LECTURE NOTES

ON

COMPUTER ORGANIZATION & ARCHITECTURE

4th SEMESTER

Sunanda Kumar Sahoo

ASST. PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GANDHI INSTITUTE OF TECHNOLOGY AND MANAGEMENT (GITAM)

Affiliated to BPUT & SCTE&VT, Govt. of Odisha

Approved by AICTE, New Delhi

(PCCS4301) COMPUTER ORGANIZATION (3-0-0)

CH-1 Module-1 (12 Hrs).

Basic Structures of Computers: Functional Units, operational 1.2 Concepts, Bus structures, Software, Performance, Computer 1.4 Architecture vs Computer Organization.

1.6 Machine Instruction and Programs: Memory Location and CH-2 addresses, Big-endian and little-endian representation. Memory operations, Instructions and instruction Sequencing, 2.2 - 2.10 Addressing modes, Assembly Language, Basic Input/output operations, subroutine, additional Instructions.

Module-2 (12 Hrs)

CH-6 Arithmetic: Addition and Subtraction of Signed numbers, 6.1 Design of Fast Adders, Multiplication of positive numbers, - 6.7 Signed-operand multiplication, Fast multiplication, Integer Division, Floating-point Numbers, and operations.

CH-7 Module-3. (12 Hrs)

7.1 Basic processing Units: Fundamental Concepts, execution of - 7.5 Complete Instructions, MultiBus Organization, Hardwired Control, Micro Programmed Control, RISC vs CISC architecture.

Memory System: Basic Concepts, Cache memory, Cache CH-5 memory mapping policies, Cache updating Schemes, Performance Consideration, virtual memories, paging - 5.9 and page replacement policies, Memory Management requirement, Secondary Storage.

Text book.

✓ 1. Computer Organization: Hamacher, ~~*~~ Vranesic, Zaky.
2. Computer organization and Design Hardware/Software Interface: David A. Patterson, John L. Hennessy.

Reference.

✓ 1. Computer Architecture and Organizations, Design Principles & application: B. Govinda Rajalu.
✓ 2. Computer System Architecture: Morris M Mano.

Computer Architecture:

Computer Architecture deals with giving operational attributes of the computer or processor to be specific. It deals with details like physical memory, ISA (Instruction Set Architecture) of the processor, the no. of bits used to represent the data types, Input output mechanism and technique for addressing memories.

Computer Organization:

Computer Organization is realization of what is specified by the computer architecture. It deals with how operational attributes are linked together to meet the requirements specified by computer architecture. Some organizational attributes are hardware details, control signals, peripherals.

Architecture and organization are independent, you can change the organization of a computer without changing it's architecture. For example, a 64-bit architecture canbe internally organized as a true 64-bit machine or as a 16-bit machine that uses four cycles to handle 64 bit values.

Architecture is the abstract view. for example Car architecture contains a gear box, where as the internal implementation is done at organization level.

64-bit Computing.

64-bit processors have datapath widths, integer size and memory address widths of 64bits. Also register length is 64bit.

MODULE-1

BASIC STRUCTURE OF COMPUTERS

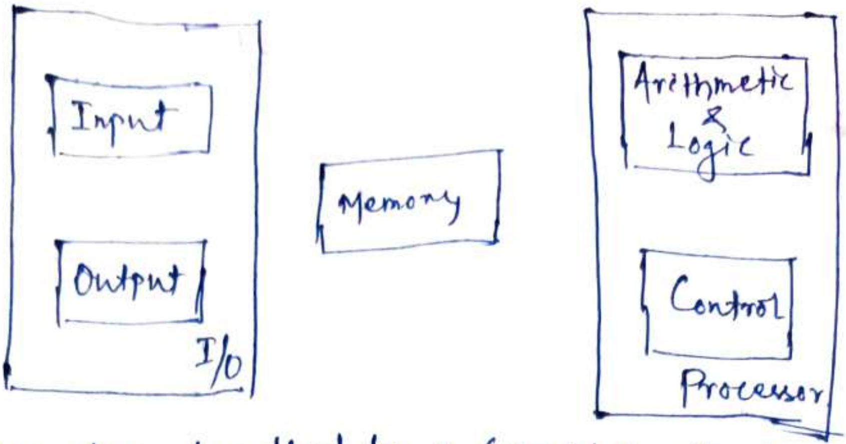
Types of Computer

A Computer is a fast electronic calculating machine that accepts digitized input information, processes it according to a list of internally stored instructions and produces the resulting output information. The list of instructions is called a computer program, and the internal storage is called computer memory.

Many types of computers exist that differ widely in size, cost, computational power, and intended use. Personal computer / desktop computers are commonly used in homes, schools and business offices. Notebook computers or Laptops are a compact version of the personal compute
Workstations with high resolution graphics input/output capability often used in engineering design work. These are of same size as desktop computers.
Enterprise Systems, or Mainframes are used for business data processing in medium to large corporations that require much more computing power and storage capacity than workstation can provide.
Servers contain sizable database storage units and are capable of handling large volume of request to access the data.
Super Computers are used for the large scale numerical calculations required in applications such as weather forecasting and aircraft design and simulation.

FUNCTIONAL UNITS

A Computer consists of five functionally independent main parts: input, memory, arithmetic and logic, output and control units. The input unit accepts coded information from electromechanical devices such as keyboards or from other computers over digital communication lines. These informations either stored or sent to ALU for desired operations. The processing steps are determined by a program stored in the memory. Finally, the results are sent back to the outside world through the output unit. All these actions are coordinated by the control unit.



Information handled by a Computer Canbe data or instructions. Instructions are the Commands to do Certain operations. Such as ALU operations or transfer data within a Computer or between the Computer. The Computer is Completely Controlled by the stored program, except possibly for external interruption. Data are numbers and encoded characters used for processing by a program. Even a set of instruction ~~a~~ program Canbe treated as data if it is used as input to another program. for example Source program used as data while ~~compilation~~ compiling it to generate object program.

ASCII (American Standard Code for Information Interchange) and EBCDIC (Extended Binary-coded Decimal Inperchange Code) are two Coding Schemes Used torepresent information in digital form. ASCII Uses a 7-bit Code nd EBCDIC Uses a 8-bit Code.

Input Unit

The most wellknown input device is the Keyboard. Whenever a Key ispressed, the corresponding letter ordigit is automatically translated into its Corresponding binary Code and transmitted over a cable toeither the memory or the processor. Other input devices are joysticks, trackballs nd mouses. These are often used ~~to capture~~ ~~audio~~ as graphic input devices in conjuction with displays

Memory Unit.

There are two classes of storage, called primary and secondary.

Primary memory is a fast memory, which contains a large no. of semiconductor storage cells, each capable of storing one bit of information, But information is read/write in group of bits, usually called word. Typical length of word ranges from 16 to 64 {16, 32, 64}

To provide easy access to any word, a distinct address is associated with each word location. Addresses are numbers that identify successive locations. Normally each byte (8bit) has unique addresses.

Primary memory refers to memory popularly known as RAM (random access memory). Memory in which any location can be reached in a short and fixed amount of time known as random access memory. The memory of a computer is normally implemented as a memory hierarchy of RAM units of different speeds and sizes. The small and fast RAM units are called caches and the largest and slowest unit is refererred to as the mainmemory. Cache is often present on the same I.C chip of processor to achieve high performance.

Although primary memory is essential as programs must be stored in the primary memory while they are being executed, it tends to be expensive. Thus additional, cheper secondary storage is used for large amount of data. Magnetic disks, tapes, optical disks (CD-ROM) and harddisks are examples of secondary memory.

Arithmatic and Logic Unit

Most computer operations such as addition, multiplication, division, comparison etc. are executed in the arithmatic and logic unit. The required operands are brought into processor and stored in highspeed storage elements called registers.

The control unit and ALU are faster than other devices connected to a computer system. This enables a single processor to control such devices like keyboards, displays etc.

Output Unit

The output unit is the counterpart of the input unit. It's function is to send processed results to the outside world. Printer and display unit are two such example. Display unit a the graphic display unit also ~~display~~ helps in inputing data. That's why some use I/O unit for graphic display.

Control Unit

The memory, ALU and Input and output units store and process information and perform input and output operations. The operation of these units are coordinated by the control unit. The control unit is effectively the nerve center that sends control signals to other units and senses their states. Control unit generate a timing signals and based on timing signal CU controls these units operations.

The operation of a computer can be summarized

→ It accepts information in the form of data and programs through an input unit and stores in memory.

→ Information stored in memory is fetched under Program control into ALU, where it is processed.

→ Processed information given to outside through O/P unit.

→ All activities inside the machine are controlled by control unit.

BASIC OPERATIONAL CONCEPTS

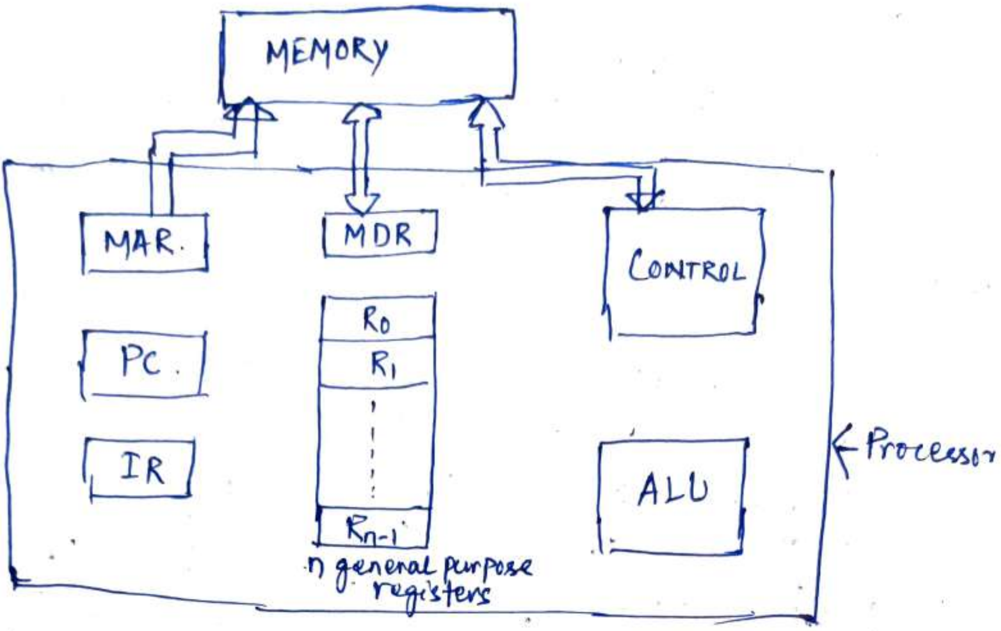
To perform a given task, an appropriate program consisting of a list of instruction needs to be executed. Instructions and data to be used as operand are stored in memory. A typical instruction may be

Add LOCA, R0

This instruction adds the content of memory location LOCA and register R0 and stores the result in R0. First the instruction is fetched from the memory into processor, Next the operand at LOCA is fetched and added to the other operand in R0. Finally sum is stored in register R0.

The Same instruction Can be written as below.

Load LOCA, R1
Add R1, R0



The transfers between the memory and the processor are started by sending the address of the memory location tobe accessed to the memory unit and issuing the appropriate control signals.

ALU and the control circuitry i.e the processor contains a number of registers used for several different purposes.

Instruction Register (IR) :- It holds the instruction that is currently being executed.

Program counter (PC) :- It contains the memory address of the next instruction tobe fetched and executed. It also known as location counter.

General purpose register used to hold temporary data tobe used for ALU operations.

Memory Address Register (MAR) :- The MAR holds the address of the location tobe accessed.

Memory Data Register (MDR) :- The MDR contains the data to be written into or read out of the addressed location.

The processor unit shown in fig 1.2 is usually implemented on a single VLSI chip, with at least one cache memory on the same chip.

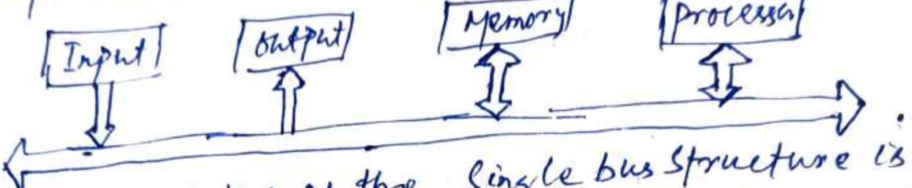
ategory - PC 17

BUS STRUCTURES.

To make the system operational the individual parts of a computer must be connected in some organized way.

A group of lines that serves as a connecting path for several devices is called a bus. There should be different buses for data, address and control signals. No. of lines in a data bus must equal to the size of a word to send a word of data simultaneouly.

The simplest way to interconnect functional units is to use a single bus.



The main virtue of the single bus structure is its low cost and it's flexibility for attaching peripheral devices. But only two units can actively use the bus at any given time. This makes system slower. To improve the performance system can contain multiple buses, which allows concurrency in operations. This leads to better performance but at an increased cost.

Some devices such as keyboards, printers are slow compared to others such as optical disks, memory & processor. To smooth out the timing differences buffer registers are used with the devices to hold the information during transfers. Say processor sends some data to printer. Now processor waits till its buffer register got the data, then printer prints without intervention of processor. This buffer register allows the processor to switch rapidly from one device to another, interweaving it's processing activity with data transfers involving several I/O devices.

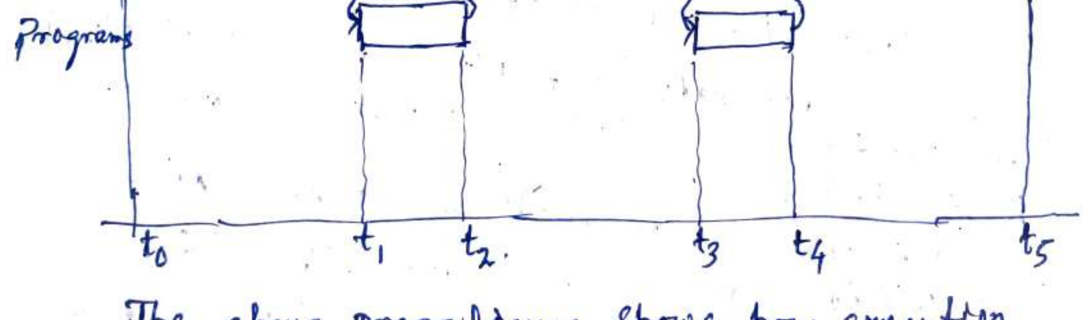
SOFTWARE

System software is a collection of programs that are executed to perform functions such as

- Receiving and interpreting user commands.
- Managing the storage nd retrieval of files in secondary storage devices.
- Controlling I/O units to receive input information nd produce output results.
- Running standard application programs such as word processors, spreadsheets, or games, with data supplied by the user.
- Linking nd running user-written application programs with existing standard library routines, such as numerical computation packages.

System software is thus responsible for the coordination of all activities in a computing system.

Some examples of system softwares are: Compiler, Text editor, operating systems etc. Good system softwares are required for better performance of system.



The above ~~progr~~ figure shows how execution control passes back and forth between the application program and the OS routines. Notice that bet^n t_0 to t_1 OS and printer are idle nd bet^n t_4 & t_5 OS & disk are idle. Computer resources can be used more efficiently if several application programs are to be processed. Concurrent execution of programs is called multiprogramming or multitasking. It is the job of operating System to look after concurrent execution of application Programs.

PERFORMANCE

The most important measure of the performance of a computer is how quickly it can execute program. For best performance, it is necessary to design compiler, the machine instruction set and the hardware in a coordinated way.

The total time required to execute the program is called elapsed time. It is a measure of the performance of the entire computer System. It is affected by the speed of the processor, the disk and the printer. But when we need to discuss the processor speed, we should only consider the processor time only. Some of the factors affecting the performance of processor are discussed below.

Processor Clock.

processor use a clock to generate timing signals of equal time called clock period. Each machine instruction is devided into no. of steps, where in each basic step some basic action is completed. Each basic step is completed in one clock cycle.

Let's $P \rightarrow$ length of one clock cycle s.

clock rate, $R = 1/P$ (no. of cycles/sec)

In Electrical Engineering cycles/s known as hertz (Hz).

Million is denoted by Mega (M).
Billion " " " Giga (G).

500 million cycles per Second abbreviated to 500 MHz.

$R = 500\,MHz$, $P = 1/R = \frac{1}{500 \times 10^{6}} = 2 \times 10^{-9}$ sec $= 2$ nanosec.

1250 million cycles per second = 1.25 GHz

$R = 1.25\,GHz$, $P = \frac{1}{1.25 \times 10^{9}} = 0.8\,ns$.

Basic Performance Equation.

$$T = \frac{N \times S}{R}$$

where $N \rightarrow$ No. of machine instructions to be executed for a program.

$S \rightarrow$ Avg. no. of basic steps per instruction.

$R \rightarrow$ Clock rate. (No. of cycles/sec).

To improve the performance parameter (T) we need to decrease N, S value and increase 'R'. N, S, & R are not independent parameters.

Pipelining and SuperScalar operation.

Overlapping of the execution of successive instruction known as pipelining. A substantial improvement in performance can be achieved by this pipelining technique.

Add R1, R2, R3. $R_1 + R_2 \rightarrow R_3$

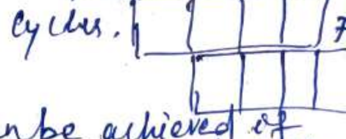
The contents of registers R1 & R2 are first transfered to the inputs of the ALU. After the add operation is performed the sum is transfered to R3.

During addition operation, processor can transfe the next instruction from memory. Now if this instruction also require ALU, then during the sum is transfered to R3, all the operands can be transferered to ALU.

In the ideal case, if all instructions are overlapped to the maximum degree possible, the effective value of 'S', approaches 1.

If we consider avg. 4 basic steps per instruction and fully pipelines then for 'N' instruction we need N+3 cycles.

$$S = \frac{N+3}{N} \simeq 1.$$



A higher degree of concurrency can be achieved if multiple instruction pipelines are implemented in the processor. This means that multiple functional units are used. to execute instructions parallaly. This mode of operation is called SuperScalar Execution.

Clock Rate.

There are twopossibilities for increasing the Clock rate, R.

→ Improving the IC technology, sothat it will reduces the time needed to Complete abasic step.

→ Reducing the amount of processing done in one basic step, which will reduce the clock period P. However if overall processing remains same, we ~~many~~ may need more basic steps per instruction.

Instruction SET: CISC & RISC.

CISC → Complex Instruction Set Computers.
RISC → Reduced Instruction Set Computers.

RISC, Simple instructions require a small no. of basic steps to exeute. For a processor that has only simple instructions, a large no. of instructions may be needed to perform a given programming task. This could lead to a large value for N and a small value for S.

CISC, Complex instructions involve a large number of steps. Fewer instructions are required to perform a given programming task. This lead to small value of N nd large value of S.

As far as pipelining is concerned, it is much easier to implement with simple instructions.

COMPILER

A Compiler translates a high-level language program into a sequence of machine instructions. To reduce N, we need to have a suitable machine instruction set and a compiler that makes good use of it. The compiler may rearrange program instructions to implement pipelining in a efficient manner. Of course, such changes must not affect the result of the computation. The ultimate aim is to reduce the total no. of clock cycles needed to perform a required programming task.

CH-2. MACHINE INSTRUCTIONS AND PROGRAMS

Binary Number System

Digits ÷ 0, 1

Decimal	Binary (4bit)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111.

If $B = b_{n-1} \dots b_1 b_0$

$V(B) = b_{n-1} \times 2^{n-1} + \dots + b_1 \times 2^1 + b_0 \times 2^0.$

$B = 1010.$

$V(B) = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
$= \underline{10}$

Using n no. of bits we can represent binary no.s from 0 to $\underline{2^n - 1}$.

To represent negative no.s we have 3 representations.

i) Sign & Magnitude.
ii) 1's Complement.
iii) 2's Complement.

Binary	Sign & Magnitude	1's Complement	2's Complement
	Corresponding decimal value		
0000	0	0	0
0001	1	1	1
0010	2	2	1
0011	3	3	2
0100	4	4	3
0101	5	5	4
0110	6	6	5
0111	7	7	6
1000	-0	-7	7
1001	-1	-6	-8
1010	-2	-5	-7
1011	-3	-4	-6
1100	-4	-3	-5
1101	-5	-2	-4
1110	-6	-1	-3
1111	-7	-0	-2
			-1

MSB used as sign bit

In these 3 representation negative no.s are represented with MSB as '1' nd Positive no.s with MSB as '0'.

In case of Sign & Magnitude it's straight forward. 0010 is 2 nd 1010 is -2.

In case of 1's Complement negative no.s are represented as Complement of it's Positive representati ie -5 as 1010 which is Complement of 0101.

1101 will be used as -(0010) ie $\underline{-2}$

In Case of 2's Complement negative no.s are represented as 1's Complement +1 of it's +ve representation, 2's Complement

-5 ⇒ it's positive representation is 0101
is Complement =

$$\begin{array}{r} 1010 \\ +\ \ \ 1 \\ \hline 1011 \end{array}$$

So.
-5 is represented using 1011.

To know a no. from it's binary representation

If MSB is '0' it's a positive no. & finding it's decimal equivalent is straight forward.
If MSB is '1' then it's a -ve number.

If nos are represented using Sign & Magnitude again it's straight forward.

1's Complement ⇒ In this case just complement the binary no.. You will find the +ve equivalent of the no. and put a -ve sign to get actual equivalent decimal number.

2's Complement ⇒ Converting a binary no. in 2's complement to actual decimal value is same as decimal to 2's complement, i.e 1'complement and add '1' to it.

* Show some examples to student.

Imp. why 2's Complement system is preffered?
As Sign & magnitude & 1's complement representation contains +0 & -0, 2'complement used in computer. Also 2'complement contains one extra no.. In case of 4 bit representation it contains -8, which is not their in other representation. Also the addition and subtraction process for 2's complement system is simple.

Range ⇒ In case 2's complement if we use 'n' bits to represent integer then it's range will be $-(2^{n-1})$ to $+(2^{n-1}-1)$

ADDITION AND SUBTRACTION.

Rules

1. To add two numbers, add their n-bit representations, ignoring the carry-out signal from the most significant bit (MSB) position. The sum will be the algebraically correct value in the 2's complement representation as long as the answer is in the range -2^{n-1} to $2^{n-1}-1$.

2. To subtract two numbers X and Y, that is to perform X-Y, form the 2's complement of Y and then add it to X, as in rule 1. Again, the result will be algebraically correct value in the 2's complement representation system if the answer is in the range.

Examples.

```
a)     0010  (+2)
     + 0011. (+3).
     ------  -----
       0101   +5.

b)      0100       (+4)   As the MSB
   -ve +(1)010     (-6)   is '1' no. is
   ------------    -----  -ve & in
   .ve. (1)110      -2    2's complement
                          form.

c)  1011   -  (-5)   If MSB is
   +1110      (-2)   1 means
   -----      ----   negative no.
  X 1001       -7    & representation
                     is 2's complement

d)  0111      +7
   +1101.     -3.
   -----      ---
  X0100       +4

e)  1101  (-3)       1101
   -1001. (-7)  =>  +0111.
   -----            ------
                   X0100  (+4)

f)  0010 (+2)       0010
   -0100 (+4)  =>  +1100.
   -----           ------
                    1110  (-2)

g)  0110 (+6)       0110
   -0011 (+3)  =>   1101
   -----           ------
                   X0011  (+3)
```

```
h)  1001 (-7)       1001
   -1011 (-5)  =>  +0101
   -----           -----
                    1110 (-2)

i)  1001 (-7)       1001
   -0001 (+1)  =>  +1111
   -----           ------
                   X1000 (-8)

j)  0010 (+2)       0010
   -1101 (-3)  =>   0011
   -----           -----
                    0101 (+5)
```

Sign Extension.

Normally there is a fixed size to represent an integer. Smaller +ve integer needs to be '0' added on the left.

representing '5' in 16 bit.

0000 0000 0000 0101

Similarly '1' is added on the left for smaller -ve numbers. This adding of '0' & '1' for +ve & -ve no.'s called Sign Extension.

-5 in 16 bit representation

1111 1111 1111 1011.

Overflow in Integer Arithmatic

When the result of an arithmetic ~~overflow~~ operation is outside the representable range, an arithmatic overflow has occured.

→ overflow can occur only when adding two numbers that have the same sign

(6-3) → The carry-out signal from the sign bit position is not a sufficient indicator of overflow when adding signed numbers.

→ When both operands x and y have the same sign, an overflow occurs when the sign of S is not the same as the signs of x and y.

(−8) → 1000   Sign of x & y = 1
+(−8)   1000.   Sign of S = 0
~~1~~0000   overflow occurs.
Result is not correct.

MEMORY LOCATIONS AND ADDRESSES.

BIG-ENDIAN & LITTLE-ENDIAN ASSIGNMENTS

There are two ways that Byte addresses Can be assigned across words.

Word Address	Byte Addresses			
0	0	1	2	3
4	4	5	6	7
		⋮		
2^k-4	2^k-4	2^k-3	2^k-2	2^k-1

Big-Endian Assignment

In Big-Endian Lower byte addresses are used for the more significant bytes of word.

Word Addresses	Byte Addresses			
0	3	2	1	0
4	7	6	5	4
		⋮		
2^k-4	2^k-1	2^k-2	2^k-3	2^k-4

Little-Endian Assignment

In Little-Endian Lower-byte addresses are used for the Less significant bytes of word.

Types of Instruction based on Operands in Memory

Three address instruction.

operation Such as $C \leftarrow A+B$ require a three address instruction to perform the operation in single instruction.

eg: Add A,B,C.

Operand A,B called Source operands and "C" is called destination operand.

Two-address instruction

An alternative approach to calculate $C \leftarrow A+B$ is

Add A,B $B \leftarrow A+B$

Move B,C. $C \leftarrow B$ (Copy Content of 'B' to 'C')

Both these instructions are two-address instruction. "A" ~~[illegible]~~ is ~~[illegible]~~ the Source and 'B' is the source and destination. In Move B,C 'B' is the Source and 'C' is the destination.

One-Address instruction

Three address instruction are very long to be accomodated in a Single word. ~~[illegible]~~ Even two words instruction does not fit into a Single word. Having ~~[illegible]~~ a Processor register accumulator leads to one ~~instruc~~ address instruction.

Load A ⇐ Loads "A" to Acc

Add B ⇐ Adds B to Acc & Store result in

Store C. ⇐ Stores result in 'C'.

Introduction of General purpose register changes the one address inst^{n} format. In this way the register holds one operand and it explicitly represented in instruction. where as one address instruction involves accumulator does not explicitly wrotes acc in the instruction

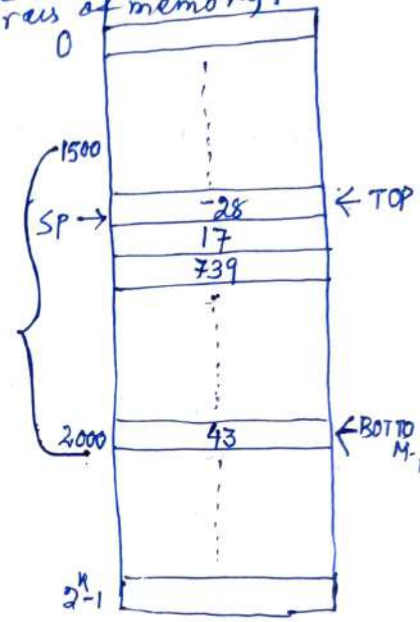
```
Move   A, Ri.
Add    B, Ri.
store  Ri, C.
```

Number notation. # - immediate value.
#% - " in binary
#$ - " Hexa.

STACKS AND QUEUES

A Stack is a list of data elements, usually words or bytes, with the accessing restriction that elements Canbe added or removed at one end of the list only. This end is Called the top of the Stack and the other end is Called the bottom. The Structure is Sometimes refered to as a pushdown Stack. It is a Common practice that Stack inserts new element in a decreasing address of memory.

A Processor register is used to Keep track of the address of the element of the Stack that is at the top at any given time. This register is Called the stack pointer (SP).



Routine for a Safe pop operation.

```
SAFEPOP   Compare    #2000, SP
          Branch>0   EMPTYERROR.
          Move       (SP)+, ITEM.
```

Routine for a Safe push operation.

```
SAFEPUSH   Compare    #1500, SP
           Branch≤0   FULLERROR
           Move       NEWITEM, -(SP).
```

Queue.

Another useful datastructure that is Similar to the Stack is Called queue. Data are Stored & retrieved on a FIFO basis. In case of queue, it is a Common practice that queue grows in the direction of increasing addresses in the memory. We need two pointers to hold the addresses of first and last data in Queue. Both ends of a queue move to higher addresses as data are added at the back and removed from the front. Without any boundary a queue would Continuously move through the memory of a Computer in the direction of higher addresses. One way to limit the queue to a fixed region in memory is to use a Circular buffer.

SUBROUTINES

Subroutines are Subtask of a program, whichis often perform manytimes on different data values. When Program branches to a Subroutine, we say that it is Calling the Subroutine. After execution of the ~~progra~~ Subproutine the return ~~state~~ instruction is executed and Control returns to the program.

The way in which a Computer makes it possible to call and return from Subroutine is referred to as its Subroutine linkage method.
A dedicated register Called link register Used to Store the return address at the time of Call and ~~use~~ this value is used at the time of return.

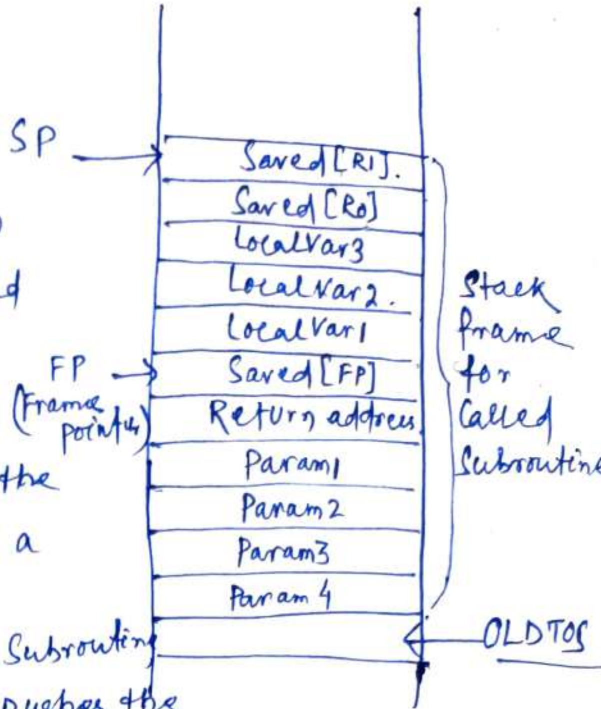
The Call instruction is just a Special branch instruction that performs the following operations.
→ Store the Contents of the PC in the Link register.
→ Branch to the target address Specified by the inst^n.
The Return instruction is a Special branch inst^n that performs the operation:
→ Branch to the address Contained in the link register.

SUBROUTINE NESTING.

A Subroutine Can Call another Subroutine, whichis Called Subroutine nesting. If a single link register is used to store the return address then then in case of subroutine nesting the previous content of link register will be lost, while calling Subroutine from another Subroutine Hence it is essential to Save the Contents of the link register in Some other location before calling another Subroutine. On this Scenario Stack is used to store the information while Calling a Subroutines, which LIFO properties help us to return from the recent Subroutine Call properly.

THE STACK FRAME

The locations in a Stack used by a Subroutine from subroutine Call to Subroutine return is called the stack frame of that subroutine.



A Stack frame holds the below information for a Subroutine.

→ Before Calling the Subroutine the Calling program pushes the Parameters on to Stack.

→ Then the return address is pushed on to Stack when the Call instruction is executed.

→ Then Previous Frame pointer value is pushed onto stack.

→ Then all the local variables are pushed on to stack.

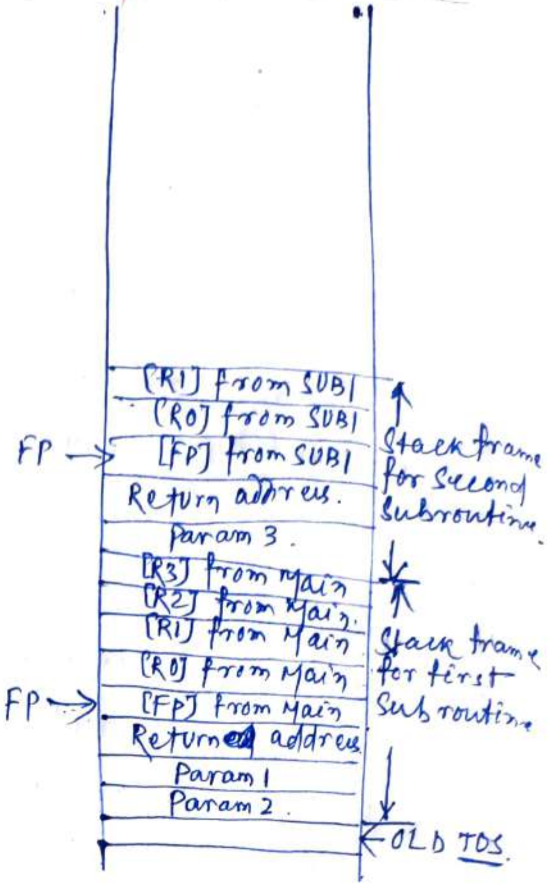
→ Then the Content of registers which are going to be used by the Subroutines are Saved..

Frame pointer (FP) is a general purpose register. Frame pointer Points to the location just above the Stored return address, helps to access the parameters and the local variables by using the Index addressing mode. (8(FP), 12(FP), -8(FP)

The Content of FP is fixed through out the executi of Subroutime unlike Sp which Points to the Top of Stack.

When the task is Completed, the Subroutime Pops the Saved values of ~~R1 and R0~~ registers back into those registers and removes the local variables by setting SP to point previous FP value The previous FP value is poped and SP Point to return address. After returning to the Calli program, the Calling Program removes the param and SP Points to old TOS.

Stack frames for nested Subroutines.



ADDITIONAL INSTRUCTION

LOGIC INSTRUCTIONS.

Logic operations such as AND, OR and NOT applied to individual bits.

NOT dst – It Complements all the bits in destination.

NOT R0 }
Add #1, R0 } – find's 2's complement of Content of R0.

Some Computers have a Single inst^n "NEGATE R0". to do above operation.

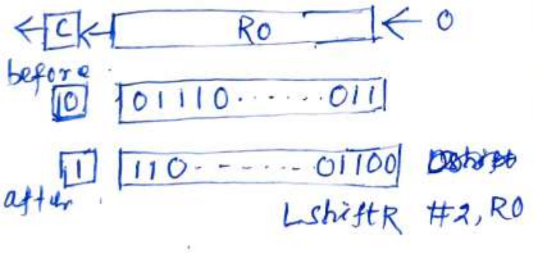
```
AND      #$FF000000, R0  ← Makes all bits to zero except first 8bit (MSB)
COMPARE  #$5A000000, R0  ← Checks whether first character is equal to 'Z' or not. Hexadecimal of 'Z' is 5A(01011010)
BRANCH=0  YES.
```

If there is match, branch inst^n branches to YES.

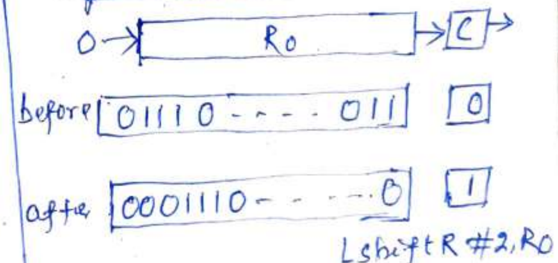
SHIFT AND ROTATE INSTRUCTIONS.

Logical Shift. → Specified no. of bits shifts left or right and zero(0) is filled for empty bits. The Carry flag (C) holds the last bit shifted out from the number.

Logical Shift Left.



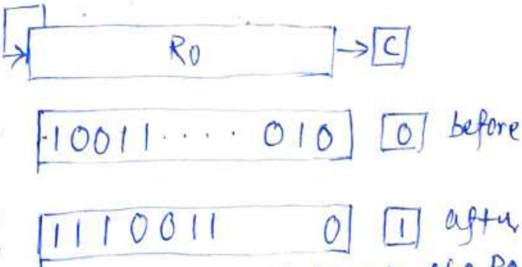
Logical Shift Right.



Arithmatic Shift Left is Same as logical Shift left.

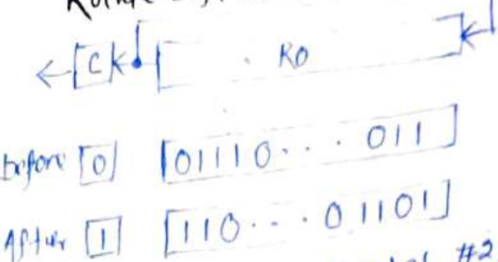
In Case of Arithmatic Shift right, the fill in bit is the sign bit that may be 1 or 0. in our example it is 1.

Arithmatic Shift Right.



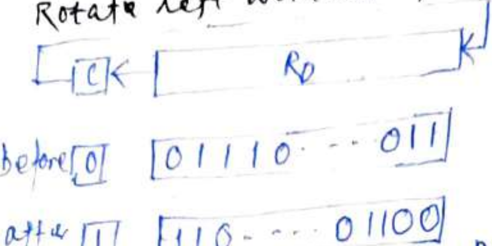
ROTATE OPERATION

Rotate left without carry



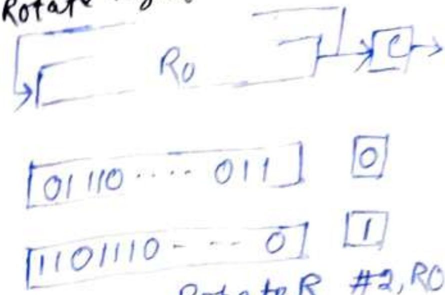
RotateL #2,R0

Rotate left with Carry.



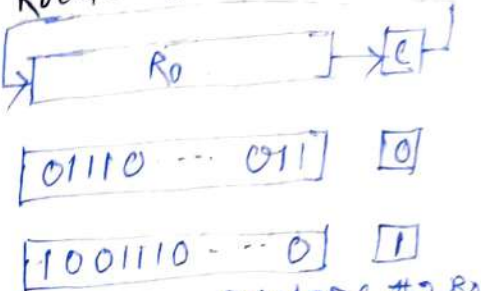
RotateLC #2,R0

Rotate right without Carry.



RotateR #2,R0

Rotate right with Carry.



RotateRC #2,R0

Rotate without Carry does not use Carry, but Carry holds ~~that~~ the last bit shifted out.

MULTIPLICATION AND DIVISION

Multiply Ri, Rj

$R_j \leftarrow [R_i] \times [R_j]$

Result of two ~~[illegible]~~ nbit multiplication is 2nbits. So many Computers use two adjacent registers to store the result. (R_j, R_{j+1})

Divide Ri, Rj

$R_j \leftarrow [R_j]/[R_i]$.

Some Computers use Rj to store quotient and R_{j+1} to store the remainder, Else remainder is Lost.

Short Question.

1) What do you mean by Stored Program Computer ?

2) Write the full form of ASCII, EBCDIC, VLSI.

3) what is four layers of memory hierarchy?

4) what do you mean by interrupt-service routine and when it is called for execution,

5) what do you mean by BUS nd list out advantage nd disvantage of Single Bus Structure over multiple Bus Structure.

6) Write the basic Performance equation.

7) What are the clock period of 1 GHz nd 750 MHz processors.

8) what do you mean by pipelining.

9) Differentiate CISC & RISC.

10) Explain SPEC rating nd it's importance in measuring performance of a computer.

11) find out 2's Complement representation of below no.s Considering 8 bit to represent the integer.
87, 37, -88, -23, -53.

12) Perform the below operations using 8 bit representation of integers.
i) 18 + 26 ii) 75 + 53 iii) 77 - 47 iv) -65-64

13) what do you mean by arithmatic overflow, How you can know there is a arithmatic Overflow?

14) what is byte addresability!

15) Differentiate Big-Endian and Little-Endian

16) what do you mean by WORD as far as memory locations are concerned?

17) what do you mean by straight line Sequencing of instructions?

18) Explain Different phases of instruction execution.

19) what do you mean by Status register?

20) With example, explain assembler directive.

21) what is the job of loader and debugger?

22) why it is mandatory to use stack for Subroutine Nesting.

23) What do you mean by frame pointer?

long Questions

1) Describe the Basic functional Units of a Computer with a neat diagram.

2). List out the jobs of Control Unit.

3) what are the jobs of below functional units of a Computer ? iii) MAR iv) MDR
i) PC ii) IR

4) What are the jobs of System Software? Give Some examples of System Software.

5) Discuss different aspect of a computer, which affects the performance of a computer in terms of speed of execution.

6) List the steps needed to execute the machine instruction Add LOCA, R0 and also for instruction Add R1, R2, R3.

7) With example explain three-address, two-address, one-address and zero-address instructions.

8) Discuss all the Addressing modes with example.

9) ~~Draw the~~ Explain Basic Input/output operations with a neat diagram.

10) How a Stackframe is Created with in Stack for a Subroutine Call and how the data are removed ~~for~~ when Subroutine finishes it's execution.

11) Draw a Stack frame for Subroutine nesting.

12) Explain all Shift and Rotate instructions with example.

CH-6
Mod-II

ARITHMETIC

ADDITION/SUBTRACTION LOGIC UNIT.

Full Adder

$\oplus$ - XOR.
$\odot$ - XNOR.

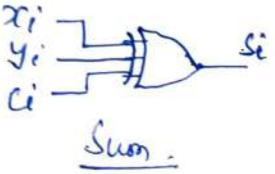
x_i	y_i	c_i (carry-in)	(S_i) Sum	Carry-out C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S_i = \bar{x}_i\bar{y}_i c_i + \bar{x}_i y_i \bar{c}_i + x_i\bar{y}_i\bar{c}_i + x_i y_i c_i$$
$$= \bar{x}_i(y_i \oplus c_i) + x_i(y_i \odot c_i)$$
$$= \underline{x_i \oplus y_i \oplus c_i}$$

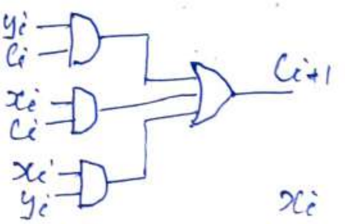
$$C_{i+1} = \underline{y_i c_i + x_i c_i} \; \underline{+ x_i y_i}$$

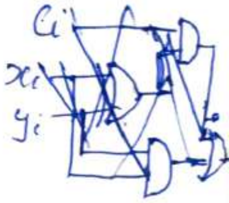
Using K-map.

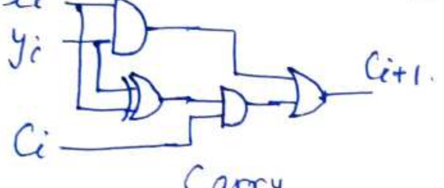
$$C_{i+1} = \bar{x}_i y_i c_i + x_i \bar{y}_i c_i + x_i y_i \bar{c}_i + x_i y_i c_i$$
$$= c_i(x_i \oplus y_i) + x_i y_i .$$



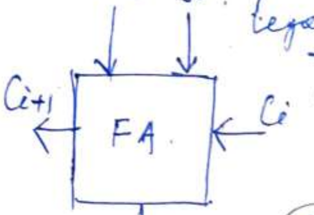
Sum.







Carry.



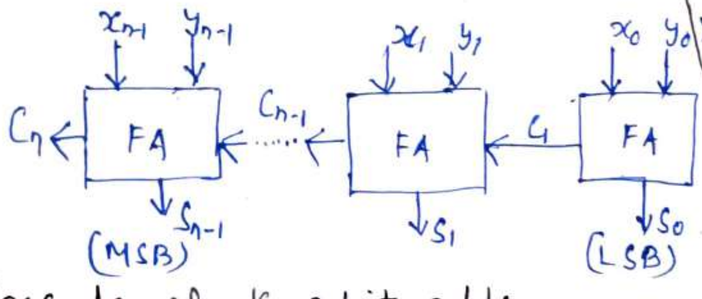
legend for stages

x_i \ y_ic_i	00	01	11	10
0	$_0$	$_1$	$_3$ 1	$_2$
1	$_4$	$_5$ 1	$_7$ 1	$_6$ 1

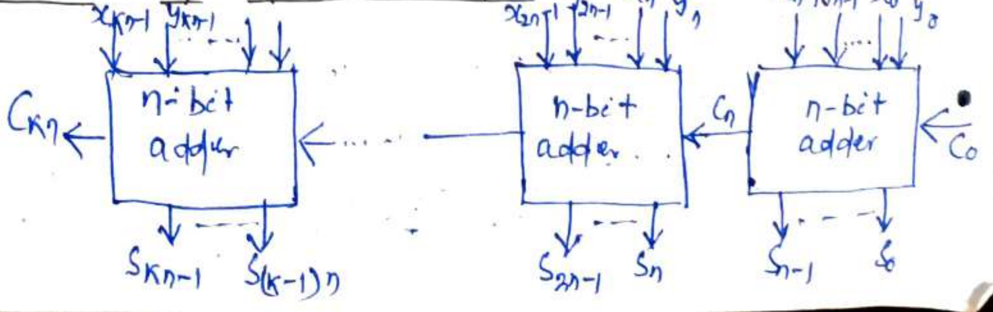
y_i (columns 11, 10); c_i (columns 01, 11)

$$= x_i c_i \rightarrow \boxed{5,7}$$
$$+ y_i c_i \rightarrow \boxed{3,7}$$
$$+ x_i y_i \rightarrow \boxed{7,6}$$

n-bit ripple-carry adder.



Cascade of K n-bit adders



Arithmatic overflow

Method 1→ Overflow Can occure when Sign of the two operand is Same and Sign of the result is different than of operands.

$$\text{Overflow} = x_{n-1} y_{n-1} \bar{S}_{n-1} + \bar{x}_{n-1} \bar{y}_{n-1} S_{n-1}.$$

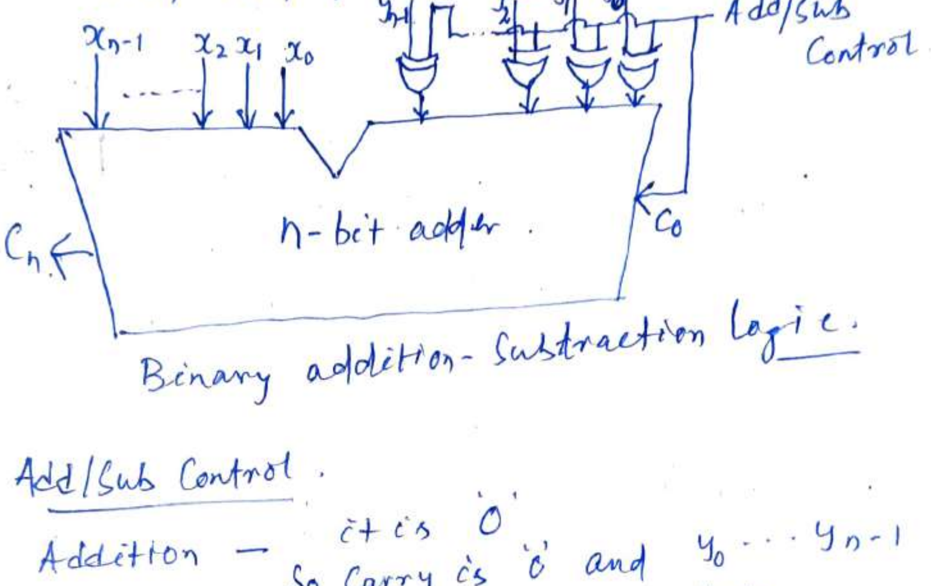
Method 2→

~~Pass~~ When the Carry bits C_n and C_{n-1} are different, overflow occurs.

$$\text{overflow} = C_n \oplus C_{n-1}.$$

Doing Addition & Subtraction in a Single logic unit :-

With heep of 2's Complement we can do that



Binary addition-Subtraction logic.

Add/Sub Control.

Addition — it is '0' So Carry is '0' and $y_0 \cdots y_{n-1}$ will be Supplied as it is.

	y_i	Add/Sub	$y_i \oplus$ Add/Sub
add	0	0	0
	1	0	1
Sub	0	1	1
	1	1	0

Subtraction — it is 1.
So $y_0 \cdots y_{n-1}$ is Complemented due to XOR.

C_0 is 1, So the Complemented value is added with '1' making it 2's Complement.

And we know that $X - Y$ is equivalent to $X + (\text{2's Complement of } Y)$

$$= X + ((\text{1's complement of } y) + 1).$$

DESIGN OF FAST ADDERS.

If an n-bit ripple-Carry adder is used in the addition/Subtraction, it may have too much delay in developing it's outputs, S_0 through S_{n-1} and C_n. Each FA gives Sum and Carry in 2 gate delay So for getting C_n, we need 2n gate delays. If we consider the XOR gates on the y input and the $C_n \oplus C_{n-1}$ for overflow, the total delay is 2n+2 gate delay for the whole n-bit ripple-carry adder.

CARRY-LOOKAHEAD ADDITION.

$$S_i = x_i \oplus y_i \oplus c_i$$

and $C_{i+1} = x_i y_i + x_i C_i + y_i C_i$

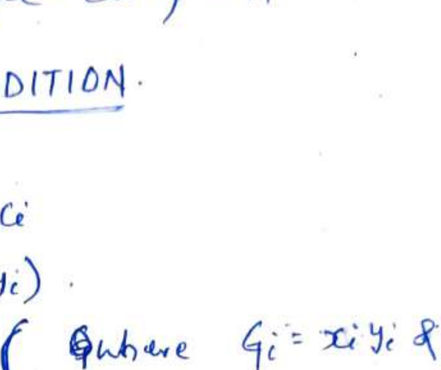
$$= x_i y_i + C_i(x_i + y_i)$$

$$= G_i + P_i C_i \quad (\text{where } G_i = x_i y_i \;\&\; P_i = x_i + y_i)$$

G_i and P_i are Called generate and propagate function for Stage 'i'.

$C_{i+1} = 1$, if 'G_i' the generate function $=1$, ie x_i & y_i both are 1, independent of C_i.

In this implementation P_i is $x_i \oplus y_i$, which differs from actual P_i, when x_i & y_i both 1. But in this Case G_i is 1, So it does not matter whether P_i is 1 or zero.



$$C_{i+1} = G_i + P_i C_i$$

$$= G_i + P_i(G_{i-1} + P_{i-1}C_{i-1})$$

$$= G_i + P_i G_{i-1} + P_i P_{i-1} C_{i-1}.$$

So Continuing this type of expression for any Carry variable is

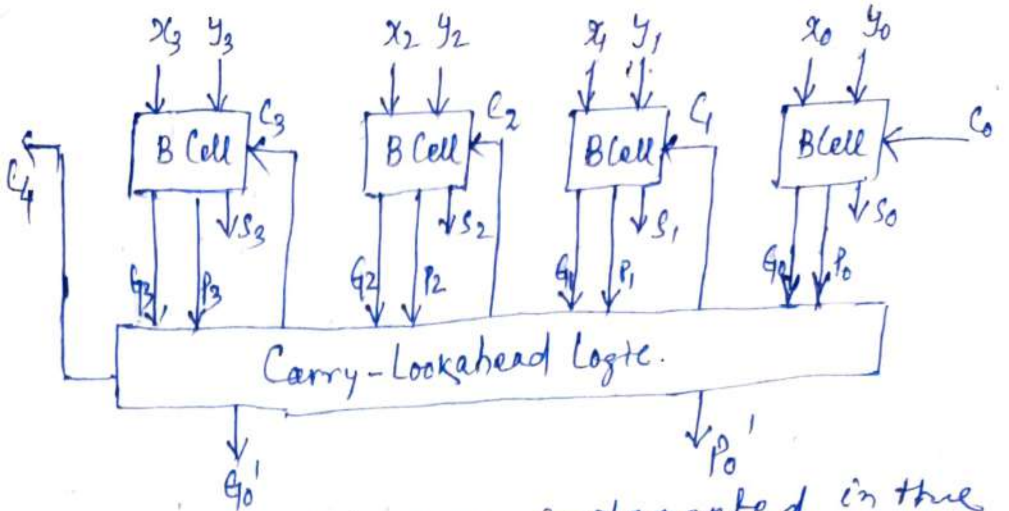
$$C_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + \cdots + P_i P_{i-1} \cdots P_1 G_0 + P_i P_{i-1} \cdots P_0 C_0.$$

for a 4-bit adder

$C_1 = G_0 + P_0 C_0$, $C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0$

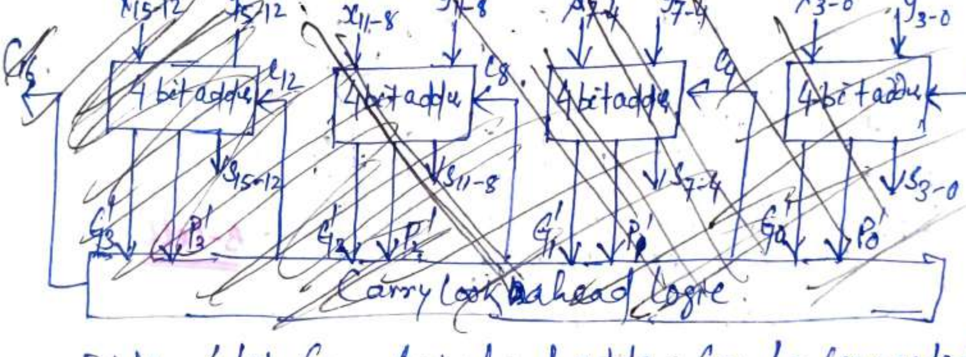
$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

$C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0.$



The carries are implemented in the block labeled carry-lookahead logic. Delay through the adder is 3 gate delays for all Carry bits and 4 gate delays for all Sumbits. In comparison, a 4bit ripple-carry adder requires 7 gate delays for S_3 and 8 gate delays for C_4.

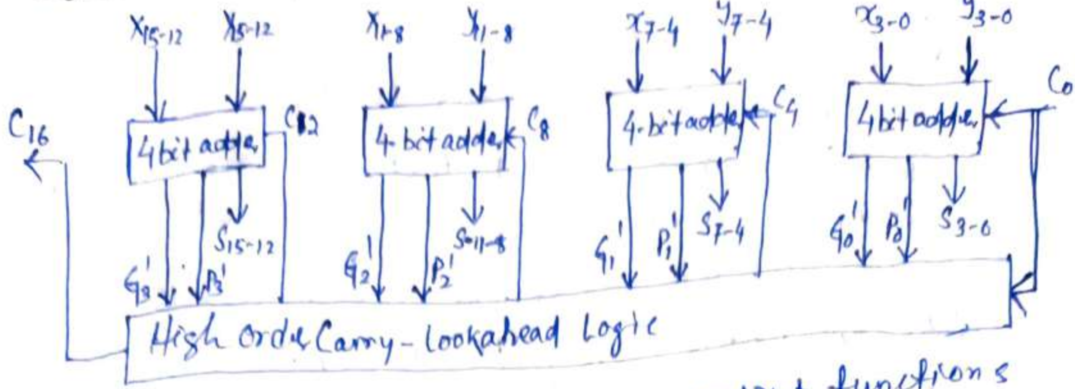
If we try to extend the carry-lookahead adder ~~as~~ for longer operands, we run into a problem of gate fan-in constraints. For C_4 in the 4-bit adder, a fan-in of 5 is required. This is abt the limit for practical gates.



Eight 4-bit Carry Lookahead adders can be connected ~~as in figure above~~ to form a 32 bit adder. The delays in getting C_4 from the low order adder is available in 3 gate delay, Then, C_8 is available after a further 2 gate delay, C_{12} is available after a further 2 gate delays and so on.

Finally C_{28} available after $6 \times 2 + 3 = 15$ gate delay and C_{32} available after 17 gate delay. All the Sums available after 18 gate delay.

Higher-Level Generate and Propagate functions



Each 4bit adder provide new output functions defined as G_K' and P_K', where $K=0$ for the first 4-bit block, $K=1$ for the Second 4-bit block and so on.

$$P_0' = P_3P_2P_1P_0 \text{ and } G_0' = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$$

With these new functions available it is not necessary to wait for Carries to ripple through the 4-bit blocks.

$$C_{16} = G_3' + P_3'G_2' + P_3'P_2'G_1' + P_3'P_2'P_1'G_0' + P_3'P_2'P_1'P_0'C_0.$$

We should note that in this design we may ignore C_4, C_8, C_{12} and C_{16} generated internally by the 4-bit adder blocks as they are generated by the higher level Carry-lookahead Circuits.

Delay. G_K' and P_K' functions require two and one gate delay respectively, after the generation of G_i and P_i. ~~Therefore, all carries produced by the carry-lookahead circuits are available 5 gate delays after X, y and C_0 applied as inputs~~

G_i and P_i takes 1 gate delay. ~~so carries C_4, C_8, C_{12}, C_{16} needs 5 gate delay after x, y and C_0 applied as inputs~~. then to generate Carry C_4, C_8, C_{12}, C_{16} we need another two gate delay and another one gate delay for sum. So over all 16 bit adder need 8 gate delay to produce the sum. Carry (C_{16}) can be generated in 7 gate delay. But if four 4-bit adder cascaded together for 16 bit addition it requires 9 and 10 gate delays to generate Carry (C_{16}) and Sum (S_{15}).

Delay.

Within 4bit adder

G_i & P_i takes 1 Gate delay.

G_k' & P_k' takes 2 & 1 Gate delay.

So overall after 3 gate delay we get G_k' & P_k'

∴ So from G_k' & P_k' Carries C_4, C_8, C_{12}, C_{16}. Produced after 5 gate delay.

S_{3-0} generated after 4 gate delays.

~~But S_{7-4}, S_{11-8}, S_{15-12} generated after C_4, C_8 & C_{12} is generated.~~

~~So S_{16} is generated after~~

C_{1-3} generated after 3 gate delays.

C_4 generated within 4bit adder is ignored.

But Carry generated within next 4 bit adders such as C_{5-7}, C_{9-11} & C_{13-15} requires C_4, C_8, C_{12} as Carry to the 4bit adder.

Same for S_{15-4}.

All these Carries need another 2 gate delays. After that another 1 gate delay for Sum.

So S_{15-4} needs $5+2+1=8$ gate delays.

C_{16} needs 5 gate delay

and S_{15} " 8 " ".

where as 16 bit adder implemented by Cascading 4 bit Carry-lookahead adder block requires 9 and 10 gate delays for developing C_{16} and S_{15}.

32 bit adder.

Two 16 bit adder blocks can be implemented a 32-bit adder.

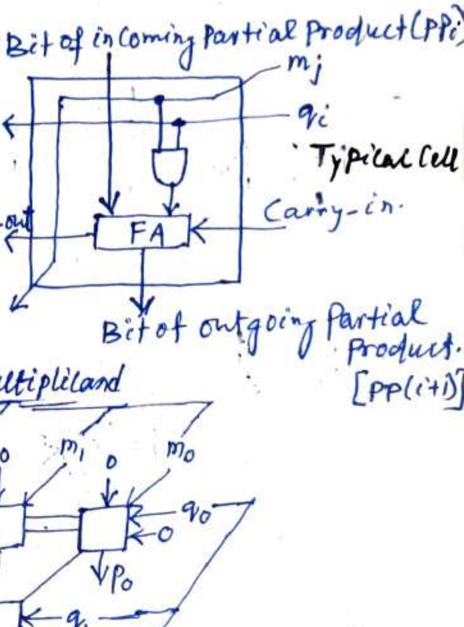
2nd 16 bit adder gets the carry C_{16} after 5 gate delays. Within that Period G_k' & P_k' of 2nd 16 bit adder was ready. So after another 2 gate delay $C_{20}, C_{24}, C_{28}, C_{32}$ is out. in the high-order block. Total 7 gate delay. After getting C_{20}, C_{24}, C_{28} all other Carry $C_{21-23}, C_{25-27}, C_{29-31}$ will require another 2 gate delay. So C_{31} is out after 9 gate delay. and S_{31} is available after 10 gate delay.

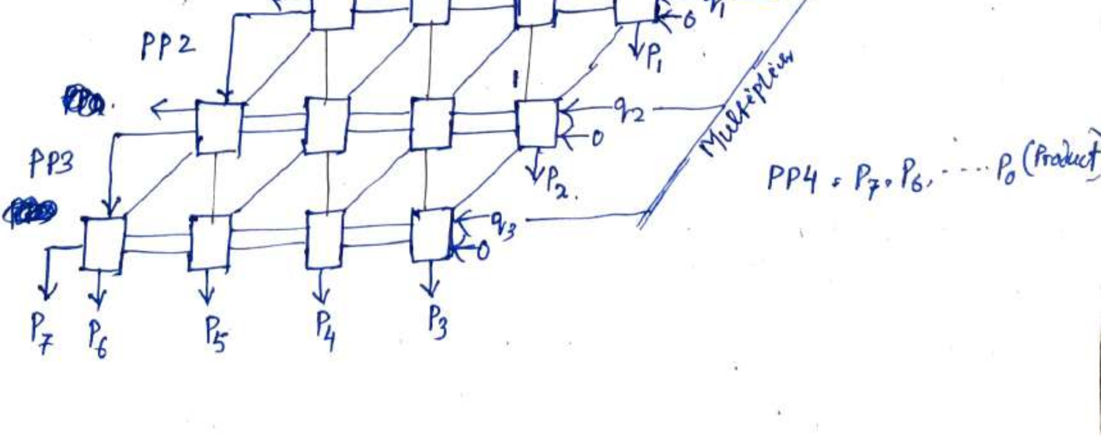
Overall C_{32} & S_{31} are available after 7 & 10 gate delay.

where as C_{32} & S_{31} in case of cascade of eight 4-bit adder need 17 & 18 gate delay.

MULTIPLICATION OF POSITIVE NUMBERS.

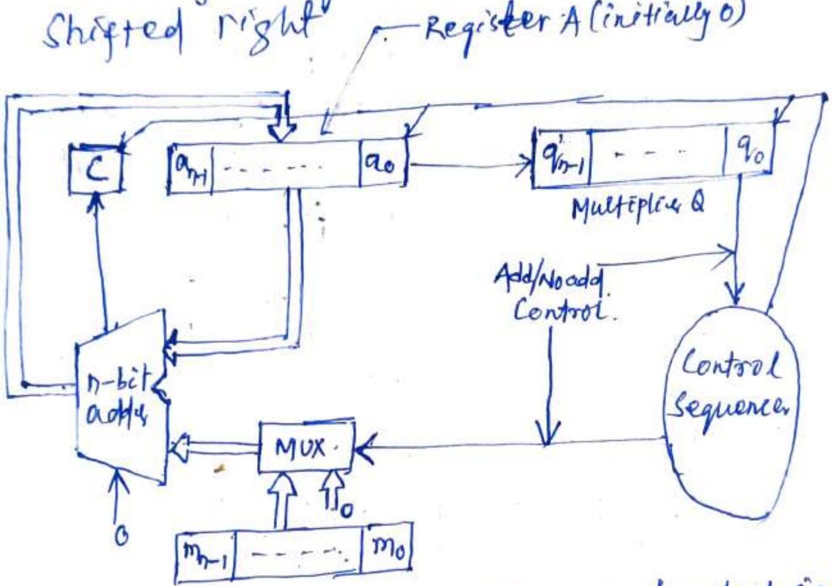
```
      1101   (13) Multiplicand M
      1011   (11) Multiplier Q.
      ----
      1101
     1101
    0000
   1101
  --------
  10001111   (143) Product P
```





PP4 = P7, P6, ... P0 (Product)

The Simplest way to perform multiplication is to use the adder circuitry in the ALU for a no. of Sequential Steps. Adder will add two binary no.s and the partial result is stored in a register. Register 'A' is used to store the partial products. Register 'Q' is used to store the Multiplier. Instead of Shifting the Multiplicand to the left as done by hand, here the Combination of Carry, A register and Q register Content is Shifted right



At the Start, the multiplier is loaded into register Q, the multiplicand into register M, and 'C' and 'A' are cleared to 0. Register A and Q Combined hold PPi while multiplier bit qi generates the signal Add/Noadd. This Signal Controls the addition of the Multiplicand, M, to PPi to generate PP(i+1). If $q_i = 0$, then MUX gives all 0s and if $q_i = 1$, then MUX gives Multiplicand 'M' to the adder. The Product is Computed in n cycles. The partial Product grows in length by one bit per cycle from the initial vector, PPO of n 0s in register 'A'. The carry out from the adder is stored in flipflop 'C'.

At the end of each cycle, C, A, and Q are shifted right one bit position to allow for growth of the partial product as the multiplier is shifted out of register Q. Because of this shifting LSB of register Q contains q_i for Add/Noadd signal. After n-cycles, the high-order half of the product is held in register A and the low-order half is in register Q.

M

1101

C	A	Q		
0	0000	1011		
0	1101	1011	— Add	} first Cycle
0	0110	1101	— Shift	
1	0011	1101	— Add	} 2nd Cycle.
0	1001	1110	— Shift	
0	1001	1110	— No add	} 3rd Cycle
0	0100	1111	— Shift	
1	0001	1111	— Add	} 4th Cycle
0	1000	1111	— Shift	

1000 1111 — Result.

SIGNED-OPERAND MULTIPLICATION.

[2's Complement Signed operand]

Case-1. -ve multiplicand +ve multiplier.

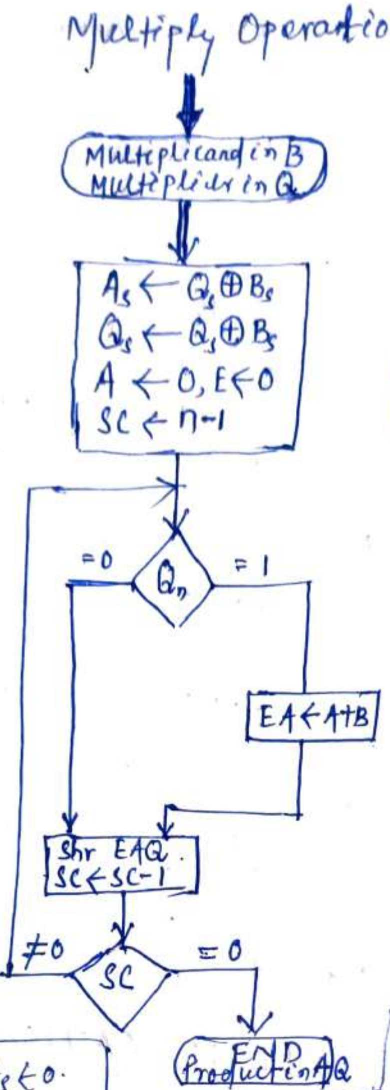
When we add a -ve multiplicand to a partial product, we must extend the sign bit value of the multiplicand to the left as far as the product will extend.

```
          10011   (-13)
          01011   (+11)
     ----------
     1111110011
     111110011
     00000000
     1110011
     000000
     ----------
     1101110001 (-143)
```

This method does not work for +ve multiplicand and -ve multiplier.

The hardware discussed earlier can be used for negative multiplicands if it provides for sign extensions of the partial products.

Case-II. for negative multiplier, a straightforward solution is to form the 2's complement of both the multiplier and the multiplicand and proceed as in the case of a positive multiplier. (Remember originally the no.s are in 2's comflowchart for Signed/Unsigned multiplication for Sign and magnitude ~~n~~ number format.



Q_s - Sign of multiplier
B_s - Sign of multiplicand

n → no. of bits.
Process will run for (n-1) times as one bit for Sign bit.

Register A, B, Q are of size (n-1) and B, Q holds multiplicand and multiplier without Signbit.

0110 (+6)
0101.

$A_s \leftarrow 0$.
$Q_s \leftarrow 0$

B.
0110

E	A	Q	
	0000	0101	
	0110	0101	- Add.
0	0011	0010	- Shift.
0	0011	0010	- No add.
0	0001	1001	- Shift.
0	0111	1001	- Add.
0	0011	1100	- Shift.

(32+16+8+4)

including sign bit is not possible

~~(-6) × (-5)~~
~~2's. 1010 1011~~

~~1010~~
~~1011.~~
~~1010~~
~~1010~~
~~0000~~
~~1010~~
~~1101110~~
~~0010010~~

~~0101 +1 0110~~

Imp

Booth Multiplication Algorithm

Booth algorithm gives a procedure for multiplying binary integers in signed-2's complement representation. In this method the multiplier is recoded and according to that multiplicand is selected for each stage.

Booth Multiplier Recoding table

Multiplier		Recoded Multiplier Bit i	Version of multiplicand Selected by bit i
Bit i-1	Bit i		
0	0	0	0 X M
0	1	-1	-1 X M ← In this case 2's complement of multiplicand is added
1	0	+1	+1 X M.
1	1	0	0 X M

eg:-

0 0 1 0 1 1 0 0 1 1 1 0 1 0 1 1 0 0

0 +1 -1 +1 0 -1 0 +1 0 0 -1 +1 -1 +1 0 -1 0 0

for the rightmost bit i.e LSB, assume it's previous bit is 0.

Worst Case Multiplier. 0 1 0 1 0 1 0 1 0 1 0 1

⇩

+1 -1 +1 -1 +1 -1 +1 -1 +1 -1 +1 -1

Ordinary Multiplier. 1 1 0 0 0 1 0 1 1 0 1 1 1 0 0

⇩

0 -1 0 0 +1 -1 +1 0 -1 +1 0 0 -1 0 0

Good Multiplier. 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1

0 0 0 +1 0 0 0 0 -1 0 0 0 +1 0 0 -1

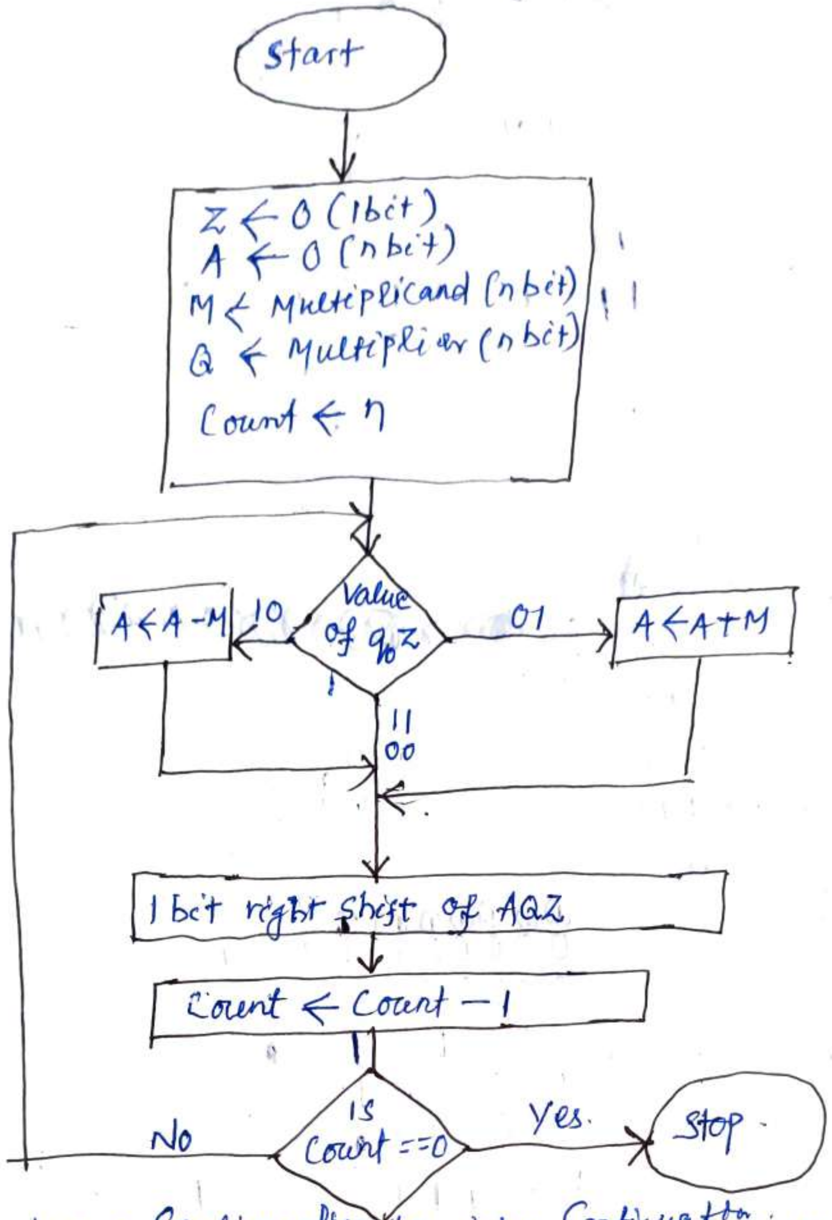
Booth algorithm has two attractive features

1- It handles both positive and negative uniformly

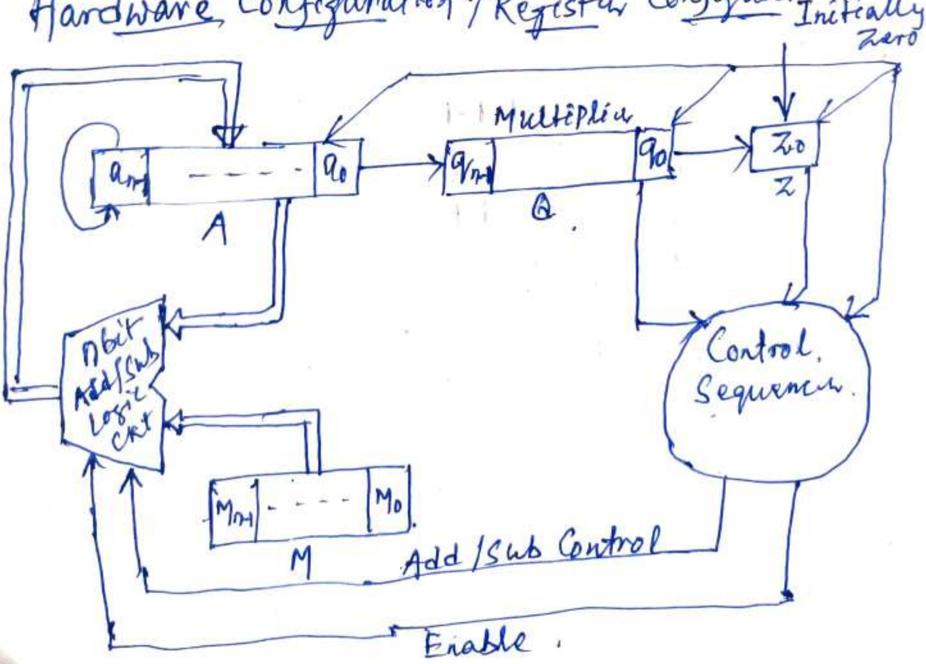
2- It achieves some efficiency in the number of additions required when the multiplier has a few large blocks of 1s.

On avg, the speed of doing multiplication with the booth algorithm is the same as with the normal algorithm.

Flow chart for Booth Algo Multiplication



Hardware Configuration / Register Configuration



FAST MULTIPLICATION

BIT-PAIR RECODING OF MULTIPLIERS

Bit-pair recoding halves the maximum number of summands. It is derived from booth algorithm.

The Pair (+1, −1) is equivalent to (0, +1).

Adding $\bar{1}$ × M at position i to +1 × M at Position i+1 is equivalent to +1 × M at position 'i'.

Simillarly (+1, 0) equivalent to (0, +2) and.
(−1, +1) " to (0, −1) and so on.

Sign extension
$\boxed{1}$ 1 1 0 1 0 $\boxed{0}$ ← Implied 0 to right of LSB

$$\Downarrow$$

$$\underbrace{0\quad 0}_{0}\quad \underbrace{-1\quad +1}_{-1}\quad \underbrace{-1\quad 0}_{-2}$$

example of bit pair recoding derived from booth recoding.

Table of Multiplicand Selection decisions for bit-pair recoding

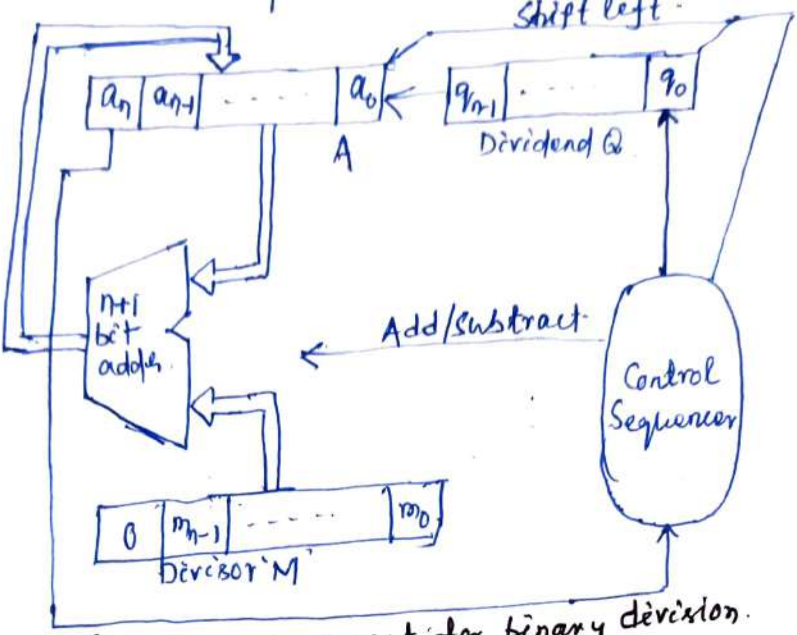
Multiplier bit-pair		Multiplier bit on the right	Multiplicand Selected at position i		
i+1	i	i−1			0
0	0	0	0 × M	— 0	
0	0	1	+1 × M.	— 1	+1
0	1	0	+1 × M.	— 2	
0	1	1	+2 × M	— 3	+2
1	0	0	−2 × M	— 4	−2
1	0	1	−1 × M	— 5	−1
1	1	0	−1 × M.	— 6	
1	1	1	0 × M.	— 7	0

```
  +13      01101              0 1 1 0 1
 × −6      11010              0   −1  −2.
                      1 1 1 1 1 0 0 1 1 0
                      1 1 1 1 0 0 1 1
                      0 0 0 0 0 0 0
 10011                1 1 1 0 1 1 0 0 1 0  (−78)
```

.TEGER DIVISION

→ Restoring Division
→ Non Restoring Division.

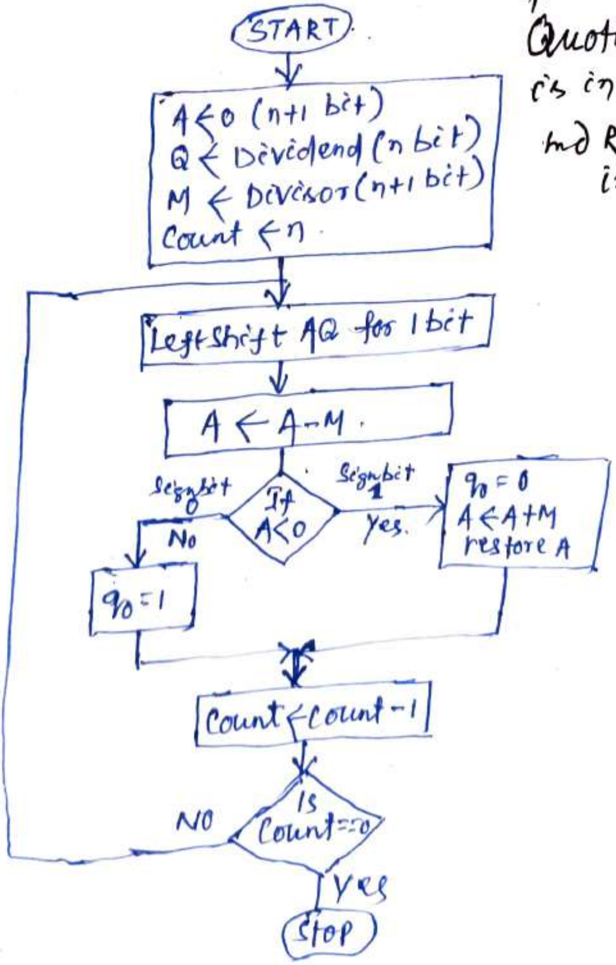
```
          10101 ← Quotient
  1101)100010010 ← Dividend
    ↑   1101
Divisor 10000
         1101
          1110
          1101
             1 ← Remainder
```



Circuit arrangement for binary division.

Restoring Division

At the end Quotient (n bit) is in register Q and Remainder in Register A



Restoring Division.

1. Shift A and Q left one binary Position (don't fill q_0)
2. Subtract M from A, and place the answer back in A.
3. If the Sign of A is 1, Set q_0 to 0 and add M back to A (that is, restore A); otherwise, Set q_0 to 1.

Do the above n-times.

```
      10
11 ) 1000
     11
     ----
      10
```

example.

```
Initially            A 00000    Q 1000
                     M 00011

Shift                  00001      000□
Subtract               11101
(Add 2's Comp)        -------
                      (1)1110
Set q0 = 0.                11
Restore 'A'           -------
                       00001      000[0]

Shift                  00010      00[0]□
Subtract               11101
                      -------
                      (1)1111
Set q0 = 0                 11
Restore 'A'           -------
                       00010      00[0][0]

Shift                  00100      0[0][0]□
Subtract               11101
                      -------
                      (0)0001
Set q0 = 1             00001      0[0][0][1]

Shift                  00010      [0][0][1]□
Subtract               11101
                      -------
                       11111
Set q0 = 0                 11
Restore 'A'            00010      0010
```

An [n]-bit positive divisor is loaded into register M and an [illegible]-bit positive dividend is loaded into register Q. Register A is set to 0. After the division is complete, the n-bit quotient is in register Q and the remainder is in register A. The required Subtraction is done by 2's Complement Arethmati

Nonrestoring-division:- Restore operations are no longer needed. No need to add the divisor to restore 'A'. In this method in each step either we add or subtract the divisor depending on the sign of 'A'

Do the following 'n' times.

If sign bit of 'A' is '0'

1. Shift A & Q left one bit position and subtract M from 'A' and set q_0 opposite of sign bit of 'A' after subtract.

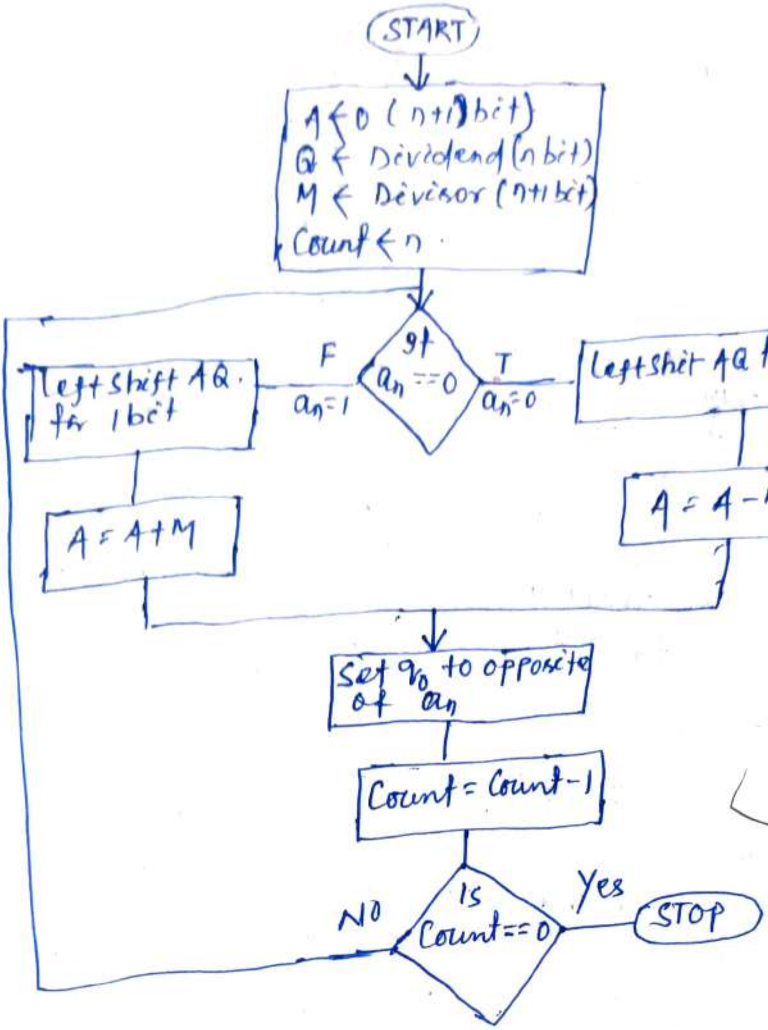
2. If sign bit of 'A' is 1, shift A&Q left one bit and add M to A and set q_0 opposite of sign bit of 'A' after addition.

3. At the end of 'n' cycles if sign of 'A' is 1 add 'M' to A to leave the proper positive remainder in 'A'.

Initially	00000 00011	1000	
Shift	00001	000□	First Cycle.
Subtract	11101		
Set q_0	11110	0000	
Shift	11100	000□	Second Cycle.
Add	00011		
Set q_0	11111	0000	
Shift	11110	000□	Third cycle.
Add	00011		
Set q_0	00001	0001	
Shift	00010	001□	fourth cycle.
Subtract	11101		
	11111	0010 Quotient.	

As sign of 'A' is 1 add M to A.

```
 11111
 00011.
 -----
 00010 — Remainder.
```



Nonrestoring Division.

FLOATING-POINT NUMBERS AND OPERATIONS

$$1.01101 = 1\times2^{0} + 1\times2^{-2} + 1\times2^{-3} + 1\times2^{-5}$$
$$= 1 + 1\times\frac{1}{4} + 1\times\frac{1}{8} + 1\times\frac{1}{32}.$$
$$= 1 + 0.25 + 0.125 + 0.03125$$
$$= 1.40625.$$

Range of values for 32bit, signed, fixed-point format = 0 to $\pm 2.15\times10^{9}$. (Only Integer)

for 32bit only fraction = $\pm 4.55\times10^{-10}$ to ± 1.

Neither of this range is enough to represent values such as Avogadro's no. (6.0247×10^{23} mole^{-1}) & Planck's constant (6.6254×10^{-27} erg.s).

To represent very large integers and very small fractions we must have a floating binary point. Such no. are called floating point no.s. where fixed-point number have it's binary point fixed.

In case of below no. 6.0247×10^{23}
6.6254×10^{-27}

the no. of significant digits = 5.
Scale factors = $10^{23}, 10^{-27}$.

Scale factors indicate the position of decimal point.

If the decimal point is given after first significant bit, then those no.s are called normalized no.s.

A string of significant digits commonly called the mantissa.

Scale factor known as exponent.

IEEE-754 Standard for Floating-Point Numbers

1. ~~Sg~~ Single-Precision (32 bit)
2. Double Precision (64 bit)
3. Extended Precision (80 bit).

Single Precision

S	E'	M.
Sign of number 0 → +ve 1 → -ve	8 bit signed exponent in excess-127 representation	23-bit mantissa fraction

Instead of the Signed exponent E, the value actually stored in the exponent field is an Unsigned integer E' = E+127. This is called the excess-127 format. Thus E' is in the range $0 \leq E' \leq 255$. The end values 0 and 255 are used to represent special values, as described below. Therefore E' for normal values is $1 \leq E' \leq 254$. This means E is in the range -126 to 127.

As normalized ~~m~~ mantissa is used to store in mantissa the significant bit of mantissa i.e. left of decimal point is always 1. 23 bit mantissa does not explicitly stores this bit, only the fractional part is represented.

0	00101000	001010 - - - - - - 0

Exponent.
40 - 127
= -87

value represented = $1.001010 \cdots 0 \times 2^{-87}$

Special values.

When E' = 0 and mantissa fraction M is 0, the value is exactly 0.

when E' = 255 and M = 0, then values is ∞.

As sign bit is still part of these representation, ±0, ±∞ canbe represented.

when E' = 0 and M ≠ 0, denormal no.s i.e less than smallest normal no. is represented $(\pm 0.M \times 2^{-126})$

This is to allow gradual underflow.

When E' = 255 and M ≠ 0, the value is not a number.

- ~~A~~ A NaN is the result of performing an [illegible] -invalid operation such as 0/0 and ∞.

$0.0010110\ldots \times 2^9$ is represented as follows

0	10000101	0110 ....

$1.0110\ldots \times 2^6$ normalized version.

6+127
$E' = 133 = 128+4+1$

Double Precision.

The Scalefactor of single precision has a range of 2^{-126} to 2^{127}, which is approximately equal to $10^{\pm 38}$. The 24-bit mantissa provides approximately the same precision as a 7-digit decimal value.

The double precision has 11-bit excess-1023 exponent E', which has the range $-1022 \leq E \leq 1023$, providing scalefactor of 2^{-1022} to 2^{1023}. The 52-bit mantissa provides a precision equivalent to about 16 decimal digits.

If number can't be represent within the representable range then we might not get actual value. If the no. requires exponent less than -126 incase of single precision we call it underflow. On other side if exponent requires more than 127 we call it overflow.

64 bit

S	E'	M
	11-bit excess-1023 exponent.	52-bit mantissa.

Exceptions.

A processor must set exception flags, if any of the following occurs. Underflow, overflow, divide by zero, inexact, invalid. Inexact is the name for a result that requires rounding in order to be represented in one of the normal formats. An invalid exception occurs if operations such as 0/0 or $\sqrt{-1}$ are attempted.

Q 7 1) Convert 32.5×2^{10} in floating point single precision format.

2) 162.75×2^{-23}.

3) 157.5.

ARITHMATIC OPERATIONS ON FLOATING-POINT NUMBERS.

Add / Subtract Rule

1. Choose the number with the smaller exponent and shift it's mantissa right a number of step's equal to the difference in exponents.
2. Set the exponent of the result equal to the langer exponent
3. perform addition/subtraction on the mantissas and determine the sign of the result.
4. Normalize the resulting value, if necessary.

Q. $A = 1.02356 \times 10^{15}$, $B = 1.37853 \times 10^{18}$.
as exponent of A is less, shift mantissa of A

$A = 0.00102356 \times 10^{18}$.

Add mantissa of A & B.

$$\begin{array}{r} 0.00102356 \\ 1.37853 \\ \hline 1.37955356 \end{array}$$

Result of A+B = $1.37955356 \times 10^{18}$.

Q. Add two single precision floating point number.

A = 44900000H. B = 42A00000H.

Sign exponent mantissa.

0|10001001|00100000000000000000000

$E'_a = 137$

Sign exponent mantissa.

0|10000101|01000000000000000000000

mb.

$E'_b = 133$.

as E'_a is 4 more than E'_b, shift right 4-bit of mantissa of B.

mb = 1.0100 0000 - - - - -

mb = 0001 0100 0000 0000 0000 0000

Add ~~fractional part of~~ mantissa of A & B.

$$\begin{array}{rl} m_a = & 1.0010\,0000\,0000\,0000\,0000\,000 \\ mb = & 0.0001\,0100 \\ \hline & 1.0011\,0100 \end{array}$$

Result.

0	10001001	00110100 0000 00000000000

Multiply Rule.

1. Add the exponents and subtract 127.
2. Multiply the mantissas and determine the sign of the result.
3. Normalize the resulting value, if necessary

Divide Rule.

1. Subtract the exponents and add 127.
2. Divide the mantissas and determine the sign of the result.
3. Normalize the resulting value, if necessary

GUARD BITS AND TRUNCATION.

Mantissa is limited to 24 bits, including the implied leading 1. During intermediate steps or the final result, we may get mantissa morethan 24bits. These extra bits are called Guard Bits.

Keeping guardbits for intermediate steps will increase the accuracy. However guard bits of final results will be truncated.

When we simple remove the guard bits, it is called chopping. The error in the 3-bit result from 6bit ranges from 0 to 0.000111 i.e from 0 to almost 1 in the least significant position of retained bits. The result of chopping is a biased approximation because the error range is not symmetrical about '0'.

In Von Neumann rounding, the Least Significant bit of retained bits set to 1 if any of the guard bit is 1 otherwise guard bits are simple ignored. The error in this truncation method ranges from -1 to +1 in the LSB position of retained bits.

In rounding procedure, the LSB of retained bits is added with 1 if MSB of guard bits is 1, otherwise guard bits are ignored. The error range is approximately $-\frac{1}{2}$ to $+\frac{1}{2}$ in the LSB position of the retained bits. This is the IEEE default mode of truncation.

MOD-3

THE MEMORY SYSTEM

Some Basic Concepts

The maximum size of memory that can be used in any computer is determined by the addressing scheme. A 16bit addresses is capable of addressing upto $2^{16} = 64K$ memory location.

$2^{10} = 1KB$, $2^{20} = 1Mb$, $2^{30} = 1Gb$, $2^{40} = 1Tb$.

A machine whose instructions generate 32-bit addresses can utilize a memory that contains upto $2^{32} = 4G$ (giga) memory locations. The no. of locations represents the size of the address space of the computer.

Most modern computers are byte addressable. As far as the memory structure is concerned, there is no substantial difference between big endian & little endian schemes.

The memory is usually designed to store and retrieve data in word-length quantities. In a Read operation, other bytes may be fetched from the memory, but they are ignored by the processor. Simillarly if a byte of data needs to be written then control circuitry of the memory must ensure that the contents of other bytes of the same word are not changed.

If MAR is k bits long then the memory unit may contain up to 2^k addressable locations.

If MDR is n bits long, then during a memory cycle, n bits of data are transfered between the memory and the processor. (n - length of word)

The processor bus has 'k' address lines and 'n' datalines The bus also includes the control lines Read/write and Memory Function Completed (MFC) for coordinating data transfers. Other control lines may be added to indicate the no of bytes to be transfared.

During read, processor set $R/\bar{W}$ to 1 and load the address in MAR. Memory controller sets MFC after it load data in datalines.

During write, processor set $R/\bar{W}$ to 0 and load data in MDR.

Memory Accesstime : Memory accesstime is the time that elapses between the initiation of an operation nd the completion of that operation.
eg: Time betn Read nd MFC Signal.

Memory Cycle time : It is the minimum time delay required between initiation of two Successive memory operation.
eg: Time betn two Successive Read operation.
These two are important measure of the Speed of memory units. The cycle time is usually Slightly longer than the access time.

A memory Unit is called random-access memory if any location can be accessed in some fixed amount of time that is independent of the location's address.

Cache Memory

This is a Small, fast memory that is inserted between the larger, slower main memory nd the processor. It holds the currently active Segments of a program nd their data.

Processor can usually process instructions and data faster than can be fetched from memory Unit. To reduce the memory access time, Cache memory is used.

Virtual Memory

Size of main memory is Small nd most programs nd data don't fit into main memory. Virtual memory Concept makes us feel like we have main memory much more than the actual main memory.

The memory control circuitry translates the address specified by the program into an address that can be used to access the physical memory. This address generated by the processor called virtual address.

Data are addressed in a virtual address space that can be as large as the addressing capability of the processor. But at any given time, only the active portion of this Space is mapped onto locations in the

Physical memory. The remaining virtual addresses are mapped onto the bulk storage devices used, which are usually magnetic disk.

CACHE MEMORY

The speed of the main memory is very low in Comparison with the speed of modern processors. Processor Cannot spend much of it's time waiting to access instructions nd data in main memory. An efficient solution is to use a fast Cache memory which essentially makes the main memory appear to the processor to be faster than it really is.

The effectiveness of the Cache mechanism is based on a property of computer programs called locality of reference. Analysis of programs shows that many $inst^n$s in localized areas of the program, are executed repeatedly during some time period, and the remainder of the program is accessed relatively infrequently. This is referred to as locality of reference.

There are two aspects of Locality of reference:

1- Temporal   2- Spatial.

Temporal → Recently executed instruction is likely to be executed again very soon.

Spatial → Instructions in close proximity to a recently executed $inst^n$ are most likely to be executed soon.

To take advantage of this locality of reference, It is useful to fetch several items that reside at adjacent addresses as well. This block of contiguous address locations of some size also refer to Cacheline in case of Cache.

If the active segments of a Program Canbe placed in a fast cache memory, then the total execution time Canbe reduced significantly. But Size of Cache is not so big. Usually, the Cache memory Can store a reasonable no. of blocks at any given time, but this number is small Compared to the total number of blocks in main memory.

Replacement Algorithm

When the cache is full and a memory word that's not in the cache is referenced, the cache control hardware must decide which block of data must be replaced for the new block. The collection of rules for making this decision constitutes the replacement algorithm.

hit / miss

If the referenced data is available in Cache then we call it as read or write hit. Otherwise we call it Cache miss / Read miss. Write miss.

Write-through and Write-back or Copy-back

When we read from Cache, there is no problem. If we need to write something onto memory then we can proceed in two ways.

In case of write-through the cache location and the main memory location are updated simultaneously.

In write-back or Copy-back protocol, only update the Cache location and mark the associated flag bit, often called the dirty or modified bit. The main memory is updated later when the block containing this marked word is to be removed from the Cache.

Read Miss.

When a read miss occurs, the block contains the word loaded into cache. The requested word can be sent to processor after loading it to cache or can be sent after reading from main memory. This latter approach, is called load through, or early restart, reduces the processor's waiting period at the expense of more complex circuitry.

Write miss.

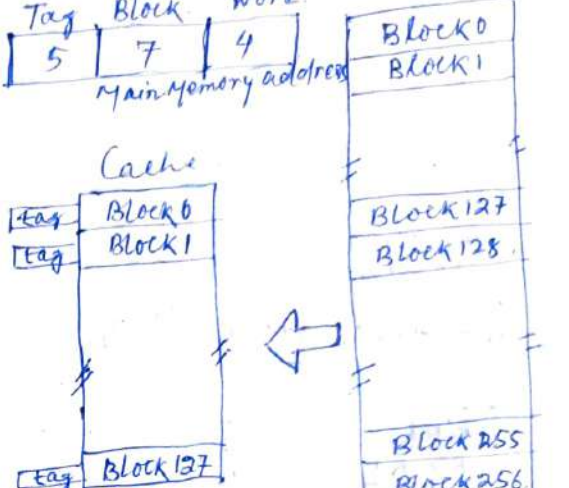
In case of write through protocol, information is written directly into the main memory. In the case of the write-back, the block containing the addressed word is first brought into the cache and then the desired word in the cache is overwritten with the new information.

MAPPING FUNCTIONS

The Correspondence between the main memory blocks and those in the Cache is specified by a mapping function.

Assumption. For Simplicity let's assume Cache is of 128 blocks of 16 words each, for a total of 2048 words. and Main memory is addressable by a 16-bit address. The main memory has 64K words, which we will view as 4K blocks of 16 words each. Also we will assume that Consecutive addresses refer to Consecutive words

Direct Mapping.



This is the Simplest way to determine the Cache Locations in which to Store memory blocks.

→ In this technique, block 'j' of main memory maps onto block j modulo 128 of the Cache (j%128).

eg:- 0, 128, 256 loaded on the Block '0' of Cache. Block 1, 129, 257 ... loaded on the Block '1' of Cache.

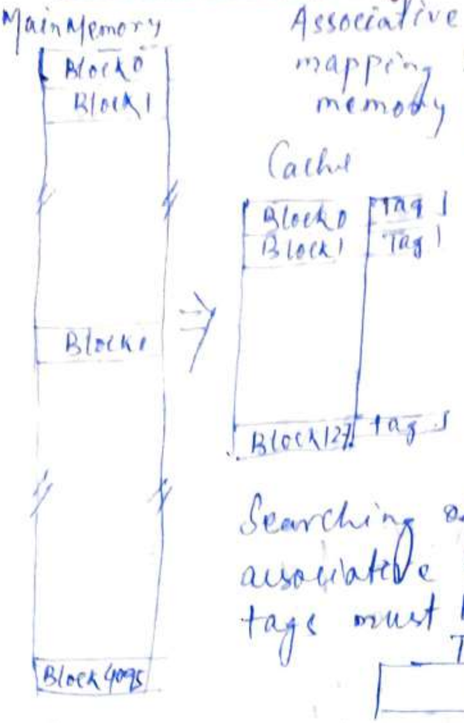
Disadvantage: Since One Block of Cache is reserved for more than one block of main memory, Contention may arise for that position even when the Cache is not full.

eg:- Instruction of a program may start in block 1 and continue in block 129, possibly after a branch.

Replacement Strategy is trivial. No replacement algo. required.

→ ~~Memory~~ From ~~to~~ 16 bit memory address of Main memory, 4bit from LSB decides the word. Next 7bit decides the block no. where the block is stored in Cache. Rest ~~@~~ 5bit is tag, which is used to find whether that particular block present in Cache or not by Comparing the tag bits of Corresponding Cache block with this 5 bit tag bits.

Associative Mapping

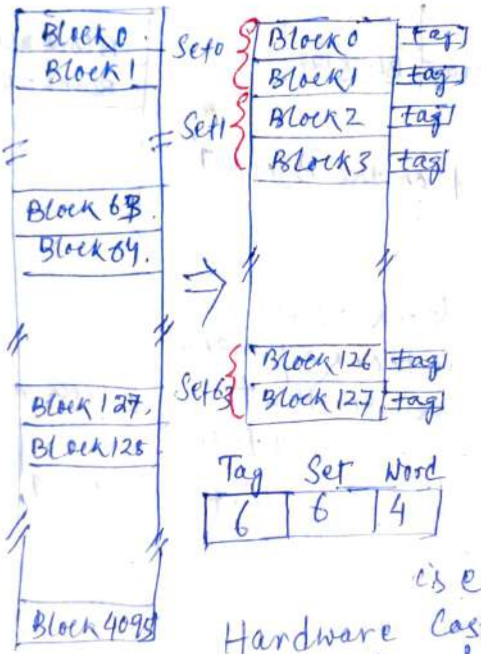


Associative mapping is a much more flexible mapping than Direct mapping. A main memory block Can be placed in any block of Cache memory. 4 LSB bits of main memory address is for word. Rest 12 bits used for tag. During Search 12bits tag is Compared with tag bits of all the blocks of Cache. Searching of 12 bits tag is known as associative Search. For Performance reason tags must be searched in parallel.

Tag	Word
12	4

In this Case We need replacement algorithm. Hardware Cost is high due to associative Search

Set Associative Mapping.



A Combination of the direct and associative-mapping techniques ~~Can be~~ used and is Called Set-Associative Mapping. In this Blocks of the Cache are grouped into sets, and the mapping allows a block of the main memory to reside in any block of Specific Set.

Hence the Contention problem is eased by having a few choices. Hardware Cost is reduced as the size of associative Search reduced.

In our example for 2-block set, memory blocks 0, 64, 128, ..., 4032 map into Cache Set 0, and they Can occupy either of the two block positions within this Set. Having 64 Sets means that 6-bit set field determines the set and 6bit tag is Compared for that set during Comparison.

A Cache that has K-blocks per set ~~reffered~~ referred to as a K-way Set associative Cache.

	Set	Tag
2-way -	6	6
4-way -	5	7
8- " -	4	8

Valid bit This is a Control bit which indicates, whether the block Contains valid data. This is different than dirty or modified bit. 0-invalid/stale data. 1-validdata. It ensures stale data doesnot present in cache.

Transfers from Disk to the main memory are Carried out by a DMA mechanism. (Direct Memory Access) Normally DMA bypass Cache for Cost & performance reason.

Cache Coherence problem :-

If write-back protocol is used, until we transfer the block to mainmemory from Cache, mainmemory doesnot have the updated copy. If during this period DMA transfer is made from main memory to disk, we have two different versions of data i.e one with disk and other with processor. This problem known as Cache coherence problem.

One Solution to this is, before DMA operation transfer the blocks with dirty bit '1' to mainmemory and then perform DMA.

Replacement Algorithm.

In a direct-mapped Cache, the position of each block is predetermined. hence no replacement strategy exists. In associative and set-associative caches there exists some flexibility.

The property of locality of reference in programs gives a clue to a reasonable strategy. Because Programs usually stay in localized areas for reasonable periods of time, there is a high probability that the blocks that have been referenced recently will be referenced again soon. Therefore, when a block is to be overwritten it is sensible to overwrite the one that has gone the longest time without being referenced. This block is called the least recently used (LRU) block and the technique is called the LRU replacement algo.

LRU Replacement Algorithm

Cache controller uses a counter for each block to track references to that block. for 4-block set associative cache a 2bit counter is required for each block. The counter of each block is set by following rule.

1. When a hit occurs, the counter of the block that is referenced is set to '0'. Counter with values originally lower than the referenced one are incremented by one, all other remains unchanged.
2. When a miss occurs and the set is not full, the counter associated with the new block loaded from main memory is set to '0' and all other counters of filled block are increased by one.

3. When a miss occurs and the set is full, the block with the highest counter value (for 4 set it is 3) is removed, the new block is put in its place, and its counter is set to 0. The other block counters are incremented by 1.

LRU Performs well except when access are made to sequential elements of an array that is too large to fit into cache. Performance of LRU algorithm can be improved by introducing a small amount of randomness in deciding which block to replace.

The simplest algorithm is to randomly choose the block to be overwritten. Infact this simple algorithm has been found to be quite effective.

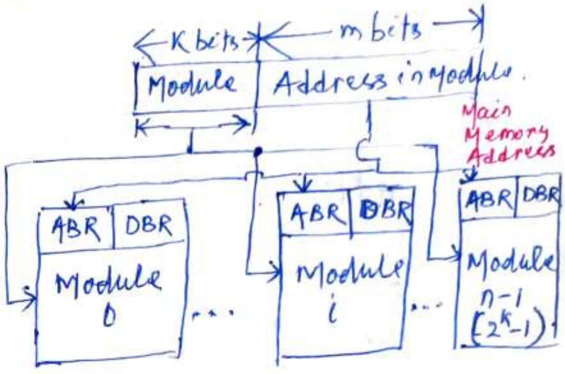
PERFORMANCE CONSIDERATIONS

Two Key faltors in the Commercial Success of a Computer are performance and Cost. A Common measure of Success is the price/performance ratio.

Interleaving.

If the main memory of a computer is structured as a collection of physically separate modules, each with it's own address buffer register (ABR) and data buffer register (DBR), memory access operations may proceed in more than one module at the same time.

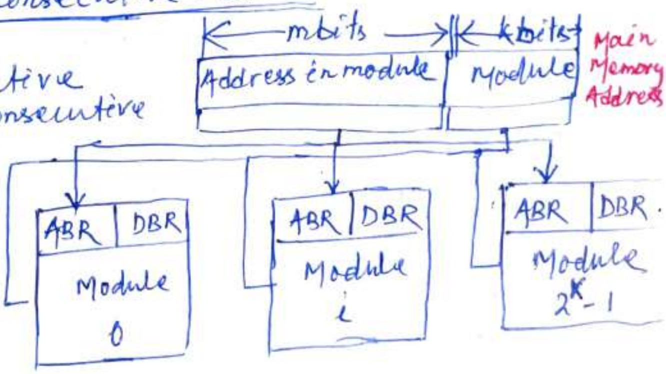
Consecutive Words in a Module.



If whole mainmemory is divided into 2^k modules, Keeping Consecutive Words in a Module, then the first K-bits decides the Module and rest m-bits decides the address in module.

In this method if Consecutive locations are accessed, Only one module is involved. At the Same time, however, devices with DMA abibity may be accessing information in other modulas.

Consecutive Words in Consecutive modules.



In this method Consecutive words are stored in Consecutive modules. If request is generated to access Consecutive memory locations, then Several modules will be busy at any one time. This results in both faster access to a block of data and higher avg. utilization of the memory system as a whole. This method of dividing main memory into modules Known as memory iterleaving. In this method the low-order K bits of the memory address select a module and high-order m bits name a location with in that module.