specific operation. It is usually divided into two parts.

1. The first part specifies the operation to be performed (add, multiply, shift, complement…).

2. The second part specifies an address of the registers or the memory words.

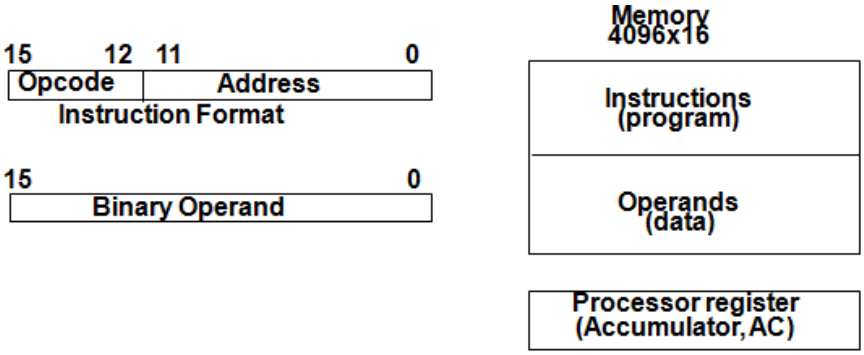
An instruction code must, therefore specify not only the operation but also the register or memory words where the operands are to be found as well as the register or memory word where the result is to be stored.

3.2 Stored program organisation:

Instructions are stored in one section of memory and data in another section of memory. For memory unit with 4096 words, we need 12 bits to specify an address since 2^{12}=4096.

For Example: If we store each instruction in a 16 bit memory word. Out of 16, 12 bits are used to specify the address of an operand and the remaining four are available for operation code (opcode) to specify one out of (2^4) 16 possible operations.

The control reads a 16 bit instruction from the program memory. It uses the 12 bit address part of the instruction to read a 16 bit operand from the data portion of memory. It then executes the operation specified by the operation code.



3.3 Immediate, Direct and Indirect address:

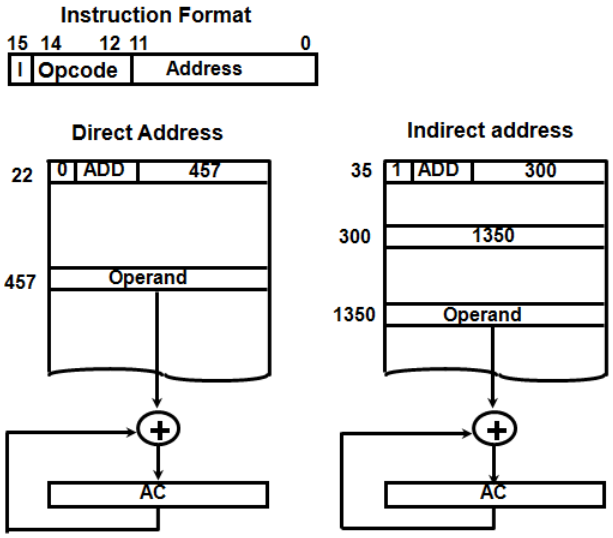
i). Immediate operand (immediate instruction): If we use the address bits of an instruction code as actual operand, not as an address then the instruction is said to have an immediate operand.

ii). Direct address: If the second part specifies the address of an operand, the instruction is said to have a direct address.

iii). Indirect address: If the second part designates an address of a memory word in which the address of the operand is found then it is referred as an indirect address. The15th bit of the instruction code is used to distinguish between a direct and indirect address.

Effective Address: Effective address is the address of the operand. Thus the effective address in the instruction of fig (a) is 457 and in the instruction of fig (b) is 1350.

Fig 3.1. Demonstration of direct and indirect address:



3.4 Computer registers:

Computer instructions are stored in consecutive memory locations and are executed sequentially one at a time. For this purpose it needs several registers to handle data, address, counter and so on. The below table lists several registers for the basic computer.

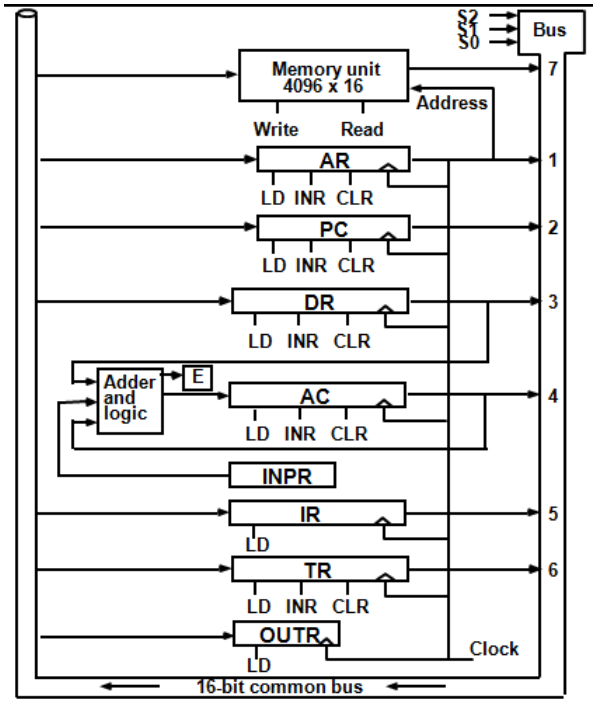
Register symbol	No. of bits	Register name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Process register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

* The data register (DR)holds the operand read from memory.
* The accumulator (AC) is a general purpose register.
* The instruction register (IR) holds the instruction read from memory.
* The temporary register (TR) is used for holding temporary data during the processing.
* The memory address register (AR) holds the address for memory.
* The program counter (PC) holds the address of the next instruction to be read from memory after the current execution is executed.
* The input register (INPR) receives an 8 bit character from an input device.
* The output register (OUTR) holds an 8 bit character for an output device.

3.5 Common Bus System: The connection of the registers and memory of the basic computer to a common bus system:

Fig 3.2. Basic computer registers connected to a common bus.

Description:



* The registers and memory unit are connected to 16 bit common bus as shown in the figure.

* The output is selected by the selection variables S_2,S_1,S_0. When $S_2S_1S_0$ = 011 (3): The 16 bit outputs of DR are placed on the bus lines.

* The memory receives the content when its write input is activated and when the read is activated it ($S_2S_1S_0$=111) sends the data on to the bus.

* When the register LD (load) input is enabled register receives the data from the bus. DR, AC, IR and TR are 16 registers. AR and PC have 12 bits each since they hold a memory address.

* The input register INPR and the output register OUTR have 8 bits each. The INPR receives a character from AC and delivers it to an output device. There is no transfer from OUTR to any other registers. Five registers have three control inputs: LD (load), INR (increment) and CLR (clear).

* Memory bus is connected to AR. Therefore AR must always be used to specify a memory address. The 16 inputs of AC come from an adder and logic circuit. This circuit has 3 sets of inputs.

* One set of 16 bits input comes from the output of AC. They are used to implement register micro operations such as complement AC and shift AC.

* Second set 16 bits comes from data register DR. The inputs from DR and AC are used for arithmetic and logic micro operations, such as add DR to AC or AND DR to AC. The result is of the addition is transferred to AC and the end carry out of the addition is transferred to flip flop E(extended AC bit). Third set of 8 bit inputs come from the input register INPR.

Note: 1. The content of any register can be applied to the bus and an operation can be performed in the adder and logic circuit during the same clock cycle.

2. The clock transition at the end of the cycle transfers the content of the bus into the designated destination register and the output of the adder and logic circuit into AC.

For example: DR ← AC and AC → DR micro operations can be executed at the same time. This can be done by placing the content of AC on the bus (with $S_2S_1S_0$=100), enabling the LD (load) input of DR, transferring the content of DR through the adder and logic circuit into AC and enabling the LD (load) input of AC, all during the same clock cycle. The two transfers occur upon the arrival of the clock pulse transition at the end of the clock cycle.

3.6 Computer instructions:

The basic computer has three instruction code formats, each format has 16 bits. The opcode part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered.

The type of instruction is recognized by the computer basing on the four bits (12 to 15) positions of the instruction.

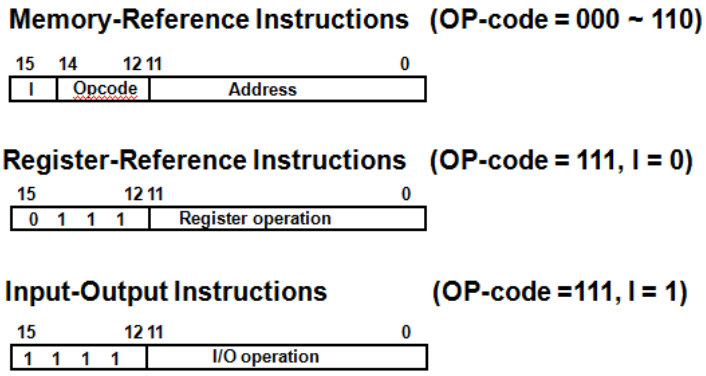
If the bits 12 to 14 are not 111 then the instruction is memory referenced. If the bits are 111 then the instruction can be register reference or IO reference.

Then it checks for the 15 bit, If the 15 bit (I=0) is zero it is register reference instruction, if the 15 bit (I=1) is one then it is Input output instruction.

a). Memory reference instruction:

A memory reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I (15^{th} bit) .

I = 0 for direct address,
I = 1 for indirect address.



b). Register reference instruction:

The register reference instructions are recognised by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction.

c). Input – Output instruction:

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 total number of instructions chosen for the basic computer is equal to 25.

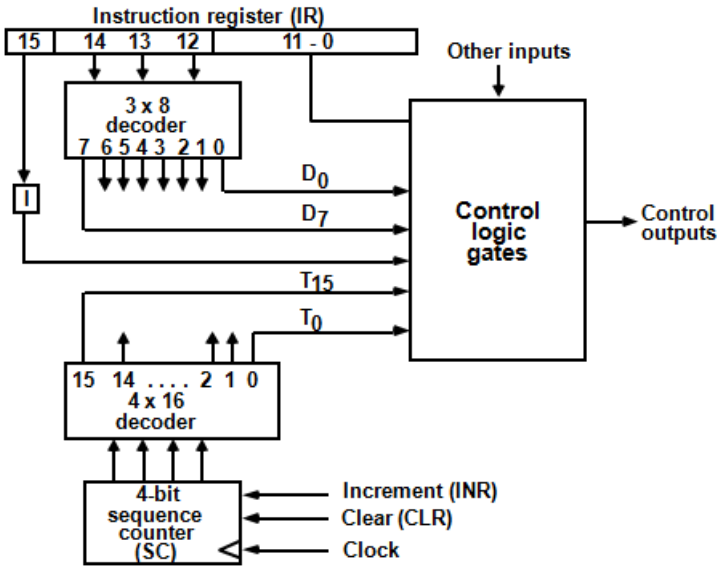
3.7 Control Organisation:

There are two types of control organisation. Hardwired control and micro-programmed control.

i). Hardwired control has the advantage that it can be optimized to produce a fast mode of operation. But any design modification needed to physical change in the circuit.

ii). In micro programmed organisation the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of micro operations. In the micro programmed control any required changes can be done by updating the micro program in control memory.

Fig 3.3. Control unit of basic computer:



The control unit consists of two decoders, a sequence counter, and a number of control logic gates.

An instruction read from memory is placed in the instruction register (IR).

The instruction register is divided into three parts: the I bit, the operation code, and bits 0 through 11.

The operation code in bits 12 through 14 are decoded with a 3 x 8 decoder.

The eight outputs of the decoder are designated by the symbols D0 through D7.

The subscripted decimal number is equivalent to the binary value of the corresponding operation code.

Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I. Bits 0 through 11 are applied to the control logic gates.

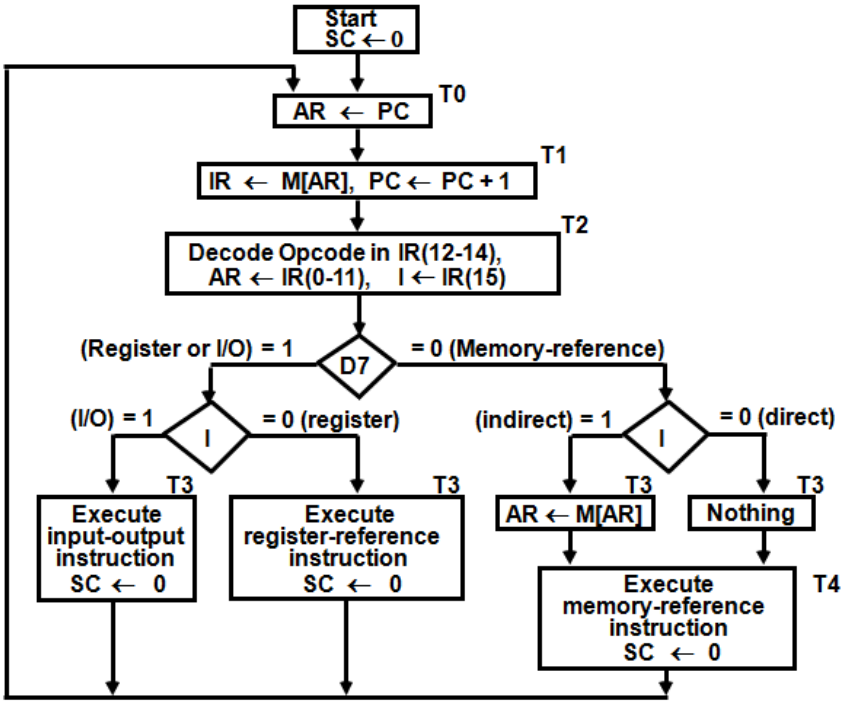
The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T0 through T15.

3.8 Instruction Cycle:

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

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

Fig 3.4 Instruction cycle.



3.8.1 Fetch and Decode:

1. Initially the register PC is loaded with the address of the first instruction in the program.

2. The sequence counter SC is cleared to 0, providing a decoded timing signal T_0.

3. After each clock cycle SC is incremented and timing signals go through T_0,T_1,T_2 and so on..

T_0: AR ← PC

T_0: since only AR is connected to the address inputs of memory, it is necessary to transfer the address from PC to AR during T_0.

T_1 : IR ← M[AR], PC ← PC +1

T_1: The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing signal T_1. PC is incremented by one to prepare it for the address of the next instruction in the program.

Note: Only three bits of the instruction are used for the operation code, means the maximum of 8 distinct operations. The decoder outputs are 0000 0001 (D_0), to 1000 0000 (D_7).

T_2: D_0,….D_7 ← decode IR(12-14), AR ← IR(0-11), I ← IR(15)
T_2: The operation code in IR is decoded,

i). The indirect bit is transferred to flip flop I, and
ii). the address part of the instruction is transferred to AR.
iii).The SC is incremented after each clock pulse to produce the sequence T_0, T_1 and T_2.

3.8.2 Determine the Type of Instruction:

During the time T_3, the control unit determines the type of instruction that was just read from memory.

a). The decoder output D_7 is equal to 1 if the operation code is 111, then the instruction must be a register reference or input-output type.

b). The decoder output D_7 is equal to 0 if the operation code is 000 through 110, which is a memory reference instruction.

c). Control then inspects the first bit (15^{th} bit which is in I flip-flop). If $D_7 = 0$ and I = 1, we have a memory reference instruction with an indirect address. It is then necessary to read the effective address from memory.

$$AR \leftarrow M[AR]$$

AR holds the address part of the instruction. The word at the address given by AR is read from memory and placed on the common bus. The LD input of AR is then enabled to receive the indirect address that resided in the 12 least significant bits of the memory word.

T_3: The three instruction types are subdivided into four separate paths.

D'_7IT_3 : $AR \leftarrow M[AR]$
$D'_7I'T_3$: Nothing (Since I=0 means the effective address is in AR)
$D_7I'T_3$: Execute a register reference instruction
D_7IT_3 : Execute an input-output instruction

When a memory reference instruction with I=0 is encountered, it is not necessary to do anything since the effective address is already in AR.

T_4: The sequence counter SC must be incremented when $D'_7T_3=1$, so that the execution of the memory reference instruction can be continued with timing variable T_4.

A register reference or input-output instruction can be executed, SC is cleared to 0 and control returns to the fetch phase with $T_0 = 1$.

Note that the SC is either incremented or cleared to 0 with every positive clock transition.

3.8.3 Decoded instructions:

Register reference : $D_7 I' T_3$
I/O Reference : $D_7 I T_3$
Memory Reference (Direct) : $D_7 I' T_3$
Memory Reference (Indirect) : $D_7 I T_3$

3.9 Register Reference instructions:

Register reference instructions are recognized by the control when $D_7 = 1$ and I = 0. These instructions use bits 0 through 11 of the instruction code to specify one of 12 instructions.

These 12 bits are available in IR(0-11). They were also transferred to AR during time T_2.

$D_7I'T_3 = r$ common to all register reference instructions.
$IR(i) = B_i$ (bit in IR(0-11) that specifies the operation)

	r:	SC ← 0	Clear SC
CLA	rB_{11}:	AC ← 0	Clear AC
CLE	rB_{10}:	E ← 0	Clear E
CMA	rB_9:	AC ← AC'	Complement AC
CME	rB_8:	E ← E'	Complement E
CIR	rB_7:	AC ← SHR AC, AC(15) ← E, E← AC(0)	Circulate right
CIL	rB_6:	AC ← SHR AC, AC(0) ← E, E← AC(15)	Circulate left
INC	rB_5:	AC ← AC + 1	Increment AC
SPA	rB_4:	If (AC(15) = 0) then (PC ← PC +1)	Skip if positive
SNA	rB_3:	If (AC(15) = 1) then (PC ← PC +1)	Skip if negative
SZA	rB_2:	If (AC= 0) then (PC ← PC +1)	Skip if AC zero
SZE	rB_1:	If (E = 0) then (PC ← PC +1)	Skip if E zero
HLT	rB_0:	S ← (S is a start stop flip flop)	Halt computer

3.10 Memory reference Instructions:

The seven memory reference instructions are given below. The effective address of the instruction is in the address register AR and was placed there during timing signal T_2 when I = 0 or during timing signal T_3 when I=1. The execution of the memory reference instructions starts with timing signal T_4.

Symbol	Operation decoder	Symbolic description
AND	D_0	AC ← AC ^ M[AR]
ADD	D_1	AC ← AC + M[AR], E ← Cout
LDA	D_2	AC ← M[AR]
STA	D_3	M[AR] ← AC
BUN	D_4	PC ← AR
BSA	D_5	M[AR] ← PC, PC ← AR + 1
ISZ	D_6	M[AR] ← M[AR] + 1
		If M[AR] + 1 = 0 then PC ← PC +1

i). AND to AC:

This is the instruction that performs the AND logic operation on pairs of bits in AC and the memory word specified by the effective address. The result of the operation is transferred to AC. The micro operations that execute this instruction are:

D_0T_4: DR ← M[AR]
D_0T_5: AC ←AC ^ DR, SC←0

ii). ADD to AC:

This instruction adds the content of the memory word specified by the effective address to the value of AC. The sum is transferred into AC and the output carry C out is transferred to the E (extended accumulator) flip-flop.

D_1T_4: DR ← M[AR]
D_1T_5: AC ←AC + DR, E ← C_{out}, SC←0

iii). LDA: load to AC

This instruction transfers the memory word specified by the effective address to AC.

D_2T_4: DR ← M[AR]
D_2T_5: AC ← DR, SC←0

iv). STA: Store AC

This instruction stores the content of AC into the memory word specified by the effective address.

D_3T_4: M[AR] ← AC , SC ← 0

v). BUN: Branch Unconditionally

This instruction transfers the program to the instruction specified by the effective address. The PC holds the address of the instruction to be read from memory in the next instruction cycle.

D_4T_4: PC ← AR, SC ← 0

vi). BSA: Branch and Save Return address

This instruction is useful for branching to a portion of the program called a subroutine or procedure. When executed the BSA instruction stores the address of the next instruction in sequence into a memory location specified by the effective address. The effective address plus one is then transferred to PC to serve as the address of the first instruction in the subroutine.

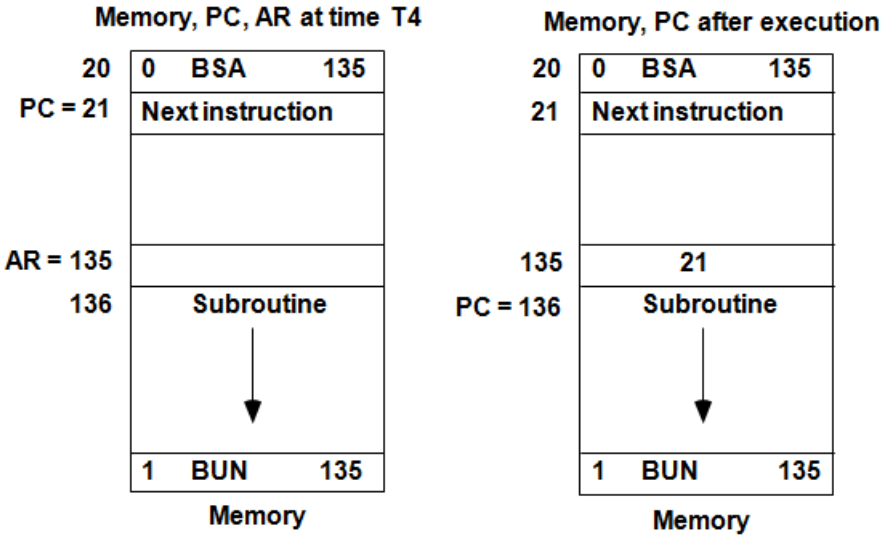
M[AR] ← PC, PC ← AR +1

Example of BSA instruction execution:

Ex: M[135] ← 21, PC ← 135 +1 = 136.

AR holds the effective address 135, PC contains 21, which is the address of the next instruction in the program (referred to as the return address).

Fig 3.5. BSA instruction execution**.**



a). Memory, PC and AR at time T_4 b). Memory and PC after execution

It is not possible to perform the operation of the BSA instruction in one clock cycle when we use the bus system. The BSA instruction must be executed with a sequence of two micro operations.

D_5T_4: M[AR] ← PC, AR ← AR + 1
D_5T_5: PC ← AR, SC ← 0

vii). ISZ: Increment and skip if Zero

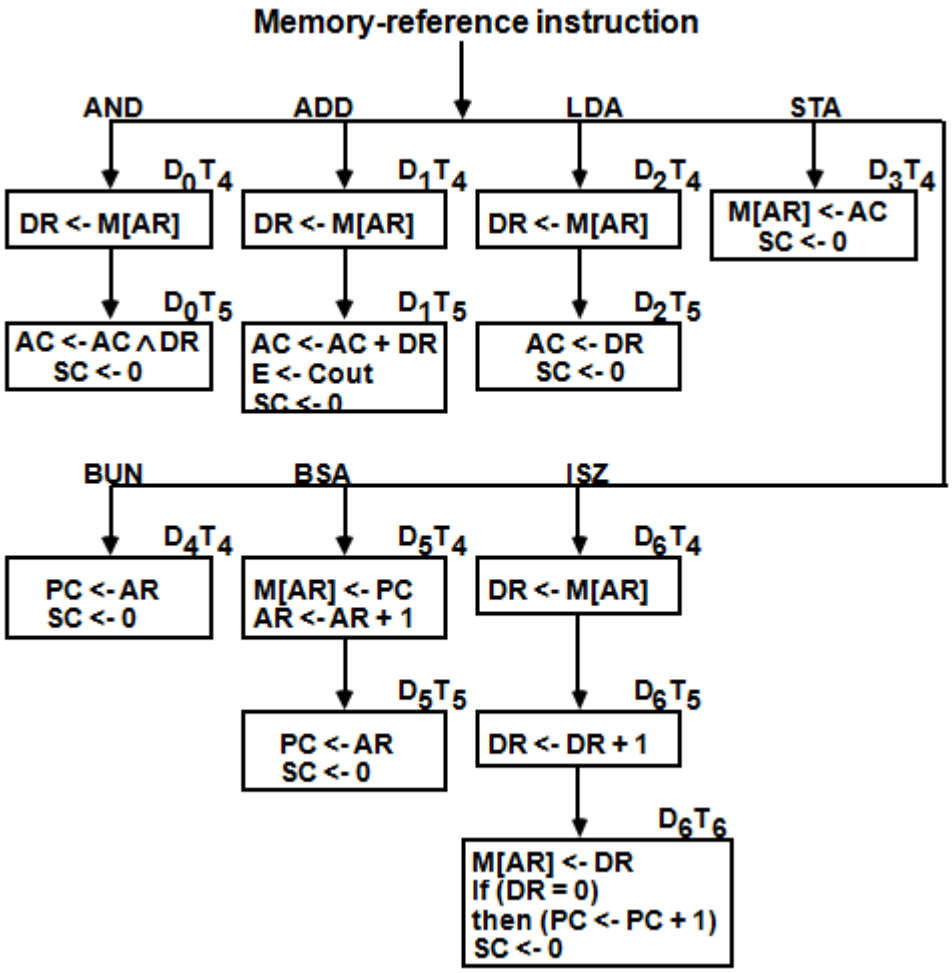
This instruction increments the word specified by the effective address and if the incremented value is equal to 0, PC is incremented by 1.

Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR and store the word back into memory.

This is done with the following sequence of micro operations:

D_6T_4: DR ← M[AR]
D_6T_5: DR ← DR + 1
D_6T_6: M[AR] ← DR, if (DR=0) then (PC ←PC + 1), SC ← 0

Fig 3.6. Flowchart For Memory Reference Instructions**:**



3.11 Input – Output Configuration:

The terminal sends and receives information which has eight bits of an alphanumeric code. Register INPR is used for input and OUTR is used for output. These two registers communicate with a communication interface serially and with the AC in parallel as illustrated in figure.

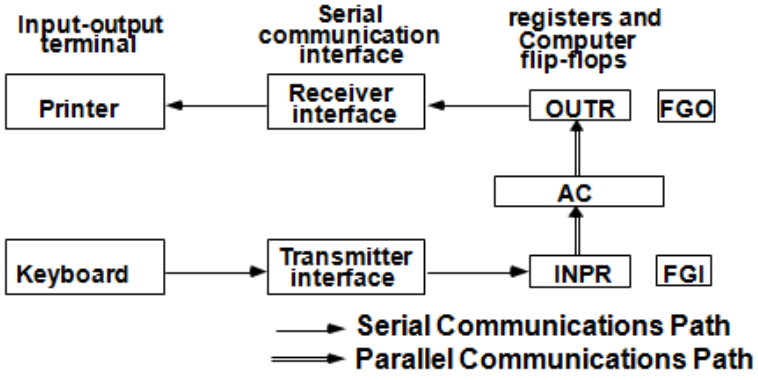
Initially the input flag (FGI) is 0.

When a key is struck in the keyboard FGI is set to 1.

Initially the output flag (FGO) is 1.

The computer checks the flag bit, if it is 1 the information from AC is transferred to OUTR the FGO is cleared to 0.

Fig 3.7. I/O configuration.



3.11.1 Input Output Instructions:

I/O instructions have an operation code 1111 and are recognized by the control when $D_7 = 1$ and I = 1.The remaining bits of the instruction specify the particular operation. These instructions are executed with the clock transition associated with timing signal T_3.

D_7IT_3 = p (common to all input-output instructions) IR(i) = Bi [bit in IR(6-11) that specifies the instruction)			
	P:	SC ← 0	Clear SC
INP	pB_{11}:	AC (0-7) ← INPR, FGI ← 0	Input Character
OUT	pB_{10}:	OUTR ←AC (0-7),FGI ← 0	Output Character
SKI	pB_9:	If (FGI = 1) then (PC ←PC + 1)	Skip on input flag
SKO	pB_8:	If (FGO = 1) then (PC ←PC + 1)	Skip on output flag
ION	pB_7:	IEN ← 1	Interrupt enable on
IOF	pB_6:	IEN ← 0	Interrupt enable off

3.12 Program Interrupt:

The process of communication seen above is referred to as programmed control transfer. The computer keeps checking the flag bit, and when it finds it set, it initiates and information transfer.

Here the computer time is wasted drastically due to continuously checking the flag bit even though it is not set on. The alternative way is to use interrupt cycle.

3.12.1 External (Device) Interrupt:

An alternative to the programmed controlled procedure is to let the external device inform the computer when it (the data) is ready to transfer.

Here the computer does not check the flags instead when a flag is set the computer is momentarily interrupted from proceeding with the current program and is informed to the fact that a flag has been set.

The computer deviates momentarily from what it is doing to take care of the input or output transfer.

Once it is over it returns back to the current program to continue, what it is doing before the interrupt.

For this The interrupt enable flip flop IEN is used, to set (1) ION instruction is used to clear (0) IOF instruction is used.

3.13 Flow chart for Interrupt Cycle:

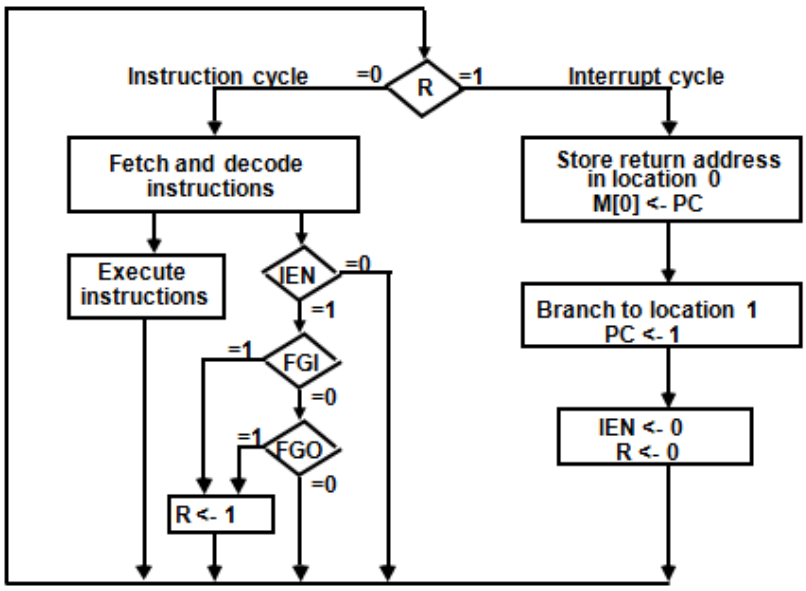
When the interrupt flip flop R = 0 the computer goes to instruction cycle. During the instruction cycle, IEN is checked. If it is 0, the control continues with the next cycle.

Fig 3.8 Interrupt cycle

If IEN is 1, control checks the flag bits.

If both flags are 0, it indicates input output registers are not ready. So the control continues with the next instruction cycle.

If either flags are set to 1, R is set to 1. At the end of the execute phase, control checks the value of R if it is 1, it goes to an interrupt cycle instead of instruction cycle.

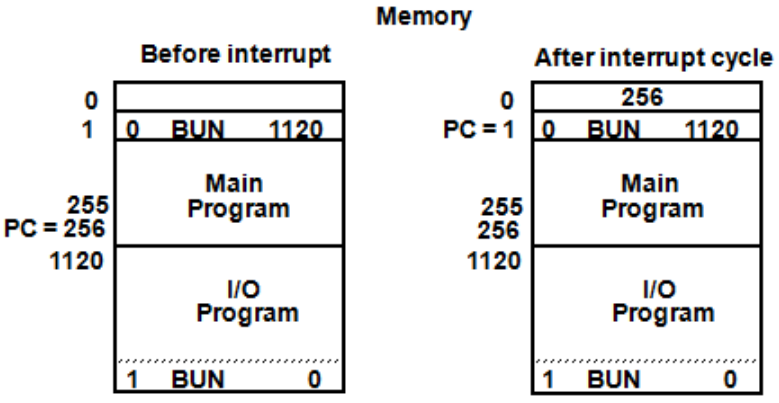


The interrupt cycle is a hardware implementation of a branch and save return address operation. The return address available in PC is stored in a specific location (CPU register, memory stack..) where it can found later when the program returns to the instruction at which it was interrupted.

Example:

Suppose an interrupt occurs (R=1) while the control is executing the instruction at address 255 and the next (return) address holds by PC is 256.

Now before serving the interrupt the content of PC (256) is stored in memory location 0, PC is set to 1, and R is cleared to 0.



The instruction is read from memory at address 1. The branch instruction at address 1 causes the program to transfer to the input-output service program at address 1120.

The Programmer has already placed an input-output service program in memory starting from address 1120 and a BUN 1120 instruction at address 1.

This program checks the flags, determines which flag is set, and then transfers the required input or output information. Once this is done, the program returns to the location where it was interrupted means PC (256).

3.14 Interrupt Cycle:

The interrupt cycle is initiated if the interrupt flip-flop R is equal to 1. This flip-flop is set to 1 if IEN = 1 and either FGI or FGO are equal to 1.

The interrupt cycle stores the return address (available in PC) into memory location 0, branches to memory location 1, and clears IEN, R, and SC to 0. This can be done with the following sequence of microoperations:

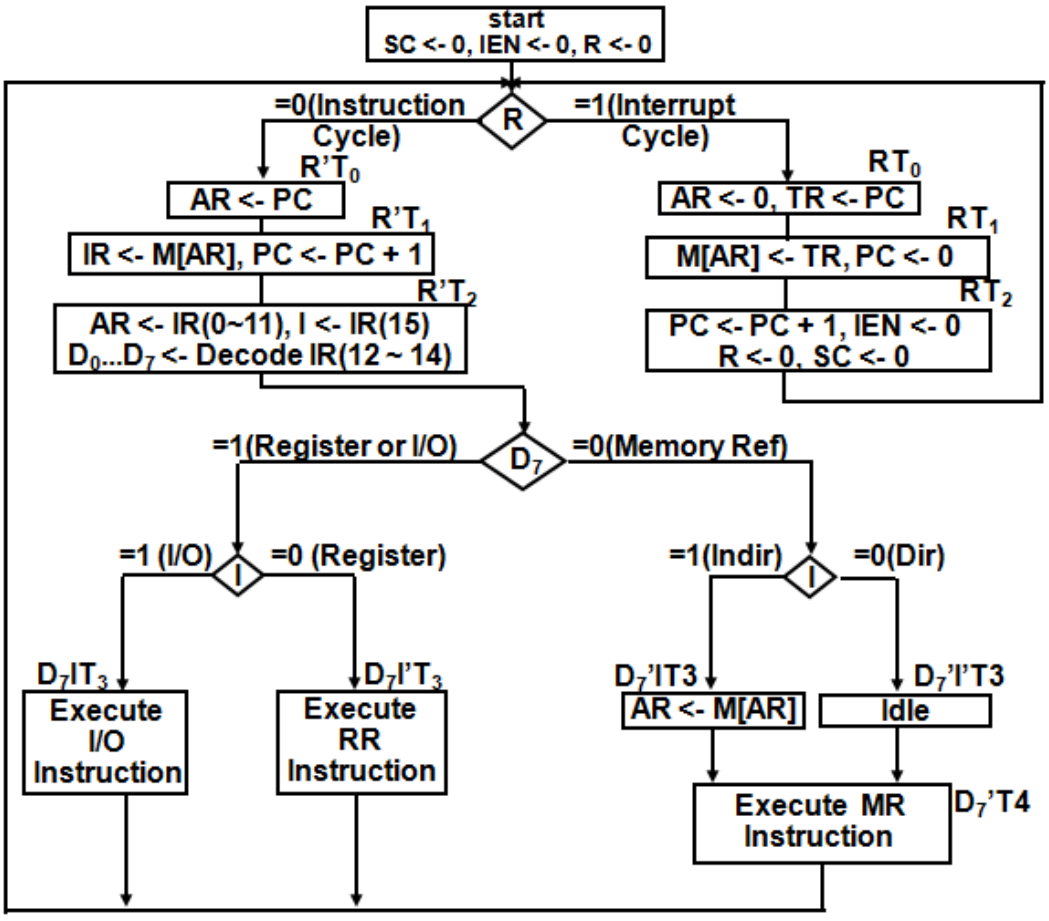
i). RT0: AR ← 0, TR ← PC (During the first timing signal AR is cleared to 0, and the content of PC is transferred to the temporary register TR).

ii). RT1: M[AR] ← TR, PC ← 0 (With the second timing signal, the return address is stored in memory at location 0 and PC is cleared to 0).

iii). RT2: PC←PC + 1, IEN← 0, R ← 0, SC←0 (The third timing signal increments PC to 1, clears IEN and R, and control goes back to T0 by clearing SC to 0).

The beginning of the next instruction cycle has the condition R'T0 and the content of PC is equal to 1. The control then goes through an instruction cycle that fetches and executes the BUN instruction in location 1.

Fig 3.9. Complete Computer Description:



3.15 Questions:

1. Explain stored program organisation.
2. What is effective address? How effective address is found using direct and indirect address.
3. Elucidate common bus system.
4. Differentiate hardware and microprogram control unit.
5. Explain the control unit of the basic computer.
6. Explain about instruction cycle with the help of the example
7. Explain execution of a complete instruction with the help of an example.
8. Explain memory reference instructions.
9. Explain branch instructions with an example.
10. Illustrate the operation of input-output configuration.
11. Explain interrupt cycle with the help of the example.
12. Explain complete computer description with the help of flow-chart.

Multiple choice questions:

1. The opcode 000 in a 16 bit instruction specifies
a). Register reference b) Memory reference c). I/O reference d). None

2. Which of the following device gives faster access……
a) Hard disk b) Compact disk c) SSD d) Floppy Disk

3. Modern processors use ..... controlled organisation
a). Hybrid b). Hardwired c). Microprogram d). None

4. Which one of the below instruction is register reference instruction
a) ADD b) AND c) Clear AC d) BSA

5. If the interrupt input flip flop R value is 0, it indicates …..……
a) Instruction cycle b) Interrupt cycle c) Memory cycle d) Register cycle

Fill in the blanks:

1. The effective address is the address of the …….
2. If the I (15^{th}) bit is 1 and all opcode bits are 1's is a ……..... reference instruction.
3. The INPR register is used to hold .....
4. …….. control organisation consists of a physical circuit which is difficult to change.
5. Which control organisation is better if the user wants to change program frequently...….
6. In the instruction cycle, $D_7I'T_3$ indicates the execution of a ……
7. The SKI and SKO are ..... instructions
8. The BUN instruction means .....
9. The FGI and FGO are checked only when ……
10. The External interrupts are caused by .....

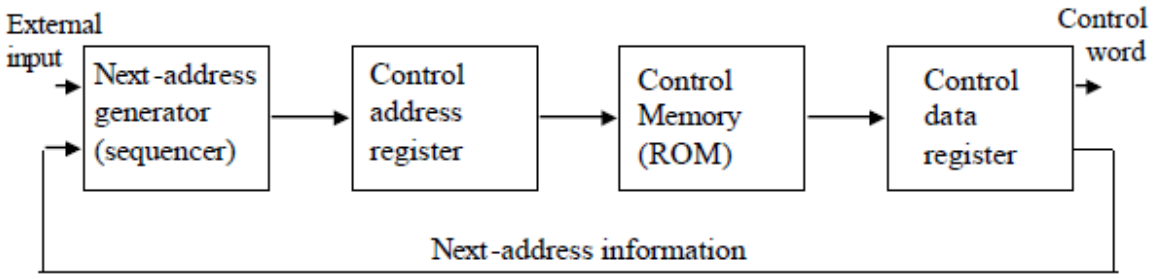
4. MICRO PROGRAMMED CONTROL

4.1 Control Memory: The function of the control unit in a digital computer is to initiate sequences of micro-operations. A memory that is a part of the control unit is referred as control memory.

4.2 Hardwired control: When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be hardwired. Most computers based on the RISC architecture use hardwired control which gives better performance than the micro-program control.

4.3 Micro Programmed Control Organisation:

A control unit whose binary control variables are stored in the memory is called a micro-programmed control unit. This control unit has two separate memories: a main memory and a control memory. The control memory holds a fixed micro-program, whereas the main memory is used for storing the user programs.



The control unit of a micro programmed processor consists:

i). The sequencer: Determines the next address from within control memory. The next address generator called a micro-program sequencer, determines the address sequence that is read from control memory.

ii). Control address register: Contains the address of the next microinstruction to be executed. The control address register specifies the address of the micro operation.

iii). Control memory: Where microinstructions are stored. The control memory is a ROM, within which all control information is permanently stored.

iv). Control data register: Contains the microinstruction to be executed. The control data register holds the present micro instruction while the next address is computed and read from memory.

It allows the execution of the micro operations specified by the control word simultaneously with the generation of the next microinstruction. The main advantage of the micro programmed control is the fact that there is no need to change any hardware or wiring to change the control sequence for the system. The only thing that must be changed is the micro program residing in control memory.

4.4 Definitions:

i). Micro operation: Micro operations are the operations, executed by the registers in a computer.

ii). Micro instruction: The micro instruction contains a control word that specifies one or more micro-operations for the (data processor) system.

iii). Micro program: A micro program consists of micro operations that specify various internal control signals for execution of register micro operations.

Micro program involves placing all control variables in words of ROM for use by the control unit through successive read operations.

iv). Micro code: Micro code generation is called micro programming and is a process similar to conventional machine language programming.

v). Routine: A set of programming instructions designed to perform a specific limited task. Micro instructions are stored in control memory in groups, with each group specifying a routine.

vi). Subroutine: Subroutines are programs that are used by other routines to accomplish a particular task. A sub routine can be called from any point within the main body of the micro program.

4.5 Mapping:

Every instruction has its own micro program routine stored in the control memory. The transformation from the instruction code bits to an address in control memory where the routine is located is referred as a mapping process.

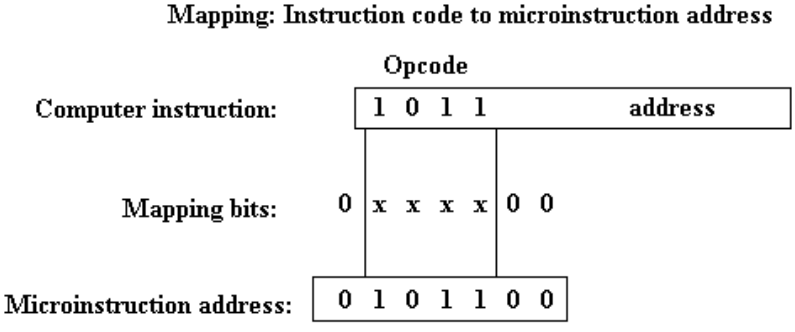
Mapping procedure is a rule that transforms the instruction code into a control memory address. It provides flexibility for adding instructions for control memory as the need arises.

Example: A computer with a simple instruction format has an operation code of 4 bits and if the control memory has 128 words, it requires an address of seven bits.

One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory is shown in the Figure.

This mapping consists of placing a 0 in the MSB, and clearing the two LSB of the control address register.

For each operation code there exists a micro program routine in control memory that executes the instruction.

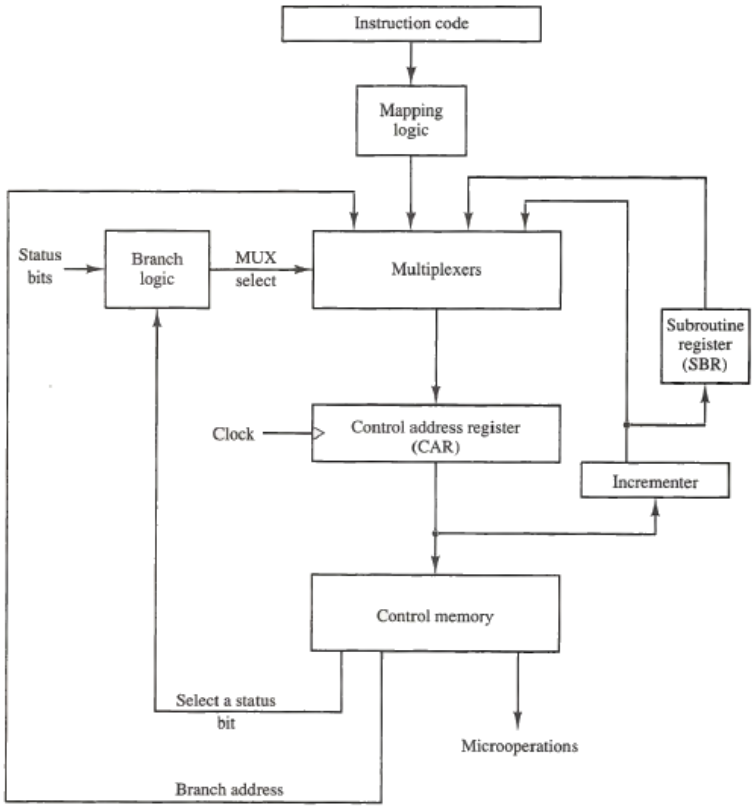


4.6 Address Sequencing:

Microinstructions are stored in the control memory in groups, where each group specifying a routine and each routine specifies how to carry out a routine. Each routine must be able to branch to the next routine in the sequence.

When the power is turned on, the first address loads into CAR is the address of the first microinstruction that activates the instruction fetch routine.

The control memory goes through the routine and the effective address is computed in control memory.



The micro operations are generated that execute the instruction fetched from memory by mapping process.
The micro instructions are sequenced by incrementing the CAR.
When the execution of the instruction is completed, control must return to the fetch routine.
This is done using an unconditional branch microinstruction.

Summary:

a). Incrementing of the control address register (CAR)
b). Unconditional branch or conditional branch depending on status bit conditions.
c). A mapping process from the bits of the instruction to an address for control memory.
d). Handling subroutine call and return.

4.7 Micro program Example:

The block diagram of the computer consists of two memory units: a main memory for storing instructions and data, and a control memory for storing the micro program.

The processor unit contains four registers: PC (program counter), AR (address register), DR (data register), and AC (accumulator register).

The control unit has two registers: CAR control address register and SBR subroutine register.

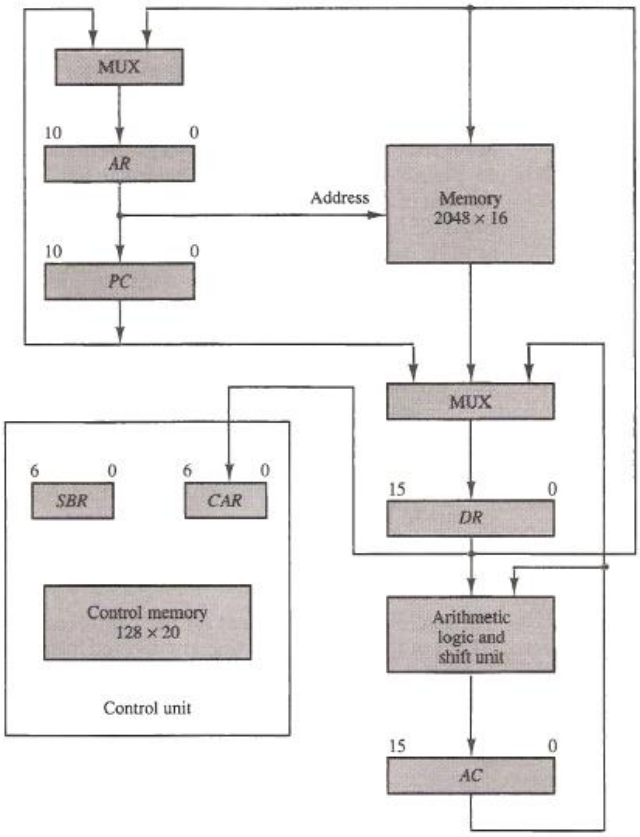
The control memory and its registers are organized as a microprogrammed control unit.

The transfer of information among the registers in the processor is done through multiplexers rather than a common bus.

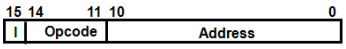
DR can receive information from AC, PC, or memory. AR can receive information from PC or DR. PC can receive information only from AR.

The arithmetic, logic, and shift unit performs microoperations with data from AC and DR and places the result in AC.

Memory receives its address from AR. Input data written to memory come from DR, and data read from memory can go only to DR.



Machine instruction format



The computer instruction format consists of three fields: a 1-bit field for indirect addressing symbolized by I, a 4-bit operation code (opcode), and an 11-bit address field. Below Figure lists four of the 16 possible memory-reference instructions.

The ADD instruction adds the content of the operand found in the effective address to the content of AC.

The BRANCH instruction causes a branch to the effective address if the operand in AC is negative.

The STORE instruction transfers the content of AC into the memory word specified by the effective address.

Sample machine instructions

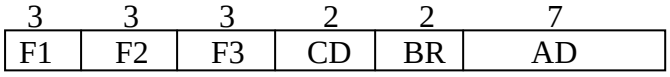
Symbol	OP-code	Description
ADD	0000	AC ← AC + M[EA]
BRANCH	0001	if (AC < 0) then (PC ← EA)
STORE	0010	M[EA] ← AC
EXCHANGE	0011	AC ← M[EA], M[EA] ← AC

EA is the effective address

The EXCHANGE instruction swaps the data between AC and the memory word specified by the effective address.

4.8 Micro instruction Format:

The micro instruction consists of 20 bits which are divided into four functional parts. The three fields F1, F2 and F3 specify micro operations for the computer.



= 20 bits

F1, F2, F3: Micro operation Fields
CD: Conditional for Branching
BR: Branch field
AD: Address field.

The micro operations are subdivided into three fields of three bits each. The three bits in each field are encoded to specify seven distinct micro operations as listed below. This gives a total of 21 micro operations.

Symbols and Binary Code for Microinstruction Fields.

F1	Microoperation	Symbol
000	None	NOP
001	AC ← AC + DR	ADD
010	AC ← 0	CLRAC
011	AC ← AC + 1	INCAC
100	AC ← DR	DRTAC
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	AC ← AC - DR	SUB
010	AC ← AC ∨ DR	OR
011	AC ← AC ∧ DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTDR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR

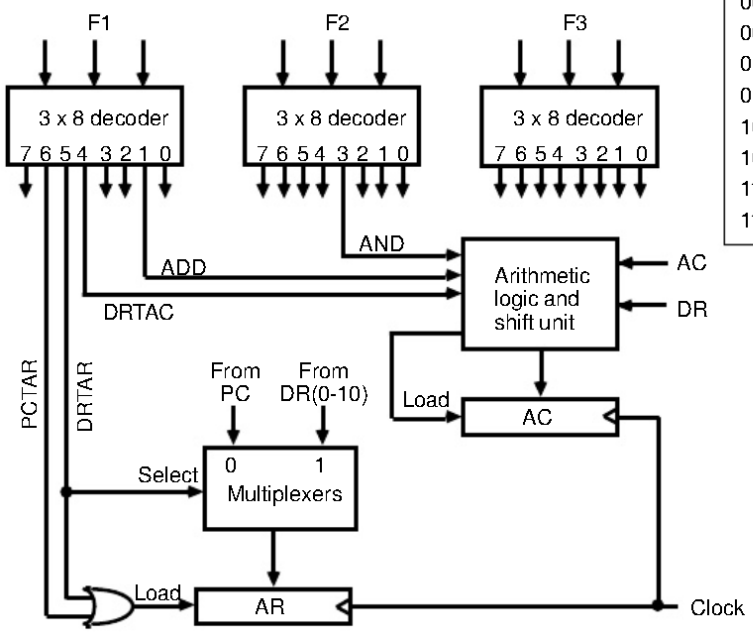
F3	Microoperation	Symbol
000	None	NOP
001	AC ← AC ⊕ DR	XOR
010	AC ← AC'	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	Reserved	

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

BR	Symbol	Function
00	JMP	CAR←AD if condition =1 CAR←CAR + 1 if condition = 0
01	CALL	CAR←AD, SBR ← CAR +1 if condition = 1 CAR←CAR + 1 If condition = 0
10	RET	CAR←SBR (Return from subroutine)
11	MAP	CAR(2-5)←DR (11-14), CAR(0,1,6)← 0

4.9 Design of control Unit:

The bits of the microinstruction are into fields, with each field defining a distinct, separate function. The nine bits (F1, F2 and F3) of micro operation fields are divided into three sub fields of three bits each.



F1	Symbol
000	NOP
001	ADD
010	CLRAC
011	INCAC
100	DRTAC
101	DRTAR
110	PCTAR
111	WRITE

F2	Symbol
000	NOP
001	SUB
010	OR
011	AND
100	READ
101	ACTDR
110	INCDR
111	PCTDR

F3	Symbol
000	NOP
001	XOR
010	COM
011	SHL
100	SHR
101	INCPC
110	ARTPC
111	

F1	Microoperation	Symbol
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR

F1	Microoperation	Symbol
001	AC ← AC + DR	ADD
100	AC ← DR	DRTAC

F2	Microoperation	Symbol
011	AC ← AC ∧ DR	AND

The control memory output of each sub field must be decoded to provide the distinct operations. The outputs of decoders are connected to the appropriate inputs in the processor unit. The three fields output from the control memory are decoded with a 3 x 8 decoder to provide eight outputs.

For example:

When F1 = 101 (binary 5), transfers the content of DR (0-10) to AR (DRTAR).
When F1 = 110 (binary 6) Transfer from PC to AR (PCTAR).

Outputs 5 and 6 are connected to the load input of AR so that either one of these outputs are active information from the multiplexers is transferred to AR.

The multiplexer selects DR when output 5 is active (1) and from PC when output 5 is inactive (0).

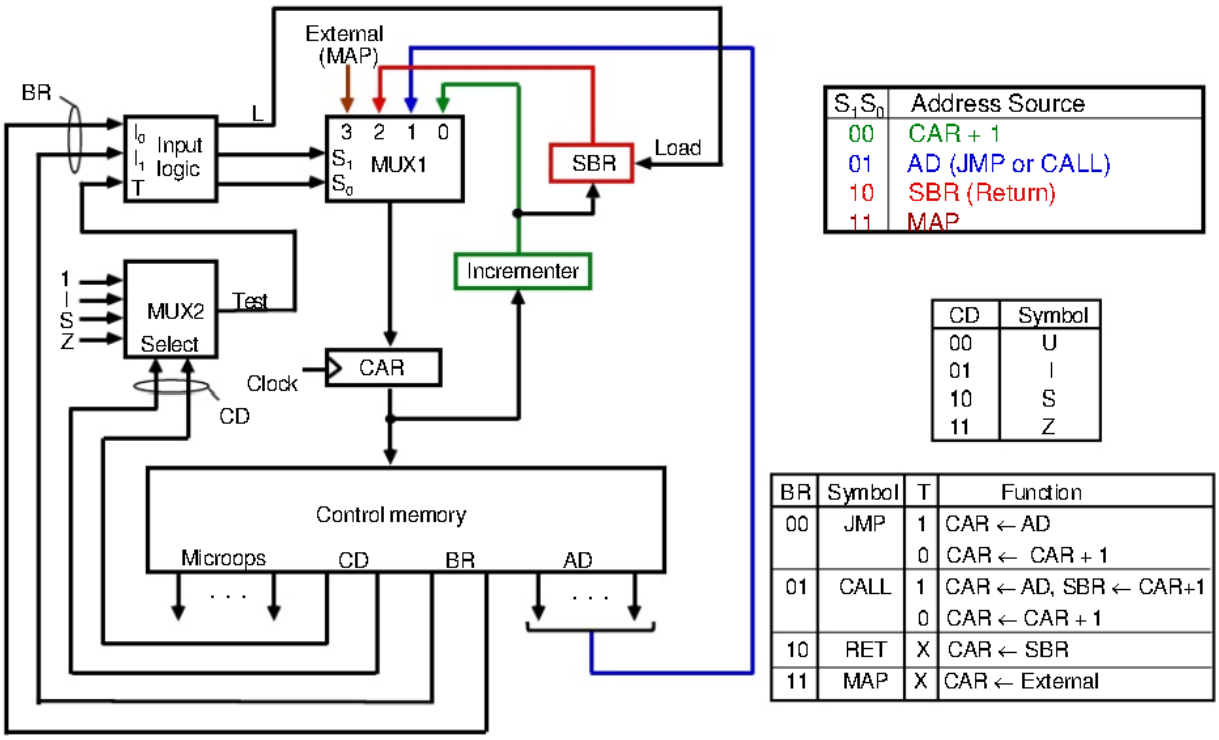
The transfer into AR occurs with a clock pulse transition only when output 5 or 6 of the decoder is active.

The other outputs of the decoders are connected as shown in the figure.

Inputs ADD, AND & DR are connected to arithmetic logic shift unit from their respective decoder outputs. Outputs of the decoder, associated with an AC operation must be connected to the arithmetic logic shift unit in similar fashion.

4.10 Micro program sequencer:

The basic components of a micro programmed control unit are the control memory and the circuits that selects the next address. The address selection part is called a micro program sequencer. The purpose of a micro program sequencer is to present an address to the control memory so that a microinstruction may be read and executed.



S_1S_0	Address Source
00	CAR + 1
01	AD (JMP or CALL)
10	SBR (Return)
11	MAP

CD	Symbol
00	U
01	I
10	S
11	Z

BR	Symbol	T	Function
00	JMP	1	CAR ← AD
		0	CAR ← CAR + 1
01	CALL	1	CAR ← AD, SBR ← CAR+1
		0	CAR ← CAR + 1
10	RET	X	CAR ← SBR
11	MAP	X	CAR ← External

There are two multiplexers in the circuit. MUX1 and MUX2.

MUX1 selects an address from one of the four sources and routes it into a control address register CAR.

MUX2 tests the value of a selected status bits and the result of the test is applied to an input logic circuit.

The output from CAR provides the address for the control memory. The content of CAR is incremented and applied to one of the multiplexer inputs and to the subroutine register SBR.

MUX 1 Inputs: Input 0 From the incrementer.
Input 1 From the address field of the present micro instruction and
Input 2 From the output of SBR (sub routine register) and
Input 3 From the external source that maps the instruction.

The CD (condition) field of the micro instruction selects one of the status bits in the second multiplexer.

If the bit selected is equal to 1, the T (test) variable is equal to 1; otherwise it is equal to 0. The T value together with the two bits (I_0 and I_1) from the BR (branch) field go to an input logic circuit.

The input logic in a particular sequencer will determine the type of operations that are available in the unit. Typical sequencer operations are increment, branch or jump, call and return from subroutine, load an external address, push pop the stack and other address sequencing operations.

With three inputs the sequencer can provide up to eight address sequencing operations:

The input of logic circuit has three inputs I_0, I_1 and T and three outputs S_0, S_1 and L. Variables S_0 and S_1 select one of the source addresses for CAR.

The subroutine register is loaded (L) with the incremented value of CAR during a call microinstruction (BR = 01) provided that the status bit condition is satisfied (T = 1).

For example with S1 S0 =10 multiplexer input number 2 is selected and establishes a transfer path from SBR to CAR.

Note that each of the four inputs as well as the output of MUX 1 contains a 7 bit address.

$S1 = I_1$
$S0 = I_1 I_0 + I'_1 T$
$L = I'_1 I_0 T$

Truth table:

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

The circuit can be constructed with three AND gates, an OR gate, and an inverter.

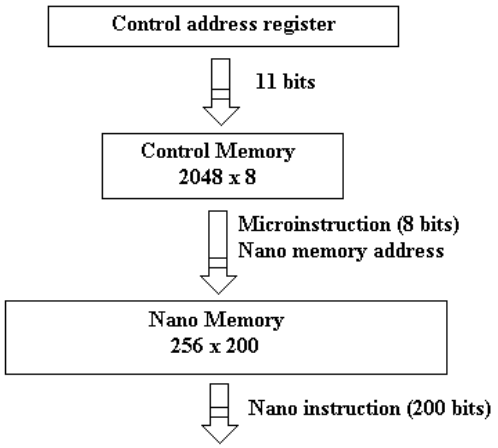
4.11 Nano instructions:

In micro programmed processors an instruction fetched from memory is interpreted by a micro program stored in a single control memory CM. However in few micro programmed processors the micro instructions do not directly used by the decoder to generate control signals.

Instead they are used to access second control memory called a nano control memory (nCM).

The decoder circuits in a vertical micro program storage organization can be replaced by a ROM by two levels of control storage.

At first level vertical format micro program is used at the second level Horizontal format nano program interprets the microinstruction fields, thus converts a vertical microinstruction format into a horizontal nano instruction format.



Usually, the micro program consists of a large number of short microinstructions, while the nano program contains fewer words with longer nano instructions.

4.12 Techniques of Grouping of Control Signals:

The grouping of control signals can be done either by using technique called vertical organisation or by using technique called horizontal organisation.

i). Vertical Organisation:

Highly encoded scheme that use compact codes to specify only a small number of control functions in each microinstruction are referred to as a vertical organisation. Every action is encoded in density.

Vertical micro code are slower but they take less space and their actions at execution time need to be decoded to a signal.

ii). Horizontal Organisation:

The minimally encoded scheme, in which resources can be controlled with a single instruction, is called a horizontal organisation. In this type of code the micro code contains the control signal without any intermediary.

Horizontal micro code instruction contains a lot of signals and hence due to that the number of bits also increase.

Comparison between horizontal and vertical organisation.

Sr. No.	Horizontal	Vertical.
1.	Long formats.	Short formats.
2.	Ability to express a high degree of parallelism.	Limited ability to express parallel microoperations.
3.	Little encoding of the control information.	Considerable encoding of the control information.
4.	Useful when higher operating speed is desired.	Slower operating speeds

The advantages and disadvantages of horizontal and vertical organisations can be summarized as follows:

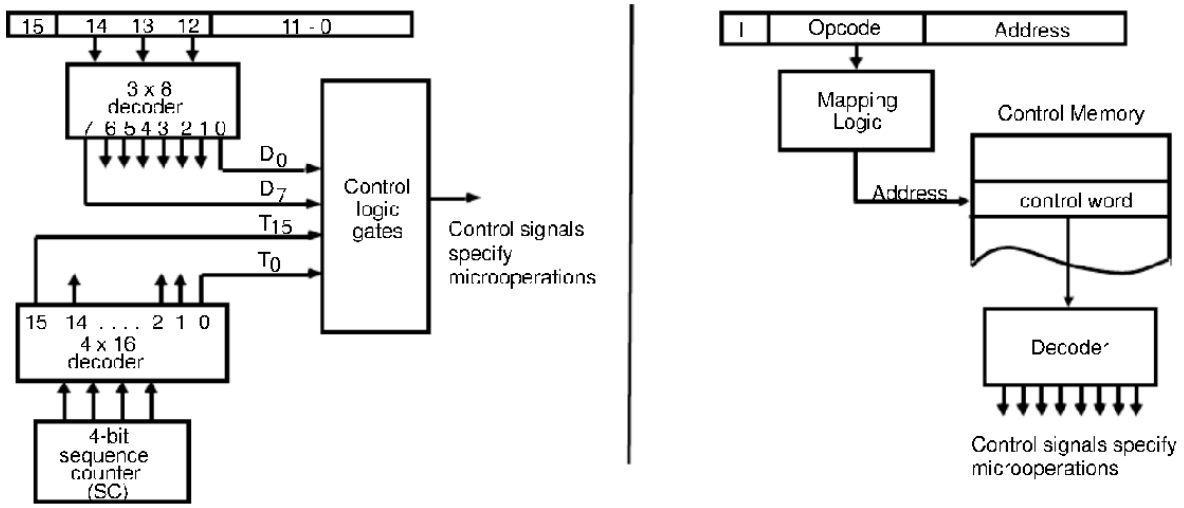
* The horizontal organisation approach is suitable when operating speed of computer is a critical factor and where the machine structure allows parallel usage of a number of resources.

* Vertical approach results in slower operations speed but less bits are required in the microinstruction.

4.13 Hard wired control Vs Micro programmed control:

Hardwired control unit:	**Micro-programmed Control Unit**:
- It uses flags, decoder, logic gates and other digital circuits	- It uses sequence of micro-instruction in micro programming language
- As name implies it is a hardware control unit.	- It is mid-way between Hardware and Software.
- On the basis of input Signal output is generated.	- It generates a set of control signal on the basis of control line.
- Difficult to design, test and implement.	- Easy to design, test and implement.
- Expensive and high error.	- Cheaper and less error.
Advantage: - Faster mode of operation.	**Advantage**: - Flexible to modify. (Control program can be modified.)
Disadvantage: - Inflexible to modify. (Require change in wiring of the design.)	**Disadvantage**: - Slower mode of operation
Application: - Used in RISC processor	**Application**: - Used in CISC processor.

Comparison between Hardware and Micro programmed control Unit:



4.14 Questions:

1. Draw and explain micro programmed control unit.
2. Explain about a) Micro Program, b) Micro Code, c) Micro Instructions d) micro operation e) subroutines.
3. Describe in detail about address sequencing.
4. What is microinstruction? How do we reduce number of microinstructions?
5. Explain the different branching techniques used in micro programmed control unit.
6. Draw and explain the hardwired control unit.
7. Explain the organization and functioning of Micro program sequencer.
8. Explain nano program and nano instructions
9. Compare horizontal and vertical organization. Give their advantages and disadvantages.
10). Hard wired control unit is faster than Micro Programmed Control unit. Justify this statement.
11. Differentiate between Hardwired control and micro programmed control.
12. Formulate a mapping procedure that provides eight consecutive micro instructions for each routine. The operation code has 7 bits and control memory has 2048 words.

Multiple choice questions:

1. The Control Memory is a
a). RAM b) ROM c). NVRAM d). PROM
2. Micro operations in a PC are executed by ……
a) programs b) registers c) memory d) none
3. How many multiplexers are used in micro program sequencer .....
a). 2 b) 3 c). 4 d). 5
4. Which organisation use long formats .....……
a) Vertical b) Horizontal c) Inclined d) none
5. The Control Unit is easy to design .....
a). Hard wired b) Micro programmed c). Hybrid programmed d) None

Fill in the blanks:

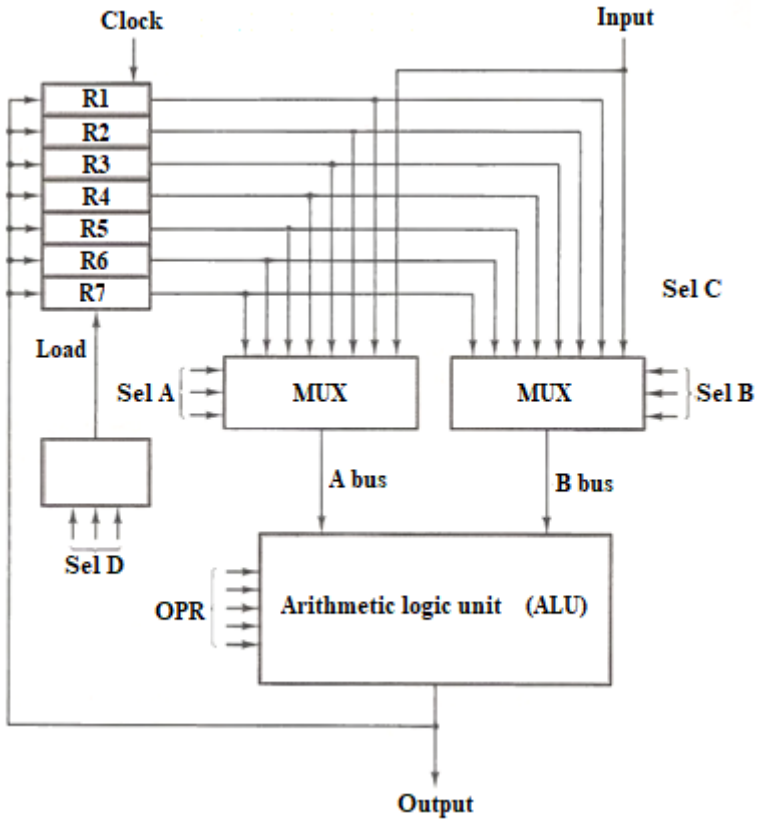
1. A routine is a set of programming instructions designed to …….
2. Mapping is a rule that transforms ……....
3. The CAR in address sequencing is a .....
4. Describe the meaning of AC←AC + M[EA] .....
5. Subroutine are used to accomplish a .....

This page intentionally left blank.

5. CENTRAL PROCESSING UNIT

5.1 General Register Organization:

CPU process registers are very fast and efficient then memory. To communicate between these registers as well as to perform arithmetic, logic, and shift microoperations it uses a common bus system as shown in the below figure.



A bus organization consists of seven CPU registers. The output of each register is connected to two multiplexers MUX A and MUX B to form the two buses A and B. These two buses A and B provide inputs to the arithmetic logic unit (ALU).

Selection signals SELA and SELB are used to select one register (or the input data) for each multiplexer to transfer to ALU.

For example, to perform the operation R1← R2 + R3

The control unit provides binary selection variables to

1. MUX A selector (SELA): to place the content of R2 into bus A.

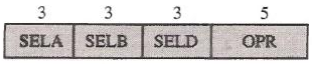
2. MUX B selector (SELB): to place the content of R3 into bus B.

3. ALU operation selector (OPR): to provide the arithmetic addition A + B.

4. Decoder destination selector (SELD): to transfer the content of the output bus into R l.

5.2 Control Word:

The control word consists of 14 bits in which SELA, SELB, SELC takes 3 bits each and OPR holds 5 bits.



Microoperation	Symbolic Designation				Control Word
	SELA	SELB	SELD	OPR	
$R1 \leftarrow R2 - R3$	R2	R3	R1	SUB	010 011 001 00101
$R4 \leftarrow R4 \vee R5$	R4	R5	R4	OR	100 101 100 01010
$R6 \leftarrow R6 + 1$	R6	—	R6	INCA	110 000 110 00001
$R7 \leftarrow R1$	R1	—	R7	TSFA	001 000 111 00000
Output $\leftarrow R2$	R2	—	None	TSFA	010 000 000 00000
Output $\leftarrow$ Input	Input	—	None	TSFA	000 000 000 00000
$R4 \leftarrow$ sh1 $R4$	R4	—	R4	SHLA	100 000 100 11000
$R5 \leftarrow 0$	R5	R5	R5	XOR	101 101 101 01100

The most efficient way to generate control words with a large number of bits is to store them in a memory unit as a control memory.

By reading consecutive control words from memory, it is possible to initiate the desired sequence of microoperations for the CPU.

This type of control is referred to as microprogrammed control.

5.3 Stack Organisation:

* A stack is a storage device organized as a collection of a finite number of memory words or registers.

* It stores information in such a manner that the item stored last is the first item retrieved (LIFO – Last in first out).

* The register that holds the address of the stack is known as stack pointer (SP) and its value always points at the top item in the stack.

Two operations *push* (push down) to insert item and *pop* (pop up) to delete the item are used.

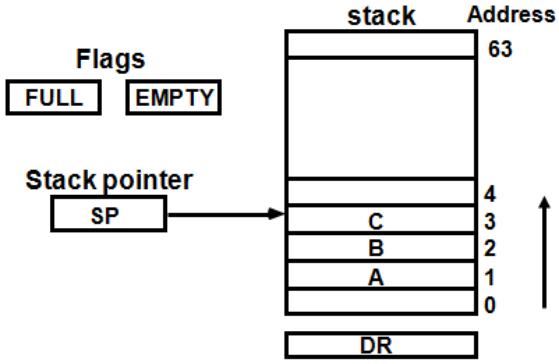
These operations are simulated by incrementing or decrementing the stack pointer register.

5.3.1 Register Stack:

Figure shows the organization of a 64-word register stack. The stack pointer register SP contains the address of the word (in binary) that is currently on top of the stack. For example if three items *A, B,* and C are placed in the stack then item C is on top of the stack and the content of SP is now 3.

PUSH: To insert a new item, the stack is pushed by incrementing SP and writing a word in the next-higher location in the stack.

SP ← SP + 1 Increment stack operator.
M[SP] ← DR Write item on top of the stack.



If (SP = 0) then (FULL ←1) stack is full then (EMPTY ← 0) stack is not empty.

(In a 64-word stack, the stack pointer contains 6 bits because (2^6 = 64). So it can access up to 63 (111111). When 63 is incremented by 1, the result is 0. since 111111 + 1 = 1000000 in binary, but SP can accommodate only the six least significant bits. Similarly, when 000000 is decremented by 1, the result is 111111.

The one-bit register FULL is set to 1 when the stack is full, and the one-bit register EMTY is set to 1 when the stack is empty of items.

POP: A new item is deleted from the stack if the stack is not empty (if *EMTY* = 0). The pop operation consists of the following sequence of microoperations:

The pop up operation consists of the following sequence of micro operations:

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

Initially, SP is cleared to 0, EMPTY is set to 1 and FULL is cleared to 0. By default the SP points to the word at address 0 and the stack is empty (EMPTY =1) and not full (FULL = 0).

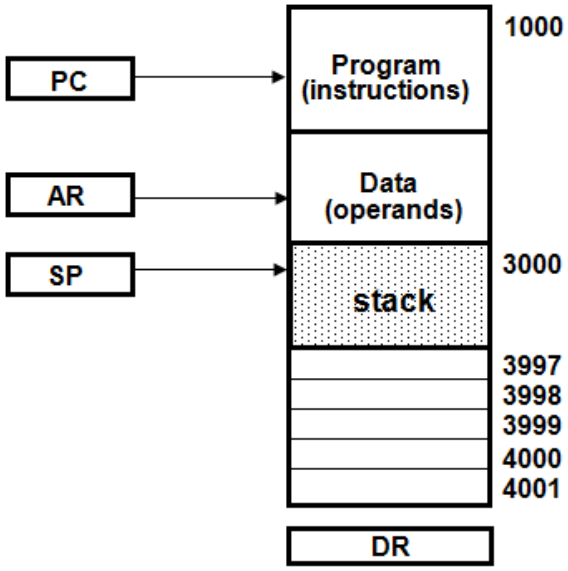
5.3.2 Memory Stack:

The memory stack consists of three segments: Program, data and stack.

i). The program counter PC points at the address of the next instruction in the program.
ii). The address register AR points at an array of data.
iii). The stack pointer SP points at the top of the stack. The three registers are connected o a common bus.

PC is used during fetching phase to read an instruction.
AR is used during the execute phase to read an operand.
SP is used to push or pop items into or from the stack.

As shown in the figure the initial value of SP is 4001 and the stack grows with decreasing addresses.



We assume that the items in the stack communicate with a data register DR.

PUSH: A new item is inserted with the push operation as follows:

SP ← SP - 1 Decrement stack operator.
M[SP] ← DR Write item on top of the stack.

The stack pointer is decremented so that it points at the address of the next word.

A memory write operation inserts the word from DR into the top of the stack.

POP: A new item is deleted with a pop operation as follows:

M[SP] ← DR
SP ← SP + 1

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

STACK LIMITS:

Most computers do not provide hardware to check for stack overflow (full stack) or underflow (empty stack).

The stack limits can be checked by using two processor registers: one to hold the upper limit (3000 in this case), and the other to hold the lower limit (4001 in this case). After a push operation, SP is compared with the upper-limit register and after a pop operation, SP is compared with the lower-limit register.

5.4 Stack Notations:

A stack organization is very effective for evaluating arithmetic expressions. The common mathematical method of writing arithmetic expressions imposes difficulties when evaluated by a computer.

i). Infix Notation: The common arithmetic expressions are written in *infix* notation, with each operator written between the operands.

A * B + C * D

To evaluate this arithmetic expression it is necessary to compute the product A * B, store this product while computing C * D and then sum the products.

ii). Polish notation: This can be represented in prefix notation. The operator is placed before the operands.

iii). Reverse Polish Notation (RPN): The postfix notation referred to as reverse polish notation (RPN), places the operator after the operands. This is suitable for stack manipulation.

Examples:

Infix notation	Prefix or polish notation	Postfix or reverse polish notation
A + B	+AB	AB+

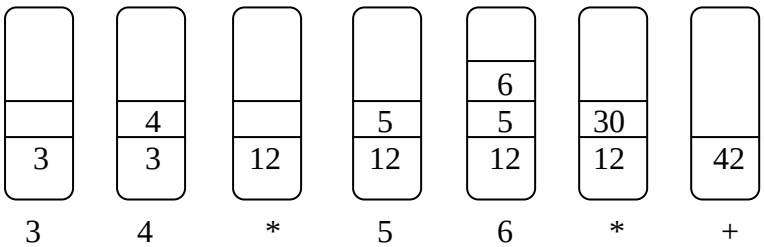
The conversion from infix notation to reverse Polish notation: must take into consideration the operational hierarchy adopted for infix notation. This hierarchy dictates that we first perform all arithmetic inside inner parentheses, then inside outer parentheses, and do multiplication and division operations before addition and subtraction operations.

Consider the expression. (A + B) * [C * (D + E) + F]
After conversion AB + DE + C * F + *

Ex: A * B + C * D is written in reverse polish notation as
AB * CD * + is evaluated as follows:

First Scan the expression from left to right. When an operator is reached, perform the operation with the two operands found on the left side of the operator. Remove the two operands and the operator and replace it with the result. Now continue the scan and repeat the process for every operator until there are no more operators.

Ex: stack operation to evaluate : (3*4) + (5*6) can be expressed as 34* 56* +



5.5 Types of CPU Organisations:

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

i). Single accumulator organization.
ii). General register organization.
iii). Stack organization.

i). An accumulator-type organization: In this organisation all operations are performed with an implied accumulator register. The instruction format in this type of computer uses one address field. For example:

ADD X

where X is the address of the operand. The ADD instruction in this case results in the operation *AC* ← *AC* + M[X]. *AC* is the accumulator register and M[X] symbolizes the memory word located at address X.

ii). General register type of organization: The instruction format in this type of computer needs three register address fields. For example:

ADD R l, R2, R

to denote, the operation R l ← R2 + *R3*. The number of address fields in the instruction can be reduced from three to two if the destination register is the same as one of the source registers. Thus the instruction

ADD R l, R2

would denote the operation R l ← R l + R2. Only register addresses for R l and *R2* need be specified in this instruction.

iii). The stack-organized CPU:

Computers with stack organization would have PUSH and POP instructions which require an address field. Example:

PUSH X

will push the word at address X to the top of the stack. The stack pointer is updated automatically. Operation-type instructions do not need an address field in stack-organized computers. This is because the operation is performed on the two items that are on top of the stack.

Example: ADD

This operation has the effect of popping the two top numbers from the stack, adding the numbers, and pushing the sum into the stack. There is no need to specify operands with an address field since all operands are implied to be in the stack.

5.6 Instruction Formats:

A computer will usually have a variety of instruction code formats of several different lengths containing varying number of addresses.

i). Three address Instructions:

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

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

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

M[A] = Operand at memory address symbolized by A. R1 & R2 are processor registers

ii). Two address instructions:

Two address instructions are the most common in commercial computers. Here again each address field can specify either a processor register or a memory word.

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

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

iii). One address instruction:

One address instruction use an implied accumulator (AC) register for all data manipulation.

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

LOAD	A	AC← M[A]	
ADD	B	AC←AC + M[B]	
STORE	T	M[T] ← AC	
LOAD	C	AC ← M[C]	
ADD	D	AC← AC + M[D]	
MUL	T	AC ← AC * M[T]	T is the temporary memory location.
STORE	X	M[X] ← AC	

iv). Zero address Instruction:

Stack organized computer does not use an address field for the instructions. Zero address means absence of an address field in the computational instructions. For arithmetic expressions in a stack computer it is necessary to convert the expression into reverse polish notation.

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

PUSH	A	TOS ← A	(TOS = top of stack).
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	

v). RISC Instructions:

A program for a RISC type CPU consists of LOAD and STORE instructions that have one memory and one register address and computational type instructions that have three addresses with all three specifying processor register.

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

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, R1, R2	R1← R1 + R2
ADD	R3, R3, R4	R3← R3 + R4
MUL	R1, R1, R3	R1← R1 * R3
STORE	X, R1	M[X] ← R1

The load instructions transfer the operands from memory to CPU registers. The ADD and multiply operations are executed with data in the registers without accessing memory. The result of the computations is then stored in memory with a store instruction.

5.7 Addressing modes:

The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.

Instruction format with mode field:

Opcode	Mode	Address

i). Implied mode: In implied addressing mode, the instruction itself specifies the data to be operated. In fact all register reference instructions that use an accumulator and zero address instructions in a stack organized computer are implied mode instructions.

Ex: CMA - Complement the content of accumulator

ii). 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 the address field.

EX. MVI B, 3EH - Move the data 3EH given in the instruction to B register

iii). Register mode: In register addressing mode, the instruction specifies the name of the register in which the data is available..

EX. MOV A, B - Move the content of B register to A register.

iv). Register indirect mode: In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.

EX. MOV A, M - The memory data addressed by H L pair is moved to A register.

v). Auto increment or Auto decrement: This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory.

vi). 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 their instruction.

EX. LDA 1050H - Load the data available in memory location 1050H in to accumulator

vii). Indirect address mode: In this mode the address field of the instruction gives the address where the effective address is stored in memory. In this case the number is usually enclosed with square brackets.

Ex: LD Acc, [5]

Load the value stored in the memory location pointed to by the operand into the accumulator

A few addressing modes require that the address field of the instruction be added to the content of a specific register in the CPU to obtain the effective address.

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

The CPU register 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.

viii). 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.

Example: If PC contains the number 825 and the address part contains the number 24.

The instruction at location 825 is read from memory during the fetch phase and the program counter is then incremented by one to 826.

Now the effective address computation for the relative address mode is 826 + 24 = 850.

ix). Indexed Addressing mode: In this mode the content of an index register is added to the address part of the instruction to obtain the effective address. The index register is a special CPU register that contains an index value.

Content of index register + address part = effective address

500 + 280 = 780

x). 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, similar to indexed address mode.

Content of base register + address part = effective address

500 + 280 = 780

Table: Eight Addressing modes for load (LD) instruction:

Mode		**Register Transfer**
Direct address	LD ADR	AC ← M[ADR]
Indirect address	LD @ADR	AC ← M[M[ADR]]
Relative address	LD $ADR	AC ← M[PC + ADR]
Immediate operand	LD #NBR	AC ← NBR
Index addressing	LD ADR(X)	AC ← M[ADR + XR]
Register	LD R1	AC ← R1
Register indirect	LD (R1)	AC ← M[R1]
Auto increment	LD (R1)+	AC ← M[R1], R1 ← R1 + 1
Auto decrement	LD -(R1)	R1 ← R1 - 1, AC ← M[R1]

5.8 Data Transfer and Manipulation:

Most computer instructions can be classified into three categories:

i). Data transfer instructions:
ii). Data manipulation instructions:
iii). Program control instructions:

i). Data transfer instructions:

Data transfer instructions move data from one place in the computer to another without changing the data content.

The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers themselves.

Table gives a list of eight data transfer instructions used in many computers. Accompanying each instruction is a mnemonic symbol. Different computers use different mnemonics for the same instruction name.

Load: The load instruction has been used mostly to designate a transfer from memory to a processor register, usually an accumulator.

The store instruction designates a transfer from a processor register into memory.

The move instruction designate a transfer from one register to another. Or data transfers between CPU registers and memory or between two memory words.

The exchange instruction swaps information between two registers or a register and a memory word.

The input and output instructions transfer data among processor registers and input or output terminals.

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

The push and pop instructions transfer data between processor registers and a memory stack.

ii). Data Manipulation instructions: Data manipulation instructions are those that perform arithmetic, logic, and shift operations. The data manipulation instructions in a typical computer are usually divided into three basic types.

a). Arithmetic instructions
b). Logical and bit manipulation instructions
c). Shift instructions

Arithmetic instructions Logical & bit manipulation instructions Shift instructions

Name	Mnemonic	Name	Mnemonic	Name	Mnemonic
Increment	INC	Clear	CLR	Logical shift right	SHR
Decrement	DEC	Complement	COM	Logical shift left	SHL
Add	ADD	AND	AND	Arithmetic shift right	SHRA
Subtract	SUB	OR	OR	Arithmetic shift left	SHLA
Multiply	MUL	Exclusive OR	XOR	Rotate right	ROR
Divide	DIV	Clear carry	CLRC	Rotate left	ROL
Add with carry	ADDC	Set carry	SETC	Rotate right through carry	RORC
Subtract with borrow	SUBB	Complement carry	COMC	Rotate left through carry	ROLC
Negate (2's complement)	NEG	Enable interrupt	EI		
		Disable interrupt	DI		

iii). Program control Instructions:

Each time an instruction is fetched from consecutive memory, program counter is incremented so that it contains the address of the next instruction in sequence. On the other hand a program control type of instruction, when executed may change the address value in the program counter and cause the flow of control to be altered.

Program control instructions specify conditions for altering the content of the program counter, as a result of the execution of a program control instruction causes a break in the sequence of instruction execution.

Branch and jump: Instructions may be conditional or unconditional.

An unconditional branch instruction causes a branch to the specified address without any conditions.

In case of conditional branch, the program counter is loaded with the branch address and the next instruction is taken from this address.

If the condition is not met, the program counter is not changed and the next instruction is taken from the next location in sequence.

The skip instruction: A conditional skip instruction will skip the next instruction if the condition is met. If the condition is not met, control proceeds with the next instruction in sequence.

The call and return instructions are used in conjunction with subroutines.

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

The compare perform a subtraction between two operands, but the result of the operation is not retained.

The test instruction performs the logical AND of two operands and updates certain status bits without retaining the result or changing the operands.

The status bits of interest are the carry bit, the sign bit, a zero indication, and an overflow condition.

Interrupt:

An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing.

The processor responds by suspending its current activities, saving its state, and executing a function called an *interrupt handler* (or an interrupt service routine, ISR) to deal with the event. This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities.

5.9 Types of Interrupts:

There are three major types of interrupts that cause a break in the normal execution of a program. they can be classified as:

i). Internal interrupts: Internal interrupts arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called traps.

Examples of interrupts caused by internal error conditions are register overflow, attempt to divide by zero, an invalid operation code, stack overflow and protection violation.

ii). External Interrupts: External interrupts come from Input – output (I/O) devices, from a timing device, or from any other external source.

iii). Software interrupts: Software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call. This is used by the programmer to initiate an interrupt procedure at any desired point in the program.

5.10 CISC and RISC:

1. Large number of instructions typically from 100 to 250 instructions.	1. Relatively few instructions
2. Some instructions that perform specialized tasks and are used **infrequently**.	2. Relatively few addressing modes
3. A large variety of addressing modes, typically from 5 to 20 different modes.	3. Memory access limited to load and store instructions
4. Variable length instruction formats.	4. All operations done within the registers of the CPU.
5. Instructions that manipulate operands in memory.	5. Fixed length easily decoded instruction format
	6. Single cycle instruction execution
	7. Hardwired rather than micro programmed control.

CISC: A computer with a large number of instructions is classified as a complex instruction set computer (CISC).

RISC: The concept of RISC architecture involves an attempt to reduce execution time by simplifying the instruction set of the computer. The major characteristics of a RISC processor are:

The small set of instructions of a typical RISC processor consists mostly of register to register operations, with simply load and store operations for memory access.

RISC instruction format is easy to decode. For faster operations a hardwired control is preferable over a micro programmed control.

RISC processors execute one instruction per clock cycle. This is done by overlapping the fetch, decode and execution phases of two or three instructions by using a procedure called pipelining.

Other characteristics are:
1. A relative large number of registers in the processor unit
2. Use of overlapped register windows to speed up procedure call and return
3. Efficient instruction pipeline
4. Compiler support for efficient translation of high level language programs into machine language programs.

5.11 The Performance Equation:

The following equation is commonly used for expressing a computer's performance ability:

$$\frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{instructions}}{\text{program}}$$

The CISC approach attempts to minimize the number of instructions per program, sacrificing the number of cycles per instruction. RISC does the opposite, reducing the cycles per instruction at the cost of the number of instructions per program.

5.12 Questions:

1. Explain General Register Organisation with neat diagram
2. What is a Stack? Explain the organization of a Stack
3. Describe types of CPU Organisations.
4. Elicit various types of instruction formats with examples Z=A*B+C
5. Explain how Z = (X-Y) / (X+Y) is evaluated in a stack based computer.
6. Explain the instructions used in RISC and derive ex: Z=A*B+C.
7. What is an addressing mode? Explain different addressing modes.
8. Explain the DATA Transfer, Manipulation and Program control instructions
9. What is an interrupt? Explain different types of interrupts
10. Explain the differences between RISC and CISC
11. What are the properties of RISC architecture? Explain the advantages and disadvantages of RISC architecture.

Multiple choice questions:

1. In stack organisation the data is retrieved in …….
a). FIFO b) LIFO c). LRU d). None
2. Which notation is commonly used in stack computer .....
a). Polish b) infix c). RPN d). None
3. ADD X instruction format is an example of .....…… organisation
a) Single accumulator b) General register c) Stack organization d) none
4. MOV A, M is an example for ..... address mode.
a). Register direct b) Register indirect c). Direct address d) Indirect address mode
5. If PC contains the number 825 and the address part contains the number 24, what is the effective address in relative address mode .....
a) 825 b) 849 c) 850 d) 866

Fill in the blanks:

1. In stack organisation the stack pointer always points to …….
2. RISC instructions use …….... and ..... instructions.
3. Zero address instructions use ..... instructions
4. Internal interrupts arises from .....
5. A computer with a large number of instructions is classified as .....

This page intentionally left blank.

6. COMPUTER ARITHMETIC

6.1 Algorithm:

A solution to a problem that is stated by a finite number of well defined procedural steps is called an algorithm. A conventional method for presenting algorithms is a flow chart.

There are three ways of representing **negative fixed-point binary numbers**:

i). Signed-magnitude,	Ex: -14	1 000 1110
ii). Signed-l's complement,	Ex: -14	1 111 0001
iii). Signed-2's complement.	Ex: -14	1 111 0010

* For integers, signed-2's complement representation is commonly used when performing arithmetic operations.

* For floating-point operations, most computers use the signed-magnitude representation for the mantissa.

Addition and Subtraction with signed magnitude Data:

6.2 Addition (subtraction) algorithm: The result is based on the signs and the magnitude of A and B as shown in the below table.

i). If the signs of A and B are identical add the two magnitudes and attach the sign of A to the result.

ii). If the signs of A and B are different

- compare the magnitudes and subtract the smaller number from the larger.
- choose the sign of the result to be same as A if A> B or
- complement of the sign of A if A < B.

iii). If the two magnitudes are equal, subtract B from A and make the sign of the result positive.

Operation	ADD magnitudes	Subtract Magnitudes		
		When A > B	When A < B	When A = B
(+A) + (+B)	+ (A+B)			
(+A) + (-B)		+ (A-B)	- (B - A)	+ (A-B)
(-A) + (+B)		- (A-B)	+ (B - A)	+ (A-B)
(-A) + (-B)	- (A+B)			
(+A) - (+B)		+ (A-B)	- (B - A)	+ (A-B)
(+A) - (-B)	+ (A+B)			
(-A) - (+B)	- (A+B)			
(-A) - (-B)		- (A-B)	+ (B - A)	+ (A-B)

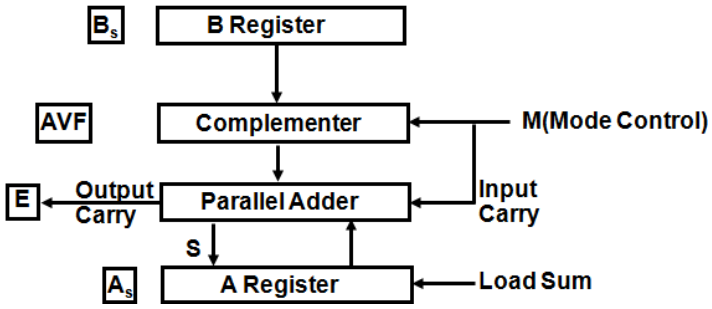
6.2.1 Hardware Implementation (for addition and subtraction):

The two numbers magnitudes are stored in registers A and B, and corresponding signs are stored in flip-flops As and Bs. The result of the operation may be transferred to a third register or to A and As.

The block diagram consists of registers A and *B* and sign flip-flops As and Bs. Subtraction is done by adding A to the 2's complement of B.

The output carry is transferred to flip-flop E, where it can be checked to determine the relative magnitudes of the two numbers.

The add-overflow flip-flop *AVF* holds the overflow bit when A and B are added.



The A register provides other microoperations that may be needed when we specify the sequence of steps in the algorithm.

The addition of A plus B is done through the parallel adder which consists of full-adder circuits. The *S* (sum) output of the adder is applied to the input of the A register.

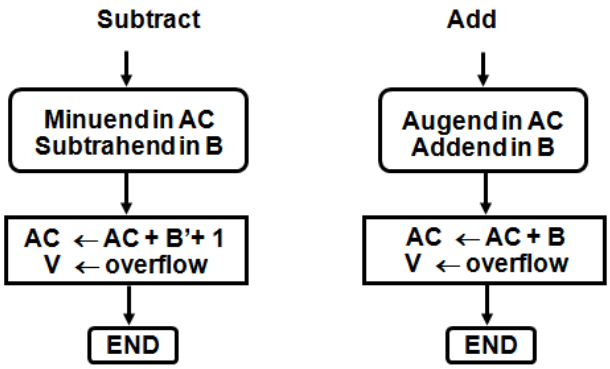
The complement circuit consists of exclusive-OR gates provides an output of *B or* the complement of B depending on the state of the mode control M.

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

When *M* = 0, the output of B is transferred to the adder, the input carry is 0, and the output of the adder is equal to the sum A + B.

When M = 1, the l's complement of *B* is applied to the adder, the input carry is 1, and output *S* = *A* + *B** + 1.

This is equal to *A* plus the 2's complement of B, which is equivalent to the subtraction *A*- B.



6.2.2 The flowchart for the hardware algorithm: (ADD and SUBTRACT operations)

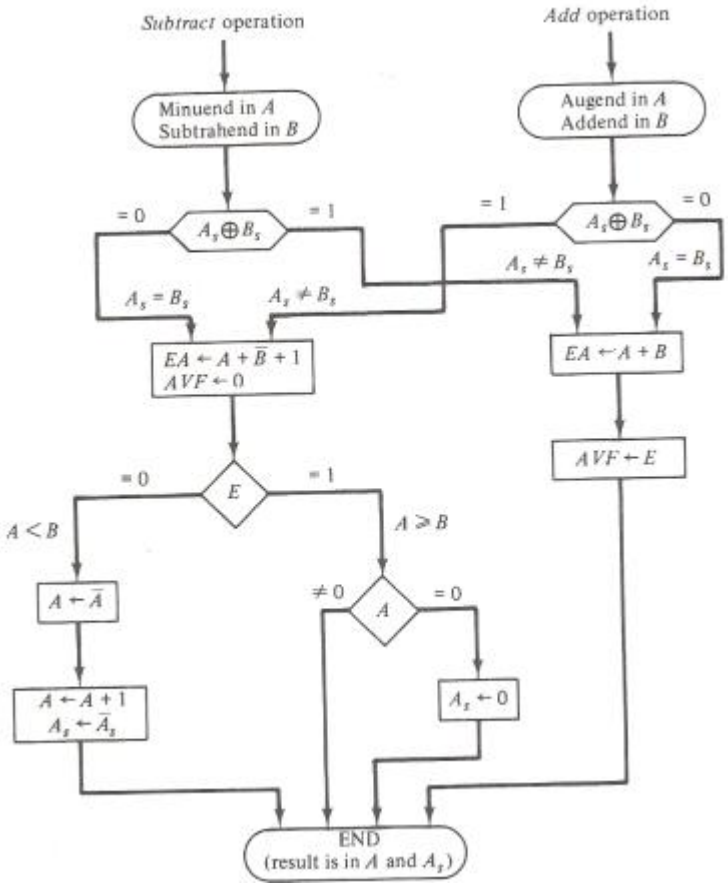
The two signs As and Bs are compared by an exclusive-OR gate. If the output of the gate is 0, the signs are identical; if it is 1, the signs are different.

For an *add* operation, identical signs dictate that the magnitudes be added. For a *subtract* operation, different signs dictate that the magnitudes be added.

The magnitudes are added with a microoperation $EA \leftarrow A + B$, where *EA* is a register that combines *E* and *A*.

The carry in E after the addition constitutes an overflow if it is equal to 1. The value of *E* is transferred into the add-overflow flip-flop *AVF*.

The two magnitudes are subtracted if the signs are different for an *add* operation or identical for a *subtract* operation.



The magnitudes are subtracted by adding *A* to the 2's complement of B. No overflow can occur if the numbers are subtracted so *AVF* is cleared to 0.

* A value 1 in E indicates that A ≥ B and the number in A is the correct result. If this number is zero, the sign *As* must be made positive to avoid a negative zero.

* A value 0 in E indicates that A < B. For this case it is necessary to take the 2's complement of the value in A. This operation can be done with one microoperation $A \leftarrow A^* + 1$.

However, we assume that the A register has circuits for microoperations *complement* and *increment*, so the 2's complement is obtained from these two microoperations.

In other paths of the flowchart, the sign of the result is the same as the sign of A, so no change in As is required. However, when A < B, the sign of the result is the complement of the original sign of A. It is then necessary to complement As to obtain the correct sign.

The final result is found in register *A* and its sign in *As*. The value in *AVF* provides an overflow indication. The final value of E is immaterial.

6.3 Addition and subtraction with signed-2's complement data:

The leftmost bit of a binary number represents the sign bit 0 for positive and 1 for negative. If the sign bit is 1, the entire number is represented in 2's complement form.

Ex: +33 = 0010 0001, -33 = 1101 1111 (2's complement of +33).

The addition of two numbers in signed-2's complement form consists of adding the numbers with the sign bits treated the same as the other bits of the number. A carry-out of the sign-bit position is discarded.

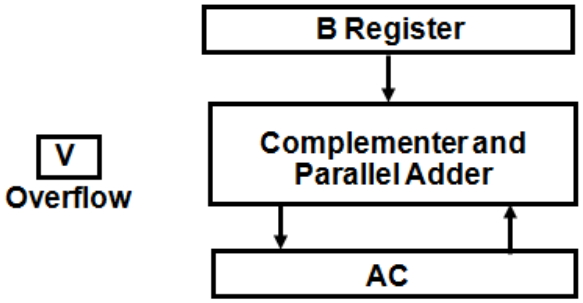
The subtraction consists of first taking the 2's complement of the subtrahend and then adding it to the minuend.

Overflow: When two numbers of n digits each are added and the sum occupies n+1 digits, we say that an overflow occurred. An overflow can be detected by applying the last two carries to an exclusive OR gate, the overflow is detected when the output of the gate is equal to 1.

6.3.1 Hardware implementation Addition and Subtraction with Signed-2's complement data:

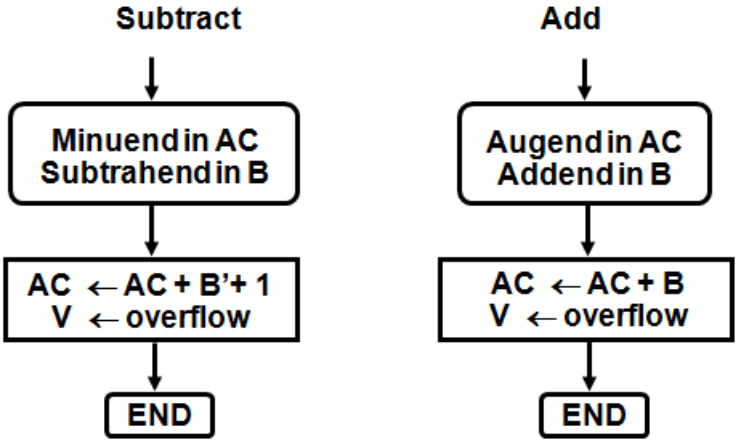
Let us take two registers A register AC (accumulator) and the B register BR.

The leftmost bit in AC and BR represent the sign bits of the numbers.



The two sign bits are added or subtracted together with the other bits in the complementer and parallel adder. The overflow flip-flop V is set to 1 if there is an overflow. The output carry in this case is discarded.

The algorithm for adding and subtracting two binary numbers in signed 2's complement representation is shown in the below flow chart.



6.4 Multiplication Algorithms:

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

If the multiplier bit is a 1, the multiplicand is copied down; otherwise, zeros are copied down.

The numbers copied down in successive lines are shifted one position to the left from the previous number.

Finally, the numbers are added and their sum forms the product.

```
 23      10111     Multiplicand
 19    × 10011     Multiplier
       ------
       10111
      10111
     00000    +
    00000
   10111
  ---------
 437 110110101   Product
```

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

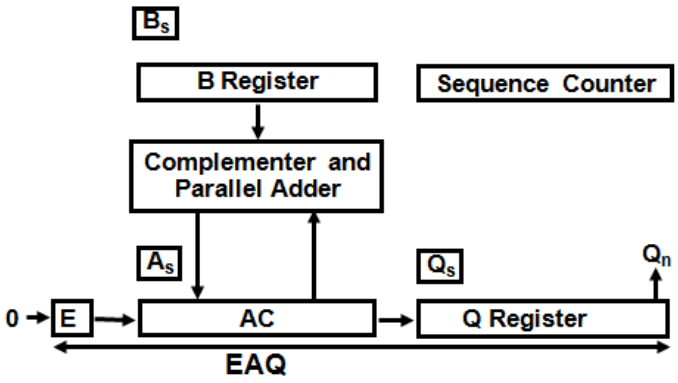
6.4.1 Hardware Implementation for Signed-Magnitude Data:

i). When multiplication is implemented in a digital computer, First, only two binary numbers are added and successively accumulate the partial products in a register.

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

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

The hardware for multiplication is as shown in the Fig. The multiplier is stored in the Q register and its sign in Qs.



The sequence counter SC is initially set to a number equal to the number of bits in the multiplier.

The counter is decremented by 1 after forming each partial product. When the content of the counter reaches zero, the product is formed and the process stops.

Initially, the multiplicand is in register B and the multiplier in Q . The sum of A and B forms a partial product which is transferred to the EA register.

Both partial product and multiplier are shifted to the right. This shift will be denoted by the statement shr EAQ to designate the right shift.

The least significant bit of A is shifted into the most significant position of Q, the bit from E is shifted into the most significant position of A, and 0 is shifted into E.

After the shift, one bit of the partial product is shifted into Q, pushing the multiplier bits one position to the right. In this manner, the rightmost flip-flop in register Q, designated by Q„, will hold the bit of the multiplier, which must be inspected next**.**

6.4.2 Multiplication Hardware Algorithms:

Initially the multiplicand is in B and the multiplier in Q, Their corresponding signs are in Bs and Qs respectively.

The signs are compared, and both A and Q are set to correspond to the sign of the product since a double length product will be stored in registers A and Q.

Registers A and E are cleared and the sequence counter SC is set to a number equal to the number of bits of the multiplier.

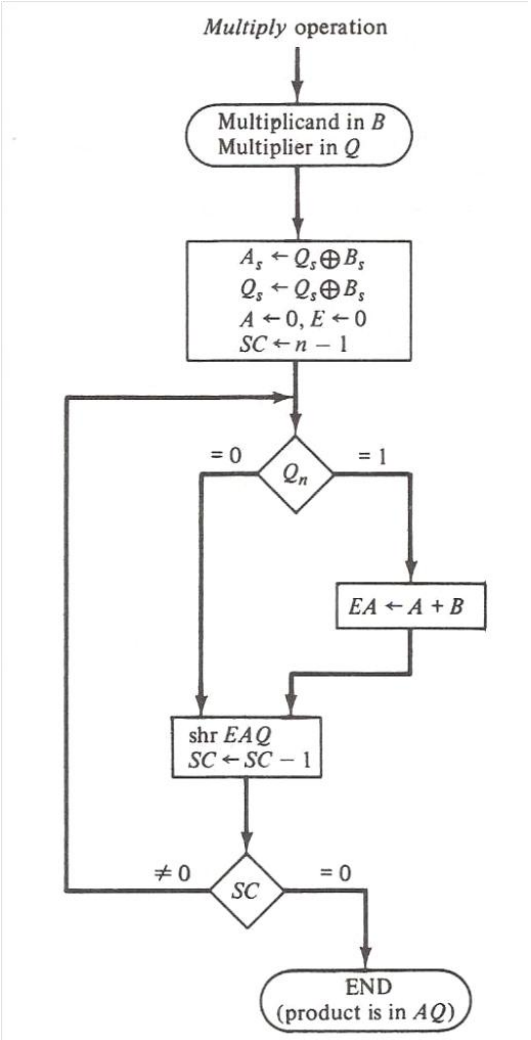
After the initialisation the low order bit of the multiplier in Qn is tested.

i). If it is a 1, the multiplicand in B is added to the present partial product in A.

ii). If it is 0, nothing is done.

iii). Register EAQ is then shifted once to the right to form the new partial product.

The final product is available in both A and Q.



Flow chart for multiply operation:

Multiplicand B = 10111 (23)	E	A	Q	SC
Multiplier in Q = 10011(19)	0	00000	10011	101
Qn = 1 (right most bit); add B		10111		
First partial product	0	10111		
Shift right EAQ	0	01011	11001	100
Qn = 1; add B		10111		
Second partial product	1	00010		
Shift right EAQ	0	10001	01100	011
Qn=0; shift right EAQ	0	01000	10110	010
Qn=0; shift right EAQ	0	00100	01011	001
Qn=1; add b		10111		
Fifth partial product	0	11011		
Shift right EAQ	0	01101	10101	000
Final Product in AQ = 0110110101				

6.5 Flow chart for Booth algorithm (for multiplication of signed-2's complement numbers):

Initially AC and Qn + 1 are cleared to 0 and the sequence counter SC is set to n where n is equal to the number of bits in the multiplier.

1). Inspect the two bits of Qn and Qn+1.

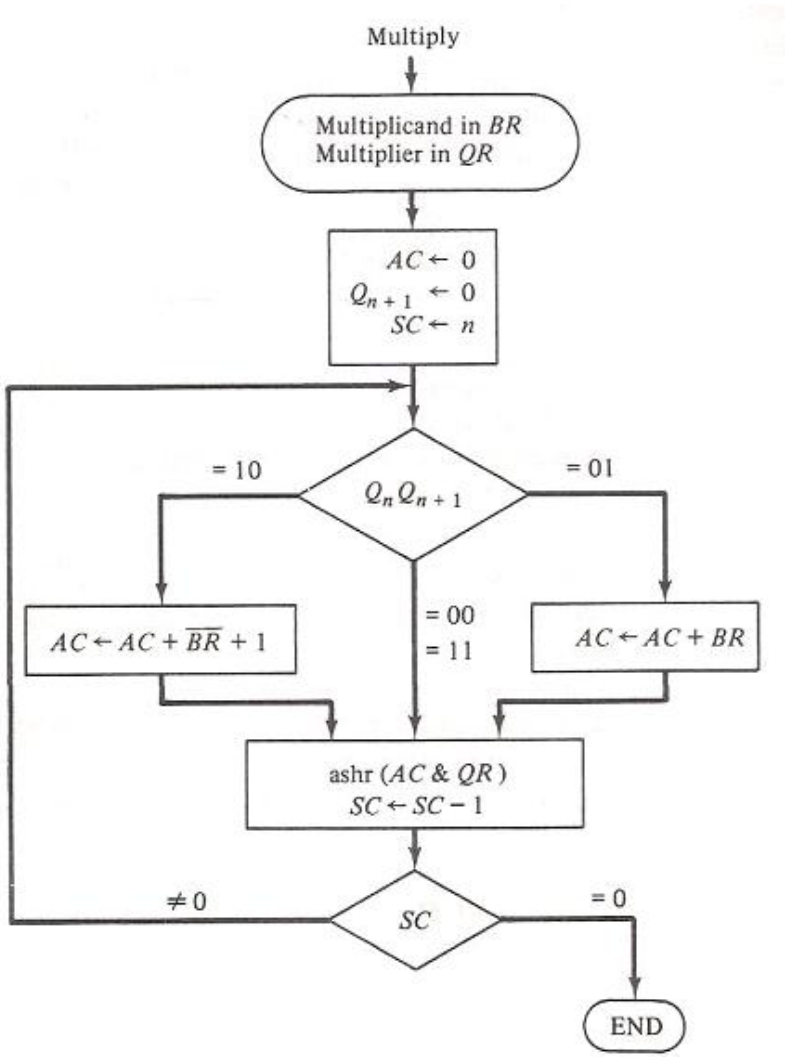
i). If the two bits are equal to 10 subtract the multiplicand from the partial product in AC.

ii). If the two bits are equal to 01, add the multiplicand to the partial product in AC.

iii). If the two bits are equal, the partial product does not change.

2). The next step is to shift right the partial product and the multiplier (including bit Qn+1).

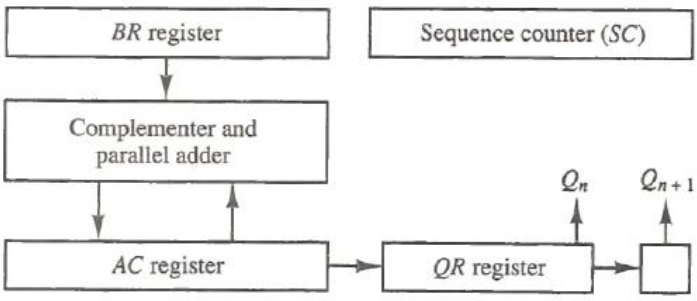
This is an arithmetic shift right (ashr) operation which shifts AC and QR to the right and leaves the sign bit in AC unchanged.



3). The sequence counter is decremented and the computational loop is repeated n times.

6.5.1 The hardware implementation of Booth algorithm:

It consists the register configuration shown in Fig. The sign bits are not separated from the rest of the registers. We rename registers A, B, and Q, as AC, BR, and QR, respectively.



Qn designates the least significant bit of the multiplier in register QR.

An extra flip-flop Qn+1 is appended to QR to facilitate a double bit inspection of the multiplier.

6.5.2 Booth Multiplication Algorithm:

The algorithm was invented by Andrew Donald Booth in 1950. Booth algorithm gives a procedure for multiplying binary integers in signed 2's complement representation.

The Advantage of Booth's algorithm over Unsigned Binary Multiplication is it facilitates the process of multiplying signed numbers.

It requires examination of the multiplier bits and shifting of the partial product.

1. The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier.

2. The multiplicand is added to the partial product upon encountering the first 0 (*provided that there was a previous 1*) in a string of 0's in the multiplier.

3. The partial product does not change when the multiplier bit is identical to the previous multiplier bit.

Ex: Multiplication with booth algorithm

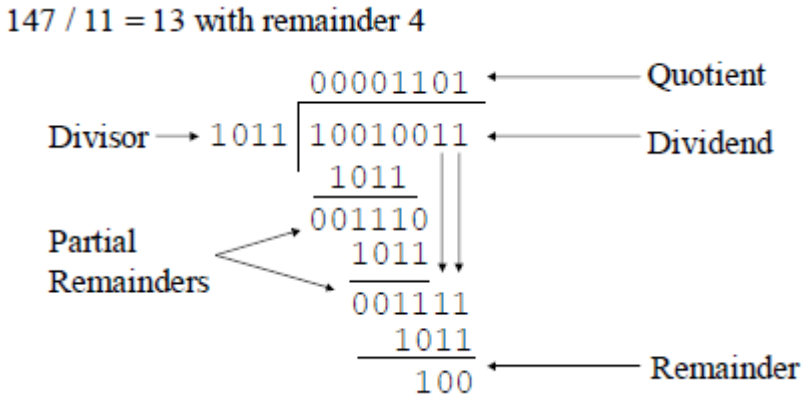
A numerical example of Booth algorithm is shown in Table for n = 5. It shows the step-by-step multiplication of (-9) x (-13) = +117. Note that the multiplier in QR is negative and that the multiplicand in BR is also negative. The 10-bit product appears in AC and QR and is positive. The final value of Qn+1 is the original sign bit of the multiplier and should not be taken as part of the product.

BR = 10111(-9); BR* +1 = 01001, QR = 10011 (-13) multiplier.

Qn	Qn+1		AC	QR	Qn+1	SC
		Initial	00000	10011	0	101
1	0	Subtract BR	01001			
			01001			
		Ashr	00100	11001	1	100
1	1	Ashr	00010	01100	1	011
0	1	Add BR	10111			
			11001	01100	1	
		Ashr	11100 *	10110	0	010
0	0	Ashr	11110	01011	0	001
1	0	Subtract BR	01001			
			00111	01011	0	
		Ashr	00011	10101	1	000
				117		

* Arithmetic right shift operation (ashr) shifts AC and QR to the right and leaves the sign bit in AC unchanged.

6.6 Division Example:



6.6.1 Restoring Method: Using same registers (A,M,Q, count) as multiplication. Results of division are stored in quotient (Q) and remainder (A):

Initial values

* Q← 0
* A ← Dividend
* M or B ← Divisor
* Sequence Counter ← n

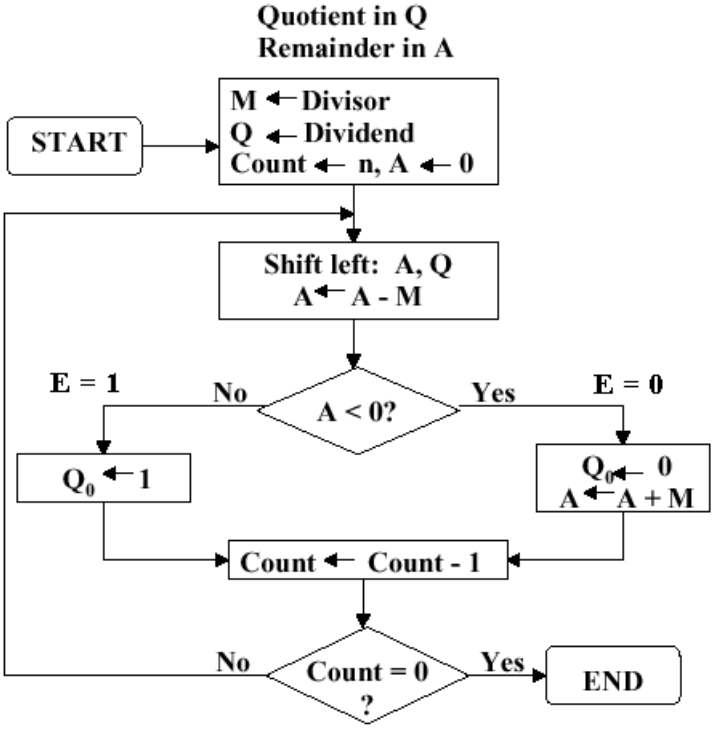
Steps:

1. Shift left Both A & Q.
2. Subtract Divisor (M) from Dividend (A).
3. Check the value of the Dividend (A).

A > 0, Add 1 to quotient (Q_0)

A< 0, Add 0 to quotient (Q_0) and ADD Divisor M to Dividend A.

4. Decrement the counter (SC-1).



6.6.2 Other Division Methods:

i). Comparison Method: In the comparison method A and B are compared prior to the subtraction operation. Then if A > B, B is subtracted from A. If A < B nothing is done. The partial remainder is shifted left and the numbers are compared again. The comparison can be determined prior to the subtraction by inspecting the end-carry out of the parallel-adder prior to its transfer to register E.

ii). Non-restoring Method: In the non restoring method, B is not added if the difference is negative but instead, the negative difference is shifted left and then B is added.

6.7 Division Algorithms: Division of two fixed-point binary numbers in signed-magnitude representation is done by successive compare, shift, and subtract operations as illustrated below.

Process:

1. Place the dividend in A (reminder), Q (quotient - all zeros) and set the SC no.
2. Shift left EAQ
3. Subtract the divisor from the remainder.
4. Check the remainder;

* If E = 1 then the remainder is >= 0, set the quotient rightmost bit to 1.
* If E = 0 then the remainder is < 0, Restore the original value by adding the divisor and set the quotient rightmost bit to 0.

4. Repeat the process till the SC reaches to 0.

Example. The divisor B consists of five bits (10001) and the dividend A (01110 00000).

If E = 1, it signifies that $A \geq B$. A quotient bit 1 is inserted into Q_n and the partial remainder is shifted to the left to repeat the process.

If E = 0, it signifies that $A < B$ so the quotient in Q_n remains a 0. The value of B is then added to restore the partial remainder in A to its previous value.

The partial remainder is shifted along with quotient bits to the left and the process is repeated again until all five quotient bits are formed.

Sign: If the two signs (dividend and the divisor) are alike, the sign of the quotient is plus. If they are unalike, the sign is minus. The sign of the remainder is the same as the sign of the dividend.

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

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

6.7.1 Divide Overflow:

A divide-overflow condition occurs if the High order half bits of the dividend constitute a number greater than or equal to the divisor. Another problem is division by zero must be avoided.

The divide-overflow condition is usually detected when a divide-overflow flip-flop (DVF) is set.

6.8 Floating Point Arithmetic Operations:

A floating point number in computer registers consists of two parts: a mantissa *m* and an exponent *e*. The two parts represent a number obtained from multiplying *m* times a radix r raised to the value of e; thus $\boldsymbol{m \times r^e}$

Alignment of Mantissas: Adding or subtracting two numbers requires first an alignment of the radix point since the exponent parts must be made equal before adding or subtracting the mantissas. The alignment is done by shifting one mantissa while its exponent is adjusted until it is equal to the other exponent. Consider the sum of the following floating-point numbers:

$.5372400 \times 10^2 + .1580000 \times 10^{-1}$

We can either shift the first number three positions to the left, or shift the second number three positions to the right. When the mantissas are stored in registers, shifting to the left causes a loss of most significant digits. Shifting to the right causes a loss of least significant digits. The second method is preferable because it only reduces the accuracy, while the first method may cause an error.

Usually the mantissa that has the smaller exponent is shifted to right till the exponents are equal. After this, the mantissas can be added:

$.5372400 \times 10^2 + .0001580 \times I0^2 = .5373980 \times 10^2$

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

$.56780 \times 10^5 - .56430 \times 10^5 = .00350 \times 10^5$

Underflow: A floating-point number that has a 0 in the most significant position of the mantissa is said to have an *underflow*. To normalize a number that contains an underflow, it is necessary to shift the mantissa to the left and decrement the exponent until a nonzero digit appears in the first position.

In the example above, $.00350 \times 10^5$ it is necessary to shift left twice to obtain $.35000 \times 10^3$.

Floating-point multiplication and division do not require an alignment of the mantissas. The product can be formed by multiplying the two mantissas and adding the exponents. Division is accomplished by dividing the mantissas and subtracting the exponents.

The *biased* exponent: In this representation, the sign bit is removed from being a separate entity. The advantage of biased exponents is that they contain only positive numbers.

Exponent: Excess representation: actual exponent + Bias, Ensures exponent is unsigned
Single precision: Bias = 127 and Double precision: Bias = 1203

For a single-precision number, an exponent in the range −126 .. +127 is biased by adding 127 to get a value in the range 1 .. 254 (0 and 255 have special meanings).

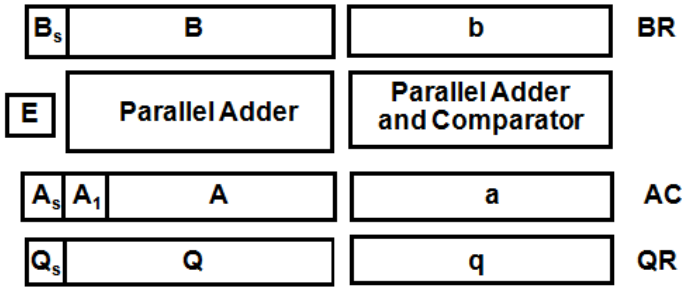
For a double-precision number, an exponent in the range −1022 .. +1023 is biased by adding 1023 to get a value in the range 1 .. 2046 (0 and 2047 have special meanings).

6.9 The register (Hardware) organization:

* There are three registers, BR, AC, and QR. Each register is subdivided into two parts. The mantissa part (uppercase letter) and the exponent part (lowercase letter).

It is assumed that each floating-point number has a mantissa in signed magnitude representation and a biased exponent. Thus the AC has a mantissa whose sign is in As and a magnitude that is in A. The exponent is in the part of the register denoted by the lowercase letter symbol a. The diagram shows explicitly the most significant bit of A, labelled by A_1.

The bit in this position must be a 1 for the number to be normalized. Note that the symbol AC represents the entire register, that is, the concatenation of As, A, and a.



Similarly, register BR is subdivided into BS, B and b, and QR into Qs, Q, and q.

* A parallel-adder adds the two mantissas and transfers the sum into A and the carry into E. A separate parallel-adder is used for the exponents.

* Since the exponents are biased, they do not have a distinct sign bit but are represented as a biased positive quantity.

It is assumed that the floating-point numbers are so large that the chance of an exponent overflow is very remote, and for this reason the exponent overflow will be neglected.

The exponents are also connected to a magnitude comparator that provides three binary outputs to indicate their relative magnitude.

The number in the mantissa will be taken as a fraction, so the binary point is assumed to reside to the left of the magnitude part.

To avoid these problems, we adopt a fraction representation. The numbers in the registers are assumed to be initially normalized. After each arithmetic operation, the result will be normalized.

Thus all floating-point operands coming from and going to the memory unit are always normalized.

6.10 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:

i). Check for zeros.
ii). Align the mantissas.
iii). Add or subtract the mantissas.
iv). Normalize the result.

i). **Check for zeros**:

If BR is equal to zero, the operation is terminated, with the value in the AC being the result. If AC is equal to zero, we transfer the content of *BR* into *AC* and also complement its sign if the numbers are to be subtracted. If neither number is equal to zero, we proceed to align the mantissas.

ii). **Align the mantissas**:

The magnitude comparator attached to exponents *a* and *b* provides three outputs.

a) a = b : If the two exponents are equal, we go to perform the arithmetic operation.

b) a < b or a> b : If the exponents are not equal, the mantissa having the smaller exponent is shifted to the right and its exponent incremented. This process is repeated until the two exponents are equal.

iii). **Add or subtract the mantissas**:

The addition and subtraction of the two mantissas is identical to the fixed-point addition and subtraction algorithm. The magnitude part is added or subtracted depending on the operation and the signs of the two mantissas.

Overflow: If an overflow occurs, it is transferred into flip-flop E. If E is equal to 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.

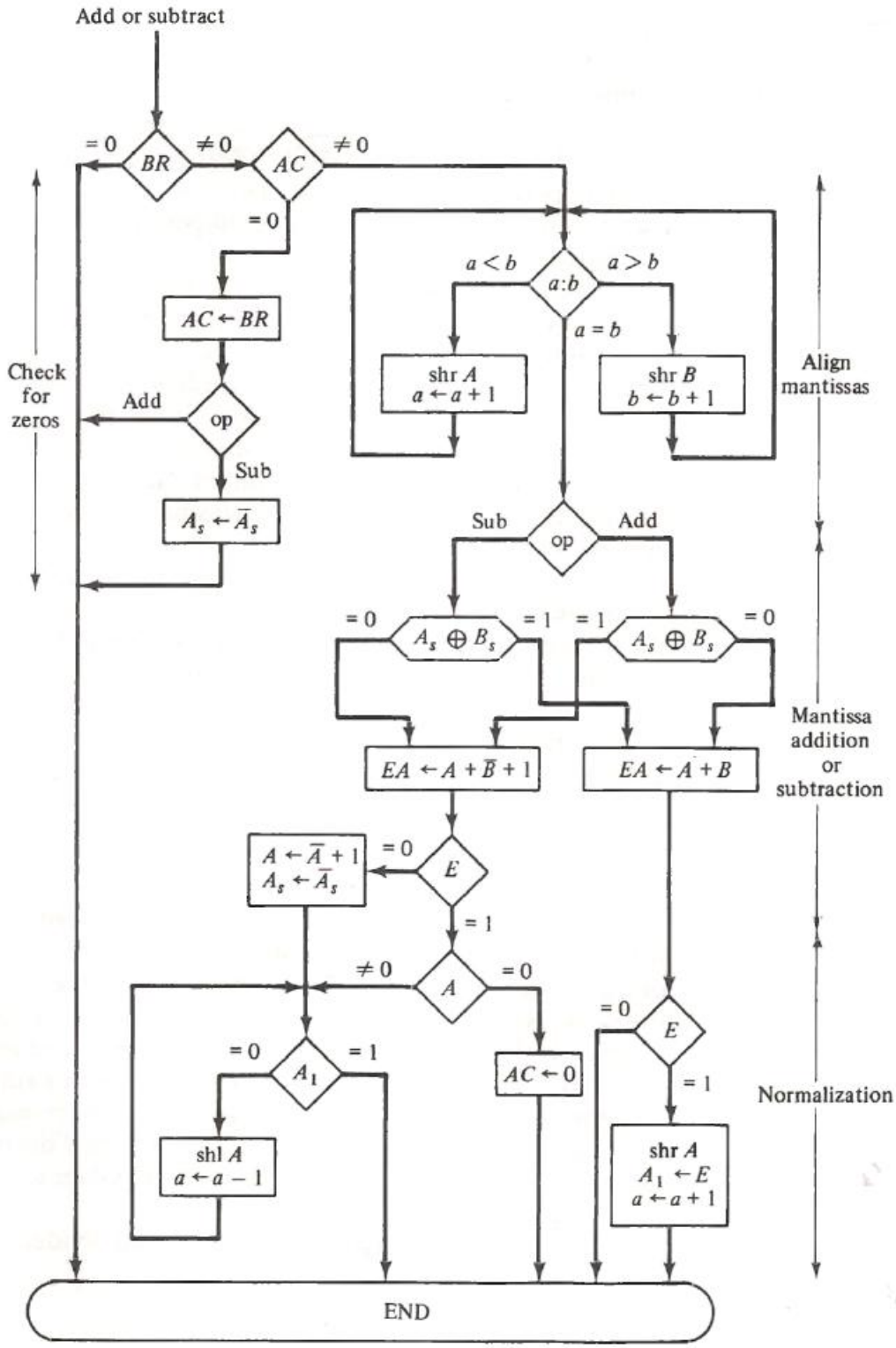
Underflow: No underflow may occur in this case because the original mantissa that was not shifted during the alignment was already in a normalized position.

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

iv). Normalisation:

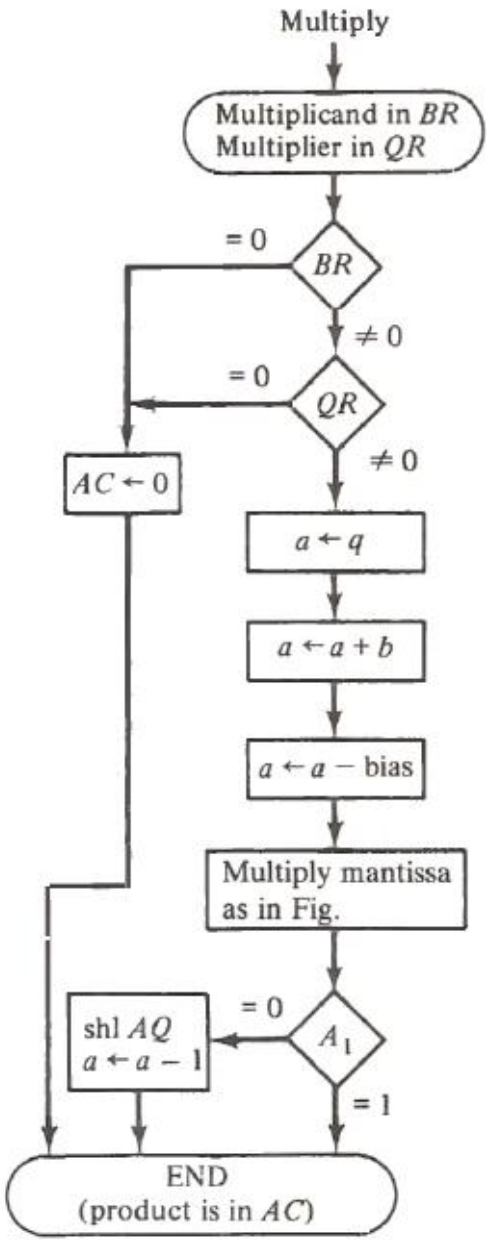
The mantissa has an underflow if the most significant bit in position A_1 is 0. In that case, the mantissa is shifted left and the exponent decremented. The bit in A_1 is checked again and the process is repeated until it is equal to 1.

When $A_1 = 1$, the mantissa is normalized and the operation is completed. The flowchart for adding or subtracting two floating-point binary numbers is shown in Fig.



6.11 Multiplication:

The multiplication of two floating-point numbers requires that we multiply the mantissas and add the exponents. No comparison of exponents or alignment of mantissas is necessary.



The multiplication of the mantissas is performed in the same way as in fixed-point to provide a double-precision product. In floating-point, the range of a single-precision mantissa combined with the exponent is usually accurate enough so that only single precision numbers are maintained.

The multiplication algorithm can be subdivided into four parts:

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

Steps 2 and 3 can be done simultaneously if separate adders are available for the mantissas and exponents.

Flowchart: Check both the operands. If either operand is equal to zero, the product in the *AC* is set to zero and the operation is terminated. If neither of the operands is equal to zero, the process continues with the exponent addition.

The multiplier exponent which is in *q* is transferred to a and then the multiplicand exponent *b is added to a.*

Since both exponents are biased by the addition of a constant, the exponent sum will have double this bias. The correct biased exponent for the product is obtained by subtracting the bias number from the sum.

The multiplication of the mantissas is done as in the fixed-point case with the product residing in *A* and *Q*. Overflow cannot occur during multiplication, so there is no need to check for it.

The product may have an underflow, so the most significant bit in *A* is checked. If it is a 1, the product is already normalized. If it is a 0, the mantissa in *AQ* is shifted left and the exponent decremented. Note that only one normalization shift is necessary.

Although the low-order half of the mantissa is in Q, we do not use it for the floating-point product. Only the value in the *AC* is taken as the product.

6.12 Division:

Floating-point division requires that the exponents be subtracted and the mantissas divided. In fraction representation, a single-precision dividend is placed in register *A* and register *Q* is cleared. **Divide overflow**: with floating-point numbers the divide-overflow imposes no problems.

Dividend alignment: If the dividend is greater than or equal to the divisor, the dividend fraction is shifted to the right and its exponent incremented by 1. For normalized operands this ensures that no mantissa divide overflow will occur.

The division algorithm can be subdivided into five parts:
1. Check for zeros.
2. Initialize registers and evaluate the sign.
3. Align the dividend.
4. Subtract the exponents.
5. Divide the mantissas.

If the divisor is zero, the operation is terminated. If the dividend in *AC* is zero, the quotient in *QR* is made zero and the operation terminates.

The sign of the quotient is stored in *Q*s. The sign of the dividend in *As* is left unchanged to be the sign of the remainder. The Q register is cleared and the sequence counter SC is set to the number of bits in the quotient.

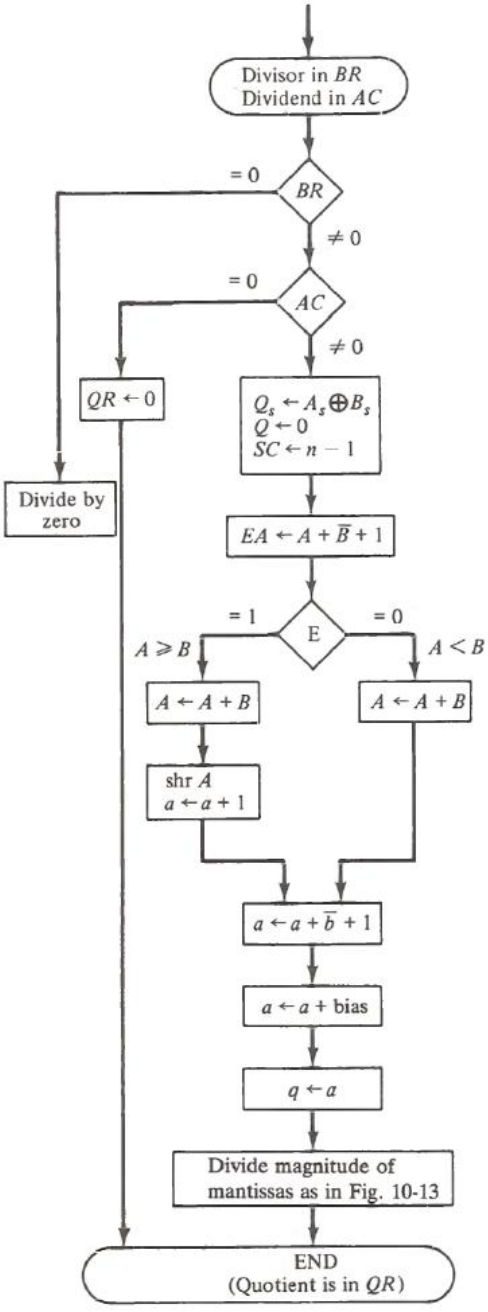
The dividend alignment requires that the fraction dividend be smaller than the divisor. The two fractions are compared by a subtraction test. The carry in E determines their relative magnitude. The dividend fraction is restored to its original value by adding the divisor.

If $A \geq B$, it is necessary to shift A once to the right and increment the dividend exponent. Since both operands are normalized, this alignment ensures that $A < B$.

Next, the divisor exponent is subtracted from the dividend exponent. Since both exponents were originally biased, the subtraction operation gives the difference without the bias.

The bias is then added and the result transferred into q because the quotient is formed in *QR*.

The magnitudes of the mantissas are divided as in the fixed-point case. After the operation, the mantissa quotient resides in *Q* and the remainder in *A*.



The floating-point quotient is already normalized and resides in *QR*. The exponent of the remainder should be the same as the exponent of the dividend.

6.13 A BCD adder:

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.

(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.)

To add 0110 to the binary sum, we use a second 4-bit binary adder as shown in the 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.

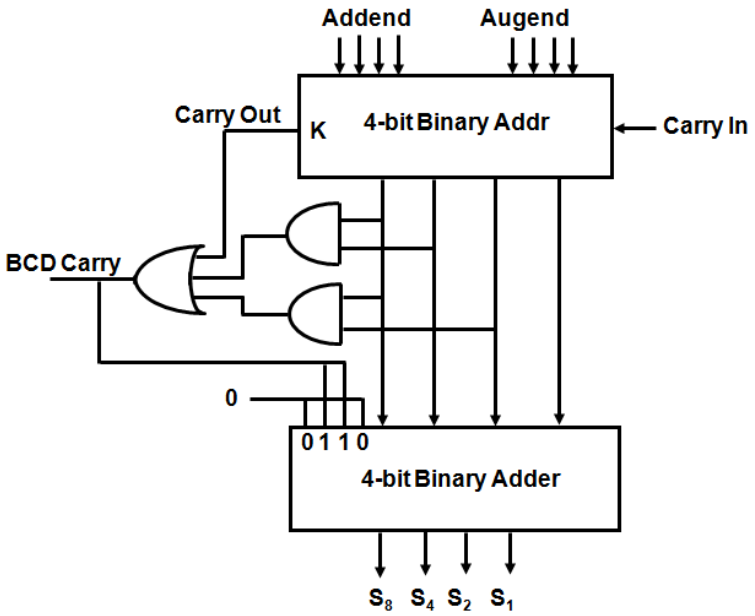
When the output-carry is equal to 0, nothing is added to the binary sum.

Binary Sum					BCD Sum					
K	Z8	Z4	Z2	Z1	C	S8	S4	S2	S1	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 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.



6.14 BCD Subtraction:

It is more economical to perform the subtraction by taking the 9's or 10's complement of the subtrahend and adding it to the minuend. The 9's complement of a decimal digit represented in BCD may be obtained by complementing the bits in the coded representation of the digit provided a correction is included.

There are two possible correction methods:

In the first method, binary 1010 (decimal10) is added to each complemented digit and the carry discarded after each addition.

Ex: As a numerical illustration, the 9's complement of BCD 0111 (decimal 7) is computed by first complementing each bit to obtain 1000. Adding binary 1010 and discarding the carry, we obtain 0010 (decimal 2).

In the second method, binary 0110 (decimal 6) is added before the digit is complemented.

Ex: add 0110 to 0111 to obtain 1101. Complementing each bit, we obtain the required result of 0010.

The 9's complement of a BCD digit can also be obtained through a combinational circuit.

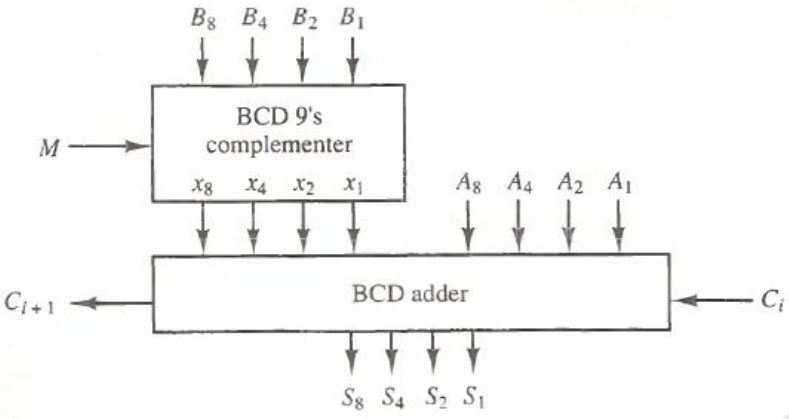
6.14.1 One stage of a decimal arithmetic unit:

One stage of a decimal arithmetic unit that can add or subtract two BCD digits is shown in the Fig. It consists of a BCD adder and a 9's complementer.

The mode *M* controls the operation of the unit. With M = 0, the *S* outputs from the sum of A and B. With *M* = 1, the *S* outputs from the sum of A plus the 9's complement of B.

The best way to subtract the two decimal numbers is to let M = 1 and apply a 1 to the input carry C_i of the first stage.

The outputs will form the sum of A plus the 10's complement of B, which is equivalent to a subtraction operation if the carry-out of the last stage is discarded.



6.15 Questions:

1. Draw and explain the hardware for integer addition and subtraction.
2. Draw and explain the hardware for parallel Adder / Subtractor with overflow.
3. Draw a flow chart which explains multiplication of two signed magnitude fixed point numbers.
4. Multiply 10101 and 10111 with the above procedure.
5. Explain Booths Multiplication with example
6. Explain a Hardware algorithm for 'Divide' operation with a neat flow chart.
7. Explain the Overflow, Underflow, Bias and Normalisation in floating point arithmetic.
8. What is overflow and underflow? What are the reasons for overflow and underflow?
9. Draw a flow chart for Floating point Add/subtract operations
10. Draw and explain the flowchart for floating point addition.
11. Draw and explain the flowchart for floating point division.
12. Explain BCD adder.
13. Subtract $(101011)_2$ from $(111001)_2$ using the 1's complement and 2's complement method.

Multiple choice questions:

1. -14 is represented in signed magnitude as …….
a). 1000 1110 b) 1111 0001 c). 1111 0010 d). 0101 1111
2. The add-overflow flip-flop *AVF* holds the ..... bit
a) sign bit b) overflow bit c) mode bit d) complementary bit
3. In add and subtract operations the two signs As and Bs are compared by using .....
a). NAND gate b) NOR gate c). XOR gate d) AND gate.
4. In BCD adder when the binary sum is greater than 1001, which binary number is added for correct BCD representation .....
a) 1000 b) 1001 c) 1010 d) 1101
5. Alignment of mantissas is done by increasing or decreasing the ……. value
a) Mantissa b) Radix c) Exponential d) Base value.

Fill in the blanks:

1. Find the arithmetic result for : (- 23) - (-32) .....
2. Booth algorithm is used for multiplication of ..... numbers
3. In addition and subtraction algorithm the final result sign bit is based on .....
4. Define underflow in floating point operations .....
5. How normalisation is done in floating point operations .....

This page intentionally left blank.

7. INPUT-OUTPUT ORGANIZATION

7.1 Input Output (I/O) Interface:

Input Output (I/O) interface provides a method for transferring information between internal storage and external I/O devices. The purpose of the interface is to resolve the differences that exist between the computer and each peripheral. The major differences are:

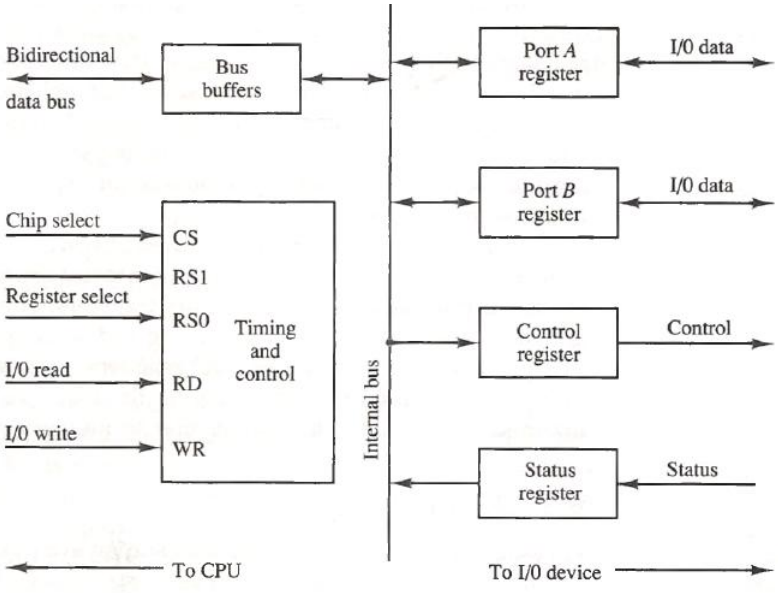
1. The computer is a digital system where the peripherals are either electromagnetic or electro mechanical devices.
2. The CPU data transfer rate is very high where as the peripheral data transfer is slow.
3. Data codes and formats in peripherals are different from the CPU word format.
4. The operating modes of peripherals are different to each other.

Each peripheral have its own interface controller to resolve these differences and to synchronise all input and output transfers in between the computer and the peripherals.

Example of an I/O interface unit:

It consists of two data registers called *ports*, a control register, a status register, bus buffers, and timing and control circuits. The interface communicates with the CPU through the data bus.

The chip select and register select inputs determine the address assigned to the interface.



The I/O read and write are two control lines that specify an input or output, respectively. The four registers communicate directly with the I/O device attached to the interface.

The address bus selects the interface unit through the chip select (CS) and the two register select inputs.

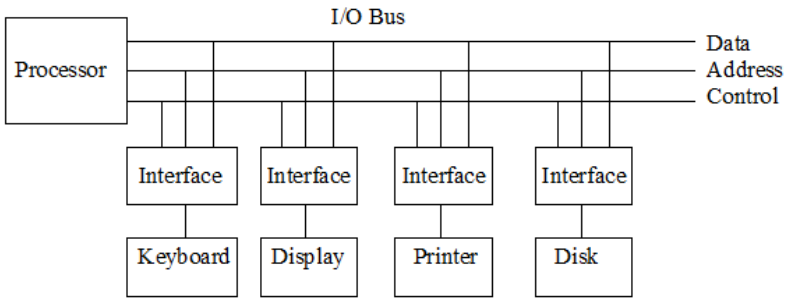
These inputs select one of the four registers in the interface and the CPU transfers the binary information accordingly via the data bus.

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

7.2 I/O Bus and Interface Modules:

The I/O bus consists of data lines, address lines, and control lines generated by the processor and attached to all peripheral interfaces.

Fig. Connection of I/O bus to Input Output devices:



To communicate with a particular device, the processor places a device address on the address lines.

When the interface detects its own address it activates the path between the bus lines and the device that it controls.

7.2.1 I/O versus memory Bus:

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

i). Use two separate buses, one for memory and the other for I/O. *(using IOP)*
ii). Use one common bus for both memory and I/O but have separate control lines for each. *(isolated I/O)*
iii). Use one common bus for memory and I/O with common control lines. *(Memory mapped I/O)*

7.2.2 Isolated versus Memory mapped I/O:

a). Isolated I/O method*:* Many computers use one common bus to transfer information between memory or I/O and the CPU.

In case of Isolated I/O the CPU specifies whether the address on the address lines is for a memory word or for I/O interface register by enabling one of two possible read or write lines.

IOR (I/O read), IOW(I/O write) & MEMR (Memory Read), MEMW(Memory Write)

The isolated I/O method isolates memory and I/O addresses so that memory address values are not affected by interface address assignment since each has its own address space.

b). Memory mapped I/O*:* In memory mapped I/O computers employ only one set of read and write signals and do not distinguish between memory and I/O addresses.

There are no specific input or output instructions. The CPU manipulate I/O data residing in interface registers with the same instructions that are used to manipulate memory words.

Disadvantage is the assigned addresses for interface registers cannot be used for memory words, which reduces the memory address range available.

7.3 Asynchronous Data Transfer:

Digital systems operate based on the clock pulses provided by the clock generator.

Synchronous: If the registers in the interface share a common clock with the CPU registers, the transfer between the two units is said to be synchronous.

Asynchronous: If the internal timing of each unit is independent from the other unit then they are said to be asynchronous to each other.

Asynchronous data transfer requires control signals to indicate the time at which data is being transmitted. For this either of the below method is used.

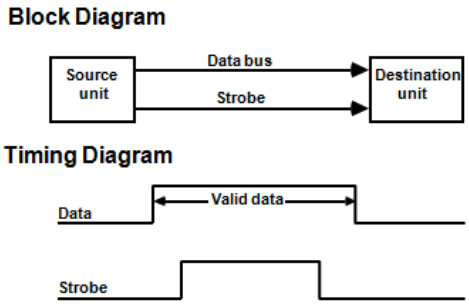
i). Strobe Method
ii). Handshaking method.

7.3.1 Strobe Method:

In the strobe control method the strobe control signal is activated either by the source or destination for every data transfer.

a). Source Initiated Strobe for Data Transfer:

The fig illustrates source initiated strobe for data transfer. The source unit places the data on data bus. Once data is settled the source activates the strobe pulse.

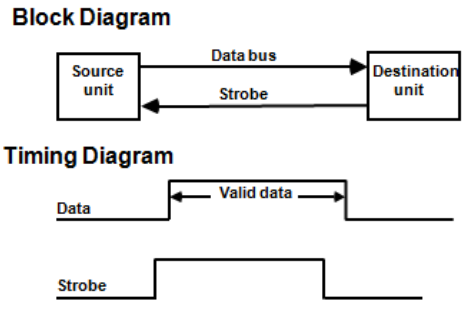


Often the destination unit uses the falling edge of the strobe pulse to transfer the contents of the data bus into one of its registers.

The same process is to be repeated for each and every data transfer.

b). Destination Initiated Strobe:

The fig illustrates destination initiated strobe for data transfer.



In this case the destination unit activates the strobe pulse, informing the source to provide the data. The source unit responds and places the data on data bus.

The destination unit receives the data and disables the strobe pulse.

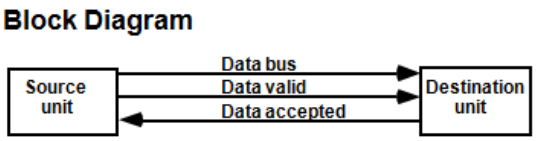
The same process is to be repeated for each and every data transfer.

7.3.2 Handshake Method:

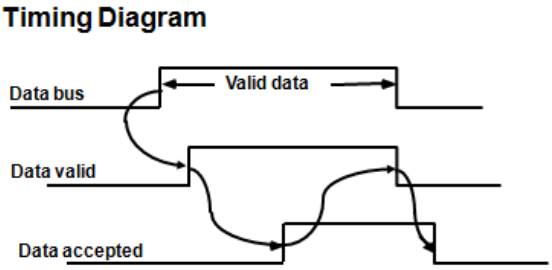
The main disadvantage in the strobe method is either the source or the destination does not have any knowledge, whether the other end unit received or placed the data on the bus.

a). Source-initiated data transfer using handshake:

The hand shake method solves this problem by adding a second control signal that provides an acknowledgement to the unit that initiates the data transfer.



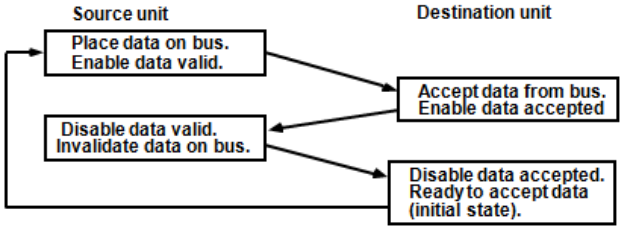
The fig illustrates the two hand shaking control signals **data valid**, which is generated by the source unit and **data accepted** generated by the destination unit.



The timing diagram shows the exchange of signals between the two units.

The source unit initiates the data transfer by placing the data on the bus and enabling its *data valid* signal.

The *data accepted* signal is activated by the destination unit after it accepts the data from the bus.

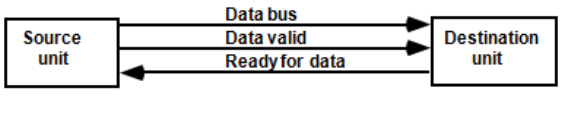


The source unit then disables its *data valid* signal.

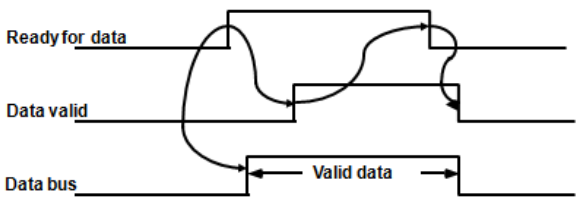
The destination unit disables the *data accepted* signal. For the next data transfer the same procedure is followed.

b). Destination-initiated transfer using handshake:

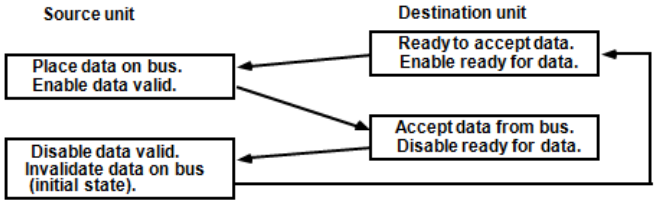
In the destination unit data transfer the destination raises the signal *ready for data.*



Once the source receives the *ready for data* signal from the destination it places the data on the data bus and enable its data valid signal.



Once the destination receives the data from the data bus it disables the *ready for data* signal, followed by the source which disable the *data valid* signal and removes the data from the data bus.



7.4 Asynchronous Serial Transfer:

Serial transmission can be synchronous or asynchronous.

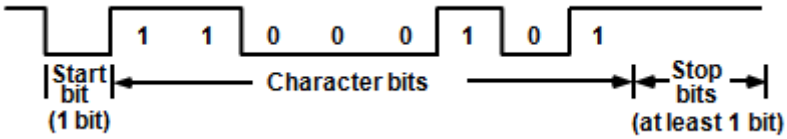
Synchronous: In synchronous transmission, the two units share a common clock frequency and bits are transmitted continuously at the rate dictated by the clock pulses.

In long distant serial transmission. Each unit is driven by a separate clock of the same frequency. Synchronization signals are transmitted periodically between the two units to keep their clocks in step with each other.

Asynchronous: In asynchronous transmission, binary information is sent only when it is available and the line remains idle when there is no information to be transmitted.

A serial asynchronous data transmission technique each character consists of three parts: a start bit, the character bits, and stop bits. The convention is that the transmitter rests at the 1-state when no characters are transmitted.

The first bit, called the start bit, is always a 0 and is used to indicate the beginning of a character. The last bit called the stop bit is always a 1 as shown in the below figure.



A transmitted character can be detected by the receiver from knowledge of the transmission rules:

1. When a character is not being sent, the line is kept in the 1-state.

2. The initiation of a character transmission is detected from the start bit, which is always 0.

3. The character bits always follow the start bit. The receiver knows the transfer rate of the bits and the number of character bits to accept.

4. After the last bit of the character is transmitted, a stop bit is detected when the line returns to the 1-state for at least one bit time.

The line remains in the 1-state until another character is transmitted. The stop time ensures that a new character will not follow for one or two bit times.

baudrate or bps: The units for serial communication is bits per second (bps) or baud or baudrate. Ex: 9600 bps or baudrate.

UART: Universal Asynchronous Receiver Transmitter (UART) is the integrated circuit device normally used in serial communication.

7.5 Asynchronous Communication Interface:

Asynchronous communication interface includes both a transmitter and a receiver.

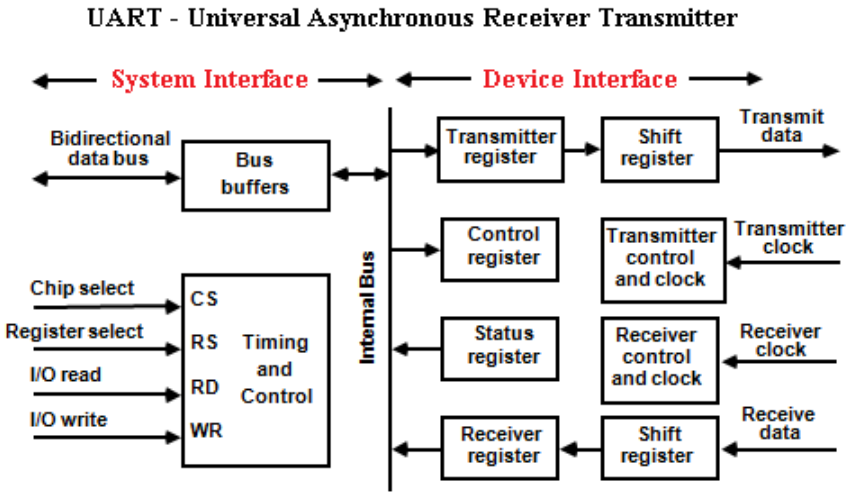
Control Register: The interface is initialized for a particular mode of transfer by means of a control byte that is loaded into its control register.

Status Register: Two bits are used for input and output flags to indicate EMPTY or FULL. The CPU can read the status register to check if any errors have occurred. Three possible errors are parity error, framing error, and overrun error. Port A and Port B are bidirectional ports and can be configured either for transmission or reception.

Transmitter: The CPU checks the flag in the status register. If it is empty, the CPU transfers a character to the transmitter register and the interface clears the flag to mark the register full.

Start bit (0) and the appropriate number of stop bits are appended into the shift register. The transmitter register is then marked empty. This cycle is repeated for every character.

Receiver: Once the receiver detects the start bit, the incoming bits of the character are shifted into the shift register.



After receiving the data bits, the interface checks for the parity and stop bits.

The character without the start and stop bits is then transferred in parallel from the shift register to the receiver register.

The chip select (CS) input is used to select the interface through the address bus. The register select (RS) is associated with the read (RD) and write (WR) controls.

Initialisation: The operation of the asynchronous communication interface is initialized by the CPU by sending a byte to the control register. The initialization procedure places the interface in a specific mode of operation as it defines:

CS	RS	Operation	Register Selected
0	X	X	None (High impedence state)
1	0	WR	Transmitter register
1	1	WR	Control Register
1	0	RD	Receiver Register
1	1	RD	Status Register

- the baud rate,
- how many bits are in each character,
- whether to generate and check parity, and
- how many stop bits are appended to each character.

The character can now be transmitted one bit at a time by shifting the data in the shift register at the specified baud rate. The CPU can transfer another character to the transmitter register after checking the flag in the status register.

7.6 Modes of Transfer:

Data transfer between the computer and I/O devices may be handled in a variety of modes. Some modes use the CPU as an intermediate path. The three possible modes are:

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

i). Programmed I/O mode:

In Programmed I/O operations each data item is initiated by an instruction in the program. Usually the transfer is to and from a CPU register and peripheral. For Example if a data has to move from memory to peripheral it has to pass through the CPU. Here CPU is used as an intermediate path.

The disadvantage in this method is the CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process since it keeps the processor busy needlessly.

ii). Interrupt initiated I/O:

In Interrupt initiated I/O method the interface monitors the device, when the device is ready with the data to transfer, it generates the interrupt request to the CPU.

Upon detecting the external interrupt request the CPU momentarily stops the task it is processing, process the I/O transfer and then return back to the task it was originally performing. This avoids time consuming as it was done in programmed I/O.

iii). Direct memory access (DMA):

In DMA, the interface transfers data into and out of the memory through memory bus. The CPU initiates the transfer by supplying the interface with the starting address and the number of words to be transferred and then proceeds to execute other tasks.

After this transfer the DMA requests the memory controller and transfers the data directly to the memory. The CPU merely delays its memory access operation to allow the direct memory I/O transfer.

7.7 Priority Interrupts:

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

Generally higher priority is given to the devices with high speed data transfers such as storage disks and low priority to low speed devices like keyboard. Establishing the priority of simultaneous interrupts can be done by software and hardware.

7.7.1 Polling (Software Method): Polling is a process of continuous checking the connected devices by a program (or device) to see whether they want to communicate.

A *polling* procedure is used to identify the highest priority source by software means. The order in which they are tested determines the priority of each interrupt. The highest priority source is tested first, followed by next priority and so on.

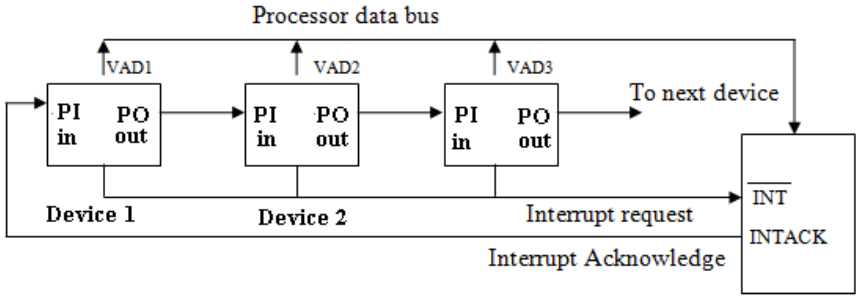
The disadvantage of the software method is that if there are many interrupts, the time require to poll them exceed the time available to service the I/O device.

7.7.2 Hardware priority interrupt unit: The hardware priority interrupt unit determines which unit has the highest priority and issues an interrupt requests to the computer.

This can be established either by a serial or a parallel connection of interrupt lines. The serial connection is also known as the daisy chaining method.

i). Daisy chaining priority: Daisy chain method is a hardware priority function with serial connection. All the devices that requests interrupt are connected serially. The device with the highest priority is placed in the first position of the chain, followed by next highest priority device and so on.

The least priority device is placed last in the chain. The below fig. illustrates the connections using daisy chain method.



By default the interrupt request input to the CPU is in high state.

When a device needs the service of the CPU it raises the interrupt input by making the interrupt request line low. The CPU responds to the interrupt request by enabling the interrupt acknowledge line to the highest priority device (device1).

When the interrupt acknowledgement signal is received by the device at its P1 (priority in) input it has two options.

1. Either it can process its pending interrupt or
2. Pass on to the next priority device.

1. If device 1 has a pending interrupt it blocks the acknowledgement signal by placing 0 at its P0 output and inserts its own vector address (VAD) into the data bus for the CPU to use the interrupt cycle.

2. If the device does not have pending interrupts it transmits the acknowledgement signal to the next priority device by placing a 1 in its PO output. This process is continued till the last priority device.

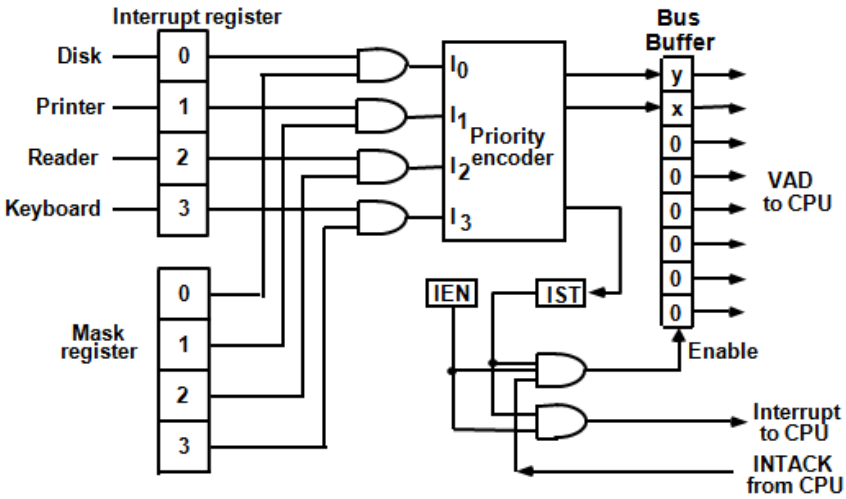
ii). Parallel priority Interrupt:

In this method, Priority is established according to the position of the bits in the register.

The priority logic consists of an interrupt register whose individual bits are set by external conditions. High speed device (disk drive) given the highest priority and slow device keyboard low priority.

In addition it also includes a mask register whose purpose is to control the status of each interrupt request.

The mask register can be programmed to disable lower priority interrupts while a higher priority device is being serviced.



It can also provide a facility that allows a high priority device to interrupt the CPU while a lower priority device is being serviced.

A mask register has the same number of bits as the interrupt register and these bits can be set (1) or reset (0) by the software instructions. Both the interrupt bits and corresponding mask bits are ANDed to produce four inputs to a priority encoder. The interrupt is recognized only if its corresponding mask bit is set to1.

The priority encoder generates two bits of the vector address, which is transferred to the CPU and also sets an interrupt status flip flop IST when an interrupt that is not masked occurs.

The interrupt enable flip flop IEN can be set or reset by the program to provide an overall control over the interrupt system.

The outputs of IST ANDed with IEN provide a common interrupt signal for the CPU. The interrupt acknowledge INTACK signal from the CPU enables the bus buffers in the output register and a vector address VAD is placed into the data bus.

Priority encoder:

The priority encoder is a circuit that implements the priority function.

When two or more inputs arrive at the same time the device having highest priority will take precedence. The table illustrates this.

Inputs				Outputs			Boolean functions
I_0	I_1	I_2	I_3	X	Y	IST	
1	x	x	x	0	0	1	x= *I_0 *I_1
0	1	x	x	0	1	1	y= *I_0 I_1 + *I_0 *I_2
0	0	1	x	1	0	1	
0	0	0	1	1	1	1	(IST) =I_0+I_1+I_2+I_3
0	0	0	0	x	x	0	

7.8 Bus arbitration:

Bus arbitration is the process of determining which competing bus master will be permitted access to the bus. The selection of bus master is usually done on the priority basis. Bus arbitration schemes usually try to balance:

i). Bus priority: The highest priority device should be serviced first
ii). Fairness: Even the lowest priority device should never be completely locked out from the bus

There are two approaches to bus arbitration: i). Centralized and ii). Distributed.

i) Centralised a). Daisy chain arbitration
b). Centralized, parallel arbitration

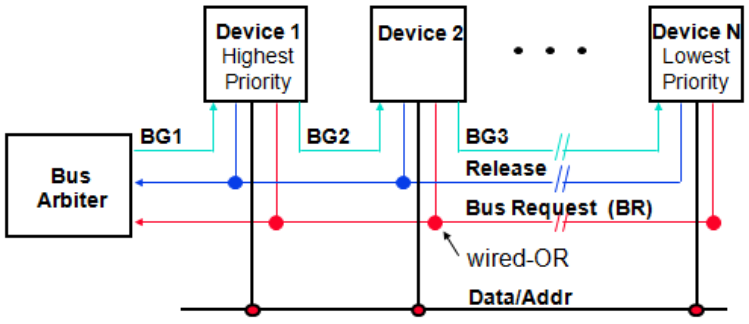
ii) Distributed c). Distributed arbitration by self-selection: Each device wanting the bus places a code indicating its identity on the bus

d). Distributed arbitration by collision detection: Device uses the bus when it's not busy and if a collision happens then the device tries again later (Ethernet).

a). Daisy chain arbitration: A single bus arbiter performs the required arbitration

* It is simple and cheaper method. All masters make use of the same line for bus request.

* In response to the bus request (BR) the controller sends a bus grant if the bus is free.

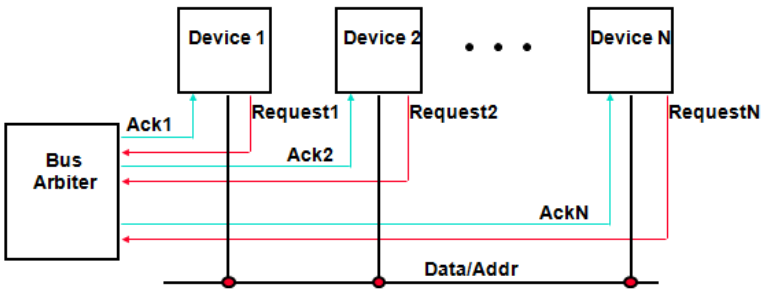


* The bus grant (BG) signal serially propagates through each master until it encounters the first one that is requesting access to the bus. This master blocks the propagation of the bus grant signal, activities the busy line and gains control of the bus.

* Therefore any other requesting module will not receive the grant signal and hence cannot get the bus access.

b). Centralized, parallel arbitration:

* In the centralized, parallel arbitration scheme, the devices independently request the bus by using multiple request lines as shown in the Figure.



* A centralized arbiter chooses from among the devices requesting bus access and notifies the selected device that it is now the bus master via one of the grant line.

* If two more devices request at the same time, highest priority device gets the bus first.

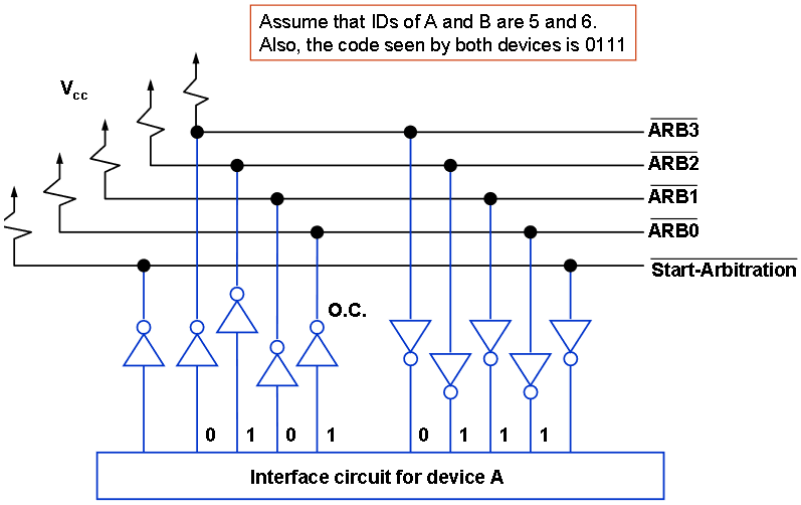
ii). Distributed:

In distributed arbitration, all devices participate in the selection of the next bus master

* Each device is assigned a 4-bit ID number.

* All the devices are connected using 5 lines, 4 arbitration lines to transmit the ID, and one line for the Start-Arbitration signal.

* To request the bus a device, asserts the Start-Arbitration signal and places its 4-bit ID number on the arbitration lines.



* The pattern that appears on the arbitration lines is the logical-OR of all the 4-bit device IDs placed on the arbitration lines.

Example:

Device A has the ID 5 and wants to request the bus:
- Transmits the pattern 0101 on the arbitration lines.

Device B has the ID 6 and wants to request the bus:
- Transmits the pattern 0110 on the arbitration lines.

Pattern that appears on the arbitration lines is the logical OR of the patterns:
- Pattern 0111 appears on the arbitration lines.

Arbitration process:

* Each device compares the pattern that appears on the arbitration lines to its own ID, starting with MSB.
* If it detects a difference, it transmits 0s on the arbitration lines for that and all lower bit positions.
* Device A compares its ID 5 with a pattern 0101 to pattern 0111.
* It detects a difference at bit position 0, as a result, it transmits a pattern 0100 on the arbitration lines.
* The pattern that appears on the arbitration lines is the logical-OR of 0100 and 0110, which is 0110.
* This pattern is the same as the device ID of B, and hence B has won the arbitration.

7.9 Vector interrupt (Interrupt vector):

Vectored interrupt is an I/O interrupt which informs the interrupt handler at the hardware level that a request for attention from an I/O device has been received and also identifies the device that sent the request.

Device requesting an interrupt identifies itself directly to the processor. The device sends a special code to the processor over the bus. The code contains:

- the identification of the device,
- starting address for the ISR (Interrupt service routine),
- address of the branch to the ISR.

The processor finds the ISR address from the code and executes using a branch address to the appropriate routine specified by the Interrupt Vector.

7.10 Direct Memory Access:

DMA or Direct memory access is a technique used to transfer the data between the fast storage devices and the memory without the involvement of the CPU.

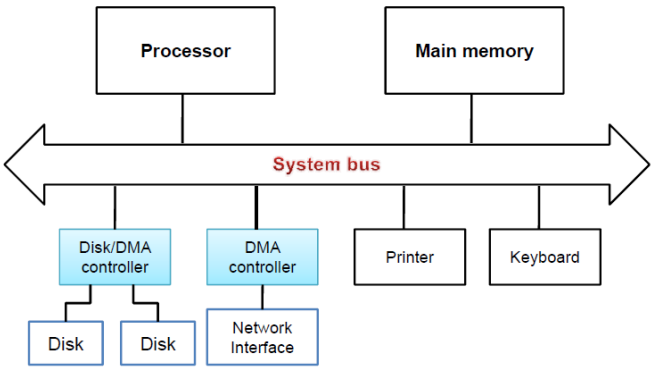
During DMA transfer the CPU is idle and has no control of the memory busses. A DMA Controller takes over the buses to manage the transfer directly between the I/O device and memory.

The DMA can transfer data in burst mode or cycle steal mode.

i). Burst mode:

In burst mode a block of words are sent continuously.

This mode of transfer is needed for fast devices such as magnetic disks, where data transmission cannot be stopped or slowed down until an entire block is transferred.



ii). Cycle steal mode:

This technique allows the DMAC to transfer one data word at a time, after which it must return control of the busses to the CPU.

The CPU merely delays its operation for one memory cycle to allow the direct memory I/O transfer to steal one memory cycle.

7.11 DMA Controller:

The DMA controller has three registers an address registers, a word count register and a control register.

a). The address register contains an address to specify the desired location in memory. It is incremented after each word that is transferred to memory.

b). The word count register holds the number of words to be transferred and decremented by one after each word transfer and internally tested to zero.

c). The control register specifies the mode of transfer.

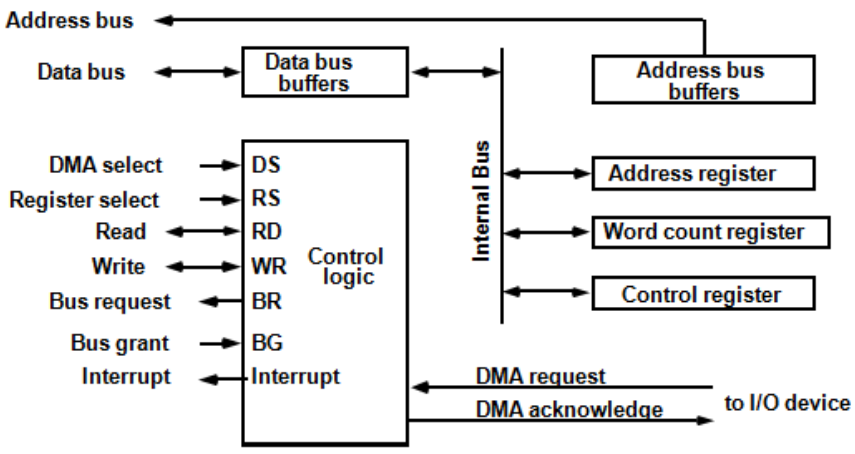
The register in the DMA are selected by the CPU through the address bus by enabling the DS (DMA select) and RS (register select) inputs.

The RD (read) and WR (write) inputs are bi directional.

When the BG bus grant is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers.

When BG = 1 The CPU has relinquished the buses and the DMA can communicate directly with the memory.

The DMA communicates with the external peripheral through the request and acknowledge lines by using hand shaking signals (DMA request and DMA Acknowledge)



The DMA is first initialised by the CPU by sending the following information through the data bus.

1. The starting address of the memory block to read or write data.
2. The word count which is the number of words in the memory block
3. Control to specify the mode of transfer such as read or write.
4. A control to start the DMA transfer.

Once the DMA is initialised the CPU stops communicating with the DMA unless it receives an interrupt signal or if it wants to check how many words have been transferred.

7.12 DMA Transfer:

Once the CPU initialises the DMA, the DMA can start the transfer between the peripheral device and the memory.

1.The peripheral device sends a DMA request.

2.DMA controller activates the bus request BR informing the CPU to relinquish the buses.

3.The CPU inform the DMAC with its bus grant (BG) informing that it relinquished the buses.

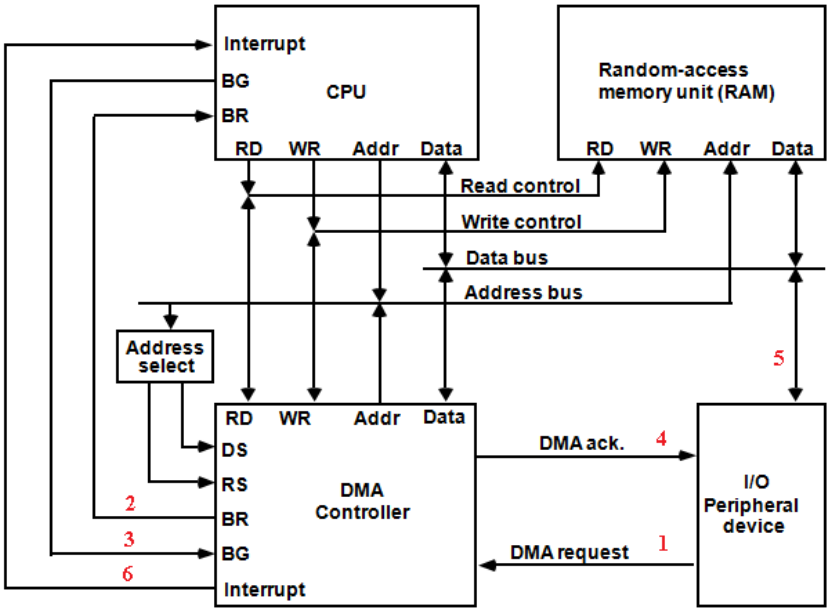
i). When BG = 0 the RD and WR are input lines allowing the CPU to communicate with the internal DMA registers.

ii). 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.

4. DMA puts the current value of its address register into the address bus, and sends the DMA acknowledgement to the peripheral device.

5.When the peripheral device receives a DMA acknowledgement it puts a word in the data bus.

6.For each word the DMA increments its address register and decrements its word count register



7. If the word count reaches zero the DMA stops any further transfer and removes its bus request

8. DMA also informs the CPU of the termination by means of an interrupt.

9. After receiving the interrupt, The CPU checks the word count register for zero value for a successful data transfer.

DMA transfer is very useful in many applications. It is used for fast transfer of information between the magnetic disks and memory, also for updating the display in an interactive terminal.

7.13 Input Output Processor:

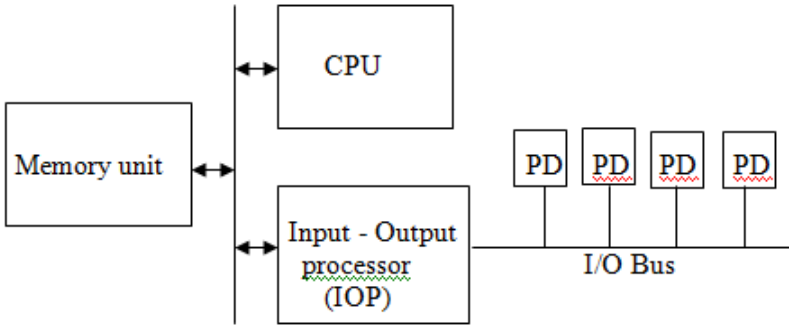
Instead of having each interface communicate with the CPU a computer may incorporate one or more external processors and assign them the task of communicating directly with all I/O devices.

An input output processor (IOP) may be classified as a processor with direct memory access capability that communicates with I/O devices.

Unlike the DMAC that must be set up entirely by the CPU, the IOP can fetch and execute its own instructions, which are specifically designed to facilitate I/O transfers.

Block diagram of a computer with I/O processor.

The fig. illustrates the block diagram of a computer with two processors.



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

2. The CPU is responsible for processing data needed in the solution of computational tasks.

3. The IOP provides a path for transfer of data between various peripheral devices (PD) and the memory unit.

4. Initially the CPU is assigned the task of initiating the I/O program. From then on the IOP operates independent of the CPU and continues to transfer data from external devices and memory.

5. The IOP must structure data words from various sources. For example the device may sent four or eight bytes which are to be converted to 32 bit before transferring to memory vice versa.

6. Once the input data is assembled they are transferred from IOP directly into memory by stealing one memory cycle from the CPU.

7. The communication between the IOP and the devices is similar to the program control method of transfer.

8. Communication to the memory is similar to direct memory access method.

9. In most computer systems, the CPU is the master while the IOP is a slave processor.

10. The CPU instructions provide operations to start an I/O transfer.

11. The IOP asks for CPU attention by means of an interrupt.

7.14 CPU_IOP Communication:

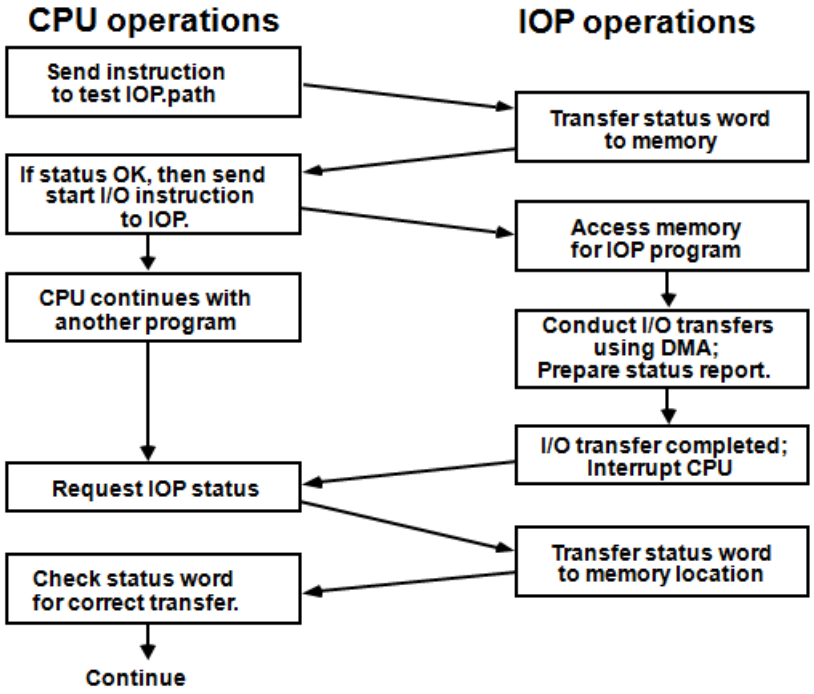
The communication between the CPU and IOP may take different forms. In most cases the memory unit acts as a message center where each processor leaves information for the other.

The sequence of operations may be carried out as shown in the below flowchart.

1. The CPU sends an instruction to test the IOP path.

2. The IOP responds by inserting a status word in memory for the CPU to check.

3. The CPU checks the status word, if status is ok then sends the instruction to start I/O transfer.



4. The memory address received with this instruction tells the IOP where to find its program.

5. The CPU continues with another program

6. The IOP continue with DMA I/O Transfer.

7. When IOP transfer is completed it sends the interrupt request to the CPU

8. The CPU issues an instruction to read the status from the IOP.

9. IOP responds by transferring the status report to the memory location. The status word indicates whether the transfer has been completed or if any errors occurred during the transfer.

10. The CPU inspects the word and determines the IO operation is completed successfully.

7.15 Questions:

1. Explain about input output Interface unit with an example.
2. Explain the differences between memory mapped, Isolated I/O method.
3. List and explain different asynchronous data transfer modes.: (strobe and handshake)
4. Draw and explain the block diagram of asynchronous serial interface.
5. Explain the asynchronous transmission with the help of timing diagram
6. Asynchronous Communication Interface
7. List and explain different I/O communication techniques. Also mention their advantages and disadvantages.
8. What are the different modes of data transfers? Explain each mode in detail. (Programmed I/O, Interrupt initiated, DMA)
9. What is priority interrupt? Briefly explain different types of priority interrupts.
10. Explain Daisy Chaining with neat sketch.
11. Explain Parallel priority interrupt in detail.
12. What do you mean by bus arbitration? Explain the parallel arbitration technique with the help of neat diagram.
13. What is an interrupt? Discuss about vector interrupt
14. What is Direct Memory Access (DMA)? What is the need for DMA? Explain the working of DMA. Also mention its advantages.
15. Give a detailed account of Direct Memory Access(DMA).
16. Explain DMA transfer in detail
17. What is an Input-Output processor? Explain the need for Input-Output processor.

Multiple choice questions:

1. Asynchronous data transfer uses ……..
a). a shared clock b) independent clock c). special clock d). None.
2. The units for serial communication is .....
a) bps b) baud c) baud rate d) all
3. In programmed I/O mode the data have to transfer through .....
a). Hard disk b) DMA c). CPU d) Serial port.
4. For data transfers interrupts are raised by .....
a) Ram b) Cache c) Hard disk d) All
5. Which mode is used for bulk data transfers
a) Programmed I/O b) Interrupt I/O c) DMA d) All.

Fill in the blanks:

1. The purpose of I/O interface is to .....
2. The isolated I/O method isolates ..... and ..... addresses.
3. Difference between strobe and handshake method is .....
4. Asynchronous serial transfer always start with ..... and ends with .....
5. UART is a .....
6. In asynchronous communication interface shift registers are used for .....
7. In daisy chain method the highest priority device is always placed at ..... in the chain

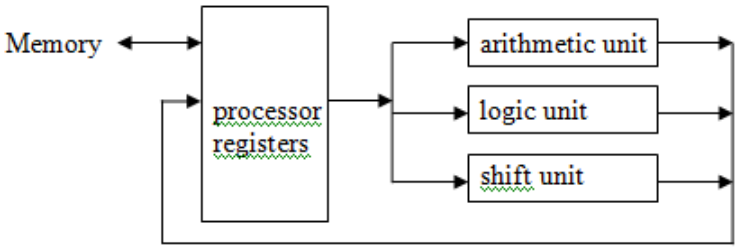
8. Bus arbitration is the process of determining which competing bus master .....
9. The DMA can transfer data in ..... mode or ..... mode.
10. The DMA controller has ..... , ... and ..... registers.

8. PIPELINE AND VECTOR PROCESSING

8.1 Parallel Processing:

Parallel processing is a technique used to provide simultaneous data processing. The purpose is to speed up the processing capability and increase its throughput. Parallel processing is established by distributing the data among the multiple functional units.

For example the arithmetic, logic and shift operations can be separated into three units and the operands diverted to each unit under the supervision of a control unit.



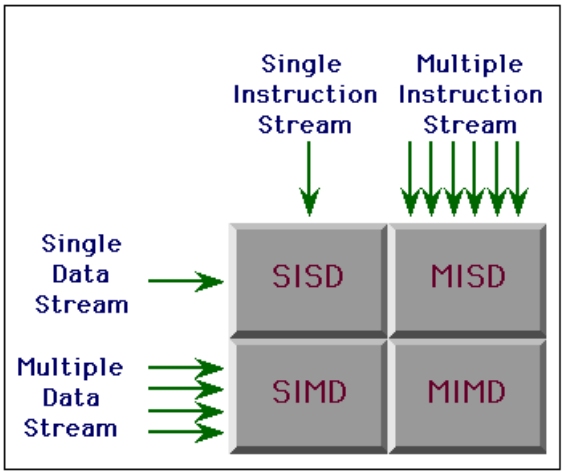
8.2 M.J.Flynn classification: It divides computers into four major groups as follows:

1. Single instruction stream, single data stream (SISD)

2. Single instruction stream, multiple data stream (SIMD)

3. Multiple instruction stream, single data stream (MISD)

4. Multiple instruction stream, multiple data stream (MIMD)

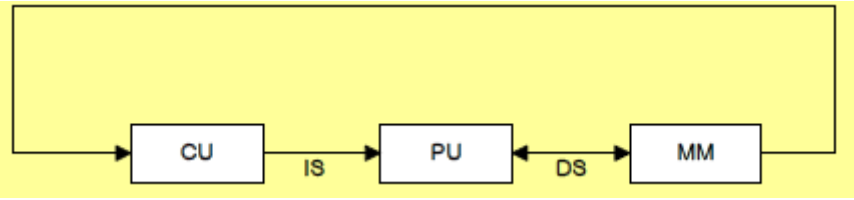


8.2.1 Single instruction stream, single data stream (SISD):

* Organisation consists of 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 may be achieved by means of multiple functional units or by pipeline processing.

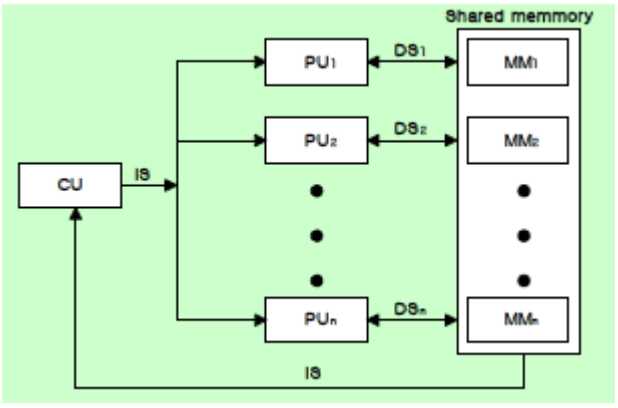


8.2.2 Single instruction stream, multiple data stream (SIMD):

* Organisation includes many processing units, under supervision of a common control unit.

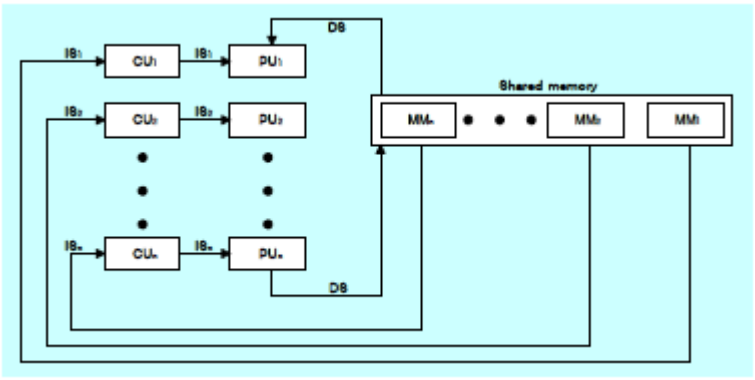
* All processors receive the same instruction from the control unit but operate on different items of data.

* The shared memory unit must contain multiple modules so that it can communicate with all processors simultaneously.



8.2.3 Multiple instruction stream, single data stream (MISD):

This structure is only of theoretical interest since no practical system has been constructed using this organisation.

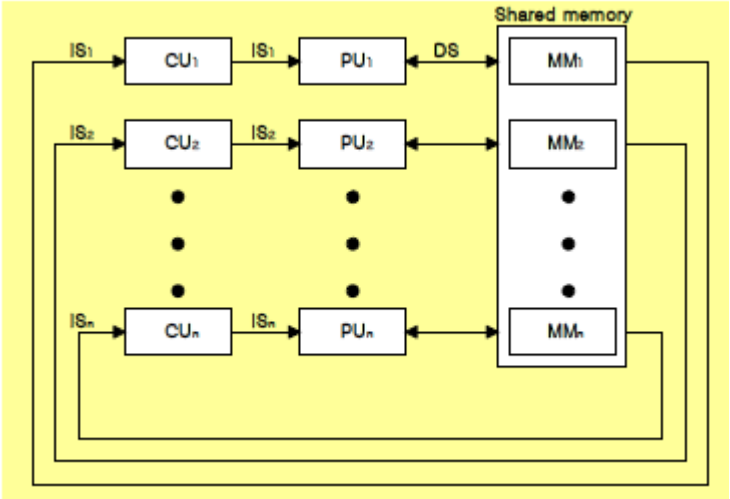


8.2.4 Multiple instruction stream, multiple data stream (MIMD):

* Organisation refers to a computer system capable of processing several programs at the same time.

* Most multiprocessors and multi computer systems can be classified in this category.

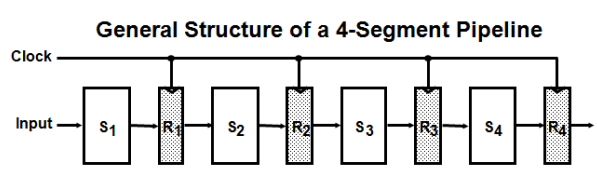
Example: Super computers.

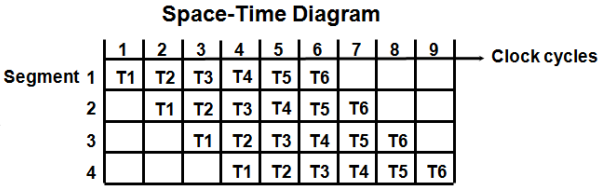


One type of classification that does not fit Flynn's classification is pipelining.

8.3 Pipelining:

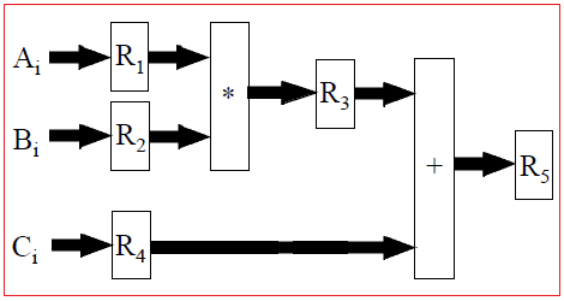
Pipelining is a technique of decomposing a sequential process into sub operations, with each sub process is executed in a special dedicated segment that operates concurrently with all other segments.





Ex: $A_i * B_i + C_i$ where for i=1,2,3,…7

Each sub operation is to be implemented in a segment within pipeline. The below table illustrates the Pipeline technique for the above example:



Clock	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1			
2	A_2	B_2	$A_1 * B_1$	C_1	
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8			$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9					$A_7 * B_7 + C_7$

8.3.1 Pipeline Speedup:

i). Conventional Machine (non pipeline unit):

the total time required for n tasks is = **$n * t_n$**. (where t_n = time equal to complete each task.)

ii). Pipelined Machine: the total time required for n tasks is = **$(k + n-1) * t_p$**

K = No. of segments, n = no. of tasks, t_p = clock cycle time.

iii). Speedup of a pipeline processing over an equivalent non pipeline processing is defined by the ratio $S_k = \frac{n*t_n}{(k + n - 1)*t_p}$

As the number of tasks increases, **n becomes much larger than k-1**, and **k + n- 1 approaches the value of n**. Under this condition, the speedup becomes $S_k = \frac{t_n}{t_p}$

If we assume that the time it takes to process a task is the same in the pipeline and nonpipeline circuits, we will have $t_n = kt_p$. Including this assumption, the speedup reduces to

This shows that the theoretical maximum speedup that a pipeline can provide is k, where k is the number of segments in the pipeline. $S_k = k$, if $t_n = k * t_p$

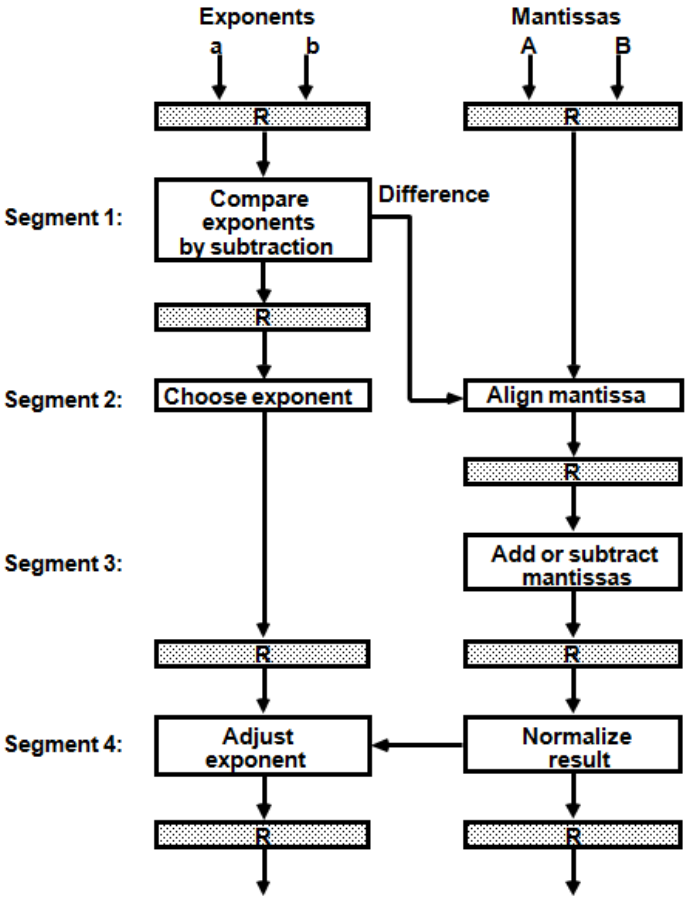
8.3.2 Arithmetic Pipeline:

Pipeline arithmetic units are used to implement floating point operations, multiplication of fixed point numbers etc. The below example illustrates the floating point addition and subtraction. The inputs to the floating point adder pipeline are two normalized floating point binary numbers.

$X = A * 2^a$
$Y = B * 2^b$

A and B are two fractions that represents the mantissas and a and b are the exponents.

The floating point addition and subtraction can be performed in four segments as shown in the below flow chart.



1. Compare the exponents
2. Align the mantissas
3. Add or subtract the mantissas.
4. Normalise the result.

Example: The following numerical example clarifies the sub operations performed in each segment. For this let us consider two normalized floating point numbers.

$X = 0.9504 * 10^3$
$Y = 0.8200 * 10^2$

1. **Compare the exponents**: The two exponents are subtracted 3–2=1. The larger exponent 3 is chosen as the exponent of the result.

2. **Align the mantissas**: To align the two mantissas under the same exponent shift the mantissa of Y to the right to obtain

$X = 0.9504 * 10^3$
$Y = 0.08200 * 10^3$

3. **Addition of the two mantissas**: $Z=1.0324 * 10^3$

4. **Normalise the result**: The sum is adjusted by normalizing the result so that it has a fraction with a non zero first digit. this is done by shifting the mantissa once to the right and incrementing the exponent by one to obtain the normalized sum.

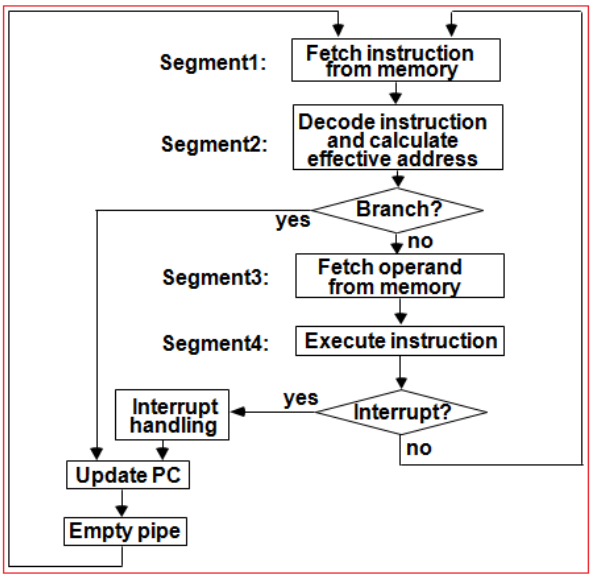
$Z=0.10324 * 10^4$

8.3.3 Instruction Pipeline:

An instruction pipeline reads instructions from memory while previous instructions are being executed in other program. This causes the instruction fetch and execute phases to overlap and perform simultaneous operations.

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.



While an instruction is being executed in seg.4, the next instruction in sequence is busy fetching an operand from memory in seg.3.

The effective address may be calculated in a separate arithmetic circuit for the third instruction, the fourth and all subsequent instructions can be fetched and placed in an instruction FIFO.

Thus up to four sub operations in the instruction cycle can overlap and up to four different instructions can be in progress of being processed at the same time.

Once in a while an instruction in the sequence may be a program control type that causes a branch out of normal sequence.

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

n that case the pending operations in the last two segments are completed and all information stored in the instruction buffer is deleted.

Similarly an interrupt request when acknowledged, will cause the pipeline to empty and start again from a new address value.

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.

8.4 Pipeline conflicts:

There are three major difficulties that cause the instruction pipeline to deviate from its normal operation.

i). **Resource conflicts**
ii). **Data dependency conflicts**
iii). **Branch difficulties** (Control Hazards)

i). Resource conflicts (Structural hazards): 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.

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

Resolving data dependency:

i). Hardware interlocks
ii). Operand forwarding
iii). Delayed load

a). Hardware interlocks: Hardware interlocking uses hardware to detect when a data dependency will occur and to insert delays into the pipeline so that the data conflict is avoided.

b). Operand forwarding: Operand forwarding is another hardware method. It also senses when a data dependency will occur, but it routes the operands from within the pipeline to the part of the pipeline which needs them.

For example, it forwards the operand A from the execute stage back to the operand fetch stage, where the second instruction needs it.

c). Delayed load: a compiler can resolve data dependency by inserting no-ops to effect a delayed load. This is similar to hardware interlocking but the task is carried out by compiler software.

Ex: 1: A=B+C FI = fetches an instruction, DA = decode instruction,

2: NOP FO = fetches the operand, EX = executes the instruction

3: D=A+E NOP = No operation

iii). Branch difficulties: Arise from branch and other instructions that change the value of PC. They occur when jump instructions are processed, because other instructions will be in the pipeline.

Ex: 1: A=B+C
2: **Jump X**
3: D=A+E

In this example, we see that equation 3 has entered the pipeline and is being processed even though it should not be executed. Also, it will be necessary to flush the pipeline with no-ops before taking the jump.

a).Prefetch target instruction: is to create two instruction streams into the pipeline and to fill one with the instructions that will be executed if the branch is taken and the other with the instructions that will be executed if it is not taken. When the branch is executed, the correct stream is accessed for future instructions.

b).branch target buffer: an associative memory into which potential instructions are fetched. If the branch is taken, this provides a quicker location for the instructions to be executed.

c). loop buffer: The loop buffer is a variation of the branch target buffer. This is a cache memory which stores loops in their entirety, thus avoiding repeated memory accesses to fetch the instructions.

d). branch prediction: Branch prediction uses probability to "guess" whether or not a branch will be taken. Depending on the odds, it fetches either the following instruction or the instruction branched to as the next instruction in the pipeline.

When the guess is correct, performance improves. This is particularly useful for loops, which always branch back except for the last iteration.

Clock Cycles	1	2	3	4	5	6	7	8
LOAD	FI	DA	FO	EX				
INCR		FI	DA	FO	EX			
ADD			FI	DA	FO	EX		
JUMP X				FI	DA	FO	EX	
NOP					FI	DA	FO	EX
NOP						FI	DA	FO
NOP							FI	DA
X:SUB								FI

e). delayed branch: In the delayed branch, the compiler rearranges instructions to enter the pipeline in order to perform useful work while the branch is being processed, without modifying the function of the program.

Fig1: Once the JUMP instruction is taken, no-ops are inserted to clear the pipeline. We seek to improve performance by getting rid of the no-ops.

We can also replace the no-ops with instructions, which doesn't change the overall result.

Clock Cycles	1	2	3	4	5	6	7	8
1. JUMP x	FI	DA	FO	EX				
2. LOAD		FI	DA	FO	EX			
3. INCR			FI	DA	FO	EX		
4. ADD				FI	DA	FO	EX	
5. X:SUB					FI	DA	FO	EX

8.5 RISC Pipeline:

RISC uses a small number of sub operations with each being executed in one clock cycle. It uses fixed length instruction format, the decoding of the operation can occur at the same time.

All data manipulation instructions have register to register operations. Since all operands are in registers there is no need of finding effective address.

Therefore the instruction pipeline can be implemented by using two or three segments.

i). One segment fetches the instruction from program memory
ii). Second segment executes the instruction in the ALU
iii). Third segment may be used to store the result of the ALU operation in a destination register.

The data transfer instructions in RISC are limited to load and store instructions.

To avoid data and instruction conflicts most RISC machines use two separate buses with two memories, one for storing the data and the other for storing the instructions.

8.5.1 Compiler support:

Instead of designing hardware to handle the difficulties associated with data conflicts and branch penalties, RISC processors rely on the efficiency of the compiler to detect and minimize the delays encountered with these problems.

The following examples show how a compile can optimize the machine language program to compensate for pipeline conflicts.

i) Three segment instruction pipeline
ii). Delayed Load
iii). Delayed Branch

i). Three segment instruction pipeline:

RISC processor consists of three types of instructions:

a). The Data manipulation instructions operate on data in processor registers.

b). The Data transfer instructions are load and store instructions that use an effective address.

c). The Program control instructions use register values and a constant to evaluate the branch address.

The instruction cycle can be divided into three sub operations and implemented in three segments:

I : Instruction Fetch
A : ALU operation
E : Executive instruction

ii). Delayed Load:

Consider now the operation of the following four instructions:

1. LOAD : R1 ← M [address 1]
2. LOAD : R2 ← M [address 2]
3. ADD : R3←R1 +R2
4. STORE : M [address 3] ←R3

FIG 1. Pipeline timing with data conflict

Clock Cycles	1	2	3	4	5	6
1. Load R1	I	A	E			
2. Load R2		I	A	E		
3. Add R1+R2			I	A	E	
4. Store R3				I	A	E

In this three segment pipeline, there will be a data conflict in instruction 3 because the operand in R2 is not yet available in the A segment.

The E segment in clock cycle 4 is in a process of placing the memory data into R2.

The A segment in clock cycle 4 is using the data from R2 but the value in R2 will not be correct value since it has not yet been transferred from memory.

FIG 2. Pipeline timing with delayed load

Clock Cycles	1	2	3	4	5	6	7
1. Load R1	I	A	E				
2. Load R2		I	A	E			
3. No operation			I	A	E		
4. Add R1+R2				I	A	E	
5. Store R3					I	A	E

In Fig 2 no-op instruction is used to advance one clock cycle in order to compensate for the data conflict in the pipeline. This concept of delaying the use of the data loaded from memory is referred to as delayed load.

The advantage of the delayed load approach is that the data dependency is taken care by the compiler rather than the hardware.

iii). Delayed Branch (realignment):

A branch instruction delays the pipe line operation until the instruction at the branch address is fetched.

Most RISC processors rely on the compiler to redefine the branches so that they take effect at the proper time in the pipeline. This method is referred to as *delayed branch.*

Fig1: **Using no-operation instructions**

Clock cycles:	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
2. Increment		I	A	E						
3. Add			I	A	E					
4. Subtract				I	A	E				
5. Branch to X					I	A	E			
6. NOP						I	A	E		
7. NOP							I	A	E	
8. Instr. in X								I	A	E

The compiler analyze the instructions before and after the branch and rearrange the program sequence by inserting useful instructions in the delay steps.

Ex:

* Load from memory to R1
* Increment R2
* Add R3 to R4
* Subtract R5 from R6
* Branch to address X

Fig 2: **Rearranging the instructions**

Clock cycles:	1	2	3	4	5	6	7	8
1. Load	I	A	E					
2. Increment		I	A	E				
3. Branch to X			I	A	E			
4. Add				I	A	E		
5. Subtract					I	A	E	
6. Instr. in X						I	A	E

8.6 Vector Processing:

Vector: A vector is an ordered set of a one-dimensional array of data items.

A vector *V* of length *n* is represented as a row vector by *V = [V1 V2 V3 ••• Vn]*.

It may be represented as a column vector if the data items are listed in a column. The element *Vi* of vector *V* is written as V(*I*) and the index *I* refers to a memory address or register where the number is stored.

Many scientific problems require arithmetic operations on large arrays of numbers. These numbers are usually formulated as vectors and matrices of floating-point numbers.

Vector Processing:

The vector processing model is one in which the processor (CPU, GPU etc.) takes one instruction and applies it to multiple data sets.

A computer capable of vector addressing eliminates the overhead associated with the time it takes to fetch and execute the instructions in the program loop. It allows operations to be specified with a single vector instruction of the form.

C(1:100) = A(1:100) + B(1:100)

The vector instruction includes the initial address of the operands, the length of the vectors and the operation to be performed, all in one composite instruction. The addition is done with a pipelined floating point adder.

Operation code	Base address source 1	Base address source 2	Base address destination	Vector length

This is essentially a three address instruction with three fields specifying the base address of the operands and an additional field that gives the length of the data items in the vectors.

8.7 Array Processors:

An Array processor is a processor that performs computations on large arrays of data. The term is used to refer to two different types of processors.

1. An attached array processor
2. An SIMD array processor

i). An attached array processor:

An attached array processor is designed as a peripheral for a conventional host computer and its purpose is to enhance the performance of the computer by providing vector processing for complex scientific applications.

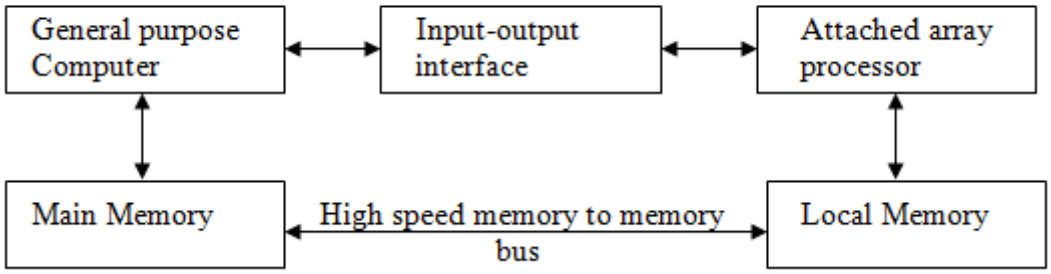
It achieves the high performance by means of parallel processing with multiple functional units. It includes an arithmetic unit containing one or more pipelined floating point adders and multipliers.

The array processor can be programmed by the user to accommodate a variety of complex arithmetic problems.

The host computer is a general-purpose commercial computer and the attached processor is a back-end machine driven by the host computer.

The below figure shows the interconnection of an attached array processor to a host computer.

RF: Attached array processor with host computer



The array processor is connected through an input-output controller to the computer and the computer treats it like an external interface.

The data for the attached processor are transferred from main memory to a local memory through a high-speed bus.

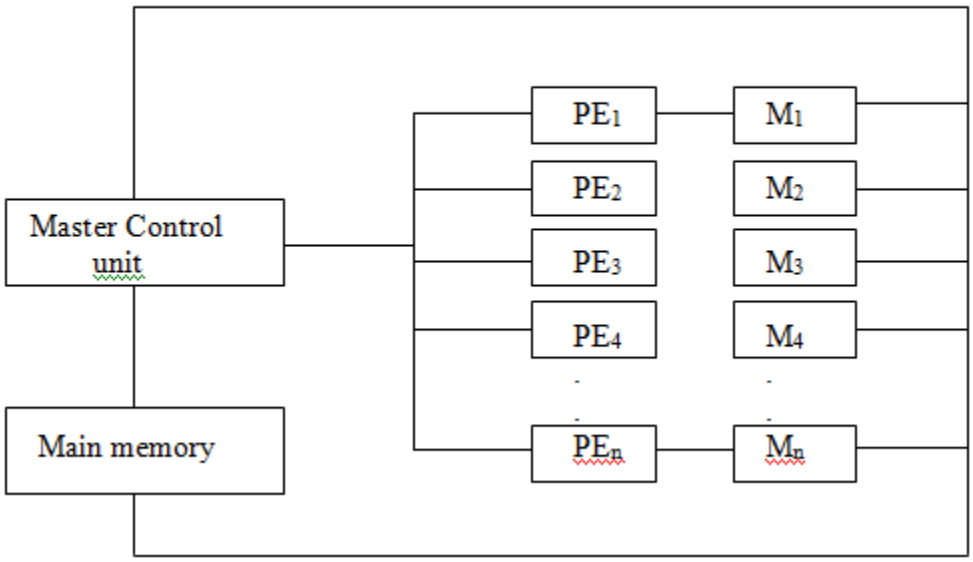
The general purpose computer without the attached processor serves the users that need conventional data processing. The system with the attached processor satisfies the needs for complex arithmetic applications.

ii). An SIMD array processor:

An SIMD array processor is a computer with multiple processing units operating in parallel. The processing units are synchronized to perform the same operation under the control of a common control unit, thus providing a single instruction stream, multiple data stream (SIMD) organization.

The below fig shows the general block diagram of an array processor.

RF 02: SMD array processor organisation.



It contains a set of identical processor elements (PEs), each having a local memory M. Each processor element includes as ALU, a floating point arithmetic unit, and working registers. The master control unit controls the operations in the processor elements.

The main memory is used for storing the program. Scalar and program control instructions are directly executed with in the master control unit. Vector instructions are broadcast to all PEs simultaneously.

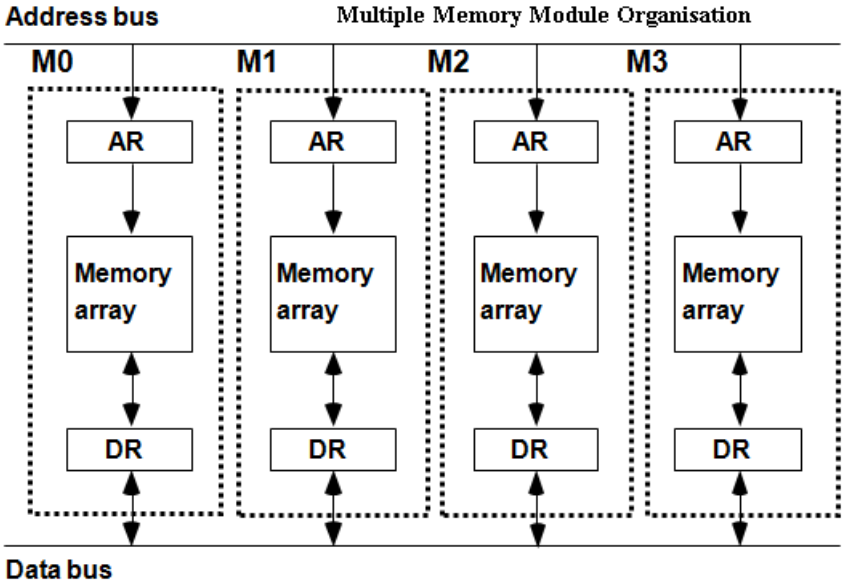
Each PE uses operands stored in its local memory. Vector operands are distributed to the local memories prior to the parallel execution of the instruction.

8.8 Modular Memory Organisation:

Pipeline and vector processors often require simultaneous access to memory from two or more sources.

An instruction pipeline may require the fetching of an instruction and an operand at the same time from two different segments. Similarly, an arithmetic pipeline usually requires two or more operands to enter the pipeline at the same time.

Instead of using two memory buses for simultaneous access, the memory can be partitioned into a number of modules connected to a common memory address and data buses. Below figure shows a memory unit with four modules.



Each memory array has its own address register *AR* and data register *DR*.

The address registers receive information from a common address bus and the **two least significant bits of the address can be used to distinguish between the four modules**.

The modular system permits one module to initiate a memory access while other modules are in the process of reading or writing a word

8.8.1 Memory Interleaving:

The advantage of a modular memory is that it allows the use of a technique called *interleaving.* In an interleaved memory, different sets of addresses are assigned to different memory modules.

For example, in a two-module memory system, the even addresses may be in one module and the odd addresses in the other.

When the number of modules is a power of 2, **the least significant bits of the address select a memory module and the remaining bits designate the specific location to be accessed** within the selected module.

A modular memory is useful in systems with pipeline and vector processing. A vector processor that uses an n-way interleaved memory can fetch *n* operands from *n* different modules.

8.9 Supercomputers:

A commercial computer with vector instructions and pipelined floating-point arithmetic operations is referred to as a *supercomputer*. Supercomputers are very powerful, high-performance machines used mostly for scientific computations.

Minimises Distance: To speed up the operation, the components are packed tightly together to minimize the distance that the electronic signals have to travel.

Heat removal: Supercomputers also use special techniques for removing the heat from circuits to prevent them from burning up because of their close proximity.

Instruction set: The instruction set of supercomputers contains the standard data transfer, data manipulation, and program control instructions of conventional computers. This is augmented by instructions that process vectors and combinations of scalars and vectors.

System configuration: A supercomputer is a computer system best known for its high computational speed, fast and large memory systems, and the extensive use of parallel processing. It is equipped with multiple functional units and each unit has its own pipeline configuration.

Applications: Supercomputers are not suitable for normal everyday operations they are specifically optimized for scientific applications, such as numerical weather forecasting, seismic wave analysis, and space research. They have limited use and limited market because of their high price.

Speed: A measure used to evaluate computers in their ability to perform a given number of floating-point operations per second is referred to as *flops*. The term *megaflops* is used to denote million flops and *gigaflops* to denote billion flops. A typical supercomputer have the ability to perform 50 to 250 megaflops.

Cray-1 supercomputer:

* The first supercomputer developed in 1976 is the Cray-1 supercomputer. It uses vector processing with 12 distinct functional units in parallel.

* All the functional units can operate concurrently with operands stored in the large number of registers (over 150) in the CPU.

* A floating-point operation can be performed on two sets of 64-bit operands during one clock cycle of 12.5 ns. This gives a rate of 80 megaflops.

* It has a memory capacity of 4 million 64-bit words. The memory is divided into 16 banks, with each bank having a 50-ns access time. This means that when all 16 banks are accessed simultaneously, the memory transfer rate is 320 million words per second.

* The new Cray-2 supercomputer is 12 times more powerful than the Cray-1 in vector processing mode.

8.10 Questions:

1. Explain the Flynn's classification to accomplish parallel processing.
2. Define Pipelining? Explain the structure of Pipelining with an example.
3. Explain Four segment Pipelining.
4. Draw the space-time diagram for a six-segment pipeline showing the time it takes to process eight tasks.
5. What is meant by arithmetic pipeline? Explain.
6. Explain the pipeline for floating point addition and subtraction.
7. What is meant by instruction pipeline? Explain.
8. Explain Pipeline conflicts?
9. Explain how to handle data dependency and branching
10. Explain about instruction Hazards in pipeline
11. What are the branch difficulties in the instruction pipelining?
12. Explain the following with related to the instruction pipeline
13. Explain RISC pipeline
14. Explain about three segment instruction pipeline
15. Explain the flowing a).Vector processing b). Array processor
16. What is an array processor? Discuss about attached array processor
17. Explain Array Processors? Explain SIMD Array Processor organization in detail.
18. Explain various issues related to performance consideration of memory.
19. Explain the practical considerations in supercomputer design.

Multiple choice questions:

1. MISD is practically used in ……..
a). PCs b) Servers c). LAN d). None.
2. Super computers use which organisation .....
a) SISD b) SIMD c) MISD d) MIMD
3. For handling branch and data conflicts RISC processors rely on .....
a). CPU b) OS c). Compiler d) Shell.
4. To avoid data and instruction conflicts most RISC machines use…..
a) Shared bus b) Parallel bus c) two busses d) All.
5. SIMD array processor is a computer with multiple .....
a) Control units b) Instruction streams c) Processor units d) All

Fill in the blanks:

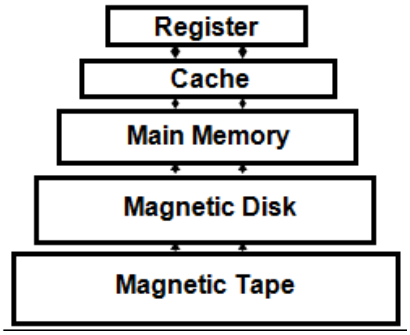
1. Parallel processing is a technique used to provide .....
2. Pipelining is a technique of decomposing a sequential process.....
3. Delayed load can resolve ..... conflicts
4. Structural hazards are caused when two segments .....
5. Data dependency conflict arises when an .....
6. RISC processors use …… segment instruction pipeline.
7. In vector processing, processor takes one instruction and .....
8. The purpose of an attached array processor is to .....
9. Branch difficulties arise when other .....
10. The array processor can be programmed by the user to accommodate …….

This page intentionally left blank.

9. MEMORY ORGANISATION

9.1 Memory Hierarchy:

i). Registers are very fast and closely associated with the processor, but they very limited in number.



ii). Cache (Static RAM): Static random access memory (SRAM) is made up of transistors. It does not need to be periodically refreshed as in DRAM. The response time of SRAM is very fast than the DRAM. SRAM is used as cache memory. It is expensive than the DRAM.

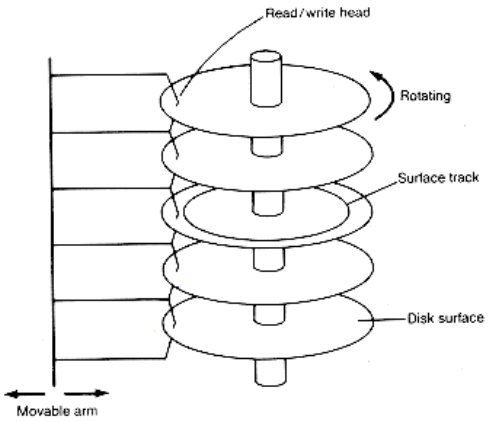
iii). Main Memory (DRAM): Dynamic random access memory is another type of random access memory that stores each bit of data in a separate capacitor within an integrated circuit.

Since real capacitors leak charge, the information eventually fades unless the capacitor charge is refreshed periodically. Because of this refresh requirement, it is a dynamic memory as opposed to SRAM and other static memory.

The advantage of DRAM is its structural simplicity: only one transistor and a capacitor are required per bit, compared to six transistors in SRAM.

This allows DRAM to reach very high densities. DRAM is inexpensive and highly dense, so it can hold more memory than SRAM within a given space. DRAM is used as main memory.

iv). Magnetic Disk: A magnetic disk is a circular plate constructed of metal or plastic coated with magnetized material. Often both sides of the disk are used and several disks may be stacked on one spindle with read/write heads available on each surface.



All disks rotate together at high speed. Bits are stored in the magnetized surface in spots along concentric circles called tracks. The tracks are commonly divided into sections called sectors.

In most systems, the minimum quantity of information which can be transferred is a sector. The subdivision of one disk surface into tracks and sectors.

v). Magnetic Tape: The Magnetic tape is a strip of plastic coated with a magnetic recording medium. Bits are recorded as magnetic spots on the tape along several tracks.

Read/write heads are mounted one in each track so that data can be recorded and read as a sequence of characters. Information is recorded in blocks referred to as records. Each record on tape has an identification bit pattern at the beginning and end.

The control recognizes the beginning of a gap. A tape unit is addressed by specifying the record number and the number of characters in the record. Records may be of fixed or variable length.

9.1.1 a). Asynchronous DRAM:

Asynchronous DRAM is a conventional DRAM. This refers to the fact that the memory is not synchronized to the system clock. A memory access is begun, and a certain period of time later the memory value appears on the bus. The signals are not coordinated with the system clock. Asynchronous memory works fine in lower-speed memory bus systems but is not nearly as suitable for use in high-speed (>66 MHz) memory systems.

9.1.2 b). Synchronous DRAM:

Synchronous DRAM or SDRAM is synchronized with the computer's clock to allow it to send instructions more efficiently by joining a pipeline of other instructions the computer is processing.

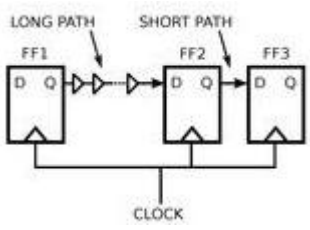
The pipelining of information in a computer allows it to receive another command before it has finished processing the previous command. This allows SDRAM to operate at much higher speeds, making it the most popular form of RAM offered on computers.

9.2 Clock skewing:

In circuit designs, **clock skew** (sometimes called **timing skew**) is a phenomenon in synchronous circuits in which the same sourced clock signal (sent from the clock circuit) arrives at different components (generally latches or flip-flops) at different times.

The operation of most digital circuit systems, such as computer systems, is synchronized by a periodic signal known as a "clock" that dictates the sequence and pacing of the devices on the circuit.

This clock is distributed from a single source to all the memory elements like registers or flip-flops.



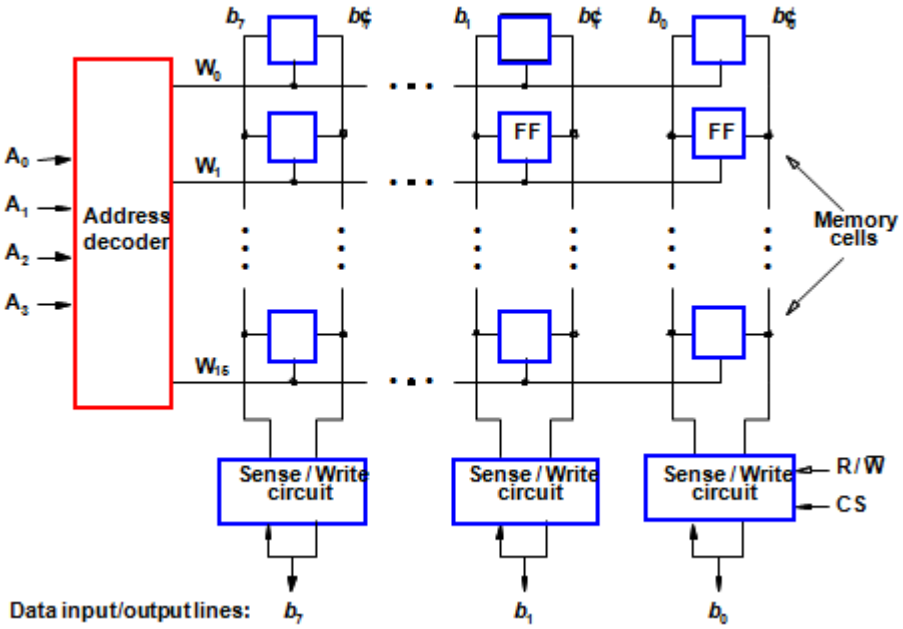
In a circuit using edge-triggered registers, when the clock edge or tick arrives at a register, the register transfers the register input to the register output, and these new output values flow through combinational logic to provide the values at register inputs for the next clock tick.

The maximum speed at which a system can run must account for the variance that occurs between the various elements of a circuit due to differences in physical composition, temperature, and path length.

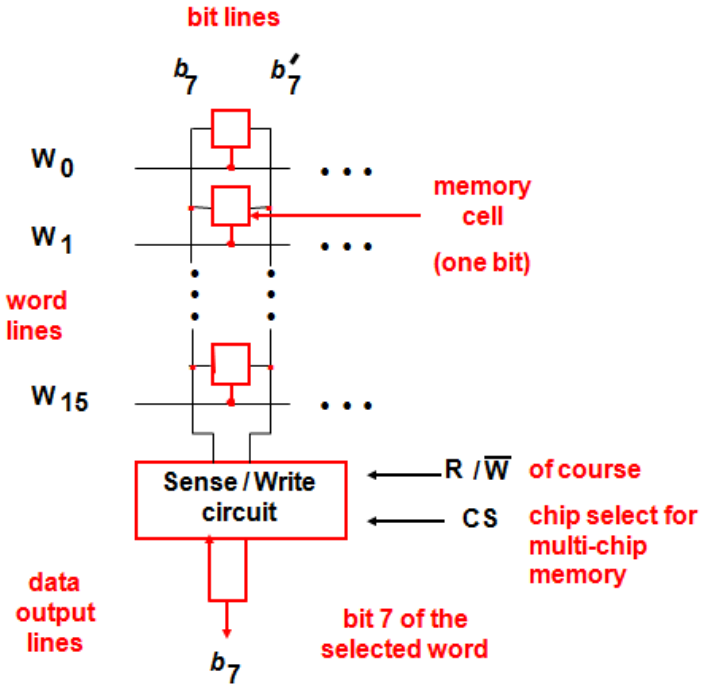
9.3 Organization of bit cells in a memory chip:

Example (128 bit chip); 16 words of 8 bits each (16 x 8).

16 external connections:
4 address lines
8 data lines
1 R / W line
1 chip select line
1 power
1 ground



9.3.1 Organization of a 1K × 1 memory chip:



9.4 Memory Address Map:

The addressing of memory either RAM or ROM can be established by means of a table that specifies the memory address assigned to each chip. The table, called a *memory address map,* is a pictorial representation of assigned address space for each chip in the system.

Ex: Assume that a computer system needs 512 bytes of RAM and 512 bytes of ROM. The RAM and ROM chips to be used are specified in the below figures.

Memory Address map for Micro computer:

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

1. The first component column specifies whether a RAM or a ROM chip is used.

2. The second column specifies a range of hexadecimal equivalent addresses for each chip.

3. The third column specifies the address bus lines. Although there are 16 lines in the address bus, only 10 lines are used others assumed to be zero.

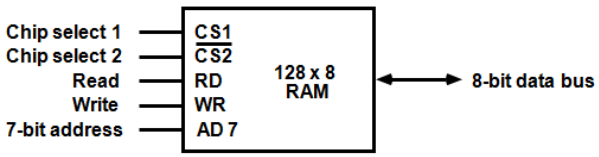
4. The small x's designate those lines that must be connected to the address inputs in each chip.

5. The RAM chips have 128 bytes and need 7 address (1 to 7) lines. The ROM chip has 512 bytes and needs 9 address (1 to 9) lines.

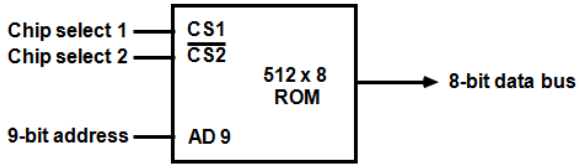
6. Bus lines 8 and 9 are used to distinguish between four RAM chips.

7. Bus line 10 is used to select RAM and ROM, If line 10 is 0, the CPU selects a RAM, and if it is equal to 1, it selects the ROM.

Typical RAM Chip:



Typical ROM Chip:



Function Table:

CS1	CS2	RD	WR	Memory function	State of data bus
0	0	x	x	Inhibit	High-impedence
0	1	x	x	Inhibit	High-impedence
1	0	0	0	Inhibit	High-impedence
1	0	0	1	Write	Input data to RAM
1	0	1	x	Read	Output data from RAM
1	1	x	x	Inhibit	High-impedence

9.5 Read Only Memories (ROM):

ROM or read only memories are non volatile memories. They can be read as like as other memories but a special writing process is needed to place the information into this memory. Since the normal operation involves only reading of stored data. This type of memories are referred as Read only memory or ROM.

9.5.1 Types of ROMS:

i). ROM: In ROM or Mask ROM, the data is physically encoded in the circuit, so it can only be programmed only during fabrication. The program cannot be erased or reprogrammed.

ii). PROM: PROM or Programmable ROM allows the data to be loaded by the user but only once in its lifetime. The reason it uses a fuse between the transistor and the ground. The user can program the chip by burning the fuses at the required locations using high current pulses. The burnt fuse leads to high state 1, unburnt fuse leads to low state 0.

iii). EPROM: EPROM or Erasable reprogrammable ROM has a similar structure to the ROM cell. However inside transistors can be programmed to behave as a permanently open switch, by injecting charge into it that become trapped inside.

By exposing to ultra violet rays the contents inside EPROM can be erased and reprogrammed. For this reason EPROM chips are mounted in packages that have transparent window.

iv). EEPROM: A significant disadvantage of EPROMs is that, they must be physically removed from the circuit for erasing or for reprogramming. The EEPROM (Electrically Erasable Programmable Read Only Memory) do not have to remove for erasure or for reprogramming. More over it is also possible to erase the cell contents selectively. The only disadvantage is that different voltages are needed for erasing, writing and reading the stored data.

v). Non-volatile random-access memory (NVRAM): is a random-access memory that retains its information when power is turned off (non-volatile). This is in contrast to DRAM and SRAM, which both maintain data only for as long as power is applied. The best-known form of NVRAM memory today is flash memory.

9.6 FLASH MEMORY:

A flash cell is based on a single transistor controlled by trapped charge just like EEPROM. The main difference between the EEPROM and the flash cell is In EEPROMS it is possible to read and write the contents of a single cell. But in Flash device it is only possible to write an entire block of cells. Reading a single cell is possible. Prior to writing the previous contents are erased.

The advantage of flash devices is they have greater density which leads to higher capacity and a lower cost per bit. They require a single power supply voltage and consume less power. Example: **Flash cards** and **Flash drives** (Solid state hard disk drives).

9.7 Cache Memories:

a). Locality of reference: Cache memories are Static RAMS which are comparatively very fast than DRAMS. Cache memory is used between the CPU and the main memory.

The effectiveness of the cache is based on a property of computer programs called *locality of reference*. This can be temporal and spatial.

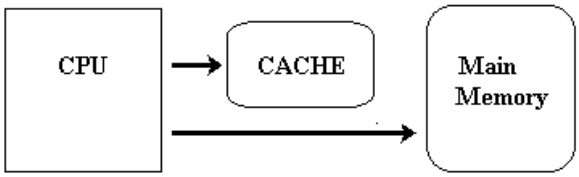
i). The temporal aspect: suggests that whenever an information item is first needed this item is brought into the cache where it will hopefully remain until it is needed again.

ii). The spatial aspect: suggests that instead of fetching just one item from the main memory to the cache, it is useful to fetch several items that reside at adjacent addresses as well.

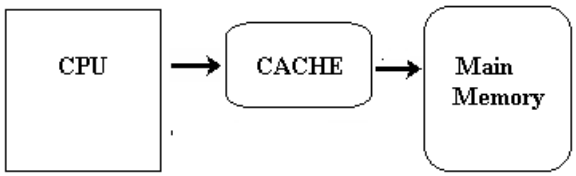
When the processor issues a read write request the cache control circuitry checks the cache if the word exists, if it exists the R/W operation is performed. This is referred as *read or write hit.*

b). Writing into Cache: For a write operation the system can follow a write through protocol or write back protocol.

i). Write through protocol: The cache and main memory are updated simultaneously. This delays the process because writing to main memory is a slow process.



ii). Write back protocol: Only the cache is updated during write operation and the main memory is updated when, that cache block is being replaced on a cache miss.



This protocol greatly reduce the memory bandwidth requirement, but control can be complex.

9.7.1 Performance of Cache memory:

The performance of cache memory is frequently measured in terms of a quantity called a hit ratio. When the CPU refers to memory and finds the word in cache, it is said to produce a hit. If it is not found it is a miss.

The ratio of the number of hits divided by the total CPU references to memory (hits and misses) is the hit ratio.

9.7.2 Cache Initialisation:

The cache is initialized when power is applied to the computer after initialization the cache is considered to be empty, but in effect it contains some invalid data. It is customary to include with each word in cache a valid bit to indicate whether or not the word contains valid data.

9.8 Auxiliary Memory (or Secondary storage):

Secondary or auxiliary storage devices are magnetic disks/tapes, Optical disks and Semiconducter (Solid state) memory devices.

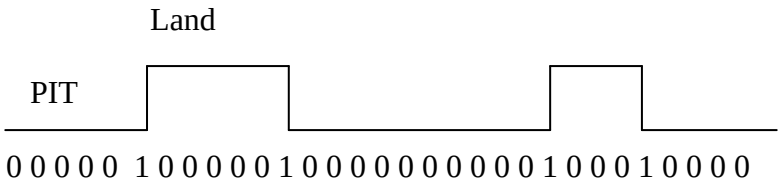
i). Magnetic disks/tapes:

Magnetic storage media can be classified as either sequential access memory (tapes) or random access memory (Floppy, Hard disk)

Magnetic storage media use iron oxide as magnetic material coated over their surface. A write head magnetizes a region by generating a strong local magnetic field, and a read head detects the magnetization of the regions. Earlier analog recording is used gradually replaced by digital recording.

ii). Optical storage devices:

In Optical storage data is recorded by making marks in a pattern that can be read back with the aid of light, usually a beam of laser light precisely focused on a spinning disc.



Stored binary pattern (reflection = 0), Non reflection = 1 (black spot)

Optical storage can range from a single drive reading a single CD-ROM (or DVD) to multiple drives reading multiple discs such as an optical jukebox.

iii). Semiconducter (Solid state) memory devices:

Semiconductor memory is an electronic data storage device. In a semiconductor memory chip, each bit of binary data is stored in a tiny circuit called a memory cell consisting of one to several transistors.

The memory cells are laid out in rectangular arrays on the surface of the chip. The 1-bit memory cells are grouped in small units called words which are accessed together as a single memory address.

Ex: RAM, ROM, pen drives etc.

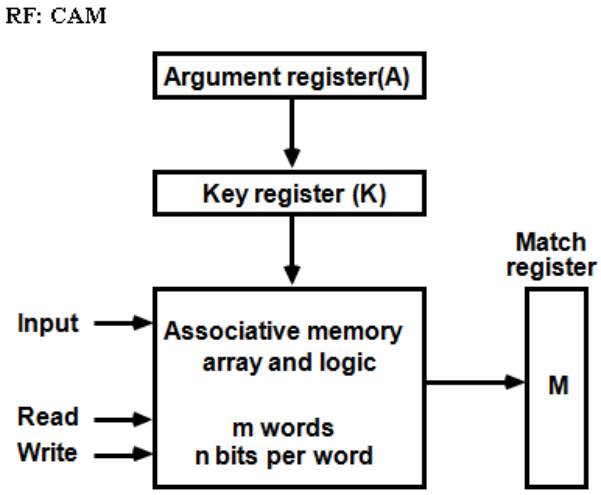
9.9 Associative Memory (CAM or Content Addressable Memory):

Search is one of the most commonly used function in both random or sequential access memory. The time required to find an item stored in memory can be reduced considerably if stored data can be identified for access by the content of the data itself rather than by an address.

A memory unit accessed by content is called an associative memory or content addressable memory (CAM). This type of memory is accessed simultaneously and in parallel on the basis of data content rather than by specific address or location.

When a word is to be read from an associative memory, the content of the word, or part of the word, is specified. The memory locates all words which match the specified content and marks them for reading**.**

9.9.1 Hardware Organisation:



It consists of a memory array and logic for m words with n bits per word. The argument register A and key register K each have n bits, one for each bit of a word.

The match register M has m bits, one for each memory word. Each word in memory is compared in parallel with the content of the argument register.

The words that match the bits of the argument register set a corresponding bit in the match register.

After the matching process, those bits in the match register that have been set indicate the fact that their corresponding words have been matched.

Example:

To illustrate with a numerical example, suppose that the argument register A and the key register K have the bit configuration shown below. Only the three leftmost bits of A are compared with memory words because K has l's in these positions.

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

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

9.10 Mapping:

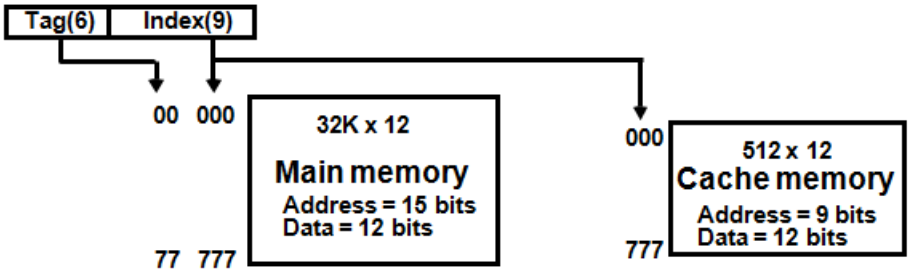
The transformation of data from main memory to cache memory is referred to as a mapping process. The following are the different types of mapping used in cache memories.

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

i). Direct mapping:

In direct mapping the CPU address of 15 bits is divided into 6 bit tag field and 9 bit index field.

The number of bits in the index field is equal to the number of address bits required to access the cache memory.

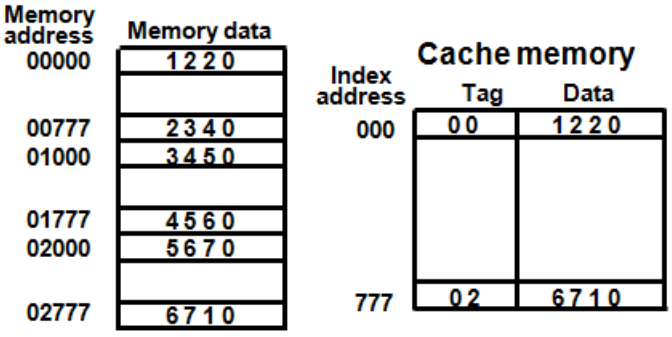


Direct cache memory organisation:

When the CPU generates a memory request, the index field is used for the address to access the cache. The tag field of the CPU address is compared with the tag in the word read from the cache.

If the two tags match there is a hit and the desired data word is in cache. If there is no match there is a miss and the required word is read from main memory.

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



ii). Associative mapping:

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

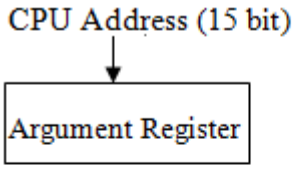
The address value of 15 bits is shown as a 5 digit octal number and its corresponding 12 bit word is shown as a 4 digit octal number.

A CPU address of 15 bits is placed in the argument register and the associative memory is searched for a matching address.

If the address is found, the corresponding 12 bit data is read and sent to the CPU.

If no match is found the main memory is accessed and the address data pair is transferred to the associative cache memory.

If there is no space basing on the replacement algorithm (FIFO) it replaces one of the pair.



Address	Data
01000	3450
02777	6710
22345	1234

iii). Set associative mapping:

This is an improvement over the direct mapping method; here each word of cache can store two or more words of memory under the same index address.

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

When the CPU generates a memory request the index value of the address is used to access the cache.

The tag field of the CPU address is then compared with both tags in the cache to determine if a match occurs.

The hit ratio will improve as the set increases because more words with the same index but different tags can reside in cache.

RF: Two way set associate mapping cache

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

Disadvantage: However an increase in the set size increases the number of bits in words of cache and requires more complex comparison logic.

9.11 Virtual Memory (VM):

i). Virtual memory: A virtual memory system is a combination of hardware and software techniques. The hardware mapping mechanism and the memory management software together constitute the architecture of the virtual memory.

ii). Where we need VM: While running large programs in the PC the memory may not be sufficient, in this case the operating system has to constantly swap information back and forth between RAM and the hard disk. This is called **thrashing**, and it can make your computer incredibly slow.

Virtual memory combines your computer's RAM with temporary space on your hard disk. When RAM runs low, virtual memory moves data from RAM to a space called a **paging file**.

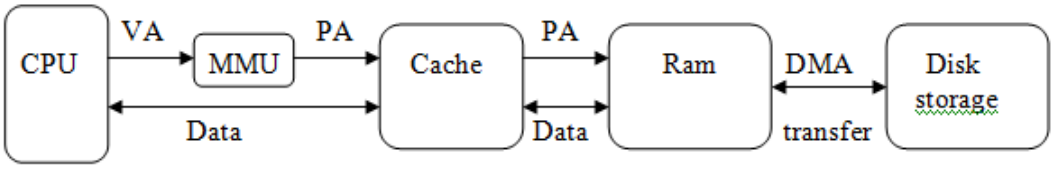
iii). Page file: The area of the hard disk that stores the RAM image is called a **page file**. It holds pages of RAM on the hard disk, and the operating system moves data back and forth between the page file and RAM.

A virtual memory system provides a mechanism for translating program generated addresses into correct main memory locations.

iv). Address space: An address used by the programmer will be called a ***virtual address*** (VA) and the set of such addresses the ***address space***.

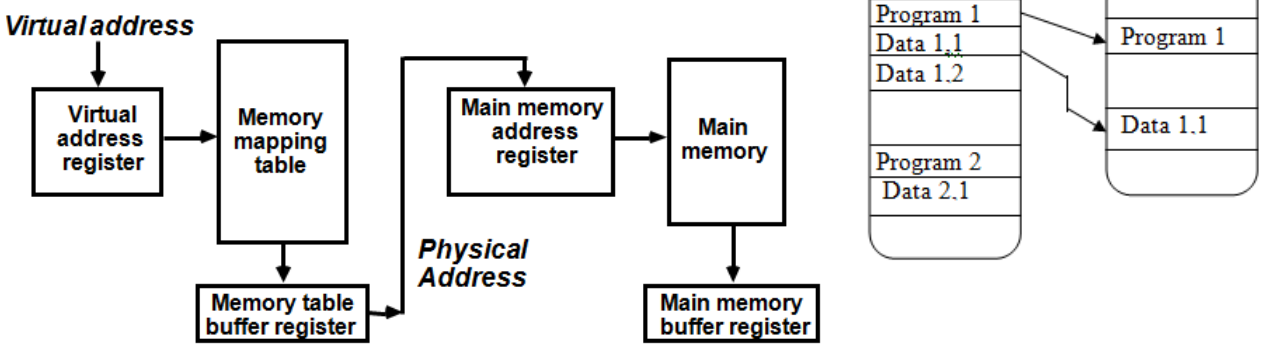
v). Memory space: An address in the main memory is called a location or ***physical address (PA)***. The set of such locations is called the ***memory space***.

vi). Memory management unit: It is a special hardware unit which translates virtual addresses into physical addresses.

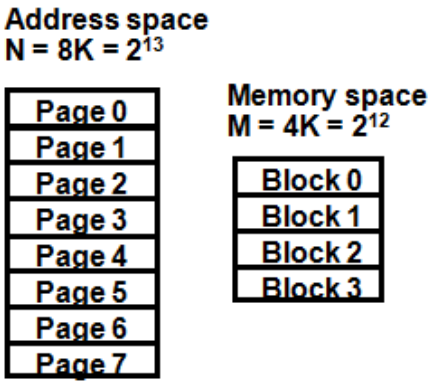


Address mapping and the relation between address and memory space in virtual memory system.

The main memory consists of physical address and the auxiliary memory (hard disk) holds the address space (virtual address)**.**



9.12 Address Mapping using Pages:



i) Page: A page, memory page, or virtual page is a fixed-length contiguous block of virtual memory, described by a single entry in the page table.

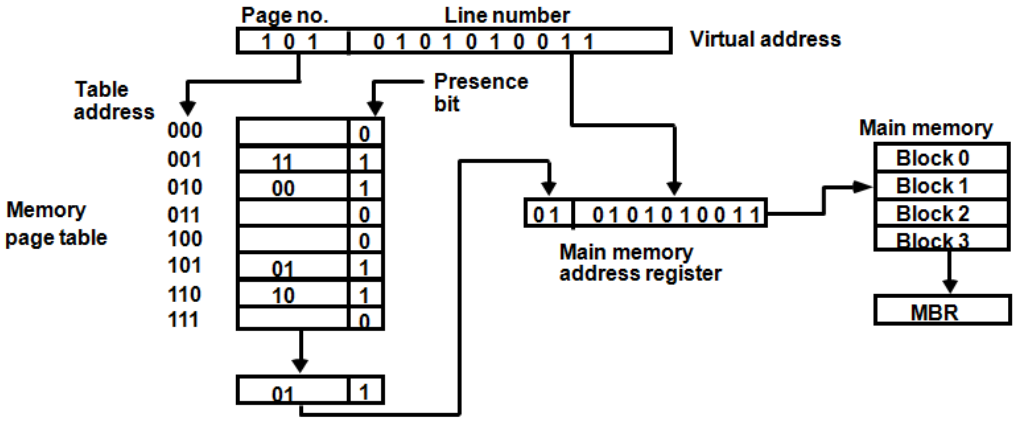
They constitute the basic unit of information (not in terms of bytes) that is moved between the main memory and the disk.

The physical memory is broken into groups of equal size called blocks. The term **page frame** is sometimes denoted a block.

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 number with in page.

The below figure illustrates the pages system.

ii) Address Translation: Memory table in a paged system:



The memory paged system consists of eight words, one for each page.

The address in the page denotes the page number and the content of the word gives the block number where that page is stored in main memory. The table shows that pages 1 (001), 2 (010), 5 (101) and 6 (110) are now available in main memory in blocks 3 (11), 0 (00), 1 (01) and 2 (10) respectively. A presence bit 0 indicates that the page is not available.

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.

If the presence bit is 1, the block number thus read is transferred to the two 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 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 is 0, a call to the operating system is then generated to fetch the required page from auxiliary memory and place it into the main memory before resuming computation.

9.13 Associative Memory Page Table:

A random-access memory page table is inefficient with respect to storage utilization. Out of eight words of memory needed one for each page, only four words are used and the remaining always be marked empty because main memory cannot accommodate more than four blocks.

A more efficient way to organize the page table would be to construct it with a number of words equal to the number of blocks in main memory. In this way the size of the memory is reduced and each location is fully utilized.

This method can be implemented by means of an associative memory with each word in memory containing a page number together with its corresponding block number.

The page field in each word is compared with the page number in the virtual address. If a match occurs, the word is read from memory and its corresponding block number is extracted.

Consider again the case of eight pages and four blocks.

We replace the random access memory-page table with an associative memory of four words as shown in Fig.

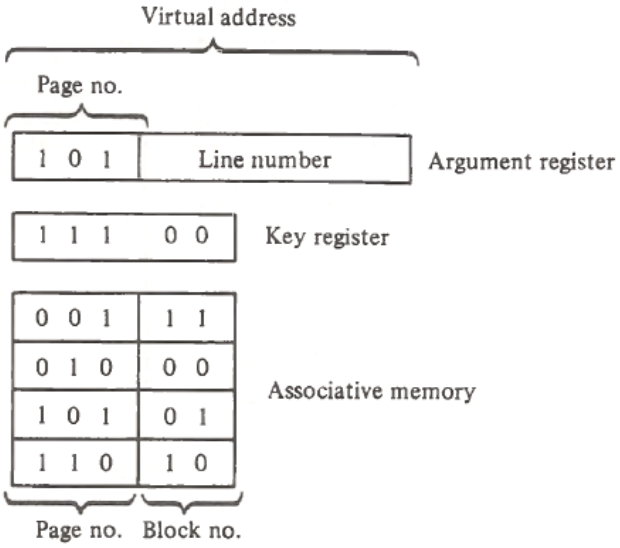
Each entry in the associative memory array consists of two fields.

The first three bits specify a field for storing the page number.

The last two bits constitute a field for storing the block number.

The virtual address is placed in the argument register. The page number bits in the argument register are compared with all page numbers in the page field of the associative memory.

An associative memory page table.



If the page number is found, the 5-bit word is read out from memory. The corresponding block number, being in the same word, is transferred to the main memory address register.

If no match occurs, a call to the operating system is generated to bring the required page from auxiliary memory.

9.14 Page Replacement:

9.14.1 Page fault:

Page fault occurs when a required page by the program, is not available in the main memory. A new page is then transferred from auxiliary memory to main memory.

* If main memory is full it is necessary to remove a page from a memory block to make room for the new page. Which page has to be removed is decided by the type of the replacement algorithm that is used.

The most commonly used replacement algorithms are:

9.14.2 Replacement algorithms:

i). FIFO (First in first out):

The FIFO algorithm selects replacement the page that has been in memory the longest time. Each time a page is loaded into memory, its identification number is pushed into a FIFO stack.

* The page to be removed is easily identified because its identification number is on the top of the FIFO stack.

* The advantage is it can be easily implemented.

* The disadvantage is that under certain circumstances pages are removed and loaded from memory too frequently.

ii). LRU (Least recently used):

The LRU algorithm can be implemented by associating a counter with every page that is in main memory. When a page is referenced its associated counter is set to zero.

* At fixed intervals of time the counters associated with all the pages are incremented by 1. The least recently used page is the page with the highest count.

* The counters are often called aging registers, as their count indicate their age, that is how long ago their associated pages have been referenced.

* The advantage of LRU is that the least recently used page is a better candidate for removal than the least recently loaded page as in FIFO.

* The disadvantage is it is more difficult to implement.

9.15 Demand Paging:

In virtual memory systems, demand paging is a type of swapping in which pages of data are not copied from disk to RAM until they are needed. In contrast, some virtual memory systems use anticipatory paging, in which the operating system attempts to anticipate which data will be needed next and copies it to RAM before it is actually required.

i). Segmentation:

Memory segmentation is the division of a computer's primary memory into segments or sections of various sizes.

Different segments may be created for different program modules, or for different classes of memory usage such as code and data segments. Certain segments may be shared between programs.

These segments may be individually protected or shared between processes. Segmentation offers better security since the segments are pre-defined and we can avoid spurious reads/writes at the time of address translations etc.

ii). Data Stripping:

Data striping is the technique of segmenting logically sequential data, such as a file, so that consecutive segments are stored on different physical storage devices.

Striping is useful when a processing device requests data more quickly than a single storage device can provide it. By spreading segments across multiple devices which can be accessed concurrently, total data throughput is increased.

It is also a useful method for balancing I/O load across an array of disks. Striping is used across disk drives in redundant array of independent disks (RAID) storage, network interface controllers, different computers in clustered file systems and grid-oriented storage, and RAM in some systems.

9.16 Questions:

1. Explain the memory hierarchy in detail?
2. Compare and contrast Asynchronous DRAM and Synchronous DRAM
3. Clock skewing
4. Explain how the Bit Cells are organized in a Memory Chip
5. Explain the organization of a 1K x 1 Memory with a neat sketch
6. Explain the construction of semiconductor RAM and ROM memories. Also mention their advantages and applications.
7. Explain the following memories with their applications.
ROM
PROM
EPROM
EEPROM
8. What is Cache memory? Describe about locality of reference in detail.
9. Explain briefly about auxiliary memories
10. Explain the concept of content addressable memory. How the time for read/write operation is minimised in associative memory?
11. Explain the different mapping techniques used in the usage of Cache memory. Direct, associative and set associative.
12. With suitable examples, explain two-way set associative mapping and four-way set associative mapping.
13. Explain various Cache Memory Mapping Techniques.
14. What is Virtual Memory? Give relation between address and memory space in Virtual Memory System.
15. Explain the working of Address (page) translation mechanism.
16. Explain virtual memory, page file, Page fault and replacement: address translation.
17. What is the need of Replacement Algorithms for a Cache Memory?
18. Explain briefly about: Demand Paging, Segmentation
19. Data Stripping
20. a). A block set-associative cache consists of a total of 64 blocks divided into four-block sets. The main memory contains 4096 blocks, each consists of 128 words.
b). How many bits are there in a main memory address?
How many bits are there in each of the TAG, SET, and WORD fields? What is the size of cache memory?

Multiple choice questions:

1. The fastest chip to store and retrieval data ……..
a). Cache b) DRAM c). Register d). All
2. Memory which does not lose its contents when the power is switched off .....
a) RAM b) Cache c) NVRAM d) None
3. Which memory needs ultra violet rays to erase its contents .....
a). ROM b) PROM c). EPROM d) EEPROM.
4. Page file resides in……
a) RAM b) CD c) Hard disk d) None.
5. Associated memory is accessed by…….
a) Data b) Instructions c) Content d) None

Fill in the blanks:

1. In …… protocol the cache and main memory are updated simultaneously.
2. In cache memory hit ratio is the number of.....
3. A virtual memory system is a combination of ……. and ……. techniques.
4. Which mapping techniques use multiple tags and single index …….
5. …….. storage devices use pits and lands.
6. Pen drives are …… storage devices.
7. Cache memory is faster than dram because it is made up of …….
8. Synchronous DRAM is …… than Asynchronous DRAM because .....
9. Which cache write protocol reduce memory bandwidth .....
10. Which secondary storage is cheaper with respective to cost per MB space ……..

This page intentionally left blank.

10. CHARACTERISTICS OF MULTI PROCESSORS

10.1 Characteristics of Multi processors:

A multiprocessor system is an interconnection of two or more CPUs, with one or more IOP's. Multiprocessors are classified as multiple instruction multiple data stream (MIMD) systems.

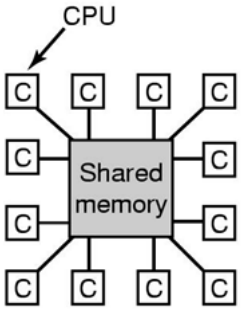
The main difference between the multiprocessor and multicomputer system is the first consists of multiple processors, where as the second one is an inter connection of multiple computers which form a computer network.

The main advantage of multiprocessor organisation is an improved performance, this is done by decomposing a program into parallel executable tasks. This can be achieved either by one of the below process.

* Multiple independent jobs can be made to operate in parallel.
* A single job can be partitioned into multiple parallel tasks.

10.2 Multi processor classification: Multi processors are classified by the way their memory is organised.

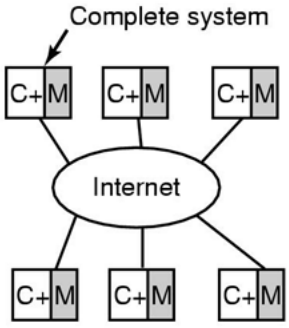
i). Shared memory or tightly coupled multiprocessor:



A multiprocessor system with common shared memory (common global memory) is classified as a shared memory or tightly coupled multiprocessor.

Tightly-coupled multiprocessor systems contain multiple CPUs connected through high speed bus. These CPUs have access to a central shared memory and low communication latency.

ii). Distributed memory or loosely coupled multiprocessor:



Each processor element in a loosely coupled system has its own private local memory. The processors are tied together and relay program and data to other processors in packets.

Loosely-coupled multiprocessor systems are stand alone (single or dual processor) systems interconnected via a high speed communication system.

Loosely coupled systems are most efficient when the interaction between tasks is minimal, where as tightly coupled systems can tolerate a higher degree of interaction between tasks.

10.3 Inter connection services: There are several physical forms available for establishing an interconnection network

i). Time shared common bus
ii). Multiport memory
iii). Crossbar switch
iv). Multistage switching network
v). Hypercube system

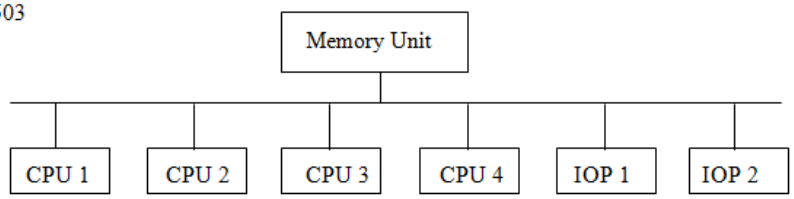
i). Time shared common bus: A multiprocessor system with number of processors connected through a common bus to a memory unit. Only one processor can communicate with the memory or another processor at any given time as shown in fig RF 503.

Any other processor wishes to initiate a transfer, it must check the bus status and if it is idle it can address the destination and initiate transfer. When more than one processor start using the bus then conflicts may arise, which are can be sorted out using a bus controller.

Disadvantage:

The disadvantage in this system is if one processor is communicating with the memory, all other processors are either busy or idle waiting for bus. Due to this overall transfer rate is limited by the speed of the single path.

RF 503

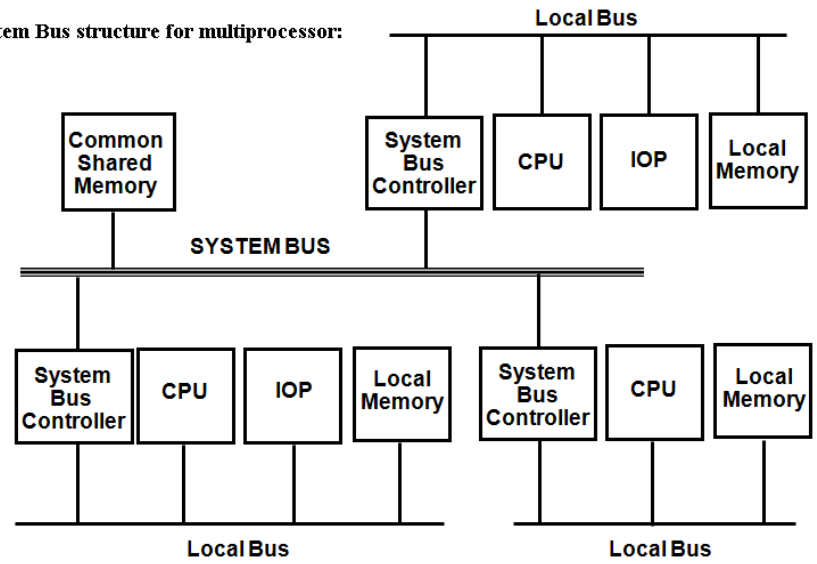


System Bus structure for Multiprocessor:

An alternative method is to use two more independent busses to permit multiple simultaneous bus transfers.

The figure RF 504 illustrates the dual bus structure, one local bus to connect to its local memory and the second bus is to connect other processors.

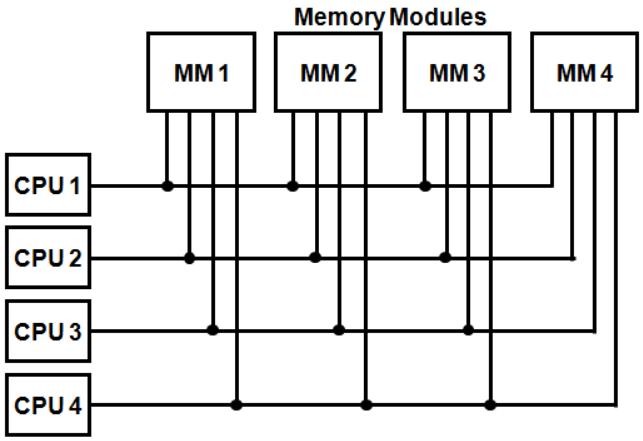
RF 504: System Bus structure for multiprocessor:



ii). Multiport memory:

A multiport memory system employs separate busses between each memory module and each CPU as shown in the fig RF505.

RF 505: Multiport Memory Organisation.



A processor bus consists of the address, data and control lines required to communicate with memory.

The memory module consists of four ports and each port accommodates one of the busses.

The module must have internal logic to determine which port will have access to memory at any given time.

The priority for memory access can be established by the physical port position. Thus CPU1 will have priority over CPU2 and so on.. CPU 4 will have lowest priority.

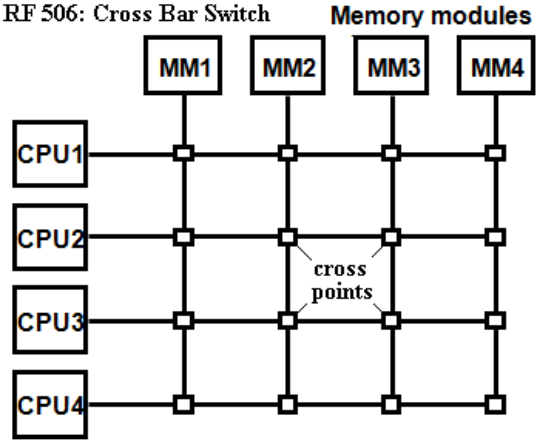
Advantage: Higher data transfer rate can be achieved because of multiple paths between the processors and memory.

Disadvantage: It requires expensive memory control logic and a large number of cables and connectors.

iii). Crossbar switch:

The cross bar switch organisation consists of a number of cross points that are placed at intersections between processor busses and memory module paths. The below fig RF 506. shows a crossbar switch interconnection between four CPUs and four memory modules.

RF 506: Cross Bar Switch



The small square in each cross point 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 which module need to be addressed. It also resolves multiple requests for access to the same memory module on a predetermined priority basis.

Advantage: A cross bar switch organisation supports simultaneous transfers from all memory modules because there is a separate path associated with each module.

Disadvantage: The hardware required to implement switch can become quite large and complex.

iv). Multistage switching network:

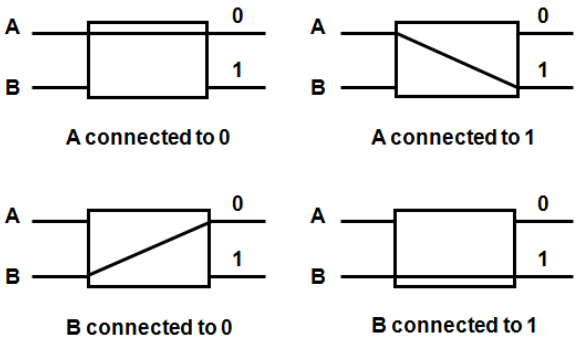
Multistage network is a two input two output interchange switch as shown in the fig. RF 507.

The 2 x 2 switch has two inputs labeled A and B, and two outputs labeled 0 and 1.

The switch has the capability of connecting input A (or B) to either of the outputs.

The switch can also arbitrate, if both the inputs A and B request the same output terminal, it allows only one input and blocks the other.

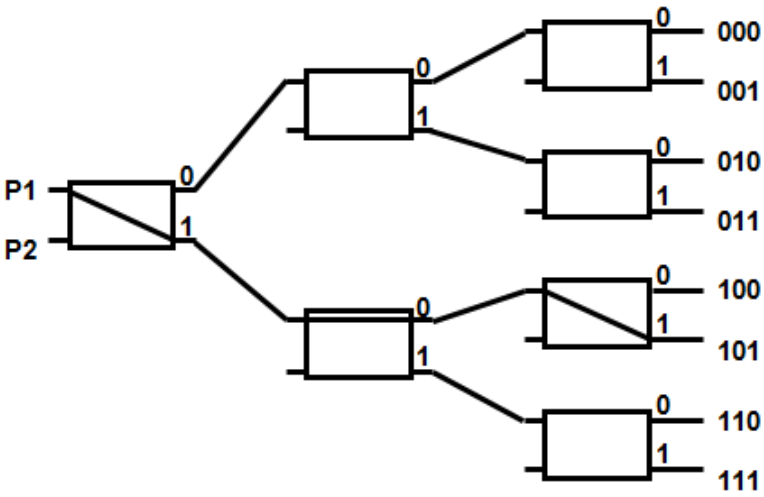
RF 507: Operation of 2 x 2 interchange Switch



Using 2 x 2 switch it is possible to build a multistage network to control the communication between a number of sources and destinations.

Ex: Binary tree with 2 x 2 switches

RF 508: Binary Tree with 2 x 2 switches

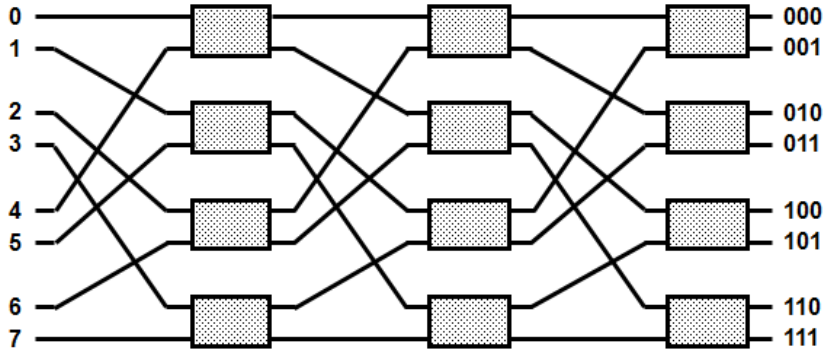


Omega network:

Omega network is one topology used in the multistage switching networks to control processor memory communication in a tightly coupled multiprocessor system.

In this configuration there is exactly one path from each source to any particular destination as shown in the fig RF 509.

RF 509: 8 x 8 Omega Switching Network.



Some request patterns however cannot be connected simultaneously.

For example any two sources cannot be connected simultaneously to destinations 000 and 001.

v). Hypercube system:

The hypercube or binary n-cube multiprocessor structure is a loosely coupled system composed of $N=2^n$ processors interconnected in an n-dimensional binary cube. Each processor forms a node of the cube.

A one cube structure has n=1 so $2^n = 2$ means it contains two processors interconnected by a single path.

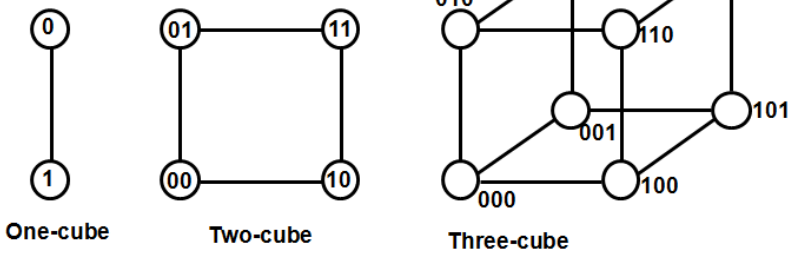
A two cube structure has n=2 means $2^n = 4$.

It contains four nodes interconnected as a square.

A three cube has eight nodes.

A n cube structure has 2^n nodes with a processor residing at each node.

RF 510: Hypercube Inter connection



Each node is assigned a binary address in such a way that the addresses of two neighbours differ in exactly one bit position.

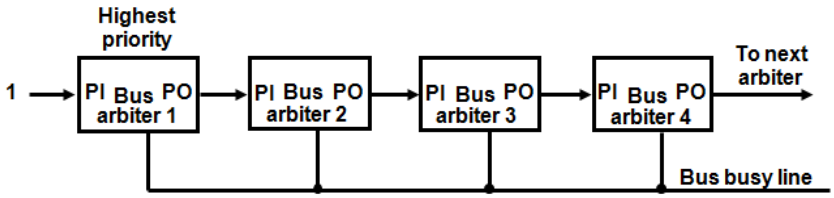
For example node 100 has neighbours are 000, 110 and 101.

10.4 Inter-processor Arbitration:

10.4.1 Static Arbitration Algorithms: Arbitration procedures service all processor requests on the basis of established priorities. Two types of arbitration procedures Serial and Parallel procedures are discussed below.

a). Serial arbitration procedure:

The serial priority resolving technique is obtained from a daisy chain connection as shown in the figure.



The device closest to the priority line is assigned the highest priority. When multiple devices concurrently request the use of the bus, the device with the highest priority is granted access to it. When the arbiter takes the bus it makes the bus line busy.

When a higher priority processor requests the bus, the lower-priority processor must complete its bus operation before it relinquishes control of the bus.

When the bus busy line returns to its inactive state, the higher-priority arbiter enables the busy line, and its corresponding processor can then conduct the required bus transfers.

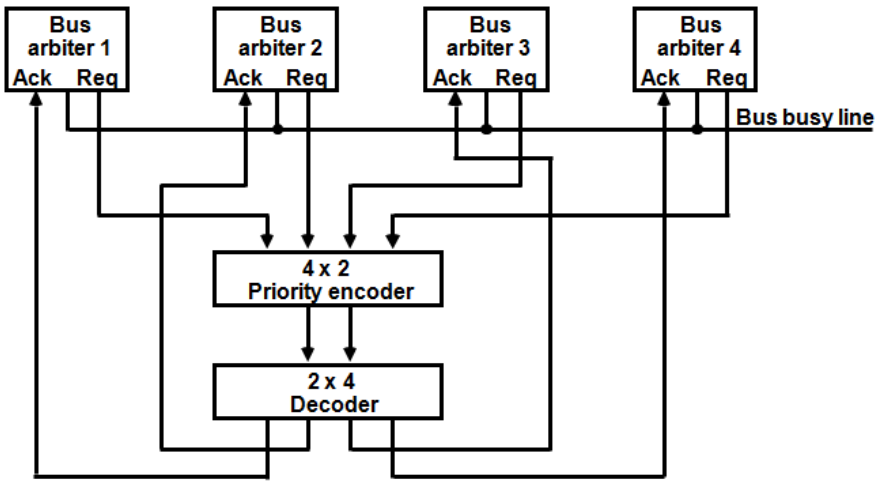
b). Parallel arbitration procedure:

The parallel bus arbitration technique uses an external priority encoder and a decoder as shown in the below figure.

Each bus arbiter in the parallel scheme has a bus request (Req) output line and a bus acknowledge (Ack) input line.

When its processor requests, arbiter enables the request to access the system bus.

The processor takes control of the bus if it gets the acknowledgement. The bus busy line provides an orderly transfer of control, as in the daisy-chaining case.



Above figure shows the request lines from four arbiters going into a 4 x 2 priority encoder. The output of the encoder generates a 2-bit code which represents the highest-priority unit among those requesting the bus. The 2-bit code from the encoder output drives a 2 x 4 decoder which enables the proper acknowledge line to grant bus access to the highest-priority unit.

10.4.2 Dynamic Arbitration Algorithms:

A Dynamic priority algorithm gives the system the capability for changing the priority of the devices while the system is in operation. (The serial and parallel arbitration techniques are static).

i). Time slice: (No priorities)

The time slice algorithm allocates a fixed length time slice of bus time that is offered sequentially to each processor in round robin fashion.

No preference is given to any particular device since each is allotted the same amount of time to communicate with the bus.

ii). Polling: (Priority software based)

Polling is a process of continuous checking the connected devices by a program (or device) to see whether they want to communicate.

A *polling* procedure is used to identify the highest priority source by software means. The order in which they are tested determines the priority of each interrupt. The highest priority source is tested first, followed by next priority and so on.

iii). LRU: (Priority for unused)

The least recently used (LRU) algorithm give the highest priority to the requesting device that has not used the bus for the longest interval. The priorities are adjusted after a number of bus cycles according to the LRU algorithm.

The priorities are dynamically changed to give every device an opportunity to access the bus.

iv). FIFO:

In this scheme requests are served in the order received, first cum first based. Each processor must wait for its turn to use the bus on a first in first out (FIFO) basis.

v). Rotating daisy chain: (loop)

The rotating daisy chain procedure is a dynamic extension of the daisy chain algorithm. There is no central bus controller and the priority out (PO) of first device connected to the priority in (PI) of the next device and so on.

The last device priority out is connected to the input of the first device thus forming a loop. Whichever device has access to the bus serves as a bus controller.

10.5 Inter-processor Communication:

Various processors in a multiprocessor system must be provided with a facility for communicating with each other.

a). Shared (common) memory: The shared (common) memory is used as a message centre, where each processor can leave message for other processors and pick up messages intended for it.

b). IOP: In addition to shared memory a multiprocessor system may have other shared resources like magnetic storage disk connected through an IOP.

A communication path between two CPUs can be established through a link between the two IOPs associated with two different CPUs. This type of link allows each CPU to treat the other as an I/O device so that messages can be transferred through I/O path.

10.6 Shared resource conflicts:

Operating system takes care of the shared resource conflicts either through one of the methods:

i). Master slave configuration: One processor designated as the master always executes the operating system functions. The remaining processors act as slaves.

If a slave processor needs an operating system service it must request it by interrupting the master.

ii). Separate operating system: This is more suitable for loosely coupled systems where each processor may have its own copy of the entire operating system.

Each processor can execute the operating system routines it needs.

iii). Distributed operating system (or floating operating system):

Here the operating system routines are distributed among the available processors. However each particular operating system function is assigned to only one processor at a time.

* In a loosely coupled microprocessors system the memory is distributed among the processors and there is no shared memory for passing information. The communication between processors is by means of message passing through I/O channels.

* The communication is initiated by one processor calling a procedure that resides in the memory of the processor with which it wishes to communicate.

Once the sending processor and receiving processor name each other as a source and destination a channel of communication is established.

10.7 Inter processor synchronization:

Synchronisation refers to the special case where the data used to communicate between processors is control information. Synchronisation is needed to enforce the correct sequence of processors.

Multiprocessor systems usually include various mechanisms to deal with the synchronisation of resources.

a). Mutual Exclusion:

Mutual exclusion is a mechanism used to protect data from being changed simultaneously by two or more processors.

This mechanism enables one processor to make use of a shared resource and lock out access to other processors when it is in *critical section.*

b). Critical section:

A critical section is a program sequence that once begun, must complete execution before another processor accesses the same shared resource.

c). Semapore:

A binary variable called a semaphore (Hardware mechanism) is often used to indicate whether or not a processor is executing a critical section.

A semaphore is a software controlled flag. Processors when they are executing a critical section set semaphore to (1) and clear (0) when they are finished.

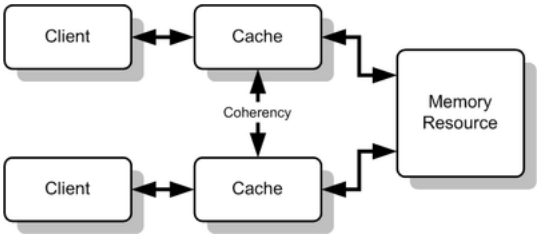
d). Hardware lock:

A hardware lock is a processor generated signal that serves to prevent other processors from using the system bus as long as the signal is active.

The test and set instruction, tests and sets a semaphore and activates the lock mechanism during the time that the instruction is being executed.

This prevents other processors from changing the semaphore between the time that the processor is testing it and the time that it is setting it.

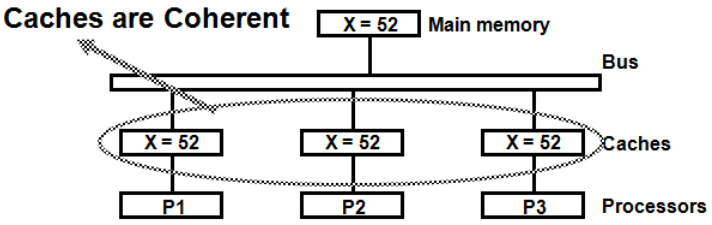
10.8 Cache Coherence: (consistency)



Cache coherence refers to the consistency of data stored in local caches of a shared (memory) resource.

Cache Incoherence (Inconsistency): For ex: Referring to the figure, if the top client has a copy of a memory block from a previous read and the bottom client changes that memory block, the top client could be left with an invalid cache of memory without any notification of the change. This leads to cache inconsistency.

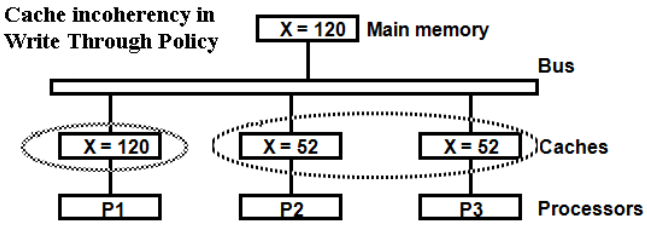
10.8.1 Conditions for Incoherence:



To illustrate the problem consider the three processor configuration with three private caches as shown in the fig.

Let us take the value X= 52 in the main memory. When this value is loaded into the three processors P1, P2 and P3, it is also copied into their respective caches. The load of X to the three processors results in consistent copies in the caches and main memory.

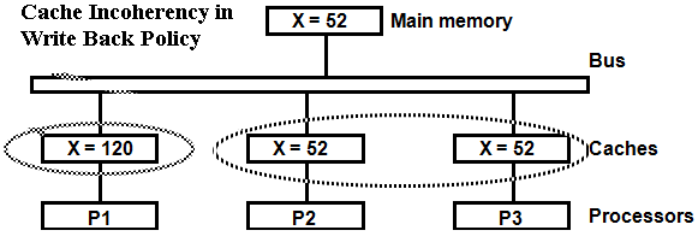
Case 1: Write Through Policy.



In WT policy all write operations are made to main memory as well as to the cache.

In the fig. the process P1 using write through policy updated both its cache memory and the main memory with the value X=120. The remaining processors (P2 and P3) are not aware of this change. This leads to inconsistent among other two caches, since they hold the old value.

Case 2: Write Back Policy:



In WB policy write operations are usually made only to the cache.

Main memory is only updated when the corresponding cache line is flushed from the cache.

In the figure as per write back policy the processor P1 updates its cache with X value to 120. The other processors (P2 and P3) are not aware of this change and their caches are not updated including the main memory, which remain inconsistent. Cache coherence is intended to manage such conflicts and maintain consistency between cache and memory.

10.9 Solutions for Cache coherence problems:

i). Software based solutions:

a). Disallow private caches (Use shared Cache):
b). Allow only non shared data in the cache:
c). Central global table:

ii). Hardware based schemes:

a). Directory protocol and
b). Snoopy protocols. i). Write invalidates or
ii). Write- update (write-broadcast)

i). Software based Solutions:

a). Disallow private caches (Use shared Cache):

A simple scheme is to disallow private caches for each processor and have a shared cache memory associated with main memory. Every data access is made to the shared cache.

* But this increases the average access time. For performance considerations it is desirable to attach a private cache to each processor.

b). Allow only non shared data in the cache:

Another scheme is to allow only non shared and read only data to be stored in cache. The system hardware makes sure that only cacheable data are stored in caches. The non cacheable data remain in main memory.

* This method restricts the type of data stored in caches and introduces an extra software overhead that may degrade performance.

c). Central global table:

A scheme that allows writable data to exist in at least one cache is a method that employs a centralized global table in its compiler.

The status of memory blocks is stored in the central global table. Each block is identified as read-only (RO) or read and write (RW). All caches can have copies of blocks identified as RO. Only one cache can have a copy of an RW block.

Thus if the data are updated in the cache with an RW block, the other caches are not affected because they do not have a copy of this block.

ii). Hardware solutions:

Hardware solutions provide dynamic recognition at run time of potential inconsistency conditions and are more effective, leading to improved performances over a software approach.

Hardware schemes can be divided into two categories:

a). Directory protocol and
b). Snoopy protocols.

a). Directory protocols: Directory protocols maintain a centralized controller with a directory information about which processors have a copy of which lines in the main memory. Before a processor can write to a local copy of a line, it must request exclusive access to the line from the controller.

Before granting the exclusive access, the controller sends a message to all processors to invalidate its copy. After receiving acknowledgement back from all the processors, the controller grants exclusive access to the requesting processor.

Disadvantage: Directory schemes suffer from the drawbacks of a central bottleneck and the overhead of communication between the various cache controllers and the central controller.

b). Snoopy Protocols: Snoopy protocols distribute the responsibility for maintaining cache coherence among all of the cache controllers in a multiprocessor system.

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 observe these broadcasted notifications and react accordingly.

Two basic approaches to the snoopy protocol have been explored:

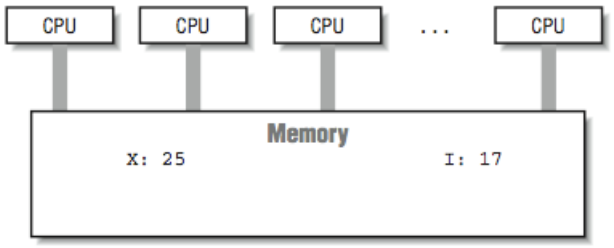
i). Write invalidates or
ii). Write- update (write-broadcast)

i). Write invalidate protocol: When one of the caches wants to perform a write to the line it first issues a notice that invalidates that line 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.

ii). 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.

10.10 Shared memory multiprocessors:



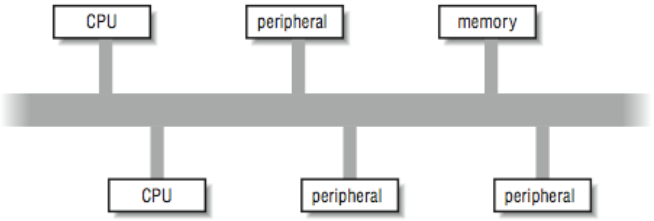
The primary advantage of shared-memory multiprocessor systems is the ability for any CPU to access all of the memory and peripherals.

Furthermore, the systems need a facility for deciding among themselves who has access to what, and when, which means there will have to be hardware support for arbitration.

The two most common architectural underpinnings for symmetric multiprocessing are:

1. Bus approach
2. Crossbar approach.

i). Bus Approach:

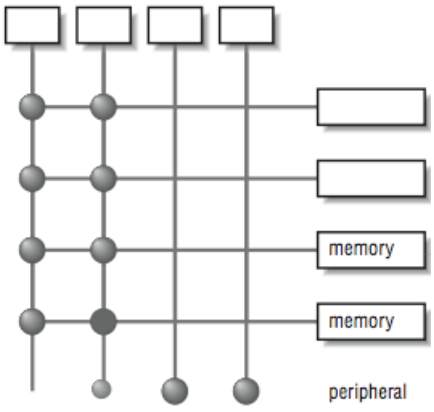


The bus is the simplest of the two approaches.

It is less expensive to build, but because all traffic must cross the bus, as the load increases, the bus eventually becomes a performance bottleneck.

ii). Cross Bar Approach:

A crossbar is a hardware approach to eliminate the bottleneck caused by a single bus. Any module can get to any other by a path through the crossbar, and multiple paths may be active simultaneously.



In the 4×5 crossbar, for instance, there can be four active data transfers in progress at one time.

Crossbar connect parties that wish to communicate, it also actively arbitrate between two or more CPUs that want access to the same memory or peripheral.

Crossbars have the best performance because there is no single shared bus.

However, they are more expensive to build, and their cost increases as the number of ports is increased. Because of their cost, crossbars typically found only in high performance systems.

10.11 Questions:

1. Explain the characteristics of multiprocessors.
2. Draw and explain the distributed shared memory architecture.
3. Draw and explain the typical multiprocessor organization.
4. Distinguish between tightly coupled microprocessors and loosely coupled Microprocessors.
5. List and explain different interconnection structures used in multiprocessors?
6. Explain system bus structure for multiprocessors with a neat sketch.
7. Explain multi port memory organization with a neat sketch.
8. Explain the functioning of Binary Tree network with 2 x 2 Switches. Show a neat sketch.
9. Draw and explain 1 to 8 ways switching using basic 2x2 switch. (Explain the functioning of omega switching network with a neat sketch).
10. Explain Processor arbitration in Multiprocessors
11. Inter processor arbitration
12. Inter process communication.
13. What is the need of Inter processor Synchronization? Explain.
14. Inter processor synchronization. (mutual exclusive, critical sec, semaphore, Hard. lock).
15.What is mutual exclusion and critical section of program? Explain the process of mutual exclusion with a semaphore.
16. What is cache coherence? Explain its importance in shared memory multiprocessor systems?
17. Cache coherence. Solutions for cache incoherence briefly.
18. Describe in detail about shared memory multiprocessors.

Multiple choice questions:

1. In a 2 x 2 switch network each input can be connected to …… outputs
a). 1 b) 2 c). 3 d). 4
2. In polling the order of procedure is based on .....
a) No priorities b) Highest priority c) Low priority d) None
3. Shared resource conflicts are taken care by .....
a). Master slave configuration b) Separate OS c). Distributed OS d) all.
4. Semapore is a ……
a) program b) routine c) hardware flag d) None.
5. Parallel arbitration procedure in inter processor arbitration is ……. Procedure.
a) Static b) Dynamic c) Specific d) None

Fill in the blanks:

1. Tightly coupled microprocessors use …….. memory
2. The advantage of cross bar switch over multiport switch is.....
3. Omega network is a ……. Stage network.
4. In hyper cube system each node should have a …….. address.
5. In inter processor communication, path between two CPUs is established through a link between the ……..
6. Mutual exclusion is used to …….

7. A program sequence that once begun, must complete execution before another processor accesses the same shared resource is known as ……...
8. Cache incoherence occurs when …….....
9. Disallow private cache can avoid .....
10. The disadvantage of bus approach in multiprocessor system is …. ……..

This page intentionally left blank.

COMPUTER ORGANISATION

CC: GR15A2076, B.Tech 2/2

SYLLABUS

Unit-IA: Basic Structure of Computers: Computer Types, Functional unit, Data Representation, Fixed Point Representation, Floating – Point Representation, Error Detection codes.

Unit-IB: Register Transfer Language and Micro operations: Register Transfer language, Register Transfer, Bus and memory transfers, Arithmetic Micro operations, Logic micro operations, Shift micro operations, Arithmetic logic shift unit.

Unit-IIA: Basic Computer Organization and Design: Instruction codes, Computer Registers, Computer instructions, Timing and Control, Instruction cycle, Memory Reference Instructions, Input – Output and Interrupt, Complete Computer Description.

Unit-IIB: Micro Programmed Control: Control memory, Address sequencing, micro program example, design of control unit, Micro program Sequencer, Hard wired control Vs Micro programmed control.

Unit-IIIA: Central Processing Unit Organization: General Register Organization, STACK organization, Instruction formats, Addressing modes, DATA Transfer and manipulation, Program control, Reduced Instruction Set Computer.

Unit-IIIB: Computer Arithmetic: Addition and subtraction, multiplication Algorithms, Floating – point Arithmetic operations, BCD Adder.

Unit-IVA: Input-Output Organization: Peripheral Devices, Input-Output Interface, Asynchronous data transfer Modes of Transfer, Priority Interrupt, Direct memory Access, Input –Output Processor (IOP).

Unit-IVB: Pipeline and Vector Processing: Parallel Processing, Pipelining, Arithmetic Pipeline, Instruction Pipeline, Dependencies, Vector Processing.

Unit-VA: Memory Organisation: Memory Hierarchy, Main memory- RAM and ROM chips, Memory Address map, Auxiliary memory – Magnetic Disks, Magnetic Tapes, Associative Memory – Hardware Organization, Match Logic, Cache Memory – Associative mapping, Direct mapping, Set associative mapping, Writing into cache and cache initialization, Cache Coherence, Virtual memory – Address Space and Memory Space, Address mapping using pages, Associative Memory page table, Page Replacement.

Unit-VB: Multi Processors: Characteristics or Multiprocessors, Interconnection Structures, Cache Coherence, Shared Memory Multiprocessors.

This page intentionally left blank.

MODEL QUESTION PAPERS

This page intentionally left blank.

II B.Tech II Semester Regular Examinations, MAY 2018
Computer Organisation
(Dept. of CSE)

Time: 3 hours **Max Marks: 70**

PART – A
Answer ALL questions. All questions carry equal marks
*****
2 * 10 = 20 Marks

1).	a	Explain about MIPS, FLOPS rating of a processor.	2M
	b	What is an Error Detection Code?	2M
	c	Differentiate between Hardwired control and micro programmed control	2M
	d	List all basic Computer Instructions	2M
	e	What is a Stack? Explain STACK organisation with neat diagram	2M
	f	Elicit various types of instruction formats	2M
	g	Explain Flynn's taxonomy	2M
	h	Draw Space Time diagram for Pipeline	2M
	i	Explain the memory hierarchy in terms of cost and performance.	2M
	j	What is Cache memory? Describe about locality of reference in detail.	2M

PART – B
Answer any FIVE questions. All questions carry equal marks

*****

10 * 5 = 50 Marks

2.	a). List and explain different types of computers with examples. Also mention their merits and demerits. b). Perform the arithmetic operations in binary using 2's complement representation for negative numbers. i). (+70)+(+80) and (-70)+(-80) ii). (+42) + (-13) and (-42)-(-13).	5M 5M
3.	a). Explain about instruction cycle with the help of the example. b). Explain micro programmed control organisation.	5M 5M
4.	a). What is an addressing mode? Explain different addressing modes. b), Explain Data transfer and manipulation commands.	5M 5M
5.	a). Explain Asynchronous Communication interface with neat diagram. b). What is priority interrupt? Briefly explain different types of priority interrupts	5M 5M
6.	a) Explain the different mapping techniques used in Cache memory. b). What is the need of Replacement Algorithms for a Cache Memory? Explain any two cache replacement strategies.	5M 5M
7.	a). Explain in detail the fixed point representation b) Explain normalisation in floating point arithmetic with example.	5M 5M
8.	a). What is instruction pipeline. b). Explain various types of ROMs and their advantages and disadvantages	5M 5M

This page intentionally left blank.

II B.Tech II Semester Regular Examinations, MAY 2018
Computer Organisation
(Dept. of CSE)

Time: 3 hours **Max Marks: 70**

PART – A
Answer ALL questions. All questions carry equal marks
*****

2 * 10 = 20 Marks

1). a	Explain PC and IR registers	2M
b	Differentiate synchronous and asynchronous bus.	2M
c	Compare horizontal and vertical organization	2M
d	Explain nanoprogram and nanoinstructions	2M
e	What is overflow and underflow?	2M
f	Explain difference types of interrupts.	2M
g	Explain the differences between memory mapped, Isolated I/O method	2M
h	Explain about Input output Interface unit with an example	2M
i	Distinguish between tightly coupled microprocessors and loosely coupled Microprocessors.	2M
j	Explain demand Paging, Segmentation	2M

PART – B
Answer any FIVE questions. All questions carry equal marks

*****

10 * 5 = 50 Marks

2.	a). List and explain different performance measures used to represent a computer system performance. b). Explain the implementation of common bus and memory transfers using multiplexers	5M 5M
3.	a). Explain the organization and functioning of Micro program sequencer. b). Formulate a mapping procedure that provides eight consecutive micro instructions for each routine. The operation code has 7 bits and control memory has 2048 words.	5M 5M
4.	a). Explain General Register Organisation with neat diagram b). Explain BCD adder with a block diagram	5M 5M
5.	a). What is meant by pipelining? Explain Pipeline conflicts	10M
6.	a) What is cache coherency? Why is it necessary? Explain different approaches for cache coherency b). List and explain different interconnection structures used in multiprocessors?	5M 5M
7.	a). Explain in detail shift microoperations b) a) Give a detailed explanation of Data Representation in digital Computer with suitable examples.	5M 5M
8.	a). What is Direct Memory Access (DMA)? What is the need for DMA? b). Explain Inter process communication.	5M 5M

This page intentionally left blank.

II B.Tech II Semester Regular Examinations, MAY 2018
Computer Organisation
(Dept. of CSE)

Time: 3 hours **Max Marks: 70**

PART – A
Answer ALL questions. All questions carry equal marks
*****

2 * 10 = 20 Marks

1).	a	Explain RTL and the control function associated with it.	2M
	b	What is the significance of Bias in IEEE 754 operations	2M
	c	What is microinstruction	2M
	d	Briefly explain address sequencing	2M
	e	Advantages and disadvantages of RISC architecture.	2M
	f	Elicit program control instructions**.**	2M
	g	Explain DMA transfer in detail	2M
	h	What is an Input-Output processor? Explain the need for Input-Output processor	2M
	i	Differentiate multi processors and multicomputers	2M
	j	What is Virtual Memory? Give relation between address and memory space in Virtual Memory System	2M

PART – B
Answer any FIVE questions. All questions carry equal marks

*****

10 * 5 = 50 Marks

2.	a). Define Computer. What are Functional Units? With a neat diagram explain them. b). Draw and explain Arithmetic Circuit Unit.	5M 5M
3.	a). Elucidate common bus system b). Explain interrupt cycle with the help of the example	5M 5M
4.	a). Draw a flow chart for Floating point Add/subtract operations b). Describe types of CPU Organisations	5M 5M
5.	a). List and explain different asynchronous data transfer modes b). Draw and explain the block diagram of typical DMA controller.	5M 5M
6.	a) Explain Processor arbitration in Multiprocessors b). What is the need of Inter processor Synchronization? Explain.	5M 5M
7.	a) Explain the concept of three state Bus Buffers b) What is instruction pipeline	5M 5M
8.	a). Explain different notations. Which notation is normally used in computers b). Draw and explain the hardwired control unit.	5M 5M

This page intentionally left blank.

II B.Tech II Semester Regular Examinations, MAY 2018
Computer Organisation
(Dept. of CSE)

Time: 3 hours **Max Marks: 70**

PART – A
Answer ALL questions. All questions carry equal marks
*****

2 * 10 = 20 Marks

1). a	Differentiate single bus and multiple bus structure	2M
b	Explain logic micro operations	2M
c	Explain why hardwired control is faster than micro programmed control	2M
d	Explain branch instructions with example	2M
e	Subtract $(101011)_2$ from $(111001)_2$ using the 1's complement and 2's complement method	2M
f	Differentiate RISC and CISC**.**	2M
g	a). What is RISC pipeline,	2M
h	b). Explain the i). Vector processing ii). Array processor	2M
i	Explain the concept of content addressable memory.	2M
j	Explain cache write through and write back protocols.	2M

PART – B
Answer any FIVE questions. All questions carry equal marks

*****

10 * 5 = 50 Marks

2.	a). Explain IEEE 754 standard representation with example b). Represent $(1259.125)_{10}$ in single precision and double precision IEEE 754 standards.	5M 5M
3.	a). Explain stored program organisation b). Explain complete computer description with the help of flow chart	5M 5M
4.	a). Explain in detail with neat sketch Booth Multiplication Algorithm. b). Explain one stage decimal arithmetic unit.	5M 5M
5.	a). List and explain different I/O communication techniques. Also mention their advantages and disadvantages. b). Explain Arithmetic Pipeline for floating point addition and subtraction.	5M 5M
6.	a) Explain the working of Address (page) translation mechanism in virtual memory. b) Draw and explain 1 to 8 ways switching using basic 2x2 switch..	5M 5M
7.	a) Give a detailed explanation of Data Representation in digital Computer with suitable examples. b) What is a Stack? Explain STACK organisation with neat diagram	5M 5M
8.	a). List any explain memory reference instructions b). Explain the practical considerations in supercomputer design	5M 5M

This page intentionally left blank.