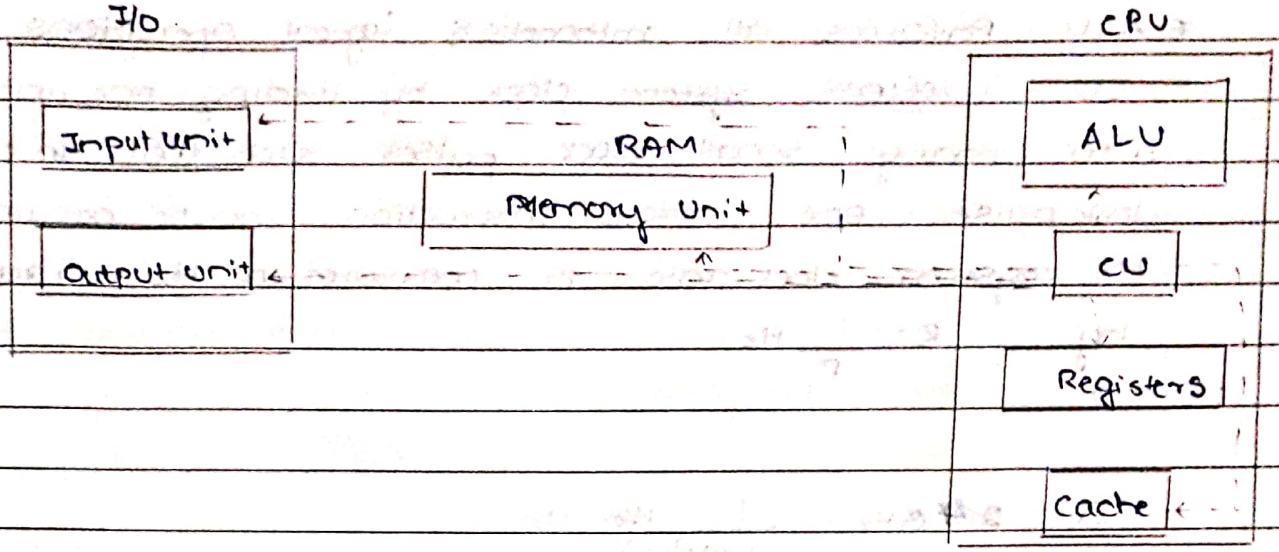
Camlin Page
Date 10 / 8 / 17

Module - I

Functional Block diagram of computer.



→ Input Unit: It accepts I/P from user encodes it to equivalent binary using ASCII standards & stores it in memory.
Ex: Keyboard, mouse.

→ Output Unit: Decodes the output in the memory & present it in human understandable
Ex: Monitor, Printer

→ Memory unit: It is made up of millions of semi-conductor cells. Each cell can hold one bit of information.
8 bits are grouped together to form one byte & each byte is given a separate address [byte addressable]
set of bytes (either 2, 4 or 8 bytes) grouped together form one word. The data transfer & processing happens word by word.
The time to access any word is same [few millisecs] so, it is called as Random Access Memory. Any prog or application or data should be brought into main memory before processing.

Date / /

→ CPU is a one inch by 1-inch chip comprising of ALU, CU, registers & Cache

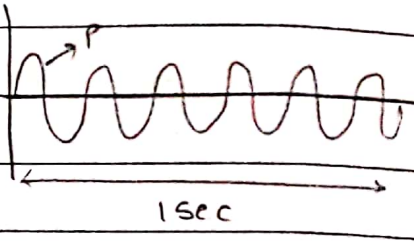
* ALU : Performs all arithmetic & logical operations

* CU : i) defines system clock by dividing one unit of time into many small clock pulses such that in each clock pulse one micro instruction can be executed. The system clock rate is measured in Hz & represented by

$$R = \frac{1}{P} \text{ Hz}$$

(clock rate, clock pulse) (2.4)

1 CP = 1 micro instruction



Ex: 2.4 GHz = $\frac{1}{2.4 \times 10^9 (P)}$ Hz.

i.e processor can executed 2.4×10^9 micro instructions in 1 sec.

2) Control unit sends control signal to all functional units in each clock pulse specifying what action to be taken or task to perform.

* Cache (semiconductor memory) : is used to store the frequently accessed data on processor chip

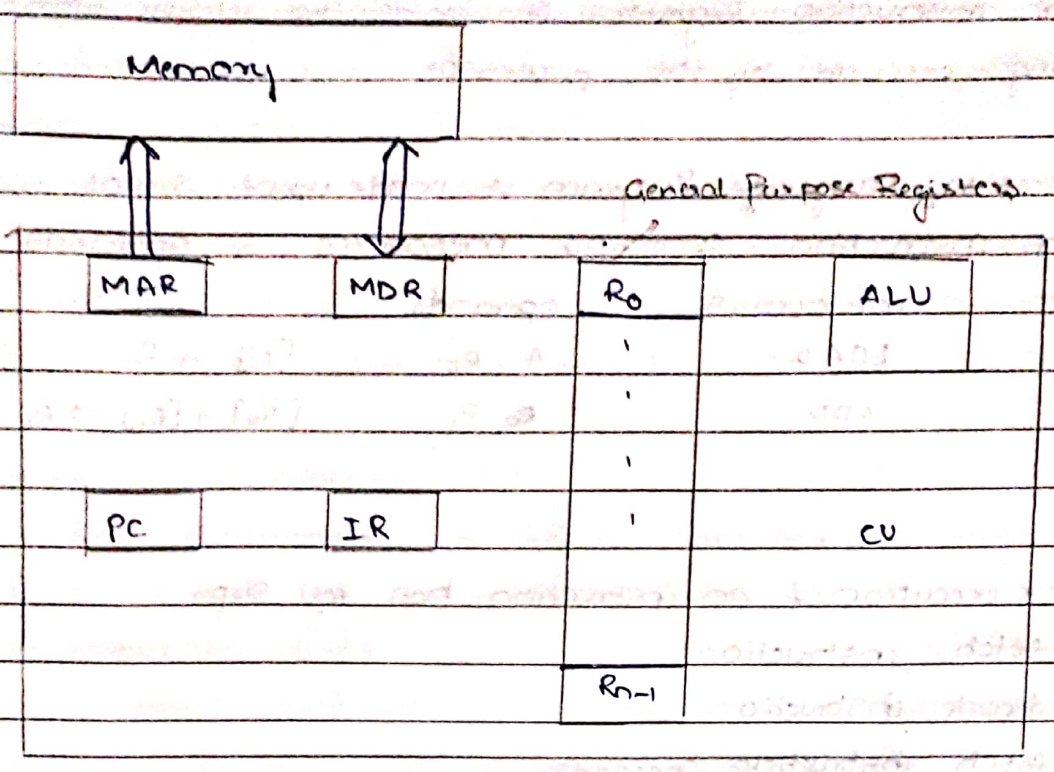
* Registers are semi-conductor memory on processor chip used to store intermediate results

Computer organisation deals with 2 things 1) Computer Hardware design like memory, IO, Arithmetic etc
2) Computer architecture like instruction set, addressing mode etc.

Camlin Page
Date 12 / 8 / 17

★ BASIC STRUCTURE OF COMPUTER

(memory related questions)



The diagram shows the connectivity b/w processor & memory for execution of an instruction.

The processor contains set of registers ; they are the semi-conductor memory on the processor chip. Registers are classified into

a) general purpose registers - used to store operands or intermediate results

b) Special purpose registers used for data transfer b/w processor & memory and co-ordinate execution of instruction

→ MAR Memory address register : It holds the address of the memory location that has to be read or ~~return~~ written into by processor

→ MDR Memory Data Register : It holds the data that is read from or to be written into memory

→ PC Program counter It holds address of next instruction to be executed Initially pc will be loaded with the address of 1st instruction later it will be automatically incremented to point to next instruction.

Date / /

→ IR Instruction Register: It holds the instruction currently being executed by the processor.

NOTE: Assembly Language Program is made up of set of instructions. Each instruction contains menomics & operands

Ex:-

menomics	operands	
LOAD	A, R_0	$[A] \rightarrow R_0$
ADD	R_0, R_1	$[R_0] + [R_1] \rightarrow R_1$

(R_0 → source, R_1 → Destination.)

The execution of an instruction has four steps.

→ fetch instruction

→ decode instruction

→ fetch ~~instruction~~ operands.

→ Execute instruction

→ store back ~~instruction~~ result.

[ ] → contents

( ) → pointer.

Ex:

2000	LD	A, R_0
2004	LD	B, R_1
2008	ADD	R_0, R_1
2062	ST	R_1, C

	C = A + B	1st instruction	2nd	3rd
①	2000 → PC	(A) → MAR	(B) → MAR	$[R_0] + [R_1] \rightarrow R_1$
②	[PC] → MAR	Read memory	Read memory	go to ②
	Read memory	(MAR) → MDR	(MAR) → MDR	
③	(MAR) → MDR	[MDR] → R_0	[MDR] → R_1	4th
④	[MDR] → IR	go to ②	go to ②	$[R_1] \rightarrow$ MDR
⑤	PC ++			(C) → MAR
	Decode instruction.			write memory

Camlin Page
Date / /

★ Basic operational concepts of executing an instruction happens in following order

- Fetch instruction

initially Pc will be loaded with the adress of 1st instruction & the instruction will be fetched using MAR and MDR and will be placed in instruction register later PC is incremented to point to next instruction

- Decode instruction

The instruction in IR is decoded by instruction decoder

- fetch operands

If the instruction needs any of the operands, it will brought into registers using MAR & MDR

- Execute the instruction

The instruction will be executed by ALU & CU.

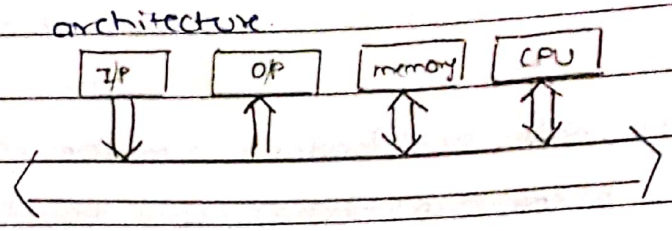
M = n*p

3000	LD	n, R_0	
3004	LD	p, R_1	
3008	MUL	R_0, R_1	$R_0 * R_1 \rightarrow R_1$
3012	ST	R_1, M	

①	3000 → PC	1st inst.	2nd.	3rd
②	[PC] → MAR	(n) → MAR	(p) → MAR	$[R_0] * [R_1] \rightarrow R_1$
③	Read memory	Read memory	Read memory	goto ②
④	(MAR) → MDR	(MAR) → MDR	(MAR) → MDR	4th
⑤	[MDR] → IR	[MDR] → R_0	[MDR] → R_1	$[R_1] \rightarrow$ MDR
⑥	PC++	go to ②	goto ②	(M) → MAR
⑦	Decode instruction			write memory
				goto ②

Bus Structure:

Single bus architecture



The simple way of connecting all the functional units of a computer is by using single bus architecture. A bus is a set of wires which can transmit electromagnetic signals. The problem with single bus is at any point of time only 2 devices can communicate. So, to avoid multiple buses are used to connect the functional units.

Performance of computer

It is measured by the amount of elapsed time for executing a program or process. The elapsed time includes CPU time & I/O time. Further, the CPU time involves memory time & wait time. To reduce the amount of memory time a small piece of semi-conductor memory is placed on processor chip called as cache.

The cache holds the data & instructions which are repeatedly used. When any data is brought from memory a copy is placed on the cache.

When processor requests for the data 1st it will be checked on cache if it is present it will be used (hit) else it will brought from memory (miss).

When cache becomes full the data which are not frequently accessed are replaced by new one

★ Basic performance equation

$$E = \frac{N * S}{R}$$

E = elapsed time
N = No. of instructions to be executed.
S = avg no of basic steps / instruction

E shld be less so N↓, S↓ & R↑

To improve the performance, elapsed time should be reduced i.e no. of instructions (N) & avg no of basic steps(S) need to be reduced & the clock rate of processor has to be increased.

The clock rate of processor can be increased by improving processor circuitary. The no. of instructions to be executed can be reduced by code optimisation by compiler. The compiler either rearranges the instruction or replace few instructions so as to improve performance of code. To reduce avg no of basic steps to be executed, two techniques are used

→ pipelining - overlaping the execution of successive instructions

→ parallel execution - by multiple cores of the processor.

There are 2 types of machine Architecture.

* RISC (Reduced instruction set computer)
* CISC (Complex instruction set computer

RISC	CISC
→ Simple instruction	→ Complex instruction
→ Simple address mode	→ Complex address mode
→ N↑ S↓	→ N↓ S↑
→ Pipelining	→ parallel execution.

So as to improve the performance with RISC pipelining is used & with CISC parallel execution is used.

imp * SPEC (System performance Evaluation Corporation)

$SPEC_i$ = Elapsed time on Reference machine / Elapsed time on New machine.

$$SPEC \left[\sum_{i=1}^{n} SPEC_i\right]^{1/n}$$

To improve the performance, elapsed time should be reduced & no. of instructions (N) & avg no of basic steps(S) need to be reduced & the clock rate of processor has to be increased

The clock rate of processor can be increased by improving processor circuitary. The no. of instructions to be executed can be reduced by code optimisation by compiler

The compiler either rearranges the instruction or replace few instructions so as to improve performance of code.

To reduce avg no of basic steps to be executed two techniques are used

→ pipelining - overlaping the execution of successive instructions

→ parallel execution - by multiple cores of the processor.

There are 2 types of machine Architecture.

* RISC (Reduced instruction set computer)
* CISC (Complex instruction set computer

RISC	CISC
→ Simple instruction	→ Complex instruction
→ Simple address mode	→ Complex address mode
→ N↑ S↓	→ N↓ S↑
→ Pipelining	→ parallel execution.

So as to improve the performance with RISC pipelining is used & with CISC parallel execution is used.

imp ~~*~~ SPEC (System performance Evaluation Corporation)

$$SPEC_i = \frac{\text{Elapsed time on Reference machine}}{\text{Elapsed time on New machine.}}$$

$$SPEC \left[\sum_{i=1}^{n} SPEC_i\right]^{1/n}$$

The Basic performance equation measures the performance based on N, S, & R. The spec provides the performance rating for each machine based on

→ Reference machine ie. PDP 11

→ n - bench mark programs chosen from different domains of computation such as Database, graphics, machine learning, etc. The time consumed for executing each of the bench mark program is recorded on reference machine & New machine & SPEC; is calculated

The average SPEC of n bench mark programs gives the performance rating

Machine instruction & Programs

Number representation: Decimal no.s are represented in Binary using BCD representation which is of the form

$B_n \times 2^n + \ldots + B_1 \times 2^1 + B_0 \times 2^0$

$\underset{D}{5} = \underset{B}{101}$ [0 to $2^n - 1$]

$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5$

using one bit we can represent 2 decimal no.s

Bin	Dec
0	0
1	1

2 bit:

Bin	dec.
0 0	0
0 1	1
1 0	2
1 1	3

3 bit:

bin	dec.
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

4 bit:

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

Camlin Page
Date / /

BCD represents only unsigned no.s.

17/8/17

signed no. representation

There are 3 different representations used to represent signed no.

1) Sign & Magnitude representation

The most significant bit of binary number represents the sign. If MSB = 0 +ve, MSB=1 -ve.

Ex:- 0101 +5 1101 -5

01010 +10 11010 -10

[disadv 0000 +0, 1000 -0]

2) 1's complement representation

The -ve no. is represented by obtaining the complement of each bit.

ex:- 101 +5 1010 +10

010 -5 0101 -10

Neither sign & magnitude nor 1's complement Representation is used to perform arithmetic by computers.

3) 2's complement representation

To obtain o -ve equivalent of a BCD no. 1st obtain 1's complement of a no. and add 1 at least significant bit

MSB ← 0101 ← LSB +5

1010

\+ 1

1011 -5

0001 +1

1110

\+ 1

1111 -1

3 bit

-4 to +3

	BCD	2's
000	0	0
001	1	1
010	2	2
011	3	3
100	4	-4
101	5	-3
110	6	-2
111	7	-1

-8 to 7

4 bits	BCD	2's
0 0 0 0	0	0
0 0 0 1	1	1
0 0 1 0	2	2
0 0 1 1	3	3
0 1 0 0	4	4
0 1 0 1	5	5
0 1 1 0	6	6
0 1 1 1	7	7
1 0 0 0	8	-8
1 0 0 1	9	-7
1 0 1 0	10	-6
1 0 1 1	11	-5
1 1 0 0	12	-4
1 1 0 1	13	-3
1 1 1 0	14	-2
1 1 1 1	15	-1

5 bits :- -16 to +15

6 bits :- -32 to +31

n bits :- -2^{n-1} to $+2^{n-1}-1$

2's complement Arithmetic

It is performed by addition. If it is +ve no then BCD equivalent of no. is considered & if it is negative no. then 2's c of no is considered.

After performing addition if there is a carry at MSB it will be ignored.

Ex: 2 + 2

```
  0 0 1 0
+ 0 0 1 0
---------
  0 1 0 0   ⇒ (4)
```

4 + 2

```
  0 1 0 0
+ 0 0 1 0
---------
  0 1 1 0   ⇒ (6)
```

4 − 2

```
  1 0 1 0 0
  1 1 1 0
---------
  0 0 1 0   ⇒ (2)
```

−3 −2

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

5 + 5

```
  0 1 0 1
  0 1 0 1
---------
  1 0 1 0   wrong (out of range)
```

−5 − 5

```
  1 0 1 1
  1 0 1 1
---------
  0 1 1 0   wrong
```

* Arithmetic overflow

while performing 2's complement arithmetic the following situation is called Arithmetic overflow

→ both operands are either positive or negative

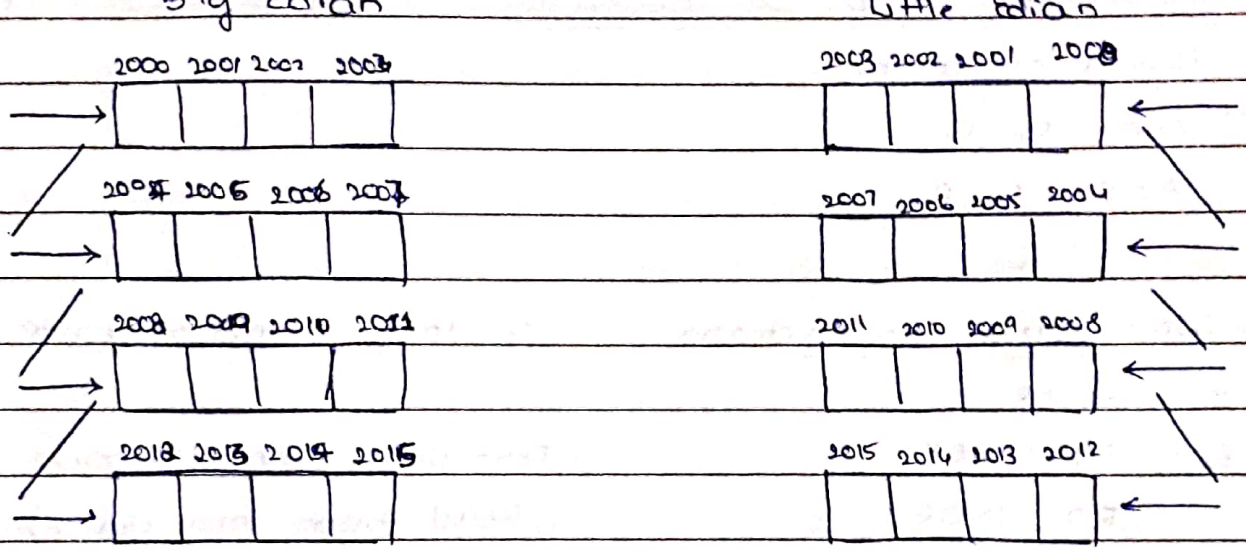
→ Result of the arithmetic is exceeding -2^{n-1} to $+2^{n-1}-1$ where n is no. of bits

Camlin Page
Date 19 / 8 / 17.

Memory location Address

Memory is made up of millions of semi-c cells. Each cell can hold one bit of information. 8 bits is grouped to form a byte & each byte is given a separate address. **Byte addressability**. To address 2^k bytes, k bits of address is needed

Big Edian & Little Edian representation



1 KB = 1024 bytes = 2^{10} location(bytes) → 10 bits of address

1 MB = 1024 × 1024 = 2^{20} bytes → 20 bits

1 GB = 1024 × 1024 × 1024 = 2^{30} bytes → 30 bits

Bytes within a word is arranged in 2 ways

→ Big Edian: If the lower order bytes are towards MSB & higher order bytes are towards LSB, then it is Big Edian

→ Little Edian: If lower order bytes are towards LSB & higher order bytes are towards MSB.

char - 1 byte.
int - 1 word.
float - 2 words.

Camlin | Page
Date / /

Instruction & Instruction sequency:

Every machine architecture will have its own built in set of instructions. These instructions can be categorised as

→ Data transfer instructions

Transfer data b/w memory & processor

Ex: LOAD A, R0
STORE R2, C

→ Arithmetic & logical instructions: To perform arithmetic & logical operations

Ex: ADD R0, R1
AND R3, R1

→ Input output instructions : To test & read or write I/O devices.

Ex: TD INDEX (Test_device deviceName)
RD INDEX (Read from input device).

→ Branch instructions : Depending on result of previous instruction jump to a particular address.

Ex: BR >0 LOOP

* Assembly Lang. program contains 3 types of statements

a) Declaration statements : used to reserve memory for variables & constants

Ex: A RESW 1 (A is int)
B RESB 1 (B is char)
C BYTE M (initialising char C to M) constant
D WORD 10 (" int D to 10) "

b) Instructions : To perform logical, arithmetic, I/O or branch operations
Ex: ADD R_1, R_0

c) Assembler Directives : The guidelines for assembler about how to assemble the program.
Ex: START 2000
:
STOP
Continuation *

NOTE Register transfer notations :-
Characters A, B, C etc represents memory location & general purpose registers are represented by $R_0, R_1, \ldots R_{n-1}$

We assume RISC machine where,
Size of instruction = size of int = size of Register = 1 word = 4 bytes (32 bit machine)

22/8/17 :

Basic instruction Type
→ Address instructions are classified based on no. of operands in instruction

* 3 address instruction : consists of 3 operands. The 1st & 2nd operand acts as source & the third is the destination.
Ex: ADD A, B, C
[A] + [B] → [C]
MUL R_0, R_1, R_2
$[R_0] * [R_1] \rightarrow R_2$

* 2 address register instruction
Ex: ADD A, B
[A] + [B] → B
MUL R_0, R_1
$[R_0] + [R_1] \rightarrow R_1$
Contains 2 operands, both acts as source & 2nd operand acts as destination.

Camlin Page
Date / /

* 1 address instruction

Ex: ADD R_0 LD C

[Accumulator] + [R_0] → [Accumulator] [C] → Accumulator

Contains one operand in instruction which acts as source. The 2nd operand & the destination will be accumulator which is a special purpose register.

* 0 address instruction

Ex: INC DEC

increment decrement

[ACC] + 1 → [ACC] [ACC] − 1 → [ACC]

No operands will be specific with instruction. The default operand is accumulator

Straight Line sequencing of instruction

Refer to Basic operation concept (fetching...)

imp * ADDRESSING MODES:

Different ways of specifying the operands in the instruction is called as addressing modes.

→ Register mode
→ Direct mode : Register direct
→ Immediate mode
→ Indirect mode : • Register & indirect • memory indirect
→ Index mode : Index with base register
→ PC relative
→ Auto Increment
→ Auto Decrement.

1) Register mode: The operands in instruction will be stored in registers and the result after the operation will also be stored in registers.

Ex: ADD R_0, R_1 MUL R_0

[R_0] + [R_1] → [R_1] [R_0] * [ACC] → ACC

RISC addressing mode

2) Direct mode: Cisc addressing mode

The operands specified in instruction are the name of memory location. The source & destination operands will be memory locations.

Ex: ADD A, B

[A] + [B] → B.

MUL M, N, Q

[M] * [N] → Q

Register direct : If one operand is memory location & other operand is register it is called Register direct

Ex: LD A, R0

[A] → R0

ST R2, C

[R2] → C

3) Immediate addressing mode : The value of operand is directly specified in instruction, prefixed by #

Ex: LD #20, R0 ; ADD #20, R1 ; MUL #3, A

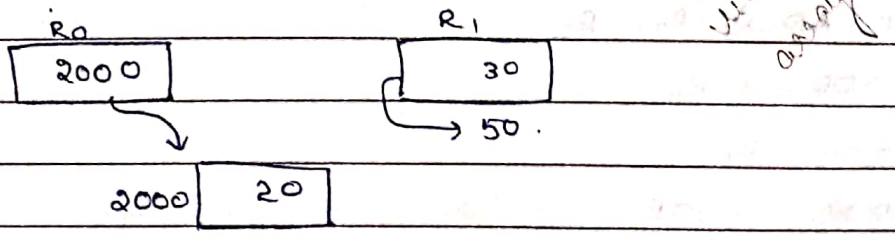
20 + [R1] → R1

3 * [A] → A

4) Indirect mode

→ Register direct : The register acts as a pointer, the register holds address of operand. During operation, the value of operand is fetched by using the address specified in register

Ex: ADD (R0), R1



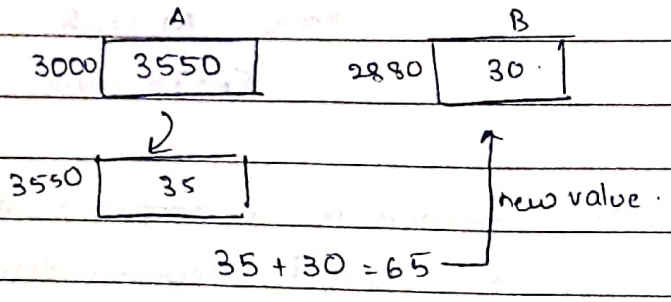
20 + [R1] → R1

20 + 30 → R1

50 → R1

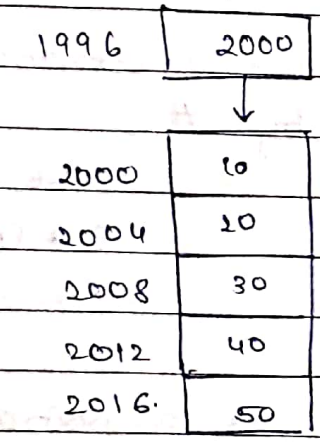
→ Memory indirect : The address of operand is specified in another memory location which is fetched during the operation.

ADD (A), B



Why we use indirect method.

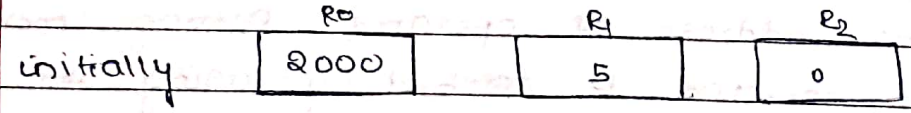
int a[5] = {10, 20, 30, 40, 50};



a[i] = base + i * w

word length.

23/8/17



LD #2000, R_0

LD #5, R_1

LOOP ADD (R0), R_2

ADD #4, R_0

DEC R_1

BR>0 LOOP (result of prev stmt)

The starting address of array is loaded on to R_0 and no. of elements in array is stored in R_1

ADD (R_0), R_2 will add the 1st element of array with R_2 then R_0 is incremented by 4 to point to 2nd element. R_1 is decremented and if R_1 value is greater than zero then we go back to add instruction

and add 2nd element to R_2. This process is repeated until R_1 becomes 0.

5) Indexed addressing mode

$X(R_i)$ — Symbolical representation

offset, base

$EA = X + [R_i]$; Effective address is computed using this formula

ADD 12(R_0), R_1

R_0	R_1
2000	50

EA = 12 + 2000 = 2012

40 + [R_1] → R_1

90 → R_1

2012	40

Why we use this? Advantage:-

```
struct student
{   int rollno;
    int m1;
    int m2;
    int m3;
};
struct student s[5];
```

R_0	R_1
2016	3

R_2	R_3
sum of R_4	Sum of R_2

	sum of R_3

1996	2000	
2000	1	S1
2004	55	S1
2008	58	S1
2012	63	S1
2016	2	S2
2020	40	S2
2024	50	S2
2028	60	S2
2032	3	S3
2036	80	S3
2040	90	S3
2044	100	S3
2048	4	

```
        LD   #2000, R0
        LD   #5, R1
LOOP    ADD  4(R0), R2
        ADD  8(R0), R3
        ADD  12(R0), R4
        ADD  #16, R0
        DEC  R1
        BR>0 LOOP
```

Scanned by CamScanner

The index addressing mode is used to perform operations on records or structures and matrix

Eg: Consider an array of structures having 5 elements where each record containing roll no, marks1, m2, m3. If the sum of each marks has to be added, R_0 is loaded to starting address of array, R_1 is loaded with no. of Records

```
ADD  4(R0), R2
ADD  8(R0), R3
ADD  12(R0), R4
```

} this will add m1 of 1st student to R_2, m2 of 1st student to R_3 & m3 of 1st student to R_4 respectively.

And ADD #16, R_0 will make R_0 to point to next student record

DEC R_1 ; reduces the count and if count is greater than zero, we go back and add next student marks.

This will be repeated until R_1 is zero

Indexing with base register

→ ADD (R_1, R_2), R_3

EA = [R_1] + [R_2] = 2020

2000 20

2020 | 10

10 + [R_3] → R_3

→ ADD 12(R_0, R_1), R_2

EA = 12 + [R_0] + [R_1]

[EA] + R_2 → R_2

=

6) PC relative addressing mode

It is used for conditional & unconditional branching

Branching: The sequencial execution of instruction is altered by unconditional or conditional branch instructions where execution changes from current instruction to target instruction

PC relative is symbolically represented as %(PC)

eg: TA = offset + PC

Ex: 3000 LD #3000, R_0 | TA = offset + PC

3004 LD #5, R_1 | offset = TA - PC

3008 LOOP ADD (R_0), R_2 | = 3008 - 3024

3012 ADD #4, R_0 | = -16

3016 DEC R_1 | TA = -16 + 3024

3020 BR>0 LOOP (-16) | = 3008

3024 ST R_2, Sum.

7) Auto increment: It is represented by (Ri) +. The effective address is computed by accessing the value pointed by the address in the register.

Incrementing the register to point to next element

ADD (R_0)+, R_1 ≡ R_0 2000 | R_1 10 | 2000 20

↓↓

ADD (R_0), R_1 10 + 10 = 20 (R_1)

ADD #4, R_0 after: R_0 2004

8) Auto decrement: Symbolically represented as -(R_0)

It represents, decrement contents of register to point to previous element. Access the operand by using address in register and perform operation.

ADD -(R_0), R_1 R_0 2000 | 1996 | R_1

↓↓

SUB #4, R_0 → R_0 1996 → 25 | 65

ADD (R_0), R_1 25 + 25 = 50 → R_1

LD #2000, R0

LD #5, R1

LOOP ADD #4, R0

ADD (R0)+, R2

ADD (R0)+, R3

ADD (R0)+, R4.

DEC R1

BR>0 LOOP

R0 = 20

R1 = 2000

R2 = 3000

R3 = 40

2020 = 35

2040 = 85

3040 = 65

3012 = 5

3012 = 50

2000 = 10

2996 = 68

3000 = 48

→ ADD 12(R1, R2), R3.

EA = 12 + 2000 + 3000 = 5012

50 + 40 = 90 → R3

→ ADD (R2)+, R3.

EA = 3000 + 48 + 48

= 3096 → R2

R2
3096

Indirect

→ ADD @(R1), R3

10 + 40 = 50 → R3.

→ ADD #50, (R0) immediate, indirect

50 + 10 = 60 → 2000. 2000 60

→ ADD 12(R2), R3 INDEXed.

12 + 3000 = 3012 → 5 + 40 = 45 → R3

→ ADD (R3, R2), R0

EA = 40 + 3000 = 3040.

65

65 + 20 = 85. → R0

→ ADD (R2)+, R3.

EA = 2996 + 0

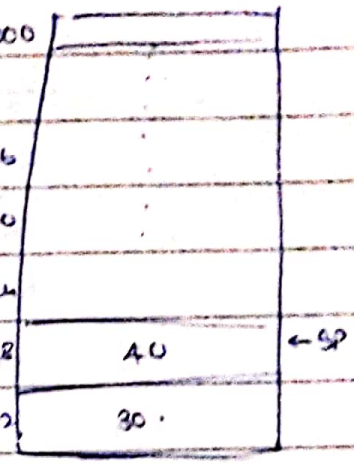
= 68 + 40

= 108

R3
2996

Stack

PUSH	POP
SUB #4, SP	MOV (SP)
MOV item, (SP)	ADD #4, SP
⇓	⇓
MOV item, -(SP)	MOV (SP)+, item
Auto decrement	Auto increment



used to perform stack operation.

Stack is a linear data structure with the list of elements & access restriction of Last In First Out.

In memory stack grows from higher address to lower address. Two operations are performed on stack, PUSH & POP. The top of stack is pointed by stack pointer.

Overflow and underflow.

PUSH	POP
CMP 2000, SP	CMP 2032, SP
BR < 0 overflow	BR > 0 underflow.

Compilation & Assembly language.

Assembly & execution of program.

```
SUM   EQU  3000
ORIGN 3004
N     WORD 3
ARR   RESW 5
ORIGIN 2000
ARR ADD START.
MOV #ARR, R0
MOV  N , R1
ADD  (R0), R2
ADD  #4, R0
DEC  R1
BR>0 LOOP
MOV R2, SUM
STOP
```

1) LOCCTR initially it will be 3000

2) SYMTAB

Sym	Type	Var/const	value/no.	Addr
N	int	const	5	3004
Arr	int	var	5	3008
SUM	int	var	1	3000

3) OPTAB

Menomics	opcode
MOV	FFH
ADD	FAH
BR	ABH.

2 Passes, 1st is declaration, in 2nd all the menomics is changed to opcode & all symbols with

The assembler assembles source prog in 2 passes

→ 1st pass:- evaluates assembler directives, assignes address for all the declared variables & constants & make entry into ~~single~~ symbol table.

The location counter allocates address for each instruction relatively.

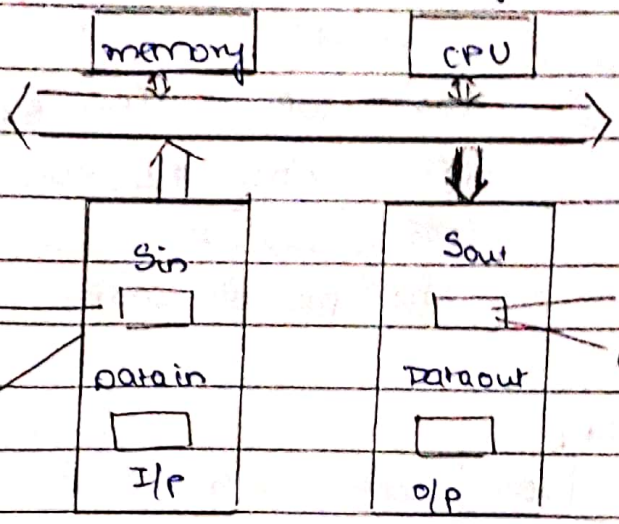
→ Pass 2:- All the executable instruction are converted into object code by refering opcode table

The optab contains menomics & its equivalent opcode in hexadecimal

In pass 2 all the operands will be converted into address. The resulting object code is written into a separate file

BASIC IO OPERATION: (Program controlled IO & Memory mapped IO)

Prog. controlled



Input & Output devices will have status flags Sin & Sout and the buffers $data_{in}$ & $data_{out}$.

Input operation:

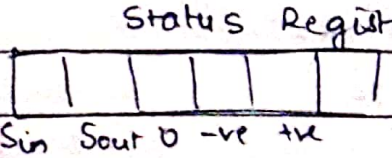
Initially Sin is checked. If $S_{in} = 0$ indicates input device is ready then one word of data is transfered from the device to $data_{in}$ and Sin is set to 1 indicating device is busy. After data is transfered from datain to memory Sin is again set to 0

Output Operation:

Initially Sout is checked. If Sout = 1 indicates ready then one word of data is transfered from memory to $data_{out}$ & Sout is set to 0. After data is transfered from $data_{out}$ to O/P device Sout is set to 1

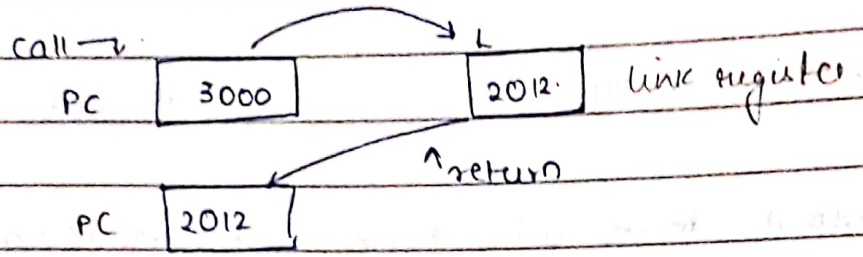
Memory mapped

It represents IO operation blw memory and IO devices. The Sin & Sow flags are present in Status register One of the memory location is used as IO buffer to store the data read or to be written



Subroutine

```
2000  MOV   #arr, R0          arradd
2004  MOV   N, R1       3000  LOOP  ADD (R0), R2
2008  call  arradd.      3004        ADD #4, R0
2012  MOV   R2, SUM      3008        DEC R1
                         3012        BP >0  LOOP
```



A subroutine is a set of instructions that perform a sub-task. The program which calls the subroutine (caller) branches to subroutine by storing the return address i.e the current value of PC is stored on Link register and PC will be loaded with the starting address of subroutine.

During return from subroutine the contents of Link register is loaded onto PC.

s|17

Parameter passing techniques

Pass by value The value of the operand is copied from actual to formal. Any change made to the formal will never affect the actual.

```
Ex:  MOV  N, R1
     DEC  R1
```

Pass by reference: The address of the actual parameter is passed. Any change made will affect the actuals.

```
Ex: MOV #arr, R0
    ADD (R0), R2
```

Camlin | Page
Date / /

Subroutine call using stack.

```
1000  MOV  #arr, -(SP)
1004  MOV  N , -(SP)
1008  call arradd
1012  MOV  SP, sum
```

2016	Sum ← SP
2020	[R2]
2024	[R1]
2028	[R0]
2032	1012 returnadd
2036	N
2040	#arr
2044	

```
arradd
.   MOV  R0, -SP      } Backup reg values
    MOV  R1, -SP
    MOV  R2, -SP.
    MOV  20(SP), R0   arr
    MOV  16(SP), R1   N
LOOP ADD (R0)+, R2
    DEC R1
    BR > 0  LOOP.
    MOV  R2, -(SP)
    MOV  12(SP), R0   } Restore
    MOV  8(SP), R1    } Reg values.
    MOV  4(SP), R2
    BR  16(SP)   - Return
```

During the call of subroutine if there is nested sub-routine call or more no. of parameters has to be passed then use of Link Register will not be a suitable sol^n. Hence stack is used to store the parameters before sub-routine call and the return address is automatically pushed to stack during call of sub-routine and after the execution of subroutine, control returns back to the caller by using the return address on the stack the contents on the stack is accessed by using indexed addressing mode.

Frame pointer:

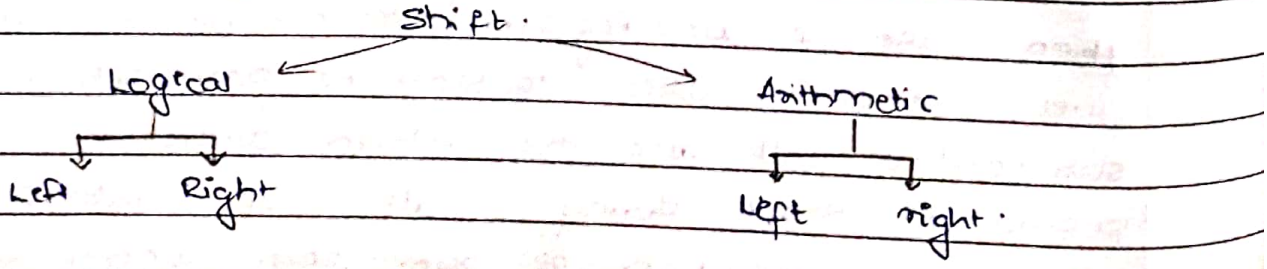
```
MOV  P1, -(SP)
MOV  P2, -(SP)
MOV  P3, -(SP)
Call Sub 1
MOV  FP1, -(SP)
MOV  P4, -(SP)
MOV  P5, -(SP)
Call Sub 2.
MOV  FP2, -(SP)
MOV  P6, -(SP)
MOV  P7, -(SP)
Call Sub 3
```

2004	ret Dummy	← SP
2008	FP_2	← FP_2
2012	ret Dummy	
2016	P_5	
2020	P_4	
2024	FP_1	← FP1
2028	return add	
2032	P_3	
2036	P_2	
2040	P_1	
2044		

Frame pointer is used to access all the parameters passed by a subroutine call in an easier way and to restrict unauthorised access. Every access to parameters is done by using indexing operation with frame pointers.

17

Additional instructions/operations:



Logical shift left:

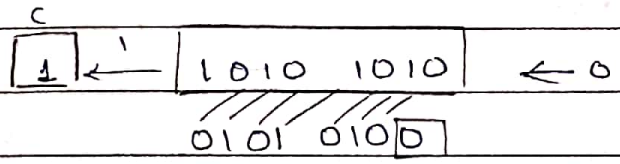
The specified no. of bits from MSB is moved to carry. The rest of the bits are shifted left and vacant position at LSB is filled with 0

Date / /

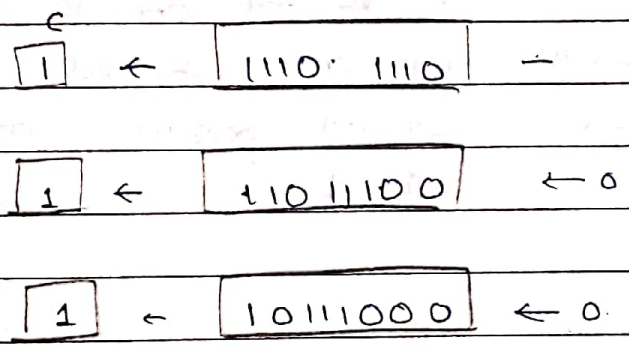
Logical shift Right

~~xxxx~~the specified no. of bits from LSB is shifted to carry. Rest of the bits are shifted right & the vacant position at msb is filled with zero

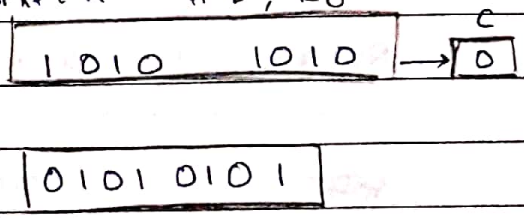
Ex:- LShiftL #1, R0



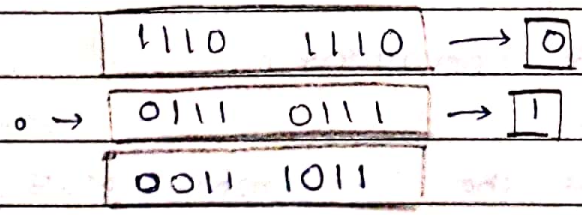
LShiftL #2, R0



LShiftR #1, R0



LShiftR #2, R1



Date / /

Arithmetic shift

Arithmetic left shift is same as logical left shift

Arithmetic shift right

Ex: AShift R #1, R_0

1010	1010	→	0

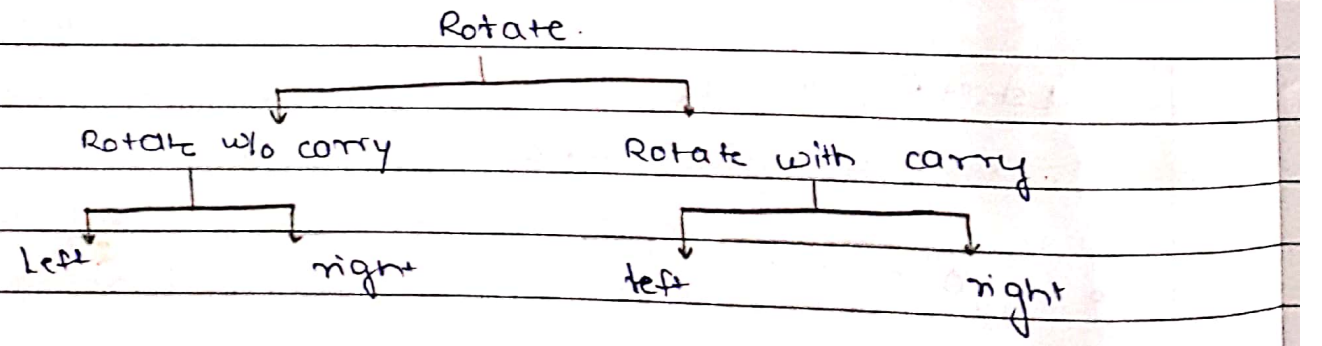
1 → | 1101 | 0101 |

which ever bit you shift always fill with 1

The specified no. of bits at LSB is shifted on to carry & the rest of bits are shifted right and the vacant position at MSB is filled with 1

AShR #3, R_1

1110	1110	→	0
1 → 1111	0111	→	1
1 → 1111	1011		



Rotate operation will place the shifted out bit to the other end either MSB or LSB.

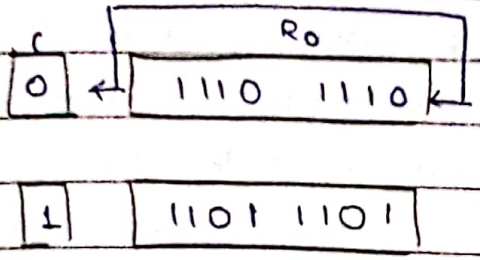
2 types of rotate operations

→ Rotate without carry

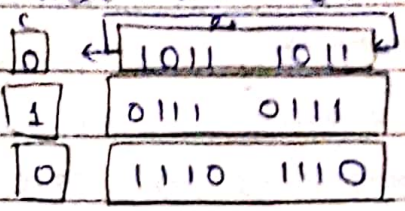
a) Rotate left - shift the bit from MSB, place the same bit in carry & also at LSB

b) Rotate right - shift bit from LSB, place the same bit in carry & also at MSB.

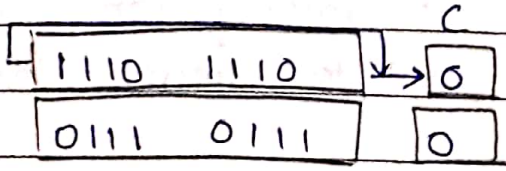
Ex: Rotate L#1, R_0



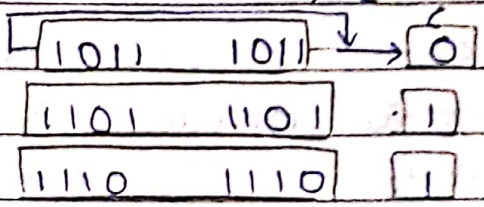
Rotate L#2, R_3



Rotate R#1, R_0

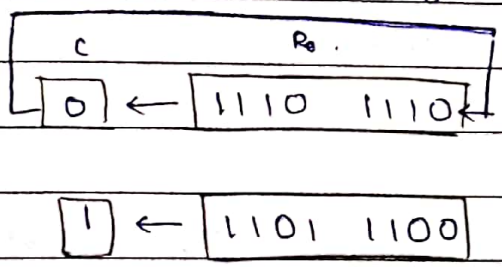


Rotate R#2, R_2

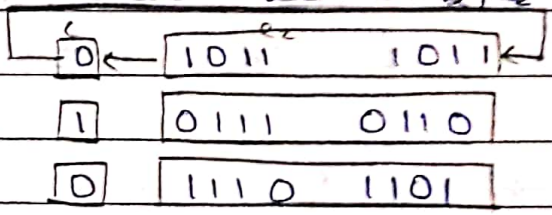


→ Rotate with carry

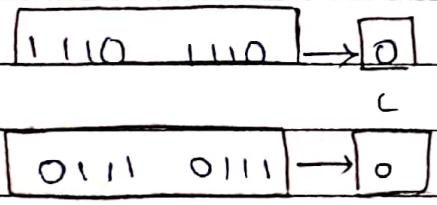
Ex: Rotate wc L#1, R_0



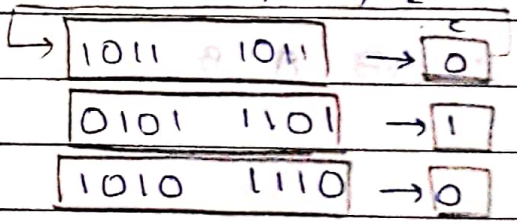
Rotate wc L#2, R_2



Rotate ~~wc~~ wcR#1, R_0



Rotate ~~wc~~ wcR#2, R_2



a) Rotate left - Shift bit from MSB to carry and place the carry bit at LSB

b) Rotate right - Shift bit from LSB to carry and place the carry bit at MSB

In RISC we can accommodate it 32 bits

Camlin Page
Date 6/9/17

Encoding Machine instructions:

Assembly language programs are written using mnemonics & operands. The assembler has to convert mnemonics to opcode and operands to address during assembly.

The process of conversion is called as Encoding

★ Assumptions

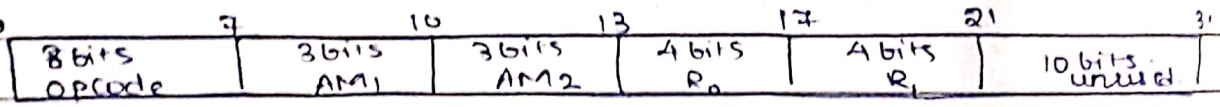
1) Each instruction is of 32-bit among which 8 bits specify opcode, 3 bits is used to specify addressing mode of each operand

2) The machine has 1MB of memory that is equal to 2^{20} bytes = 20 bits of address

3) 4 bits is used to represent each register
2^4 = 16 registers are used.

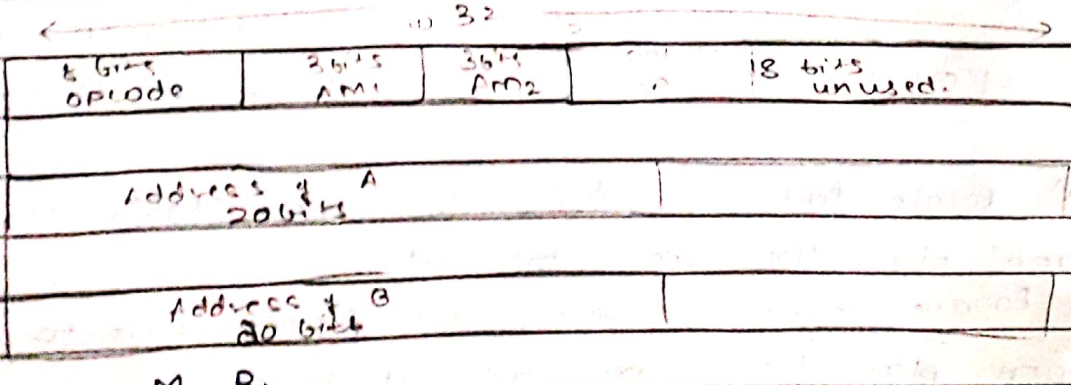
→ ADD R_0, R_1

CISC



→ ADD A, B

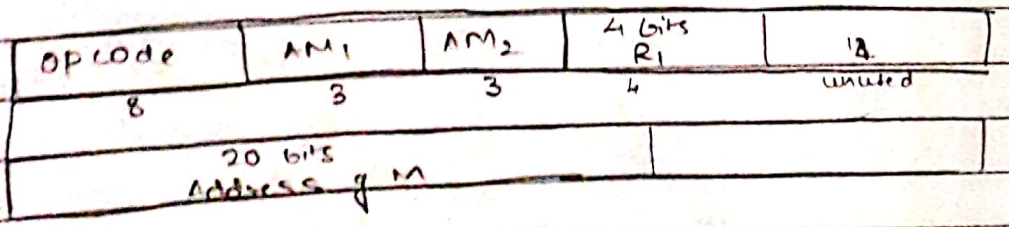
RISC



A & B are memory locations not registers

→ ADD M, R_1

CISC



8 bit + 8 bit → 8 bit

8 bit × 8 bit → 16 bit

8 bit ÷ 8 bit → 8 bit quo & 8 bit remainder

→ ADD #20, R1

-2^{14-1} to $+2^{14-1}-1$

20 can be represented in this range

10100

Opcode	AM1	AM2	R1	Offset
8	3	3	4	Values are assuming

→ MUL R0, R1

[R0] × [R1] → [R1]

32bit 32bit 64bit

If 32 bit is stored in R1 & next 32 bit in R0

Opcode	AM1	AM2	R0	R1	
8	3	3	4	4	10

We done by to specify cor opcode will tell

→ DIV R0, R1

[R0] / [R1] → Quotient [R0] Remainder [R1]

Opcode	AM1	AM2	R0	R1	
8	3	3	4	4	Unused

→ MOV (A), B.

opcode	AM1	AM2	
8	3	3	14

Address of A	
20	12

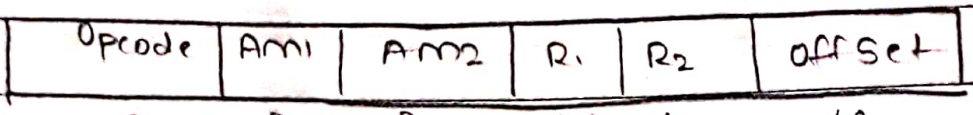
Address of B	
20	12

→ MOV #arr, R0

opcode	AM1	AM2	R0	
8	3	3	4	

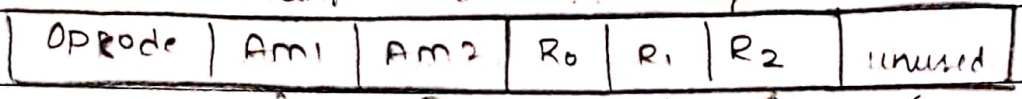
Address of arr	
20	

→ MOV 20(R1), R2

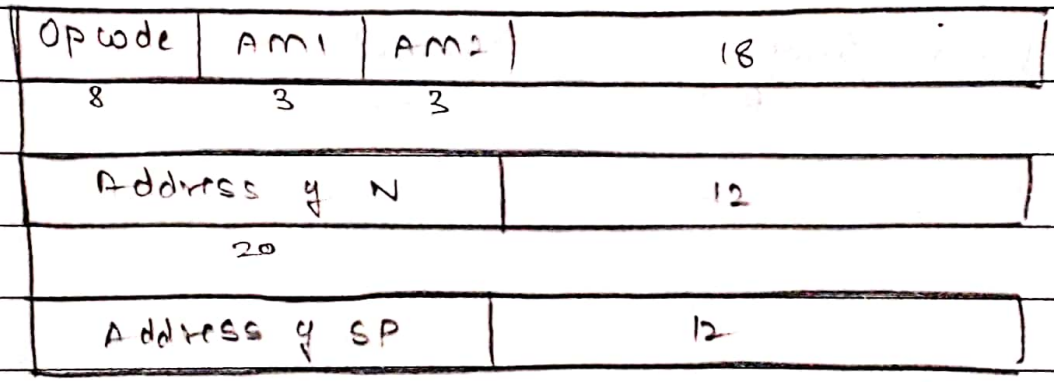


★ → ADD (R0, R1), R2

target address : only 2 addressing modes.



→ MOV N, -(SP)



MODULE-II

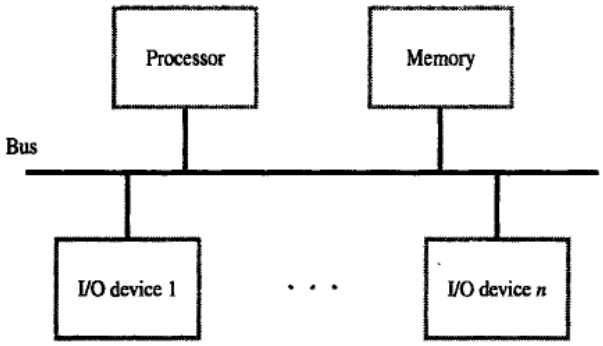
INPUT OUTPUT ORGANIZATION

Objectives

1. Accessing I/O Devices
2. Interrupts
   - 2.1 Interrupt Hardware
   - 2.2 Enabling & Disabling Interrupts
   - 2.3 Handling Multiple Devices
   - 2.4 Controlling Device Requests
   - 2.5 Exceptions
3. Direct Memory Access
4. Buses

1. Accessing I/O Devices:

A simple arrangement to connect I/O devices to a computer is to use a single bus arrangement.



1) The bus enables all the devices connected to it to exchange information.
2) Bus consists of three sets of lines used to **carry, data,** and **control lines**.
3) Each I/O device is assigned a **unique set of addresses**.
4) When the processor places a particular address on the address line, the device that recognizes this address responds to the commands issued on the control lines.
5) The processor requests either a **read** or a **write** operation, and the requested data are transferred over the data lines.

1.1 I/O interface for an input device:

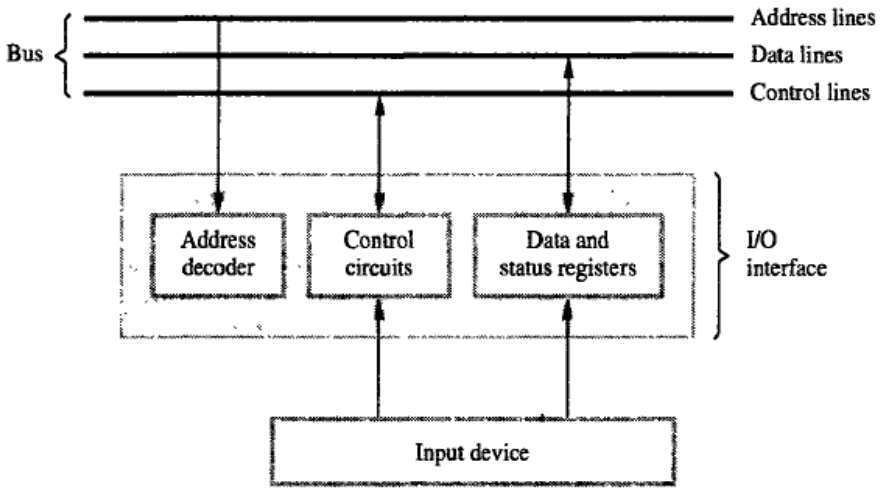
The hardware arrangement of connecting Input device to the system bus is called as "**Device Interface Or I/O Interface".**

I/O interface has three modules:

i)Address Decoder

ii) Control circuits

iii) Data & Status registers



i) Address Decoder:

- Address decoder is connected to the address lines of the bus as shown in the fig:
- **The function of address decoder is:** Address decoder enables the device to recognize its address when this address appears on the address bus.

ii) Data and Status registers:

- Data register holds the data being transferred to or from the processor.
- Status register holds information necessary for the operation of the I/O device.
- Data and status registers are connected to the data lines, and have unique addresses.

iii) Control Circuits:

- The control bus of system bus is connected to control circuit as shown in the fig.
- It controls the read write operations with respect to I/O device.
- I/O interface circuit coordinates I/O transfers.

1.3 i)Memory-mapped I/O:

- In this technique both memory and I/O devices can share the common memory to store the instruction as well as the operands.
- Memory related instructions are used for data transfer between I/O and processor.
- In case of memory mapped I/O input operation can be implemented as,

  MOVE DATAIN, R_0
- Similarly output can be implemented as,

  MOVE R_0,DATAOUT

ii) I/O mapped I/O:

- In this technique address spaces are different for I/O devices and memory.

  Processor has special IN and OUT instructions to perform I/O transfers.

  IN DATAIN, R0
- This instruction reads the data from the DATAIN and stores it into the register R0; DATAIN is the address of the buffer associated with the input device..

  OUT R0, DATAOUT
- This instruction sends the contents of the register R0 to location DATAOUT; DATAOUT is the address of the buffer associated with the output device.

Memory Mapped I/O	I/O Mapped I/O
1 .In this technique both I/O devices and memory share the same address space.	1. In this technique address spaces are different for I/O devices and memory.
2. Any machine instruction that can access memory can be used to transfer data to or from an I/O device.	2. In this technique processor has special IN and Out instructions to perform I/O transfers.

iii)Program-controlled I/O:

- In program-controlled I/O scheme the processor repeatedly checks a status flag of the I/O devices to achieve the required synchronization between the processor and an input or output device.

Disadvantage:

- Processor checks the status flag of I/O device, if I/O device not ready for data transfer, processor enters to wait loop.
- During this period, the processor is not performing any useful computation.
- There are many situations where other tasks can be performed while waiting for an I/O device to become ready.

2. Interrupts:

Definition: Interrupt is an event which suspends the execution of one program and begins the execution of another program.

Interrupt signals and its function:

i) INT(Interrupt):

Interrupt is a hardware signal sent by I/O device when it is become ready to alert the processor.

ii) INTR(Interrupt Request):

At least one of the control lines called interrupt request line is usually dedicated for this purpose.

iii) INTA(Interrupt Acknowledgement):

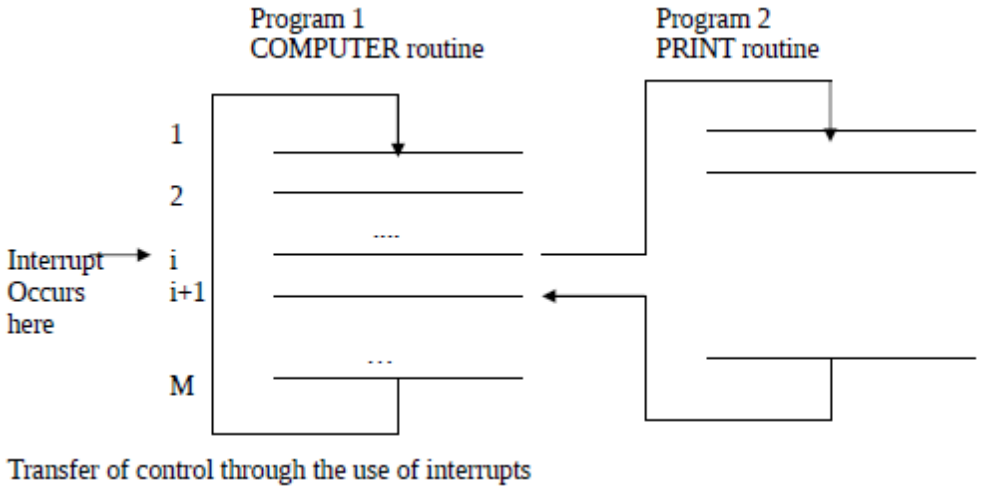
Interrupt-Acknowledge signal is a special signal issued by the processor to the device that its request has been recognized so that it may remove its interrupt request signal.

iv) ISR(interrupt-service routine):

The routine executed in response to an interrupt request is called the interrupt-service routine(interrupt program).

Working procedure of Interrupts

Assume that an interrupt request arrives during execution of instruction i.



Transfer of control through the use of interrupts

1) The processor first completes execution of instruction i.
2) Processor is executing the instruction located at address i when an interrupt occurs.
3) Interrupt program executed in response to an interrupt request is called the **interrupt-service routine.**
4) When an interrupt occurs, control must be transferred to the interrupt service routine.
5) But before transferring control, the current contents of the PC (i+1), must be saved in a known location.
6) This will enable the **return-from-interrupt instruction** to resume execution at i+1.
7) Return address, or the contents of the PC are usually stored on the processor stack.
8) Treatment of an interrupt-service routine is very similar to that of a subroutine.
9) the differences between subroutine and interrupts:
    a. A subroutine performs a task that is required by the calling program.

b. Interrupt-service routine may not have anything in common with the program it interrupts.
c. Interrupt-service routine and the program that it interrupts may belong to different users.
d. As a result, before branching to the interrupt-service routine, not only the PC, but other information such as condition code flags, and processor registers used by both the interrupted program and the interrupt service routine must be stored.
e. This will enable the interrupted program to resume execution upon return from interrupt service routine.

10) Saving and restoring information can be done automatically by the processor or explicitly by program instructions.

11) Saving and restoring registers involves memory transfers:

a. Increases the total execution time.
b. Increases the delay between the **time an interrupt request is received, and the start of execution of the interrupt-service routine**.
This delay is called **interrupt latency.**

12) In order to reduce the interrupt latency, most processors save only the minimal amount of information:

a. This minimal amount of information includes **Program Counter and processor status registers.**

13) Any additional information that must be saved , must be saved explicitly by the program instructions at the beginning of the interrupt service routine.

14) When a processor receives an interrupt-request, it must branch to the interrupt service routine.

15) It must also inform the device that it has recognized the interrupt request.

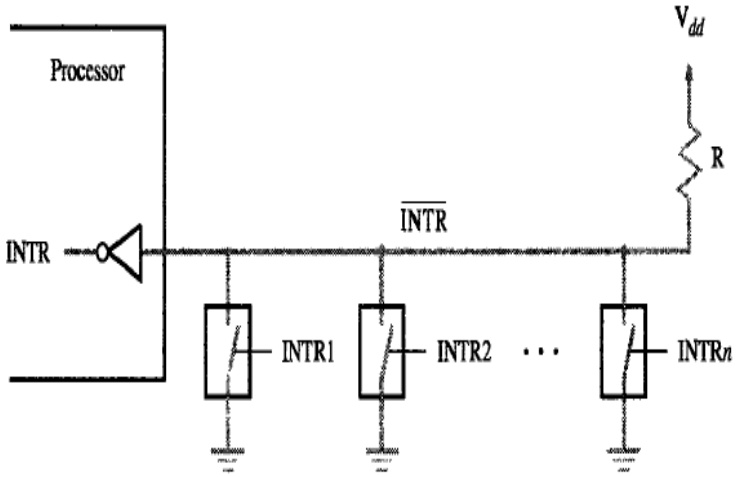
This can be accomplished in two ways:

i) Some processors have an explicit interrupt-acknowledge control signal for this purpose.

ii) In other cases, the data transfer that takes place between the device and the processor can be used to inform the device.

2.1 Interrupt hardware:-

1) The external device (I/O device) request the processor by activating one bus line and this bus line is called as interrupt request line.
2) The one end of this interrupt request line is connected to input power supply by means of pull up register is as shown in the fig.



3) Another end of interrupt request line is connected to INTR (Interrupt request) signal of processor as shown in the figure above.
4) The I/O device is connected to interrupt request line by means of switch as shown in the fig.
5) INTR is a INTerrupt Request signal which is sent by I/O device to request interrupt, INTR is active-low signal.
6) Depends on the interrupt switch on and off we can consider two stares:

i) In-Active State

ii) Active State

i)In-Active State:

- When all the switches are open the voltage drop on interrupt request line is equal to the V_{DD}.

  Therefore $\overline{INTR}=0$

- This state is called as in-active state of the interrupt request line.

ii)Active State:

- The I/O device interrupts the processor by closing its switch.

7) When switch is closed the voltage drop on the interrupt request line is found to be zero.

   Therefore $\overline{INTR}=1$

8) The signal on the interrupt request line is logical OR of requests from the several I/O devices.

   Therefore, $INTR=\overline{INTR_1} + \overline{INTR_2} + \ldots\ldots\ldots + \overline{INTR_n}$

2.2. Enabling and Disabling Interrupts:-

Disadvantages of interrupt:

1) Interrupt-requests interrupt the execution of a program, and may alter the intended sequence of events:

2) Sometimes such alterations may be undesirable, and must not be allowed.

Example: The processor may not want to be interrupted by the same device (either input or output device) while executing its interrupt-service routine.

To overcome above disadvantages, Processors generally provide the ability to enable and disable such interruptions as desired called Interrupt-Enable and Interrupt-Disable

There are 3 techniques to enable and disable the interrupts.

Method 1:

To avoid interruption by the same device during the execution of an interrupt service routine:

- First instruction of an interrupt service routine can be Interrupt-disable.
- Last instruction of an interrupt service routine can be Interrupt-enable.

Method 2: suitable for a simple processor with only one interrupt-request line:

1) The processor automatically disables interrupts before starting the execution of the interrupt service routine.

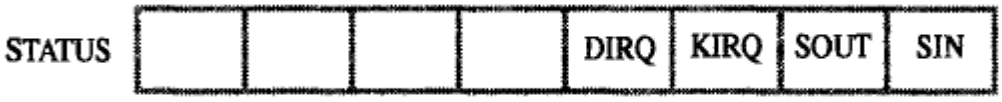
2) Uses the PS (Processor status) register,(1 bit in the PS register called interrupt-enable indicates whether interrupts enabled),when this bit is 1 interrupt is accepted.
3) After saving the contents of the PC and the processor status register (PS) on the stack, the processor clears the Interrupt-enable bit in its PS register, thus disabling further interrupts.
4) When return-from-interrupt is executed, the contents of the PS are restored from the stack, setting the interrupt-enable bit to 1.

Method 3: Uses special INTR line:

1. The device raises an interrupt request.
2. The processor interrupts the program currently being executed.
3. Interrupts are disabled by changing the control bits in the Processor Status register
4. The device is informed that its request has been recognized, and in response, it deactivates the interrupt-request signal.
5. The action requested by the interrupt is performed by the interrupt-service routine.
6. Interrupts are enabled and execution of the interrupted program is resumed.

2.3 HANDLING MULTIPLE DEVICES:-

1) The information needed to determine whether an input or output device is requesting an interrupt is available in its status register.



2) When a device raises an interrupt request, it sets IRQ bit to 1, which is in its status register.

Example: Bits KIRQ and DIRQ are the interrupt request bits for the keyboard and the display, respectively.

3) The simplest way to identify the interrupting device is to have the interrupt service routine poll all the I/O devices connected to the bus.
4) The first device encountered with its IRQ bit set is the device that should be serviced.
5) An appropriate subroutine is called to provide the requested service.

2.3.1 Polling scheme:

1) Processor uses polling mechanism to poll the status registers of I/O devices to determine which devices are polled.

2) The first device with status bit is set to 1 is the device whose interrupt request is accepted.
3) The polling scheme is easy to implement.
4) Its main disadvantage in polling scheme is the time spent interrogating the IRQ bits of all the devices that may not be requesting any service.

An alternative approach is to use vectored interrupts.

2.3.2 Vectored Interrupt:

1) To reduce the time involved in the polling process, using vector interrupts.

2) "A device requesting an interrupt indentifies itself by sending a special code to the processor over the bus".

3) This enables the processor to identify individual devices even if they share a single interrupt request line.

4) Interrupt vector code:

a) The **code** supplied by the device may represent the **starting address of the interrupt service routine for that device.**
b) The code length is typically in the range of **4 to 8 bits.**
c) The location pointed to by the interrupting device is used to store the starting address of the interrupt service routine.
d) The processor reads this address called as interrupt vector, and loads it into PC.
e) The interrupt vector may also include a new value for the processor status register.
f) In most computers, I/O Devices send the interrupt vector code over the data bus using the bus control signals to ensure that the devices do not interface with each other.
g) When a device sends an interrupt request, the processor may not be ready to receive the interrupt vector code immediately.
h) Then the processor can immediately transfer its service to interrupt service routine. Such interrupts are known as vectored interrupts.

5) The remainder of the address is supplied by the processor based on the area in its memory where the addresses for interrupt service routines are located.

2.3.3 Interrupt Nesting: -

1) Processor accepts the interrupt request from high-priority device while it is servicing another request from low-priority device.
2) Hence, branching from one interrupt –service to another interrupt-service is called interrupt-nesting.

Priority level:

3) An interrupt request from a high-priority device should be accepted while the processor is servicing another request from a lower-priority device.
4) To implement this scheme, assign a priority level to the processor that can be changed under program control.
5) The priority level of the processor is the priority of the program that is currently being executed.
6) The processor accepts interrupts only from devices that have priorities higher than its own.
7) The processor's priority is usually encoded in a few bits of the **processor status word**.

Privileged instructions:

8) Processor's priority can be changed by program instructions that write into the PS, These are privileged instructions, which can be executed only while the processor is running in the supervisor mode.
9) Processor works in different modes mainly supervisor mode and user mode
10) **supervisor mode:** The processor is in the supervisor mode only when executing operating system routines
11) **User mode:** The processor is in the user mode only when executes user(including I/O Interrupt programs) application program.

12) The processor is in the supervisor mode only when executing operating system routines.
13) It switches from supervisor mode to the user mode before beginning to execute application programs.
14) Thus, a user program cannot accidentally, or intentionally, change the priority of the processor and disrupt the system's operation.
15) An attempt to execute a privileged instruction while in the user mode leads to a special type of interrupt called a privileged instruction.

b)Multiple-Priority Scheme:

1) A multiple-priority scheme can be implemented easily by using separate interrupt-request and interrupt-acknowledge lines for each device, as shown in figure.

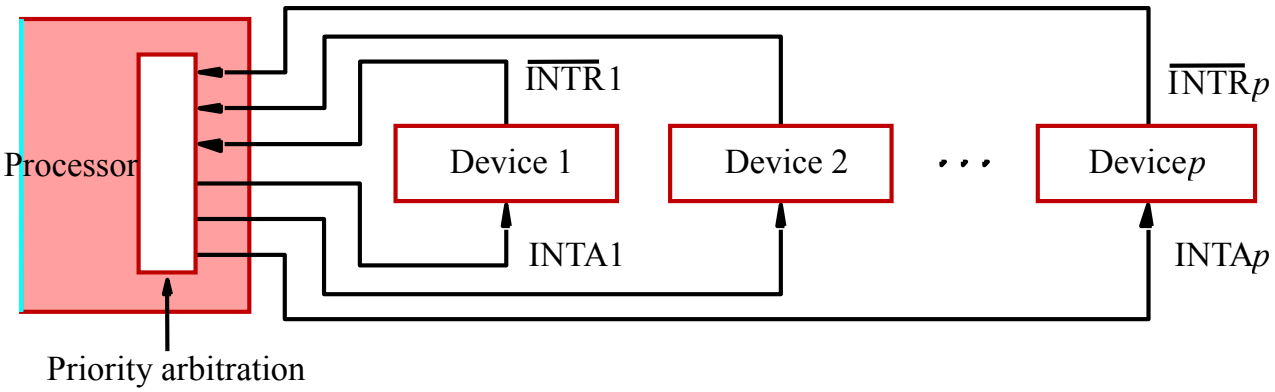


Fig: Implementation of interrupt priority using individual interrupt-request and acknowledge lines.

2) Each device has a separate interrupt-request and interrupt-acknowledge line.
3) Each interrupt-request line is assigned a different priority level.
4) Interrupt requests received over these lines are sent to a priority arbitration circuit in the processor.
5) If the interrupt request has a higher priority level than the priority of the processor, then the request is accepted.

2.3.4 Simultaneous Requests:-

Consider the problem of simultaneous arrivals of interrupt requests from two or more devices.

Daisy chain scheme:

Devices are connected to form a daisy chain as shown in the below fig.

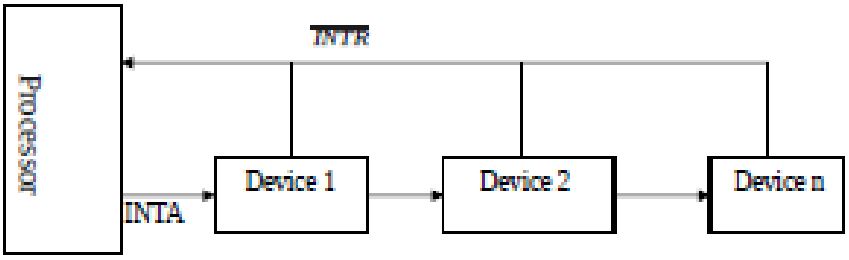


Fig: Daisy chain scheme

1) The processor simply accepts the requests having the highest priority.
2) Devices share the interrupt-request line using common (single) bus line.
3) Interrupt-acknowledge line is connected to form a daisy chain fashion.
4) When devices raise an interrupt request, the interrupt-request line is activated.
5) The processor in response activates interrupt-acknowledge.
6) Received by device 1, if device 1 does not need service, it passes the signal to device 2.
7) Device that is electrically closest to the processor has the highest priority.

Arrangement of priority groups using daisy-chain fashion:

1) When I/O devices were organized into a priority structure, each device had its own Interrupt-request and interrupt-acknowledge line.
2) When I/O devices were organized in a daisy chain fashion, the devices shared an interrupt-request line, and the interrupt-acknowledge propagated through the devices.
3) A combination of priority structure and daisy chain scheme can also used.

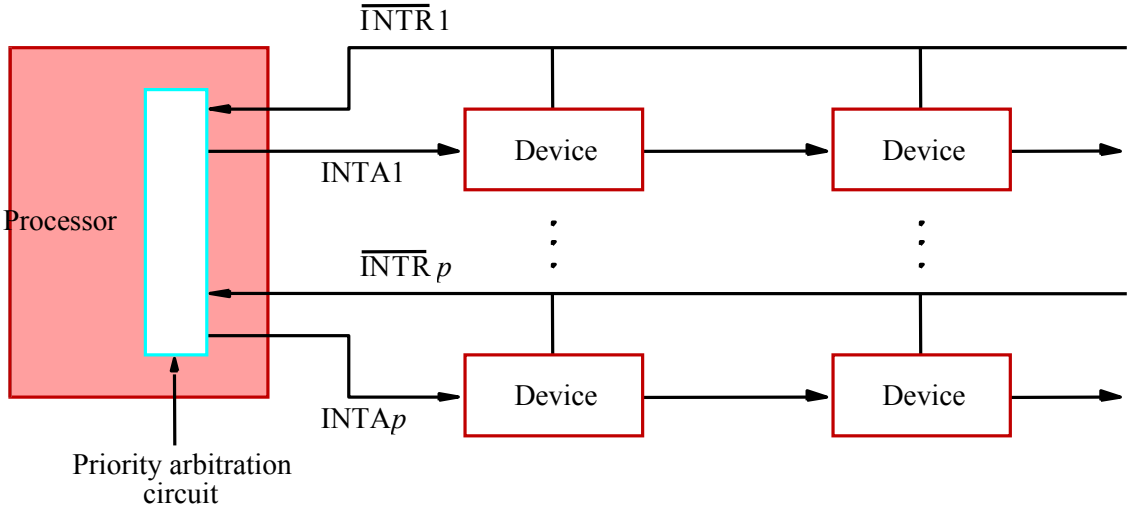


Fig: Arrangement of priority groups

4) Devices are organized into groups.
5) Each group is assigned a different priority level.
6) All the devices within a single group share an interrupt-request line, and are connected to form a daisy chain

2.3.5 Controlling Device Requests: -

1) It is important to ensure that interrupt requests are generated only by those I/O devices which is used by a given program.
2) Idle devices must not be allowed to generate interrupt requests, even though they may be ready to participate in I/O transfer operations.
3) The control needed is usually provided in the form of an interrupt-enable bit in the device's interface circuit.
4) The keyboard interrupt-enable, KEN, and display interrupt-enable, DEN, flags in **register CONTROL** perform this function.
5) If either of these flags is set, the interface circuit generates an interrupt request whenever the corresponding status flag in **register STATUS** is set.
6) At the same time, the interface circuit sets bit **KIRQ** or **DIRQ** to indicate that the keyboard or display unit, respectively, is requesting an interrupt.
7) If an interrupt-enable bit in PS is equal to 0, the interface circuit will not generate an interrupt request, regardless of the state of the status flag.

Summary:

There are two independent mechanisms for controlling interrupt requests:

1) At the device end, an interrupt-enable bit in a control register determines whether the device is allowed to generate an interrupt request.
2) At the processor end, either an interrupt enable bit in the PS register or a priority structure determines whether a given interrupt request will be accepted.

2.3.5 Exceptions:-

Interrupt: An interrupt is an event that causes the execution of one program to be suspended and the execution of another program to begin.

- The interrupt mechanism is used in a number of other situations.
- The term exception is used to refer to any event that causes an interruption.
- Hence, I/O interrupts are one example of an exception.

i)Recovery from Errors:-

- Computers use a variety of techniques to ensure that all hardware components are operating properly.

  Example:

  - Many computers include an error-checking code in the main memory, which allows detection of errors in the stored data.
  - If errors occur, the control hardware detects it and informs the processor by raising an interrupt.
- The processor may also interrupt a program if it detects an error or an unusual condition while executing the instructions of this program.

  Example:

  - The OP-code field of an instruction may not correspond to any legal instruction, or an arithmetic instruction may attempt a division by zero.
- When exception processing is initiated as a result of such errors, the processor proceeds in exactly the same manner as in the case of an I/O interrupt request.
- It suspends the program being executed and starts an Exception-Service Routine (ESR).
- This routine takes appropriate action to recover from the error, if possible, or to inform the user about it.
- Recall that in the case of an I/O interrupt, the processor completes execution of the instruction in progress before accepting the interrupt.
- However, when an interrupt is caused by an error, execution of the interrupted instruction cannot usually be completed, and the processor begins exception processing immediately.

ii)Debugging:-

Another important type of exception is used as an aid in debugging programs.

- System software usually includes a program called a debugger, which helps the programmer find errors in a program.
- The debugger uses exceptions to provide two important facilities called trace and breakpoints.

Trace:

1) When a processor is operating in the trace mode, an exception occurs after execution of every instruction, using the debugging program as the exception-service routine.
2) The debugging program enables the user to examine the contents of registers, memory locations, and so on.
3) On return from the debugging program, the next instruction in the program being debugged is executed, and then the debugging program is activated again.
4) The trace exception is disabled during the execution of the debugging program.

Breakpoint:

1) Breakpoint provides a similar facility, except that the program being debugged is interrupted only at specific points selected by the user.
2) An instruction called Trap or Software-interrupt is usually provided for this purpose.
3) Execution of this instruction results in exactly the same actions as when a hardware interrupt request is received.

Working of debugging program:

1) While debugging a program, the user may wish to interrupt program execution after instruction i.
2) The debugging routine saves instruction i+1 and replaces it with a software interrupt instruction.
3) When the program is executed and reaches that point, it is interrupted and the debugging routine is activated.

4) This gives the user a chance to examine memory and register contents.
5) When the user is ready to continue executing the program being debugged, the debugging routine restores the saved instruction that was a location i+1 and executes a Return-from-interrupt instruction.

iii)Privilege Exception:-

To protect the operating system of a computer from being corrupted by user programs, certain instructions can be executed only while the processor is in supervisor mode. These are called privileged instructions.

Example:

- when the processor is running in the user mode, it will not execute an instruction that changes the priority level of the processor or that enables a user program to access areas in the computer memory that have been allocated to other users.
- An attempt to execute such an instruction will produce privilege exceptions, causing the processor to switch to the supervisor mode and begin executing an appropriate routine in the operating system.

Direct Memory Access

Definition for DMA: "A special control unit used to provided to transfer a block of data with high speed directly between an I/O device and the main memory, without continuous intervention by the processor, this approach is called direct memory acces(DMA)"

Ex: Internal memory (RAM) data transfers and disk transfers uses DMA.

3.1)Direct Memory Access (DMA):

1) Control unit which performs DMA transfers is a part of the I/O device's interface circuit. This control unit is called as a DMA controller.
2) DMA controller performs functions that would be normally carried out by the processor:
   - For each word, it provides the memory address and all the control signals.
   - To transfer a block of data, it increments the memory addresses and keeps track of the number of transfers.

3) DMA controller can transfer a block of data from an external device to the processor, without any intervention from the processor.
4) However, the operation of the DMA controller must be under the control of any program executed by the processor. That is, the processor must initiate the DMA transfer.

5) To initiate the DMA transfer, the processor informs the DMA controller of:
   - Starting address
   - Number of words in the block.
   - Direction of transfer (I/O device to the memory, or memory to the I/O device).
6) After initiating the DMA transfer, the processor suspends the program that initiated the transfer, and continues with the execution of some other program.
   The program whose execution is suspended is said to be in the **blocked state.**
7) On receiving this information, the DMA controller proceeds to perform the requested operation.
8) Once the DMA controller completes the DMA transfer, it informs the processor by raising an interrupt signal.
9) While a DMA transfer is taking place, the program that requested the transfer cannot continue, and the processor can be used to execute another program.
10) After the DMA transfer is completed, the processor can return to the program that requested the transfer.

DMA with OS:

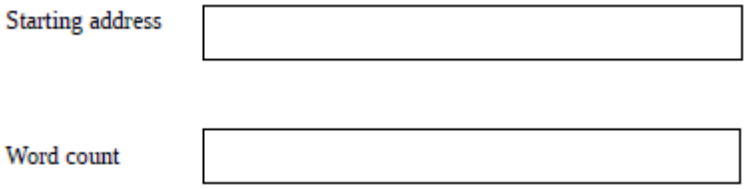
11) I/O operations are always performed by the operating system of the computer in response to a request from an application program.
12) The OS is also responsible for suspending the execution of one program and starting another.
13) Thus, for an I/O operation involving DMA, the OS puts the program that requested the 1transfer in the Blocked state, initiates the DMA operation, and starts the execution of another program.

14) When the transfer is completed, the DMA controller informs the processor by sending an interrupt request

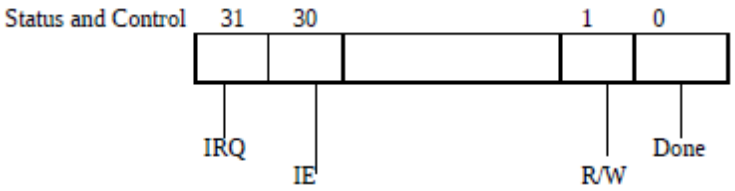
15) In response, the OS puts the suspended program in the Runnable state so that it can be selected by the scheduler to continue execution.

i) Registers of DMA:

1) Figure shows an example of the DMA controller three registers that are accessed by the processor to initiate transfer operations.
2) Two registers are used for storing the Starting address and the word count.



3) The third register contains status and control flags.



a)Read/Write flag register:

The R/W bit determines the direction of the transfer as explained below:

Case 1: Read operatio

When **R/W bit=1** by a program instruction,

The controller performs a **Read** operation,

That is, it transfers data from the **memory** to the **I/O device**.

Case 2: Write operation

When **R/W bit=0** by a program instruction,

The controller performs a **Write** operation,

That is, it transfers data from the **I/O device** to the **memory**.

b) DONE flag register:

- This information is informed to CPU by means of DONE bit.
- When done flag=1 then DMA controller is ready to receive another command
- When done flag=0 then DMA controller is not ready to receive another command

c**) Interrupt enable(IE) flag register:**

- The DMA controller enables the interrupt enable bit after the completion of DMA operation
- Bit 30 is the Interrupt-enable flag,

  when IE=1: The controller to raise an interrupt

  when IE=0: The controller do not raise an interrupt

d). **Interrupt request (IRQ) flag register:**

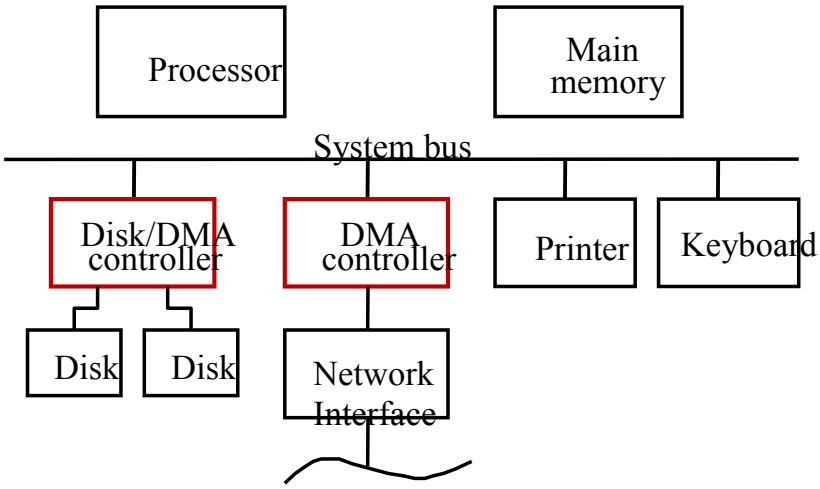
- The DMA controller requests the CPU to transfer new block of data from source to destination by activating this bit.
- Bit 30 is the the **Interrupt request,**

  Controller sets the IRQ bit= 1 when it has requested an interrupt.

  Controller sets the IRQ bit= 0 when it has not requested an interrupt.

ii)**Use of DMA controllers in a computer system:**

1) An example of a computer system is given in below figure, showing how DMA controllers may be used.



2) A DMA controller connects a high-speed network to the computer bus.
3) The disk controller, which controls two disks, also has DMA capability and provides two DMA channels.
4) It can perform two independent DMA operations, as if each disk had its own DMA controller.
5) The registers needed to store the **memory address, the word count,** and so on are duplicated, so that one set can be used with each device.

Working of DMA Controller:

6) To start a DMA transfer of a block of data from the main memory to one of the disks, a program writes the address and word count information into the registers of the corresponding channel of the disk controller.
7) The DMA controller proceeds independently to implement the specified operation.
8) When the DMA transfer is completed,
   This fact is recorded in the status and control register of the DMA channel by setting the Done bit.
9) At the same time, if the IE bit is set that is IE=1, the controller sends an interrupt request to the processor and sets the IRQ bit that is IRQ=1.
10) The status register can also be used to record other information, such as whether the transfer took place correctly or errors occurred.

iii)DMA data transmission modes:

DMA transmits data using 2 modes:

1) cycle stealing

2) Burst mode

1) Cycle stealing:

1) Memory accesses by the processor and the DMA controller are interwoven.
2) Requests by DMA devices for using the bus are always given higher priority than processor requests.
3) Among different DMA devices, top priority is given to high-speed peripherals such as a disk, a high-speed network interface, or a graphics display device.
4) Since the processor originates most memory access cycles, the DMA controller can be said to "steal" memory cycles from the processor.
5) Hence, the interweaving technique is usually called cycle stealing.

2) Burst mode:

1) In this mode,the DMA controller may be given exclusive access to the main memory to transfer a block of data without interruption. This is known as block or burst mode.
2) Most DMA controllers incorporate a data storage buffer. In the case of the network interface

   Example: the DMA controller reads a block of data from the main memory and stores it into its input buffer. This transfer takes place using burst mode at a speed appropriate to the memory and the computer bus.
3) Then, the data in the buffer are transmitted over the network at the speed of the network.

Conflicts in DMA:

- A conflict may arise if both the processor and a DMA controller or two DMA controllers try to use the bus at the same time to access the main memory.
- To resolve these conflicts, an arbitration procedure is implemented on the bus to coordinate the activities of all devices requesting memory transfers.

3.2 Bus Arbitration:-

- Processor and DMA controllers both need to initiate data transfers on the bus and access main memory.
- The device that is allowed to initiate transfers on the bus at any given time is called the **bus master.**

- When the current bus master relinquishes its status as the bus master, another device can acquire this status.

Bus Arbitration: The process by which the next device to become the bus master is selected and bus mastership is transferred to it is called bus arbitration.

Purpose of Bus Arbitration:

- Bus arbitration is required to resolve the conflict that arises when both the Processor and a DMA controller or two DMA controllers try to use the bus at same time to access main memory.
- Bus arbitration is required coordinate the activities of all devices requesting memory transfers.

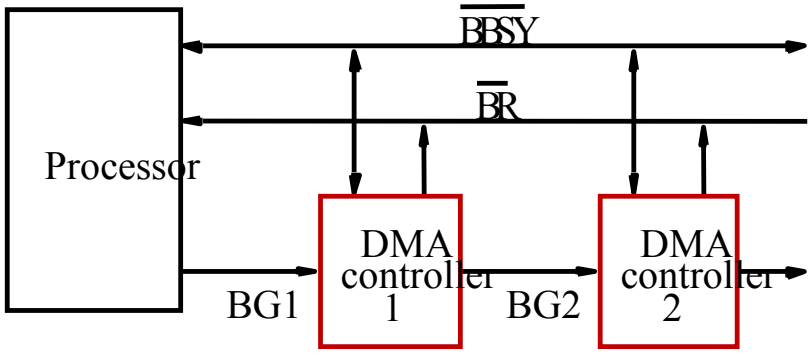
There are two approaches to bus arbitration:

i) Centralized arbitration: a single bus arbiter performs the required arbitration.

ii) Distributed arbitration: all devices participate in the selection of the next bus master.

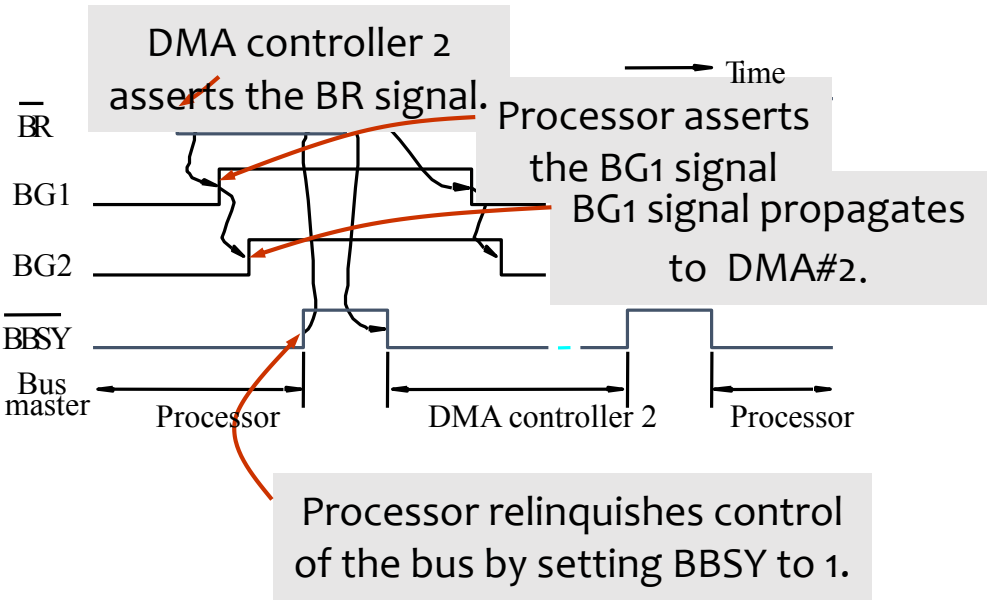
i)Centralized Arbitration:-

1) In centralized arbitration, the bus master may be the processor or a separate unit connected to the bus.
2) Figure shows a basic arrangement in which processor contains the bus arbitration circuit.



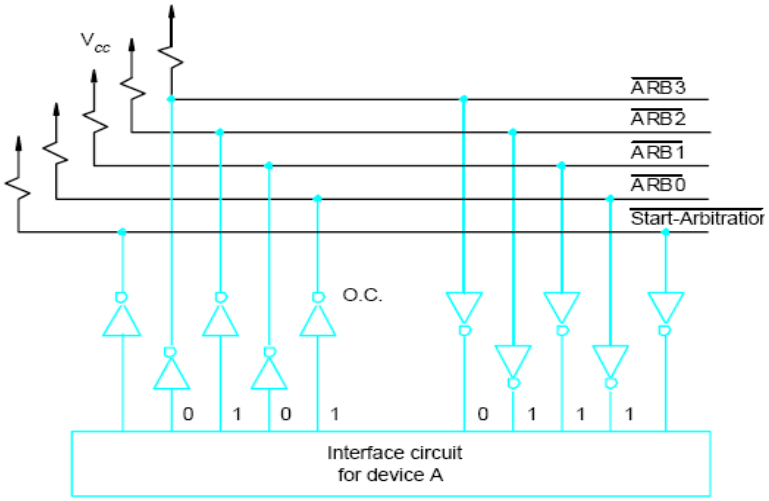
3) In this case, the processor is normally the bus master unless it grants bus mastership to one of the DMA controllers.
4) A DMA controller indicates that it needs to become the bus master by activating the BUS request line, BR.
5) This signal is connected to all DMA controllers using a DAISY-CHAIN arrangement.
6) When the bus request line is activated, the processor activates the bus grant signal, BG1 indicating to the DMA controllers that they may use the bus when it becomes free.
7) This signal is connected to all DMA controllers using a DAISY-CHAIN arrangement.

8) Thus, if DMA controller 1 is requesting the bus, it blocks the propagation of the grant signal to the other devices; otherwise, it passes the grant signal to next device.
9) The current bus master indicates to all devices that it is using bus by activating another line called BUS-BUSY (BBSY).
10) Hence, after receiving the BUS –grant signal, a DMA controller waits for BUS-BUSY to become inactive, then it gets the BUS Mastership. at this time it activates BUS-BUSY.
11) The timing diagram in the figure shows the sequence of events for the devices.



Distributed Arbitration:

1) In distributed arbitration *all devices participate* in the selection of next bus master.
2) A simple method for distributed arbitration is shown in the figure 4.22.



3) Each device on the bus is assigned a *4-bit identification number.*

4) When one or more devices request the bus, they assert the *start arbitration signal* and place their 4-bit identification numbers on four lines, ARB0 through ARb3.

5) A winner is selected as a result of the interaction among the signals transmitted over these lines by all contenders.

6) If one device puts 1 on the bus and another device puts 0 on the same bus line, the bus line status will be 0.

Example:

1) Consider that two devices A and B having ID numbers 5 and 6 respectively are requesting the use of bus.
2) Device A transmits the pattern 0101, and device B transmits the pattern 0110.
3) The code seen by both devices is 0111.
4) If it detects a difference at any bit position, it disables its drivers at that bit position and for all lower-order bits. It does so by placing 0 at the input of these drivers.
5) In our example device A detects the difference on the line ARB1; hence it disables its drivers on lines ARB1 and ARB0. This causes the pattern on the arbitration lines to change to 0110, which means that device B has won the contention.

MODULE-II
I/O Operations(part2)

4.1 Interface Circuits

An I/O interface consists of the circuitry required to connect an I/O device to a computer bus. On one side of the interface, we have bus signals. On the other side, we have a data path with its associated controls to transfer data between the interface and the I/O device – port. We have two types:

Serial port and
Parallel port

A parallel port transfers data in the form of a number of bits (8 or 16) simultaneously to or from the device. A serial port transmits and receives data one bit at a time. Communication with the bus is the same for both formats. The conversion from the parallel to the serial format, and vice versa, takes place inside the interface circuit. In parallel port, the connection between the device and the computer uses a multiple-pin connector and a cable with as many wires. This arrangement is suitable for devices that are physically close to the computer. In serial port, it is much more convenient and cost-effective where longer cables are needed.

Typically, the functions of an I/O interface are:

- Provides a storage buffer for at least one word of data
- Contains status flags that can be accessed by the processor to determine whether the buffer is full or empty
- Contains address-decoding circuitry to determine when it is being addressed by the processor
- Generates the appropriate timing signals required by the bus control scheme
- Performs any format conversion that may be necessary to transfer data between the bus and the I/O device, such as parallel-serial conversion in the case of a serial port

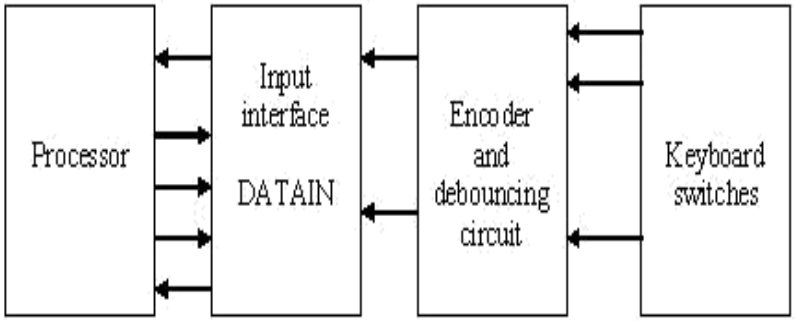
i)Parallel Port

The hardware components needed for connecting a keyboard to a processor Consider the circuit of input interface which encompasses (as shown in below figure):

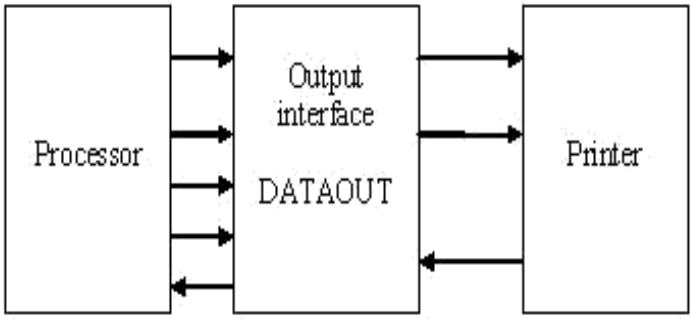
– Status flag, SIN
– R/~W
– Master-ready
– Address decoder

A detailed figure showing the input interface circuit is presented in figure 4.29. Now, consider the circuit for the status flag (figure 4.30). An edge-triggered D flip-flop is used along with read-data and master-ready signals.

Keyboard to processor connection



Printer to processor connection



The hardware components needed for connecting a printer to a processor are: the circuit of output interface, and

– Slave-ready
– R/~W
– Master-ready
– Address decoder
– Handshake control

The input and output interfaces can be combined into a single interface. The general purpose parallel interface circuit that can be configured in a variety of ways. For increased flexibility, the circuit makes it possible for some lines to serve as inputs and some lines to serve as outputs, under program control.

ii)Serial port

A serial interface circuit involves – Chip and register select, Status and control, Output shift register, DATAOUT, DATAIN, Input shift register and Serial input/output – as shown in figure 4.37.

4.2 Standard I/O interfaces

Consider a computer system using different interface standards. Let us look in to Processor bus and Peripheral Component Interconnect (PCI) bus. These two buses are interconnected by a circuit called bridge. It is a bridge between processor bus and PCI bus. An example of a computer system using different interface standards is shown in figure 4.38. The three major standard I/O interfaces discussed here are:

– PCI (Peripheral Component Interconnect)
– SCSI (Small Computer System Interface)
– USB (Universal Serial Bus)

4.2.1 PCI (Peripheral Component Interconnect):

The topics discussed under PCI are: Data Transfer, Use of a PCI bus in a computer system, A read operation on the PCI bus, Device configuration and Other electrical characteristics. Use of a PCI bus in a computer system is shown in figure 4.39 as a representation.

Host, main memory and PCI bridge are connected to disk, printer and Ethernet interface through PCI bus. At any given time, one device is the bus master. It has the right to initiate data transfers by issuing read and write commands. A master is called an initiator in PCI terminology. This is either processor or DMA controller. The addressed device that responds to read and write commands is called a target. A complete transfer operation on the bus, involving an address and a burst of data, is called a transaction. Device configuration is also discussed.

4.2.2 SCSI Bus:

It is a standard bus defined by the American National Standards Institute (ANSI). A controller connected to a SCSI bus is an initiator or a target. The processor sends a command to the SCSI controller, which causes the following sequence of events to take place:

- The SCSI controller contends for control of the bus (initiator).
- When the initiator wins the arbitration process, it selects the target controller and hands over control of the bus to it.
- The target starts an output operation. The initiator sends a command specifying the required read operation.

The target sends a message to the initiator indicating that it will temporarily suspends the connection between them. Then it releases the bus.

- The target controller sends a command to the disk drive to move the read head to the first sector involved in the requested read operation.
- The target transfers the contents of the data buffer to the initiator and then suspends the connection again.

- The target controller sends a command to the disk drive to perform another seek operation.
- As the initiator controller receives the data, it stores them into the main memory using the DMA approach.
- The SCSI controller sends an interrupt to the processor to inform it that the requested operation has been completed.

The bus signals, arbitration, selection, information transfer and reselection are the topics discussed in addition to the above.

4.2.3Universal Serial Bus (USB):

The USB has been designed to meet several key objectives such as:

- Provide a simple, low-cost and easy to use interconnection system that overcomes the difficulties due to the limited number of I/O ports available on a computer

- Accommodate a wide range of data transfer characteristics for I/O devices, including telephone and Internet connections
- Enhance user convenience through a "plug-and-play" mode of operation

a)Port Limitation:

Here to add new ports, a user must open the computer box to gain access to the internal expansion bus and install a new interface card. The user may also need to know how to configure the device and the software. And also it is to make it possible to add many devices to a computer system at any time, without opening the computer box.

b)Device Characteristics:

The kinds of devices that may be connected to a computer cover a wide range of functionality - speed, volume and timing constraints. A variety of simple devices attached to a computer generate data in different asynchronous mode. A signal must be sampled quickly enough to track its highest-frequency components.

c)Plug-and-play:

Whenever a device is introduced, do not turn the computer off/restart to connect/disconnect a device. The system should detect the existence of this new device automatically, identify the appropriate device-driver software and any other facilities needed to service that device, and establish the appropriate addresses and logical connections to enable them to communicate.

ii)USB architecture

To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure. Each node has a device called a hub. Root hub, functions, split bus operations – high speed (HS) and Full/Low speed (F/LS).

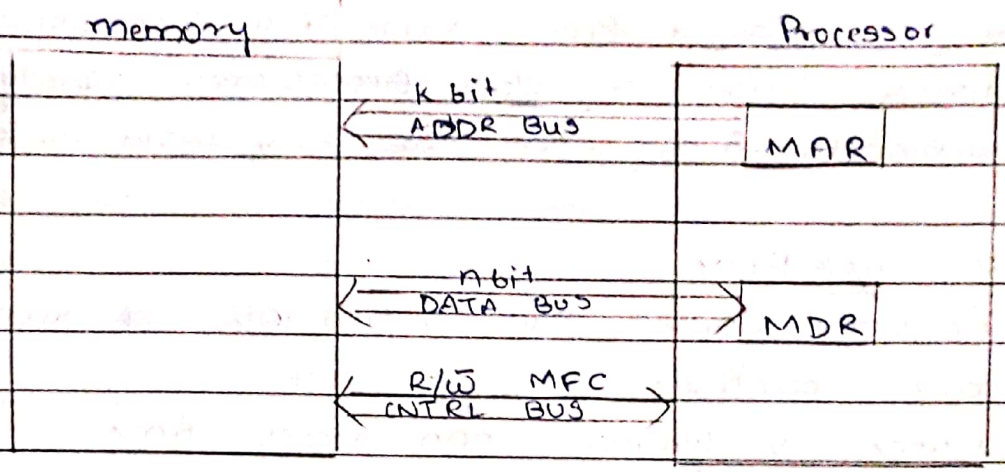
4.8 Concluding remarks

The three basic approaches of I/O transfers are discussed. The simplest technique is programmed I/O, in which the processor performs all the necessary control functions under direct control of program instructions. The second approach is based on the use of interrupts. The third I/O scheme involves DMA, the DMA controller transfers data between an I/O device and the main memory without continuous processor intervention. Access to memory is shared between the DMAQ controller and the processor.

Three popular interconnection standards – PCI, SCSI, USB are discussed. They represent different approaches that meet the needs of various devices and reflect the increasing importance of plug-and-ply features that increase user convenience.

MODULE-3 MEMORY UNIT

Camlin Page
Date / /

① 

* 2^k locations/bytes – k bits. R/W̄ → 1 = read
0 = write.

Memory is made up of millions of semi-conductor cells. Each can hold one bit of information. 8 bits together form one byte. Each byte is given a separate address

* A group of 2 or 4 or 8 bytes form one word and Data transfer happens word by word.

The processor uses 2 special purpose registers, MAR & MDR to perform memory read or write

MAR holds address to be read or written (k bits)

MDR holds data read or to be written (n bits) = 1 word

It uses k bit address bus, n bit data bus and 2 control signals * R/W̄ = 1 – read
= 0 – write

* memory function complete (MFC) 0 initially, MFC = 1 after a memory read or write.

Read

k bit addr → MAR

R/W̄ = 1 , MFC = 0

n bit data → MDR

MFC = 1

write.

k bit addr → MAR

n bit data → MDR

R/W̄ = 0 MFC = 0

[MDR] → (MAR)

MFC = 1

Memory Access time:

The time elapsed from start of a memory operation till its completion is called access time. Usually it is nanoseconds & is same for any word in memory

Memory cycle time:

The time elapsed b/w initialisation of two consecutive memory operations is cycle time

Cycle time is greater than access time

16x8 Memory cell

②

19/17

Memory cells are arranged in the form of 2D array

The cells are connected to the row on word line

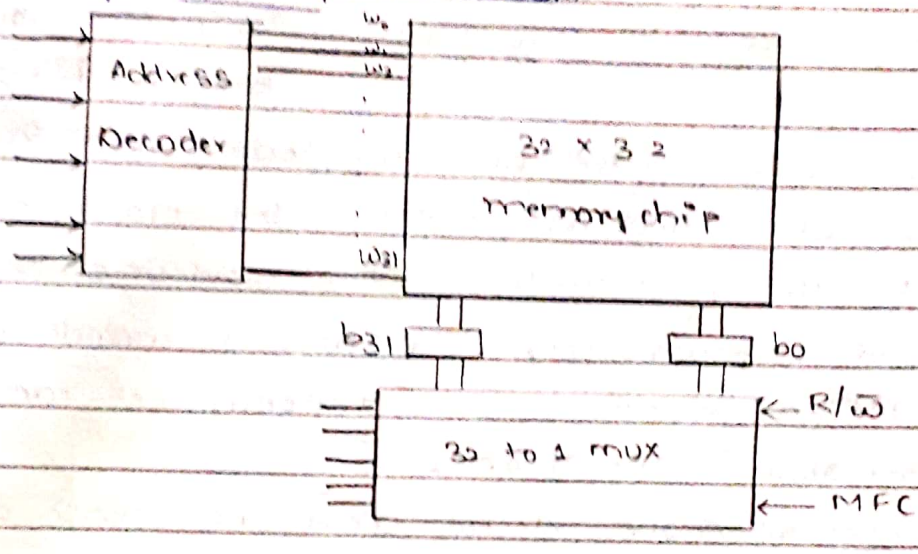
The no. of rows in memory chip indicates no. of words

The cells are connected to bit lines on the column

In turn the bit lines are connected to sense/write circuitary

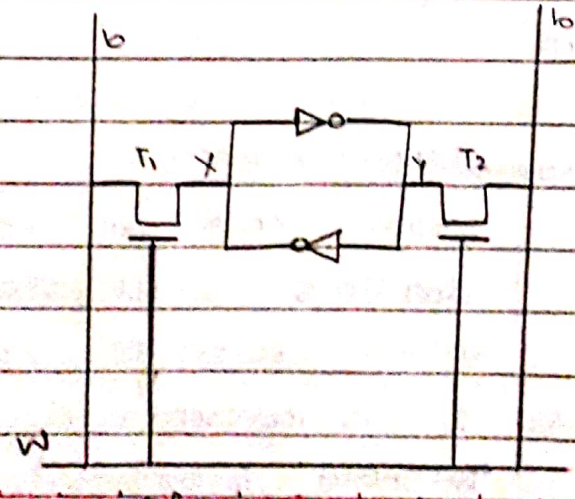
The word lines are connected to address decoder. The read operation is performed by selecting a word based on the address & sensing the contents of all the bits on the word line write operation is performed by copying the data on to the bits of selected word line

③ 1K Memory chip implementation



1K memory chip is implemented by 32 x 32 chip i.e 32 words each having 32 bits. 10 bits of address is used among which 5 bits are used to select one of 32 words & 2 bits is used to select one of 4 bytes in a words Read & write operations are performed using R/$\bar{W}$ & MFC The bit lines are inturn connected to 32 x 1 multiplexer

④ STATIC RAM



Schematic representation

Word line is activated

0

X = 0 Y = 1

T_1 = Off T_2 = On

b = 0 $b' = 1$

1

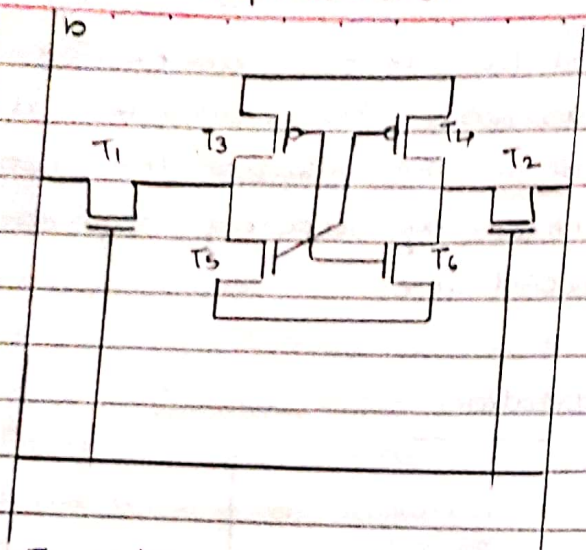
X = 1 Y = 0

T_1 = On T_2 = Off

b = 1 $b' = 0$

Implementation:

(5)



0

T_1 = off T_2 = on

T_3, T_6 = off T_4, T_5 = on

b = 0 b' = 1

1

T_1 = on T_2 = off

T_3, T_6 = on T_4, T_5 = off

b = 1 b' = 0

Two invertors are cross-coupled with 2 transistors T_1 & T_2 which acts as switch. The transistors are connected to bit line & word line. When word line is used (selected), it is connected to powersupply. The unused word lines will be connected to ground. Each staticram cell can hold one bit of information either 0 or 1.

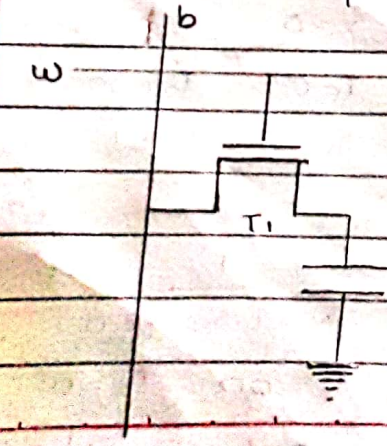
Read operation: The selected word line sense circuit read the bit lines b & b' & decides binary 0 or 1 based on values of b & b'

Write operation: The write circuit will turn on or off T_1 & T_2 to write binary 0 or 1

★ Drawbacks of Static Ram

→ continuous power supply of +5 V is needed to store binary 0 or 1

→ Circuit occupies lot of space hence less bits can be accomodated per chip



ASYNCHRONOUS DYNAMIC RAM

ADRAM cells are made up of capacitors which stores the information in the form of charge. A binary 1 is represented by charge above threshold & binary 0 by charge below equal to ground or below threshold

Camlin | Page
Date / /

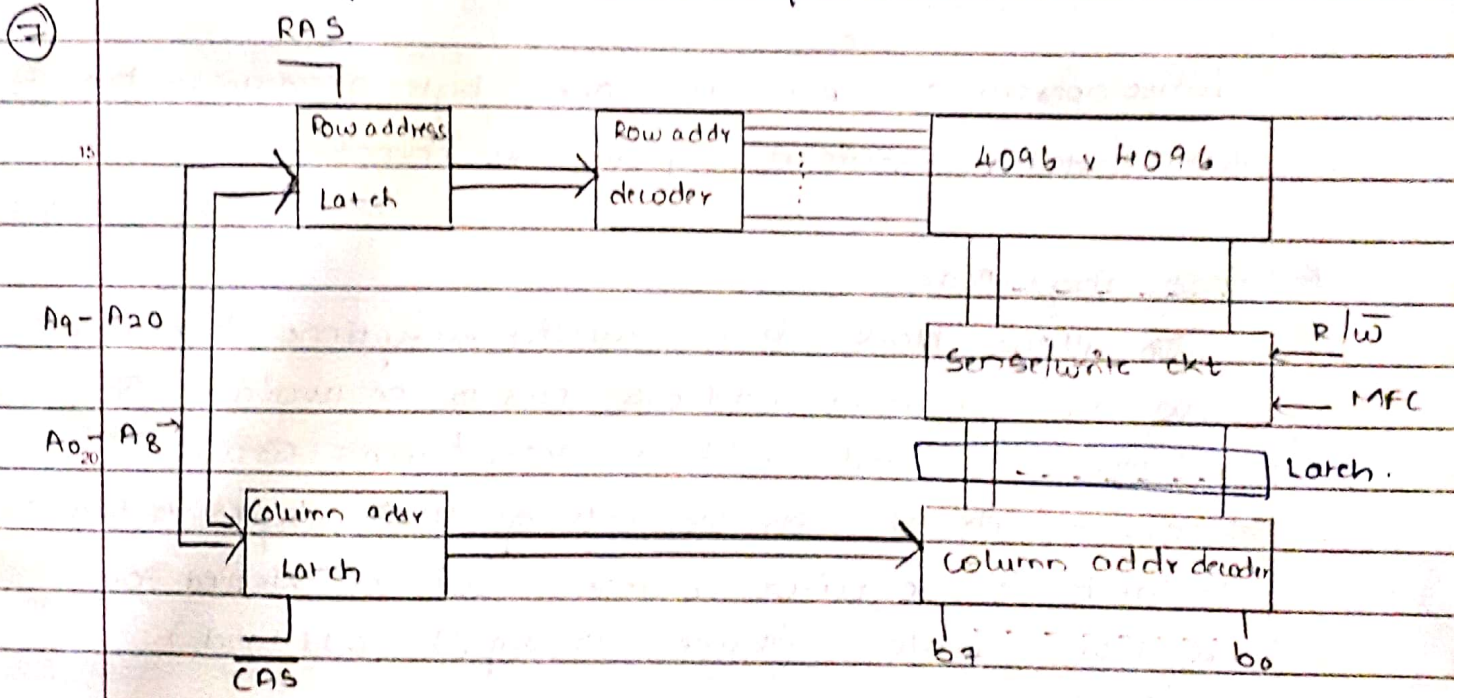
the charge on capacitor will discharge after a few milliSec which has to be periodically refreshed. Hence a separate refreshing unit is used to refresh the cells holding binary 1

Read operation: Select the word line to be read. The sense circuit connected to bit line sense the charge on capacitor. If it is above threshold read as 1 & capacitor is refreshed else it is read as 0

Write operation: Select the word line to be written. To store binary 1 charge the capacitor to full, to store 0 connect capacitor to ground

2M x 8 Asynchronous DRAM chip

(7)



2M x 8 = 16MB

4096 x 512 x 8 = 4096 x 4096 = 16777216 bits

16 x 1024 x 1024 = 16777216 bits

$\therefore \frac{4096}{8}$ = 512 bytes/row

16777216 / 8 = 2097152 bytes = 2^{21} bytes = 21 bits address

2^K - K bits add, Row = 12 bits = 2^{12} = 4096 bits

Column = 9 bits = 2^9 = 512 bytes/row

Camlin Page
Date / /

16MB DRAM is implemented using a memory chip of 4096 x 4096 or 4096 x 512 x 8 i.e it contains 4096 words each word contains 4096 bits or 512 bytes. 21 bits of address is used among which higher order 12 bits is used to select one of 4096 words & lower order 9 bits is used to select one of 512 bytes

Read Operation : 12 bits of row address is applied to row address latch, $R/\bar{W} = 1$. After row is selected RAS signal is issued. 9 bits of address is applied to column address latch. after selecting column address $\overline{CAS}$ signal is issued. One byte of data is transferred to data input register

[RAS = 1, $\overline{CAS}$ = 0]

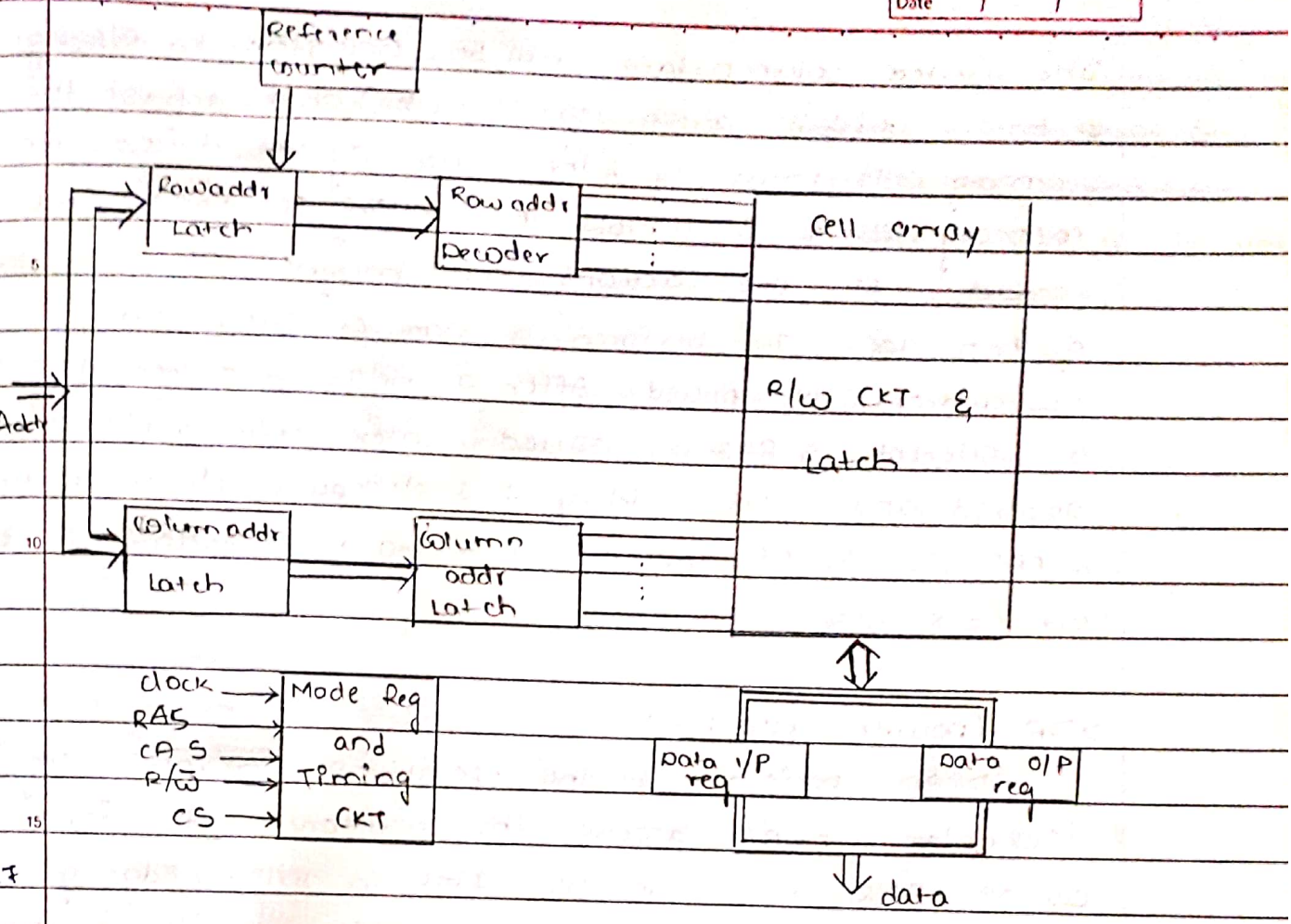
Write Operation : (first all same as read operation) $R/\bar{W} = 0$. One byte of data is transferred from data output register to memory.

* Fast Page Mode

To allow block data transfer every time selecting row and column address has to be avoided. So a latch is added with a sense / write circuit. When a row is selected the contents of all 512 bytes is transferred to latch & required columns will be selected for read or write (same as dig ④ add latch)

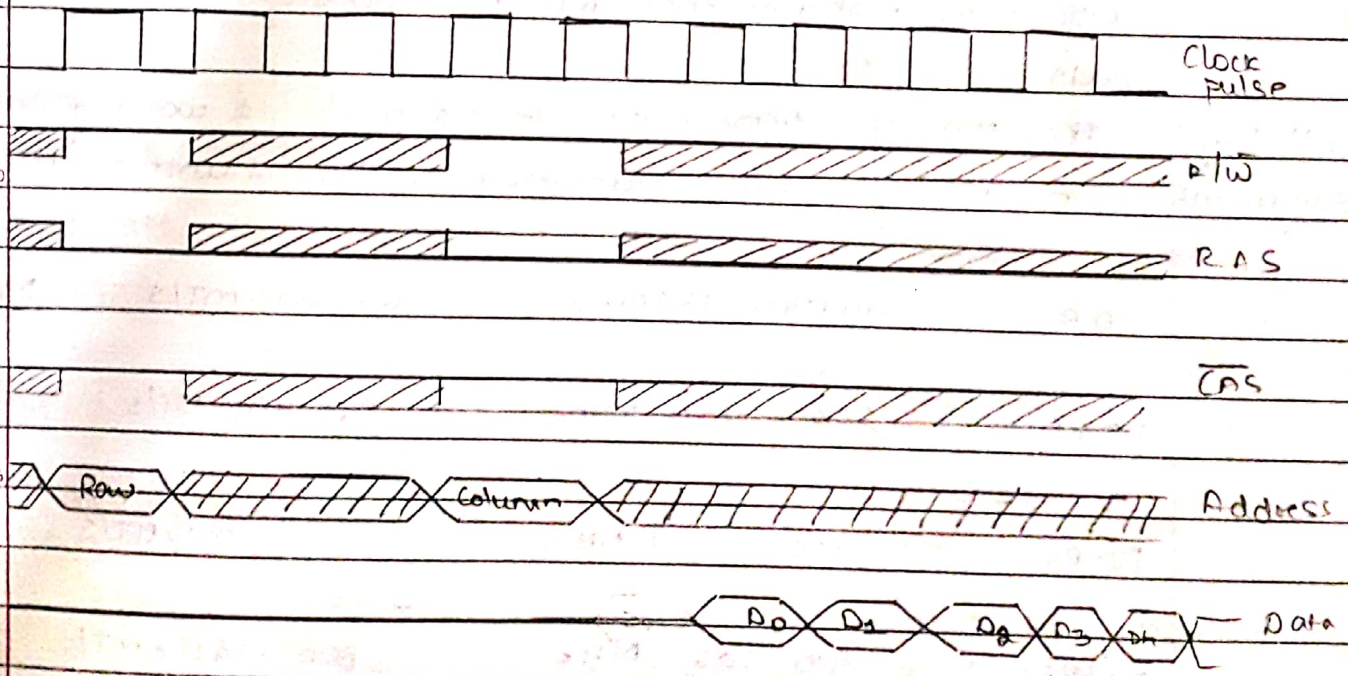
SYNCHRONOUS DRAM

Camlin Page
Date / /



9/9/17

(9)



A DRAM which is synchronised with external clock is called Synchronous D-RAM

SDRAM will have a row address & column address latch. The sense/write circuitary is connected to a latch when a word is selected the entire row contents is transferred to latch

The selected column data will be transfered to O/P register. It has a built in refresh register(or circuitry) which refresh the memory cells periodically. The mode register defines the memory clock & controls the working of SDRAM in different modes. All the actions are triggered at rising edge of the clock. To perform a read operation R/$\bar{W}$=1, Row address is applied. After a delay of 2 clockpulses row is selected & RAS is issued. Later column address is applied and after a delay of 1 clockpulse column is selected & $\overline{CAS}$ is issued. The 1st bit of data is transfered in next clock pulse.

DDR (Double Data Rate)

SDRAM performs all the operations in raising edge of clockpulse. DDR access the memory in the same way as of SDR but transfers 1 bit in raising edge of clockpulse and another bit in trailing edge of clock pulse.

Latency: The amt of time taken to transmit 1 word of data

Bandwidth: The amt of data transfered within 1 unit of time.

SDR	66 MHz - 133 MHz	66-100 MT/s	5V
DDR1	266 - 400 MHz	266-400 MT/s	2.5V
DDR_2	400 - 677 MHz	533-800 MT/s	1.8V
DDR_3	800 - 1066 MHz	800 - 1600 MT/s	1.5V
DDR_4		2133 - 3200 MT/s	1.2V

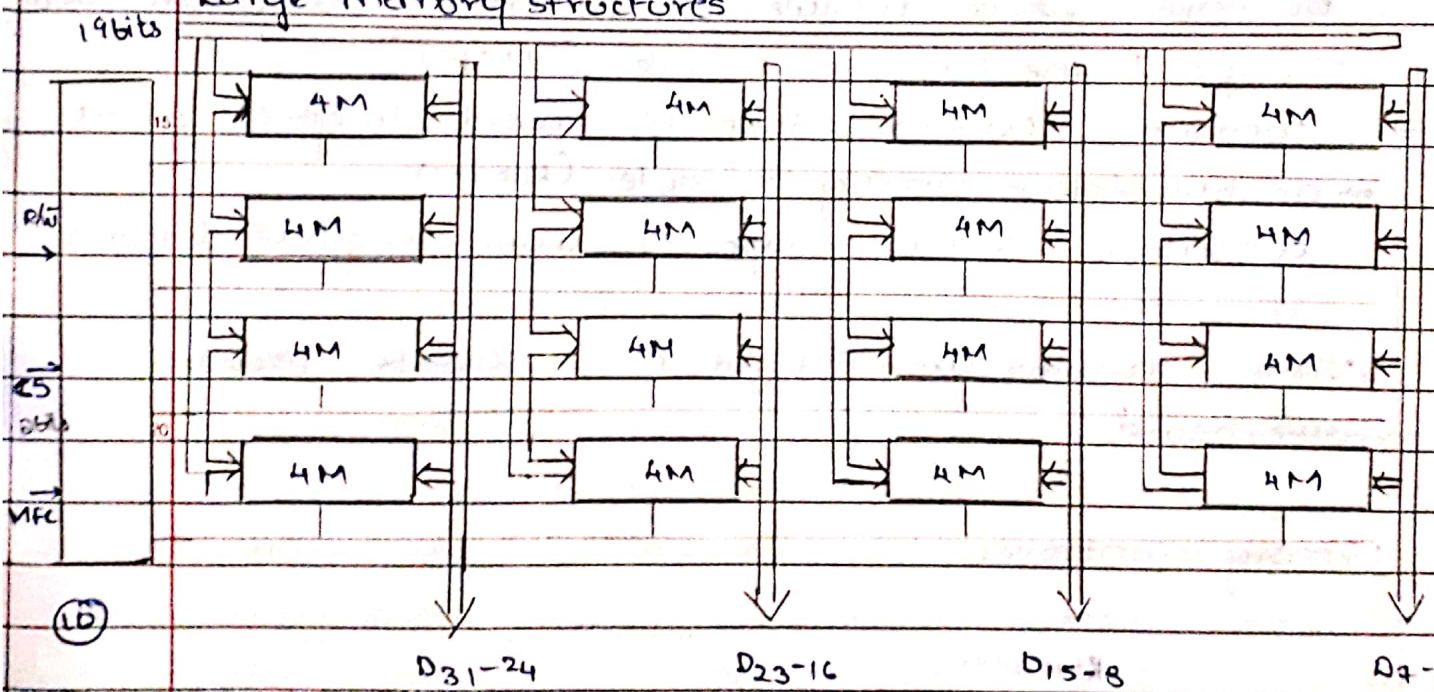
Camlin Page
Date 18/9/17

RAM Bus

The performance of the memory depends on structure of memory & the speed of data lines. So as to increase the speed of data transfer the speed of bus need to be increased which can be done by increasing no. of data lines but it complicates the structure. Hence a narrow bus is used with fast signaling method known as RAM Bus.

In RAM bus instead of sending +5V or 0V to represent binary 1 or 0, a constant voltage of +2V is maintained and 0.3V above the constant voltage is read as binary 1

Large memory structures



4194304/8

→ 4X1024X1024 bits = 4194304 bits = 524288 bytes

524288/1024 = 512 KB

i.e 4Mb chips ⇒ 512kB

→ 2^{19} = 524288 → 19 bits of address to select 1 out of 512 k bytes

64MB of RAM is implemented using 16 chips each of 4Mb. Each 4M chip is made up of 524288 rows x 8 cols or 512K x 8 chip. 21 bits of address is used out of which 2 bits is chip select which selects one out of 4 rows.

19 bits of address is applied to all 4 chips in a row to select one out of 512k rows in each chip. 8 bits of data is transfered from each chip resulting in 32 bit of data transfer from all 4 chips.

Dynamic memory system

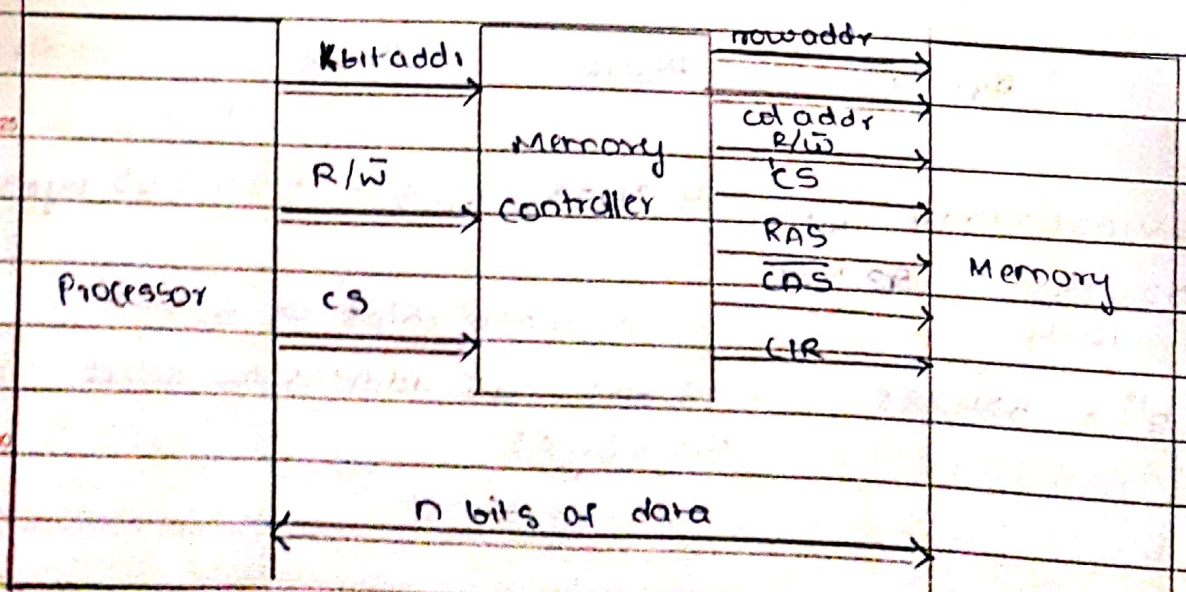
The physical implementation of memory is done in the form of a module. 2 different modules are used

1) Single Inline Memory Module (SIMM)
   contains 100 pins. Supports word length of 32 bit
2) Double Inline Memory module (DIMM)
   contains 168 pins supports word length of 64 bits

This modules are plugged into sockets provided on motherboard.

Memory controller

(11)



The memory controllers acts as interface between processor and memory. The processor sends k bits of address. The controller splits it into raw address & column address and selects appropriate row and column. The controller specifies read or write operation to be performed based on R/$\overline{W}$. It also specifies the chip to be selected based on CS signal. The data transfer happens without intervention of controller.

CACHE MEMORY

The speed of main memory operations is slow compared to processor operation. Major time of instruction execution is spent in memory read and write operations.

So as to improve the performance of computer no. of memory operations should be reduced. Hence a small piece of semiconductor memory is placed on the processor called as cache memory.

Major execution time of program is spent in loops, sub-routine, conditions etc which contains frequently accessed data called as locality of reference.

There are two types of locality of reference

→ temporal : which assumes recently used data will be used soon. So, a copy of every fetched instruction or data is placed on the cache.

→ spitial : It assumes instruction or data which is in close proximity of current instruction or data will be used soon. So, the entire block of instruction or data related to current instruction or data will be placed on cache

21/9/24

The processor requests memory to perform read or write operation. The requested data is 1st searched in cache & if it is found it called as HIT if not it is called as MISS.

→ Read hit: processor requests data for read. It is found on cache & is used.

→ Read miss: processor requests data for read. It is not found on cache. Hence to be brought from main memory

→ write hit: processor requests data for write. It is found on cache & processor writes. Now the data is changed on cache that has to be updated to main memory.

2 possibilities:- a) write through: Any change in the cache will be immediately updated back to main memory

b) write back: The changed value on cache is marked as dirty and will be updated back to main memory when data is removed from cache (not frequently used or execution complete)

→ write miss: Processor requests data for write & it is not found on cache. 2 types of load

a) load through: Bring the requested data from main memory and directly give it to processor without having a copy on cache. The changed value is immediately updated back to main memory.

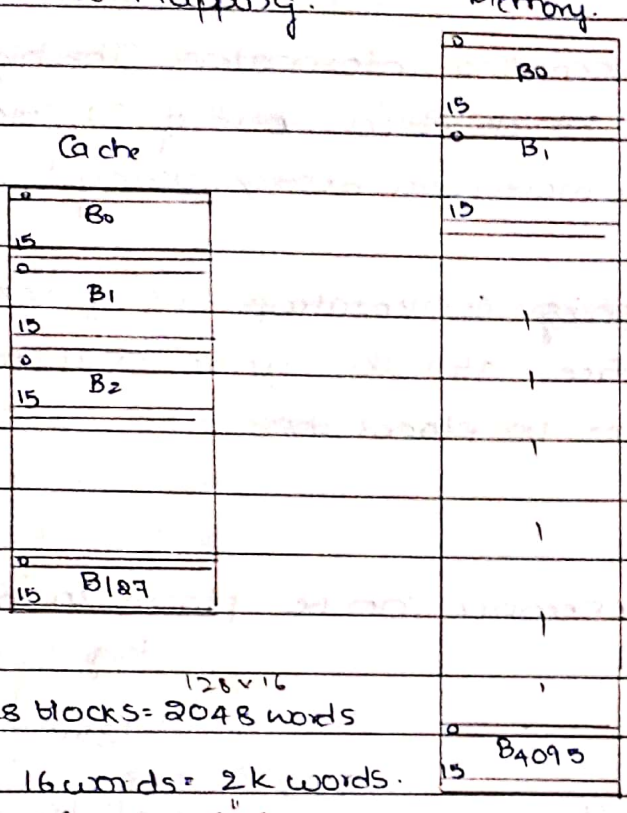
load back: Bring requested data from main memory to cache. The updated data will be marked dirty and it will be updated back to main memory when data is removed from cache

CACHE REPLACEMENT:

During a cache miss the data has to be brought from main memory & if there is no space on cache existing data has to be moved back to main memory

The data chosen for replacement will be least frequently used data by using a counter.

1) Direct Mapping.



128 blocks = 2048 words (128 × 16)

16 words = 2k words. (2048 / 1024)

Mapping — J → any block of main memory; n → no. of blocks in cache

4096 blocks = 65536 words (4096 × 16) = 2^{16} = 16 bits of address

16 words

65536 / 1024 = 64 k words.

0 % 128 = 0	128 % 128 = 0	256 % 128 = 0
1 % 128 = 1	129 % 128 = 1	257 % 128 = 1
2 % 128 = 2	130 % 128 = 2	258 % 128 = 2
		259 % 128 = 3

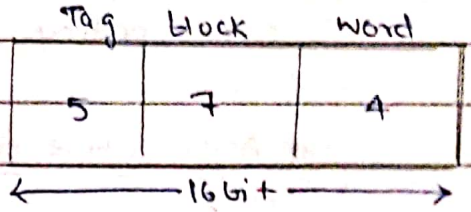
B_0 - 0, 128, 256, 384, 512, ...

B_1 - 1, 129, 257, 385, 513, ...

B_2 - 2, 130, 258, 386, 514, ...

B_3 - 3, 131, 259, 387, 515, ...

In direct mapping J^{th} block of main memory is maped to cache memory is maped to J % n block of cache memory where n is no. of blocks of cache memory. Hence 4096 / 128 = 32 blocks of main memory is mapped to 1 block of cache memory



The placement of main memory block to cache memory block is specified using address which is divided into three parts

→ word: Lower order 4 bits specify which word out of 16 words in a block

→ block: Next 7 bits specify which block of cache out of 128 blocks is used

→ tag: which represents mapping information. The higher order 5 bits specify which block out of 32 blocks reserved for a cache block is currently using it

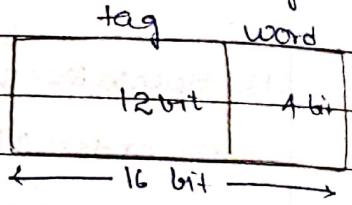
Drawback of direct mapping: contentation i.e if some block of cache memory is free still the ~~un~~ mapped blocks of main memory cannot be placed there

2) Associative mapping

Any block of main memory can be placed in any block of cache memory.

$2^{12} = 4096$ ($B_0 - B_{4095}$)

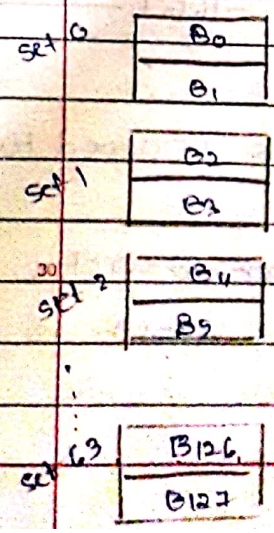
Main memory.

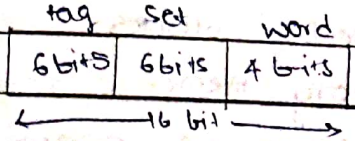


Drawback: *All the blocks of cache memory has to be searched before deciding cache hit or miss.

* If cache is full and processor requests a new block the cache replacement has to be done based on some policy

Set Associative mapping





2 words = 64 sets

0 % 64 = 0
1 % 64 = 1
2 % 64 = 2
3 % 64 = 3
64 % 64 = 0

set-0 = 0, 64, 128, 256, 512.
Set-1 = 1, 65, 129, 257, 513
Set-2 = 2, 66, 130, 260, 514

Date / /

so as to avoid drawbacks of associative mapping combination of associative & direct mapping is used for the set associative mapping.

Cache blocks are grouped together to form set. Usually 2 way or 4 way or 8 way sets. Hence 128 blocks of cache will form 64 sets in 2 way mapping

64 blocks of main memory is mapped to one set of cache memory

Advantages: Lesser search time & easy replacement

NOTE: Direct mapping is One block per set

Associative mapping - All blocks of cache form one set.

Set Associative - specified no. of blocks form one set.

1. Cache
64 blocks
128 words / block
↳ 8192 words = 8K words

memory
4096 blocks
128 words / block.
4096 × 128 = 524288 words = 512K word.
2^{19}

direct

tag	block.	words
6	6	7

← 19 →

0, 128, . . .

$2^{13} > 7$

2 way.

tag	set.	word
7	5	7

← 19 →

$\frac{64}{2} = 32$ $\frac{4096}{32} = 128$

128 blocks M → 1 of Cache

Associative

tag.	word.
12	7

4 way

tag	set	word.
8	4	7

← 19 →

$\frac{64}{4} = 16 = 2^4$ $\frac{4096}{16} = 256$

256 blocks of Mm mapped to 1 of cache

1 MB = 1 × 1024 × 1024

1 MB = 1 × 1024 × 1024 × 8 .

Camlin Page Date 25/9/17

* No of blocks in MM , no. of words/block , no. of bits in addr

2. A computer has main memory of 1MB containing 16 bit/word & it has a cache of 4K words & 64 words per block. Find the address mapping for direct, associative & 2 way set assoc.

cache	main memory.
4K words.	1MB
64 words per block	16 bit / word

MM

1MB = 1 × 1024 × 1024 × 8 = 8388608 = 1048576 bits.

$= 2^{20}$

20 bits

1048576 / 16 = 65536 words.

$= 2^{16}$

16 bits of address.

64 words = 1 block

65536 words ⇒ 1024 blocks. (of main memory)

wrs 16 bit addr 16 bit form one word

cache

4K = 4 × 1026 = 4096 words

64 words / blocks

4096 ⇒ $\frac{4096}{64}$ = 64 blocks. ← how many of the mapped to MM

direct mapping :-

tag	block	words
4	6	6

16

64 words / Block

2^6 = 6

64 blocks = 2^6

1024 B of MM

64 B of Cache

$\frac{1024}{64}$ = 16

Associative :-

tag	word
10	6

any of 1024 blocks can be placed in cache - 2^{10}

2 ways.

tag	set	word
5	5	6

2 blocks → 1 set

$\frac{64}{2}$ = 32 sets.

1024 blocks of MM → 32 sets.

$\frac{1024}{32}$ ⇒ 32 blocks of MM mapped to 1 set

3. Main memory consists of 2048 blocks each of 64 words & cache consists of 32 blocks

cache — MM.

32 blocks — 2048 blocks

words/block — 64 words.

words / block

Direct (interface)

tag	block	word (MM)
6	5	6 bits

17

$2048 \times 64 = 131072$

2^{17}

17 bits

words: 64 → 6 bits

$\frac{2048}{32}$ 32 blocks $\Rightarrow 2^5$

Associative

tag	word
11	6

$2^{11} = 2048$ blocks.

4 way.

4 blocks → 1 set

32 blocks → 8 sets

2048 blocks → 8 sets

256 blocks → 1 set.

2^8 → 8 bit.

tag	set	word
8	3	6

17

2^3

2 way

2 blocks → 1 set

32 blocks → 16 sets

2048 blocks → 16 sets

128 blocks → 1 set

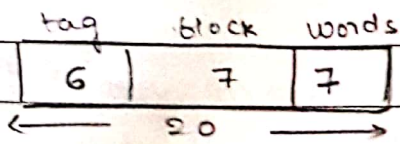
tag	set	word.
7	4	6

17

4. MM of 8192 blocks, cache of 128 blocks each of 128 words.

cache	MM
128 blocks	8192 blocks
	128 words/block

Direct



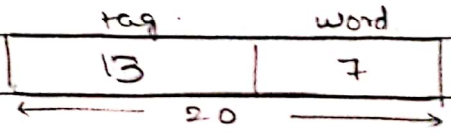
$8192 \times 128 = 1048576 = 2^{20}$

20 bits of add

128 words $\Rightarrow 2^7$ $\Rightarrow$ 7 bits

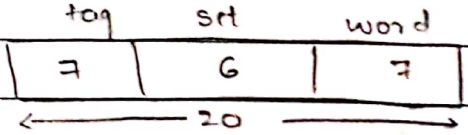
$\frac{8192}{128} = 64 \Rightarrow 2^6 \Rightarrow$ 6 bits

Associative



$2^{13} = 8192$ blocks

2way.



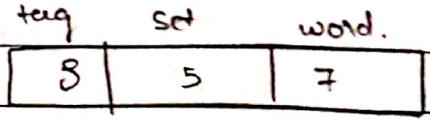
2 words → 1 set

128 words → 64 sets

8192 blocks of mm → 64 sets

$\frac{8192}{64} \Rightarrow$ 128 blocks of mm → 1 set

4way



4 words → 1 set

128 words → 32 sets.

8192 blocks of mm → 32 sets

256 blocks of mm → 1 set

Performance with Cache

When processor requests data & if it is found on cache it is a hit & no. of hits per unit of time is called Hit Rate

When processor requests data & if it is not found on cache it is called miss & no. of miss per unit of time is called miss rate

when a miss occurs, the requested data is brought from main memory ; till then the processor will be kept waiting & this is called miss penalty. The average memory time with cache is given by

$$t_{avg} = h \times c + (1-h) \times M$$

h - hit rate

c - time to access cache

(1-h) - miss rate

M - miss penalty

1. If there is 100 instructions & 30 data items requested for execution of a program. time to access cache is 1 clock cycle. Miss penalty is 17 clock cycles. hit rate for instruction is 95% & for data is 90%. Calculate performance.

inst^n	data
100	30
h - 95%	h - 90%
c - 1 clock cycle	c = 1
1-h = 0.05 (5%)	1-h = 10%
M = 17	M = 17

$(0.95 \times 1 + (0.05)\,17) \times 100 \quad + (0.90 \times 1 + 0.10 \times 17)\,30$

$1.8 \times 100 \quad + 2.6 \times 30$

258 clock cycles

$(100+30) \times 17 = 2210$ clock pulse is without cache

with cache $\frac{2210}{258} = 8.5658$

Camlin Page
Date 9/9/17

If we have 130 inst & 40 data with hit rate of 80% for inst & 85% per data miss penalty is 15 clock cycle & 1 clock cycle is needed to access cache calculate performance. without cache 10 clock cycles is needed.

inst	data
130	40
hit rate - 80%	85%
C = 1	
M = 15	without cache = 10

$t_{avg} = h_1 C_1 + (1-h_1) * M$

$= (0.8 \times 1 + (0.2) \times 15) 130 + (0.85 \times 1 + (0.15) \times 15) 40$

$494 + 124$

618 clock cycles

without cache $(130+40) * 10 = 1700$ clock pulse

Performance $\frac{1700}{618} = 2.75$.

3. If we have 100 instructions & 30 data & C_1 is 1 C_2 is 2, h_1 = 75% for inst 85% for data. h_2 is 20% for inst & 10% for data miss penalty is 15 time to access without cache is 10 clk pulses.

$t_{avg} = h_1 C_1 (1-h_1) h_2 C_2 + (1-h_1)(1-h_2) \times M$

$= (0.75 \times 1 (0.25) 0.2 \times 2 + (0.25)(0.80) 15) 100 +$
$(0.85 \times 1 (0.15) 0.1 \times 2 + (0.15)(0.90) 15) 30$

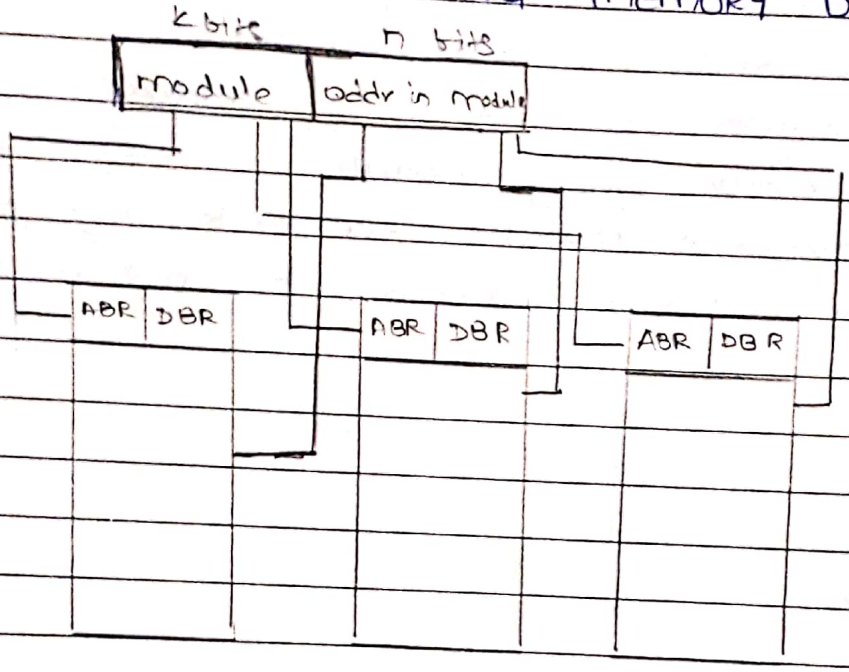
369.075

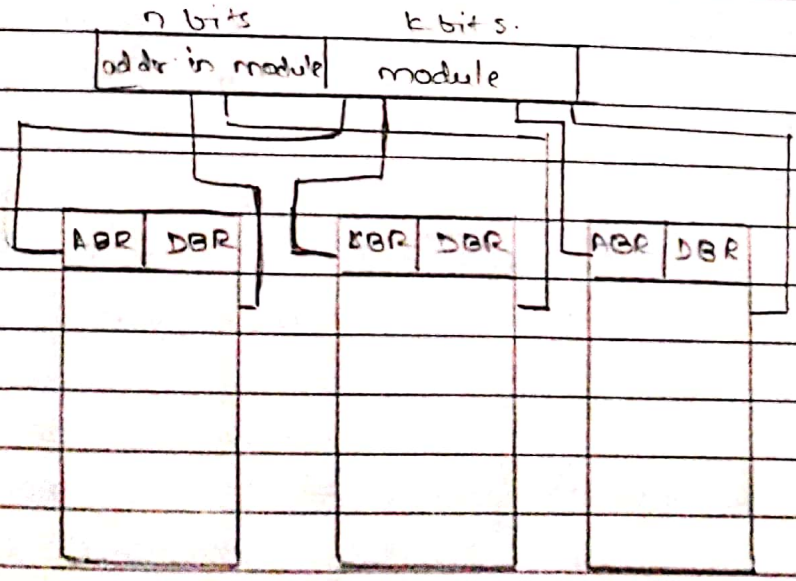
without cache = (100+30) 10

= 1300

performance = $\frac{1300}{369.015}$ = 3.523

INTERLEAVING MEMORY OPERATION.





Main requirement in memory hierarchy is to create fast access to large sized memories. This could be achieved by using parallelism through interleaving memory operations.

Main memory is implemented by a set of modules with address buffer register (ABR) & data buffer register (DBR)
Two methods of accessing modules.
→ Fig (12): lower order n bits specify the addressing module higher order k bits specify the module
If data need to be accessed from consequtive modules the entire address has to be applied
→ Fig (13): Higher order n bits specify addressing module, lower order k bits specify module
So, if consequtive modules has to be accessed, only lower order module address has to be changed.

Non-volatile memory

Both S-RAM and D-RAM are volatile so ROMs are used to store the information permanently.

Different Type of ROMs are: ROM, PROM, EPROM and EEPROM.

Read-only memory (ROM) is an integrated circuit programmed with specific data when it is manufactured.

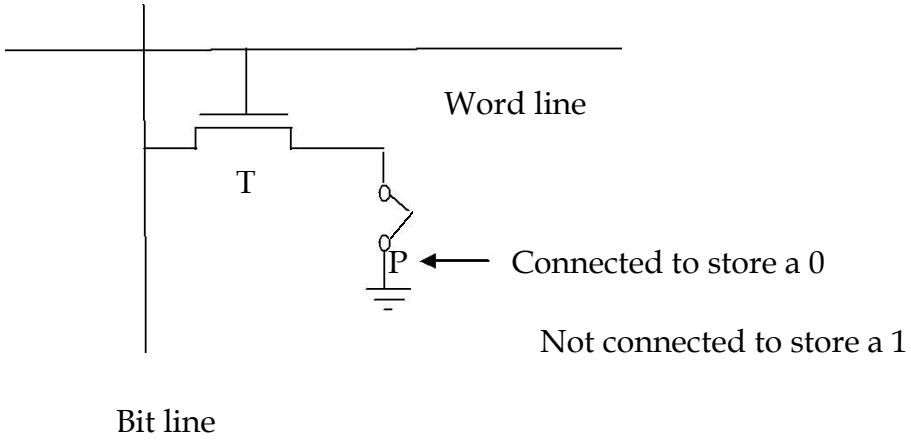


Fig: ROM Cell

ROM is made up of transistor and a switch P. If P is connected to GND it stores binary 0 and if it is not connected it stores binary 1. Bit lines are connected to power supply using resistors. To read the information the word line is activated and transistors is closed & it will read binary 0 or 1 based on P connected to GND or not.

Programmable Read-only memory (PROM)

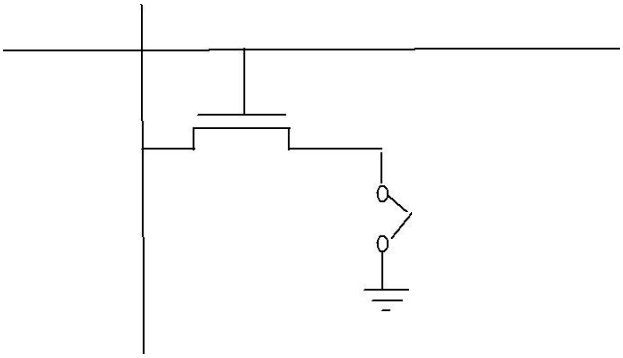


Fig: PROM Cell

Programmable Read-only memory (PROM) is an integrated non-volatile memory circuit that is manufactured to be empty.

It can be later programmed with specific data. The programming can be done only once.

After programming this data is always stored to this IC.

PROM chips contain a fuse at the switch P so all bits contain binary 0. While writing user will pass current to burn out the fuse where ever required to store binary 1. This process is known as burning the PROM.

Erasable programmable read-only memory (EPROM)

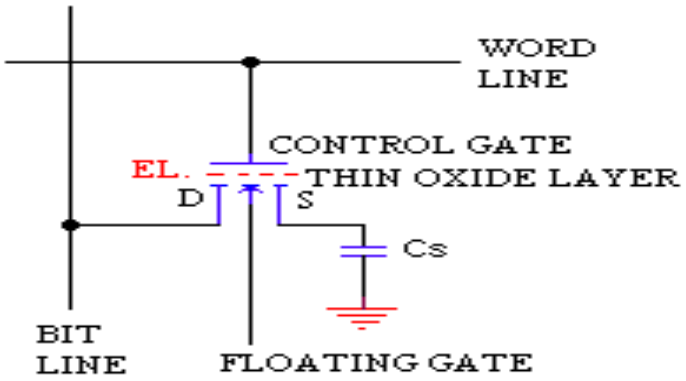


Fig: EPROM Cell

Erasable programmable read-only memory (EPROM) chips work as PROM chips, but they can be rewritten many times.

In an EPROM, the cell at each intersection has two transistors. The two transistors are separated from each other by a thin oxide layer.

One of the transistors is known as the floating gate and the other as the control gate. The floating gate is connected to the row (word line) through the control gate.

When floating gate and control gate is connected the cell has a value of 1. To change the value to 0 requires altering the placement of electrons in the floating gate.

An electrical charge, usually 10 to 13 volts, is applied to the floating gate to charge the floating gate and thus turn bit to 0.

Erasing an EPROM is done by exposing it to ultraviolet (UV) light (253.7 nm wavelength) it will erase the entire EPROM.

Electrically erasable programmable read-only memory (EEPROM

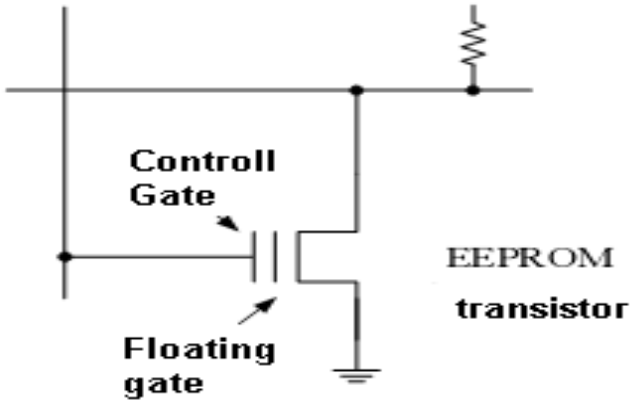


Fig: EEPROM Cell

EEPROM chips that can be electrically programmed and erased.
EEPROMs are organized as arrays of floating-gate transistors. EEPROMs can be programmed and erased in-circuit, by applying special programming signals.
Originally, EEPROMs were limited to single byte operations which made them slower, but modern EEPROMs allow multi-byte page operations.

Flash memory

Flash memory is a type of EEPROM it has in-built circuit to erase by applying an electrical field to the entire chip or blocks.
Flash memory works much faster than traditional EEPROMs because it writes data in chunks, usually 512 bytes in size, instead of 1 byte at a time.
Flash memories are implemented as Flash cards, removable solid-state storage devices (SSD) and flash drives.

Memory Hierarchy/Speed, Size and Cost

All different memory/storage devices used in computer is organized in hierarchy. The processor contain limited number of registers to store operands and intermediate results and L1 cache to store frequently used information. Register and cache are the D-RAM cells on processor chip which are high speed but limited in number/size.
On the mother board we have secondary L2 cache which is slow but larger compared to L1 cache & registers.
In the next level of hierarchy there will be main memory made up of D-RAM cells slow but larger L1 cache & registers which stores the program and data while execution by processor.

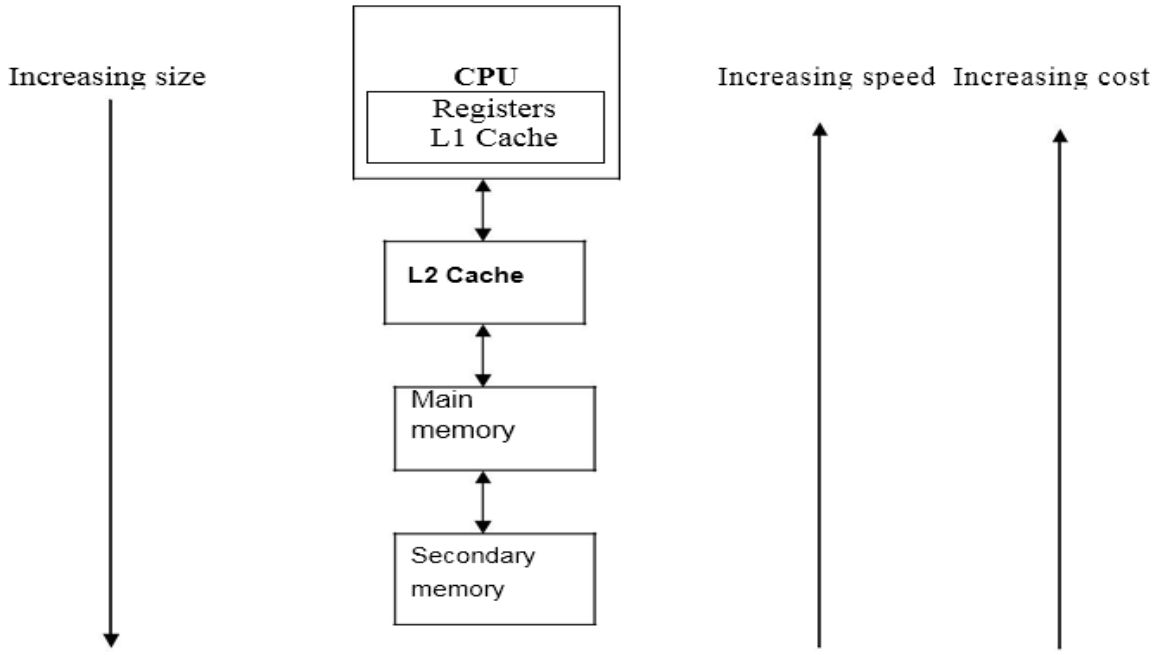


Fig: Memory Hierarchy

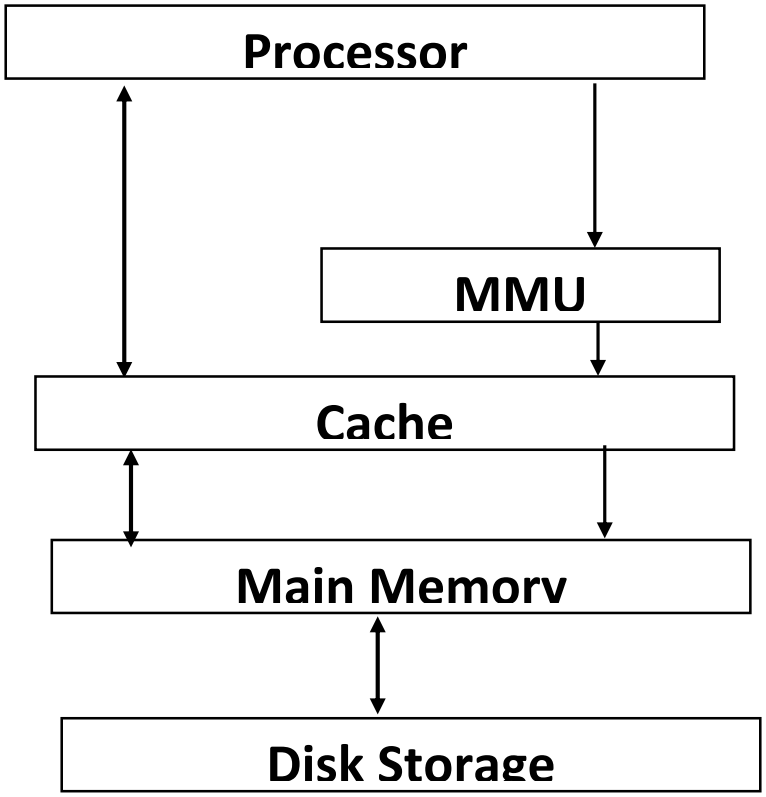
The large secondary storage devices either magnetic or optical devices form the next level of hierarchy which are slower and quite larger than main memory.

Comparison:

The speed & cost increases as we go up in the hierarchy and size increases as we go down in the hierarchy.

Virtual Memory

- The main memory of a computer ranges from 100MB to 8GB, if some program of larger size than the size of main memory need to be executed then entire program is stored in secondary storage devices (HDD) and only those part/block of the program which is currently needed for execution by the processor is placed in the main memory.
- When a new block or next part of the program is needed by the processor the existing block is moved back to HDD and the new block is brought into main memory.
- Operating system is responsible for moving the program blocks between main memory and HDD this technique is called as Virtual Memory.
- The binary address given by the processor is called virtual address.
- The MMU (Memory Management Unit) translates virtual address to physical address and search the data in cache memory /main memory if present will be accessed, if not found then data transfer is done from HDD through DMA (Direct Memory Access).



Address Translation

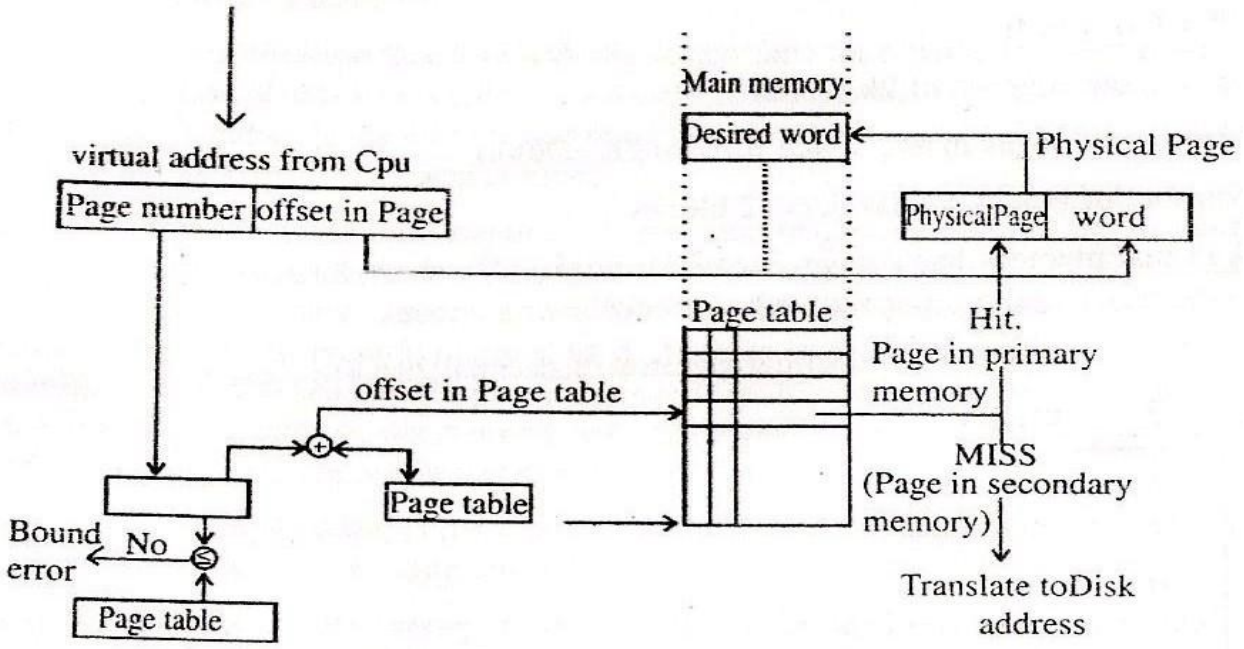


Fig. 12. Virtuel address translation in a paged MMU.

- All programs & data in the disk storage is divided into fixed length units called page, page size range from 2KB to 16KB.
- The processor generates the virtual address requesting for instructions or data.
- Virtual address consists of virtual page number (higher order bits) followed by offset (lower order bits) offset specify the location of the word in each page.
- The allocated memory for a process is divided number of blocks (page frames) such that page size =page frame size.
- Information about which page is residing in which page frame is kept in page table.
- The starting address of the page table is kept in a page table register.
- Virtual page number + page table base register=page frame address
- The page table is frequently accessed by MMU so ideally it should be stored in MMU as the size of page table is large it is kept in main memory so as to reduce the time for accessing a small portion of the page table is placed on the buffer on MMU called as Translation Look-ahead Buffer(TLB).

Address Translation proceeds as follows:

- Processor gives virtual address- MMU looks in TLB if entry is found returns the physical address of the page. If the page entry is not found in TLB then the page table on main memory is searched and the physical address of the page is returned if found.
- If the requested page is not in main memory it is a page fault then MMU requests OS to bring the required page from HDD.
- OS will suspend the execution of the current process until the desired page is brought.
- If the desired page is brought from HDD and main memory is full then an existing page has to be replaced, so least recently used pages is chosen for replacement

Hard disk Drive organization

- The HDD is the stack of rotating platters that coated with magnetic coating. There are read/write heads on the top and bottom of each platter, so information can be recorded on both surfaces. All heads move together across the platters. The platters rotate at constant speed usually 3600 rpm.
- The disk drive electronics are located on a printed circuit board attached to the disk drive.
- Data is organized on the disk platters by tracks and sectors.

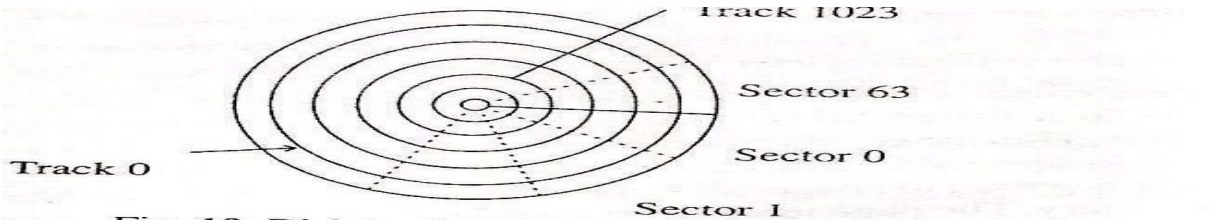


Fig. 13 Disk track and sector organization.

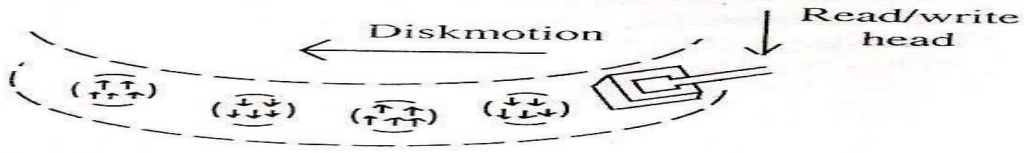


Fig. 14 Individual bits encoded on a disk track.

- The number of bytes per sector is fixed for a given disk drive, varying in size from 512 bytes to 2KB. All tracks with the same number, but as different surfaces, form a cylinder.
- The information is recorded on the disk surface 1 bit at a time by magnetizing a small area on the track with the write head. That bit is detected by sending the direction of that magnetization as the magnetized area passes under the read head.
- The 12 bytes of ECC (Error Correcting Code) information are used to detect and correct errors in the 512 byte data field.
- The operating system specifies the track, sector and surface of the desired block. The disk controller translates that requests to a series of low level disk operations.
- Seek time: - Is the average time required to move the read/write head to the desired track.
- Access time: - Is the time required to move the head from one track to adjoining one.
- Rotational latency: - Is the average time required for the needed sector to pass under head once and head has been positioned once at the correct track.
- Average Access time: - Is equal to seek time plus rotational latency.
- Burst rate: - Is the maximum rate at which the drive produces or accepts data once the head reaches the desired sector,

Optical Disks

Compact Disk (CD) Technology:- CD system is based on laser light source. A laser beam is directed onto the surface of the spinning disk. Physical indentations in the surface are arranged along the tracks of the disk. They reflect the focused beam towards a photo detector, which detects the stored binary patterns.

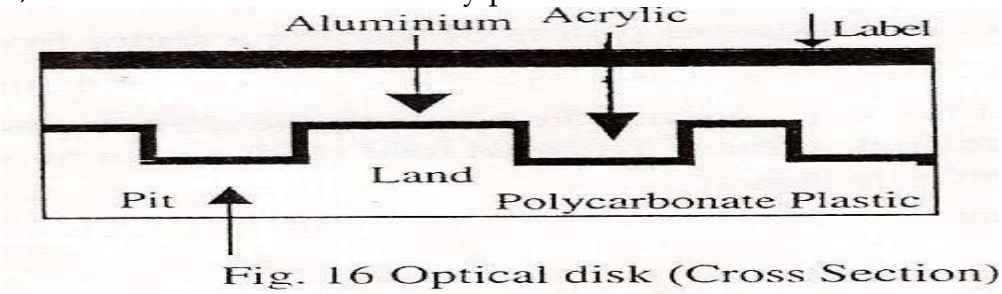
- 

Fig. 16 Optical disk (Cross Section)

- A cross section of CD shows the bottom payer is Polycarbonate plastic, which functions as a clear glass base. The surface of this plastic is Programmed to store data by indenting it with pits. The un-indented parts are called lands. A thin layer of reflecting aluminum material is placed on top of a programmed disk. The aluminum is then covered by a protective acrylic. Finally the topmost layer is deposited and stamped with a label.
- The laser source and the Photo detector are positioned below the polycarbonate plastic. The emitted bean travels through this plastic, reflects off the aluminum layer and travels back toward photo detector
- Some important optical disks are listed below
    1. CD-ROM
    2. CD-RWs (CD-re writable)
    3. DVD technology (Digital Versatile disk)

Cache Replacement Algorithm

In direct mapping the position of mapping each block of main memory to cache memory is fixed but in associative and set associative mapping if cache blocks are full the cache controller should decide which of the blocks on the cache to replace for bringing in the new block.

The objective of the replacement is to restore the block which are frequently used and replace the least frequently used blocks.

To track least frequently used blocks cache controller uses a counter:

When new block is brought into cache the counter is made equal to 0 and all other block's counter is incremented by 1. When an existing bock is referred again then the counter is set to 0 and all other block's counter is incremented by 1.

So the frequently used blocks will have least counter value than the others and the block with high counter value is chosen for replacement.

Write Buffer with cache

If write through protocol is used the cache block which are written has to be updated immediately to main memory, which consumes lot of write operations and processor should wait.

So as to improve the performance "Write Buffer" is used, the processor places all the write requests to the write buffer and continues with next instruction, these write requests in the write buffer are updated to main memory whenever the read request is not answered.

During read request the contents is first checked in the write buffer as it is the updated value.

When write back protocol is used if processor requests a new block and cache is full then the dirty data of the block chosen for replacement has to be updated to main memory during which processor has to wait. Now the dirty contents are moved to the write buffer and the new block is brought immediately to the cache and the contents of the write buffer is updated later.

Pre-fetching with cache

When a read miss occurs the requested data has to be brought from main memory during which the processor has to wait (miss penalty).

So to avoid this pre-fetching of the data before the request is done by using special instructions, the pre-fetch instructions are placed into the program by compiler.

The drawback is unnecessary data may be pre-fetched and if cache is full the pre-fetched block may be replaced before it is used.

Lock up free cache

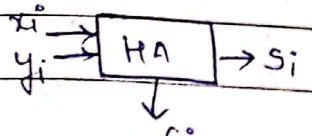
When read miss occurs cache gets locked until the requested data is brought from the main memory.

So Lockup free cache is used which allows processor to access cache during miss. This supports multiple outstanding misses.

Module - 4. Arithmetic Unit

Camlin | Page
Date 11/10/17

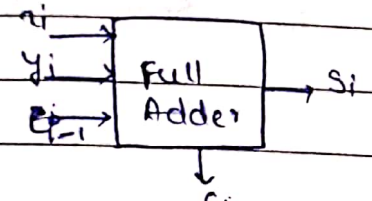
Half adder



x_i	y_i	S_i	C_i
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half adders are used to add 2-bit binary no.s & produce sum & carry

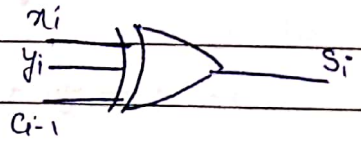
Full adder



x_i	y_i	C_{i-1}	S_i	C_i
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

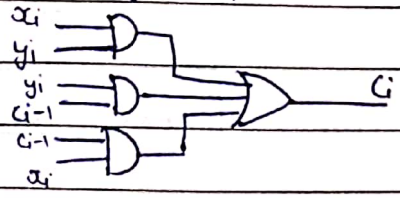
$$S_i = \bar{x_i} \bar{y_i} C_{i-1} + \bar{x_i} y_i \bar{C_{i-1}} + x_i \bar{y_i} \bar{C_{i-1}} + x_i y_i C_{i-1}$$

$$= x_i \oplus y_i \oplus C_{i-1}$$



$$C_i = \bar{x_i} y_i C_{i-1} + x_i \bar{y_i} C_{i-1} + x_i y_i \bar{C_{i-1}} + x_i y_i C_{i-1}$$

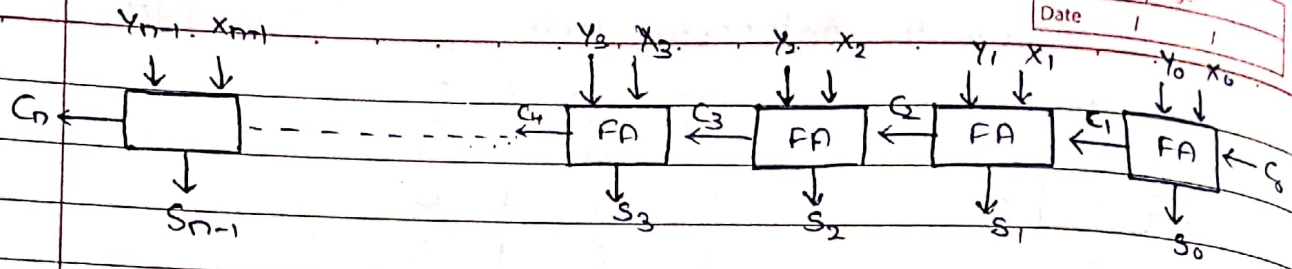
$$= x_i y_i + y_i C_{i-1} + C_{i-1} x_i$$



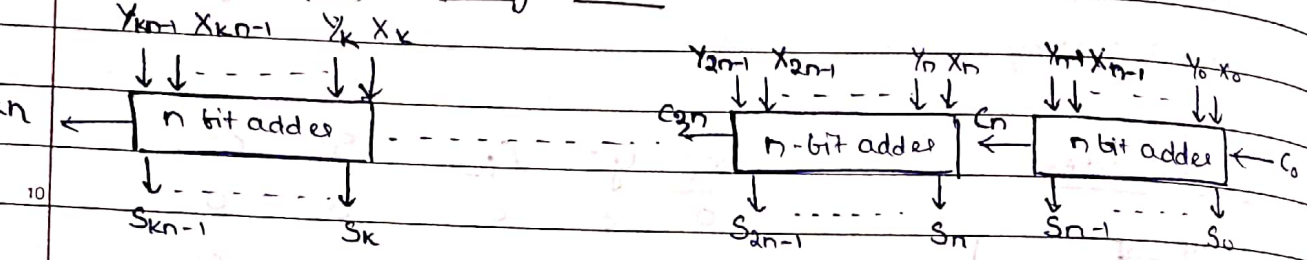
It uses a bit binary number & a carry (from prev stage) & produces sum & carry.

n-bit ripple carry adder: So as to perform n-bit addition n full adders are connected such that the carry from one full adder will propagate to the next.

roy.



k - n bit ripple carry adder.



12/10/17

Fast adder or Carry LookAhead Adder

The fast adder speed up the generation of carry. This method uses argumented logic gates to look at the operands & determine the carry from the previous stage to current stage. It uses two functions i.e G_i & P_i

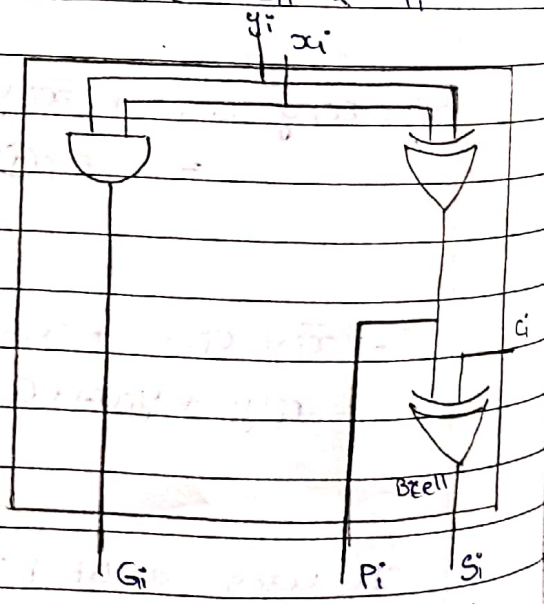
G_i : carry generator

P_i = carry propogator

$$C_{i+1} = x_i y_i + y_i c_i + x_i c_i$$

$$C_{i+1} = \underbrace{x_i y_i}_{G_i} + c_i \underbrace{(x_i + y_i)}_{P_i}$$

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



The carry at current stage is high i.e $C_{i+1} = 1$ in two cases.

case 1: when $G_i = 1$ i.e both $x_i = 1$ & $y_i = 1$

case 2: when $C_i P_i = 1$ i.e $C_i = 1$ and either $x_i = 1$ or $y_i = 1$

This is implemented by using Bit stage cell or B-cell

4-bit adder

$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + C_0P_0P_1$

$C_3 = G_2 + P_2C_2 = G_2 + P_2[G_1 + P_1G_0 + C_0P_0P_1] = G_2 + P_2G_1 + P_2P_1G_0 + C_0P_0P_1P_2$

$C_4 = G_3 + P_3C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + C_0P_0P_1P_2P_3$

Multiplication of unsigned nos

multiplicand ↓ 10 × 3 ↓ Multiplier

M Q

M × Q

13 × 12 = 156.

M Q

1101 × 1100

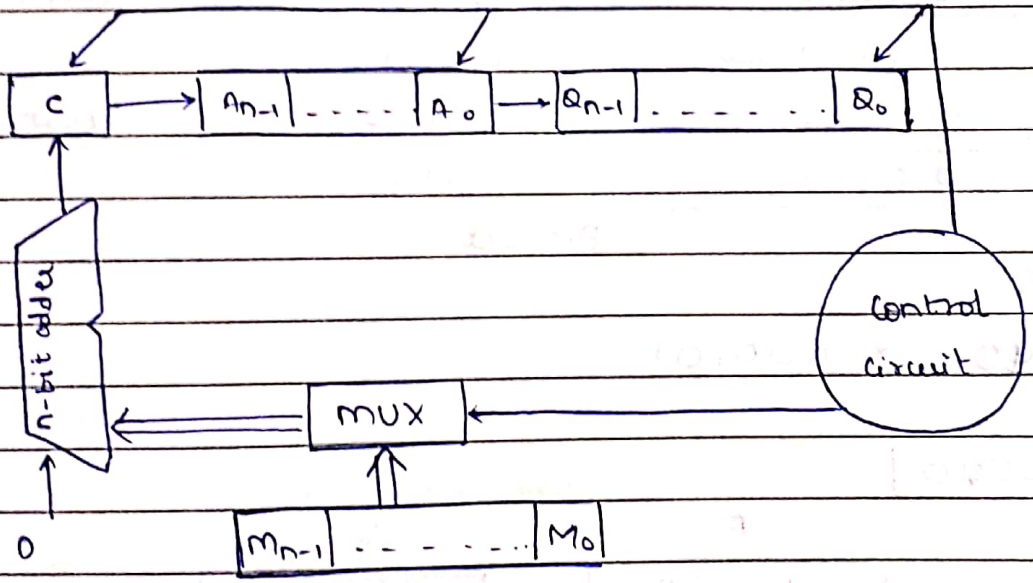
0000

0000

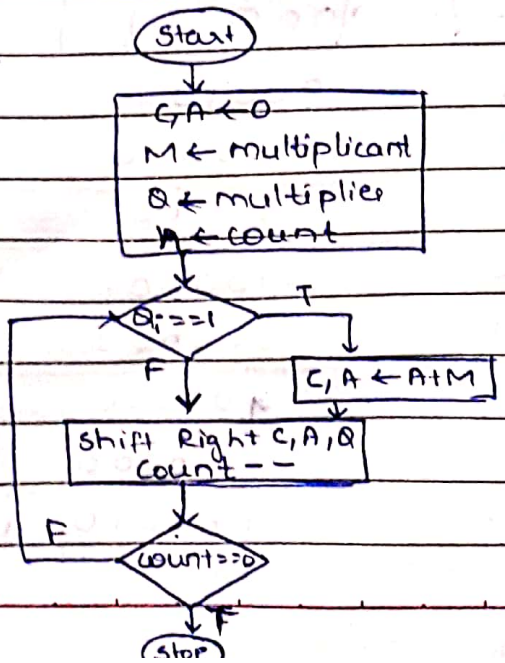
1101

1101

10011100



Algorithm



17 x 15

10001 x 01111

M

10001

C	A	Q	
0 →	00000	→ 01111	count = 5

Add	0	10001	01111	
Shift	0	01000	10111	count = 4
Add	0	11001	10111	
Shift	0	01100	11011	count = 3
Add	0	11101	11011	
Shift	0	01110	11101	count = 2
Add.	0	11111	11101	
Shift	0	01111	11110	count = 1
Shift	0	00111	11111	

Product.

34 x 5

100010 x 000101

M

100010

C	A	Q	
0 →	000000	→ 000101	count = 6

Add.	0	100010	000101	
Shift	0	010001	000010	count = 5
Shift	0	001000	100001	count = 4
Add	0	101010	100001	
Shift	0	010101	010000	count = 3
Shift	0	001010	001000	count = 2
Shift	0	000101	010100	count = 1
Shift	0	000010	101010	Product.

Camlin Page
Date / /

8421

Q X_{12}

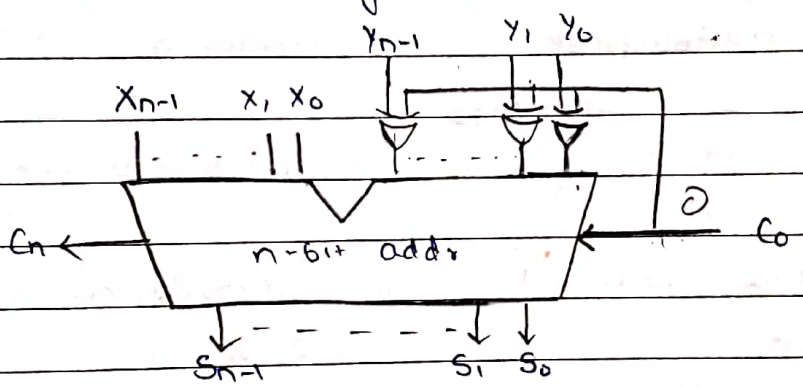
1001 X 1100

M

1001

	C	A	Q	
	0	0000	1100	count = 4.
Shift	0	0000	0110	count = 3.
Shift	0	0000	0011	count = 2
Add	0	1001	0011	
Shift	0	0100	1001	count = 1
Add.	0	1101	1001	
Shift.	0	0110	1100	Product.

✓ Addition / Subtraction Logic unit:



G/D		
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

n-bit adder circuit can be used to perform 2's complement addition & also subtraction.

→ During add operation [X+Y] control $C_0 = 0$ so Y is not complemented hence n-bit adder performs X+Y.

→ During subtraction [X-Y] $C_0 = 1$ so Y is complemented and 1 is added to LSB of complemented Y to obtain 2's complement. So performs X-Y.

-32 16 8 4 2 1
1 1 0 0 1 0

Camlin Page
Date 26/10 17

Multiplication of signed no.s.

Booth's Algorithm.

It is a powerful algorithm for signed multiplication.
It works by recoding the multiplier.
← right to left

Recoded Bit		B_i	B_{i-1}
0	←	0	0
+1	←	0	1
-1	←	1	0
0	←	1	1

Consider from right to left ie LSB to MSB.

Steps in Booth's algorithm.

1. Add 0 at MSB for both multiplier and multiplicand
2. Compute 2's complement of both multiplier and multiplicand separately.
3. If the multiplicand or multiplier is positive then use BCD
4. If either multiplier or multiplicand is negative then use 2's complement.
5. Recode the multiplier by considering an additional 0 at LSB.
6. If Q_i is 0 then substitute zero's in partial product and extend 0's for rest of bits
7. If Q_i is 1 substitute multiplicand and sign extend with MSB
8. If Q_i is → substitute 2's complement of multiplicand and extend by MSB.

Ex: 10 x -5 = -50.
M Q.

1) Add 0 in MSB.

	10		5
	01010		00101
1's	10101	1's	11010
+1	10110	+1	11011
	= -10		= -5

1+1+1+1 =

Camlin | Page
Date / /

10 x -5

01010 x 11011 [0]

01010 x 0-1+10-1

```
  1 1 1 0 1
  1 1 1 1 1 1 0 1 1 0
  0 0 0 0 0 0 0 0 0 0
  0 0 0 0 1 0 1 0
  1 1 1 0 1 1 0
  0 0 0 0 0 0 0
  1 1 1 1 1 0 0 1 1 1 0
```

neglect carry [1]

```
  1 0
  1
  1
  1
  1
  0
```

-32 x 8.

32	8
0100000	0001000
1's 1011111	1's 1110111
+ 1	+ 1
2's 1100000 = -32	2's 1111000 = -8.

-32 x 8.

11100000 x 0001000 [0]

11100000 x 0-1+1000

```
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 1 0 0 0 0 0
  1 1 1 1 1 0 0 0 0 0
  0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0
  1 1 1 1 1 1 0 0 0 0 0 0 0 0
```

-18 x -19

18 19

010010 010011

1's 101101 1's 101100

\+ 1 + 1

-18 = 101110 101101 = -19

-18 x -19

101110 x 101101 0

101110 x -1 1 0 -1 1 -1

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

1 1 1 1 1 1 0 1 1 1 0

0 0 0 0 0 1 0 0 1 0

0 0 0 0 0 0 0 0 0 0

1 1 1 0 1 1 1 0

10 00 1 00 1 0

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

-1 → BCD

101

+55 x -55

55

110111

1's 001000

\+ 1

-55 2's 001001

55 x -55

110111 x 001001 0

110111 x 0 1 -1 0 -1 -1

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

0 0 0 0 0 0 0 1 0 0 1

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 1 0 0 1

1 1 1 1 0 1 1 1

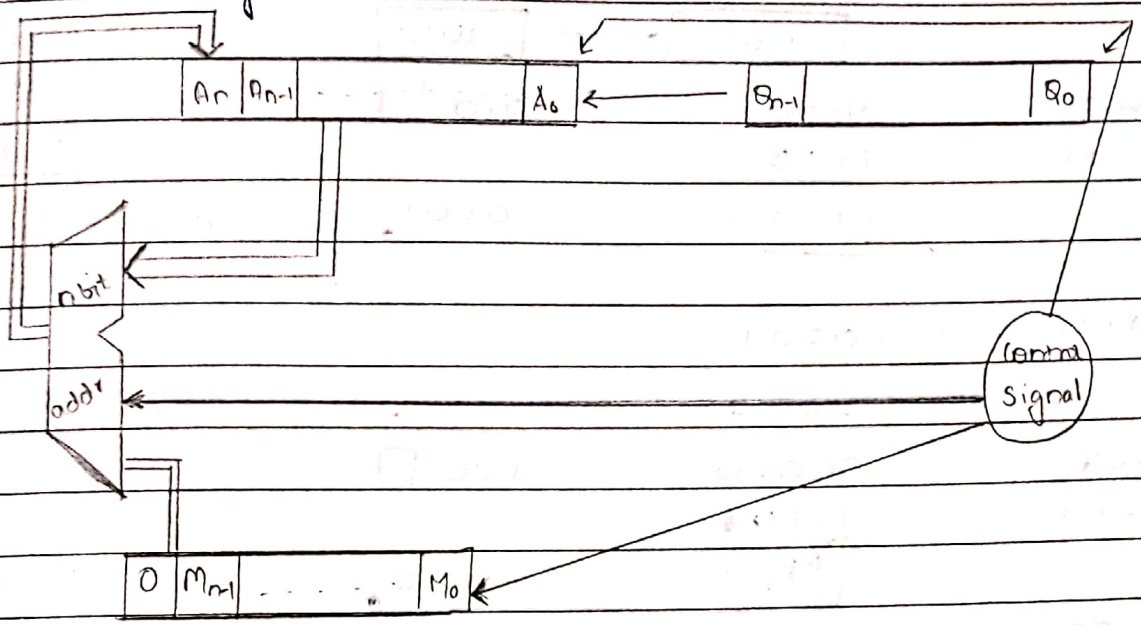
0 0 0 0 0 0 0

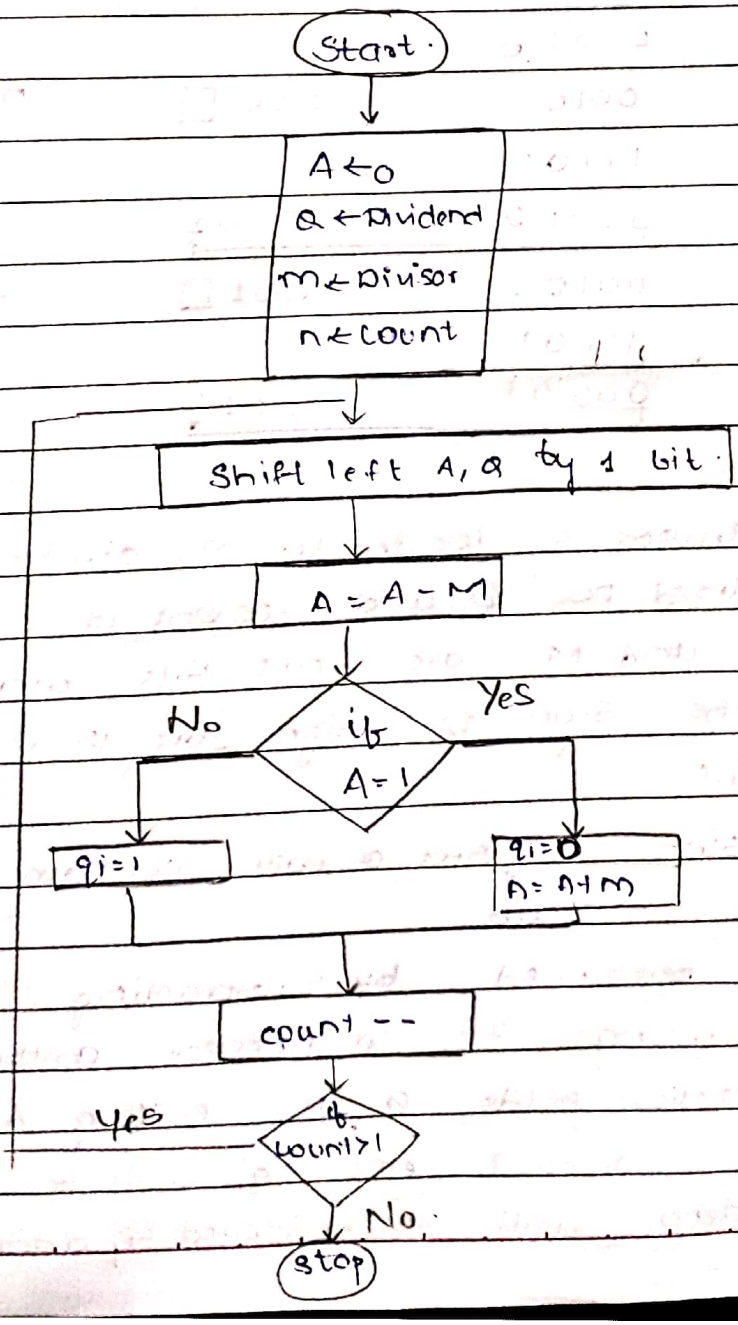
1 1 1 1 1 1 0 1 0 0 1 1

Camlin | Page
Date 30/10/17.

Integer Division (we need to get both remainder & quotient)

→ Restoring division





Q M

1) 10 ÷ 3

	A	Q	M
	00000 ←	1010	00011
Shift	00001	010 □	1's 11100
A-M	11101		+ 1
	11110	0100	11101
			n=4
A+M.	+ 00011		
	00001		n=3
Shift	00010	100 □	
A-M	11101		
	11111	1000	
A+M.	00011		
	00010		n=2
Shift	00101	000 □	
A-M	11101		
	00010	0001	
Shift	00100	001 □	n=1
A-M	11101		
	00001	0011	
			n=0

→ Initially divisor is loaded to M, dividend is loaded to Q. A is initialised to 0 and count is initialised to n

→ Register A and M are n+1 bits where the MSB represents the sign. If sign bit is 0 it is positive, 1 it is negative

→ After division register Q will contain quotient and A will contain remainder.

→ Division is performed by substracting divisor from dividend using 2's complement arithmetic

→ if substraction yields 0 at MSB of A then q_i is made equal to 1 else q_i will be equal to zero and dividend will be restored by adding back divisor

→ At each stage A & Q is shifted left one bit

2) 12 ÷ 2
Q M

	A		Q	M
	00000	←	01100	00010
				1's 11101
				+ 1
				11110
Shift	00001		100 □	
A-M	11110			n=4
	11111		1000 ↑	
A+M	00010			
	00001			
Shift	00011		000 □	n=3
A-M	11110			
	00001		0001 ↑	
Shift	00010		001 □	n=2
A-M	11110			
	00000		0011 ↑	
Shift	00000		011 □	n=1
A-M	11110			
	11110		0110 ↑	
A+M	00010			
	00000			

M → n bits

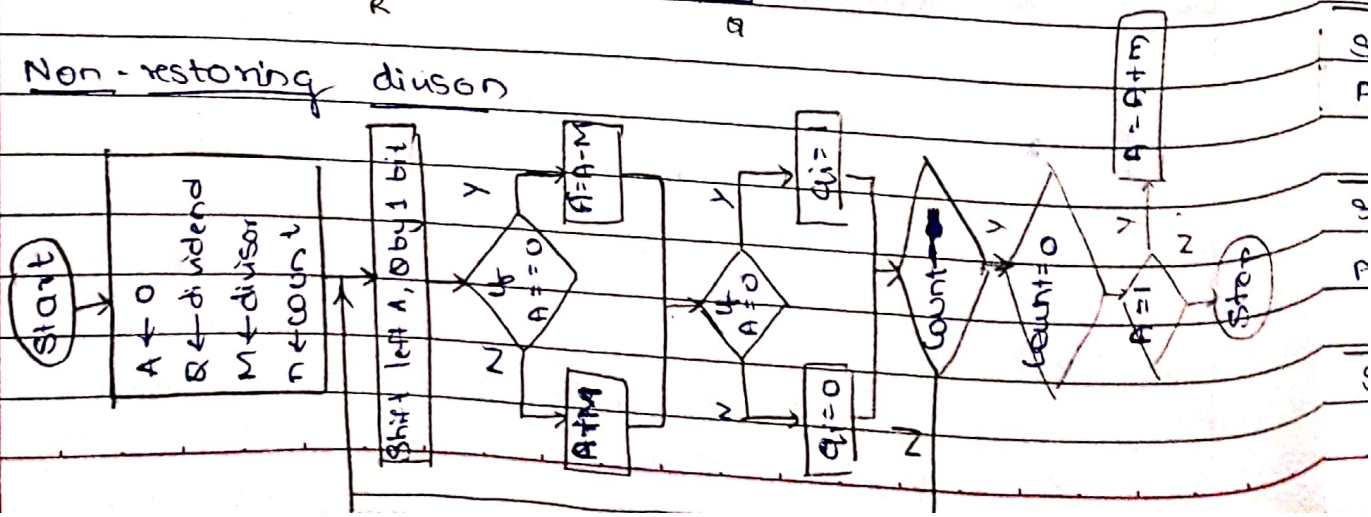
B, A → n+1 bits

Q M

27) 8 ÷ 5

	A	Q	M
	00000	1000	00101
			1's 11010
			+1
			11011
Shift	00001	000 ☐	
A-M	11011		n=4.
	11100	0000 ↑	
A+M.	00101		
	00001		
Shift	00010	000 ☐	
A-M	11011		
	11101	0000 ↑	n=3
A+M	00101		
	00010		
Shift	00100	000 ☐	
A-M	11011		
	11111	0000 ↑	n=2
A+M	00101		
	00100		
Shift	01000	000 ☐	
A-M	11011		
	00011	0001 ↑	n=1
	R	Q	

Non-restoring divison



in last step if MSB is 1 then perform one more step A+M

Camlin Page
Date 21 / 11 / 17

15/4

	A	Q	M
	00000	1111	00100 +4
			11011
			+ 1
			11100 -4.
Shift	00001	111 □	n=4.
A-M	11100		
	11101	1110	
Shift	11011	110 □	n=3.
A+M	00100		
	11111	1100	
Shift	11111	100 □	n=2
A+M	00100		
	00011	1001	
Shift	00111	001 □	n=1.
A-M	11100		
	00011	0011	

R Q

8/3.

	A	Q 111	M
	00000	1000	00011
			11100
			+ 1
			11101.
Shift	00001	000 □	n=4
A-M	11101		
	11110	0000	
Shift.	11100	000 □	n=3
A+M	00011		
	11111	0000	n=2
Shift.	11110	000 □	
A+M	00011	0001	
	00001		
Shift	00010	001 □.	n=1.
A-M	11101	0010	
	11111		

Q

Date / /

Shift 1111 0 | 0010

A+M 00011

00010

R.

13/5.

	A	Q	M
	00000	1101	00101
			11010
			+ 1
			11011

Shift. 00001 | 101 □ n=4

A-M 11011

11100 → 1010□

Shift. 11001 | 010 □ n=3

A+M 11011

10100 → 01000

Shift 01000 | 100 □ n=2

A-M 11011

00011 → 10001

Shift 00011 | 001 □ n=1

A-M 11011

00010 | 0011

Booth's algorithm with Bit pair recoding

→ To reduce no. of steps in multiplication, 3 consequetive bits are paired togther to recode the multiplier from LSB

bi	bi-1	bi-2					
0	0	0	→ 0	1	0	0	→ -2
0	0	1	→ 1	1	0	1	→ -1
0	1	0	→ 1	1	1	0	→ -1
0	1	1	→ 2	1	1	1	→ 0

Camlin Page
Date / /

Multiplication is performed by using result of bit pair such that

bitpair	partial product.
0	add 0's to partial product
1	add multiplicant to P.P & Sign extend
-1	add 2's complement of M to P.P & Sign extend
2	add M with 0 at LSB to P.P & Sign extend.
-2	add 2's C of M to with 0 at LSB to P.P & Sign extend.

NOTE 1) always use 5 or more bits for M & Q
2) Consider an implied zero at LSB before recoding
3) Sign extend the Multiplier by 1 bit (M) at MSB before recoding
4) After every step, shift P.P by 2-bits

1) 13 x -6
M Q

+13 01101
10010
+ 1
-13 10011

+6 00110
11001
+ 1
-6 11010

01101 x 1 11010 0

recode: 01101 x 0 -1 -2

1 1 1 1 1 0 0 1 1 0
1 1 1 1 0 0 0 1 x x
0 0 0 0 0 0 x x x x
0 0 0 0 0 1 1 0 1 0
1 1 1 0 1 1 0 0 1 0

Scanned by CamScanner

2) -8×-8.

m Q

+8 01000

10111

\+ 1

-8 11000

11000 × 111000 0

recode 11000 × 0 -2 0

0000000000

0001600 0 xx

0 0 0000 xxxx

0 0010000000

Hint

3) -15×3.

m Q.

+15 01111

10000

\+ 1

-15 10001

+3 00011

11100

\+ 1

-3 11101

10001 × 000011 0

10001 × 0 +1 -1

00000 01111

11110 001xx

00 000 0xxxx

11 11010011

4) $.25 \times -18$

25 00 11001

11001 10

\+ 1

-25 1100111

$16^{8} 4^{2} 1$

18 00 10010

11011 01

\+ 1

-18 11011.10.

1100111 × 1 1101.110 0

1100111 × 0 -1 0 -2

0000000011 0010

0 0 0 0 0 0 0 0 0 0 0 0 xx

0 0 0 0 0 1 1 0 0 1 xxxx

0 0 0 0 0 0 0 0 0 xxxxxx

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

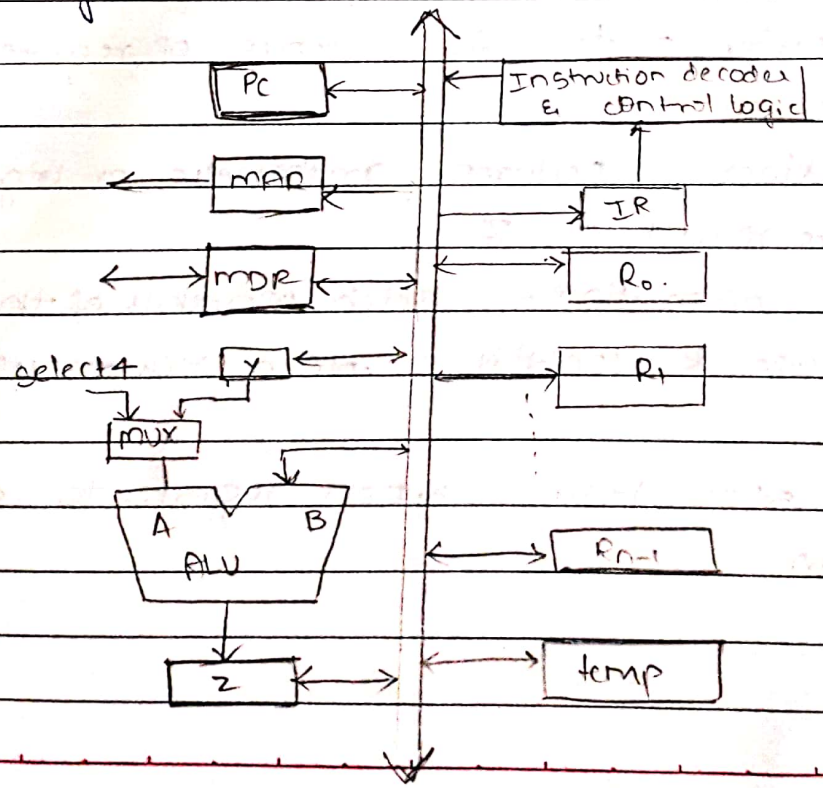
2] $\Rightarrow -2^{6-1}$ to $+2^{6-1}-1$

take odd no. 7 bits

MODULE 5

BASIC PROCESSING UNIT

single bus Architecture.



To execute a program processor fetches instruction & perform specified operations using PC & IR. Foll 3 steps are involved in execution of instruction

1) Fetch the instruction from the memory using address specified in PC & store it in IR

2) Increment PC to point to next instruction.

PC = PC + 4

3) Perform the operation specified in the instruction.

Above figure shows organisation of processor connected using single bus.

→ MAR is connected to address bus & internal processor bus

→ MDR is connected to data bus & internal processor bus

→ Registers R_0 till R_{n-1} are GPRS ~~are~~ connected to processor bus

→ Registers Y, Z, temp are temporary registers

→ mux selects either Y or value 4 as input to A of ALU

→ IR holds current instruction to be executed.

→ instruction decoder generates control signals to perform desired operations

→ instruction execution can be performed by one or more of foll operations

↳ Register transfer - transfer contents of register to another register or ALU

↳ ALU execution - perform arithmetic or logical operation & store result in Z

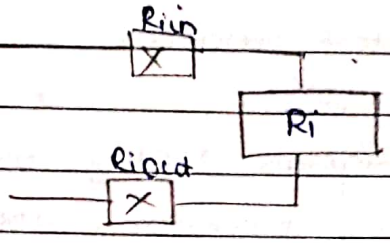
↳ Fetch data or instruction - fetch contents of the specified memory location & load it into a register using MAR & MDR

↳ Store data - store data from register to specified memory location.

Camlin Page
Date / /

Register Transfer:

Each Register will have 2 control signals $R_i in$ & $R_i out$ when $R_i in = 1$ load value into register, when $R_i out = 1$ read value from register



MOV R_1, R_2

$[R_1] \rightarrow R_2$

$R_1 out$, $R_2 in$

ALU execution: ADD R_1, R_2, R_3

$[R_1] + [R_2] \rightarrow R_3$.

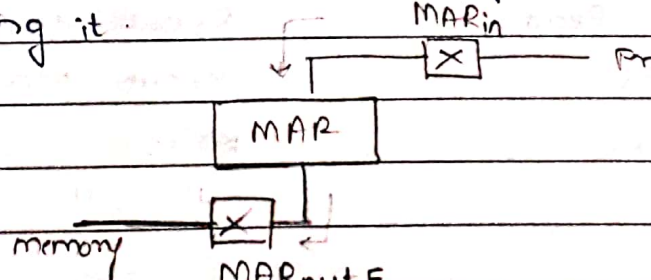
① $R_1 out$, $Y in$	$R_3 out$, Y_{in}
② $R_2 out$, Select Y	$R_4 out$, Select Y
③ ADD , $Z in$	SUB , Z_{in}
④ $Z out$, $R_3 in$	$Z out$, $R_4 in$.

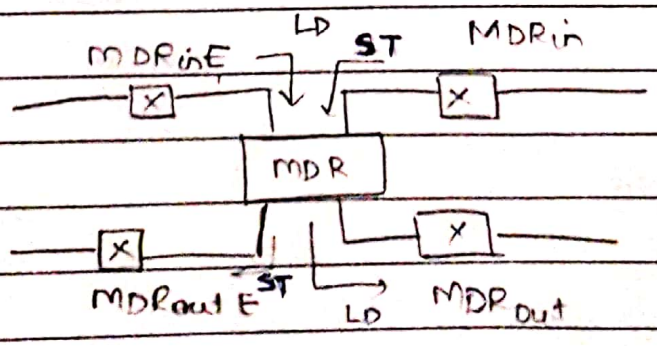
elaborate

SUB R_3, R_4

$[R_3] - [R_4] \rightarrow R_4$

Fetch instruction: fetching a word from memory & storing it.





Date 9 / 11 / 19

MAR register has 2 control signals. MARin used to specify address to be read or written from memory by processor. MARoutE used by memory to read specified address by the processor. Both control signals are used during load & store operations.

MDR has 4 control signals. MDRin used to write one word data into MDR by processor. MDRoutE used by memory to read one word of data in MDR.

These 2 control signal are used during store operation. MDRinE used by memory to write 1 word data into MDR. MDRout used by process to read 1 word of data from MDR. These 2 control signals are used during load operation.

MFC : It is a control signal used to represent the completion of memory read or write operation.

Load & store operations.

MOV $(R_1), R_2$

R_1out , MARin , Read.
MDRinE , WMFC
MDRout , R_2in

ST $R_2, (R_3)$

R_2out , MDRin
R_3out , MARin
Write.

1. ADD $(R_3), R_4$.

1) Fetch instruction
2) Decode instruction
3) Fetch operand
4) Execute instruction
5) Store result.

PC_{out}, MAR_{in}, read
MDR, Select 4, ADD, Zin
Zout, PCin, Yin
MDR_{inE}, WMFC
MDR_{out}, IR_{in} → decode
R_3out, MARin, read
MDR_{inE}, WMFC
MDR out, Yin
R_4out, Select Y, ADD, Zin
Zout, R_4in

Fetch inst^n

Fetch operand

Execute instruction

The incremented PC content in Z is placed into PC & also into register Y. If the current inst^n is a branch the branching will be done based on PC relative addressing mode i.e. the effective address is calculated by adding offset to PC.

2. ADD R_2, R_3.

→ Pcout, MARin, read
Select 4, ADD, Zin
Zout, PCin, Yin
from memory
← MDR_{inE}, WMFC.

Fetch inst^n

→ MDR_{out}, IR_{in} } decode

→ R_2out, Yin
R_3out, Select Y, ADD, Zin
Zout, R_3in

Execute inst^n

3. MUL ((R_2), R_3, R_4.

→ R_2out, MARin, read
MDR_{inE}, WMFC
MDRout, Yin
R_3out, Select Y, MUL, Zin
Zout, R_4in

4) SUB R_1, (R_2)

PCout, MARin read

Select 4, ADD, Zin

Zout, PCin, Yin

WMFC

MDRout, IRin

R_2out, MARin read WMFC

R_1out, Yin

MDRout, Select Y, SUB, Zin, Zout, MDRin

R_2out, MARin write.

5) MUL (R_3), (R_4)

PCout, MARin, read

Select 4, ADD, Zin

Zout, PCin, Yin WMFC

MDRout, IRin

R_3out, MARin, read WMFC

R_4out, MDRout, Yin, MARin, read WMFC

MDRout, Select 4, MUL Zin

Zout, MDRin, R_4out, MARin write

6) ADD 4(R_0), R_1

PCout, MARin, read.

Select 4, Add, Zin

Zout, PCin, Yin

WMFC

MDRout, IRin. Roout, Select 4, ADD, Zin

Zout, MARin read WMFC

MDRout Yin

R_1out, Select 4, ADD Zin

Zout, Rin

Camlin Page
Date / /

27) ADD -(R_0), R_1

Pcout, MARin, read
Select 4, ADD, Zin
Zout, PCin, Yin
WMFC
MDR out, IRin
R_0 out, Select 4, SUB Zin
Zout, MARin, read WMFC
MDRout, Yin
R_1out, Select Y, ADD Zin
Zout, R_1in

PCout, MARin, read
Select 4, ADD, Zin
Zout, PCin, Yin
WMFC
MDRout IRin
R_0out, Select 4, Sub, Zin, R_0in
Zout, MARin

→ SUB $(R_0)+$ R_1.

PCout , MARin , read

Select 4, ADD , Zin

Zout , PCin , Yin

WMFC

MDRout IRin

R_0 out , MARin , Read , select 4.

ADD , Zin , Zout , R_0in WMFC

MDRout , Yin , R_1 out , Select 4 , Sub , Zin

Zout , R_1 in

★★ Branching instructions.

2000	Mov	R_0, R_1	LOOP	2036	SUB . . .
2004	ADD	- . .		2040	- - - .
2008	-	- . .		2044.	- - - . .
2016	-	- - - .			
2020	BR	LOOP			
2024	-	- . . .			

Offset = EA − PC

= 2036 − 2024

= 12.

→ 1 PCout , MARin , read

→ 2 Select4, ADD$_{2024}$, Zin$_{2024}$

→ 3 Zout , PCin , Yin

WMFC

→ 4 MDRout IRin

→ 5 (12) offset in IR$_{out}$, Select Y , ADD , Zin

→ 6 Zout , PCin

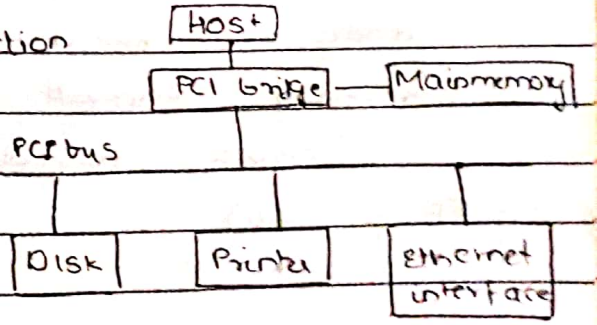
Branch instruction replaces contents of PC with Branch target address, which is obtained by adding offset with updated/incremented content of PC. Step 1 till 3 = fetch instruction, incrementing PC; the incremented PC content is also placed in register 4.

Step 5: offset is added with Y which contains incremented PC value to get the branch target address.
Step 6: The target address is placed on to PC.

5 PCI [Peripheral component Interconnect] Bus.

It supports the functions found on a processor bus but in a standardized format that is independent of any particular processor. Devices connected to PCI bus appear to processor as if they were connected directly to processor bus.

Data Transfer :- Data is transfered between cache & main memory in bursts of several words each; The words involved in such a transfer are stored at successive memory locations. When processor requests a read operation from main memory, the memory responds by sending a sequence of data words
15 starting at that address.
During write operation, processor sends memory address followed by sequence of data words, to be written in successive memory locations at that address.
A read or write operation involving single word is a
20 burst of length one.
Bus supports 3 address spaces: memory, I/O & configuration I/O address space is intended for use with processor such as Pentium; configuration space is intended to give PCI its plug & play capability. 4 bit command that
25 accompanies address identifies which of 3 spaces is being used.
PCI bridge provides separate connection for main memory.
Master maintains address
30 information on bus until data transfer is complete. This is not compulsory; address is needed long enough for slave to be selected.



Date / /

The slave can store address in its internal buffer. The result is significant reduction in cost.

At any given time only one device is bus master. It can initiate data transfers by issuing read and write commands. A master is called initiator in PCI terminology. The addressed device that responds to read & write commands is called a Target.

Data transfer signals on PCI are CLK, FRAME#, AD, C/BE#, IRDY#, TRDY#, DEVSEL#, IDSEL#

#:- asserted when in the low voltage state.

Consider a bus transaction in which processor reads 4 32-bit words from memory. In this case, initiator is processor, target is memory. A complete transfer operation on bus, involving an address & a burst of data, is called Transaction. All signal transitions trigger at raising edge of clock.

In clock 1, processor asserts FRAME#, to indicate begining of transaction, sends address on AD lines & command on C/BE# lines. command will indicate that read operation is requested.

Clock 2: Turns the AD bus lines around, address is removed & drivers are disconnected from AD lines. Selected target enables drivers & fetches required data.

Clock 3: initiator asserts IRDY# to indicate it is ready to receive data. If target has data ready TRDY# is asserted. FRAME# is negated during clock 5.

In clock 6 DEVSEL# is negated.

(Dig TB pg 264)

Camlin Page
Date / /

SCSI [Small Computer System Interface]

Devices connected to SCSI bus are not part of address space of processor in same way as devices connected to processor bus. It is connected to processor bus through SCSI controller which uses DMA for data transfer.

Controller

controller connected to SCSI bus is of one of 2 types - initiator or a target. An initiator has ability select particular target & to send commands specifying operation to be performed. Controller on processor side, such as SCSI controller must be able to operate as an initiator. DISK controller operates as a target. It carries out command it receives from initiator. There is a logical connection with intended target.

Data transfers on SCSI bus are always controlled by target controller. To send a command to target, initiator requests control of bus; after winning arbitration; selects controller it wants to communicate with & controller starts data transfer.

→ Processor sends command to SCSI controller, then

1) SCSI controller (as initiator) contends for control of bus.
2) After winning arbitration, target controller is selected & handed over to bus.
3) target starts o/p operation; initiator requests read operation
4) Target sends a msg saying it will temporarily suspend connection with initiator.
5) It reads data stored & stores in buffer.
6) contents are transfered to initiator & connection is established.
7) Target controller sends command to disk drive to perform another seek operation
8) Initiator receives and stores data in main memory (using DMA)
9) SCSI controller sends interrupt to processor saying request operation is completed.

Date / /

SCSI BUS signals.

All signal names are preceded by '-' indicating that it is active in low-voltage state. No address line; data lines are used to identify bus controllers. It is possible to have more than one address on bus at same time.

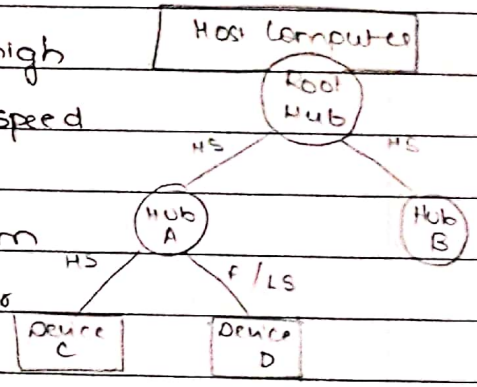
Category	Name	Function.
Data	-DB(0) to -DB(7)	Data lines: Carry one bit of info during info transfer & identify device using arbitration, selection & reselection process.
	-DB(P)	Parity bit for data bus.
Phase	-BSY	Busy: Asserted when bus is not free
	-SEL	Selection: Asserted during selection & reselection
Info. type	-C/D	control/data: Asserted during transfer of control information
	-MSG	message: indicates that info being tranfered is a message.
Handshake	-REQ	Request: Asserted by target to request a data tranfer cycle.
	-ACK	Acknowledge: Asserted by indicator when it has completed data transfer operation.
Direction of transfer	-I/O	Input/output: Asserted to indicate an I/P operation
Other	-ATN	Attention: Asserted by indicator when it wishes to send a message to a target
	-RST	Reset: causes all device controls to disconnect from bus & assume their startup state.

USB

Camlin Page
Date / /

a) Tree structure with split bus operation

USB is a serial bus that satisfies low cost & flexibility requirement. It accomodates a large no. of devices that can be added or removed at any time. It follows a tree structure. Each node of tree has a device called hub. At the root of tree, root hub connects entire tree of host computer; leaves are I/O devices. USB operates on basis of polling. A device is may send msg only in response to a poll msg from host. Hence upstream msgs do not encounter conflits or interface with each other. Devices operate either in low speed or full speed.



Hub A is connected to Root hub by a high speed link. This hub A serves one high speed device C & low speed device D. A msg to device D can be sent at low speed from root hub; for this duration no other data transfer take place. The USB protocols require that a msg transmitted on high speed (Signal) link is always transmitted at high speed even when ultimate receiver is low speed device. Hence a msg for device D can be sent at high speed from root hub to hub A then forwarded at low speed to device D.

Purpose of USB (Software protocols) is to provide bidirectional communication links btw app slw & I/O devices. These links are called pipes.

Computer systems are used in a numerous of applications; therefore, they come in a variety of organizations, sizes, and capabilities. The important factors considered for any application is performance, reliability, and cost.

Microprocessors are now commonly used in cameras, cell phones, kitchen appliances, cars, and many toys. Low cost and high reliability, small size and low power consumption are key importance in such applications.

Microprocessor chips that include I/O interfaces and some memory are called as *micro controllers*.

A physical system that employs computer control for a specific purpose, rather than for general-purpose computation, is referred to as an *embedded system.*

EXAMPLES OF EMBEDDED SYSTEMS

I. Microwave Oven

Microwave oven is based on a magnetron power unit that generates microwaves used to heat food.

When turned on, the magnetron generates its maximum power output. The power level and the total heating time can be controlled based on user's cooking options.

The specification for a microwave oven

1. Cooking options
   a) Manual selection of the power level and cooking time
   b) Manually selected sequence of cooking steps
   c) Automatic selection cooking steps, power level & time based on user specified type of food and the weight of the food.
   d) Automatic defrosting of meat by specifying weight
2. Output display of Oven:
   a) Time of day clock
   b) Decrementing clock timer while cooking
   c) Information messages to fee user
3. An audio alert signal beep tone to indicate fee end of a cooking
4. An exhaust fan and oven light
5. A door interlock -magnetron off if the Oven door is open
6. The input/output capability
   a) Input keys that comprise the number pad 0 to 9 and function keys such as Reset, Start, Stop, Power Level, Auto Defrost, Auto Cooking, Clock Set, and Fan Control.
   b) Visual output in the form of a LCD.
   c) A small speaker feat produces fee beep tone.

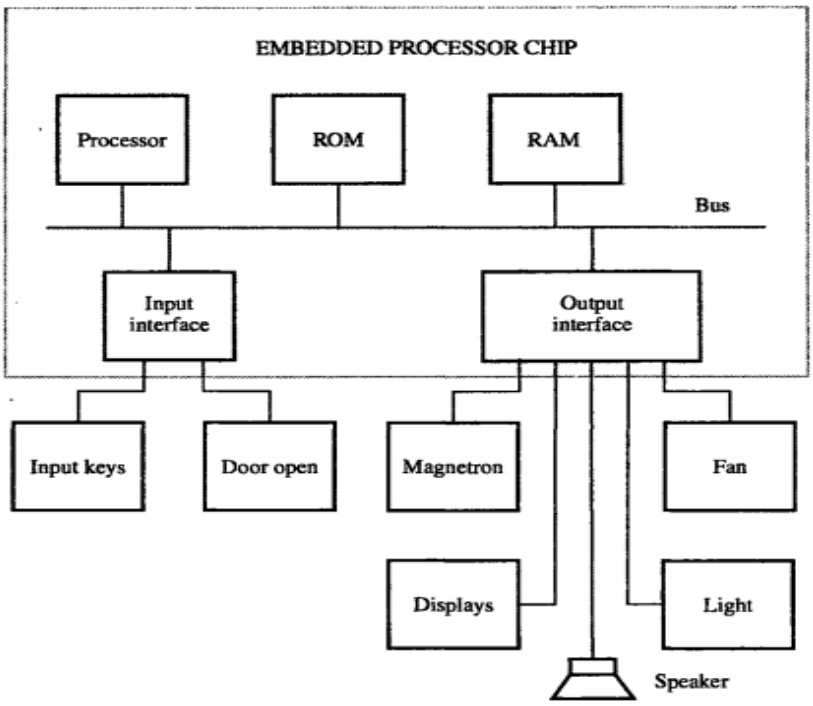


Figure 9.1 A block diagram of a microwave oven.

Implementation:

The controller for a microwave oven can be implemented by a small microprocessor based computer unit.

The computational tasks include:

- Maintain time of day clock
- Determining the actions needed in the various cooking options
- Control signals generation to turn on or off magnetron and fan
- Displaying information.

The program for controlling oven must be stored in a ROM

Oven also need RAM to store user specified data & computation

A simple processor with small ROM and RAM units and parallel I/O on a single VLSI chip is sufficient to implement the controller of oven in a cost effective way.

II. DIGITAL CAMERA

Digital camera, contains array of optical sensors for capturing images. These sensors are based on photodiodes which convert light into electrical charge.

Two different types of sensors are used:

- Charge-Coupled Devices (CCDs)
- Sensors based on CMOS technology

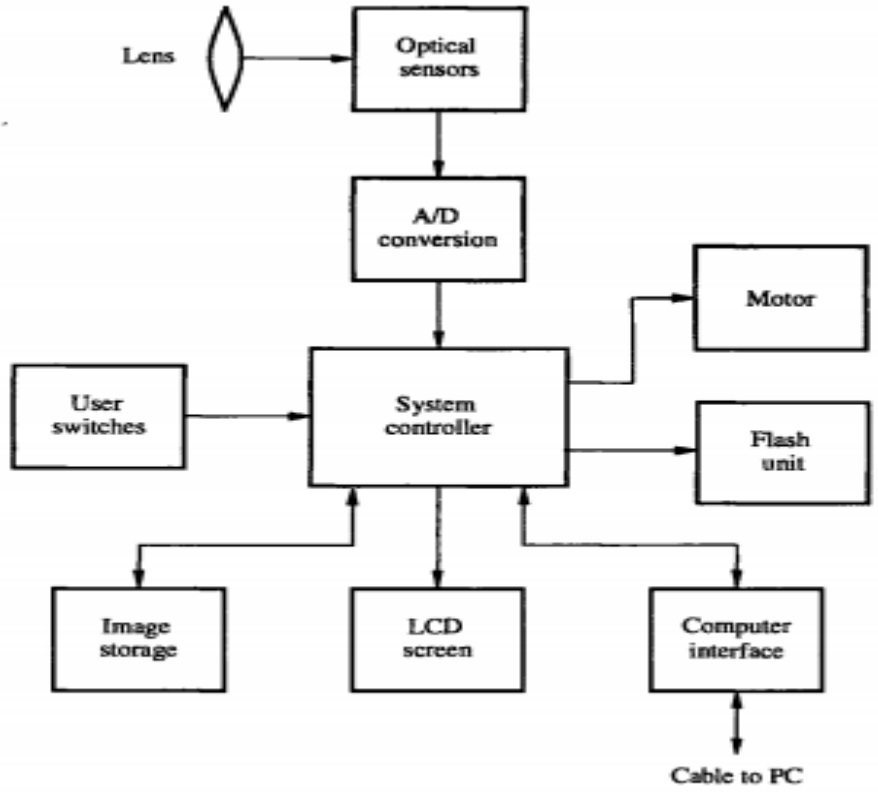


Figure 9.2 A simplified block diagram of a digital camera.

Each sensor generates a charge that corresponds to one pixel, number of pixels determines the quality of picture.
The charge is an analog which is converted into a digital using A/D conversion circuits.
In digital image the color and intensity of each pixel is represented by a number of bits.
The camera controller contains a processor, memory (both RAM and EEPROM), and set of interface circuits to connect to other parts of the system.
The processor obtain raw image from the A/D circuits and generate images in standard formats such as TIFF, JPEG etc.
A captured image can be displayed on LCD screen in the camera and are stored in a larger flash storages.
PCI or USB interfaces are used to transfer the images to a computer or a printer.
The camera controller generates signals to control the operation of the motor and the flash unit.
A digital camera requires more powerful processor with less power consumption to perform complex signal processing functions.

PROCESSORS FOR EMBEDDED APPLICATONS

A chip that contains a processor, some memory, and I/O interface circuitry useful in embedded applications is called as **embedded processor**. As they perform important control functions are also known as **microcontroller chips**.

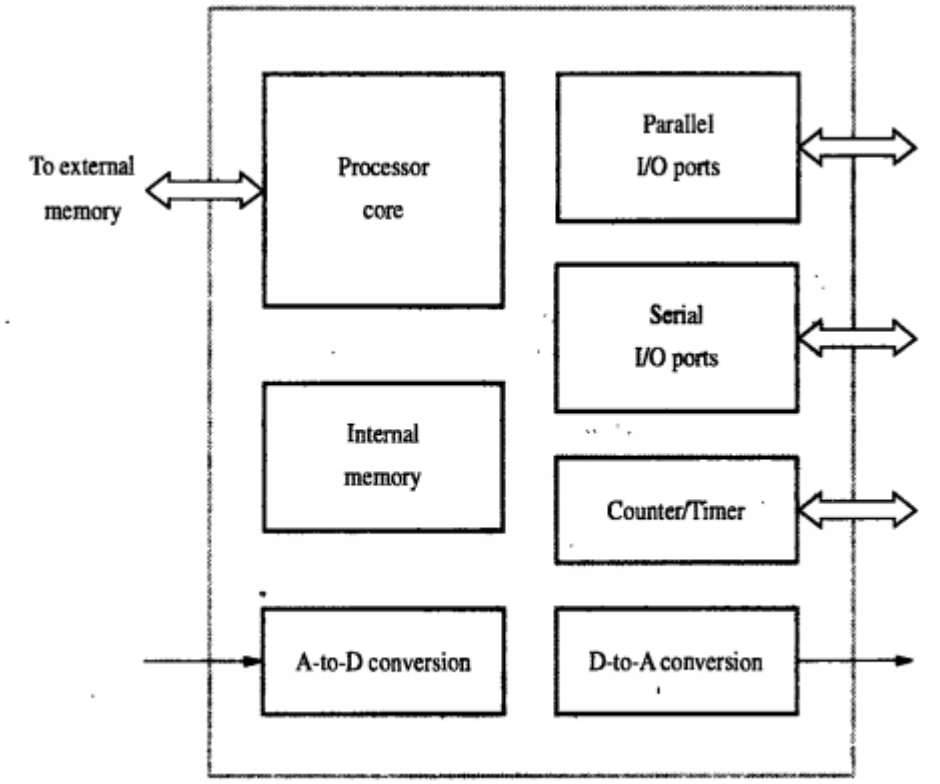


Figure 9.3 A block diagram of an embedded processor.

The main parts are:

- Processor core
- RAM to hold the data during computations
- ROM to hold the software of embedded system
- Storage are EEPROM and Flash memory
- I/O ports to provided for both parallel and serial interfaces
- Timer circuit to generate control signals
- AID and D/A conversion circuits

Some of the commercially available embedded processor chips:

CISC-type processor cores

- Motorola's 68HC11, 683xx and MCF5xxx families, Intel's 8051 and MCS-96family

RISC-type processor

- ARM microcontrollers

A SIMPLE MICROCONTROLLER

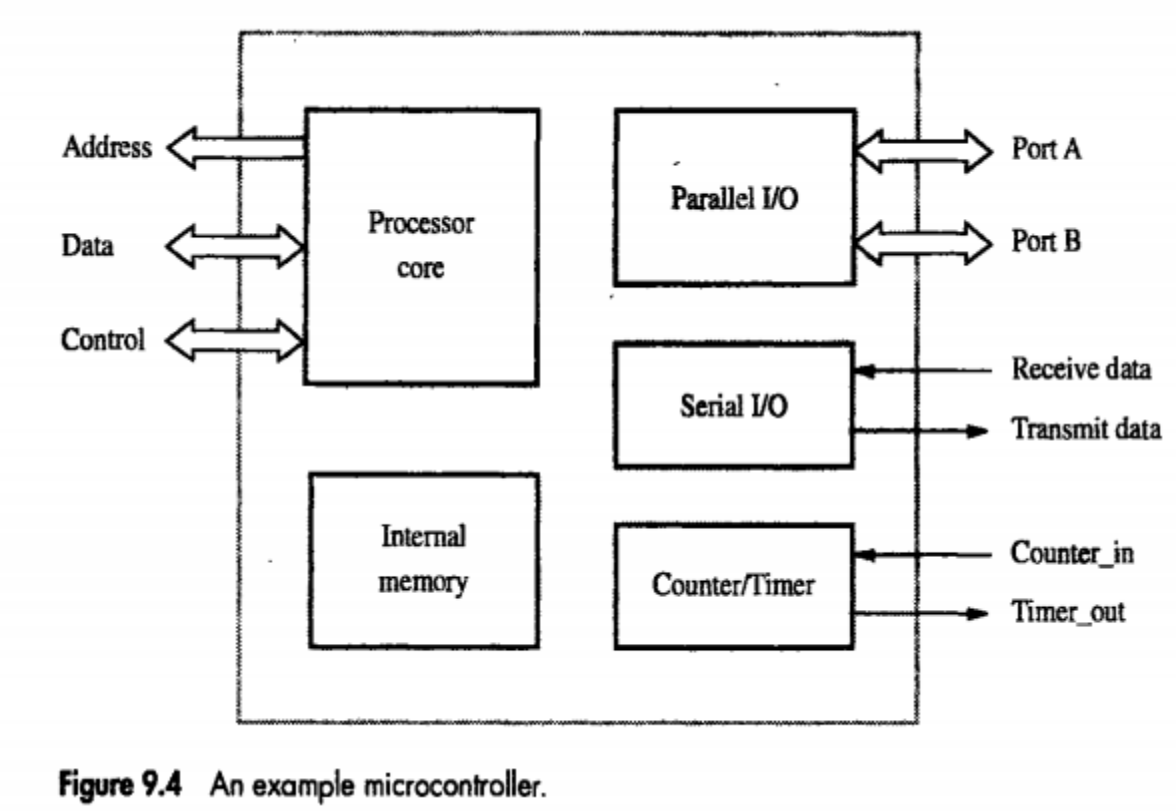


Figure 9.4 An example microcontroller.

Figure 9.4 shows the block diagram of simple microcontroller which contains:

- Processor core
- on-chip memory
- processor bus connections on the chip to connect to external memory
- two 8-bit parallel interfaces, called A and B
- one serial interface
- 32-bit counter/timer circuit to generate internal interrupts and serve as a system stop watch

PARALLEL I/O PORTS

A and B ports can be used as either inputs or outputs Figure 9.5 illustrates the bidirectional control for one bit in Port A.

Port pin PAi is treated as an input if the data direction flip-flop contains 0. In this case Read_port signal is placed on port and data onto the data line Di of the processor bus and processor reads the data on the pins.

The port pin PAi serves as an output if the data direction flip-flop is set to 1. In this case, the data is loaded into the data output flip-flop and Write Port signal, is placed on the pin.

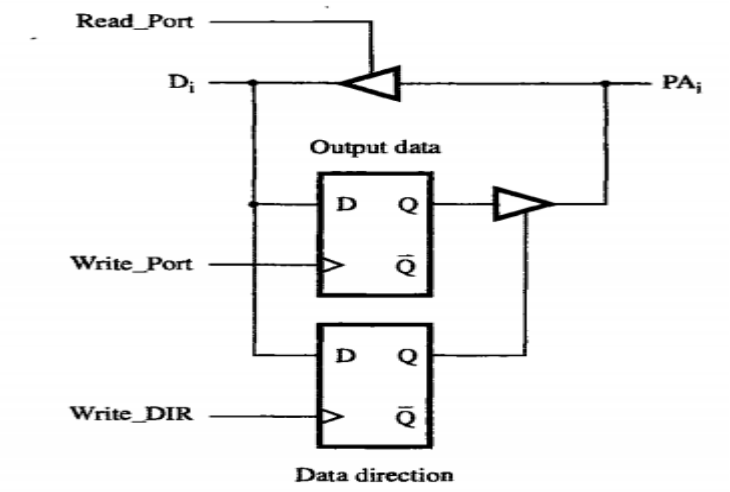


Figure 9.5 Access to one bit in Port A in Figure 9.4.

Figure 9.6 shows the eight 8-bit registers used for data transfer operations on ports A and B and addresses assigned to these registers.

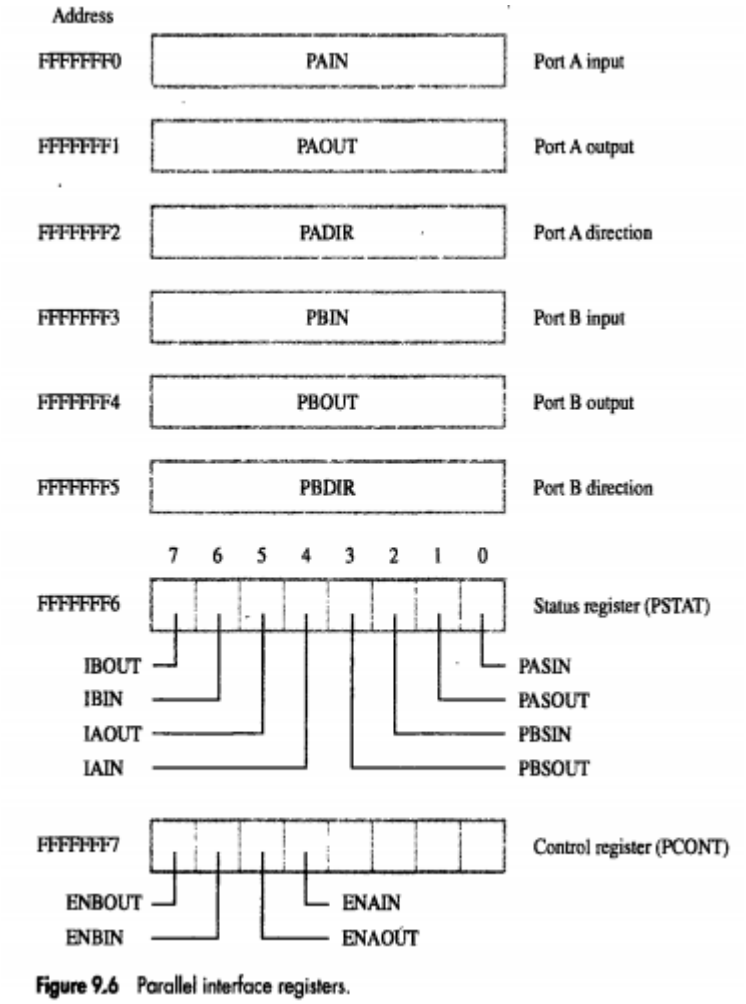


Figure 9.6 Parallel interface registers.

The status register, PSTAT, contains the status flags
The PASIN flag is set to 1 when there is new data on the pins of port A. 0 when the processor accepts the data
The PASOUT flag is set to 1 when the data in register PAOUT are transferred to the connected device and now processor can load new data into PAOUT. It is 0 when the processor writes data into PAOUT.
The flags PBSIN and PBSOUT perform the same function for port B
The status register also contains four interrupt flags.
If IAIN= 1 then interrupt is enabled and the corresponding I/O action takes place. The interrupt enable bits are held in PCONT.
A single interrupt request signal is used. The processor must examine the interrupt flags to determine the actual source of the request.
Information in the status and control registers is used for controlling the data transfers to and from the devices connected to ports A and B.
When the device places data on portA, CAIN is set to 1 it sets the PASIN to 1 after processor reads data 0 PASIN.
For an output transfer, the processor writes the data into the PAOUT which makes PASOUT bit to 0 and sends signal to CAOUT. When the device takes data, sets PASOUT to 1.

SERIAL I/O INTERFACE

The serial interface provides the UART (Universal Asynchronous Receiver Transmitter) to transfer data Figure 4.37. Double buffering is used in both transmit and receive paths, Figure 9.7. Figure 9.8 shows the addressable registers of the serial interface.
Input data are read from the 8-bit Receive buffer, and output data loaded into the 8-bit Transmit buffer. The status register, SSTAT, provides status of receive and transmit units.
SSTAT0 is set to 1 when there is data in the receive buffer.
SSTAT1 is set to 1 when the transmit buffer is empty and can be loaded with new data.
Interrupt flags:

- SSTAT2 is set to 1 if an error occurs during the receive process.
- SSTAT4 is set to 1 when the receive buffer is full.
- SSTAT5 is set to 1 when the transmit buffer is empty

SCONT, is used to hold the interrupt enable bits which can enable or disable corresponding interrupts.
DIV 32-bit register divides the system clock signal to generate the serial transmission clock.

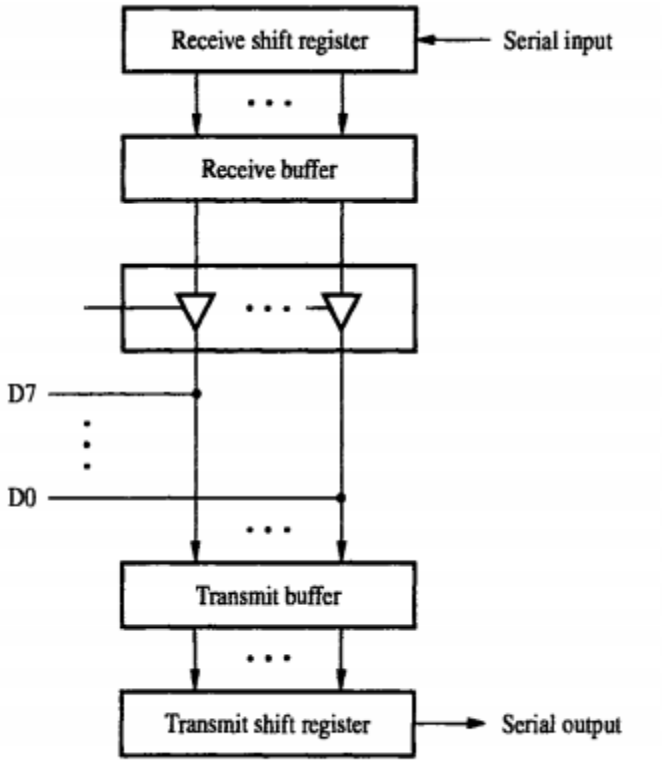


Figure 9.7 Receive and transmit structure of the serial interface.

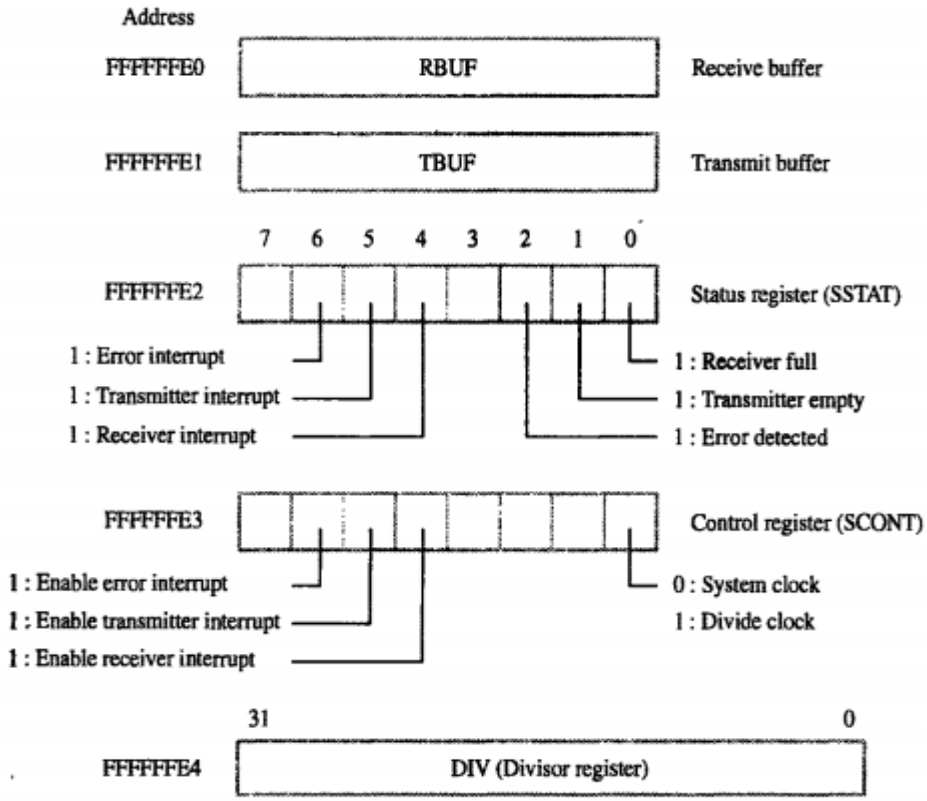


Figure 9.8 Serial interface registers.

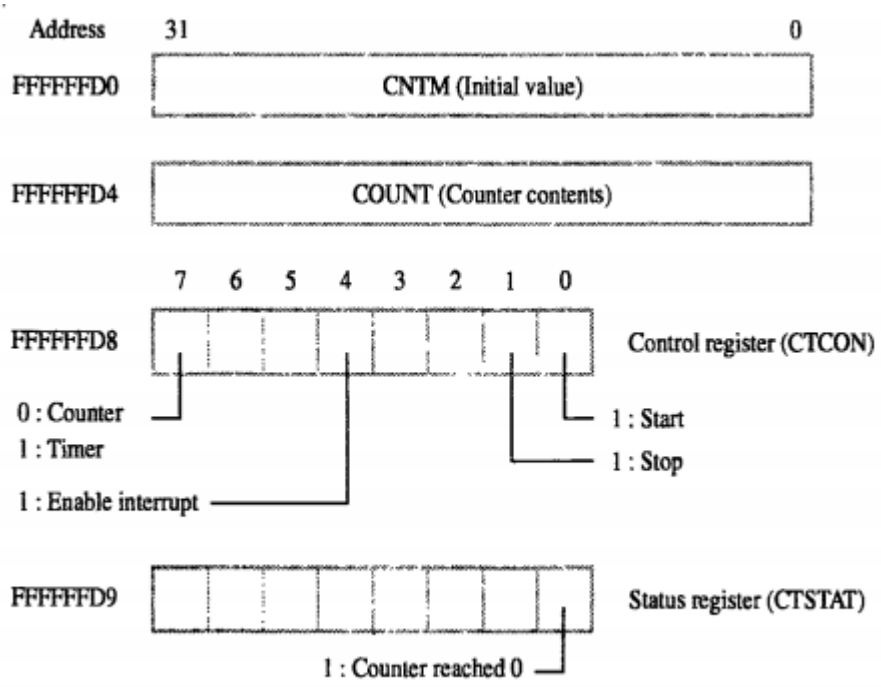


Figure 9.9 Counter/Timer registers.

COOUNTER/TIMER

A 32-bit down-counter circuit is used as counter or a timer.

The circuit loads starting value into the counter, and then decrement the counter by using internal system clock or an external clock signal then raise an interrupt when the counter reach zero.

The counter registers:

- CNTM loaded with an initial value then transferred into the counter circuit.
- CTCON specify the operating mode of the counter/timer circuit.
- CTSTAT, reflects the state of the circuit.

Counter Mode

The counter mode is selected by setting CTCON7 to 0.The starting value is loaded into the counter by writing it into register CNTM.

The counting process begins when bit CTCON0 is set to 1 by a program instruction then the CTCON0 bit is cleared to 0.

The counter is decremented by pulses on the Counter_in line. Upon reaching 0, the counter circuit sets the status flag CTSTAT0 to 1, and will raise an interrupt.

The counting process is stopped by setting CTCON1 to 1.

Timer Mode

The timer mode is selected by setting CTCON7 to 1. This mode is suitable for generating a square-wave signal on the output line Timer_out.

The counting process begins when bit CTCON0 is set to 1 by a program instruction then the CTCON0 bit is cleared to 0.

As the counter counts down, the value on the output line is held constant. Upon reaching zero, the counter is reloaded automatically with the starting value, and the output signal on the line is inverted.

INTERRUPT CONTROL MECHANISM

The microcontroller has two interrupt request lines, IRQ and XRQ. The IRQ for I/O interfaces within the microcontroller and XRQ for interrupts raised by external devices.

When IRQ line is active the processor polls to determine the sourceof the interrupt request by examining PSTAT, SSTAT and CTSTAT.

PSR, has two bits for enabling interrupts. The IRQ interrupts are enabled if PSR6 = 1, and the XRQ interrupts are enabled if PSR7 = 1.

A vectored interrupt scheme is used, with the vectors for IRQ and XRQ each vector contains the address of the corresponding interrupt service routine.

During interrupt request in addition to saving the return address in LR, the contents of the processor status register, PSR, are saved in a processor register IPSR.

Computer Organization

Module 1

1. Explain the **basic functional units** of computer
2. Explain the **basic operational concept** with a neat diagram and demonstrate with an example how fetch and execute sequence occurs.

   OR

   Draw and explain the connection between memory and processor with the respective registers.
3. Explain the methods to improve performance of computer.(Basic Performance Equation, SPEC rating, clock rate, pipeline, CISC & RISC).
1. Explain clearly SPEC rating and its significance. Assuming that the reference computer is ultra SPARCIO work station with 300 MHz ultra SPARC processor. A company has to purchase 1000 new computers hence ordered testing of new computer with SPEC 2000.Following observations were made

Programs	Runtime on reference computer	Runtime in new computer
1	50 Minutes	5 Minutes
2	75 Minutes	4 Minutes
3	60 Minutes	6 Minutes
4	30 Minutes	3 Minutes

1. Explain the following
   a. Memory Location and addressing using Big endian and little endian
   b. Overflow & Conditional codes
2. Explain the 3 ways of number representation for both signed and unsigned numbers in detail with examples. Also show how addition and Subtraction is performed.
3. Perform the following operations on the 5-bit signed numbers using 2's complement representation system. Also indicate whether overflow has occurred

   i) (-10) + (-13) ii) (10) - (+4) iii) (-3) + (-8) iv) (-10) - (+7)
4. Explain different addressing modes with examples.
5. Explain **basic instruction types.** Show how an operation C=A+B can be implemented in a computer by using 1) three address instruction 2)two address instruction 3) one address instruction Explain
6. Explain **basic input and output opearations**?differenciate between programmed controlled I/O and memory mapped I/o
7. What is a stack frame and frame pointer? Explain commonly used layout information in a stack subroutine frame.
8. Explain shift and rotate instructions with examples for a 16 bit number
9. Briefly explain about encoding machine instructions with examples
10. Define subroutine. explain subroutine linkage using link registers with sample program as an example

Module 2

11. Define and explain the following
    a. Interrupt and Interrupt service routine
    b. Vectored interrupts
    c. Interrupt nesting
    d. Interrupt enabling and disabling

12. Explain with a neat diagram the working of daisy chain with multiple priority levels and multiple devices in each level
13. Discuss the different schemes available to disable and enable interrupts
14. Explain bus arbitration, explain in detail any one approach of bus arbitration
15. Define memory mapped I/O and I/O mapped I/O with examples explain how interrupt request from several IO devices can be communicated to a processor through a single INTR line
16. What are the different methods of DMA and DMA controllers? explain them in brief
17. With a block diagram explain how keyboard is connected to processor.
18. With a block diagram explain the printer interfaced to processor.
19. Explain the serial port and serial interface.
20. Explain PCI bus
21. List SCSI bus signal with their functionalities and controllers.
22. Explain the tree structure, protocols, addressing scheme and architecture of USB.

Module-3

1. Explain internal organization of 16 Megabit DRAM chip configured as 2M*8 cells Also explain how it can be made to work fast in fast page mode.
2. Explain the working of 16-megabyteDRAM chip configured as 1M x 16 memory chip.
3. Discuss the internal organization of a 1K x 1 memory chip
4. Explain the read and write operations of a static RAM cell and CMOS cell
5. Differentiate between SRAM and DRAM giving 5 key differences.(With a cell diagram).
6. With a block diagram, explain the organization of 8M x 32 memory using 512 K x 8 memory chips
7. Explain synchronous DRAM with a neat diagram.
8. With figure analyse the memory hierarchy in terms of speed cost and size
9. What is cache? With block diagram explain Direct, Associative and set-associative mapping between cache and main memory.
10. Briefly explain any four non-volatile memory concepts(ROM, PROM, EPROM, EEPROM, Flash)
11. Discuss in detail any one feature of memory design that leads to improved performance of computer. (memory Interleaving, Cache hit and miss)
12. Calculate the average access time experienced by a processor if cache hit rate is 0.88. miss-penalty is 0.015 milliseconds and cache access time is 10 microseconds.
13. Write short notes on Hard disk/ Magnetic Disk, CD/DVD, Magnetic Tapes.
14. What is virtual memory? Explain simple method of translating virtual address into physical address.
15. Define the following:
    a. Memory Latency
    b. Memory bandwidth
    c. Hit rate
    d. Miss-penalty
    e. Memory access time
    f. Memory cycle time

g. Random access memory
h. Static memories
i. Seek time
j. Latency
k. Access time

Module- 4

1. Design 4 bit carry look ahead logic and explain how it is faster them 4 bit ripple adder. Also explain addition/subtraction Logic unit.
2. Explain with figure the design and working of a 16-bit carry look ahead adder built from 4-bit adder. (FAST ADDERS)
3. Explain booths algorithm apply booths algorithm to multiply signed numbers +13 and -6.
4. Write the circuit diagram and algorithms for restoring and non-restoring division methods.
5. Perform division of 8 by 3 using non restoring division method.
6. Perform division of 16 by 4 using restoring division method
7. Explain and Multiply (+14) and (-6) using Bit-Pair Recording Method
8. List out the rules for addition, subtraction, multiplication and division of floating point numbers. Explain with circuit diagram
9. Explain IEEE standard for Floating point numbers.

OR

Explain normalization, excess - exponent and special values with respect to IEEE floating point representation.

Module-5

1. Draw and explain the single bus organization of the data path inside a processor
2. Explain the control sequence for execution of
    1) Add (R3) R1
    2) Mul R1, (R2)
    3) Add #6, r1
    4) an unconditional branch instruction
3. Explain with block diagram the basic organization of a micro programmed control unit to support conditional branching in the micro program.
4. Write the control sequence for the instruction Add R1, R2, R3 with a neat diagram using Three bus organization (Multiple Bus Organization).
5. Explain with a neat diagram micro-programmed control method for design of control unit using micro-instructions.
6. With a neat diagram explain hardwired control unit show the generation Zin and End control signals
7. With a neat diagram explain complete processor.
8. Briefly explain the Diagram of camera and Microwave Oven. Define Embedded System.
9. Explain the organization of a simple microcontroller and discuss some features that may be used in practice.
10. Explain the structure of general purpose multiprocessor in detail.