COA UNIT 1 Notes

Computer Organization & Architecture (Dr. A.P.J. Abdul Kalam Technical University)

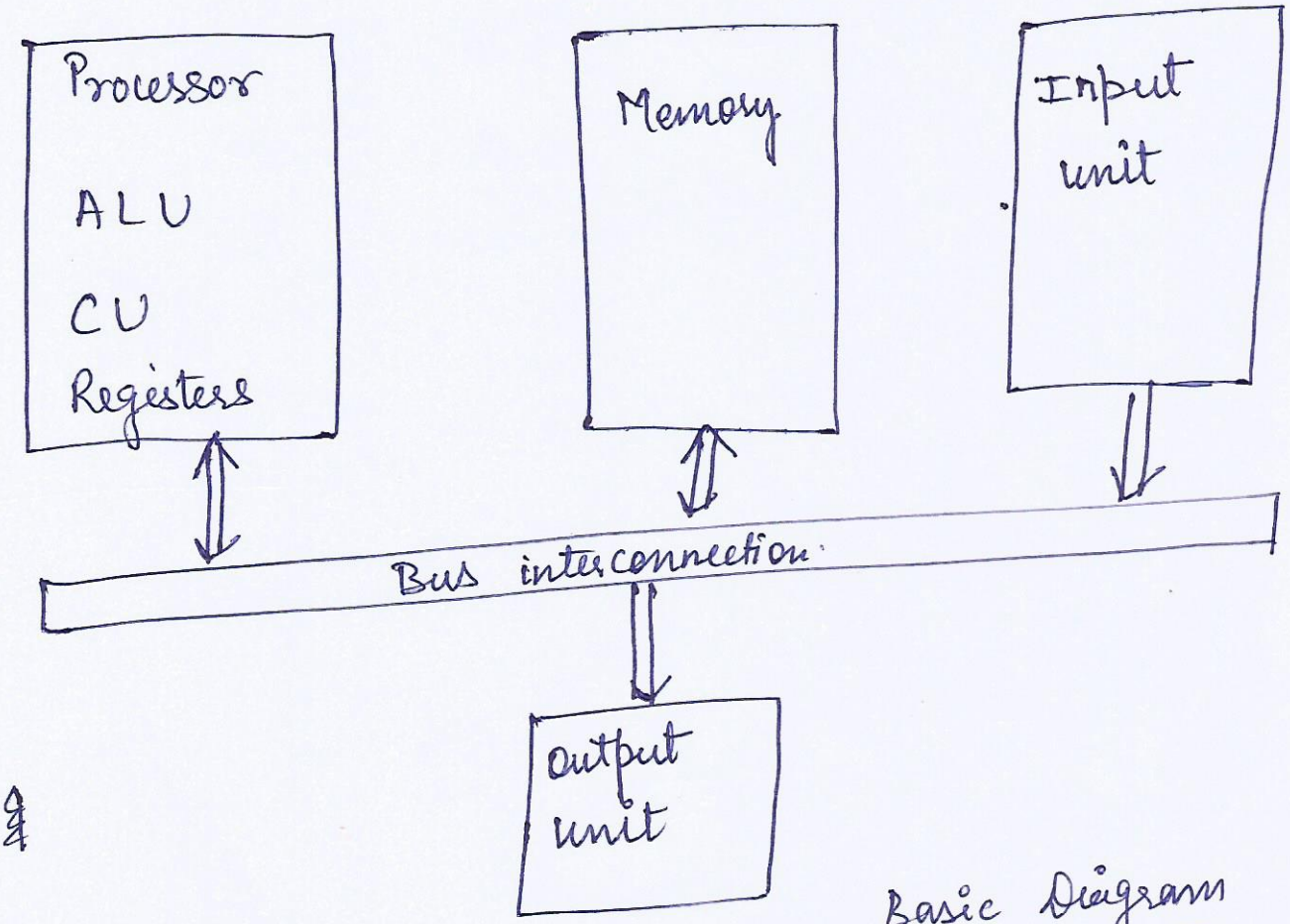
COA Unit-1 Notes

Minakshi

Functional units and their interconnections:

→ Functional units of a compiler / digital system are:

- → Input unit
- → Output Unit
- → Central Processing Unit (CPU)
- → Memory
- → Bus structure.



Basic Diagram

Note: Theory about these components are given in "Fundamental of COA" Notes and Unit 1 Notes (about bus)

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Buses, bus Architecture, types of buses and bus arbitration

Bus:

→ Bus is a communication system that helps data transfer between different modules of the computer.

→ A bus is a communication pathway connecting two or more devices.

→ A bus is a group of electrical lines/wires that carry computer signals/bits.

→ A bus consists of multiple lines. Each line is capable of transmitting signals representing 1 or binary 0.

> A bus that connects major computer components (processor, memory, I/O) is called a system bus.

→ On any bus lines can be classified into three function groups:-

1. Data Bus
2. Address Bus
3. Control Bus.

Data Bus:- Data lines provides path for moving data between components.

- Bidirectional

→ Width of databus is number of lines in data bus.

→ Each line can carry only one bit at a time.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Address Bus: (Unidirectional)

→ The address lines are used to designate (know) the source or destination of the data on the data bus.

For example: if the processor wishes to read or write a memory word, it puts the address of desired word on the address lines.

The width of address bus determines the maximum possible memory capacity of the system.

control Bus: (Bidirectional)

- The control lines are used to control the access to and the use of the data and address lines.
- control signals transmit both command and timming information

Timing signals ⇒ indicate the validity of data and address information

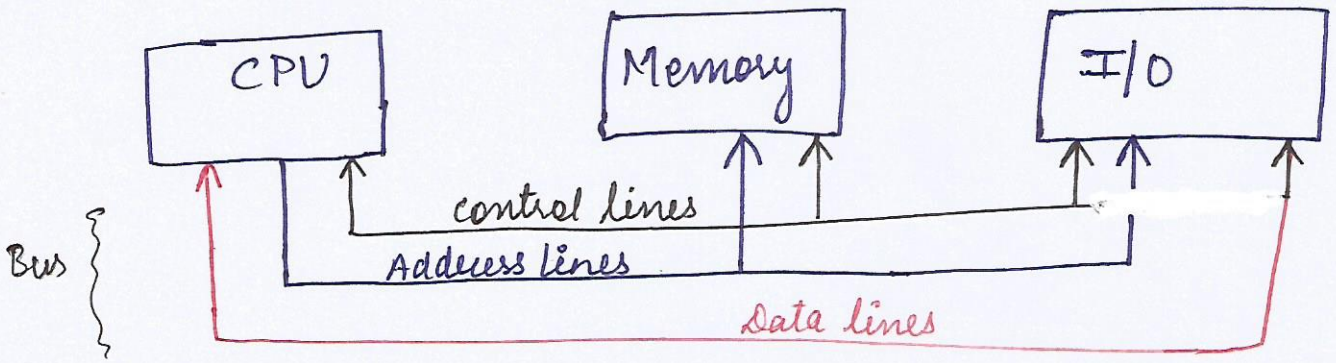
command signals ⇒ specify operations to be performed.

- Typical control lines include:-
  - Memory write
  - Memory Read
  - I/O write
  - I/O Read
  - Transfer Ack
  - Bus Request
  - Bus Grant
  - Interrupt Request
  - Interrupt Ack
  - clock
  - etc.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

4.



Bus interconnection scheme.

Types of Buses:

- Dedicated
- Multiplexed

- Dedicated bus line is permanently assigned either to one function or to a physical subset of computer components.
- Example of functional dedication :- the use of separate dedicated address and data lines.
- Multiplexed: Address and data information may be transmitted over the same set of lines using an Address Valid control line.

For example:

At the beginning of a data transfer, the address is placed on the bus and Address valid line is activated. ~~At this point~~ The address is then removed from the bus and the same bus are used for subsequent read or write data transfer. (this known as time multiplexing)

Downloaded by Sangam gupta (sangamg7318@gmail.com)

5/

Bus Arbitration:

→ More than one module may need control of the bus e.g CPU and DMA controller.

e.g. I/O module may need to read or write directly to the memory without sending the data to the processor.

The process by which multiple requests are recognized and given priority given to one of them is called arbitration

- Arbitration can be — Centralized / localized
  — Decentralized / Distributed

→ In a centralized scheme, a single hardware device, referred to as a bus controller or arbiter, is responsible for allocating time on the bus. The device may be a separate module or part of the processor.

- In distributed scheme, there is no central controller. Rather, each module contains access control logic and the modules act together to share the bus.

- With both methods of arbitration, the purpose is to designate one device, either the processor or an I/O module, as master. The master may initiate a data transfer (e.g. read or write) with some other device, which acts as slave for this particular exchange.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

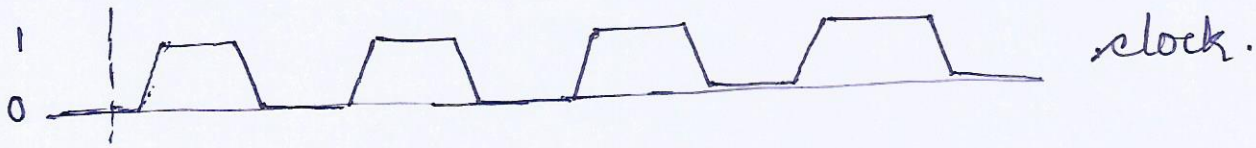
Timing

Timing Timing Refers to the way in which events are coordinated on the bus.

- Buses use — ① Synchronous Timing
  — ② Asynchronous Timing

Synchronous Timing:

- With synchronous timing the occurrence of the events on the bus is determined by a clock.
- The bus includes a clock line upon which a clock transmits a regular sequence of alternative 0's and 1's of equal duration. A single 1-0 transmission is referred to as a clock cycle or bus cycle and defines a time slot.
- All events starts at the beginning of a clock cycle.



Downloaded by Sangam gupta (sangamg7318@gmail.com)

Asynchronous Timing:

→ with asynchronous timing, the occurrence of one event on a bus follows and depends on the occurrence of a previous event.

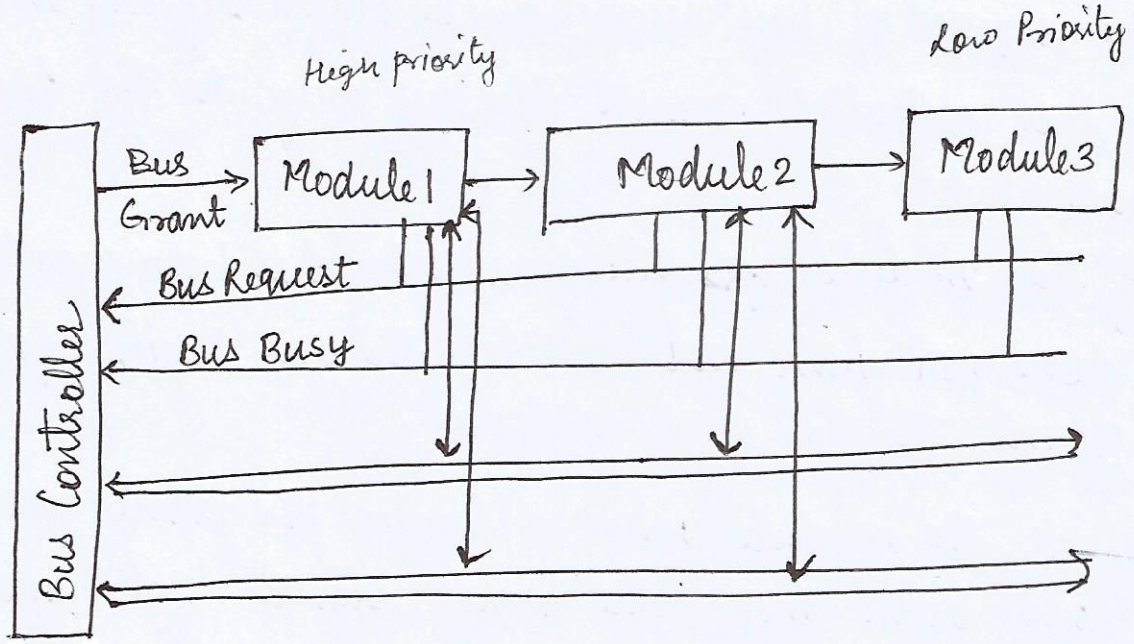
→ No clock.

This document is available free of charge on studocu

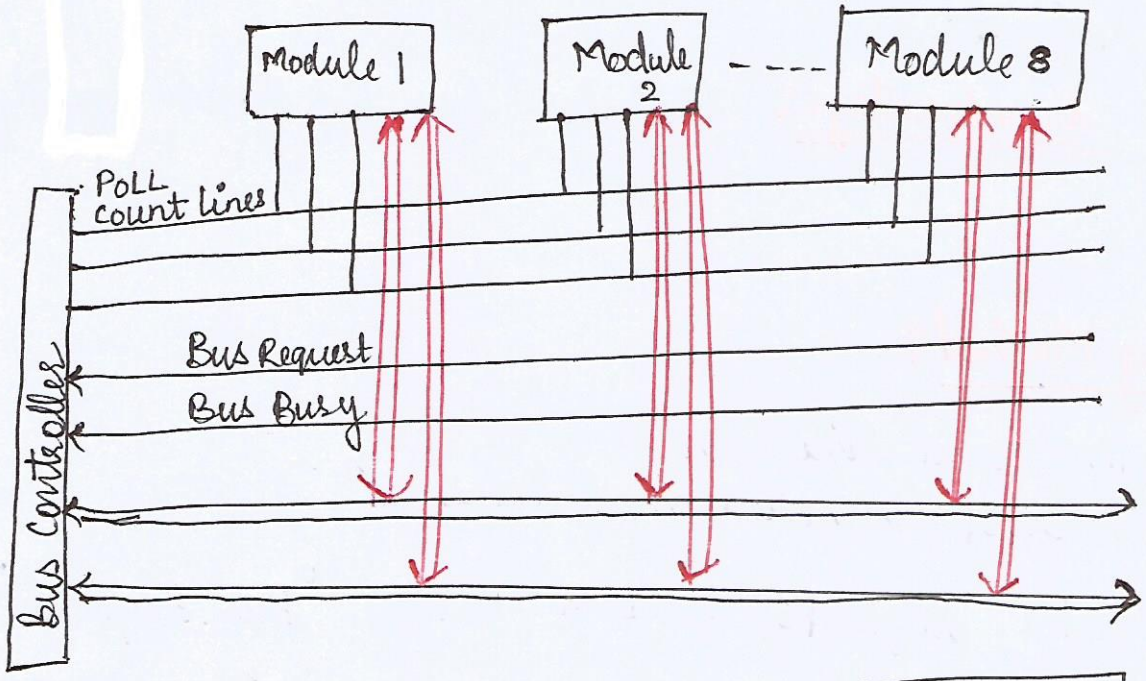
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Bus Arbitration Techniques:

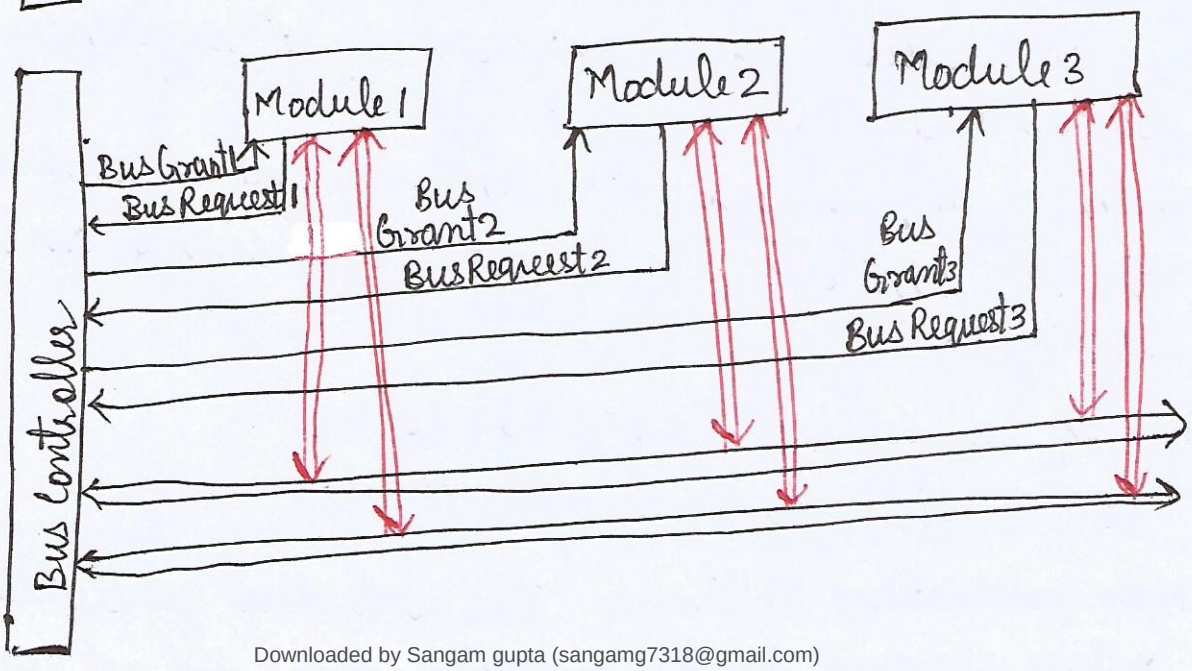
Daisy chain



Polling



Independent Requesting



Downloaded by Sangam gupta (sangamg7318@gmail.com)

Daisy-chain Method or Serial Method:

⇒ The request of bus usage can be placed by any module connected.

⇒ The bus controller grants the bus only to the first module connected.

⇒ ⇒ if the module ~~granted bus~~, who has been granted bus to, ~~does not~~ have not requested the bus, will forward the grant to the next module.

⇒ if the module who is having the grant wants to use bus, makes the busy signal 'set' and use the bus. After using bus, the module unset the busy signal.

Disadvantage: Modules has the fixed Priority.

Polling Method:

Polling count line = n

Max Number of connected modules ⇒ 2^n.

⇒ Modules requests the bus using 'Bus Request' line.

⇒ Bus controller places any one of 2^n sequences on the poll count lines.

⇒ The module, for which polling sequence is allocated by the bus controller, can set busy signal and use the bus.

⇒ Priority is set by the bus controller.

⇒ It is time consuming method.

⇒ Non-productive Polling ⇒ Bus controller generated the polling sequence for the module which did not request the bus

studocu
This document is available free of charge on
Downloaded by Sangam gupta (sangamg7318@gmail.com)

10/

Independent Request Arbitration Technique:

→ In this method, every module has connected to the bus controller by separate 'Bus grant' and 'Bus Request'

⇒ Individual module can request to the bus controller by its own 'Bus Request' line. And, Bus controller can grant the bus to the module which has requested for the bus.

⇒ But this method increases the cost of the system.

⇒ Better Performance.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

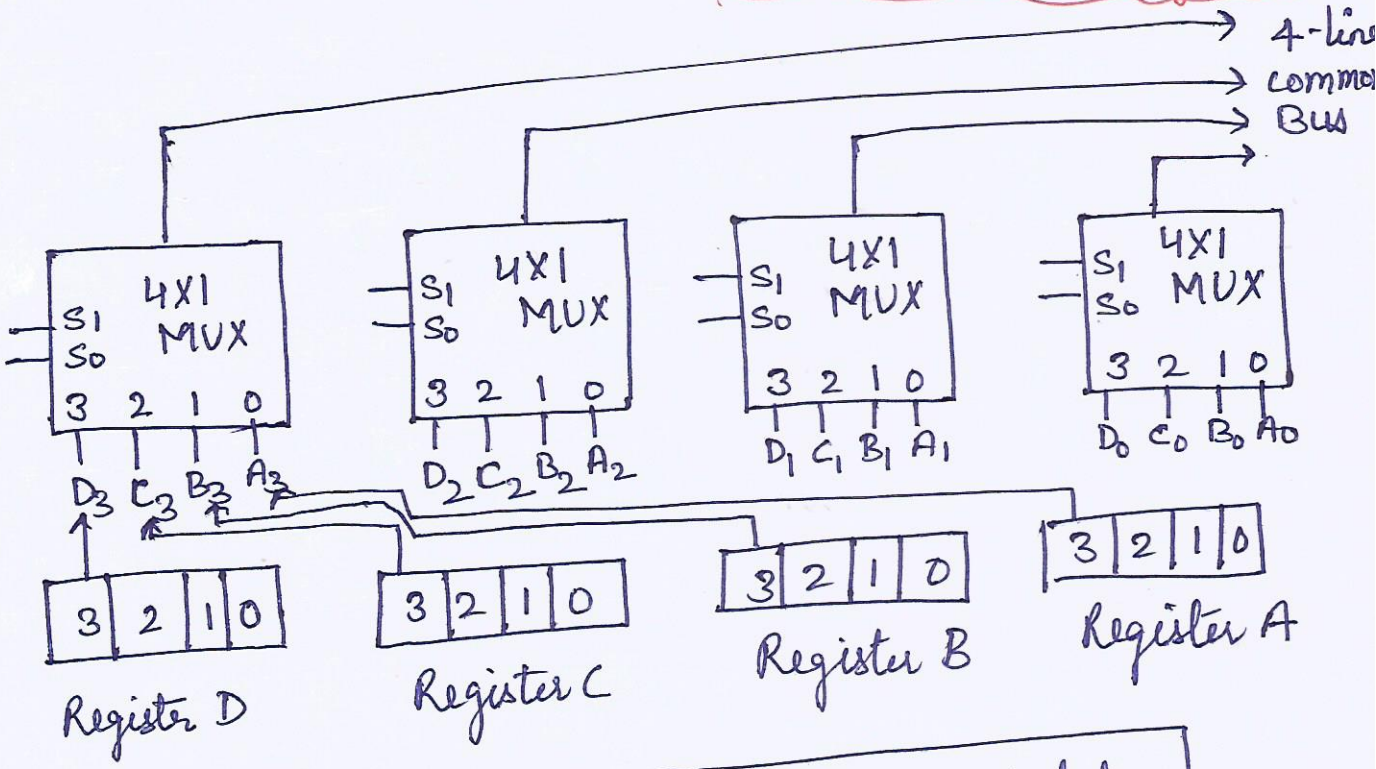
Bus and Memory Transfer:

→ A typical digital computer has many registers.
→ A path must be provided to transfer information from one register to another.
→ common bus structure
  → using Multiplexer.
  → Using Tristate Buffer.

Bus system using Multiplexer:

Example ⇒ For four registers.

Number of multiplexers = Size of register.
Size of multiplexes = Number of registers



Function Table:

S_1	S_0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Memory Transfer:

~~Memory Read: The transfer of information to be stored into memory~~

Memory Read: The transfer of information from memory to the outside environment is called a read operation

Read: DR ← M[AR]

DR ⇒ Data Register

M[AR] ⇒ memory location specified by Address Register AR.

Memory Write: The transfer of new information to be stored into the memory is called a write operation

Write: M[AR] ← DR

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Registers

- Registers are used to store data temporarily

→ The register in CPU performs two roles

- User Visible Register
  - General Purpose Registers
  - Data Register
  - Address Register
  - Condition codes
- control and Status Registers
  - Program Counter
  - Instruction Register
  - MAR
  - MDR

User Visible Register:

- A user visibe register may be referenced by means of machine language that the processor executes.
- categories – General Purpose
  - Data Register
  - Address Register
  - Condition Codes

General Purpose Registers:

- can be used by the programmer.
- contains operands for any opcode.
- can be used for addressing functions (e.g. Register indirect, displacement).

Data Register: Data Register may be used only to hold the data

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Address Register:

Address Register may be devoted to a particular addressing modes. Examples →
- Segment Pointer
- Index Register
- Stack Pointer.

- Segment Pointer holds the address of the base of the segment in segmented addressing.
- Index Registers are used for indexed addressing and may be auto indexed.
- Stack Pointer points to the top of the stack. This allows implicit addressing that is push, pop and other stack instructions need not ~~the~~ contain an explicit stack operand.

Condition codes (Also referred to as Flags):

condition codes are the bits set by the processor hardware as the result of operation.

Example:
- positive or negative result
- Zero Result
- Overflow
- etc.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

control and status Register:

→ Registers employed to control the operation of processor.
→ Not visible to the user.
→ Four Registers are essential to instruction execution

1. Program counter
2. Instruction Register
3. Memory Address Register (MAR)
4. Memory Data Register (MDR)

- Program counter contains the address of the instruction to be fetched
- Instruction Register contains the instruction most ~~recently~~ recently fetched.

→ MAR contains the address of a location in the memory.
→ MDR or MBR (Memory Buffer Register) contains a word of the data to be written to memory or the word most recently read.

PSW : Program status word :

Many processors include a set of registers known as PSW, that contain the status information
- condition code and other status information

Sign: contains the sign bit of the result.
Zero: set when the result is zero.
Carry: set if an operation resulted in a carry into or borrow out of a higher order bit.

This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Equal: set if a logical compare result is equal.

Overflow: used to indicate arithmetic overflow.

Interrupt Enable/disable: used to enable or disable interrupts

Supervisor: indicates whether the processor is executing supervisor or user mode.
certain priviledged instructions can be executed only in supervised mode, and certain memory areas can be accessed only in supervisor mode.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Topic

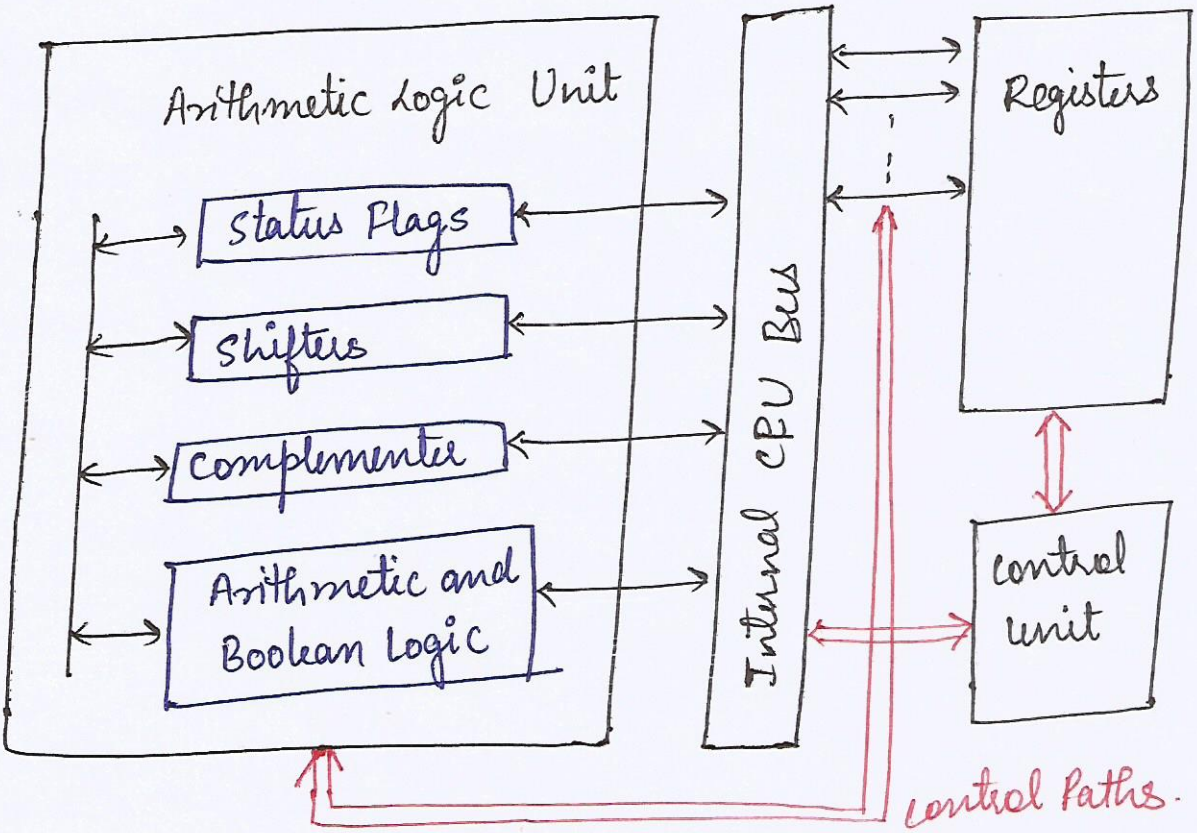
Processor Organization:

Processor contains ALU, CU and Registers

→ ALU ⇒ The ALU (Arithmetic Logic Unit) does the actual computation or processing of the data

→ CU: control unit:- CU controls the movement of data and instruction into and out of the processor and controls the operation of ALU.

→ Registers: minimal internal memory.



Internal structure of CPU

- Processor Organization means how the components of processor are connected and accomplish their tasks.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

A processor does the following things.

- Fetch Instruction
- Interpret instruction
- Fetch Data
- Process Data
- Write Data

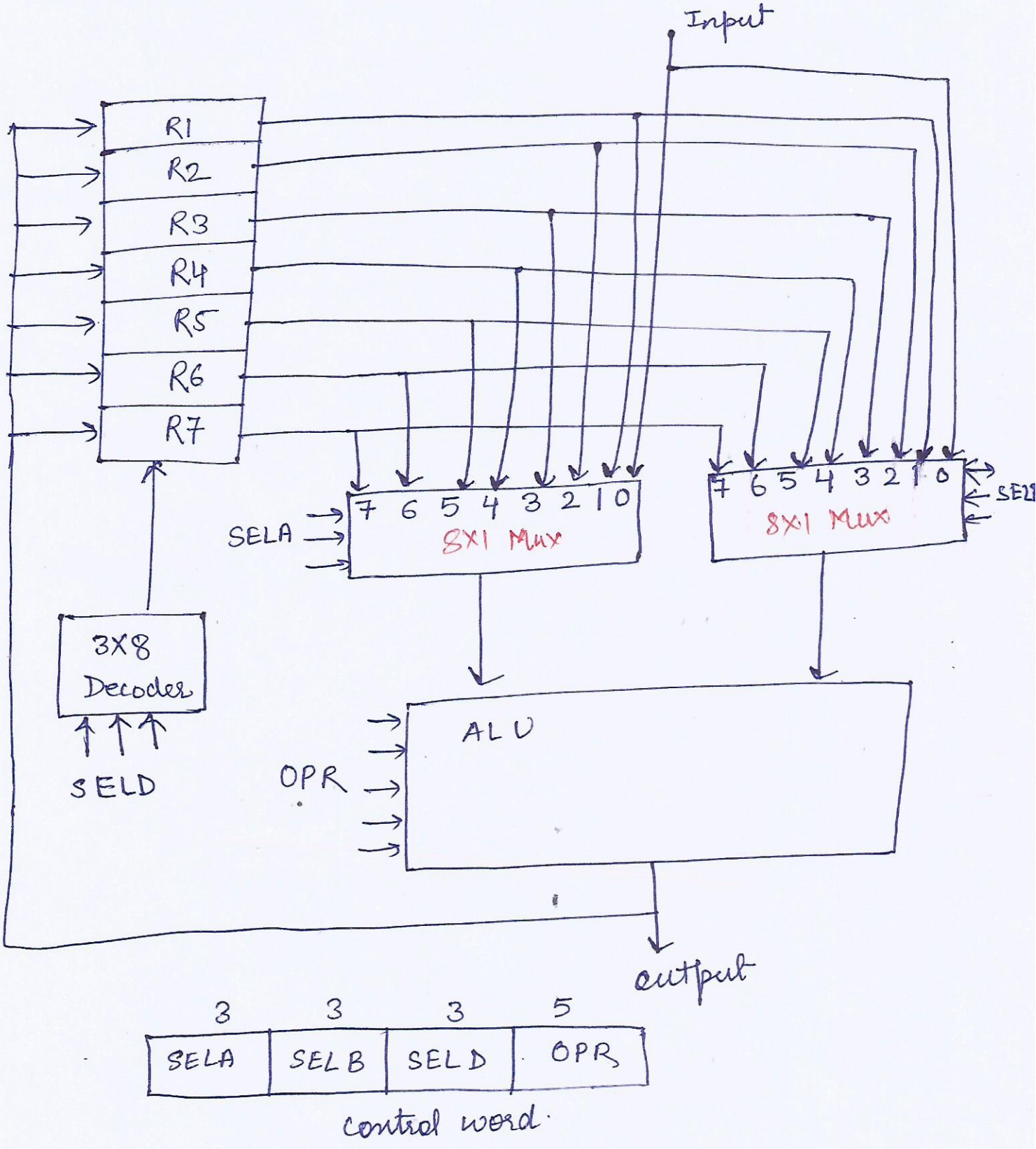
- **Fetch Instruction:** The processor reads an instruction from memory.
- **Interpret Instruction:** The instruction is decoded to determine what action is required.
- **Fetch data:** The execution of an instruction may require reading data from memory or an I/O module
- **Process Data:** The execution of an instruction may require performing some arithmetic or logical operation on the data.
- **Write Data:** The result of an execution may require writing data to memory or I/O module.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Topic

General Register Organization :

→ A bus organization for seven CPU registers.



control word.

Register set with common ALU

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

- The output of each Register is connected to two multiplexers to form the buses A and B.
- The selection lines in each multiplexer selects one register or input for the particular bus.

→ The buses A and B form the inputs to a common ALU.

→ The operation selected in the ALU determines the arithmetic or logic microoperation that is to be performed.

→ The result is available for output data and goes into the inputs of all registers. Register for output is selected by a decoder.

→ The decoder activates one of the register load inputs.

Register Selection:

Binary Codes	SELA	SELB	SELD
000	input	input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Table of operations performed by ALU

	OPR select	Operation
0	00000	Transfer A
1	00001	Increment A
2	00010	Add A+B
5	00101	Subtract A−B
6	00110	Decrement A
8	01000	AND A, B
10	01010	OR A, B
12	01100	XOR A, B
14	01110	complement A
16	10000	shift Right A
24	11000	shift Left A

This table specifies some selected operants only ⇒ 11

five bits can represent total 32 operation

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Topic.

Stack Organization:

→ stores information is LIFO order (Last In First Out)

→ The register that holds the address of the stack is called Stack Pointer.

→ Stack Pointer's value always points to the top element.

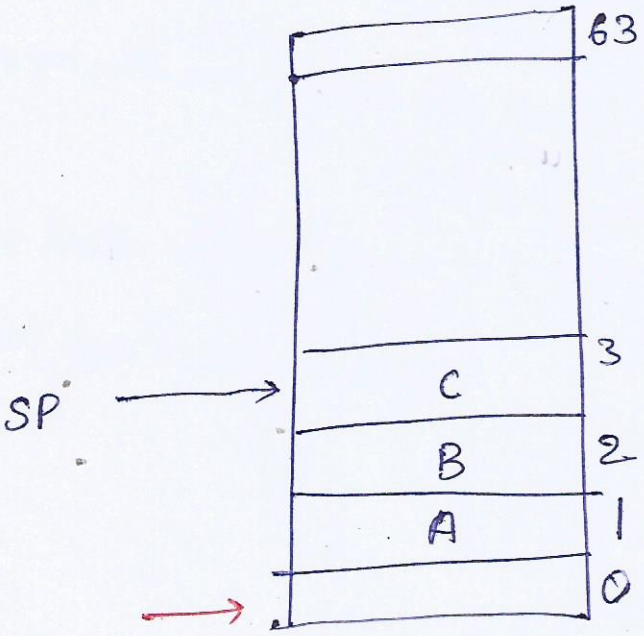
→ Two operations on stack ① Push
② Pop

→ Push ⇒ to insert an element in the stack
Pop ⇒ to delete an element from the stack

⇒ Two status Full ⇒ set to 1 when stack is full
Empty ⇒ set to 1 when stack is empty

⇒ Stack can be ⇒ collection of memory words or registers

⇒ Example 64 - word register stack



Full
Empty

Full → 1 ⇒ Full
0 ⇒ not full

Empty → 1 Empty
0 not empty

Downloaded by Sangam gupta (sangamg7318@gmail.com)

initially SP ← 0
EMPTY ← 1
FULL ← 0

if (Full = 0) ⇒ new item is inserted with push operation
if (EMPTY = 0) ⇒ an item can be deleted from stack with popoperation

Push operation :-

SP ← SP + 1 increment stack Pointer
M[SP] ← DR write item on the top of stack.
if (SP = 0) then FULL ← 1
EMPTY ← 0

M[SP] denotes the memory word specified by the address presently available in the SP.

⇒ The first item is stored in stack at address 1.
The last item is stored in the stack at address 0.

Pop Operation :

A new item is deleted from the stack if stack is not empty

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

This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)

ADDRESSING MODES:

Two issues: ① How is the address of an operand specified?
② How the bits of an instruction organized to define the operand addresses and operation of that instruction.

The operation field of an instruction specifies the operation to be performed. This operation must be executed on some data stored in the computer registers or memory words.
The way the operands are choosen during the program execution depends on the addressing mode of the instruction

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

Most Common addressing techniques:

- Immediate Addressing
- Direct Addressing
- Indirect addressing
- Register Addressing
- Register Indirect Addressing
- Displacement → Relative Addressing
  → Base Register Addressing
  → Indexing
- Stack addressing

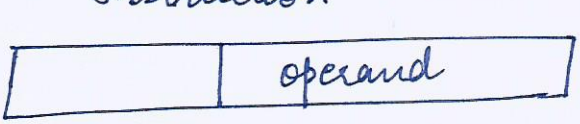
Downloaded by Sangam gupta (sangamg7318@gmail.com)

① Immediate Addressing:

→ In this addressing mode, the operand value is present in the instruction.

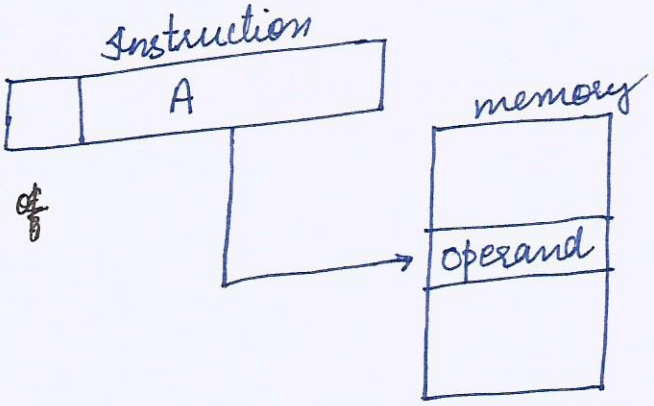
→ Advantage: no memory reference other than the instruction fetch is required to obtain the operand.

→ Disadvantage: Limited operand magnitude.



② Direct Addressing:

In this addressing mode, the address field contains the actual/effective address of the operand.



EA = A

A ⇒ content of an address field in the instruction

Advantage:

- It requires only one memory reference and no special calculation.
- Simple

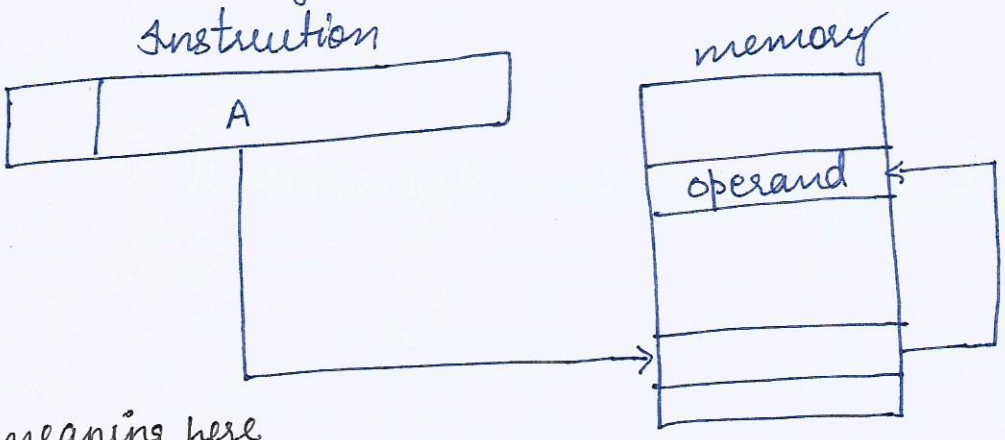
Disadvantage: Limited address space.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

(3) Indirect Addressing:

In this addressing mode, address field refers to the address of a word in the memory.



EA = (A)

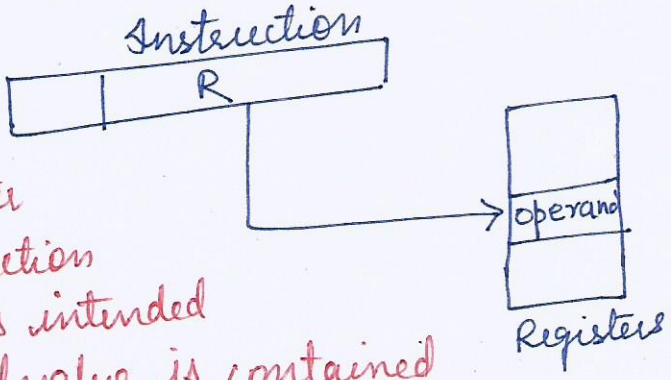
* parantheses meaning here
  ⇒ "content of"
* EA = Actual / effective address of the location containing the referenced operand.

Advantage: Large address space:
→ for a word length of N, an address space of 2^N is now available.

Disadvantage: requires two memory reference
→ one for address of operand.
→ one for to get its value (operand's)

(4) Register Addressing:

Register addressing is ~~similar~~ similar to direct addressing. The only difference is that the address field refers to a register rather than a main memory address.



EA = R

if the content of a register address field in an instruction is 5, then register R5 is intended address, and the operand value is contained in R5.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

26/ Advantages:

→ only a small address field in instruction is needed
→ no time consuming memory references.

Disadvantages:

→ very limited register address space.

"Content of the register will be operand" in Register Addressing

(5) Register Indirect Addressing:

- similar to indirect addressing
- But here, the address field of instruction specifies a register whose contents give the address of the operand in the memory.

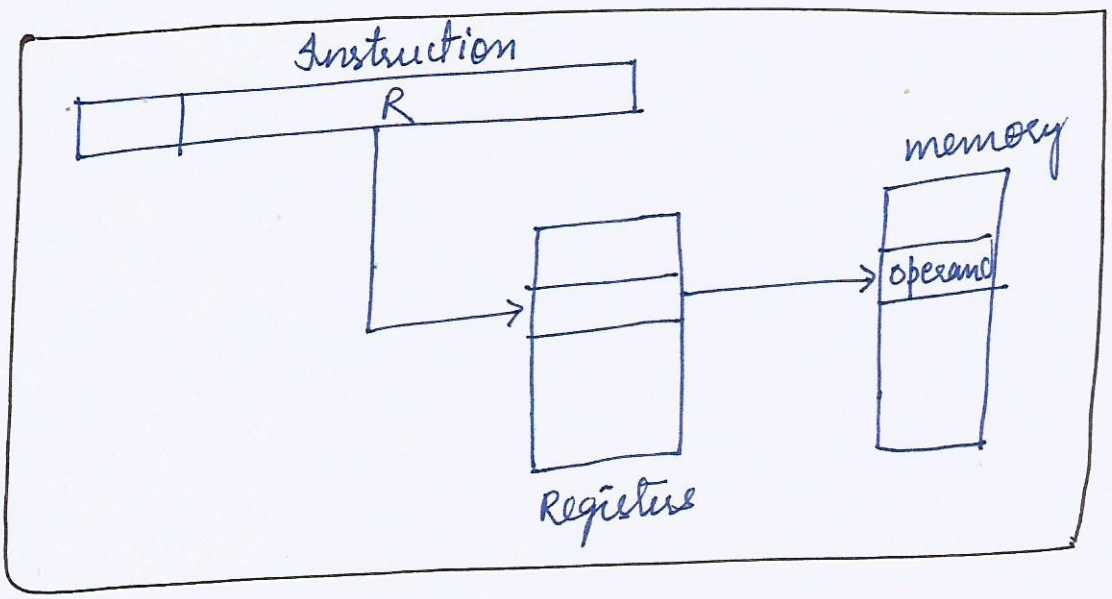
$$EA = (R)$$

Advantage:

→ fewer bits to select a register than a memory addres
→ Large address space.

Disadvantage:

- Extra memory Reference.



This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)

6 Displacement Addressing:

This addressing mode combines the capabilities of direct addressing and register indirect addressing.

$$EA = A + (R)$$

- Three most common uses of displacement addressing
  - → Relative addressing
  - → Base-register addressing
  - → Indexing

Relative Addressing:

- Also called PC-relative addressing
- In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.

Example: assuming PC = 825

address part in instruction = 24

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

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

This is 24 memory locations forward from the address of the next instruction.

- often used with branch-type instructions
- saves address bits in the instruction

Effective address = content of Program counter + address in instruction

Downloaded by Sangam gupta (sangamg7318@gmail.com)

28/

Base Register Addressing:

In this mode the content of a base register is added to the address part of instruction to get the effective address.

- A base register is assumed to hold the base address and the address part of the instruction gives a displacement relative to this base address.

Effective Address = content of Base Register + address in instruction

Indexing or Indexed Addressing mode:

- In this mode, the content of an index register is added to the address part of an instruction to obtain the effective address.
- The index register is a special CPU register that contains an index value. The address ~~field~~ of field of the instruction defines the beginning address of a data array in the memory
- Each operand in the array is stored in the memory relative to the beginning address.

Effective address = content of index Register + address in instructio

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

(7) Stack Addressing :

- A stack is a linear array of locations.
- A stack is also referred to as a push down list or last-in-first-out queue.
- LIFO concept.
- The stack mode of addressing is a form of implied addressing The machine instruction need not include memory reference but implicitly operate on the top of stack.

EA = top of the stack

Advantages: No memory reference

Disadvantage: Limited applicability.

(8) Implied Mode or Implicit mode :

- Operands are specified implicitly
- Example Instruction such as CMA. (complement Accumulator)
- In fact all register reference instructions that ~~are~~ use an accumulator are implied mode instructions.
- Zero address instructions in stack organized computer are implied mode instructions

Downloaded by Sangam gupta (sangamg7318@gmail.com)

(9) Auto increment or Autodecrement Mode:

30/

The auto increment mode is similar to register indirect mode except that the register (content) is incremented after the execution of the instruction.

Effective address = content of register + address in instruction

Now increment the content of register $R = R + 1$

In the auto decrement mode, the content of register is decremented before the execution of instruction.

$R = R - 1$

Effective address = content of registes + address in instruct

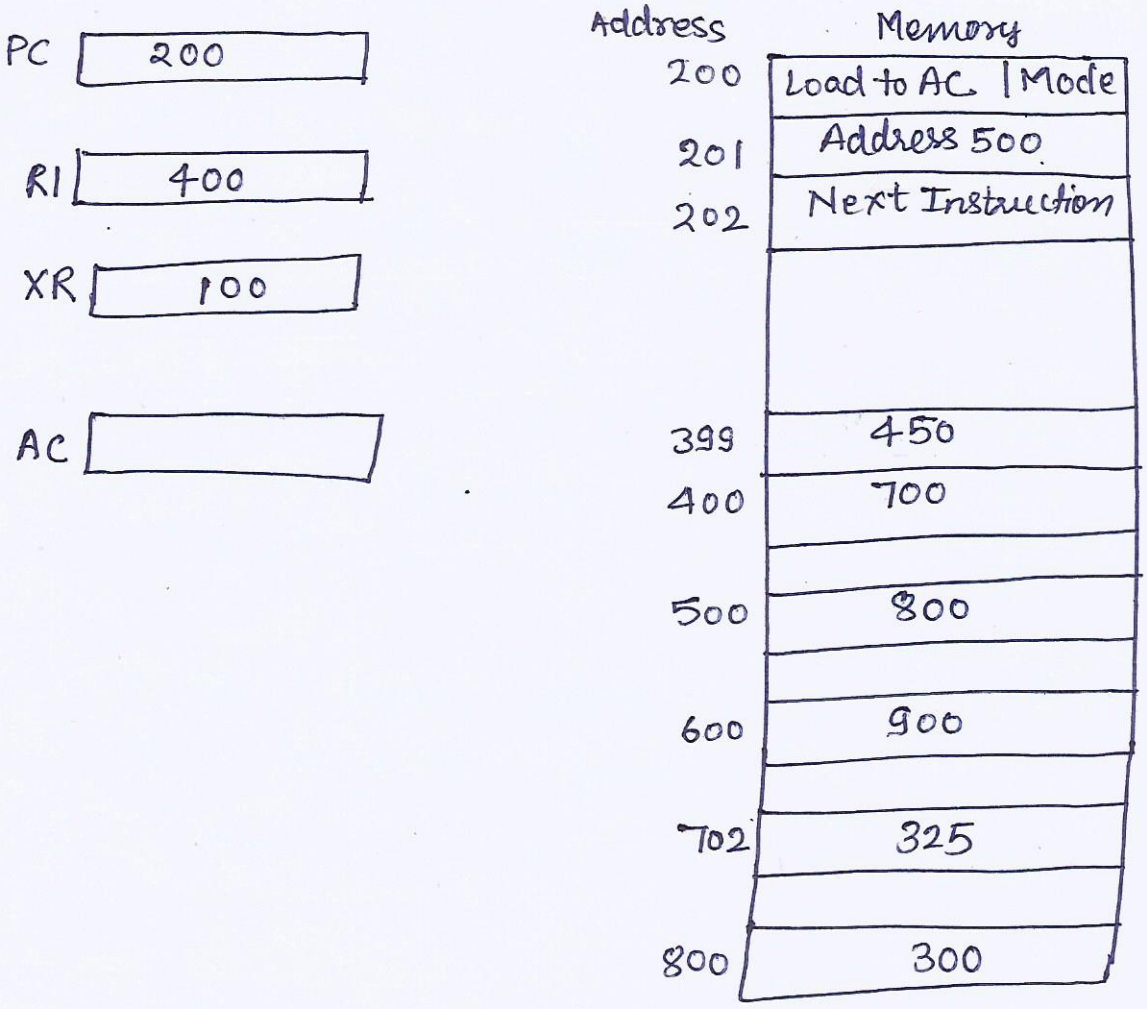
Effective address means ⇒ address of operand

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

31

Numerical Example for Addressing Modes:



⇒ The two word instruction at address 200 and 201 is a

Load to AC	Mode	Address 500

⇒ The first word of the instruction specifies the operation code and mode, and the second word specifies the address part.

→ PC (Program Counter) has the value 200 for fetching this instruction

→ The content of process register RI is 400 and the content of index register XR is 400.

→ AC (Accumulator) receives the operand after the instruction is executed.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

⇒ For each possible mode, we calculate the effective address and the operand that must be loaded into AC.

① Direct mode: ⇒ the effective address is the part of instruction.

Effective Address = 500 Operand ⇒ 800

② Immediate mode: operand is the part of instruction.

operand = 500

EA ⇒ ~~address~~ 201

③ Indirect Mode: ⇒ Effective address is stored at the address part of instruction '500'

Effective Address ⇒ 800 Operand = 300

④ Relative Mode: Effective address ⇒ content of PC + instruction address part.

EA = 202 + 500

= 702

Operand = 325

⑤ Index Mode: EA = content of index Register + instruction address part.

= 100 + 500

= 600

operand = 900

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

(6) Register Mode:

⇒ There is no effective address. content of the register R1 will be loaded to AC.

Operand = 400

(7) Register Indirect mode:

Register R1 contains the effective address.

EA = 400 . operand = 700

(8) Auto-increment Mode:

same as Register indirect mode, but the content of Register is incremented after the execution of the instruction

EA = 400, operand = 700 , R1 = 401

(9) Auto-Decremented Mode:

Same as Register indirect mode, but the content of Register is decremented before the execution of the instruction

Effective Address = 309 operand = 450

Table

	Addressing Mode	Effective address	operand
(1)	Direct	500	800
(2)	Immediate	201	500
(3)	Indirect	800	300
(4)	Relative	702	325
(5)	Indexed	600	900

Downloaded by Sangam gupta (sangamg7318@gmail.com)

	Addressing mode	effective address	operand.
(6)	Register	—	400
(7)	Register indirect	400	700
(8)	Auto increment	400	700
(9)	Auto-decrement	399	450

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

COA UNIT-2 Notes

Computer Organization & Architecture (Dr. A.P.J. Abdul Kalam Technical University)

COA Unit-2 Notes

Minakshi

Adders:

- Half Adders
- Full Adders
- n-bit Binary Parallel Adder
- Carry Look-ahead Adder (or Fast adder or speed adder).

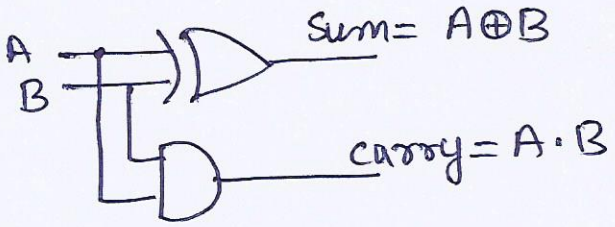
Half Adder:

Half adder is a combinational circuit that adds two bits.

Truth Table of Half Adder:

input		output	
A	B	Sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

circuit of Half adder



Full Adder:

Full adder is a combinational circuit that adds 3 bits.

Truth Table:

A	B	C	Sum	Cout
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

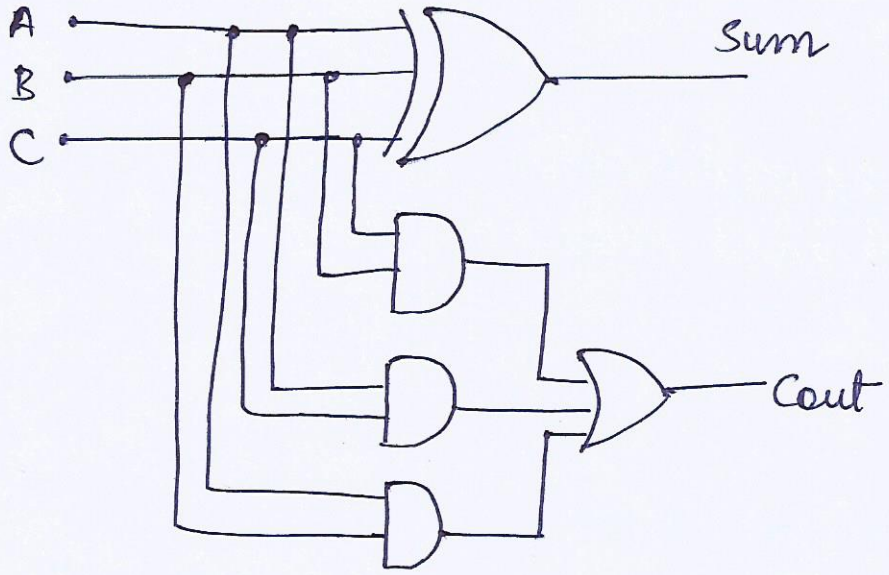
Sum= A⊕B⊕C

Cout = AB+BC+AC

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Logic diagram of Full adder:



$$C_{out} = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

Properties.

$$= \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC + ABC + ABC \quad \{\because x + x = x\}$$

$$= BC[\bar{A}+A] + AC[\bar{B}+B] + AB[\bar{C}+C] \quad \{x + \bar{x} = 1\}$$

$$= BC \cdot 1 + AC \cdot 1 + AB \cdot 1 \quad \{x \cdot 1 = x\}$$

$$\boxed{C_{out} = BC + AC + AB}$$

---

n-bit Binary Parallel Adder:

→ An n-bit binary parallel adder adds 2 n-bit numbers.

→ An n-bit binary adder uses n full adders.

Example An 4-bit binary adder adds two 4-bit binary numbers

$$A = A_3\ A_2\ A_1\ A_0$$

$$B = B_3\ B_2\ B_1\ B_0$$

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Addition

$C_3\ C_2\ C_1\ C_0=0 \Longrightarrow$ carry input

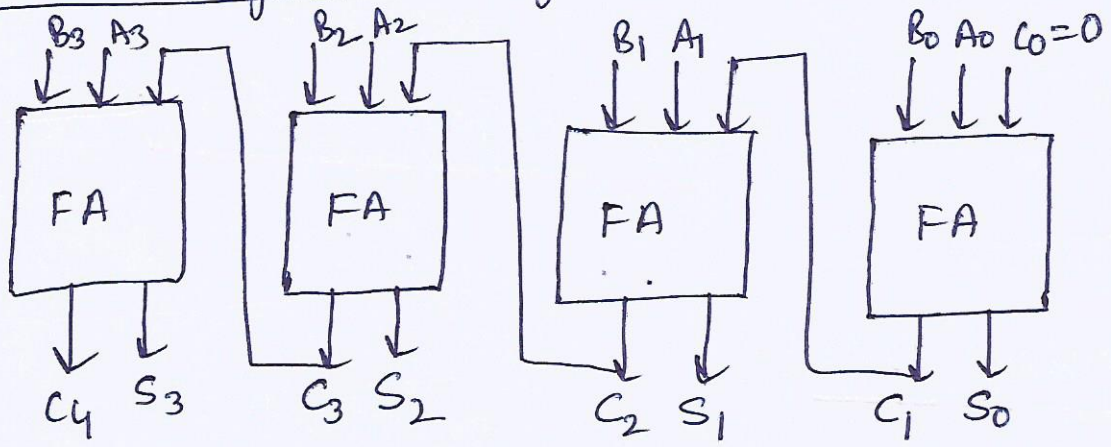
$A_3\ A_2\ A_1\ A_0$

$+\ B_3\ B_2\ B_1\ B_0$

$S_3\ S_2\ S_1\ S_0 \Longrightarrow$ sum

$C_4\ C_3\ C_2\ C_1 \Longrightarrow$ carry output.

4-bit binary adder diagram:



A 4-bit binary parallel adder consists of 4 full adders in cascade, with output carry from one full adder, connected to the carry input of the next full adder.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Carry look ahead Adder:

Drawback of n-bit parallel adder -

- The carry output of one stage (or full adder) is connected to the carry input of the next higher stage (or full adder). This carry is called ripple carry.
- Therefore, the sum and carry can not be produced until the input carry occurs.
- This lead to a time delay in addition process. This delay is known as carry propagation delay.

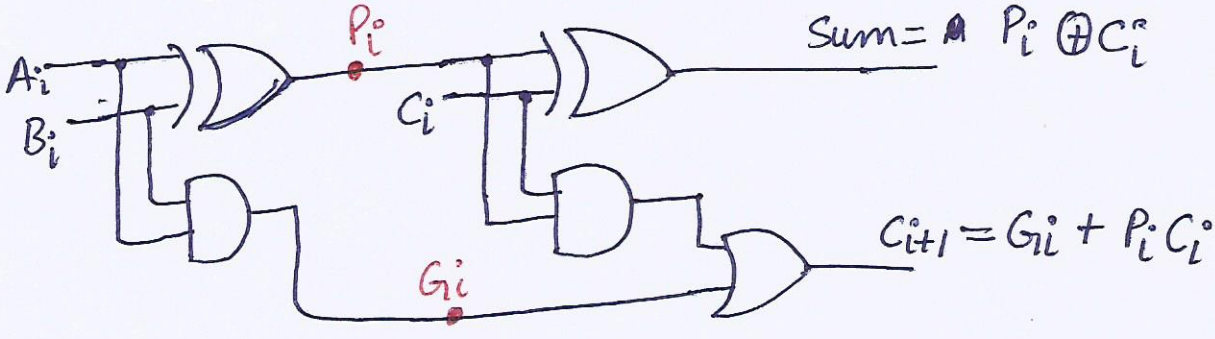
Carry - Look-ahead Adder:

This method utilizes logic gates to look at the lower order bits of augend and addend to see if a higher order carry is to be generated or not

carry look-ahead concept uses two functions

1. Carry Propagate (P_i)
2. Carry Generate (G_i)

Full adder using two half adders:



Downloaded by Sangam gupta (sangamg7318@gmail.com)

Carry Propagation $P_i = A_i \oplus B_i$

Carry Generation $G_i = A_i \cdot B_i$

Both RHS does not contain any C_i

The output $S_i = P_i \oplus C_i$

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

we can create n-bit ~~look~~ carry look-ahead adder using these two equations.

Example

4 bit carry look-ahead adder:

For 4-bit, i is 0 to 3.

To remove the dependency of C_{i+1} on C_i, we need to express C_{i+1} into A_i, B_i and C_0.

$$C_{i+1} = G_i + P_i C_i \quad \text{---------- (1)}$$

Put

$i=0$ $\quad C_{0+1} = G_0 + P_0 C_0$

$$C_1 = G_0 + P_0 C_0$$

$i=1$ $\quad C_{1+1} = G_1 + P_1 C_1$

$$C_2 = G_1 + P_1 [G_0 + P_0 C_0]$$

$i=2$ $\quad C_{2+1} = G_2 + P_2 C_2$

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

This document is available free of charge on studocu

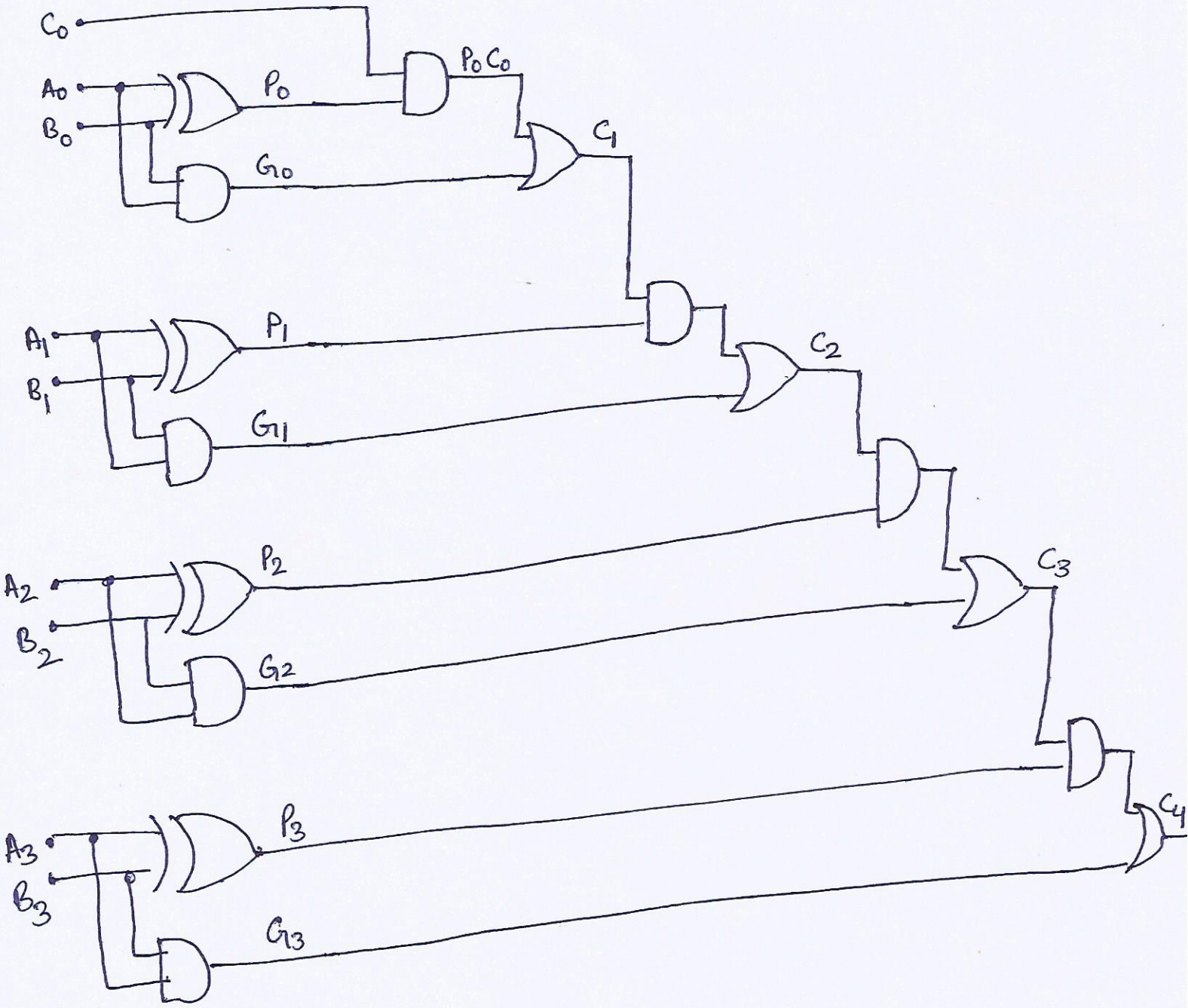
Downloaded by Sangam gupta (sangamg7318@gmail.com)

$i=3$

$$C_{3+1} = G_3 + P_3 C_3$$

$$\boxed{C_4 = G_3 + P_3 [G_2 + P_2 [G_1 + P_1 [G_0 + P_0 C_0]]]}$$

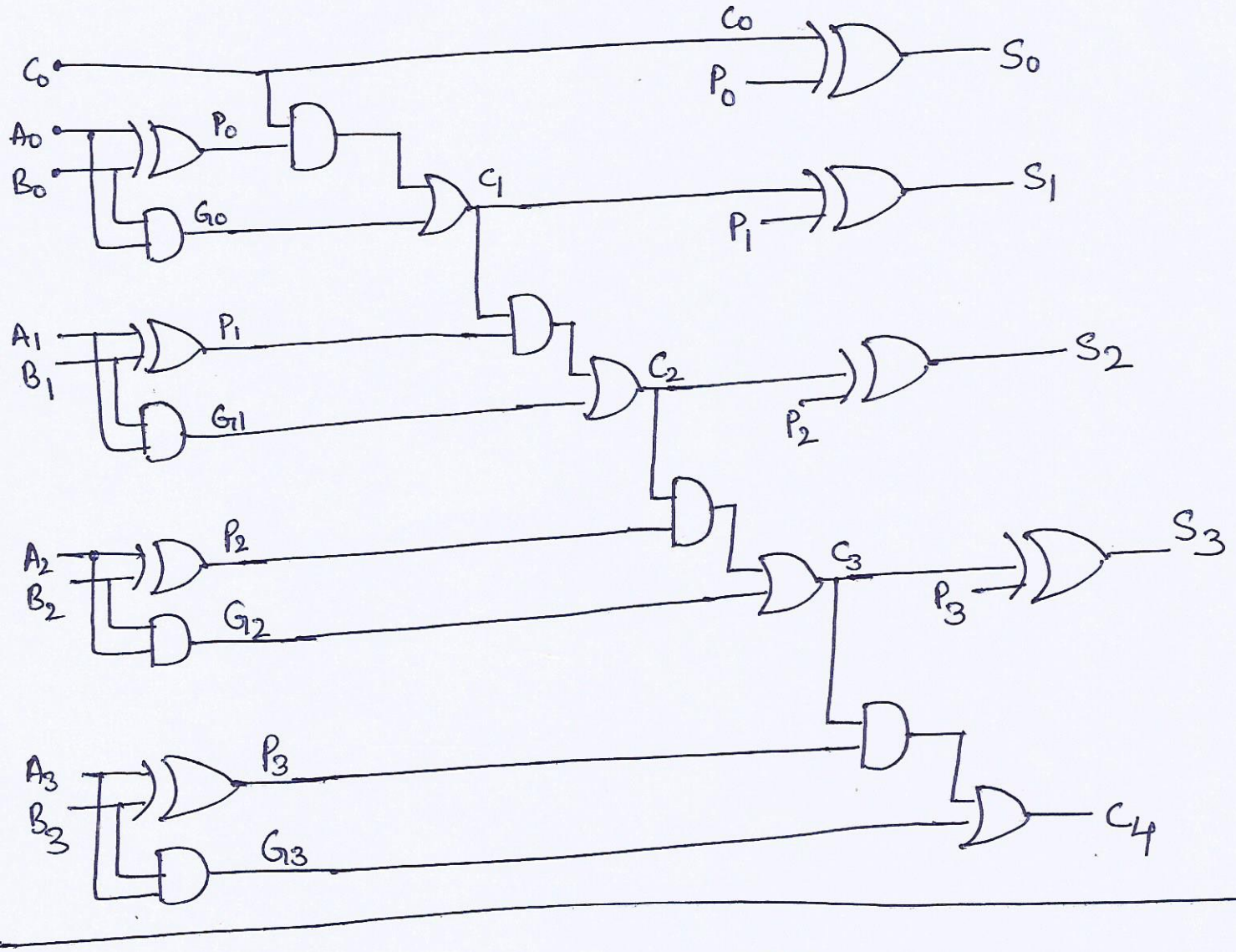
4-bit carry look-ahead ~~Adders~~ Carry Generation circuit

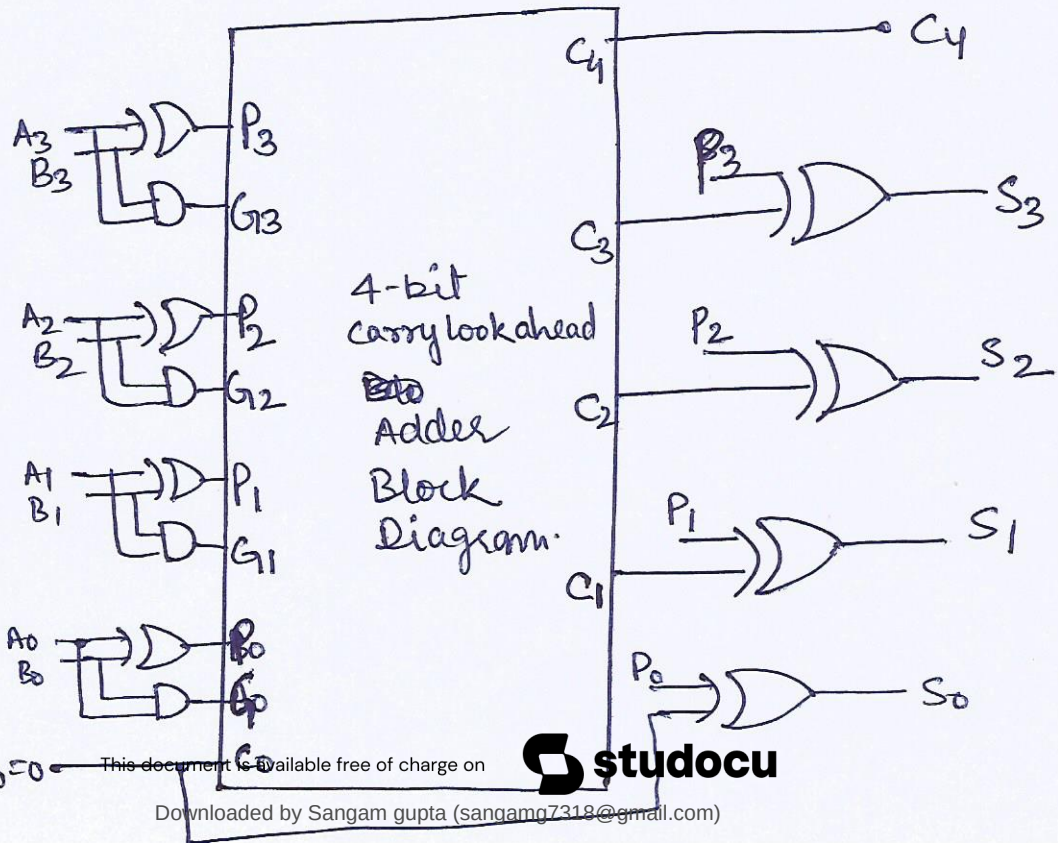


Downloaded by Sangam gupta (sangamg7318@gmail.com)

Minakshi

4 bit carry lookahead adder circuit:



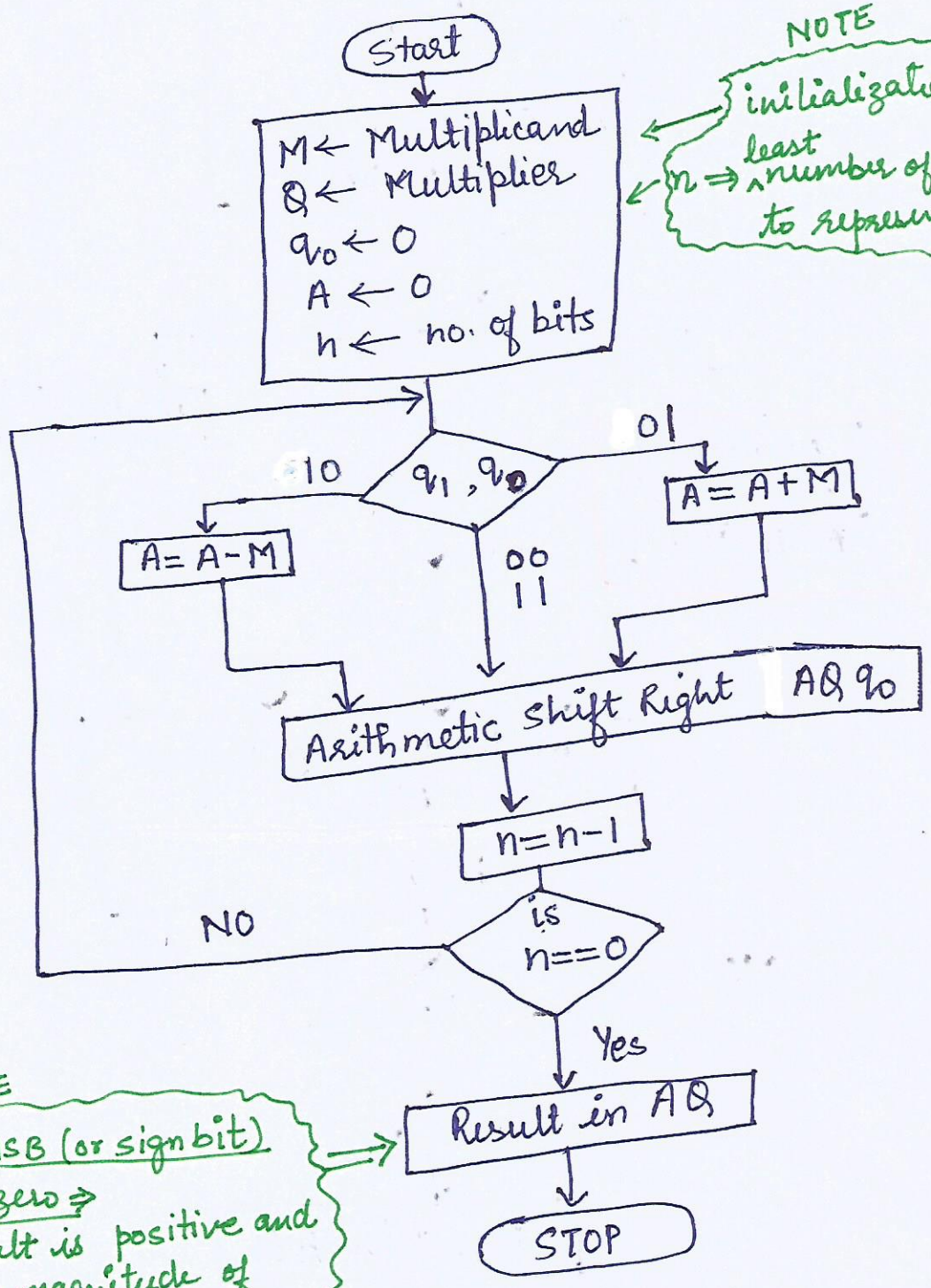


This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Minakshi

Signed Multiplication (Booths Algorithm)

Flow chart:



NOTE

if MSB (or sign bit) is zero ⇒ Result is positive and the magnitude of the product.

if MSB (or sign bit 1 Result is negative take 2's complement to get the magnitude and place negative sign

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Minakshi

Signed Booth Multiplication Example:

Example $(-7) \times (+3) = (-21)$

$M = (-7)_{10} = 2's(0111) = (1001)_2$

$-M = (0111)_2$

Tracing Table

	n	A	Q_{q1} (+3)	q_0	Action/Comment
					initialization
STEP①	4	0000	0011	0	
		0111	0011	0	A = A − M ⇒ A = A + (−M)
		0011	1001	1	ASR AQq0
		0011	1001	1	n = n − 1 n = 3
STEP②	3	0001	1100	1	ASR AQq0
		0001	1100	1	n = n − 1 n = 2
STEP③	2	1010	1100	1	A = A + M
		1101	0110	0	ASR AQq0
		1101	0110	0	n = n − 1 n = 1
STEP④	1	1110	1011	0	ASR AQq0
		1110	1011	0	n = n − 1 n = 0
	0				

Result ⇒ A Q

1110 1011

↑ sign bit is one. So the result is negative, take 2's complement to get the magnitude and place negative sign.

Final result: $(-21)_{10}$

1110 1011

↓ 1's complement

0001 0100

↓ +1

00010101 ⇒ $(21)_{10}$

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Minakshi

Array Multiplier:

Binary Multiplication is done by doing additions. Partial Products are calculated by multiplying the multiplicand by each bit of the multiplier and then summing the partial products

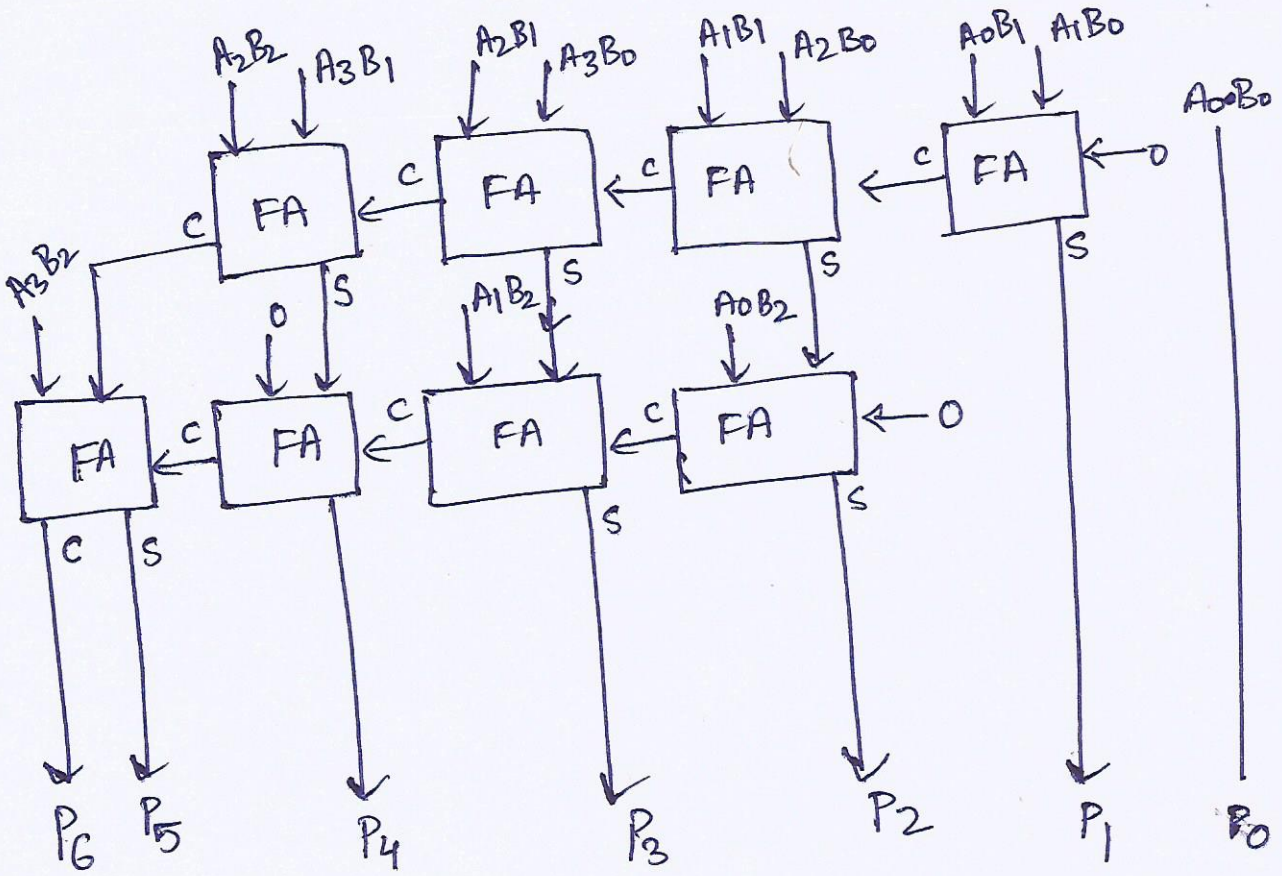
Ques: Design 4x3 binary or Array multiplier.

Given $A \Rightarrow A_3 A_2 A_1 A_0$

$B \Rightarrow B_2 B_1 B_0$

Number of bits in product $\Rightarrow 4+3=7$

$$
\begin{array}{cccc}
A_3 & A_2 & A_1 & A_0 \\
\times & B_2 & B_1 & B_0 \\
\hline
A_3B_0 & A_2B_0 & A_1B_0 & A_0B_0 \\
A_3B_1 & A_2B_1 & A_1B_1 & A_0B_1 \\
A_3B_2 & A_2B_2 & A_1B_2 & A_0B_2 \\
\hline
P_6 & P_5 & P_4 & P_3 \quad P_2 \quad P_1 \quad P_0
\end{array}
$$



Downloaded by Sangam gupta (sangamg7318@gmail.com)

Minakshi

4X4 Binary Array Multiplier:

$A = A_3\ A_2\ A_1\ A_0$

$\times\ B = B_3\ B_2\ B_1\ B_0$

A_3B_0 A_2B_0 A_1B_0 A_0B_0

A_2B_1 A_1B_1 A_0B_1 X

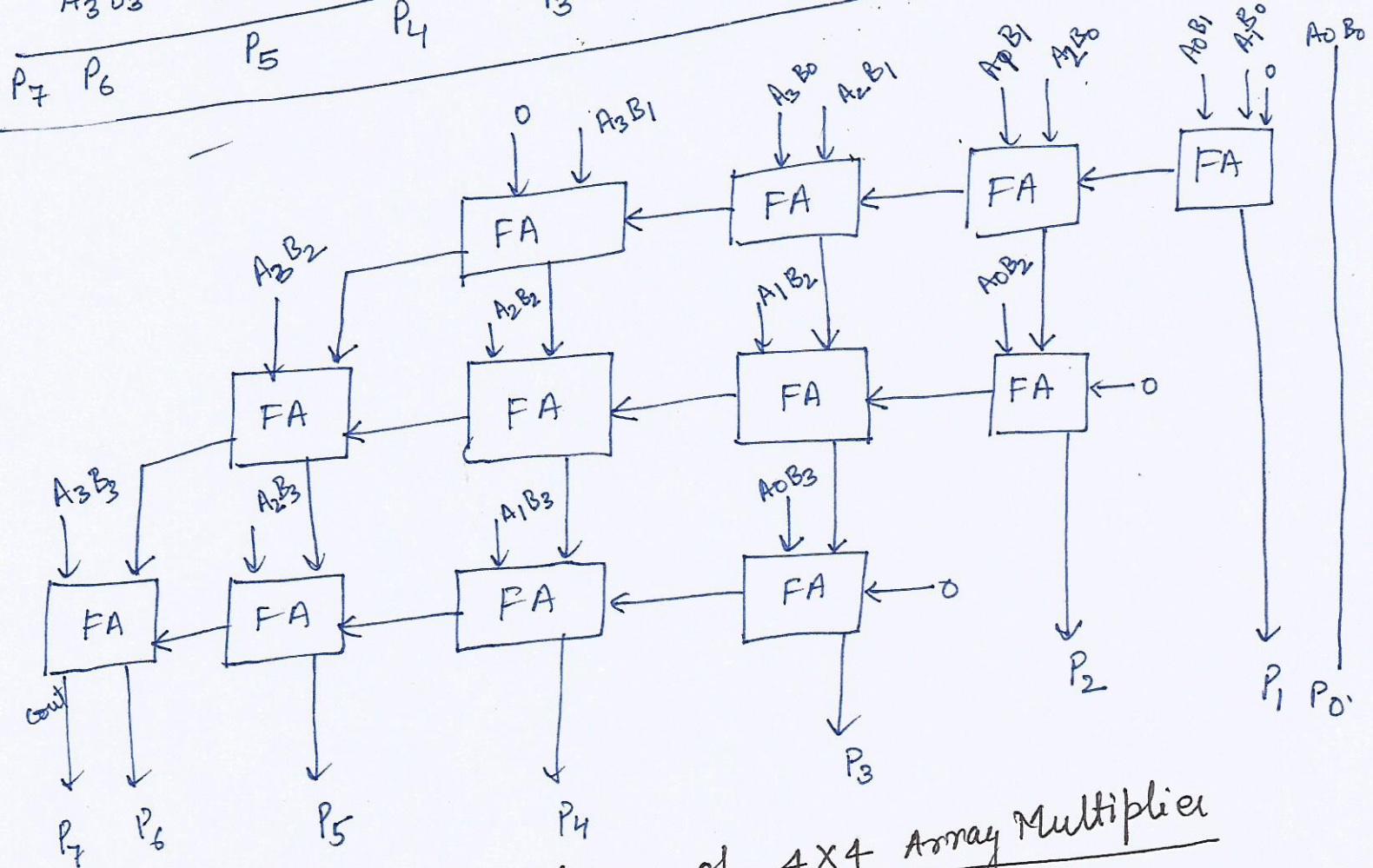
A_3B_1 A_0B_2 X

A_2B_2 A_1B_2

A_3B_2 A_0B_3

A_3B_3 A_2B_3 A_1B_3

P_7 P_6 P_5 P_4 P_3 P_2 P_1 P_0



Circuit Diagram of 4X4 Array Multiplier

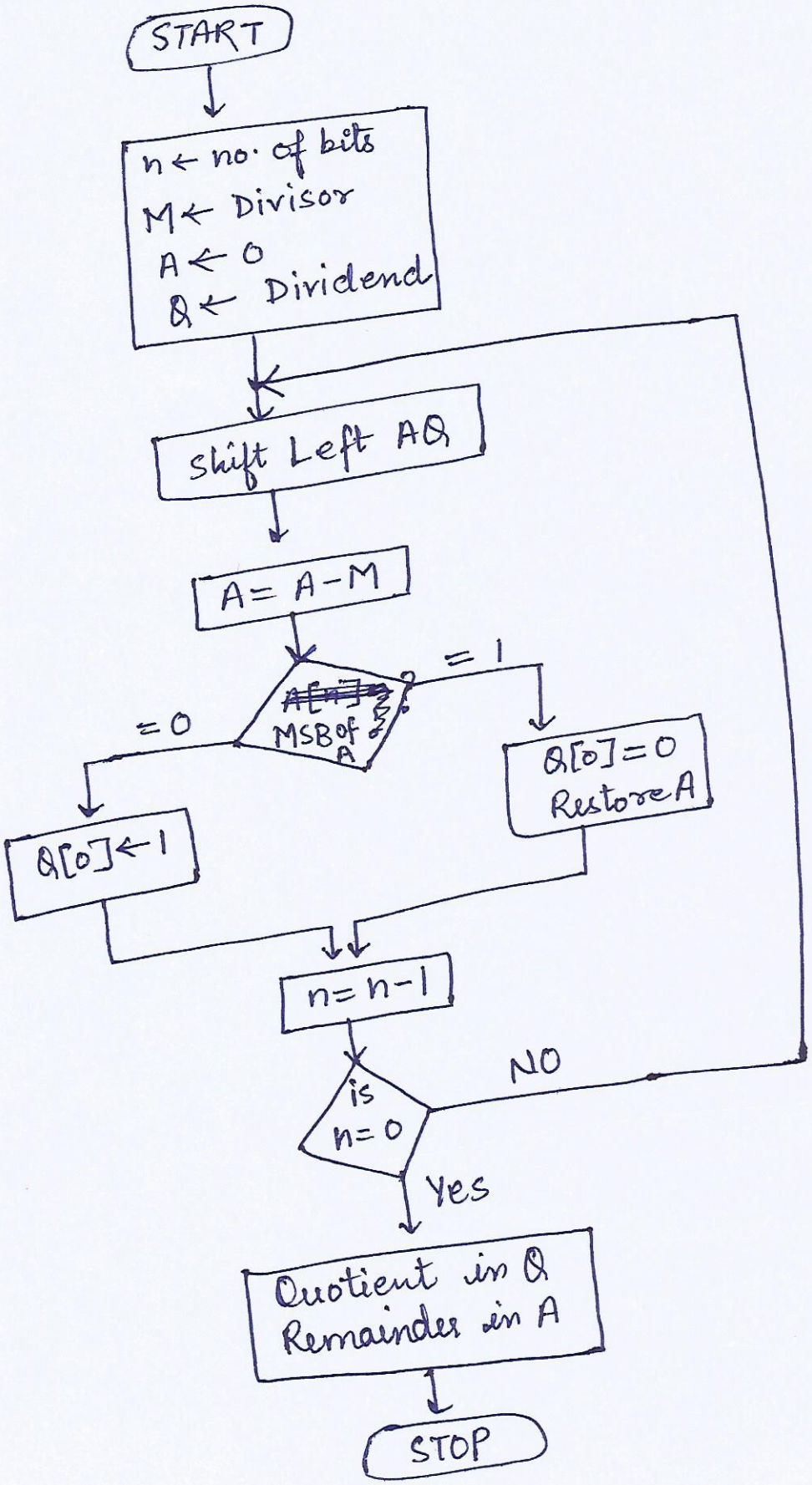
This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Minakshi

Division of unsigned Integer: Restoring Division Algorithm:

Flow chart:



Downloaded by Sangam gupta (sangamg7318@gmail.com)

Minakshi

Example: Divide 11 by 3.

(Q) Dividend = 11
(M) Divisor = 3

$M = 3 \Rightarrow (00011)_2 \Rightarrow$ Number of bits in $M = n+1$

$Q = 11 \Rightarrow (1011)_2$

$-M = (11101)_2$

Tracing Table

n	M	A	Q	Action / comment
4	00011	00000	1011	Initialization
		00001	011?	shift left AQ
		11110	011?	A = A − M ⇒ 00001 + 11101 = 11110
				A[n] == 1
		00001	0110	Q[0] ← 0 Restore A
				n = n − 1
3		00010	110?	shift left AQ
		11111	110?	A = A − M
		00010	1100	Q[0] ← 0, Restore A
		00010	1100	n = n − 1
2		00101	100?	SL AQ
		00010	100?	A = A − M
		00010	1001	Q[0] ← 1
		00010	1001	n = n − 1
1		00101	001?	SL AQ
		00010	001?	A = A − M
		00010	0011	Q[0] ← 1

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

n	M	A	Q	Action / comment
~~0~~	00011	00010	0011	n = n-1

Now Quotient in Q = 0011 ⇒ 3

Remainder in A = 00010 ⇒ 2

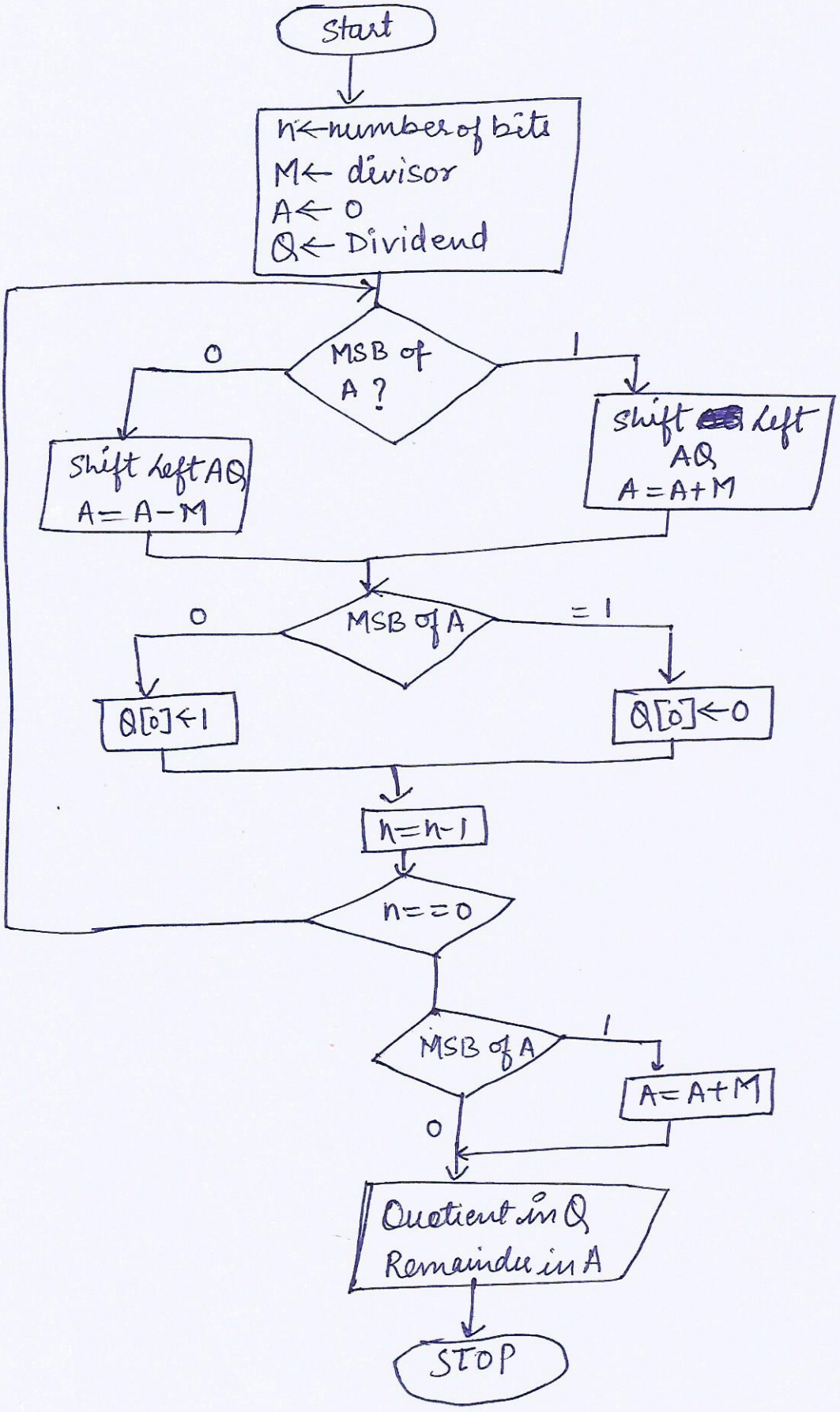
11 (Dividend) / 3 (Divisor) ⇒ 3 Quotient
2 Remainder

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Minakshi

Flow chart of Unsigned Integer Division

Non-Restoring Method



This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Minakshi

Example Divide 11 by 3.

(Q) Dividend = 11
(M) Divisor = 3

Number of bits $\Rightarrow n = 4$
Number of bits $M = n+1 = 5$
$-M = 11101$

Tracing Table:

n	M	A	Q	Action / comment
4	00011	00000	1011	Initialization
		00001	011?	SL AQ
		11110	011?	A = A − M
		11110	0110	Q[0] ← 0
		11110	0110	n = n − 1
3		11100	110?	SL AQ
		11111	10?	A = A + M
		11111	1100	Q[0] ← 0
		11111	1100	n = n − 1
2		11111	100?	SL AQ
		00010	100?	A = A + M 11111 + 00011 = 100010 ↑ discard it
		00010	1001	Q[0] ← 1, n = n − 1
1		00101	001?	SL AQ
		00010	001?	A = A − M
		00010	0011	Q[0] ← 1
		00010	0011	n = n − 1
0 (Yes)				

MSB of A = = 0 ⇒ then Result:

Quotient in Q = $(0011)_2 = 3$

Remainder in A = $(00010)_2 = 2$

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Minakshi

Ques Divide 7 by 3 using Non-Restoring method.

Q = Dividend = 7 ⇒ 111
M = Divisor = 3 ⇒ 0011
-M = 1101
n = 3

0 → Subt
1 → Add.

n	A	Q	Action
3	0000	111	Initialization
	0001	11?	SL AQ,
	+1101		A = A - M
	1110	11?	Q[0] ← 0
	1110	110	
2	1101	10?	Shift Left.
	0011		A = A + M.
carry discard	0000	101.	Q[0] ← 1
1	0001	01?	
	1101		A = A - M.
	1110	010	Q[0] ← 0
0	↓	↓	
	Remainder	Quotient.	

A ⇒ 1110.

A ⇒ 1110
+M ⇒ +0011
0001

Remainder = 1 Quotient = 2.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Floating Point Representation:

→ we can represent a floating point number in the form

$$\pm S \times B^{\pm E}$$

B ⇒ Base, For binary B = 2.

⇒ This number can be stored in a binary word with three fields

→ Sign (plus or minus)
→ significand or Mantissa (S)
→ Exponent E.

Sign	Biased Exponent	Mantissa or significand

IEEE 754 Format:

① Single Precision (32 bit)
② Double Precision (64 bit)

	sign bit	Biased Exponent bits	Bias	Fraction bits
single Precision	1	8	127	23 bits
Double Precision	1	11	1023	52 bits

Two functions for representing floating point numbers

→ Normalization
→ Biasing

Sign ⇒ 0 for positive numbers
1 for negative numbers.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

A normalized number is one in which the most significant digit of the significand is one.

For base 2 representation

$$1.bbb\text{---}b \times 2^{\pm E}$$

$$1.M \times 2^{\pm \text{Exponent}}$$

where M ⇒ Mantissa / fraction / significand

→ Because the most significant bit is always one, it is unnecessary to store.

→ Given a number that is not normalized, the number may be normalized by shifting the radix point to the right of left most 1 bit and adjusting the exponent accordingly.

→ Biasing: Exponent is stored in biased representation.

→ In single precesion case, 8 bits field yields the numbers 0 through 255.
with a bias of 127 (2^7-1), the true exponent values are in range
−127 to +128

→ In double precesion case, 11 bits field yields the numbers 0 through 2047
with a bias of 1023 ($2^{10}-1$), the true exponent values are in range. −1023 to +1024

⇒ Why Biasing: So that the bits in exponents can be treated as unsigned or non negative integers.

Instead of storing exponent in 2's complement, we can store it as unsigned number with the help of biasing

This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Ques: Represent $(-1234.125)_{10}$ into single Precision and double precision representation.

Ans:- ① First convert 1234.125 into binary number.

$(1234.125)_{10} = 10011010010.001$

② Now Normalize this number ⇒ move the decimal point right to the leftmost 1.

$1.0011010010001 \times 2^{+10}$

Single Precision: 32 bit

(i) calculate biased exponent.

Biased Exponent $E' = E + Bias$

$= +10 + 127$

$= 137$

$= 10001001$

sign = negative = 1

sign	Biased Exponent	Mantissa (23)
1	10001001	00110100100000000000

Double Precision 64 bits:

(i) calculate biased exponent $E' = E + Bias$

$= 10 + 1023$

$= 1033$

$= 10000001001$

sign	Exponent (E')	Mantissa (52)
1	10000001001	0011010010001000000---0000

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Ques Represent 0001001101.00110 using single Precision and double precision.

Given number 0001001101.00110

Normalized Number ⇒ $1.00110100110 \times 2^{+9}$

· Biased Exponent (SP): $E' = E + 127$

$= 9 + 127$

$\Rightarrow 136$

$\Rightarrow 10001000$

Biased Exponent (DP): $E' = E + 1023$

$= 9 + 1023 \Rightarrow 1032$

$\Rightarrow 10000001000$

single Precision

0	10001000	00110100110.00000000

Double Precision

0	10000001000	00110100110000---00

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Ques: Represent $-(0.000001000110011010)_2$ using single precision and double precision.

Normalize the number.

$\Rightarrow 1.00011001101 \times 2^{-6}$

Biased Exponent in SP: $E' = E + Bias \Rightarrow -6+127$

$\Rightarrow 121$

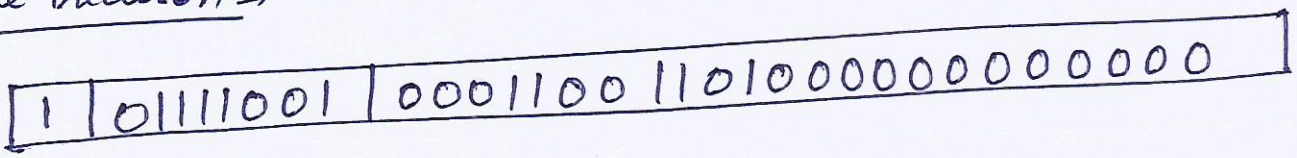
$\Rightarrow 01111001$

Biased Exponent in DP: $E' \Rightarrow E + Bias \Rightarrow -6+1023$

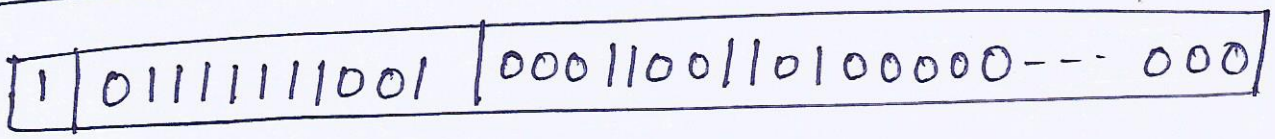
$\Rightarrow 1017$

$\Rightarrow 01111111001$

single Precision ⇒



Double Precision:-



Downloaded by Sangam gupta (sangamg7318@gmail.com)

Floating-Point Addition and Subtraction:

- Four basic phases of the algorithm for addition and subtraction.

1. check for zeros.
2. Align the significands
3. Add or subtract the significands
4. Normalize the result.

Floating Point Numbers	Addition and subtraction
$X = X_S \times B^{X_E}$ $Y = Y_S \times B^{Y_E}$	$X+Y = (X_S \times B^{X_E - Y_E} + Y_S) \times B^{Y_E}$ $X-Y = (X_S \times B^{X_E - Y_E} - Y_S) \times B^{Y_E}$ } $X_E \leq Y_E$

Example:

$$X = 0.3 \times 10^2 = 30$$
$$Y = 0.2 \times 10^3 = 200$$

$$X+Y = (0.3 \times 10^{2-3} + 0.2) \times 10^3 = (0.03 + 0.2) \times 10^3$$
$$= .23 \times 10^3$$
$$= 230$$

$$X-Y = (0.3 \times 10^{2-3} - 0.2) \times 10^3 = (0.03 - 0.2) \times 10^3$$
$$= (-.17) \times 10^3$$
$$= -170$$

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Phase ①: Zero check: If either operand is 0, the other is reported as the result.

Phase ②: Significand alignment: to manipulate the numbers so that the two exponents are equal.

Alignment may be achieved by shifting either the smaller number to right (increasing its exponent) or shifting the larger number to left.

- often small number is picked for shifting

Phase ③ Addition: Next, two significands are added together, taking into account their signs.

Phase ④ Normalization: This phase normalize the result. Normalization consists of shifting significand digits left until the most significant digit is nonzero. Each shift causes a decrement of the exponent and thus could cause an exponent underflow. Finally, the result must be rounded off and the reported.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

For floating-point addition and subtraction, it is necessary to ensure both the operands have the same exponent value. This may require shifting the radix point on one of the operands to achieve alignment.

A floating-point may produce one of these conditions:

- Exponent overflow
- Exponent Underflow
- Significand underflow
- Significand overflow

Exponent Overflow: A positive exponent exceeds the maximum possible exponent value. In some systems, this may be designated as $+\infty$ or $-\infty$.

Exponent Underflow: A negative exponent is less than the minimum possible exponent value. (e.g. -200 is less than -127) This means the number is too small to be represented and it may be reported as zero.

Significand underflow: In the process of aligning significands digits may flow off the rightend of the significand. So, some form of Rounding is required.

Significand overflow: The addition of two significands of the same sign may result in a carry out of the most significand bit. This can be fixed by Realignment.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Flow chart for Floating Point Addition and Subtraction

flow1.jpg

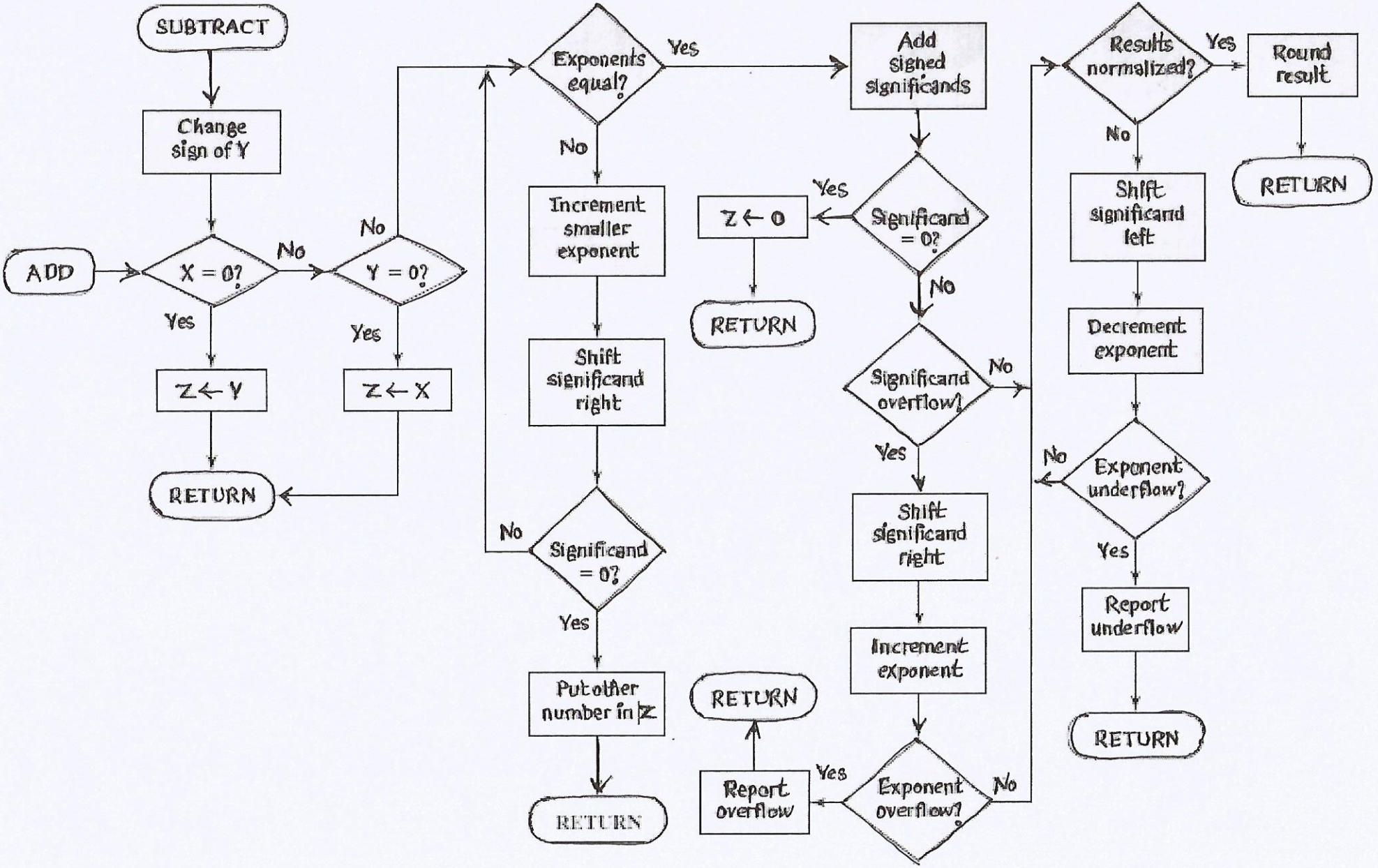


Figure 9.22 Floating-Point Addition and Subtraction ($Z \leftarrow Z \pm Y$)

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Flow chart for Floating Multiplication

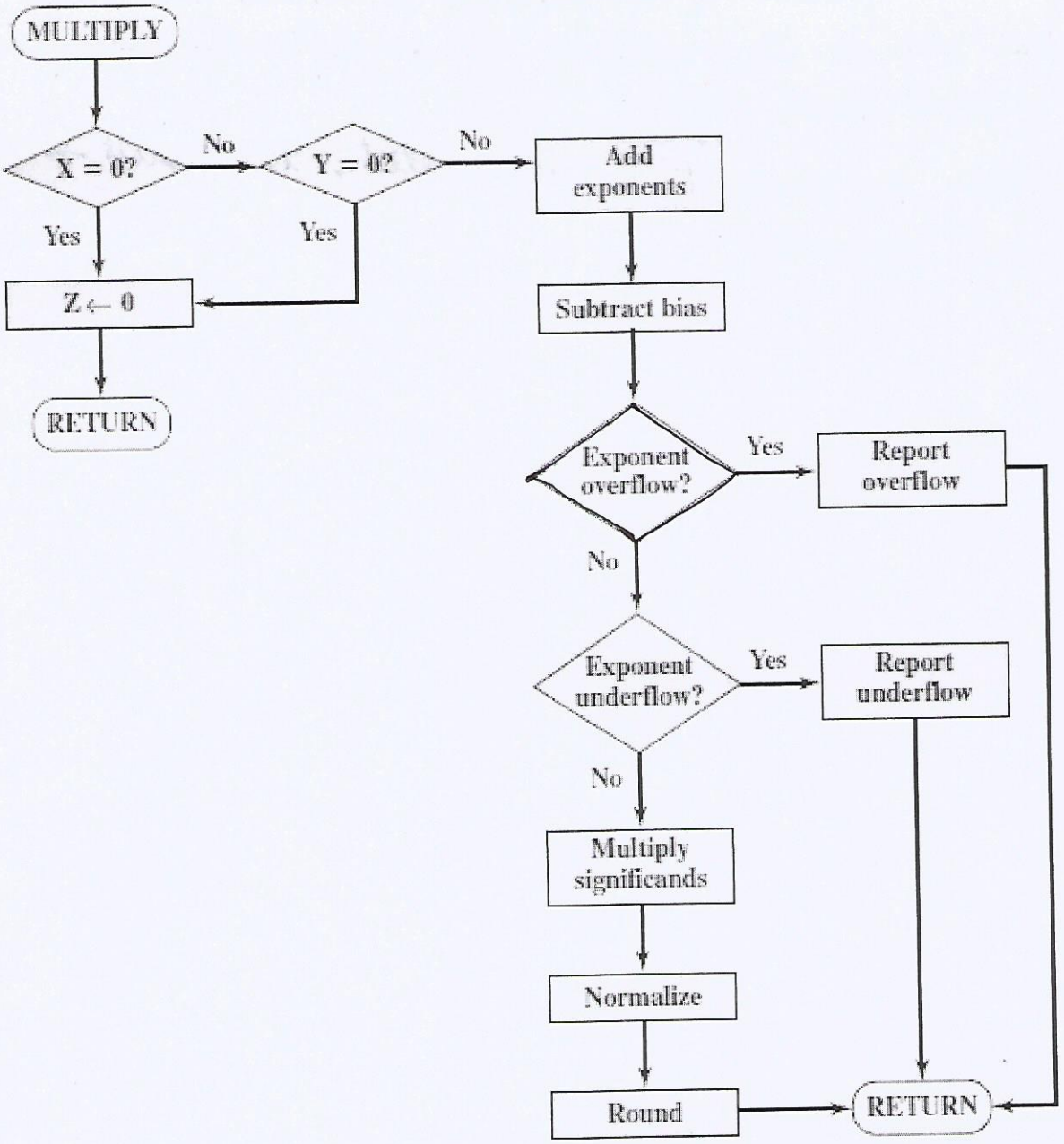


Figure 9.23 Floating-Point Multiplication ($Z \leftarrow X \times Y$)

Floating-Point Multiplication:

- if either operand is 0 (zero), zero is reported as result.
- The next, ~~step~~ step is to add the exponents. if the exponents are stored in biased form, the exponent sum would have doubled the bias, Thus, the bias must be subtracted from the sum. The result could be either an exponent underflow or exponent overflow, which would be reported.

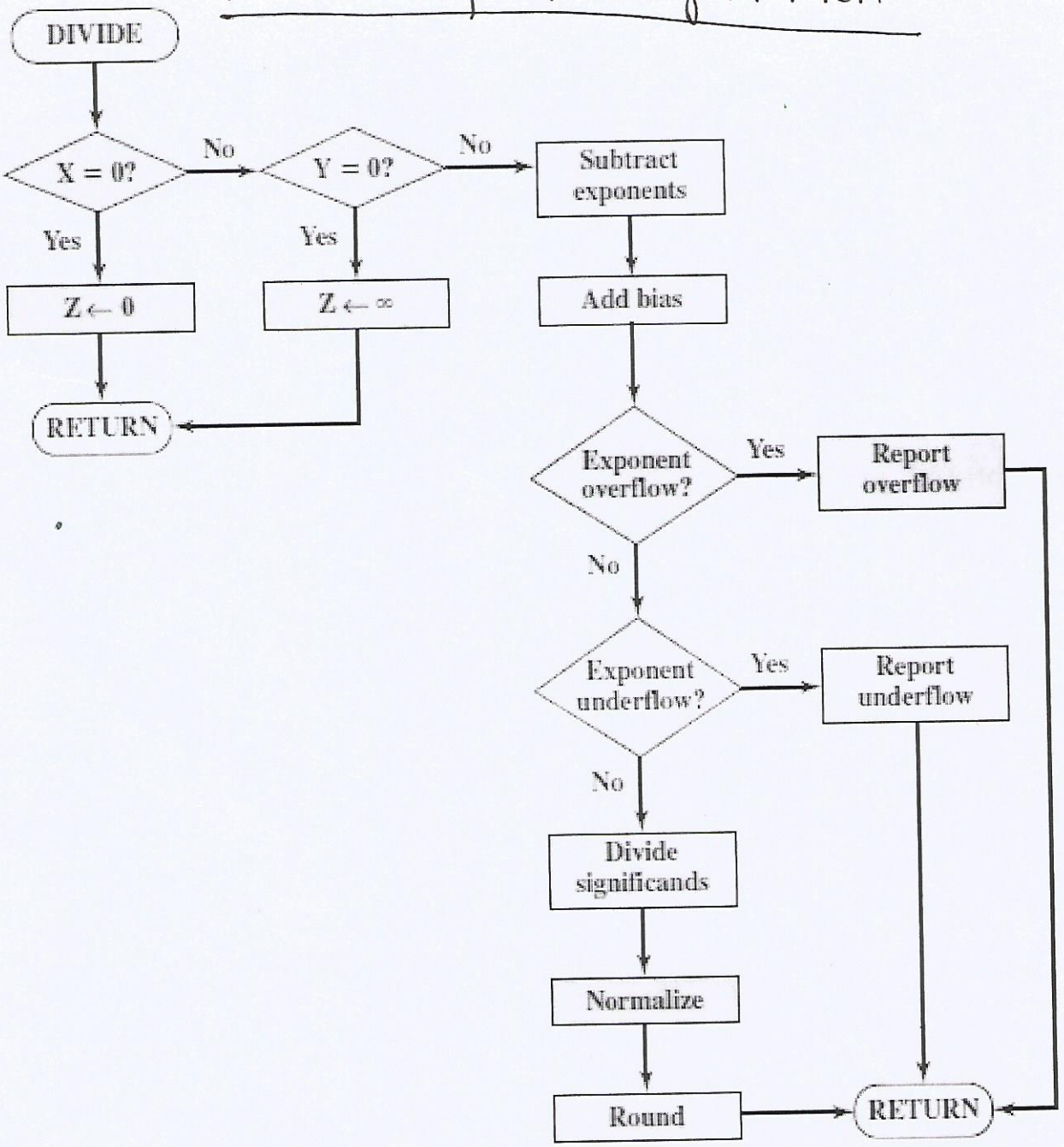
This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

- if the exponent of the product is within the range, the next step is to multiply the significand, taking into account their signs.
- After the product is calculated, the result is normalize and rounded.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Flow chart for Floating Division



Floating Point division :-

- Test for zero → if divisor is zero, an error is issued or result is set to infinity.
  → if dividend is zero, the result is zero.
- Next, the divisor exponent is subtracted from the dividend exponent. This removes bias, which must be added back in.
- Test are made for exponent underflow or overflow.
- The next step is to divide the significands.
- Followed by normalization and rounding.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Logic Microoperations :

→ Logic microoperations specify binary operations for strings of bits stored in registers. These operations consider each bit of the register separately and treat them as binary variables

For Example

- AND ($\wedge$)
- OR ($\vee$)
- NOT ($-$)
- NOR
- NAND
- X-OR ($\oplus$)
- X-NOR ($\odot$)

$R_1 = 1010$
$R_2 = 1100$

AND ⇒
$$\begin{array}{r} 1010 \\ 1100 \\ \hline 1000 \end{array}$$

OR ⇒
$$\begin{array}{r} 1010 \\ 1100 \\ \hline 1110 \end{array}$$

NOT
$$\begin{array}{r} 1010 \\ \hline 0101 \end{array}$$

NOR :
$$\begin{array}{r} 1010 \\ 1100 \\ \hline 0001 \end{array}$$

NAND :-
$$\begin{array}{r} 1010 \\ 1100 \\ \hline 0111 \end{array}$$

X-OR
$$\begin{array}{r} 1010 \\ 1100 \\ \hline 0110 \end{array}$$

X-NOR
$$\begin{array}{r} 1010 \\ 1100 \\ \hline 1001 \end{array}$$

Downloaded by Sangam gupta (sangamg7318@gmail.com)

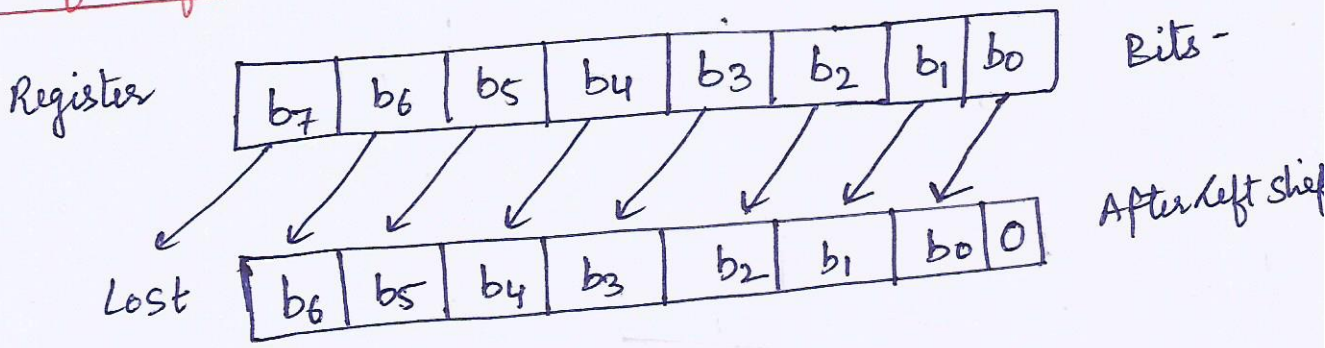
Shift Microoperations

→ Shift Micro operations are used for serial transfer of data.

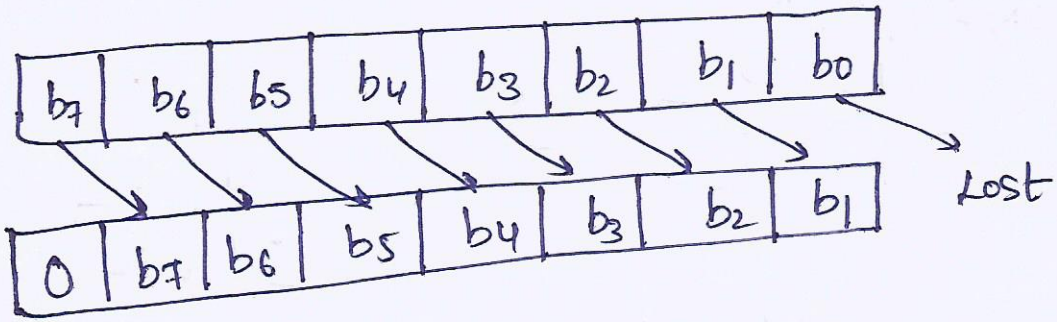
→ Types of Shift operations

- Logical shift — shift left, shift Right
- Circular shift or Rotate — circular Right shift, circular left shift
- Arithmetic — Arithmetic Right shift, Arithmetic left shift

Logical Shift Left:



Logical Shift Right:



Example

1 0 0 1 0 1 1 1

0 0 1 0 1 1 1 0 After shift Left

1 0 0 1 0 1 1 1

0 1 0 0 1 0 1 1 After shift Right

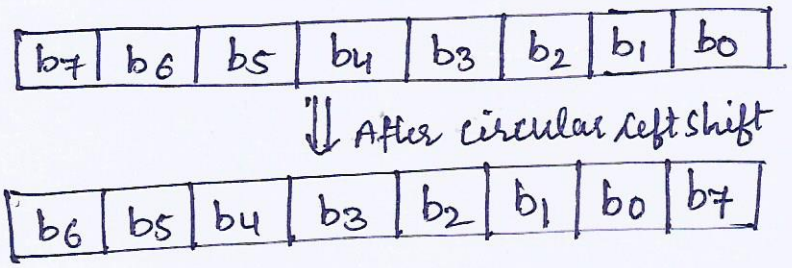
This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

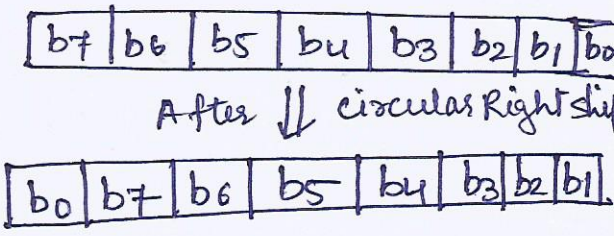
Circular shift: (Also known as Rotate operation).

→ circulates the bits of the register around the two ends without loss of information.

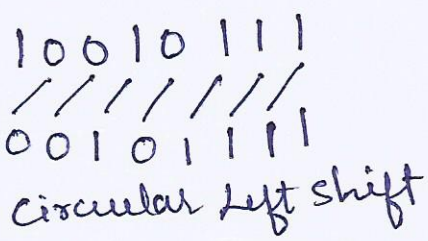
Circular left shift

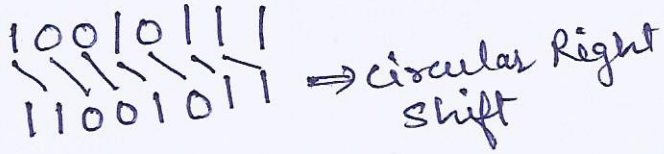


Circular Right shift



Example

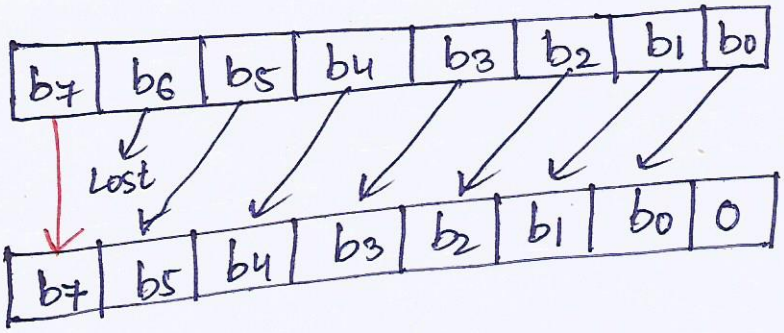


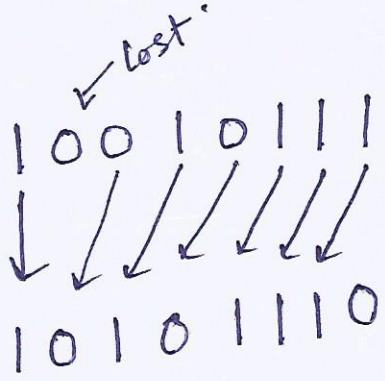


Arithmetic shift:

- An Arithmetic shift operation shift a signed binary number to the left or right
- Arithmetic shift leaves the sign bit unchanged.

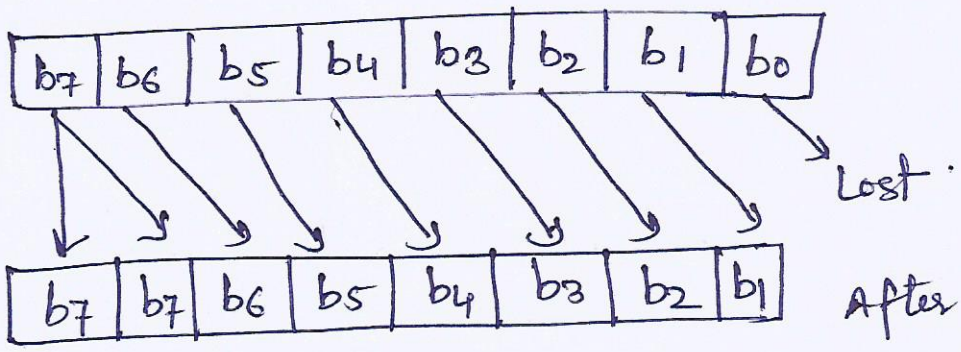
Arithmetic Left Shift



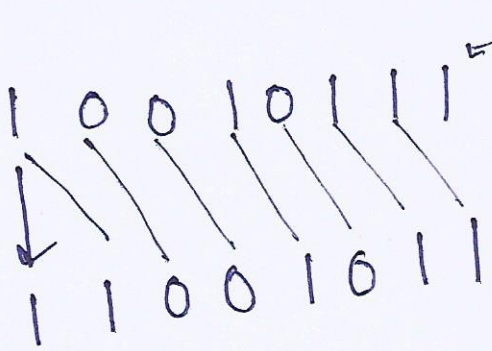


Downloaded by Sangam gupta (sangamg7318@gmail.com)

Arithmetic Right Shift :



After Arithmetic Right shift



This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Arithmetic Logic Unit :

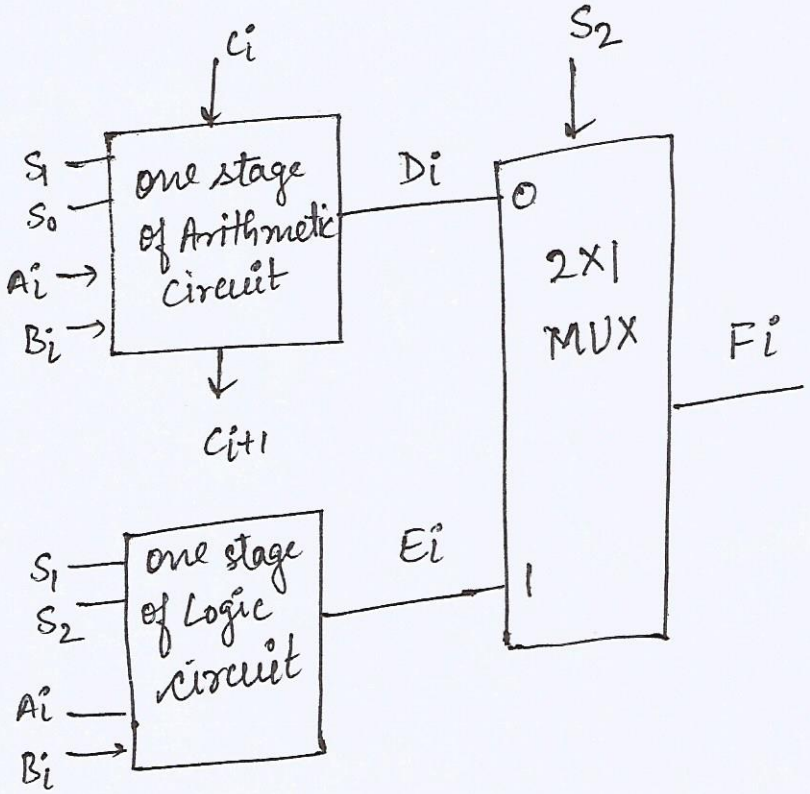
→ CPU contains ALU, CU and Registers

→ ALU is responsible for arithmetic and logical operations

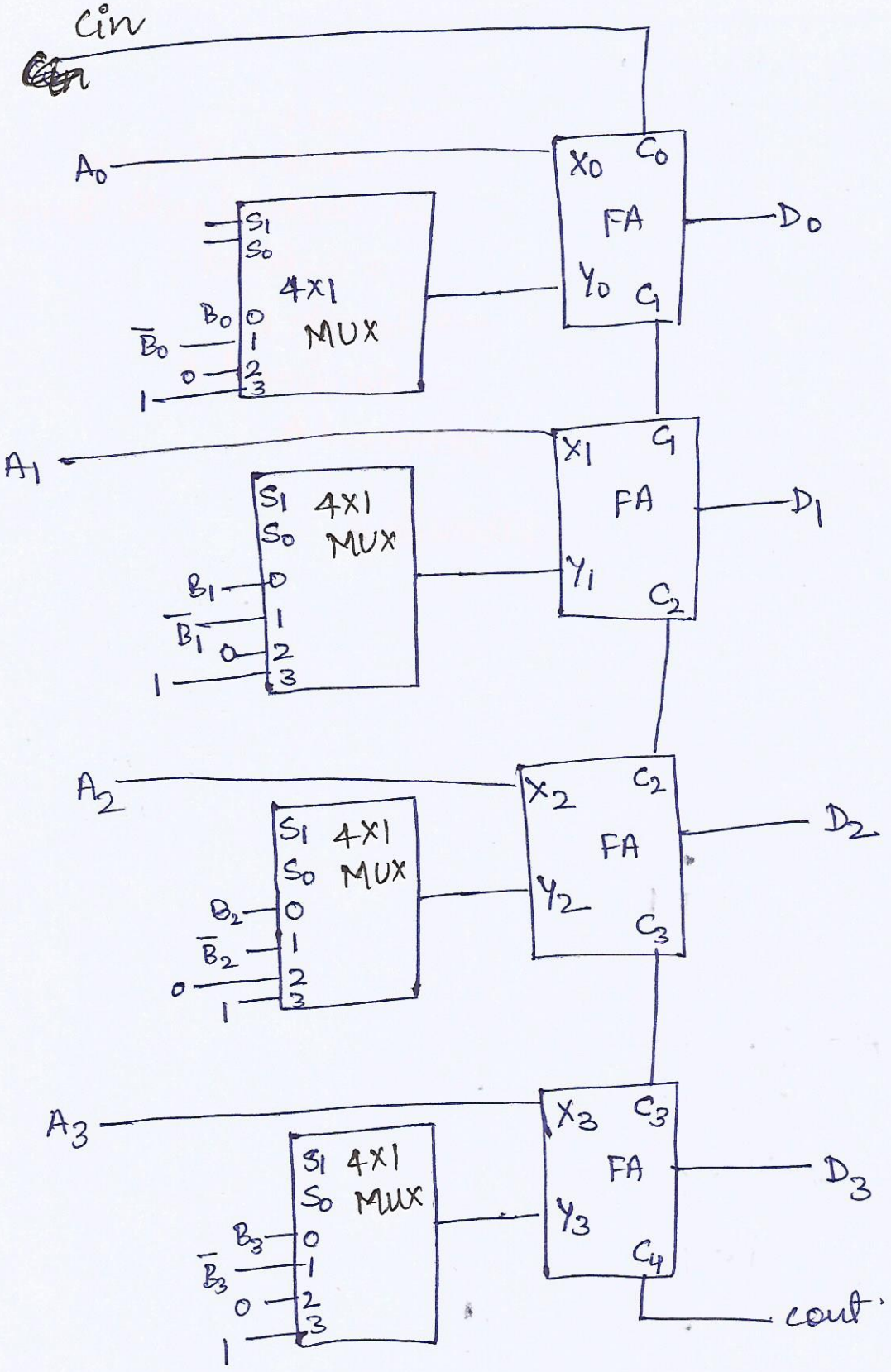
→ Basically ALU is a digital circuit that performs arithmetic operations like addition, subtraction, division and multiplication, and logical operations like AND, OR, XOR, NOT. etc.

→ Types of ALU ⇒ ① Combinational ALU
② Sequential ALU.

combinational ALU:



Downloaded by Sangam gupta (sangamg7318@gmail.com)



4 bit Arithmetic Circuit performing operations like addition, subtraction, increment, decrement and transfer etc.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

S_1	S_0	C_{in}	Input	Output $D = A + Y + C_{in}$
0	0	0	B	$D = A + B$ → Add
0	0	1	B	$D = A + B + 1$ → Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$ → Subtract with Borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$ → Subtract
1	0	0	0	$D = A$ → Transfer A
1	0	1	0	$D = A + 1$ → Increment A
1	1	0	1	$D = A - 1$ → Decrement A
1	1	1	1	$D = A$ → Transfer A

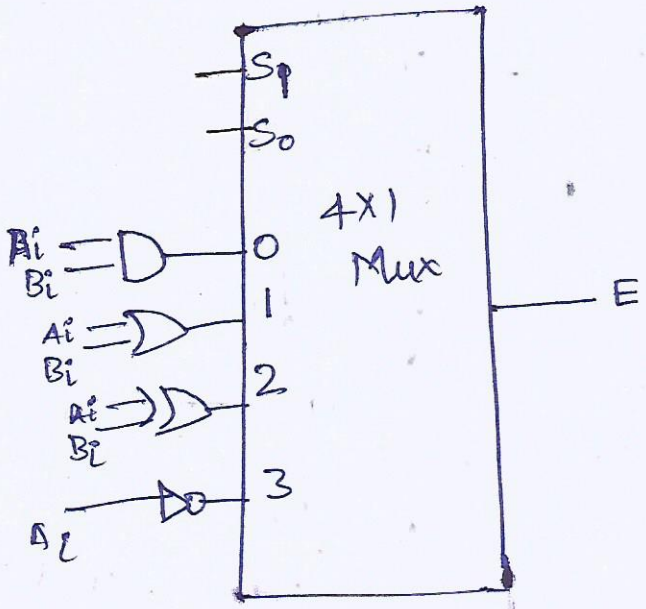
when $S_1 = 1$ and $S_0 = 1$ then Y input is $(B_3 B_2 B_1 B_0)$ 1111 which is 2's complement of 0001 ⇒ It makes

$$\begin{array}{r} A_3\ A_2\ A_1\ A_0 \\ +\ 1\ \ 1\ \ 1\ \ 1 \end{array} \Rightarrow \begin{array}{c} A \\ +2\text{'s comp(1)} \end{array}$$

$\Rightarrow A + (-1)$

$\Rightarrow A - 1$

One stage of logic circuit:-

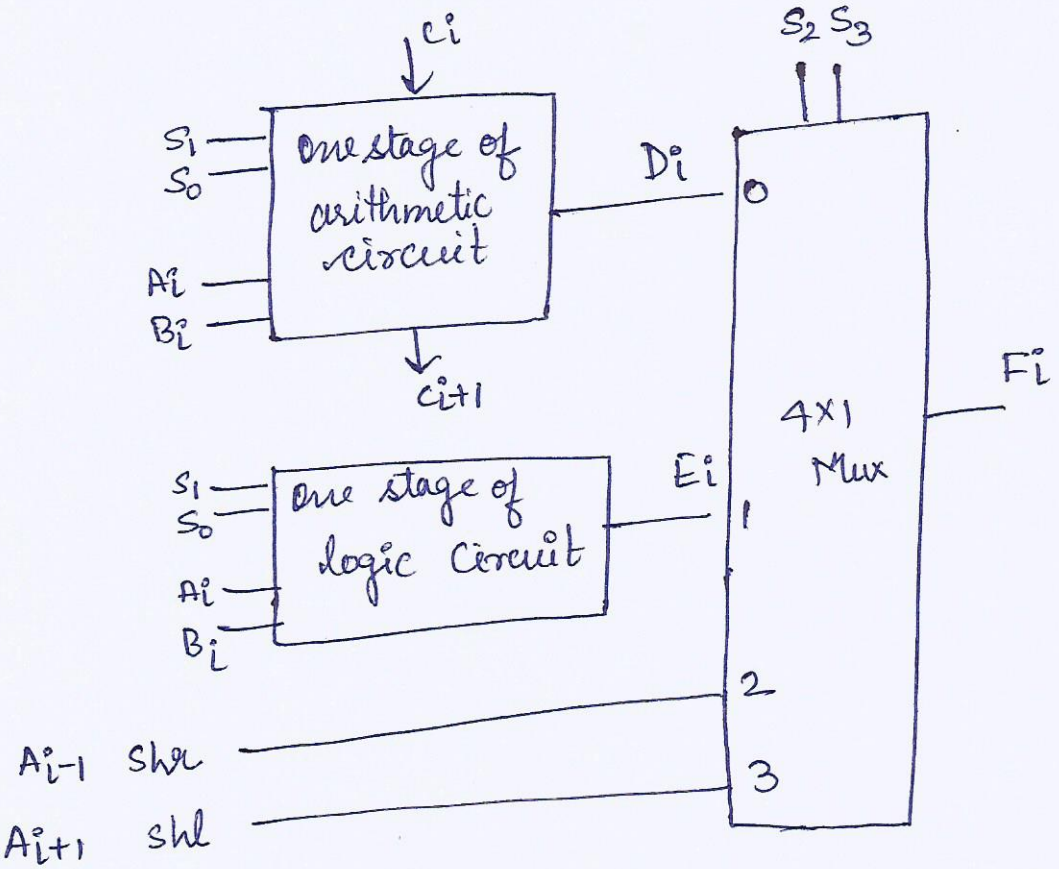


Function Table

S_1	S_0	Logic operation (E)
0	0	A_i AND B_i
0	1	A_i OR B_i
1	0	A_i XOR B_i
1	1	NOT A_i

Downloaded by Sangam gupta (sangamg7318@gmail.com)

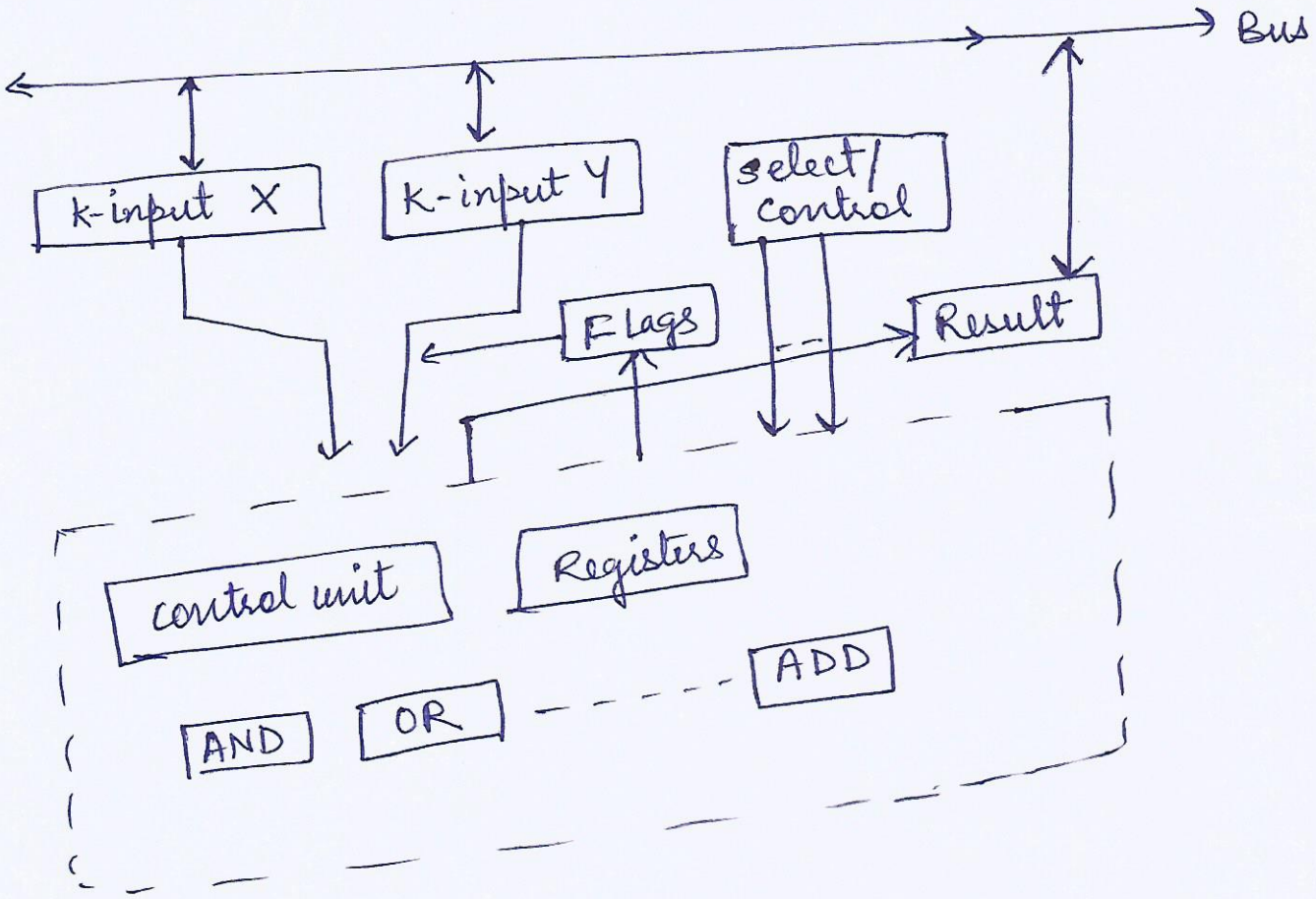
One stage of arithmetic logic shift unit



This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Sequential Logic Circuit based ALU



- Two Registers X and Y stores data or operands.
- Select/control selects the appropriate arithmetic or logic operation which is performed over the data stored on register X and Y.
- After the execution of operation the result will be stored in result register.
- After the execution of operation flags may be set such as carry, zero result, positive or negative result, overflow, division by zero etc.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

COA Unit 3 Notes

Computer Organization & Architecture (Dr. A.P.J. Abdul Kalam Technical University)

COA Unit - 3 Notes

Minakshi
IT Department.

Instruction:

→ The operation of the processor is determined by the instruction, it executes, referred to as machine instructions or computer instruction

→ The collection of different instructions that processor can execute is referred to as the processor's instruction set.

→ A computer instruction is a binary code that instruct the computer to perform a specific operation.

Elements of a Machine instruction:

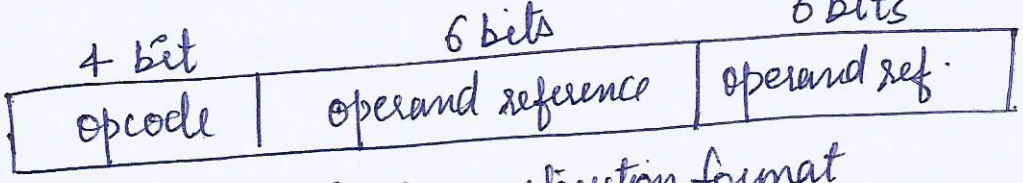
→ operation code
→ source operand reference
→ Result operand reference
→ Next instruction reference.

→ An operation code known as opcode which specifies operation to be performed.

→ Source operand reference: Refrence to operands on which the operation is to be performed.

→ Result operand reference: A reference to operand which will store the result produced by the instruction

→ A reference to the next instruction to be executed



A simple instruction format

16 bits

This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)

→ source and result can be in one of four areas.
  → main or virtual memory
  → Processor register
  → Immediate (contained in instruction directly)
  → I/O device.

→ opcodes are represented by abbreviations, called mnemonics such as ADD, SUB, LOAD, STOR etc.

→ An instruction may have many fields. A layout of a simple instruction is known as "Instruction format"

Types of Instruction:

Most computer instructions can be classified into three categories:
  → Data Transfer instructions
  → Data Manipulation instructions
  → Program control instruction.

Data transfer instructions cause transfer of data from one location to another without changing the binary information content.

Data manipulation instructions are those that perform arithmetic, logic, and shift operations.

- Program control instruction provide decision making capabilities and change the path taken by the program when executed in the computer.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Data Transfer:

→ Data transfer instructions move data from one location to another without changing the data content.

→ These instructions are used to bring data to and from memory to registers.

Examples:

MOVE : designates a transfer from one register to another.

Store : transfer ~~word~~ from a processor register into memory.

Load : a transfer from memory to register

Exchange: swaps information between two registers or a register and memory.

Input : transfer from input terminal to processor register.

output: transfer from process register to output terminal.

Push : transfer from process register to the top of stack.

POP : transfer from top of stack to process register.

clear : transfer of word 0's (zeros) to the destination

Set : transfer of word 1's (ones) to the destination.

Data Manipulation:

- Data manipulation instructions perform operations on the data.
- These can be divided into three types.

(1) Arithmetic
(2) Logical and bit manipulation instructions
(3) Shift microoperations.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Arithmetic Instructions:

These instructions are used to perform arithmetic operations.

Example:

Add → compute the sum of two operands
Subtract → Compute difference of two operands
Multiply → compute the product of two operands
Divide → Compute quotient of two operands.

Add with carry → compute the sum of two operands with carry
Subtract with borrow → compute the difference with borrow.
Negate (2's complement) → change the sign of the operand.

Increment → Add 1 to operand
Decrement → Subtract 1 from operand.

Logical Instruction:

- Logical and bit manipulation instruction.
- Logical instructions perform binary operations on string of bits stored in registers.

Example:

AND → performs AND operands
OR → performs OR of operands
Complement or NOT → takes complement
X-OR → performs XOR of operands

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Clear selected bits ⇒ to clear a bit to zero or selected bits to zero

set selected bits ⇒ to set a bit or selected bits to 1.

complement selected bits ⇒ to complement a bit or selected bits.

Test ⇒ Test specified condition (set flags)

compare ⇒ make logical comparison of two operands.

Enable Interrupt ⇒
Disable Interrupt ⇒
} ⇒ set control variables for interrupt handling, timer control etc.

Shift Instructions:

→ shifts are operations in which the bits of the word are moved to the left or right.

Example:

Logical shift Right
Logical shift Left
Arithmetic shift Right
Arithmetic shift Left
Rotate Right
Rotate Left.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Program Control: or Transfer of Control instructions.

→ A program control type of instructions, when executed, may change the address value in the program counter and cause the flow of control to be altered.

→ Program control instructions specify conditions for altering the content of the program counter.

Example Program control instruction

Jump (branch) → Unconditional transfer, load PC with specified address.

Jump (Conditional) → Test specified condition, either load PC with specified address, or do nothing based on condition

Jump to subroutine or CALL → Place current program control information in known location, Jump to specified address.

Return → Replace contents of PC and other registers from the known locations.

Skip → Increment to PC to skip next instruction.

Skip (conditional) → Test specified conditions, either skip or do nothing based on condition

Halt → Stop program execution.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Wait (hold) → Stop program execution, test specified condition repeatedly; resume execution when condition is satisfied.

No operation → No operation is performed, but program execution is continued.

Input and output Instructions:

Input (Read) → Transfer data from input terminal to a destination

Output (write) → Transfer data from specified source to output terminal

Start I/O → Transfer instruction to I/O processor to initiate operation

Test I/O → Transfer status information I/O system to the specified destination.

conversion Instruction:

- convert or translate value or words.

Example

Translate → Translates value in a section of memory based on table of correspondence

convert → Converts the content of word from one form to another.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Example Conditional Branch instructions.

Branch if zero.
Branch if not zero
Branch if carry
Branch if not carry
Branch if plus
Branch if minus
Branch if overflow
Branch if no overflow.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

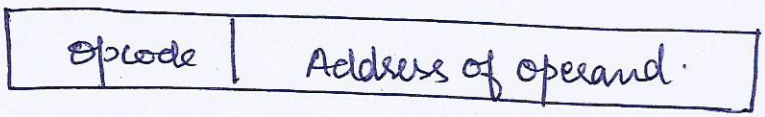
STORED PROGRAM ORGANIZATION : OR Single Accumulator Organization

OR

Processor Organization with Accumulator

→ To organize a computer in this method, Processor has a register called accumulator and an instruction code format with two parts.

→

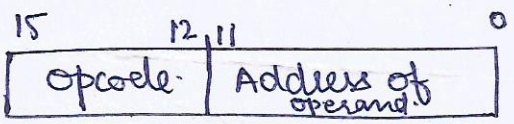


→ The first part specifies the operation to be performed and the second specifies an address of operand in the memory.

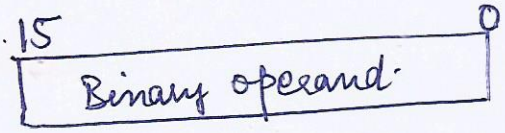
→ The other operand will be stored in the processor register

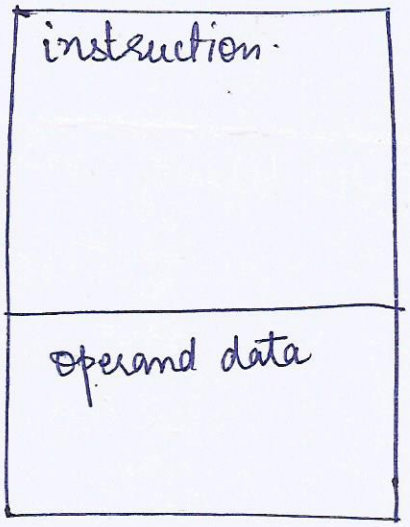
Example

Memory 4096X16



instruction format.





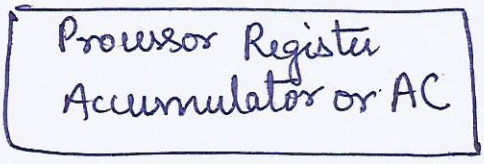
Number of bits in address = 12

Number of memory locations = 2^{12}

= 4096

The number of bits of operand or data

= 16.



In this method, processor performs the operation specified by the opcode on the data stored on the memory location and content of AC register.

This document is available free of charge on studocu
Downloaded by Sangam Gupta (sakeamg7318@gmail.com)

Instruction Format :

Most common fields of instructions are.

→ operation field which specifies the operation to be performed

→ Address field which contains the location of operand i.e, register or memory location

→ Mode field which specifies how operand is to be found.

Generally CPU organizations are

→ Single Accumulator Organization

→ General Register Organization

→ Stack Organization

→ An instruction is of various length depending on the number of addresses, it contains

On the basis of number of addresses instruction can be classified

→ Three address instruction

→ Two address instruction

→ One address instruction

→ Zero address instruction

} instruction formats.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Three address instruction :

This format of instruction has three addresses to specify a register or memory location.

Opcode	Destination address	Source Address	Source address	Mode

Examples: ADD R1, R2, R3

OR

ADD R1, A, B

Two address instruction:

This format has only two address to specify a register or a memory location.

Opcode	Destination address	Source address	Mode.

Examples: MOV R1, A

ADD R1, B

One - address instruction :

- It has only one address to specify a register or a memory location
- This uses an implied 'ACCUMULATOR' or AC Register to store one operand / result.

Opcode	Address of operand	Mode

Example LOAD A

or

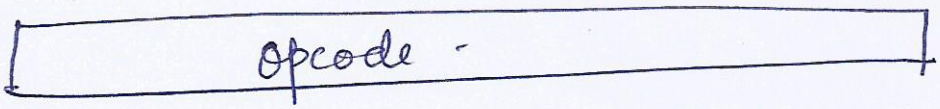
ADD B

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Zero Address Instruction:

- This instruction format does not contain any address field.
- A stack based computer do not use address field in instruction to perform operation. Because the operation is performed on the two top items of the stack.



Example ADD

Note In stack-base organization, to evaluate an expression first it is converted to Postfix notation or Reverse Polish Notation

---

Example for instruction formats to evaluate an expression.

Example ① Expression $X = (A+B) * (C+D)$

Three address Format:

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

Here R1, R2 are 2 registers

M[ ] ⟹ memory location.

Downloaded by Sangam gupta (sangamg731@gmail.com)

Two address instructions

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

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

One address instruction:

		Description
LOAD	A	AC ← M[A]
ADD	B	AC ← AC + M[B]
STORE	T	M[T] ← AC
LOAD	C	AC ← M[C]
ADD	D	AC ← AC + M[D]
MUL	T	AC ← AC * M[T]
STORE	X	M[X] ← AC

where AC is Accumulator Register

M[T] is temporary memory location to hold the intermediate Result.

M[ ] ← memory location

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Zero address Instruction:

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

Postfix notation $X = AB+CD+*$

		Description
PUSH	A	TOP = A
PUSH	B	TOP = B
ADD		TOP = A+B
PUSH	C	TOP = C
PUSH	D	TOP ← D
ADD		TOP ← (C+D)
MUL		TOP (A+B) * (C+D)
POP	X	M[X] = TOP

Ques: write a program to evaluate $\frac{(A-B)+C*(D*E-F)}{G+H*K}$

Ans: Three address instruction:

		Description
SUB	R1, A, B	R1 ← M[A] − M[B]
MUL	R2, D, E	R2 ← M[D] * M[E]
SUB	R2, R2, F	R2 ← R2 − M[F]
MUL	R2, R2, C	R2 ← R2 * M[C]
ADD	R1, R1, R2	R1 ← R1 + R2
MUL	R3, H, K	R3 ← M[H] + M[K]
ADD	R3, R3, G	R3 ← R3 + M[G]
DIV	X, R1, R3	M[X] ← R1 / R3

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Two address instructions

		Description
MOV	R1, A	R1 ← M[A]
SUB	R1, B	R1 ← R1 − M[B]
MOV	R2, D	R2 ← M[D]
MUL	R2, E	R2 ← R2 * M[E]
SUB	R2, F	R2 ← R2 − M[F]
MUL	R2, C	R2 ← R2 * M[C]
ADD	R1, R2	R1 ← R1 + R2
MOV	R3, H	R3 ← M[H]
ADD	R3, G	R3 ← R3 + M[G]
DIV	R1, R3	R1 ← R1/R3
MOV	X, R1	M[X] ← R1

One address instruction

		Description
LOAD	A	AC ← M[A]
SUB	B	AC ← AC − M[B]
STORE	T	M[T] ← AC
LOAD	D	AC ← M[D]
MUL	E	AC ← AC * M[E]
SUB	F	AC ← AC − M[F]
MUL	C	AC ← AC * M[C]
ADD	T	AC ← AC + M[T]
STORE	T	M[T] ← AC
LOAD	H	AC ← M[H]
MUL	K	AC ← AC * M[K]
ADD	G	AC ← AC + M[G]

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

STORE	T1	M[T1] ← AC
LOAD	T	AC ← M[T]
DIV	T1	AC ← AC / M[T1]
STORE	X	M[X] ← AC.

Zero address instruction

Postfix Notation AB- CDE*F-*+ GHK*+/

		Description
PUSH	A	Top ⇒ A
PUSH	B	Top ⇒ B
SUB		Top ⇒ (A-B)
PUSH	C	Top ⇒ C
PUSH	D	Top ⇒ D
PUSH	E	Top ⇒ E
MUL		Top ⇒ (D*E)
PUSH	F	Top ⇒ F
SUB		Top ⇒ ((D*E)-F))
MUL		Top ⇒ C* ((D*E)-F)
ADD		Top ⇒ (A+B) + (C*((D*E)-F)
PUSH	G	Top ⇒ G
PUSH	H	Top ⇒ H
PUSH	K	Top ⇒ K
MUL		Top ⇒ (H*K)
ADD		Top ⇒ G+(H*K)
DIV		Top ⇒ ((A-B)+C*((D*E)-F)/ (G+(H*K))
POP	X	X ⇐ Top.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Tutorial sheet

Ques: ① write an program for expression A+B*C using one address and Two address instruction

Ans: Expression ⇒ D= A+B*C

Program using One address instruction

LOAD	B	AC ← M[B]
MUL	C	AC ← AC * M[C]
ADD	A	AC ← AC + M[A]
STORE	D	M[D] ← AC

Program using Two address instruction

MOV	R1, B	R1 ← M[B]
MUL	R1, C	R1 ← R1 * M[C]
ADD	R1, A	R1 ← R1 * M[A]
STORE / MOV	D, R1	M[D] ← R1

Program using Zero address instruction

A BC*+=

PUSH	A	Top = A
PUSH	B	Top = B
PUSH	C	Top = C
MUL		Top = B*C
ADD		Top = A + B*C
POP	D	M[D] = A + B*C

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

New Topic

⇒ How would you find the number of bits ?

Example **Ques1** if a system has memory of 4096 words 16 bits per word then ⇒

Register		Size or Number of bits
Program Counter	PC	12
Address Register	AR	12
Instruction Register	IR	16
Data Register	DR	16
Accumulator	AC	16

Number of locations = 4096

Number of bits to identify 4096 locations ⇒ 12

→ Number of bits in address bus ⇒ 12
Number " " " data bus = 16

Ques2 A computer uses a memory of 256K words of 32 bit each. A binary instruction code is stored in a one word of memory. The instruction has four parts

- An indirection bit
- An operation code
- A register part to specify 64 registers and
- An address part

(i) How many bits are there in the operation code, the register code and address part?

(ii) Draw the instruction format and indicate the number of bits in each part.

(iii) How many bits are there in the data and address inputs/bus of memory

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Ans Number of words in memory = 256K ⇒ 256 × 1024 ⇒ $2^8 \times 2^{10}$

Size of the word ⇒ 32 bit

Size of the instruction = 32. { because instruction is stored in one word of memory }

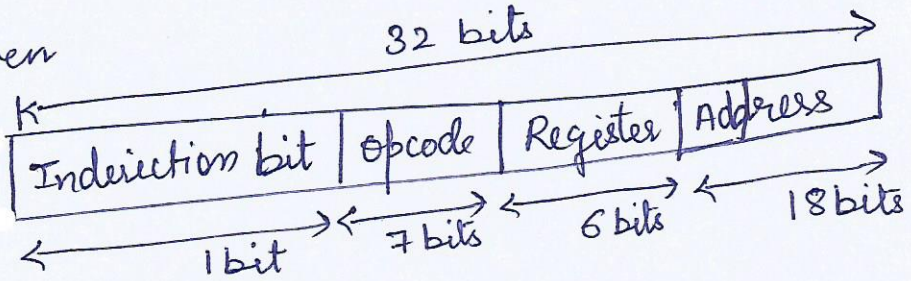
No. of Indirection bits ⇒ 1

Number of register = 64 ⇒ 2^6

bits to identify 64 registers = 6

Number of bits in address = 18

instruction format Given



Number of bits in opcode = 32 − (1+6+18)

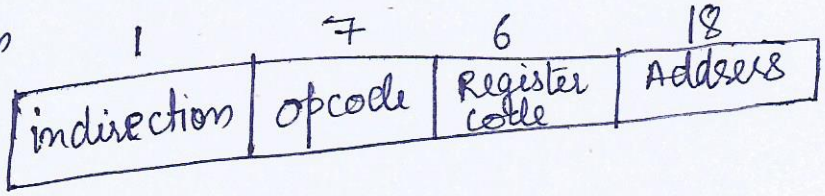
= 7 bits

(i) Number of bits in opcode = 7

Number of bits in register code = 6

Number of bits in address part = 18

(ii) Format of instruction



(iii) Number of bits in the address input = 18

Number of bits in data input = 32.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Ques For 256 instructions and 1024 word memory, find out the number of bits required in each type of instruction.

~~address instruction has~~

→ ~~Opcode~~

→ ~~Reference to operands~~

→ ~~Reference of the next instruction~~.

Ans: ~~Three~~ Four address instruction

Opcode	A_1	A_2	A_3	A_4

$A_1, A_2, A_3 \Rightarrow$ operand/Result reference

$A_4 \Rightarrow$ Next instruction reference.

- Number of possible instruction = 256

  Number of bits in opcode = 8

  $2^8 = 256$.

  Number of locations = 1024

  Number of bits in address = 10 $\{\because 2^{10} = 1024\}$

~~Three~~ Four address instruction

8 bits opcode	10 Address1	10 address2	10 A_3	10 bits A_4

Number of bits in instruction = 48

Downloaded by Sangam gupta (sangamg7318@gmail.com)

instruction

Opcode	Operand1	Operand2	- - - - - -

⇒ if operands are stored in register, then it is called register reference instruction

→ if operands are stored in memory, then it is called a memory reference instruction.

Qees: if 16-bit fixed length of instruction is stored in 128 word memory, then find out the number bits in opcode and total number of possible operations in 1 address instruction format and 2 address instruction format.

Ans: One address instruction format.

Number of locations in memory = 128

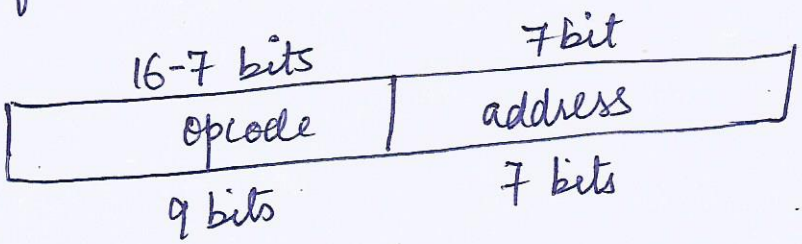
Number of bits in address to identify 128 location = n

$$2^n = 128$$

$$2^n = 2^7$$

$$\boxed{n = 7}$$

Number of bits in address field = 7

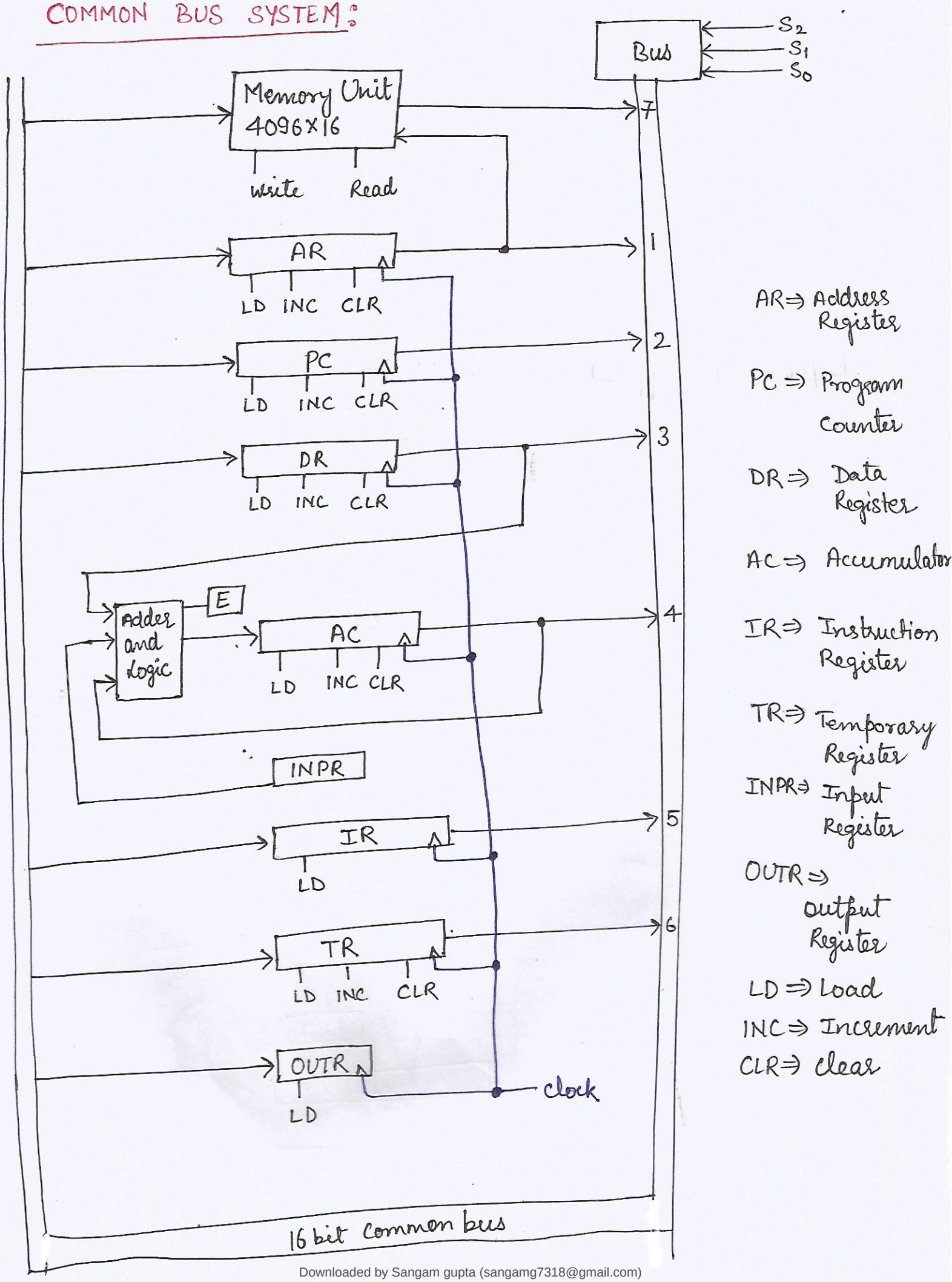


→ Number bits in opcode = 9

Number of possible operations = $2^9 \Rightarrow 512$

This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)

COMMON BUS SYSTEM:



AR ⇒ Address Register

PC ⇒ Program Counter

DR ⇒ Data Register

AC ⇒ Accumulator

IR ⇒ Instruction Register

TR ⇒ Temporary Register

INPR ⇒ Input Register

OUTR ⇒ Output Register

LD ⇒ Load

INC ⇒ Increment

CLR ⇒ clear

Downloaded by Sangam gupta (sangamg7318@gmail.com)

* Instruction cycle:

An instruction cycle consists of an instruction fetch, followed by zero or more operands fetches, followed by zero or more operand stores, followed by an interrupt check (if interrupts are enabled).

→ An instruction cycle includes the following stages / subcycles.

- Fetch
- Execute
- Interrupt

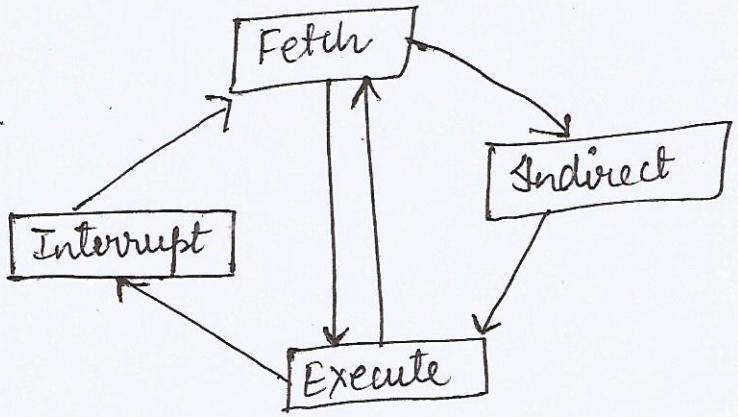
(1) **Fetch:** Read the next instruction from memory into processor.

(2) **Execute:** Interpret the opcode and perform the indicated operation

(3) **Interrupt:** If interrupts are enabled and an interrupt has occurred, save the current process state and service the interrupt.

(4) **Indirect** (additional) only when indirect addressing involved.

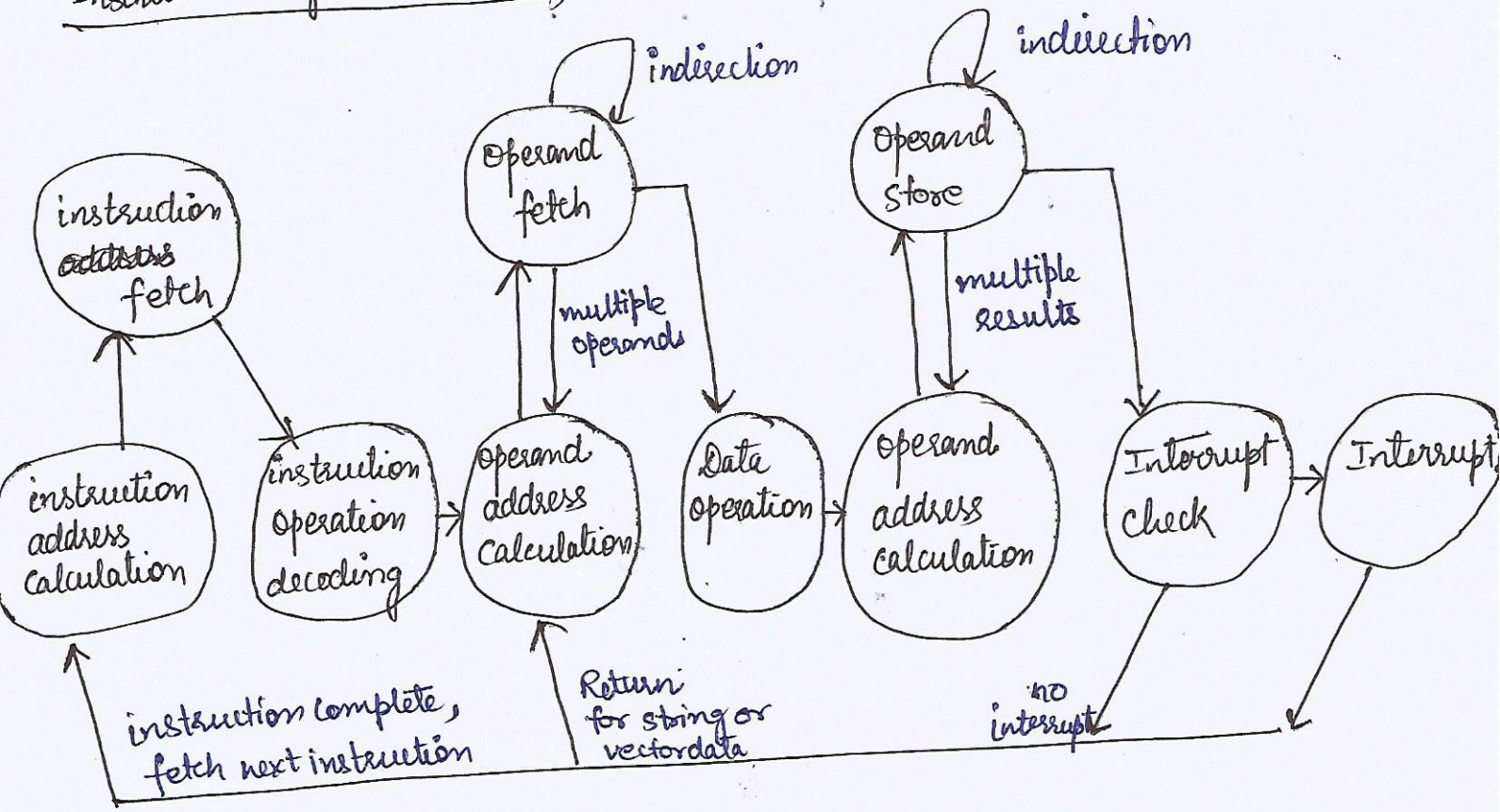
After an instruction is fetched, it is examined if any indirect addressing is involved or not, if involved, the required operands are fetched using indirect addressing. Following execution, an interrupt may be processed before next instruction fetch.



Instruction cycle (indirect addressing involved)

This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Instruction cycle state diagram:

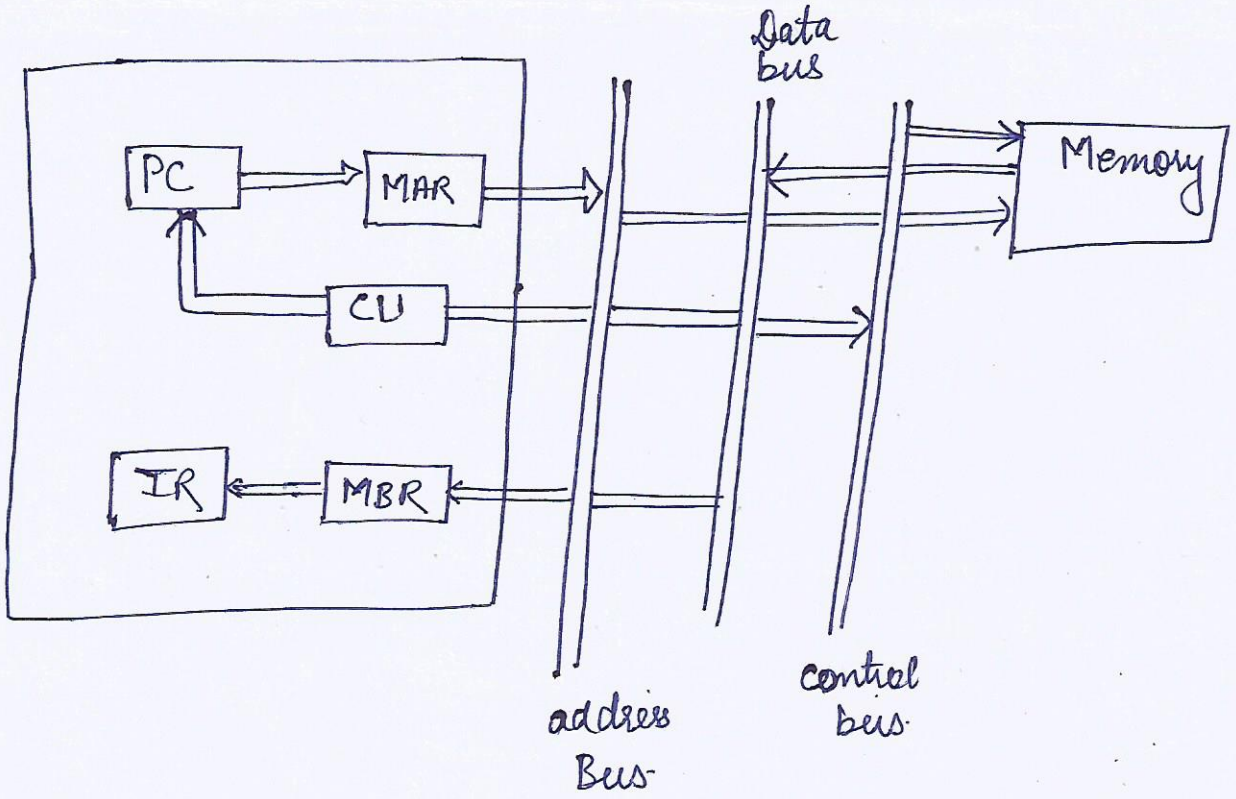


The exact sequence of events during an instruction cycle depends on the design of the processor.

Instruction Sub-cycle's Data flow:

Fetch cycle: During fetch cycle, an instruction is read from memory. PC (program counter) contains the address of next instruction to be fetched. This address is moved to MAR and placed on the address bus. The control unit requests a memory read and the result is placed on the data bus and copied to MBR and moved to IR. And then PC is incremented by 1.

Downloaded by Sangam gupta (sangam7318@gmail.com)

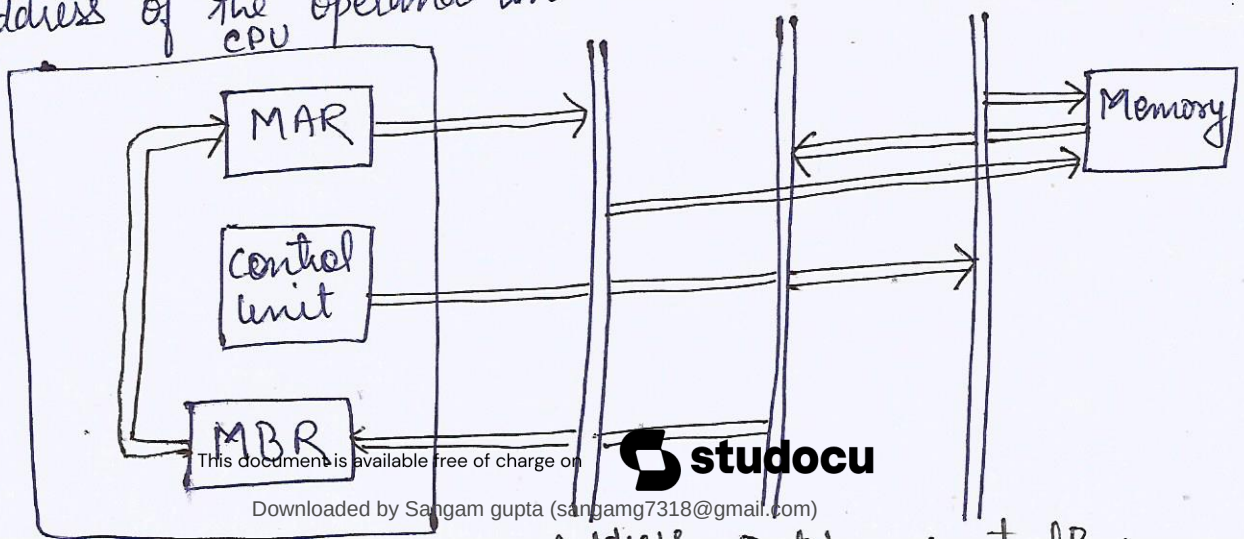


Data flow fetch cycle

Indirect Cycle:

After fetch cycle, control unit checks IR if it contains an operand specifier using indirect addressing, if so an indirect cycle is performed.

→ The right most N bits of MBR, which contains the address reference are transferred to the MAR.
Then control unit requests a memory read, to get the desired address of the operand into the MBR.



Dataflow indirect cycle

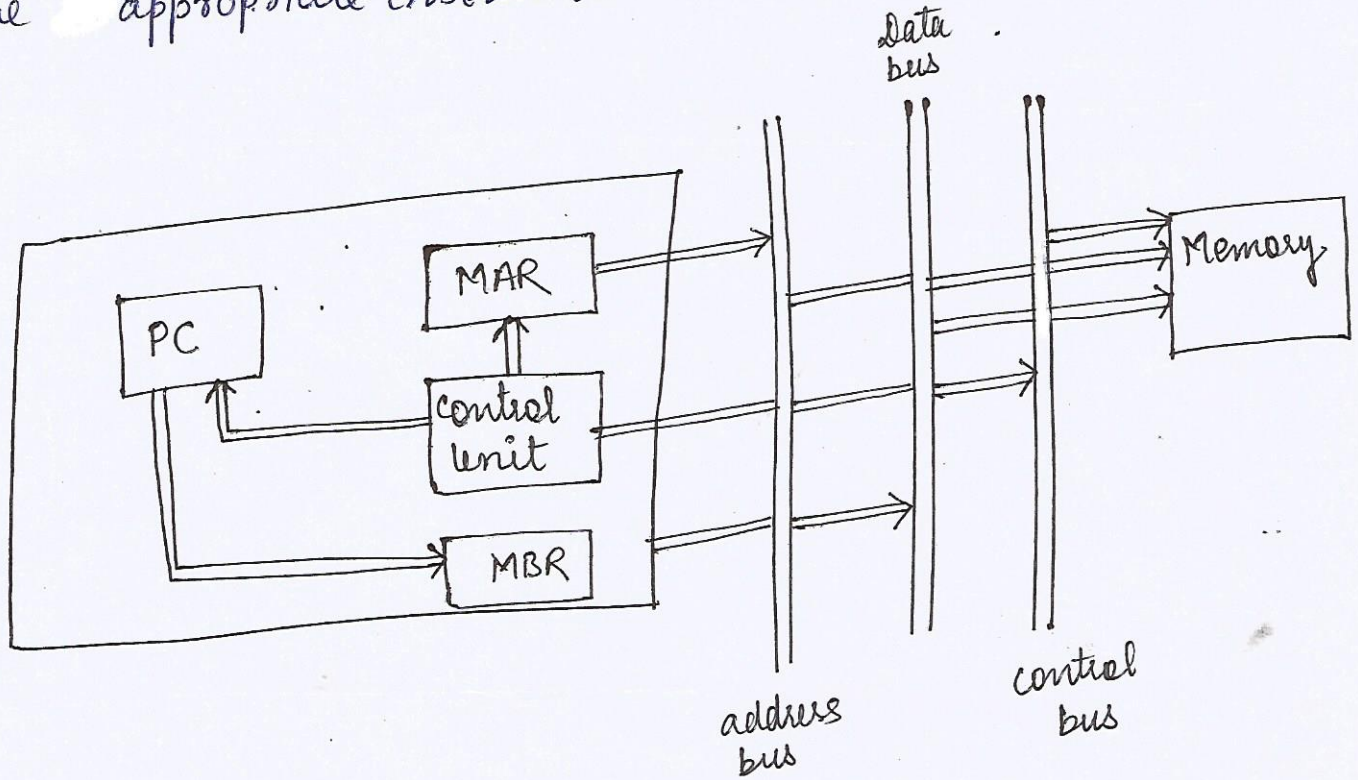
This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Interrupt cycle:

If an interrupt has occurred, the current content of PC must be saved so that the processor can resume normal activity after the interrupt.

Thus the content of PC are transferred to the MBR to be written into the memory. The special memory location reserved for this purpose is loaded into MAR from the control unit.

The PC is loaded with the address of the interrupt routine. As a result, the next instruction cycle will begin by fetching the appropriate instruction.



Data flow, interrupt cycle

Downloaded by Sangam gupta (sangamg7318@gmail.com)

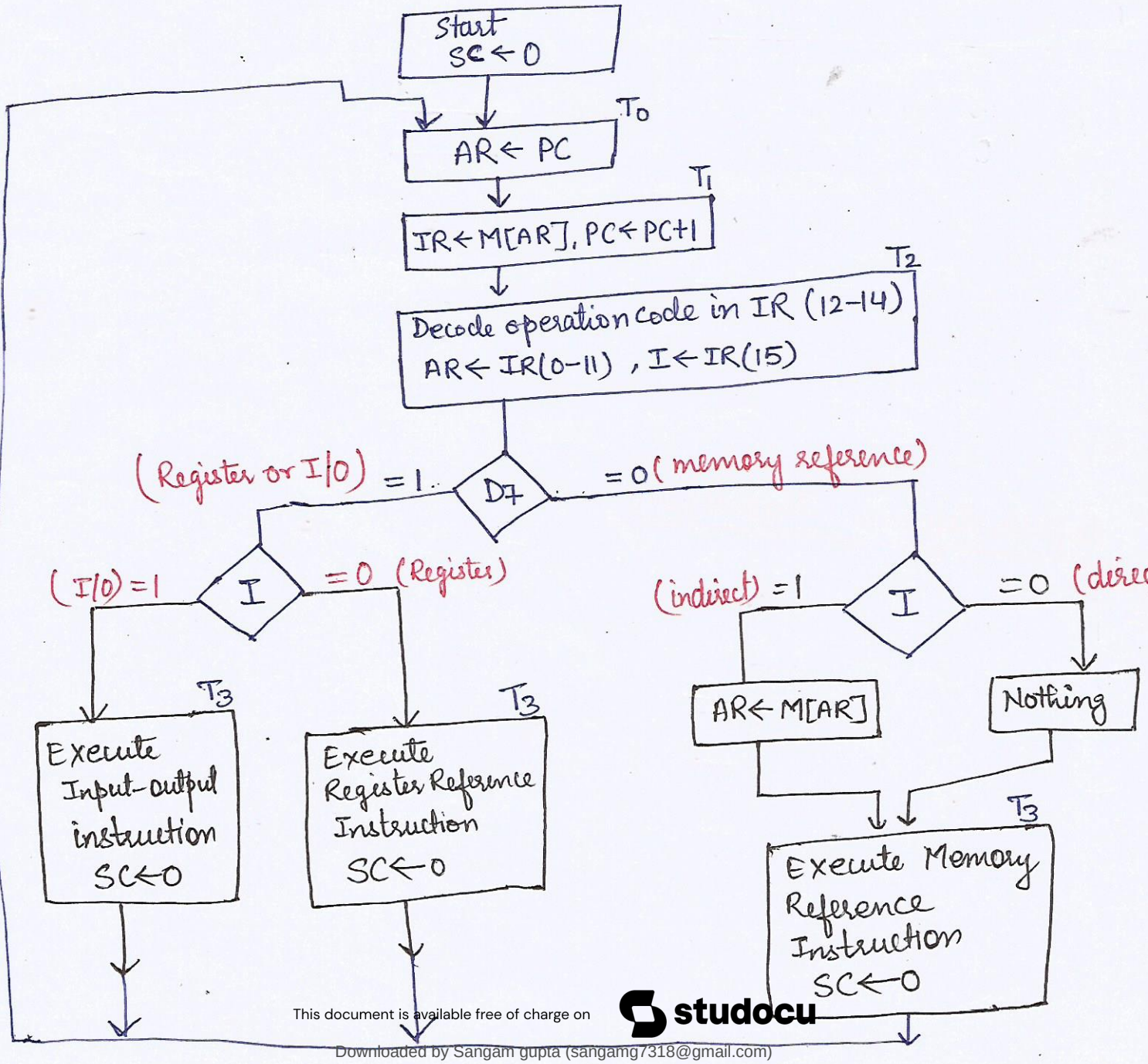
Flow chart of instruction cycle:

- An program in basic computer, each instruction cycle consists of the following:

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

After step 4, the control goes back to step 1 to fetch, decode and execute the next instruction.

This process continues unless a HALT instruction is encountered



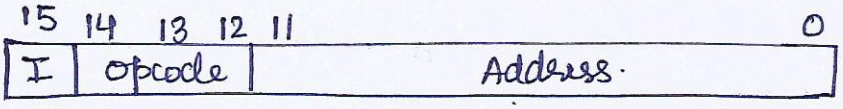
This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)

The flowchart represents initial configuration of instruction cycle. and shows how the control determines the instruction type after the decoding.

Types of instructions:

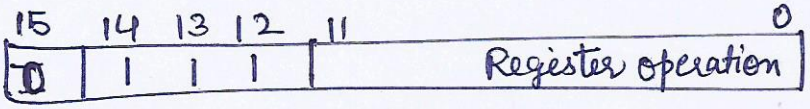
Memory Reference Instructions

Opcode from 000 to 110 i.e $D_7=0$



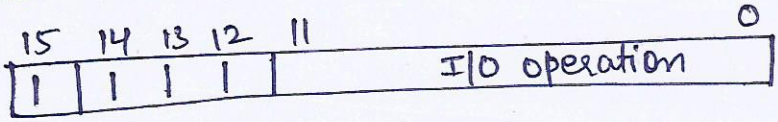
Register Reference Instruction

opcode 111 i.e $D_7=1$, $I=0$



Input-output Instruction

opcode 111 i.e $D_7=1$, $I=1$



Register Reference Instructions

(1)	CLA	$AC \leftarrow 0$	clear AC
(2)	CLE	$E \leftarrow 0$	clear E
(3)	CMA	$AC \leftarrow \overline{AC}$	complement AC
(4)	CME	$E \leftarrow \overline{E}$	complement E
(5)	CIR	$AC \leftarrow shr\ AC$, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$	circulate right

E ⇒ End carry

Downloaded by Sangam gupta (sangamg7318@gmail.com)

(6)	CIL	AC ← shl AC AC(0) ← E E ← AC(15)	circulate left.
(7)	INC	AC ← AC+1	increment AC
(8)	SPA	if (AC(15)==0) then PC ← PC+1	skip if positive
(9)	SNA	if (AC(15)==1) then PC ← PC+1	skip if negative
(10)	SZA	if (AC==0) then PC ← PC+1	skip if ~~Zero~~ AC Zero
(11)	SZE	if (E==0) then PC ← PC+1	skip if E zero
(12)	HLT	S ← 0 (S is a start-stop flip flop)	Halt computer

Memory Reference Instructions:

operation Decoder.

(1) AND — D_0

~~AC ← AC ∧ M[AR]~~

D_0T_4: DR ← M[AR]

D_0T_5: AC ← AC ∧ DR, SC ← 0

---

(2) ADD — D_1

D_1T_4: DR ← M[AR]

D_1T_5: AC ← AC + DR
E ← Cout, SC ← 0

---

(3) LDA — D_2

↑ Load to AC

D_2T_4: DR ← M[AR]

D_2T_5: AC ← DR, SC ← 0

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

(4) STA D_3 D_3T_4: $M[AR] \leftarrow AC$, $SC \leftarrow 0$.

↑ Store AC

---

(5) BUN D_4 D_4T_4: $PC \leftarrow AR$, $SC \leftarrow 0$

Branch unconditionally

---

(6) BSA D_5 D_5T_4: $M[AR] \leftarrow PC$, $AR \leftarrow AR+1$

D_5T_5: $PC \leftarrow AR$, $SC \leftarrow 0$

↑ Branch and Save Return address

---

(7) ISZ D_6 D_6T_4: $DR \leftarrow M[AR]$

D_6T_5: $DR \leftarrow DR+1$

D_6T_6: $M[AR] \leftarrow DR$, if $(DR=0)$ then $PC \leftarrow PC+1$, $SC \leftarrow 0$

Increment & skip if zero

Note that we need seven time signals (T_0 to T_6) to execute the longest instruction ISZ

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Input - output Instructions:

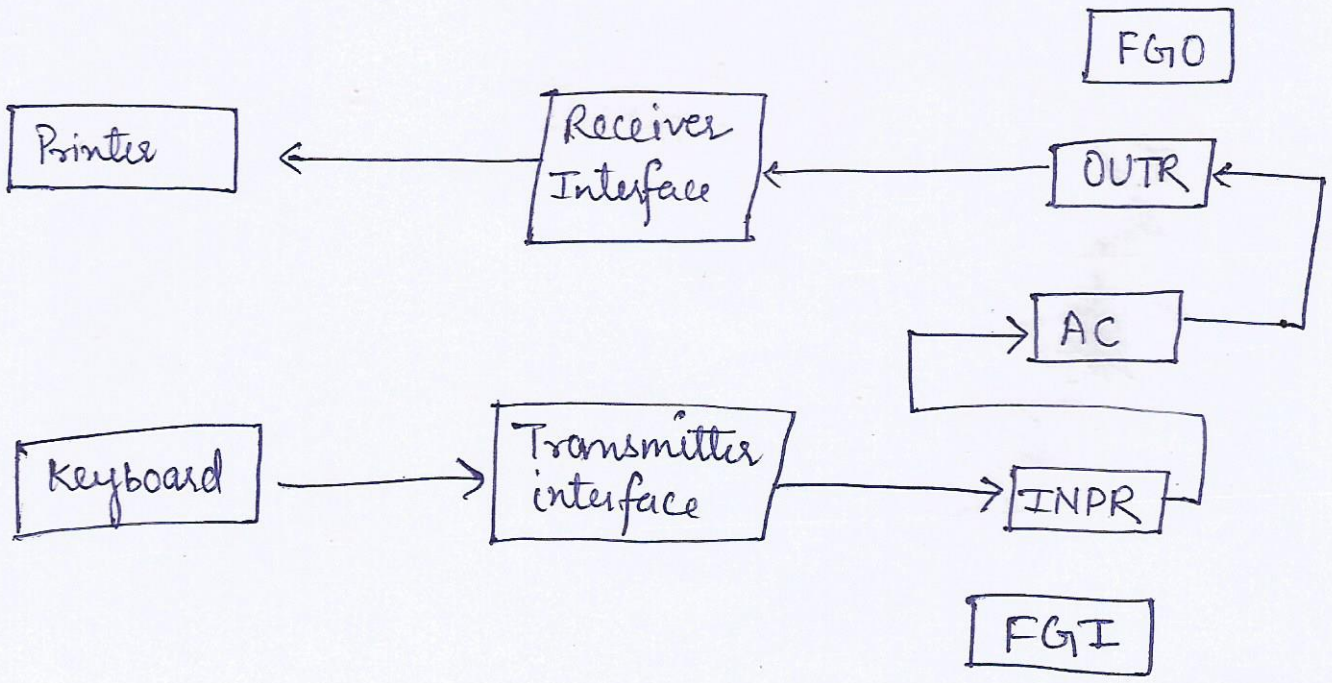
INP	AC(0-7) ← INPR , FGI←0	input character
OUT	OUTR ← AC(0-7) FGO←0	output character
SKI	if(FGI=1) then PC←PC+1	skip on input flag
SKO	If(FGO=1) then PC←PC+1	skip on output flag
ION	IEN←1	interrupt enable ON
IOF	IEN←0	interrupt enable Off

input - output configuration

input - output Terminal

Serial communication interface

Computer Register and flip flops.



This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

INPR ⇒ 8 bit input Register

AC ⇒ Accumulator

FGI ⇒ 1 bit control flip flop. This flag is set to 1 when new information is available. And, this flag is clear to zero when ~~of~~ information is accepted by the computer.

OUTR ⇒ output Register (8 bits)

FGO ⇒ is 1 bit control flip flop. If FGO is 1 then information ~~from~~ is transferred from AC to to OUTR. And then FGO is clear to zero.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Instruction codes:-

* The organization of a computer can be defined by its internal registers, the timing and control structure, and the set of instructions it uses.
* The internal organization of a digital system is defined by the sequence of micro operations it performs on data stored in its registers.

→ A computer instruction is a binary code that specifies a sequence of micro operations for the computer.

→ Instruction codes together with the data are stored in memory. (called Stored program concept)
The computer reads each instruction from memory ~~code~~. The control then interprets the binary code of instruction and proceeds to execute it by using a sequence of micro operations. Every computer has its own unique instruction set.

→ Instruction code is a group of bits that instructs the computer to perform a specific operation. It is usually divided into ~~two~~ parts

→ **Operation code:** operation code part of an instruction is a group of bits that define operations such as add, subtract, shift, complement or increment etc.

The number of bits in opcode = n will depend on the number of operations in computer

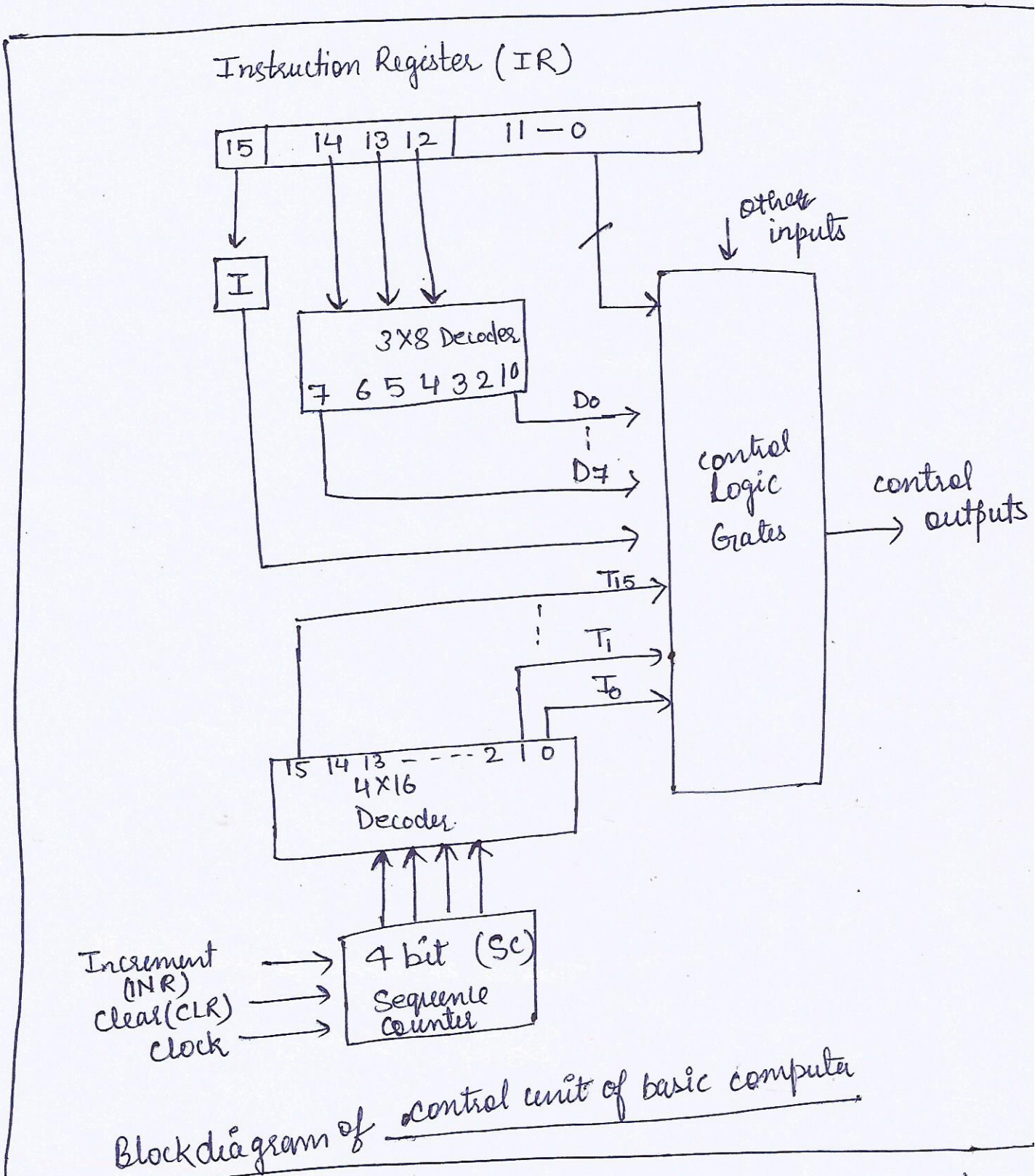
2^n ⇒ number of operations

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Control Unit:

control unit generates control signals. control signals provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and microoperations for the accumulator.



Block diagram of control unit of basic computer

→ components of control unit are

① → Two decoders

② → One sequence counter

③ → control logic gates.

Downloaded by Sangam Gupta (sangamg7318@gmail.com)

- An instruction read from memory is placed in the IR register.
- In control unit the IR is divided into three parts
  - I bit (Indirect bit)
  - the operation code bit (12-14)
  - bit 0 to 11

- The operation code bits are decoded by a 3×8 decoder as outputs D_0 to D_7.
- Bit 15 is transferred to a flip flop I.
- Bits 0 through 11 are applied to the control logic gates.
- The 4-bit Sequence counter can count in binary from 0 to 15. The outputs of counter are decoded into 16 timing signals T0 to T15

- The sequence counter SC can be incremented or cleared. Most of the time, the counter is incremented to provide the sequence of timing signals, Once in awhile, the counter is cleared to zero. causing the next active timing signal to be T0.

Example consider a case where SC is incremented to provide the signals T0, T1, T2, T3 and T_4 in sequence. At the time of T_4, SC is cleared to 0 if decoder output D_3 is active. This is expressed symbolically.

$$D_3T_4: SC \leftarrow 0$$

This document is available free of charge on studocu

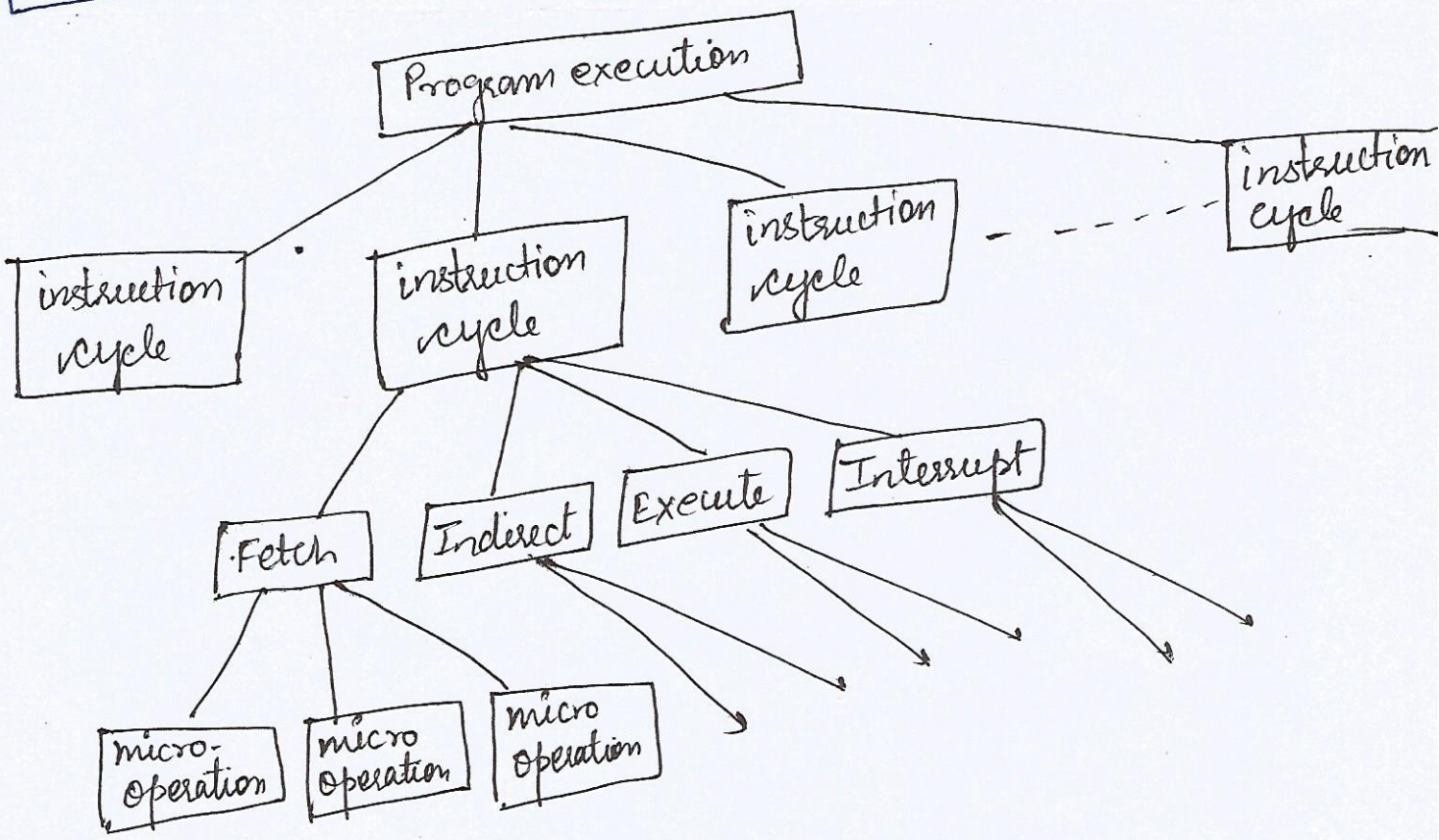
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Micro operation

→ The execution of an instruction involves the execution of a sequence of substeps, generally called cycles.

For example, an execution may consist of fetch, indirect, execute and interrupt cycles. Each cycle is in turn made up of a sequence of more fundamental operations called microoperations.

- A single microoperation generally involves a transfer between registers, transfer between a register and an external bus or a simple ALU operation



Downloaded by Sangam gupta (sangamg7318@gmail.com)

Ques: what is difference between operation (macro operation) and micro-operation.

Ans: An operation is the part of instruction. An operation is a binary code that tells the computer to perform the specific task or operation.

The control unit receives the instruction from memory and interprets the operation code bits. It then issues a sequence of control signals to initiate micro operations in internal computer registers.

For every operation code, the control issues a sequence of micro operations. For this reason, an operation code is sometimes called a macro operation because it specifies a set of micro operations.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

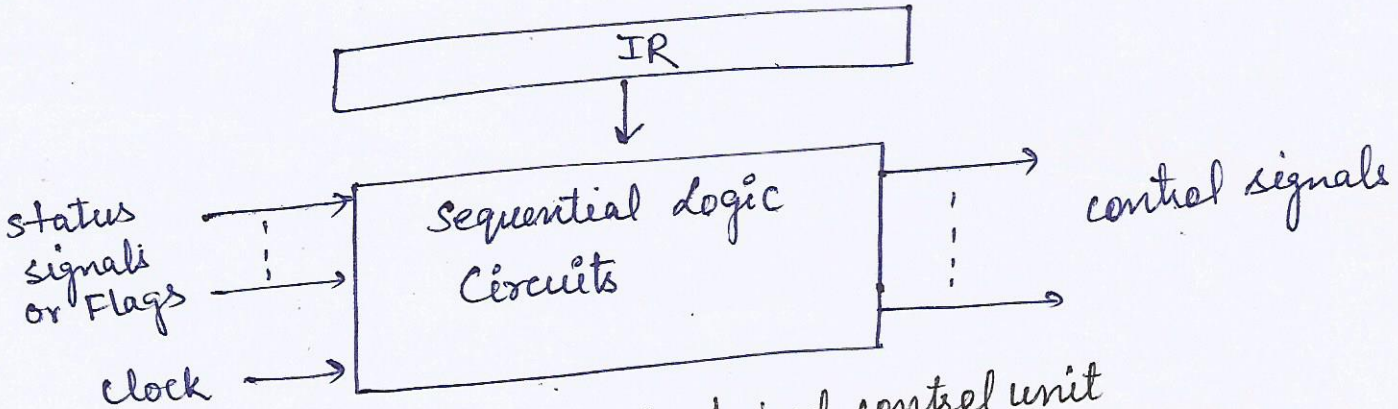
Hardwired and Microprogrammed control:

→ There are two types of control organization. A control unit can be implemented using two techniques:

- → Hardwired control unit
- → Microprogrammed control unit

Hardwired control:

- In hardwired organization, the control unit is implemented with gates, flip flops, decoders and other digital circuits.
- When the control signals are generated by the hardware using conventional logic design techniques, the control unit said to be hardwired.



Structure of a hardwired control unit

→ Hardwired control unit uses a fixed logic to interpret an instruction and generate appropriate signals.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Microprogrammed control

A microprogrammed control unit is implemented using programming approach.

→ The control unit initiates a series of sequential microoperations. The control variables can be represented by a string of 1's and 0's called a controlword.

A microprogrammed control unit is a control unit whose binary control variables are stored in memory

Control memory: A memory that is part of control unit is called control memory. control memory is the storage in the microprogrammed control unit to store the micro program.

micro-instruction: Each word in control memory contains a micro instruction. A micro instruction specifies one or more microoperations for the system.

micro program: A sequence of micro instructions constitutes a microprogram.

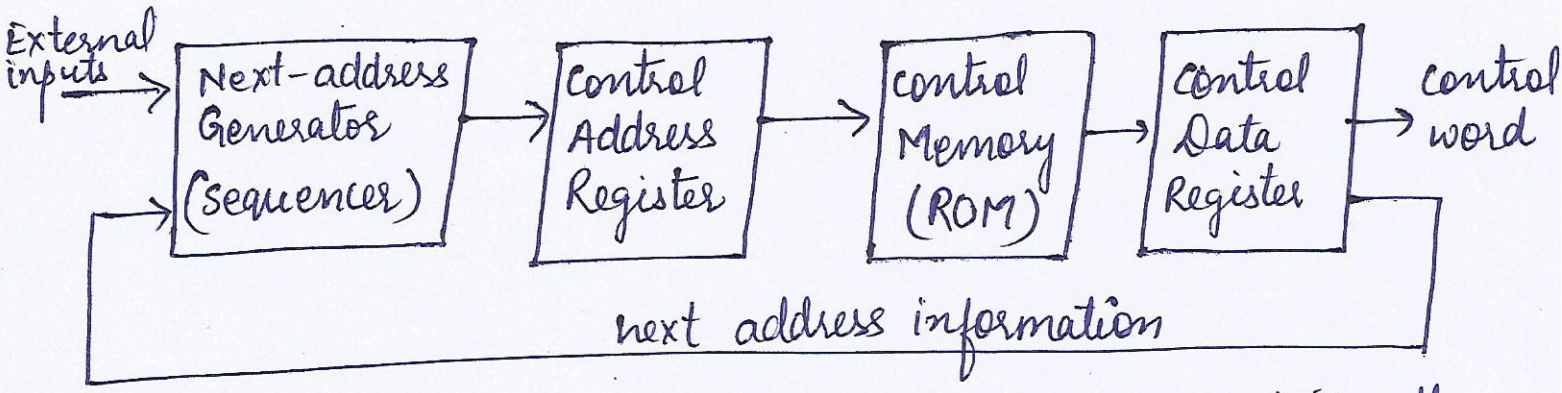
Microcode: A logically coherent portion of micro program is called micro-code.

A micro instruction can cause execution of one or more micro operation and a sequence of micro instruction (micro program) can cause the execution of an instruction.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

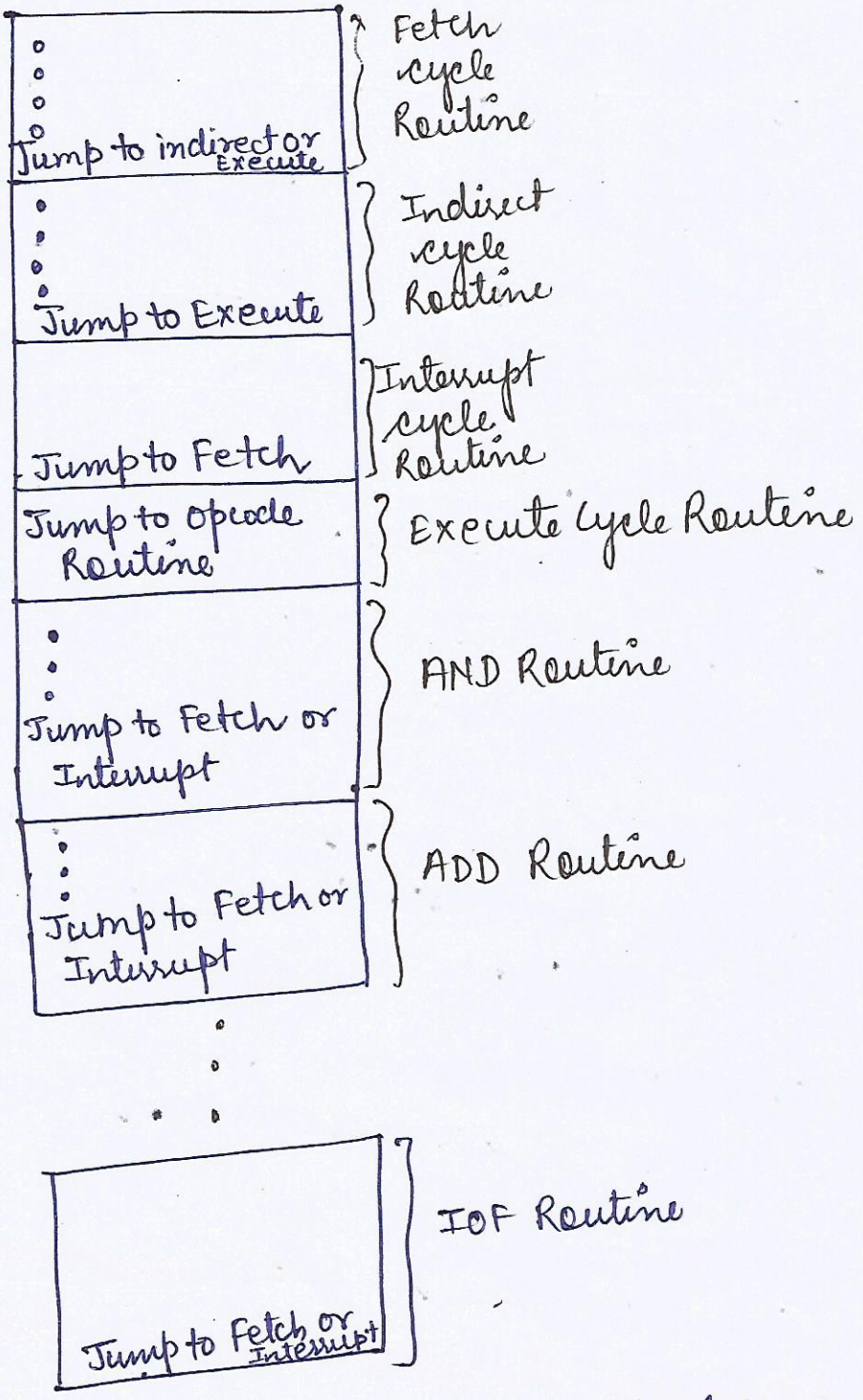
Micro programmed control organization



- The control memory is assumed to be a ROM, within which all control information is permanently stored.
- The control address register specifies the address of the microinstruction and control data register holds the microinstruction read from memory.
- The microinstruction contains a control word that specifies one or more micro operations for the processor. Once these operations are executed, the control must determine the next address.
- The location of the next micro instruction may be the one next in the sequence or it may be located somewhere else in the control memory.
- The next address generator is sometimes called a micro program sequencer as it determines the address sequence that is read from the control memory.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Organization of Control Memory:



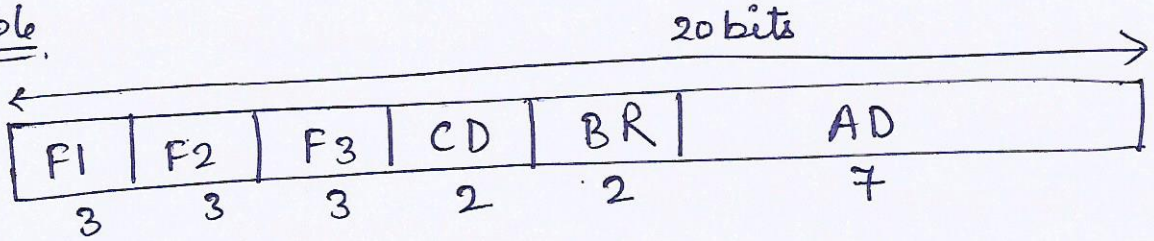
This shows control words or microinstruction arranged in control memory. The microinstruction in each routine are to be executed sequentially. Each Routine ends with a branch or jump instruction indicating where to go next.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Microinstruction Format :

Example.



→ F1, F2, F3 ⇒ specify micro operations
each F defines seven distinct micro operations
None for zero (000)
Different operations for (1 to 6)

→ CD ⇒ Field selects status bit conditions

→ BR ⇒ field specifies the type of branch to be used

→ AD ⇒ This field contains a branch address. The size of this field depends on the control memory size.
Here AD has 7 bits that means ⇒ 2^7 words in control memory.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

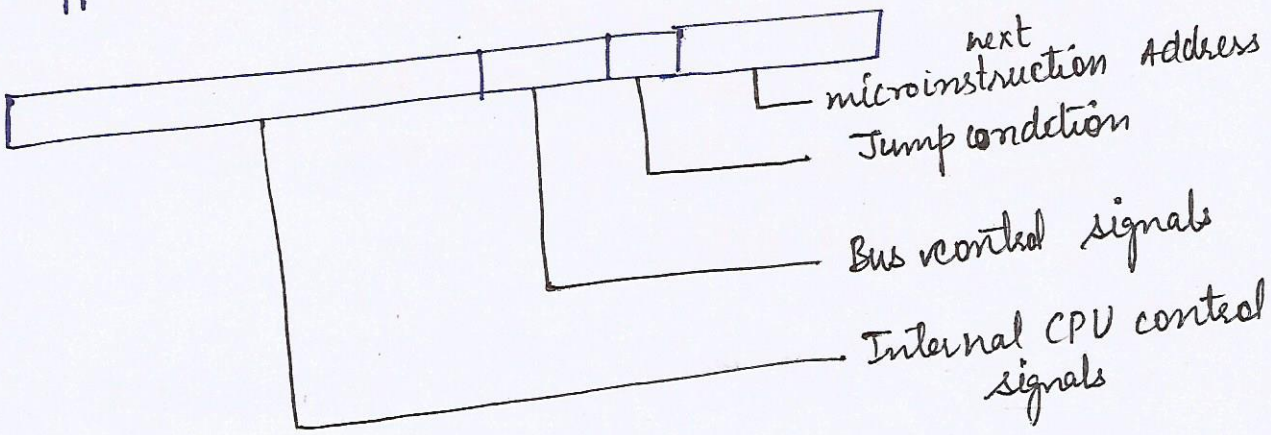
Horizontal Microprogrammed Control unit:

→ The control signals are represented in decoded binary format that is 1 bit per one control signal.

→ If 53 control signals are present in the processor then 53 bits are required

→ More than 1 control signal can be enabled at a time.

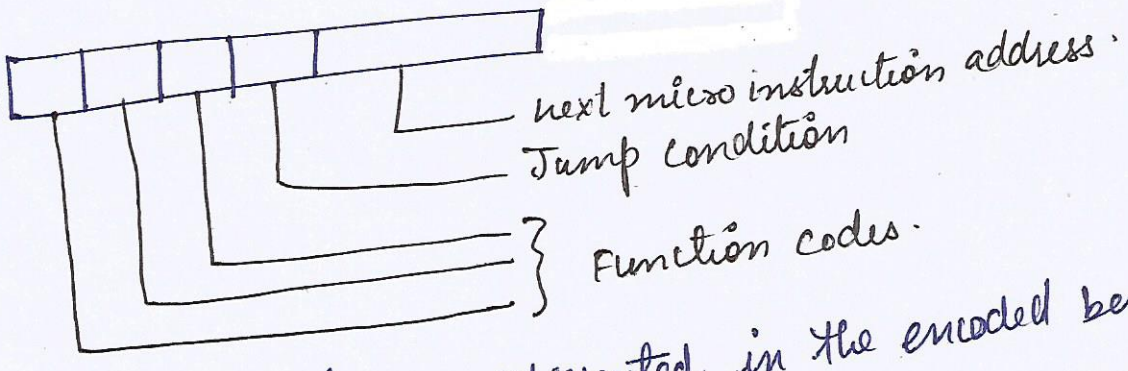
→ It supports longer control word.



→ It requires no additional hardware (decoders).

→ It is faster than Vertical microprogrammed.

Vertical Microprogrammed control unit:



→ The control signals are represented in the encoded binary format.

This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)

- For N control signals $\log_2 N$ bits are required.
- It supports shorter control words.
- It requires an additional hardware (decoders) to generate control signals.
- It is slower than horizontal microprogrammed.

Ques: suppose there are 64 signal control signals in the system and control 1024 word control memory then find out the size of microinstructions horizontal and vertical both.

Horizontal:
- 64 bits for 64 control signals
- Address bits ⇒ 10 because words = $1024 = 2^{10}$

size of microinstruction

control signal bits	Address bits
64	10

⇒ 74 bits

Vertical:

control signals = $64 \Rightarrow 2^6$

Number of bits for control signals = 6.

Address bits ⇒ 10. because $1024 \Rightarrow 2^{10}$.

size of vertical micro instruction ⇒

control signal bits	Address bits
6	10

⇒ 16 bits

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Ques: consider a Microprogrammed CU, where 1024 word control memory is used. The CU has to support 50 control signals and 16 flag conditions.

(i) How many bits required in control word.

(ii) what is the size of control memory

Ans:

Format of control word or microinstruction:

control signals	condition flags	Address
bits	bits	bits

Horizontal microprogramming:

size of control word or microinstruction ⇒ | 50 | 4 | 10 | ⇒ 64

Number of bits for control signals ⇒ 50 (bit per control signal)

Number of bits for flags ⇒ 4 because $16 \Rightarrow 2^4$

Number of address bits ⇒ 10 because $1024 = 2^{10}$.

The size of each control word ⇒ 64 bit.

control memory has words ⇒ 1024

Size of control memory = 1024 X 64 bits

⇒ 1KB X 8B

⇒ 8KB

Vertical Microprogramming:

control signal bits ⇒ 6 because Control signals ⇒ 50

flag bits ⇒ 4

Address bits ⇒ 10

size of control word ⇒ 20 bits ⇒ (6+4+10)

size of control memory ⇒ 20 X 1024

≅ 3 KB

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

→ Micro program Sequencer:

→ The next address generator is sometimes called a microprogram sequencer, as it determines the address sequence that is read from control memory.

→ Typical functions of a microprogram sequencer are:

- → Incrementing the control address register by one.
- → Loading into the control register an address from control memory
- → Transferring an external address
- → Loading an initial address to start the control operations.

→ Address Sequencing:

→ Microinstructions are stored in control memory in groups, with each group specifying a routine.

→ Each computer instruction has its own microprogram routine in control memory to generate the microoperations that execute the instruction

→ Steps that the control must undergo during the executing of a single computer instruction.

- → An initial address is loaded into the control register address register when power is turned on in the computer
- → This address is usually the address of the first micro instruction that activates the instruction fetch routine.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

→ The control unit must go through the routine that determines the effective address of the operand.

→ The next step is to generate the micro operations that execute the instruction fetched from the memory.

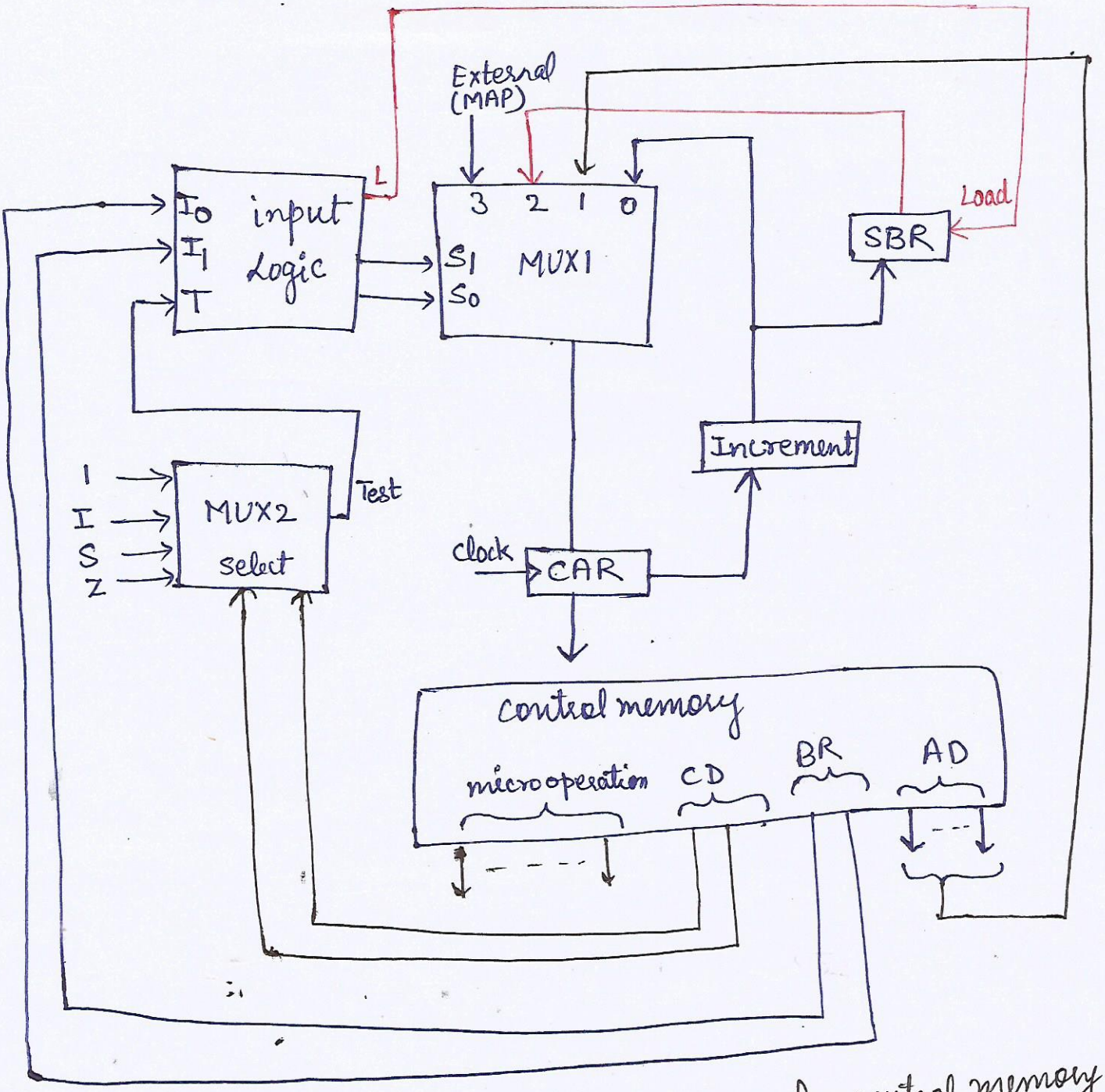
Transformation from the instruction code bits to an address in control memory where the routine is located is referred to as mapping process

→ Address sequencing capabilities required in a control memory are:
  → Incrementing the CAR
  → Unconditional branch or conditional branch, depending on status bit conditions
  → Mapping process from the bits of the instruction to an address for control memory.
  → A facility for subroutine call or return.

→ The micro-instruction contains
  → a set of bits to initiate microoperations in control registers.
  → other bits to specify the method by which the next address is obtained.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)



micro program sequencer for control memory

Explanation:

- The first multiplexer selects an address from one of four sources (incremented, External, from micro instruction or SBR) and routes it into CAR.
- The second multiplexer tests the value of a selected status bit and the result of test is applied to input logic.
- SBR ⇒ Sub Routine Register.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

CD field of microinstruction selects one of the status bits in the second multiplexer.

CD	condition	Symbol	
00	always = 1	U	unconditional branch
01		I	Indirection bit
10		S	Signbit
11		Z	Zero value in AC

BR ⇒ Branch type.

00	JMP
01	CALL
10	RET
11	MAP

Input logic design ⇒

$S_1 = I_1$

$S_0 = I_1 . I_0 + \overline{I_1} T$

$L = \overline{I_1} I_0 T$

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Reduced Instruction Set Computer (RISC)

The concept of RISC architecture involves an attempt to reduce execution time by simplifying the instruction set.

Characteristics of RISC Processors:

→ Relatively few instructions
→ Relatively few addressing modes
→ Memory access limited to load and store instructions
→ All operations done within the register of CPU
→ Fixed length, easily decoded instruction format
→ Single cycle instruction execution
→ Hardwired rather than microprogrammed.
→ Simple load and store operations for memory access.

Complex Instruction Set Computers (CISC)

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

Characteristics of CISC architecture:

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

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Pipelining:

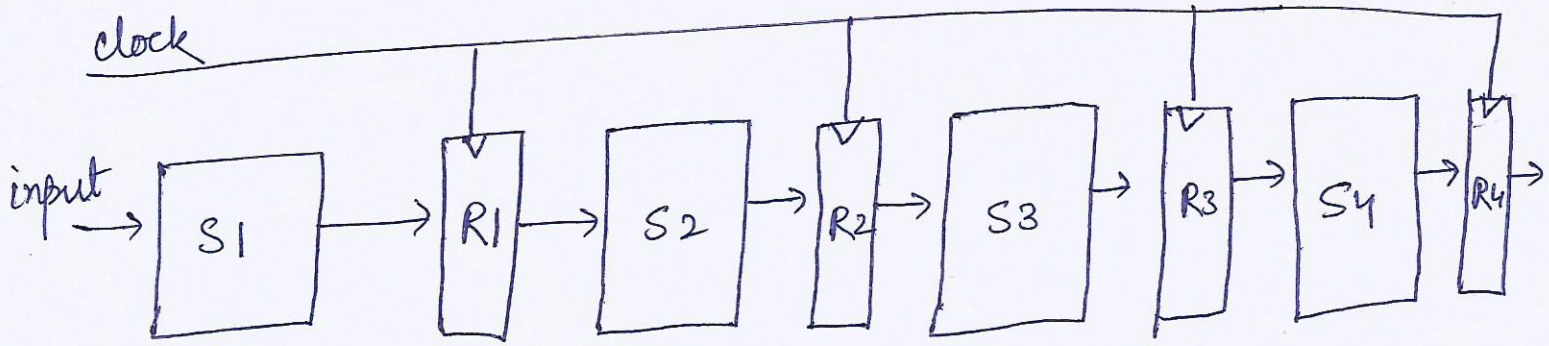
→ Pipeling is an implementation technique whereby the multiple instructions are overlapped in execution.

→ Pipeline processing is an implementation technique where arithmetic suboperations or the phases of a computer instruction cycle overlap in the execution.

→ Pipeline is divided into stages or segments and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end.

→ Pipeline increases the overall instruction throughput.

→ In pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and combinational circuit performs operations on it. The output of the next segment combinational circuit is applied to the input register of the next segment.



Four segment pipeline

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Types of pipeline:

→ Arithmetic Pipeline
→ Instruction Pipeline

Arithmetic pipeline:

→ found in very high speed computers
→ Arithmetic suboperations overlapping.
→ used to implement floating point operations, multiplication of fixed point numbers ~~and~~ etc.

Instruction pipeline:

- In this, a stream of instructions can be executed by overlapping fetch, decode and execute phases of an instruction cycle.
- ~~increase~~ increases throughput of the system.

Advantages of pipelining:

- The cycle time of processor is reduced.
- It increases throughput of the system.
- It makes system reliable.

Disadvantages of pipelining

→ The design of a pipelined processor is complex and costly to manufacture.
- Instruction latency is more.

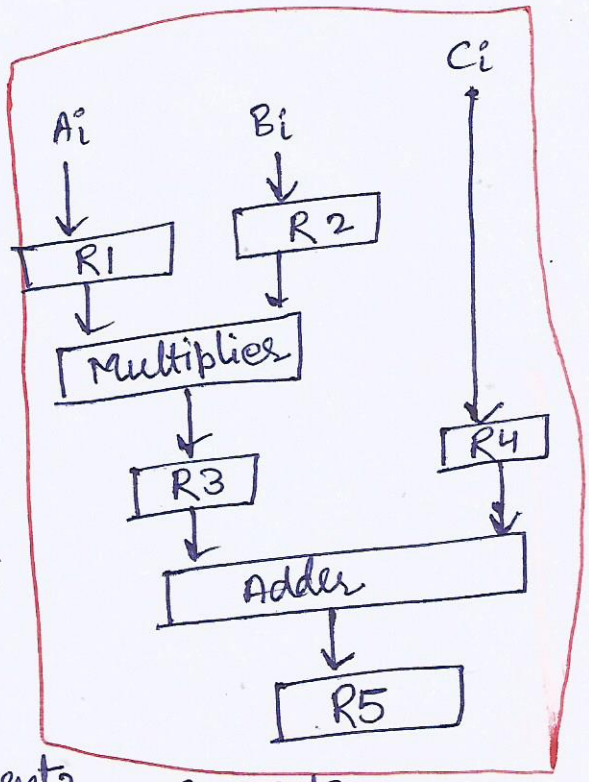
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Example for pipeline organization

- Example of performing combined multiply and add operations with a stream of numbers.

$A_i * B_i + C_i$ for $i \leftarrow 1, 2, 3 - - - 7$.

Each suboperation is to be implemented in a segment with a pipeline.



- Suboperations

$R1 \leftarrow A_i \quad R2 \leftarrow B_i$

$R3 \leftarrow A_i * B_i \quad R4 \leftarrow C_i$

$R5 \leftarrow R3 + R4$

The five registers are loaded with new data every clock pulse.

Clock pulse	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1			—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
⋮					

so on.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

General Considerations :

⇒ some applications need to repeat the same task many times with different sets of data.

⇒ The operands pass through all the segments in a fixed sequence. Each segment consists of a combinational circuit S_i that performs a suboperation over the data stream flowing through the pipe.

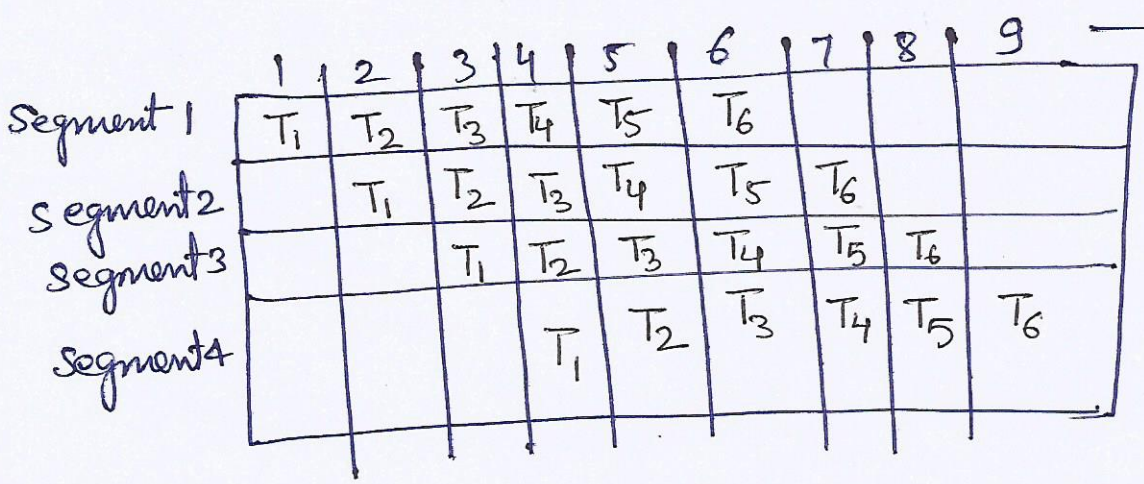
⇒ The segments are separated by registers R_i that hold the intermediate results between the stages.

⇒ Information flows between adjacent stages under the control of a common clock pulse applied to all registers simultaneously.

Task: a task as the total operation performed going through all the segments in the pipeline.

Space-time diagram: (For four pipeline stages)

⇒ shows the segment utilization as a function of time.



Assumed Number of task = 6

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Instruction Pipeline

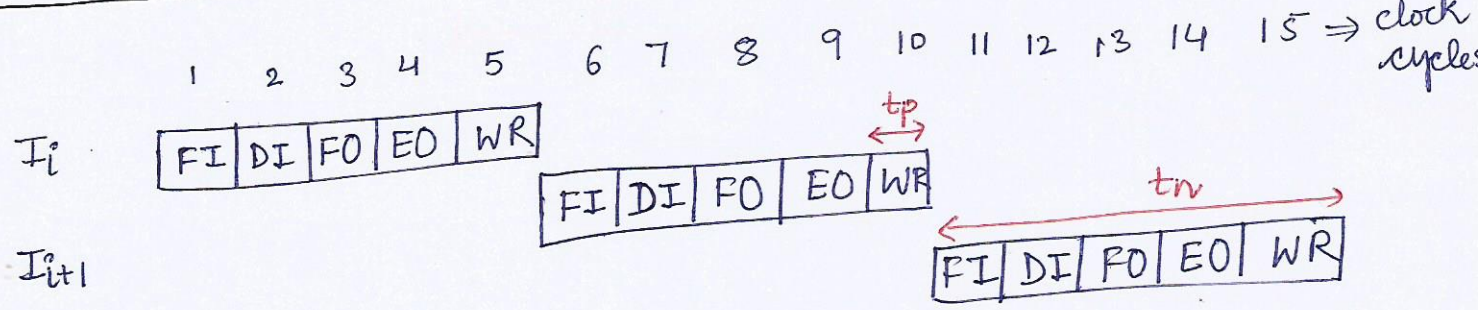
⇒ The phases of a computer instruction cycle overlap in the execution.

Instruction cycle ⇒ 5 phases.

FI — Fetch instruction
DI — Decode instruction
FO — Fetch operands
EO — Execute operation
WR — Write Result.

Suppose there are three instructions to execute I_i, I_{i+1}, I_{i+2}

Instruction execution in nonpipeline architecture



Number of cycles to complete 3 tasks ⇒ 3X cycle to complete 1 task.

⇒ 3X5

= 15

Time to execute 3 tasks ⇒ 3X t_n

= 3X time to execute or complete one task

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Instruction execution in pipeline architecture

	1	2	3	4	5	6	7
I_i	FI	DI	FO	EO	WR		
I_{i+1}		FI	DI	FO	EO	WR	
I_{i+2}			FI	DI	FO	EO	WR

⇒ clock cycles.

Number of cycle required to complete n tasks in K-stage.

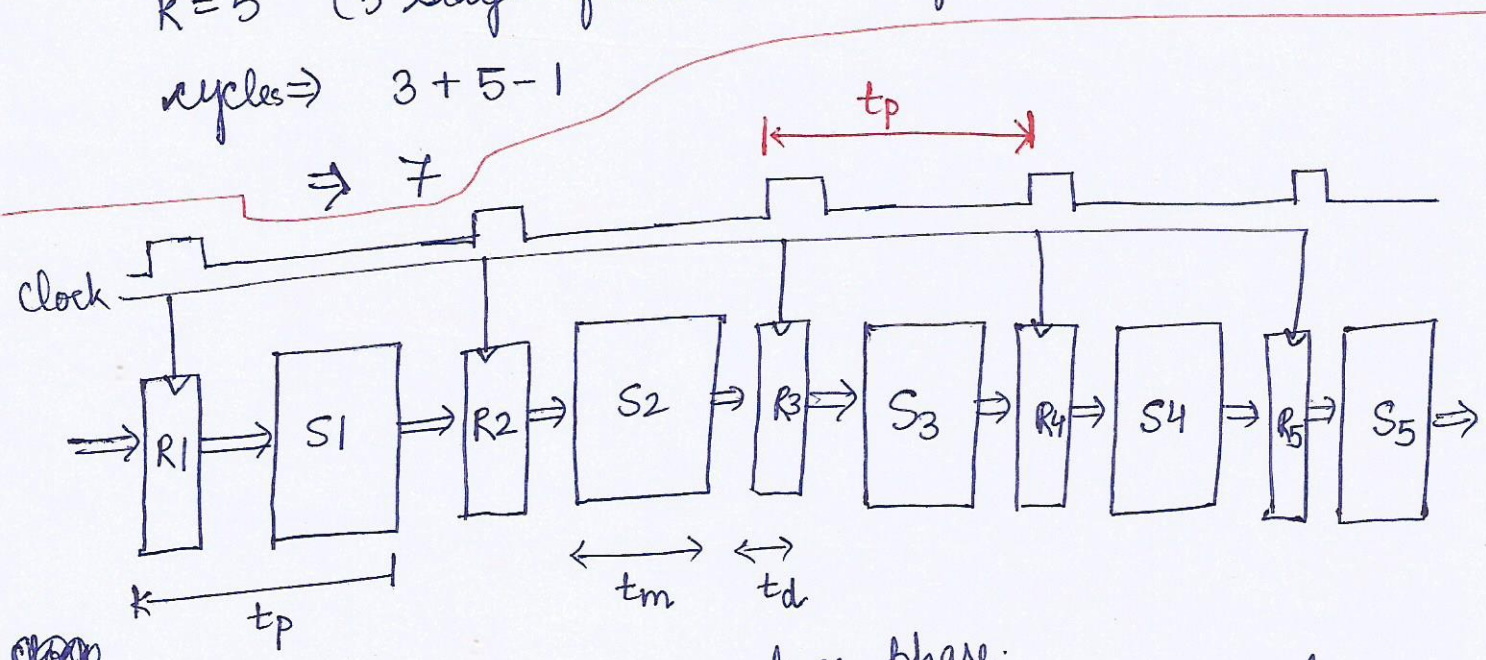
pipeline $= K+n-1$

$n = 3$ (3 instructions)

$K = 5$ (5 stages of instruction cycle)

cycles ⇒ $3+5-1$

⇒ 7



t_p ⇒ time taken by a segment or phase.

t_m ⇒ time taken by combination circuit of segment.

t_d ⇒ propagation delay of register.

$$t_p = \max_{i=1}^{k}(t_i) + t_d$$

t_p = max time taken by any segment + Register delay

5-Segment pipeline

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Formulas:

consider k-segment pipeline (or phases) ⇒ means Number of segments ⇒ k

Number of tasks to be executed = n

clock cycle time = t_p

Time taken by task 1 to complete ⇒ $k \cdot t_p$

The remaining (n-1) tasks, time equal to $(n-1) \cdot t_p$

The time taken by n tasks using k-segment pipeline

$$= k t_p + (n-1) t_p$$

$$= [k + (n-1)] \cdot t_p$$

Number of cycle by n tasks using k-segment pipeline

$$\Rightarrow k + (n-1)$$

consider a non-pipeline unit that performs the same task and takes a time equal to t_n to complete each task.

then time required to complete n tasks. ⇒ $n t_n$

$$\text{speedup ratio } S = \frac{\text{time taken by non-pipeline}}{\text{time taken by pipeline system}}$$

As the number of tasks increases n becomes much larger than k-1, and k+n-1 approaches the value of n

under this condition the speedup ⇒ $S = \frac{t_n}{t_p}$

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

we assume that the time it takes to process a task is the same in the pipeline and non-pipeline circuit, we will have

$$t_n = k t_p \Rightarrow$$

Including this assumption the speedup reduces to

$$S = \frac{k t_p}{t_p} = k$$

This shows that theoritical maximum speedup that a pipeline can provide is k (number of segment in pipeline)

Frequency $= \frac{1}{t_p}$

Throughput $H_k = \frac{n}{(k+n-1)\,t_p}$ or $\frac{nf}{(k+n-1)}$

Throughput: The number of tasks performed per time unit

Efficiency $= \frac{S_k}{k}$

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Latency: The number of time units (clock cycle) between two initiations of a pipeline is the latency between them.

Collision: Any attempt by two or more initiations to use the same pipeline stage at the same time will cause a collision

Forbidden Latencies:- that will cause collision

Permissible latencies:- that will not cause collision

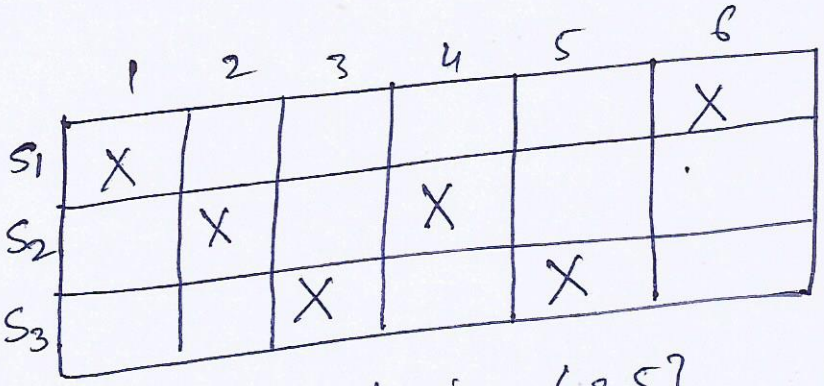
Reservation Table: A reservation table is a two-dimensional table

Row ⇒ pipeline stage or processing element

Column ⇒ represents cycles in the execution of a task

An X is placed in the entries corresponding to the stages required by the task during that cycle.

⇒ Forbidden latencies ⇒ distance between two X (check marks) in the same row of the reservation table.



⇒ 1, 6 — 6-1 ⇒ 5

⇒ 2, 4 — 4-2 ⇒ 2

⇒ 3, 5 — 5-3 = 2.

Forbidden latencies = {2, 5}

Permissible latencies ⇒ {1, 3, 4}

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

- if reservation table has n columns, the maximum forbidden latency is $m \leq (n-1)$
- permissible latency p will satisfy $1 \leq p \leq m-1$

collision vector collision vector is an m-bit vector
m = maximum forbidden latency.

$$C = C_m C_{m-1} - - - - C_2 C_1$$

$C_i = 1$ if latency i is a forbidden latency.

$C_i = 0$ otherwise

Previous Example: Forbidden latencies = {2,5}

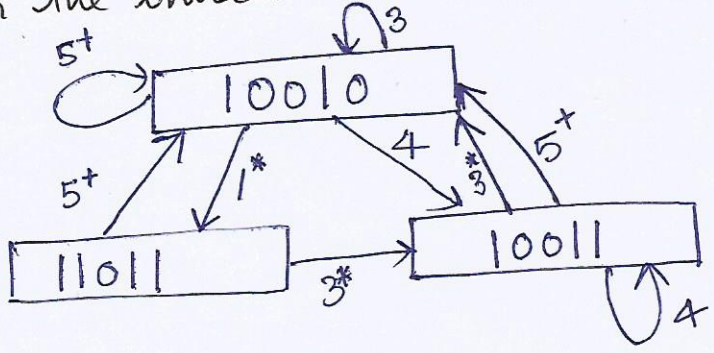
$$C = C_5 C_4 C_3 C_2 C_1$$
$$1\ 0\ 0\ 1\ 0$$

State Diagram:

From collision vector, we can construct a specifying the permissible state transitions among the successive iterations

⇒ collision vector forms the initial state

⇒ The next state at time t+p is obtained by shifting the present state p-bits to the right and ORing with the initial collision vector.



5+ means 5 or more than 5 bits shift

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Latency Sequence: A latency sequence is a sequence of permissible non forbidden latencies between successive task initiations.

Latency Cycle: A latency cycle is a latency sequence which repeats the same subsequence (cycle) indefinitely.

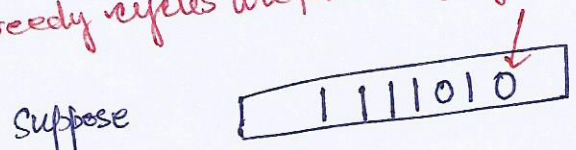
constant cycle:- A constant cycle is a latency cycle which contains only one latency value.

Simple cycle: A simple cycle is a latency cycle in which each state appear only once.

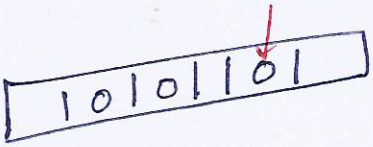
Greedy cycle: Some of the simple cycles are greedy cycles.

A greedy cycle is one whose edges are all made with the minimum latencies from their respective starting states.

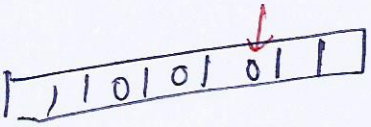
Greedy cycles are formed only using outgoing edges with minimum latencies.



minimum latency (permissible) = 1



min latency ⇒ 2.



min latency = 3

Note

Minimum latency respective to state is marked with * in state diagram.

- when number of shifts is m+1 or greater, all transitions are redirected back to the initial state.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Ques:

How to draw state diagram

collision vector 10010 ⇒ forms initial state.
it has permissible latency ⇒ 1, 3, 4

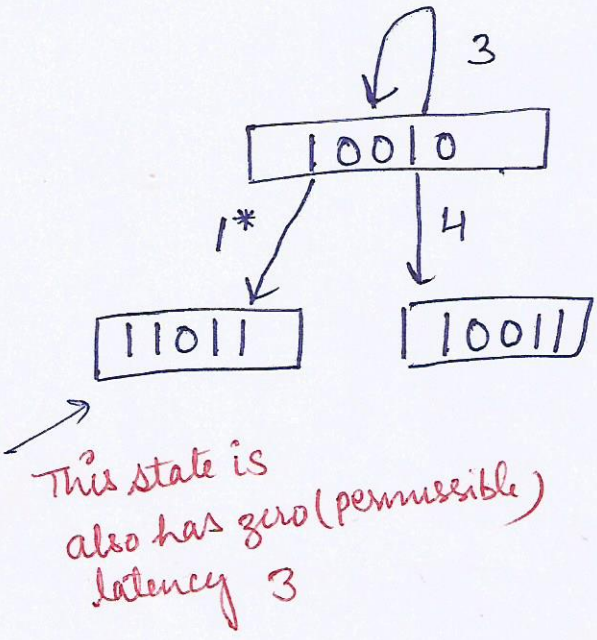
① do shifts 1, 3, or 4
② ORing with CV

initial state 10010

case 1 ⇒ 10010
↓ 1 shift Right
01001
10010 ORing
11011 next state

Case 3 ⇒ 10010 — CV
00010 — 3 shift Right
10010 — CV (ORing)
10010 — next state

Case 4 ⇒ 10010
00001 — 4-shift Right
10010
10011 — next state



Downloaded by Sangam gupta (sangamg7318@gmail.com)

State ⇒ 11011

Case 3 shift of 3.

00011 ⇒ shift of 3.
10010 ⇒ CV
10011 ⇒ ORing ⇒ next state (already present state)

State : 10011 ~~shift~~ Permissible latency ⇒ 3, 4.

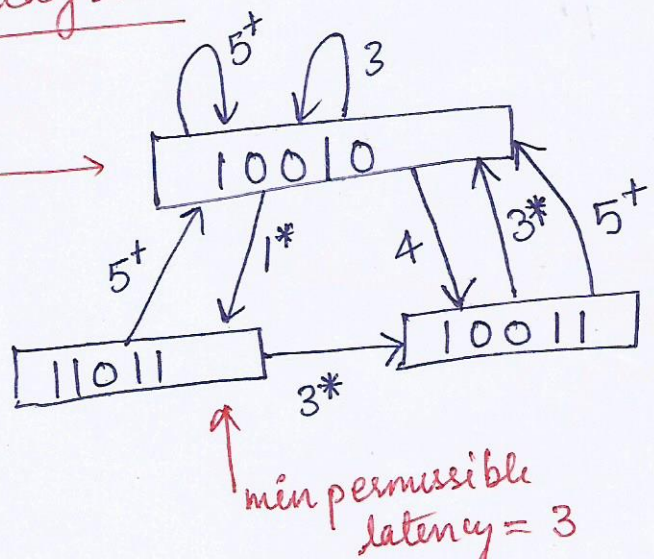
Case 3:- shift of 3

10011 → 3 shifts Right.
00010
10010 ORing.
10010

Case 4: shift of 4.

10011 → 4 shifts Right
00001
10010 — ORing
10011

Complete state diagram



minimum permissible latency of this state = 1 so marked with star.

min permissible latency ⇒ 3

min permissible latency = 3

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Simple cycle:

(3), (5), (1,5), (1,3,5), (4,3), (4,5) and so on.

Constant cycle:

(3), (5)

Greedy cycles:- Examples ⇒ (1,5), (3,4)

MAL Minimum Average Latency

Simple cycles	Average latency
3	3
5	5
(1,5)	$\frac{1+5}{2} = 3$
(1,3,5)	$\frac{1+3+5}{3} = 3$
(4,3)	$\frac{4+3}{2} \Rightarrow 3.5$

So MAL = 3

Throughput = $\frac{1}{MAL} = \frac{1}{3}$ ⇒ 16.67 million operations per second.

Throughput = if minimum constant cycle used.

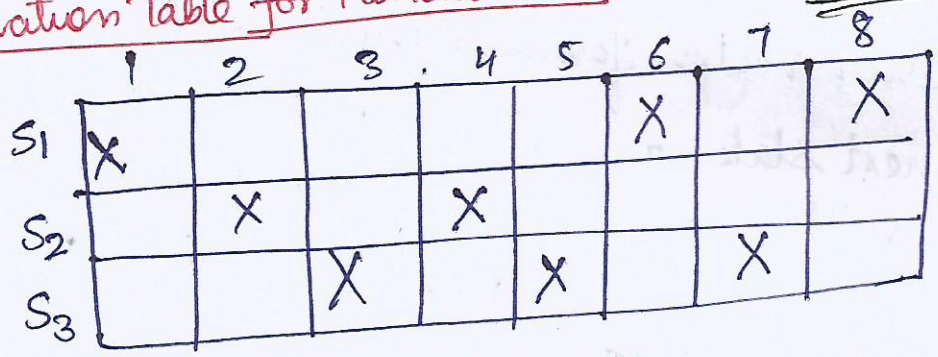
minimum constant cycle = 3.

Throughput = $\frac{1}{3}$ MOPS.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

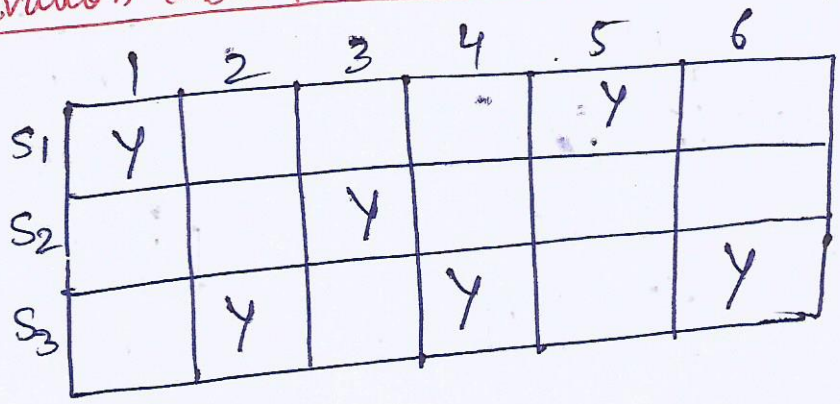
Question

Reservation Table for Function X:



Forbidden latencies
⇒ 2, 4, 5, 7

Reservation Table for function Y:



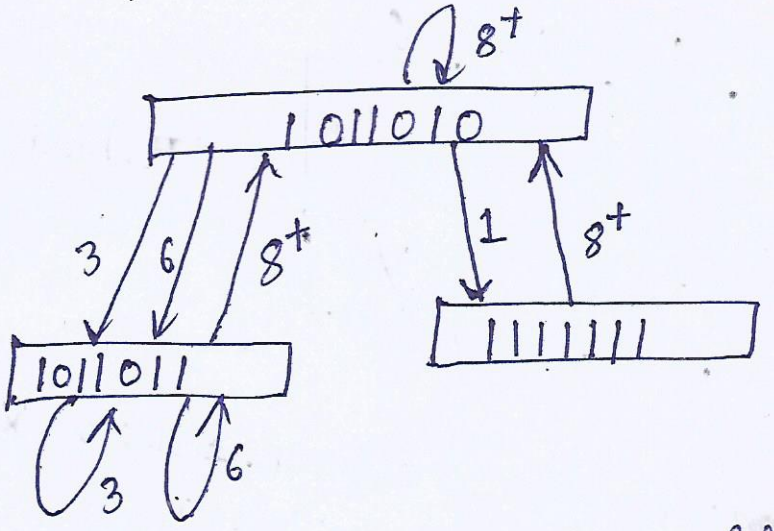
Forbidden latencies
⇒ 2, 4.

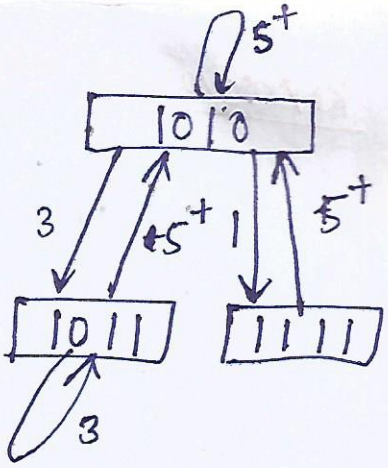
⇒ if reservation table has n columns & the maximum forbidden latency is $m \leq (n-1)$

⇒ permissible latency p will satisfy $1 \leq p \leq m-1$

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)





possible Cycles ⇒ (1,8)
(3)
(6)

MAL = 3.

Average latency
$\frac{1+8}{2} \Rightarrow 4.5$
⇒ 3
⇒ 6

(1,5) ⇒ 3.0
3 3.0
(3,5) ⇒ 4.0

MAL ⇒ 3.

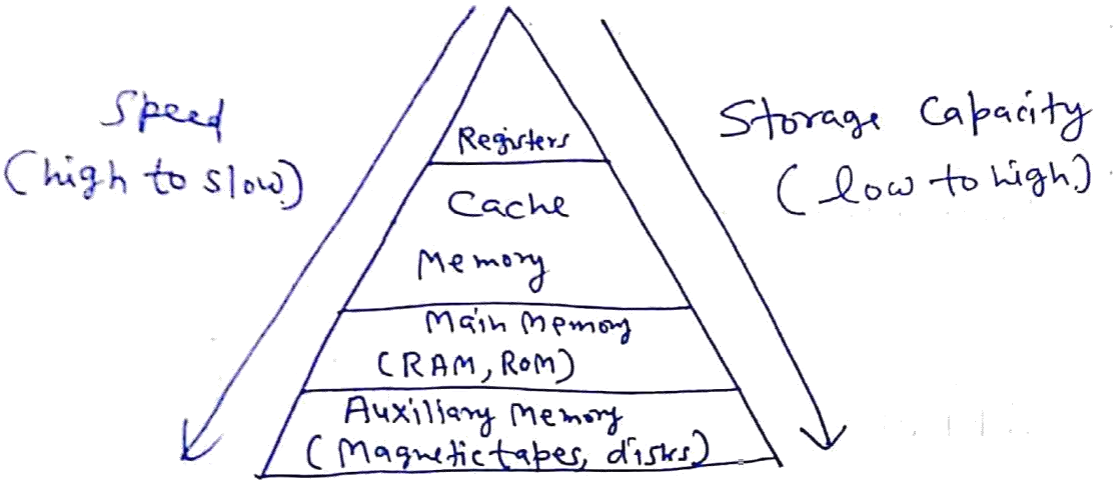
Downloaded by Sangam gupta (sangamg7318@gmail.com)

COA Unit 4 notes

Computer Organization & Architecture (Dr. A.P.J. Abdul Kalam Technical University)

Unit-4 (Part-1)

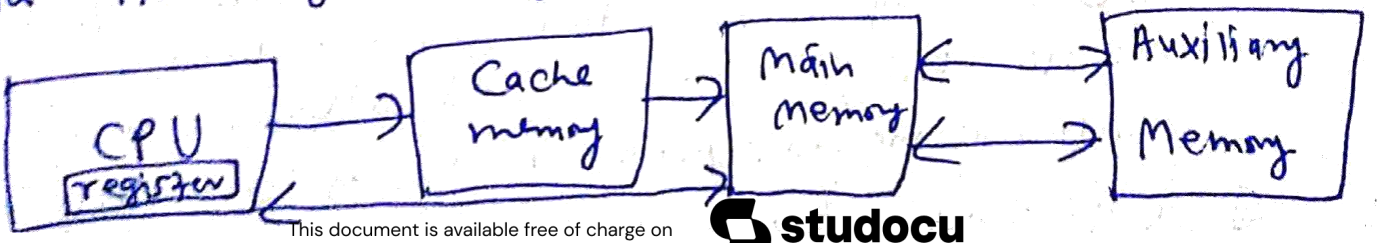
Memory Hierarchy: The memory hierarchy system consists of all storage devices employed in a Computer system from the slow (but high storage) auxiliary memory to very fast (but low storage) registers.



fig! 1 Memory hierarchy

1) **Registers!** are fastest memory with very low storage and most expensive.
2) **Cache Memory!** Slower than registers, but faster than main memory & has more storage than registers. Used to hold program & data that are heavily used.
3) **Main Memory:** storage more than Cache Memory but less than Auxiliary memory. Speed less than Cache memory but more than auxiliary memory.
4) **Auxiliary Memory:** It has maximum Storage Capacity but slowest among all devices. It is also cheapest among all devices.

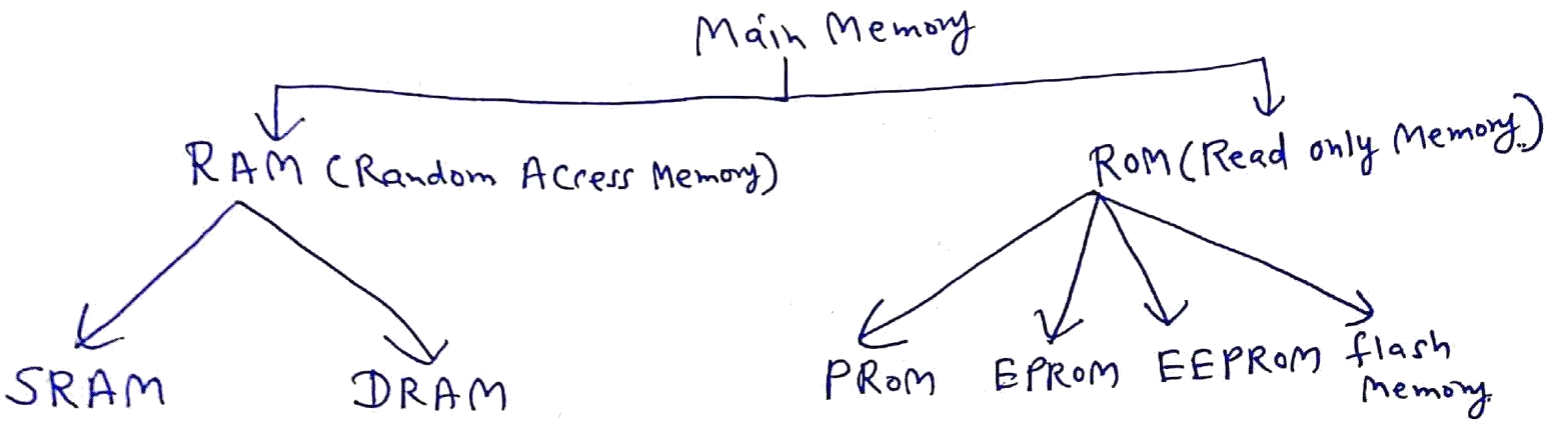
CPU can directly communicate with main memory and also via cache memory. CPU can not directly communicate with Auxiliary memory.



This document is available free of charge on studocu
Scanned by CamScanner

Main Memory

(1) It is the central storage unit in a computer system.
(2) It is used to store program and data during the computer operation.
(3) Based on semiconductor ~~IC (Integrated)~~ integrated circuits.
(4) Types of Main Memory:



RAM (Random Access Memory): (i) It is a volatile memory.
(ii) Read/write Access is random.
(iii) Both read and write operation can be performed.
(iv) RAM has two types:-

(a) **SRAM (Static RAM):** It makes use of flip-flops to store binary information. It is called static RAM because data once stored will remain same (does not change) until power is on. The memory density (memory cell/Area) is relatively less. So less access time is taken in read/write operation. So it is faster, due to this, it is used in Cache memory.

(b) **DRAM (Dynamic RAM):** It makes use of capacitors for storage. The presence of electric charge on capacitor indicates value (1). The stored charge on capacitors decreases with time. So value '1' may change to '0' if electrical charge finishes. Due to this, it is called dynamic RAM. The memory density (Memory cell per unit Area) is relatively is high. Due to this, it has larger access time but more

Scanned by CamScanner

storge capacity. Due to this, it is used mainly in main memory. (3)

(V) ROM (Read Only Memory): It is also a type of main memory. In this, data/program both are read only. No write operation can be performed in this (But versions of ROM now performs write operation also). ~~Read/write access~~ ROM is also random access. It is ~~also~~ non-volatile (No data lost, if power off) in nature. The types of ROMs are:-

(i) PROM (Programmable ROM): This ROM chip is one time programmable. Once programmed, it can not be erased.

(ii) EPROM (Erasable Programmable ROM): This ROM chip is multi time programmable. Once programmed, user can erase data using Ultra-violet rays to erase whole chip & then programme again. Any change require, whole chip to be erased.

(iii) EEPROM (Electrical Erasable PROM): This ROM chip is also multiple time programmable. In this chip, electrical signals are used to erase data at byte or block level rather than erasing whole chip & program again. faster than EPROM.

(iv) Flash Memory: It is almost similiar to EEPROM. The main difference from EEPROM is that in this, a bit level erasing & rewriting of data is possible which make it very fast. Due to this, it is called flash memory.

This document is available free of charge on studocu

Scanned by CamScanner

Numerical on RAM & ROM Chips

Q① How many 128×8 RAM chips are needed to provide a memory capacity of 2048 bytes.

Ans

Size of one RAM chip = 128×8

Total memory capacity = 2048 bytes

= 2048×8

No. of RAM chips = $\frac{\text{Total memory size}}{\text{one RAM chip size}}$

$= \frac{2048 \times 8}{128 \times 8} = \frac{2^{11}}{2^7}$

$= 2^{11-7} \Rightarrow 2^4 = 16$

Ans = 16 chips

② How many ROM chips of 1024×8 size are required for Total ROM of 4K bytes.

Ans

No. of ROM chips = $\frac{\text{Total ROM size}}{\text{one ROM chip size}}$

$= \frac{4K \text{ bytes}}{1024 \times 8}$

$= \frac{4K \times 8}{1024 \times 8}$

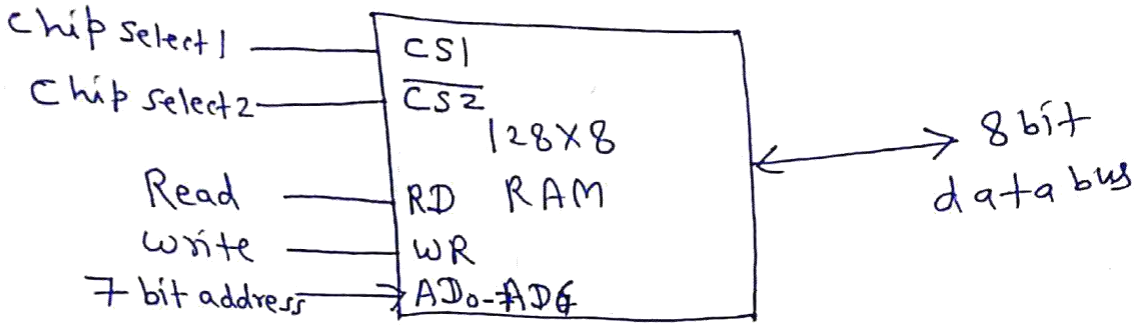
$= \frac{4 \times 1024 \times 8}{1024 \times 8}$

Ans = 4 ROM chips

Scanned by CamScanner

RAM chips & ROM chips configuration

RAM chip :



Typical RAM chip of size 128X8

$CS_1, \overline{CS_2}$ → Two chip select control inputs

RD → Read control input

WR → write control input

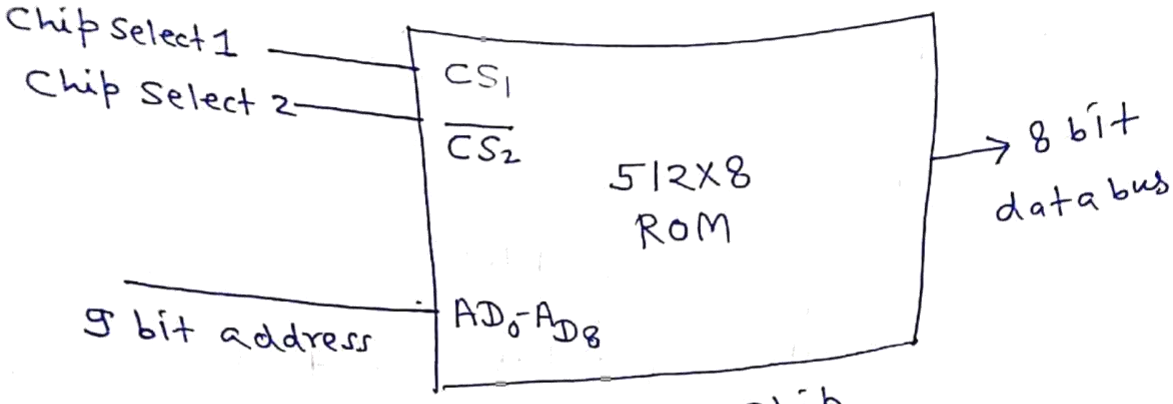
AD_0-~~AD6~~ → 7 bit address inputs
8 bit data bus

Function table for RAM chips

CS_1	$\overline{CS_2}$	RD	WR	Memory function	State of the data bus
0	0	X	X	Inhibit	High Impedance
0	1	X	X	Inhibit	High Impedance
1	0	0	0	Inhibit	High Impedance
1	0	0	1	write	Input data to RAM
1	0	1	X	Read	output data from RAM
1	1	X	X	Inhibit	~~High-impedance~~

This document is available free of charge on studocu

ROM chip:



ROM chip

Function table for ROM chip

CS_1	$\overline{CS_2}$	Memory function	State of data bus
0	0	Inhibit	High Impedance
0	1	Inhibit	High Impedance
1	0	Read	output data from ROM
1	1	Inhibit	High Impedance

Numerical Example-2

(1) How many lines of address bus must be used to access 2048 bytes of memory,

(2) How many chips of size 256X8 RAM are required.

(3) How many address lines are Common to all chips.

(4) How many lines must be decoded for chip select? Specify the size of decoder.

(5) Show how Connection are made with detailed Configuration.

Downloaded by Sangam gupta (sangamg7318@gmail.com) Scanned by CamScanner

(1) No. of address lines = 2048 bytes

$= 2048 \times 8$

$= 2^{11} \times 8$

No. of address lines $= 11$

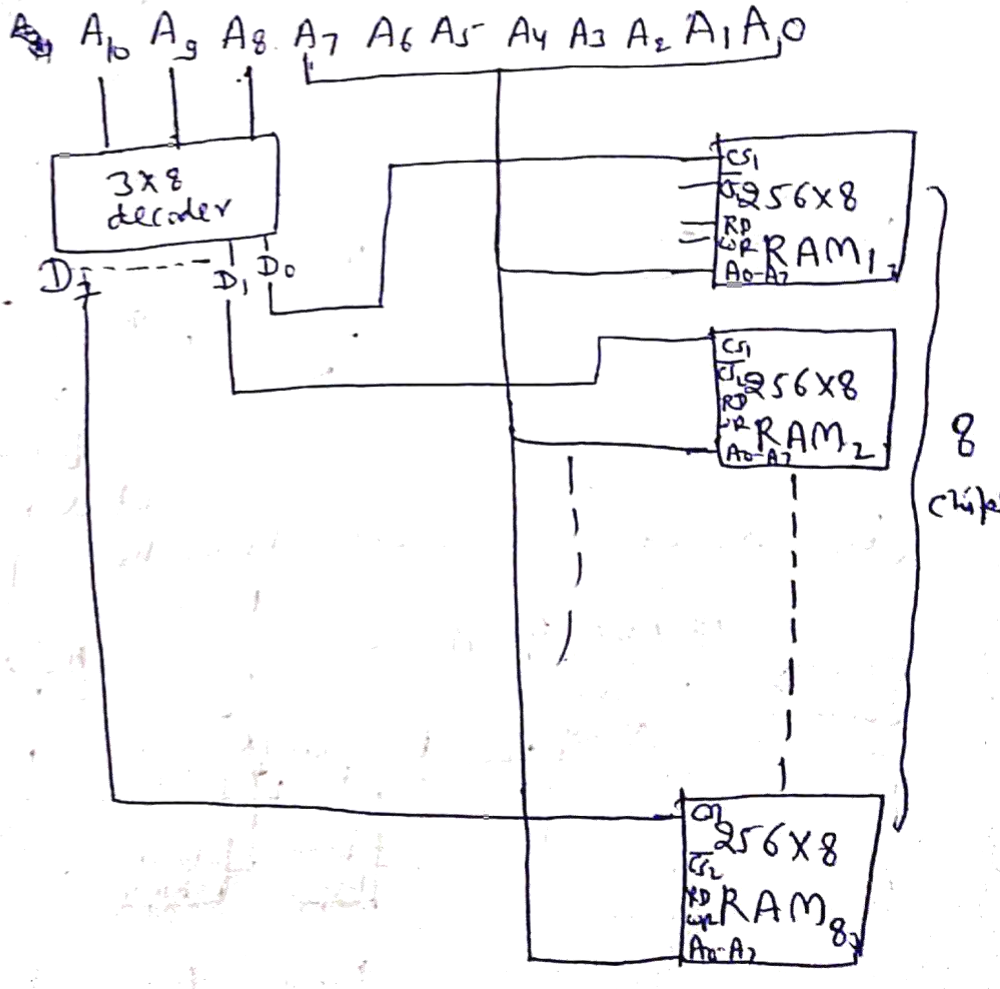
No. of data lines $= 8$

(2) No. of chips $=$ Total memory / one RAM size

$= \frac{2048 \times 8}{256 \times 8} = \frac{2^{11}}{2^8} = 2^3 = 8$

$= 8$ chips

(3)



This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)
Scanned by CamScanner

(4) 3 address line (A_8, A_9, A_{10}) must be dec for chip select.

Decoder size = 3 X 8 decoder

(5) Detailed Connection are already shown on Ans (3).

12.2 A Computer uses RAM chips of 1024X1 Capacity

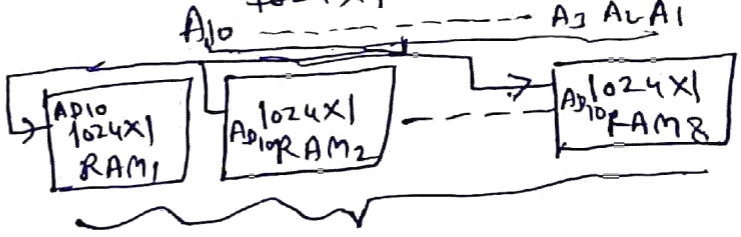
(a) How many chips are needed, and how should their address lines be Connected to provide a memory capacity of 1024 byte.

(b) How many chips are needed to provide a memory Capacity of 16 K byte. How the chips are to be connected to the address bus.

(a) No. of chips needed = $\frac{\text{Total memory}}{\text{one RAM size}} = \frac{1024 \text{ bytes}}{1024 \times 1}$

$= \frac{1024 \times 8}{1024 \times 1} = 8$ chips

Memory Connection diagram

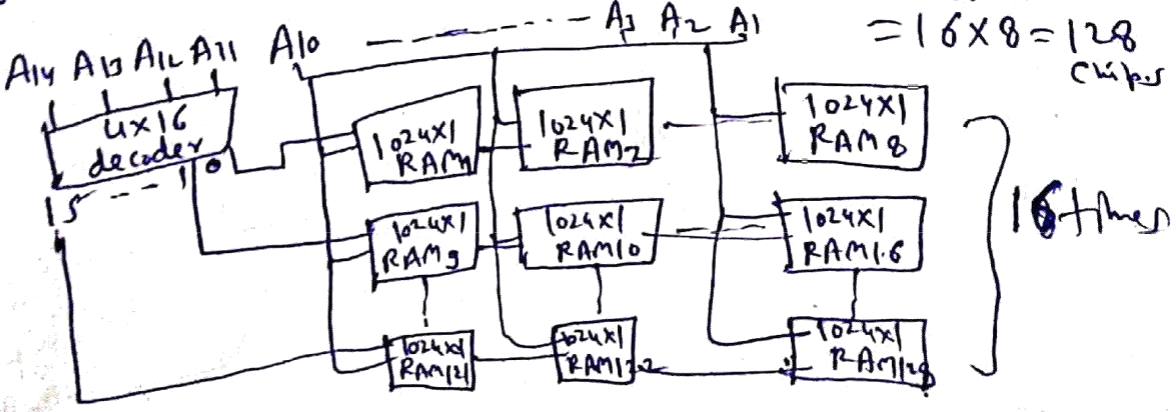


8 chips

Total memory = 1024 X 8

(b) No. of chips needed = $\frac{16 \text{ K bytes}}{1024 \times 1} = \frac{16 \text{K} \times 8}{1024 \times 1} = \frac{16\text{K} \times 8}{1\text{K} \times 1} = 16 \times 8 = 128$ chips

Memory Connection



Downloaded by Sangam gupta (sangamg7318@gmail.com) Scanned by CamScanner

Memory Address Map:

* Memory address map is the pictorial representation of assigned address space for each chip in the System.
* Memory Address Map consists of three Columns
  (a) Components
  (b) Hexadecimal address
  (c) Address Bits (Address Bus)
* Components specifies whether a RAM, ROM or I/O device
* Hexadecimal Address column specify the assigned range of hexadecimal equivalent address for each chip.
* Address Bits (Address Bus) Column specify the no. of bits required in the address bus with value of each address bits. The hexa decimal address of each chip is obtained from the Information under the address bus assignment.

Components	Hexadecimal Address	Address Bits (Address Bus) 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
RAM chip (1-4)	0000H-01FFH	0 0 0 0 0 0 0 0 ← XXXX XXXX (↓ 2x4 RAM decoder)
ROM chip (1)	0200H-03FFH	0 0 0 0 0 0 1 X XXXX XXXX (↓ ROM)

Sample Example of Memory address map for 4 RAM Chips and 1 ROM chips
(128x8 RAM) (512x8 ROM)

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

* In the RAM chip, for RAM size 128 X 8 RAM

↓

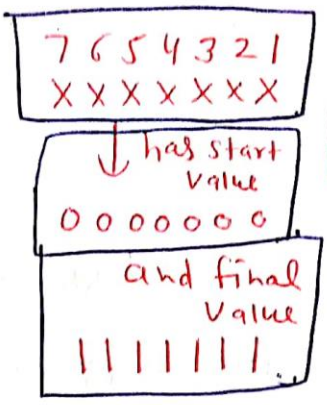
2^{7} X 8 RAM

↓

7 No of address bits represented as X

X → may be 0 or 1.

7 address bits → Bits (1-7)



Bit (8-9) used to represent 4 RAM chips as system has 4 RAM chips.

→ used 2 X 4 decoder for assigning unique value to RAM chips

→

9	8	(Address bits)
0	0	RAM 1
0	1	RAM 2
1	0	RAM 3
1	1	RAM 4

Bit (10) → used to differentiate between RAM and ROM.

0 → RAM

1 → ROM

Bit (11-16) are set to zero (0) as they are not used in this problem.

---

for ROM chip of size 512 X 8 ROM

↓

2^{9} X 8 ROM

↓

9 address bits are represented as X

X may be 0 or 1

9 address bits (1-9)

9 8 7 6 5 4 3 2 1

X X X X X X X X X

starting address 0 0 0 0 0 0 0 0 0

Final " 1 1 1 1 1 1 1 1 1

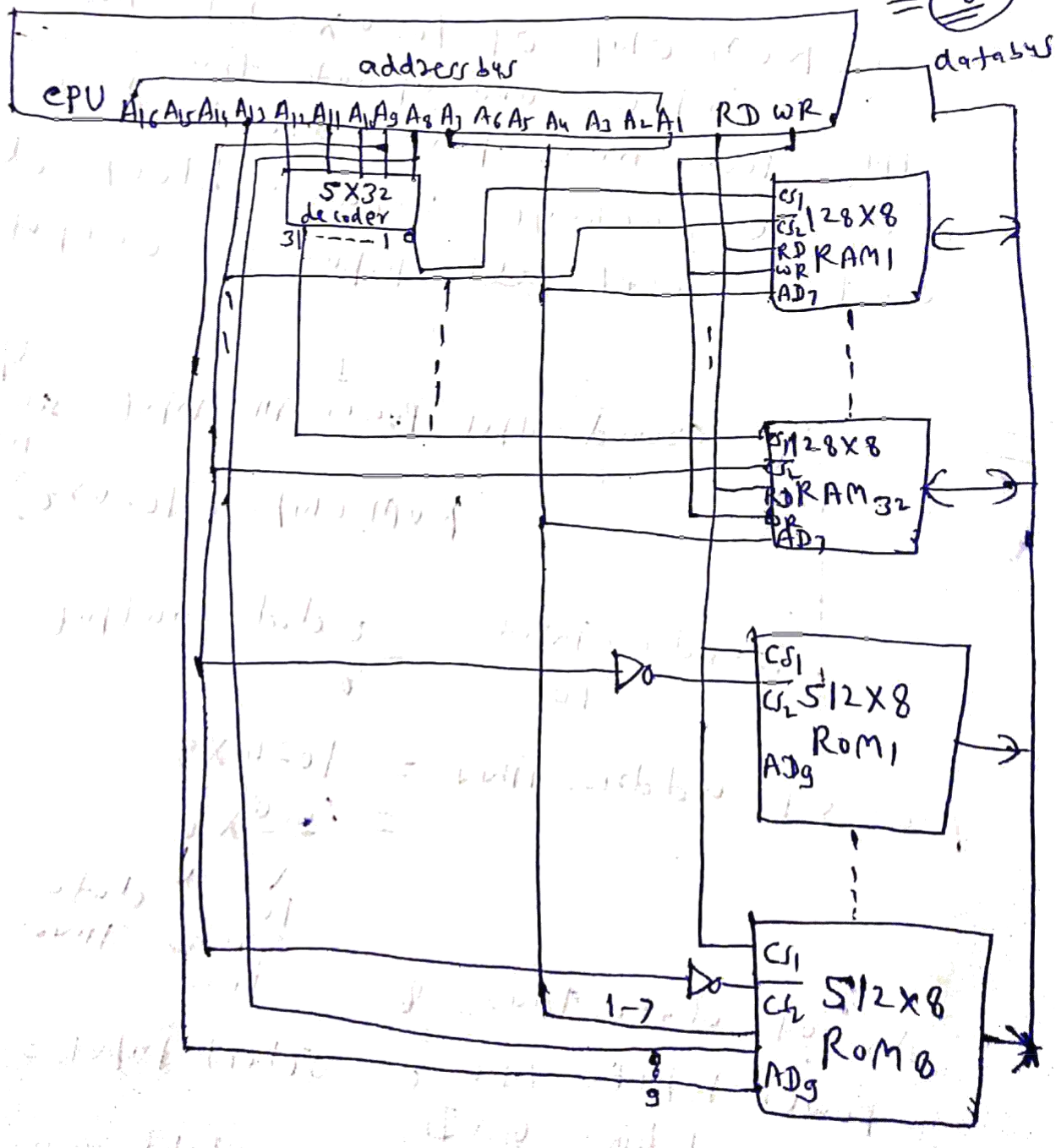
Downloaded by Sangam gupta (sangamg7318@gmail.com)

12.4 Consider a system with RAM size 128X8 and ROM of size 512X8. How many RAM, ROM chips are required for ~~Draw memory address map for No. of a total memory of~~ a total RAM of 4096 bytes and 4096 bytes of ROM. Draw Memory address Map.

Ans (i) No. of RAM chips = $\frac{4096 \text{ bytes}}{128 \times 8} = \frac{4096 \times 8}{128 \times 8} = \frac{2^{12}}{2^7} = 2^5 = 32$

No. of ROM chips = $\frac{4096 \text{ bytes}}{512 \times 8} = \frac{4096 \times 8}{512 \times 8} = \frac{2^{12}}{2^9} = 2^3 = 8$

(ii)



This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)
Scanned by CamScanner

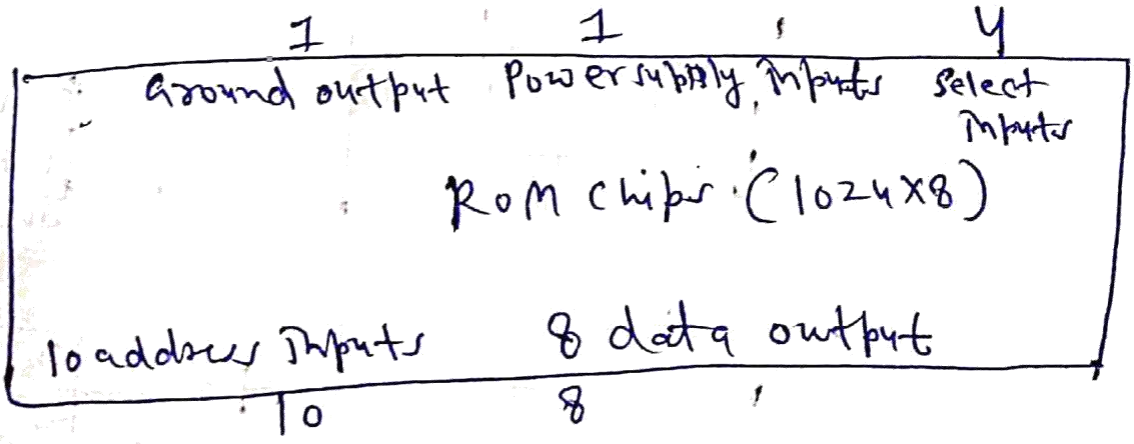
Memory Address Map

(10)

Components	Memory Addresses	$A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}A_9A_8A_7A_6A_5A_4A_3A_2A_1$
RAM	0000-0FFFH	0000 ← 5X32 decoder → XXXX XXX
ROM	1000H-1FFFH	0001 ← 3X8 decoder → XXX XXX XXX

12.3 A ROM chip of 1024 X 8 bits has four select inputs and operates from a 5V power supply. How many pins are needed for the IC package. Draw a block diagram and label all inputs & output terminals

Ans



No. of address lines. = 1024 X 8

= 2^{10} X 8

10 address lines, data lines

No. of data lines = 8

Power input = +5V, Select inputs = 4

Ground pin = GND,

Total pins = 10+8+1+1+4 = 24 pins

Downloaded by Sangam gupta (sangamg7318@gmail.com) Scanned by CamScanner

...em with RAM size 128 x ... (11)

12.5 A Computer employs RAM chips of 256X8 and ROM chips of 1024 X8. The Computer system needs 2K bytes of RAM, 4K bytes of ROM and 4 interface units, each with four registers. A memory mapped I/O Configuration is used. The two highest order bits of the address bus are assigned 00 for RAM, 01 for ROM and 10 for Interface registers.

(a) How many RAM & ROM chips are needed?

(b) Draw a memory address map for the system.

(C) Give the address range in hexadecimal for RAM, ROM and Interface.

Solution Available on Youtube video "Memory address map with example" on "LS Academy for Technical Education"

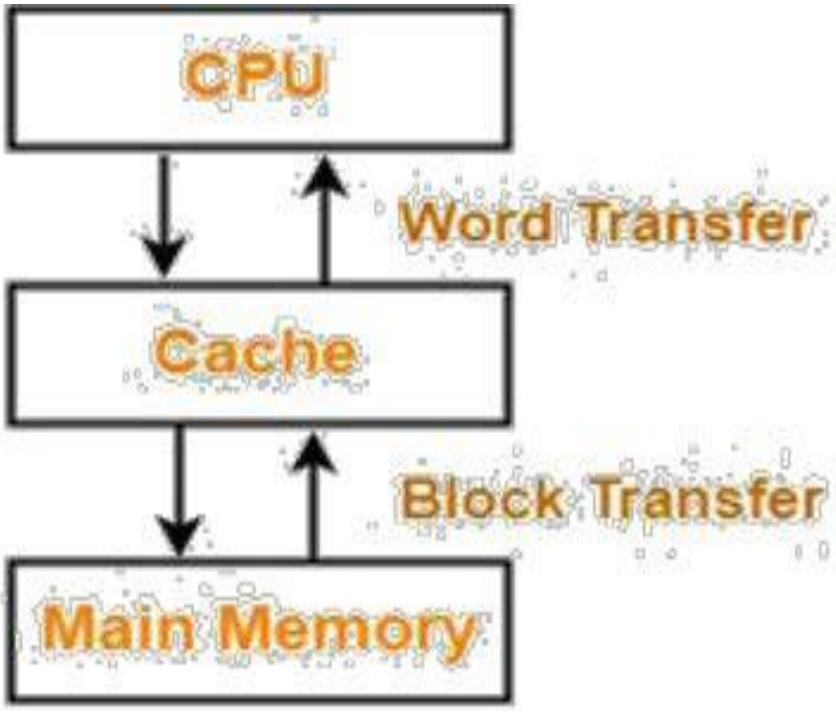
This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com) Scanned by CamScanner

Cache Memory-

- Cache memory is a Static Random-Access Memory.
- The main advantage of cache memory is its very fast speed.
- It can be accessed by the CPU at much faster speed than main memory.
- Cache memory bridges the speed mismatch between the processor and the main memory.

- Cache memory lies on the path between the CPU and the main memory.
- It facilitates the transfer of data between the processor and the main memory at the speed which matches to the speed of the processor.



Cache and Main Memory

Downloaded by Sangam gupta (sangamg7318@gmail.com)

- Data is transferred in the form of words between the cache memory and the CPU.
- Data is transferred in the form of blocks or pages between the cache memory and the main memory.

Purpose-

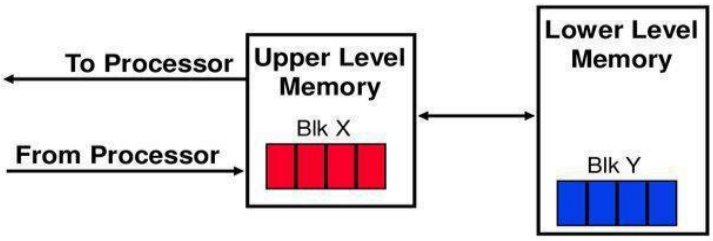
The fast speed of the cache memory makes it extremely useful.

- It is used for bridging the speed mismatch between the fastest CPU and the main memory.
- It does not let the CPU performance suffer due to the slower speed of the main memory.

Principle of Locality (Temporal)

- Programs access a small proportion of their address space at any time (Locality in Time)
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - Keep most recently accessed data items closer to the processor
  - e.g., instructions in a loop, induction variables

```
for (i=0; i<1000; i++)
  x[i] = x[i] + s;
```



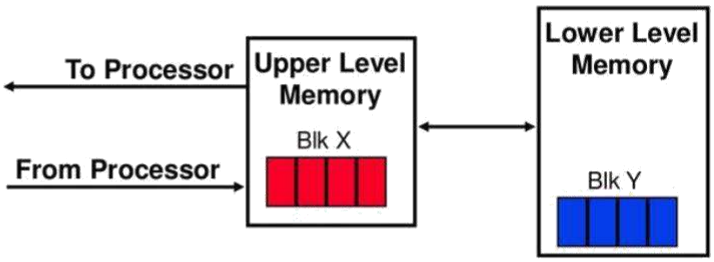
This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Principle of Locality (Spatial)

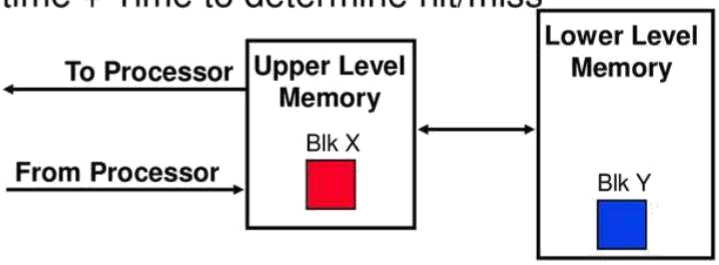
- Programs access a small proportion of their address space at any time (Locality in Space)
- Spatial locality
  - Items near those accessed recently likely to be accessed soon
  - Move blocks consisting of contiguous words to the upper levels
  - E.g., sequential instruction access, scanning an array data

```
for (i=0; i<1000; i++)
  x[i] = x[i] + s;
```



Terminology

- Hit: data is in some block in the upper level (Blk X)
  - Hit Rate: fraction of memory accesses found in upper level
  - Hit Time: Time to access the upper level which consists of
    - SRAM access time + Time to determine hit/miss



- Miss: data is not in the upper level so needs to be retrieve from a block in the lower level (Blk Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Time to bring in a block from the lower level and replace a block in the upper level with it
    + Time to deliver the block the processor
  - Hit Time << Miss Penalty

Cache Mapping-

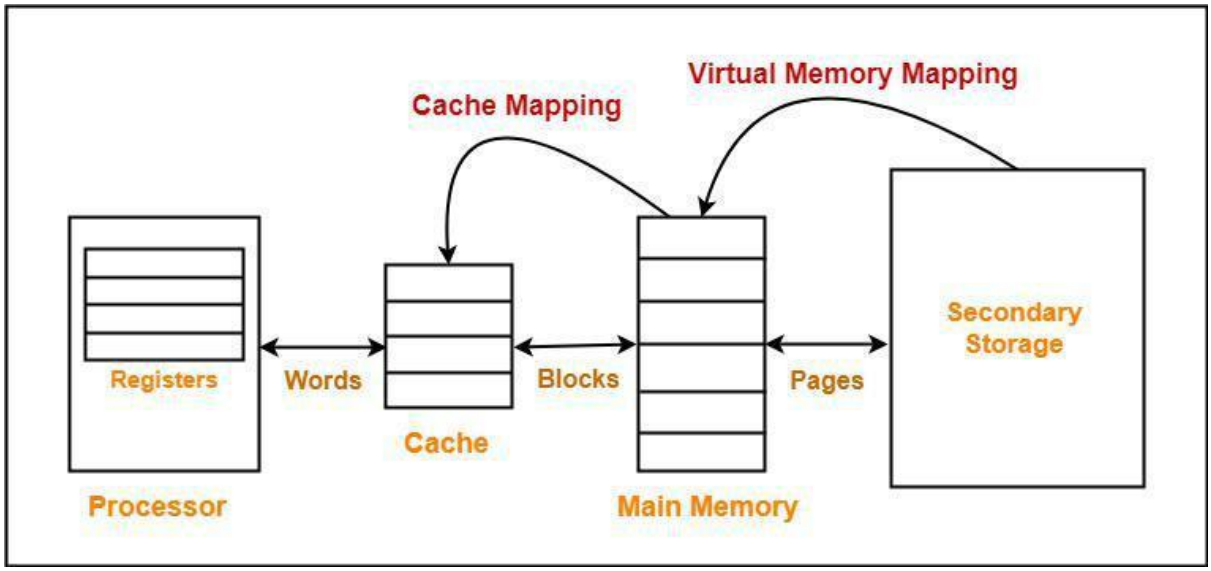
Downloaded by Sangam gupta (sangamg7318@gmail.com)

- Cache mapping defines how a block from the main memory is mapped to the cache memory in case of a cache miss.

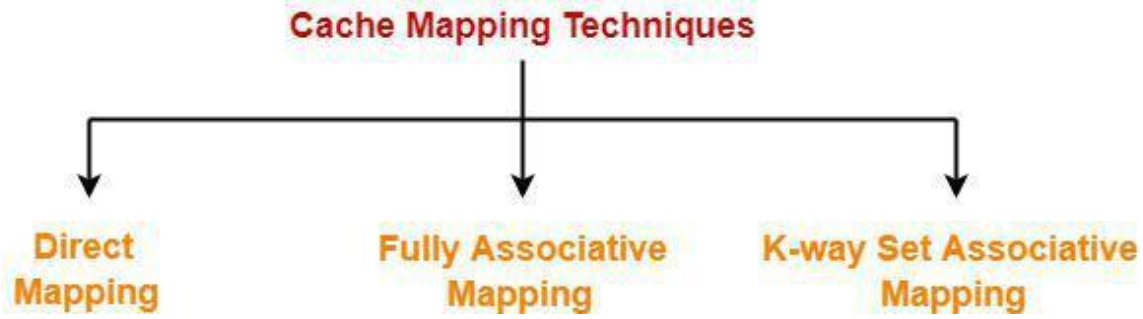
OR

- Cache mapping is a technique by which the contents of main memory are brought into the cache memory.

The following diagram illustrates the mapping process-



- Main memory is divided into equal size partitions called as **blocks** or **frames**.
- Cache memory is divided into partitions having same size as that of blocks called as **lines**.
  - During cache mapping, block of main memory is simply copied to the cache and the block is not actually brought from the main memory.



This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Direct Mapping

Block j of main memory maps onto block j modulo 128 of the cache

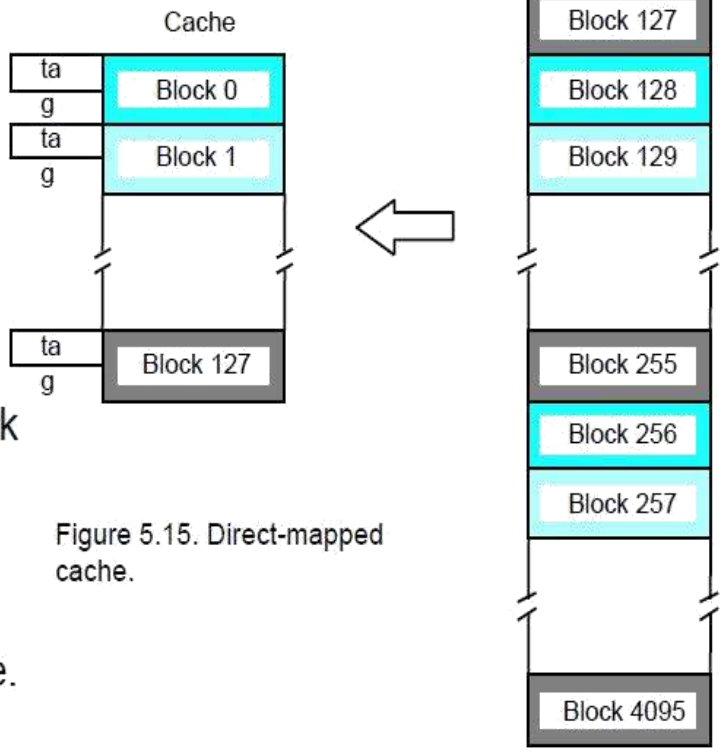
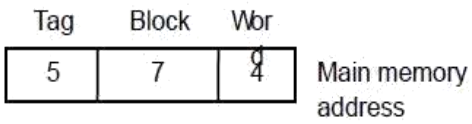


Figure 5.15. Direct-mapped cache.

4: one of 16 words. (each block has $16=2^4$ words)

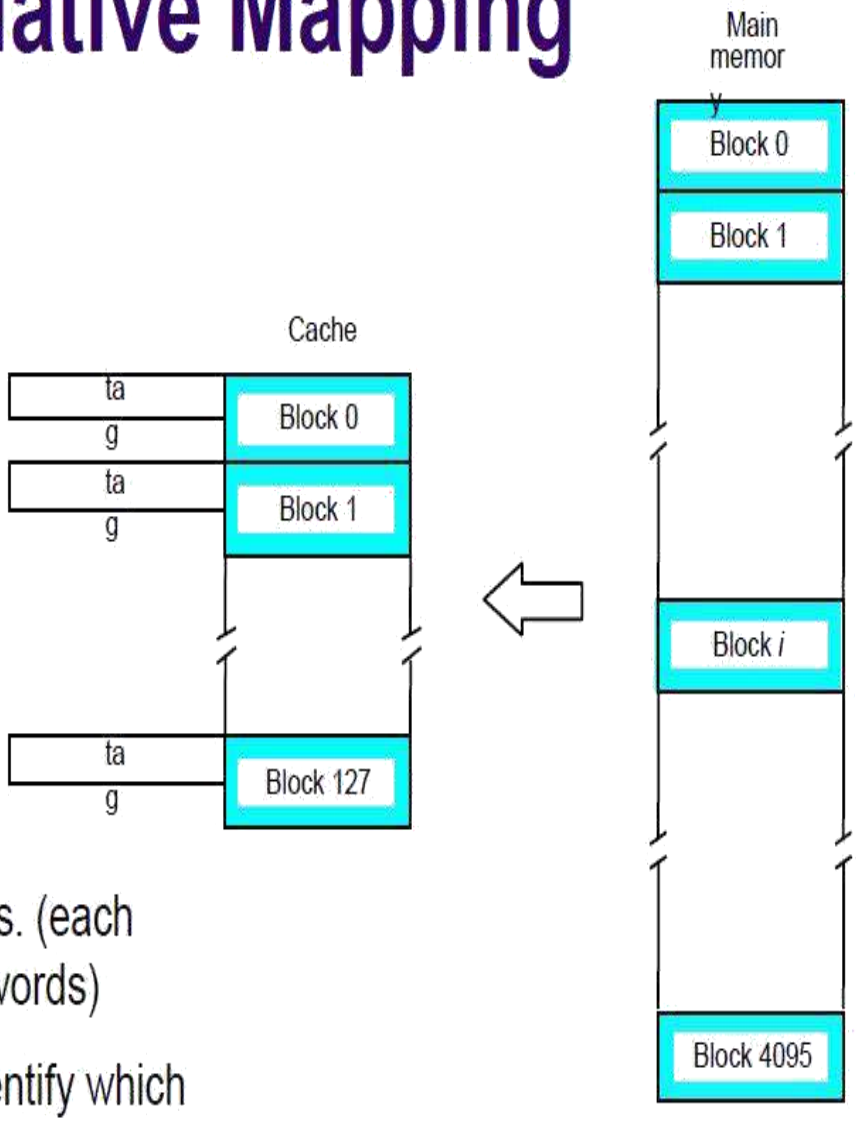
7: points to a particular block in the cache ($128=2^7$)

5: 5 tag bits are compared with the tag bits associated with its location in the cache. Identify which of the 32 blocks that are resident in the cache (4096/128).



Downloaded by Sangam gupta (sangamg7318@gmail.com)

Associative Mapping



4: one of 16 words. (each block has $16=2^4$ words)

12: 12 tag bits Identify which of the 4096 blocks that are resident in the cache $4096=2^{12}$.

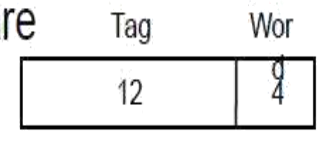
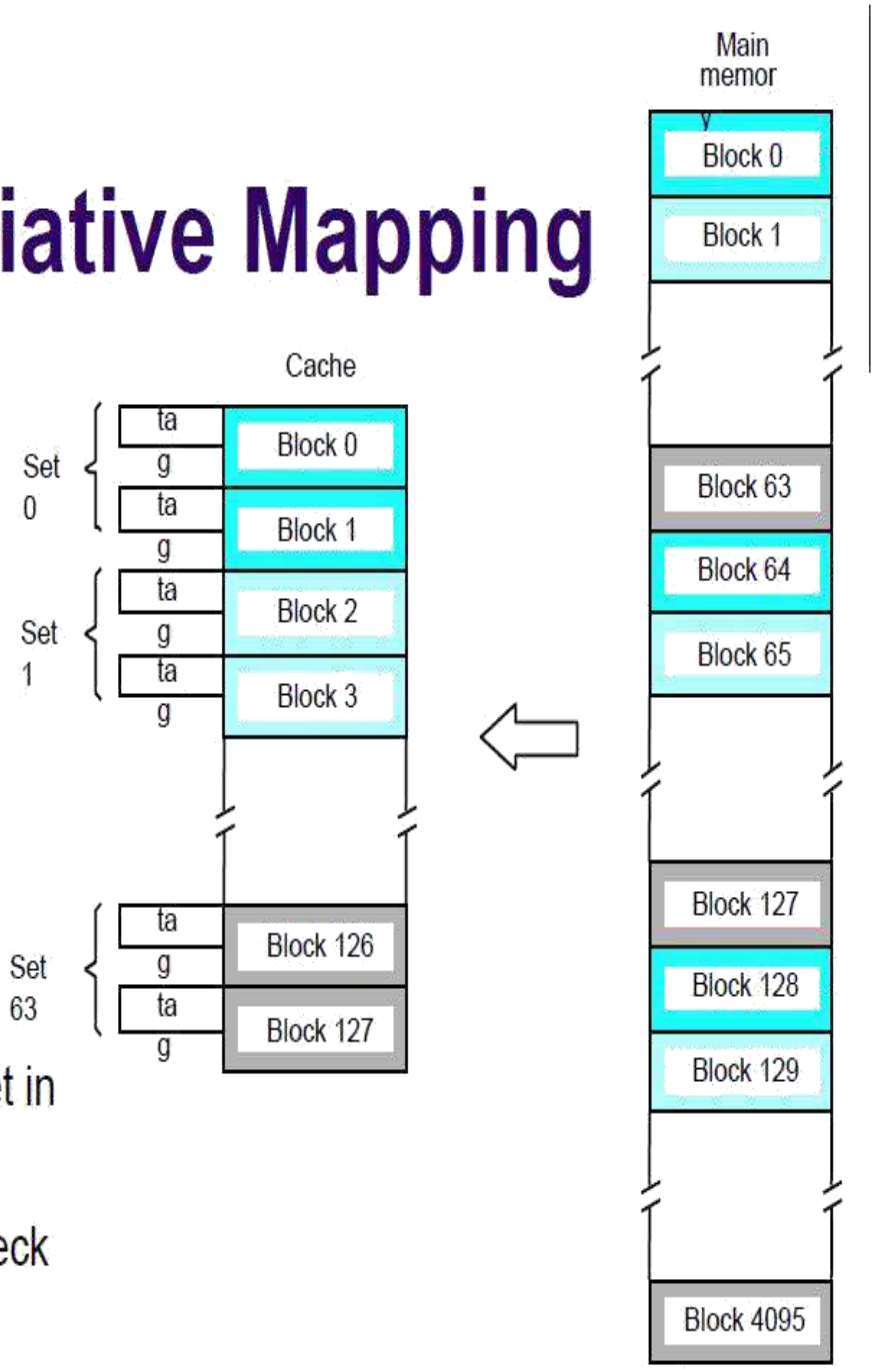


Figure 5.16. Associative-mapped cache.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Set-Associative Mapping

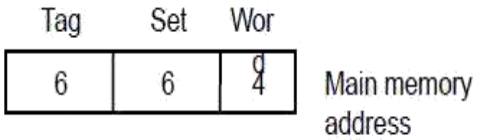


4: one of 16 words. (each block has $16=2^4$ words)

6: points to a particular set in the cache ($128/2=64=2^6$)

6: 6 tag bits is used to check if the desired block is present ($4096/64=2^6$).

Figure 5.17. Set-associative-mapped cache with two blocks per



Downloaded by Sangam gupta (sangamg7318@gmail.com)

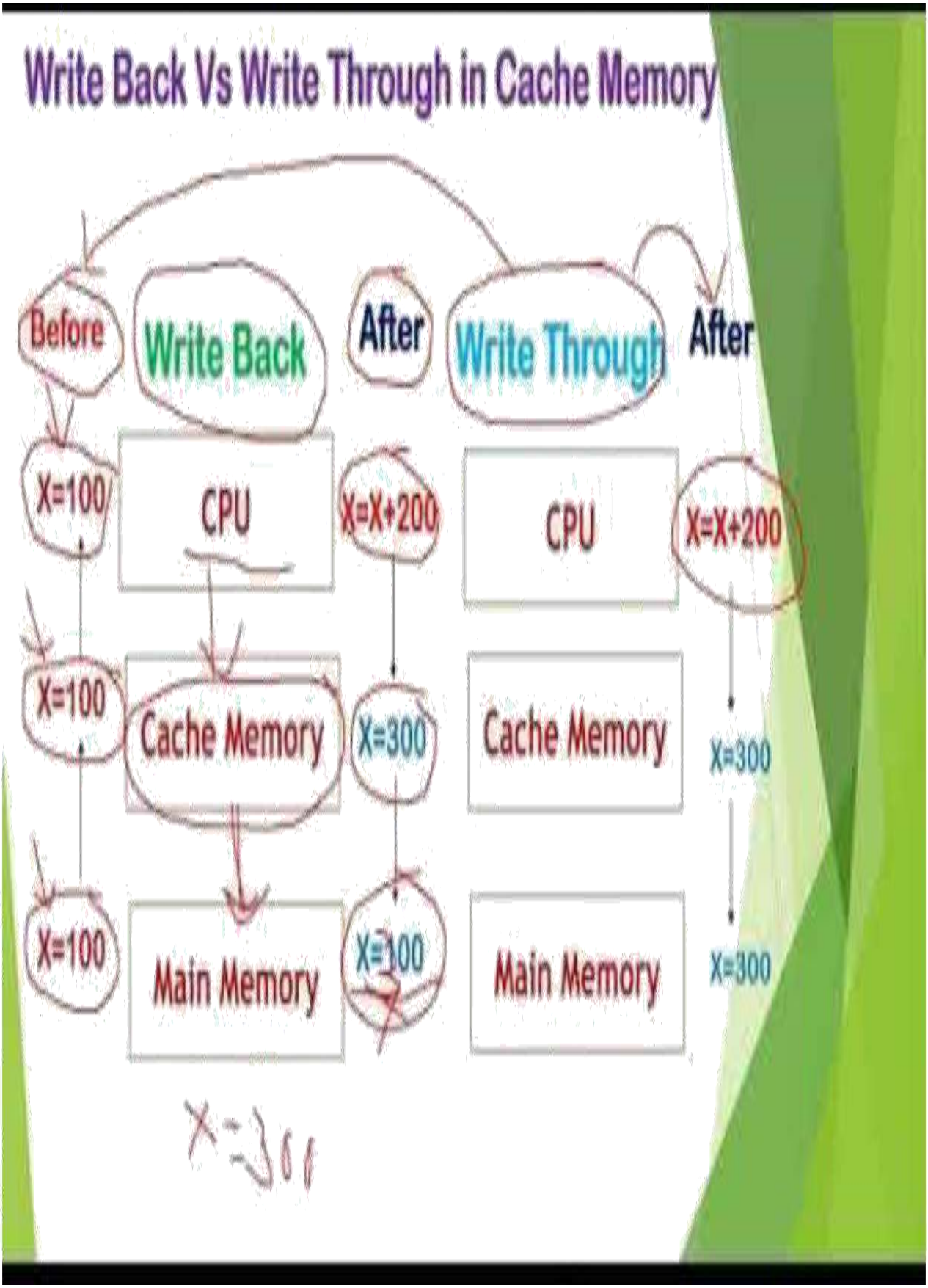
Write back Vs Write through method in Cache memory

Write Through method	Write Back method
① In this method, Cache memory as well as main memory is updated at the same time.	① In this method, Cache memory is only updated. The main memory will be updated later on.
② Implementation is easy	② Implementation is difficult.
③ Simpler logic circuit	③ Complex logic circuit.
④ Redundancy of data	④ No redundancy of data
⑤ High Reliability	⑤ Low Reliability
⑥ Data Consistency	⑥ Data Inconsistency
⑦ More memory bandwidth used	⑦ Less memory bandwidth is used.
⑧ More power is required	⑧ Less power is required

Example on next Page

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)



Downloaded by Sangam gupta (sangamg7318@gmail.com)

Associative Memory

* A memory unit accessed by content is called "Associative memory" or "Content Addressable Memory".
* This type of memory is accessed simultaneously and in parallel on the basis of data content rather than by specific memory address or location.
* When a word is written in an associative memory, no address is given. The memory is capable of finding an empty unused location to store the word.
* When a word is to be read from an associative memory, then the content of the word or part of the word is specified.
* The memory locates all words that match the specified content and marks them for reading.
* It takes relatively less time to find an item based on content rather than by an address.

Block diagram of Associative Memory:

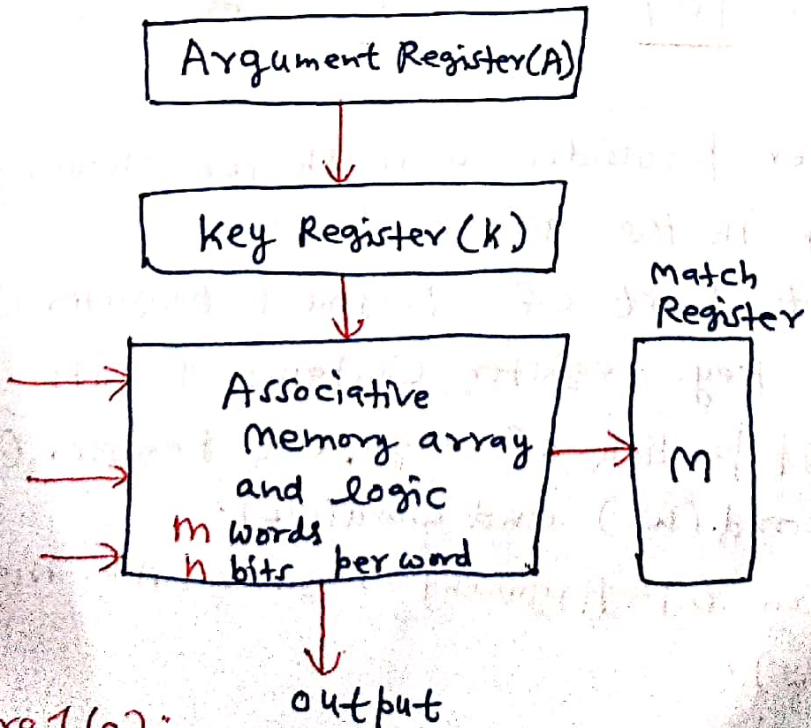
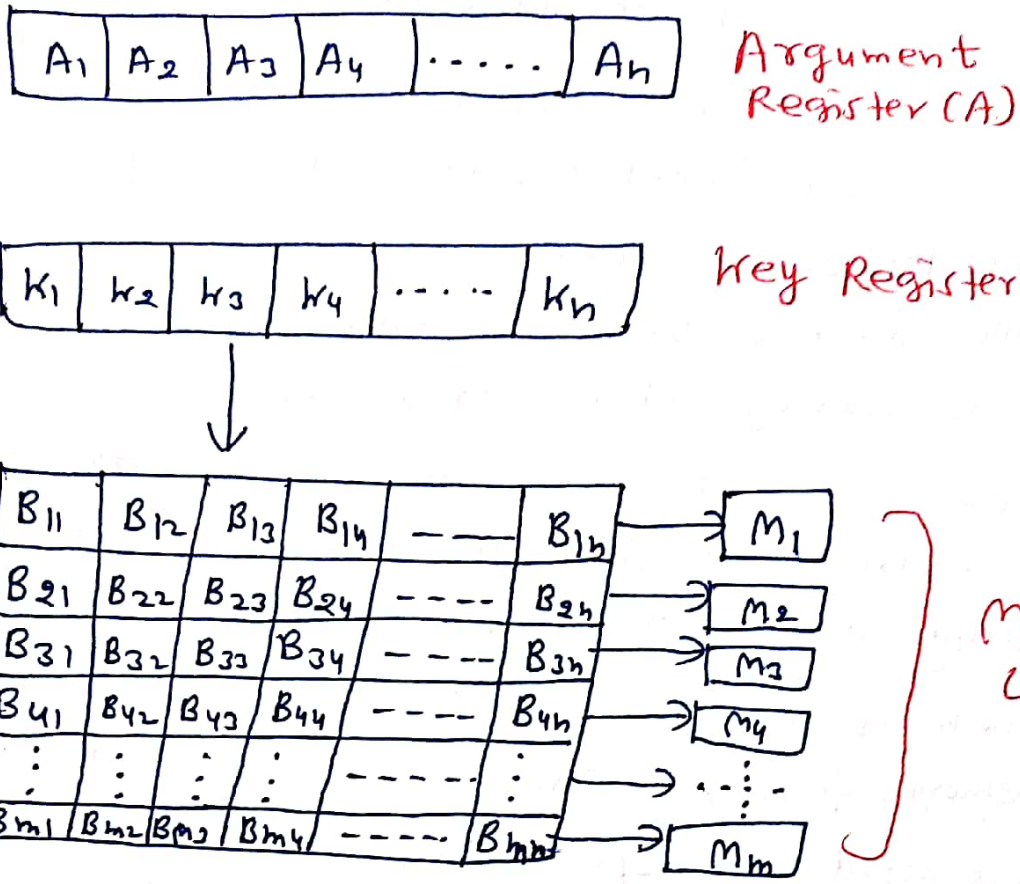


Figure 1(a):

This document is available free of charge on studocu
Scanned with CamScanner
Downloaded by Sangam gupta (sangamg7318@gmail.com)



Fig·1(b) Detailed Block diagram of Associative Memory.

A = 101 111100

K = 111 000000

w_1 = 100 111100 (No Match)

w_2 = 101 000001 (Match)

* Key Register provides a mask for choosing a particular field or key in the argument word.
* Only that part of Augment Register (A) is compared for which key register Contains 1 bits.
* Only 101 portion of argument Register (A) is compared with word (w_1) and word (w_2).
* 101 portion of Argument register (A) matches with word (w_2).

Scanned with CamS
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Cell Structure of Associative Memory with match logic (3)

* Each cell (memory cell) consists of a flip-flop storage element (F_{ij}) and circuit for reading, writing and matching the cell.
* The input bit is transferred into the storage cell during a write operation.
* The bit stored is read out during a read operation.
* The match logic compares the content of the storage cell with the corresponding unmasked bit of the argument and decides whether the bits matched or not.

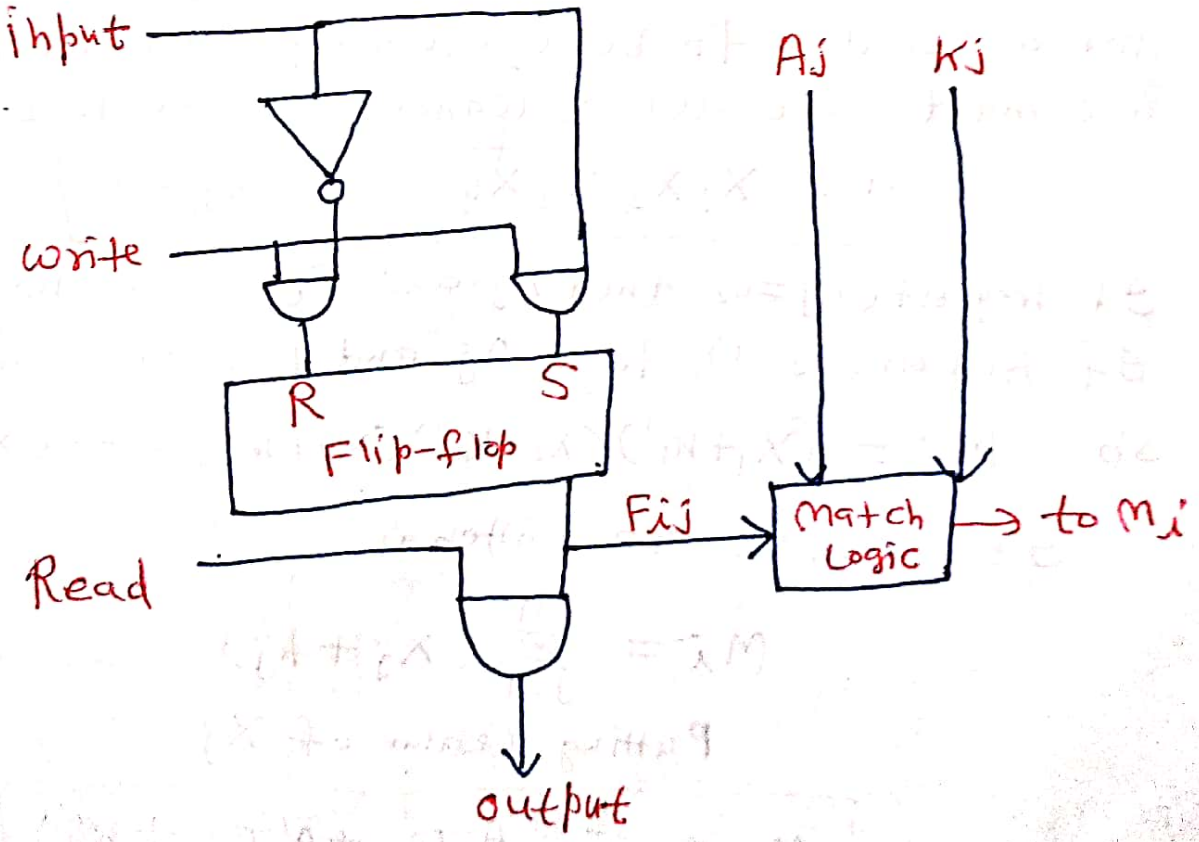


Figure 2(a): Cell Structure of Associative Memory

This document is available free of charge on studocu
Scanned with CamScanner
Downloaded by Sangam gupta (sangamg7318@gmail.com)

* Match Logic:

* The match logic for each word can be derived from the boolean function

$$X_j = A_j F_{ij} + A_j' F_{ij}'$$

(i) If the bits are equal, $X_j = 1$

(ii) If the bits are not equal, $X_j = 0$

* Each memory word (W_i) is a combination of individual memory cells ($F_{i1}, F_{i2}, F_{i3} \dots F_{in}$)

$$W_i = F_{i1}\, F_{i2}\, F_{i3} \dots F_{ij} \dots F_{in}$$

* for a word i to be equal to the argument in A, we must have all X variables equal to 1.

$$M = X_1 X_2 X_3 X_4 \dots X_n = 1$$

* If key bit ($K_j = 0$), then A_j and F_{ij} needs no comparison.
* If key bit ($K_j = 1$), then A_j and F_{ij} are compared.
* So $M_i = (X_1 + K_1')(X_2 + K_2')(X_3 + K_3') \dots (X_n + K_n')$

It can be re-written as

$$M_i = \sum_{j=1}^{n} (X_j + K_j')$$

Putting value of X_j

$$M_i = \sum_{j=1}^{n} (A_j F_{ij} + A_j' F_{ij}' + K_j')$$

So the logic circuit for matching one word is as given:

Scanned with CamS

Downloaded by Sangam gupta (sangamg7318@gmail.com)

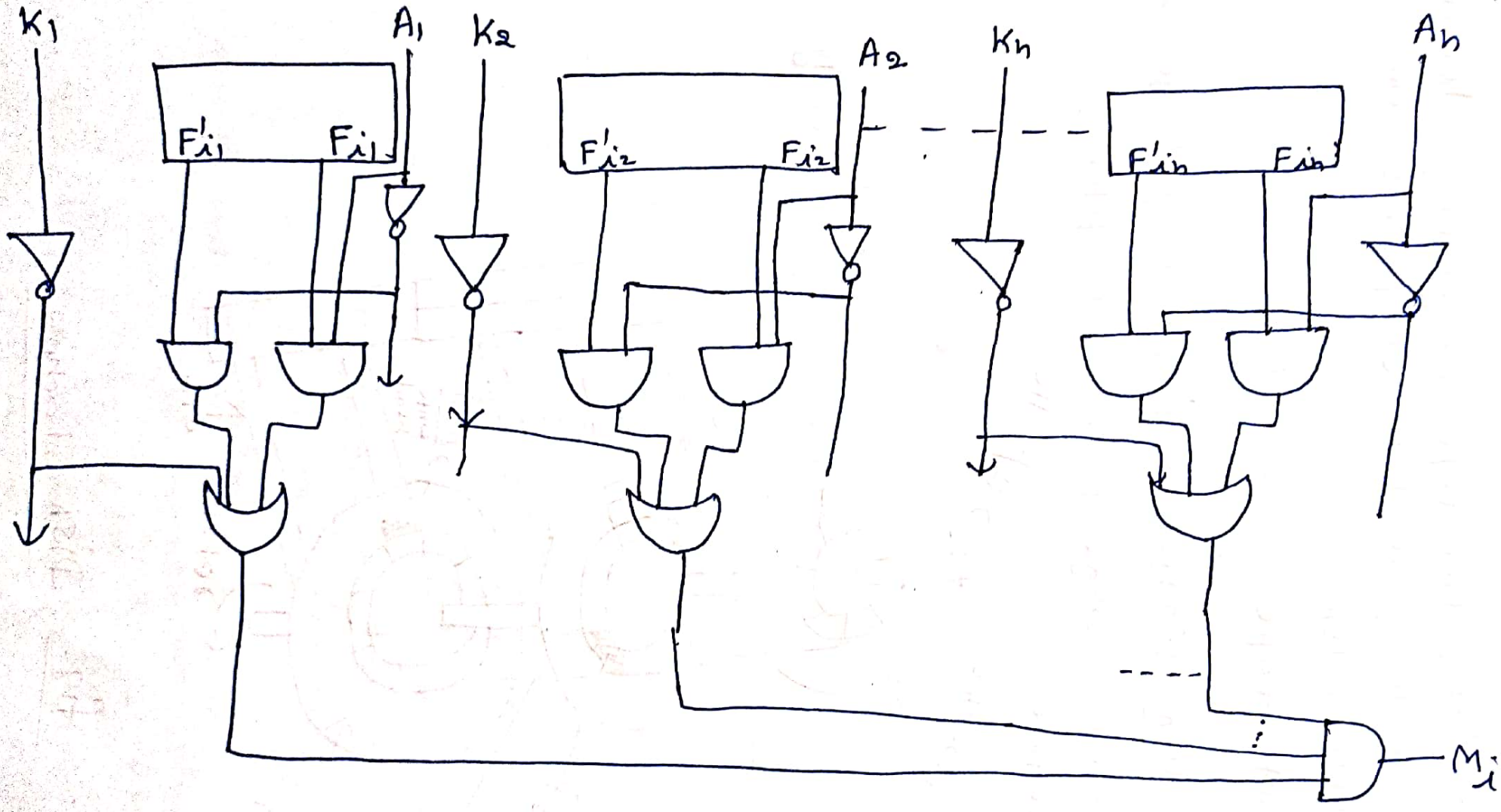


Figure: Match logic for one word of associative memory

This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Auxiliary Memory (Secondary Storage) ⑥

There are mainly two types of auxiliary devices that are commonly used in computer system:

1. Magnetic disk
2. Magnetic tape

---

① **Magnetic disk:** Magnetic disk provides a large amount of secondary storage to modern computers.

Disk structure

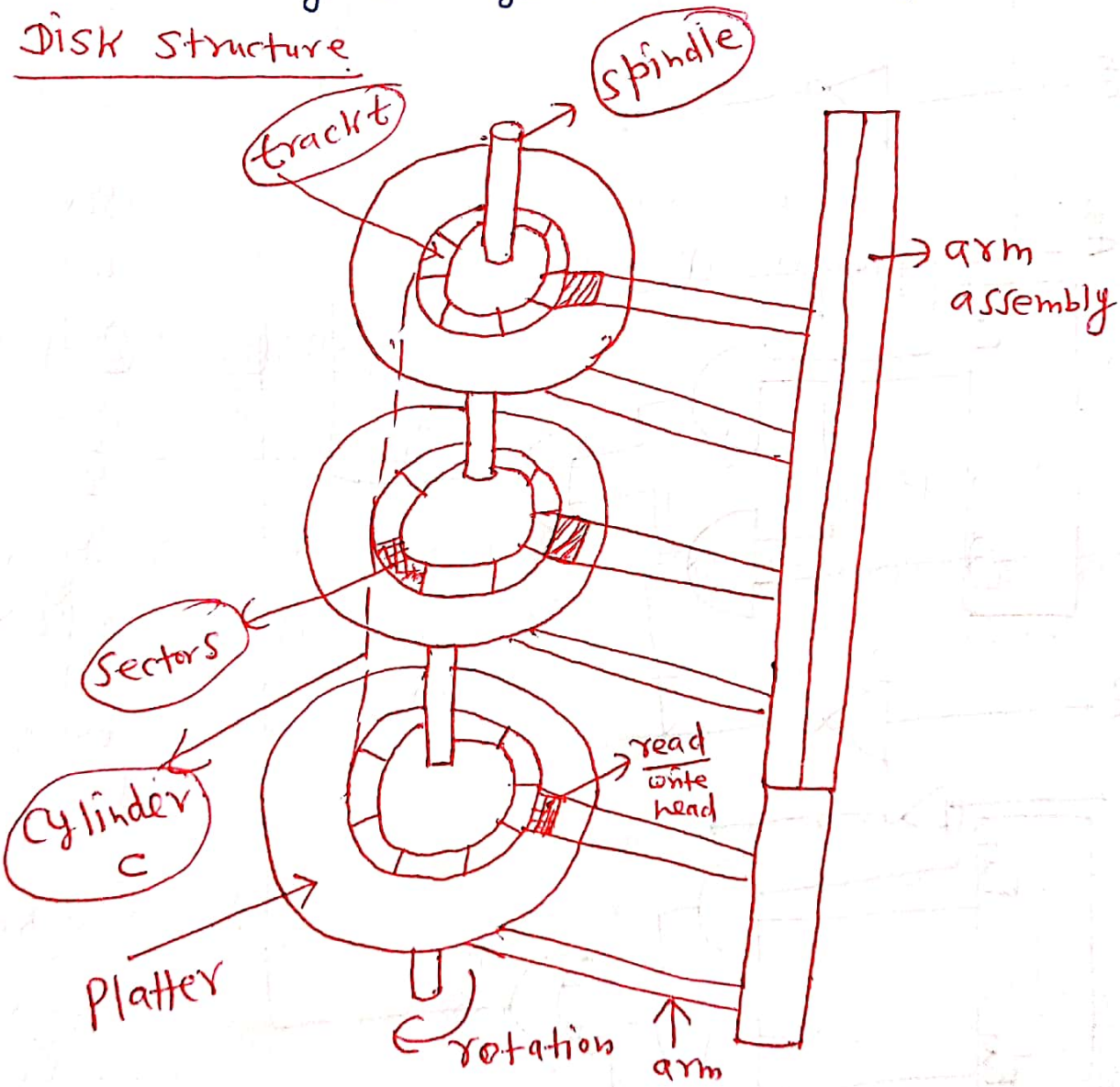


fig: Disk Structure

Scanned with CamS
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Platter: A round surface (disk) containing magnetic coating

Track: A circle on the disk surface containing data.

Cylinder: set of tracks accessed simultaneously from read/write head.

Sector: A portion of track on the disk surface. Each sector stores same no. of bytes. So the density of bits is more in sectors near the centre of disk.

Block: The smallest unit of data that can be written or read to/from the disk.

Read/write head: A transducer attached to an arm for reading/writing data to/from the disk surface

Arm Assembly: A mechanical unit holding all heads and arms.

Drive Motor: The Drive (DC) motor rotates the platters at a fixed rate (e.g. 3600 rpm.)

Head Motion: A mechanism is required to move the head assembly in/out.

Seek time: Time taken by the read/write head to reach a specific track is called seek time.

Latency time: Time taken by the correct sector to arrive under R/W head is called "Rotational Delay" or "Latency time".

This document is available free of charge on studocu
Scanned with CamScanner
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Average latency time = $\frac{1}{2 \times \text{rotational speed}}$

Average Access time:

Average access time = average seek time + average latency time + Data transfer time + Controller overhead + Queue delay.

Data Transfer time:

Data Transfer time = $\frac{1}{\text{No. of sectors} \times \text{rotational speed}}$

Data Transfer Rate:

Data transfer rate = $\frac{\text{No. of rotations}}{\text{sec}} \times \text{track capacity} \times \text{No. of heads}$

* Disks that are permanently attached to the arm assembly and can not be removed by the user are called "hard disk".
* A disk drive with removable disk are called floppy disk / Compact disk(CD) / DVD etc.
* Magnetic disks used direct access methods.

Scanned with CamScanner
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Magnetic tape:

* Magnetic tape is a medium for magnetic recording made of a thin, magnetizable coating on a long, narrow strip of plastic film.
* In this, only one side of the ribbon is used for storing data.
* It uses sequential access method (reading/writing) on memory location in a serial order.
* Data Read/write operation is slower because of sequential access.
* Magnetic tape has storage capacity ranging from 100 MB to 200 GB.

* Advantages:

1. These are inexpensive.
2. It provides backup or archival storage.
3. It can be used for copying data from hard disk.
4. It is a reuseable memory.
5. It is compact and easy to store.

* Disadvantages:

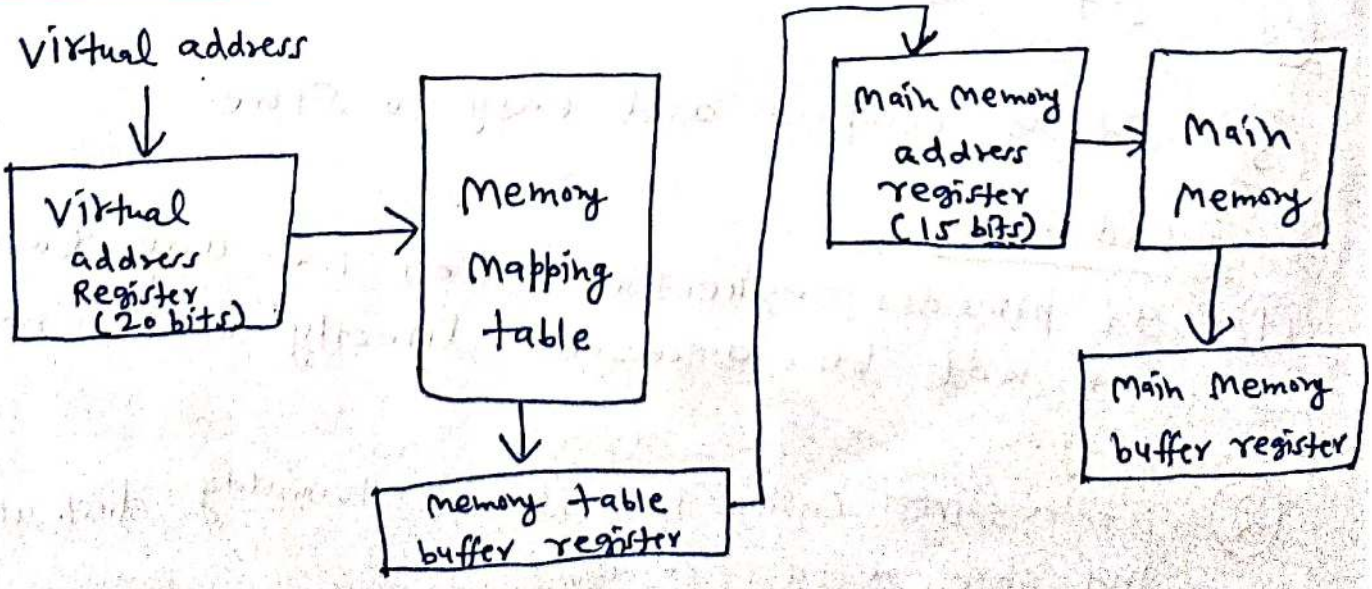
1. It provides sequential access i.e. any location can not be accessed directly which creates delay.
2. It requires caring for storage, humidity, dust also influence data reliability.
3. Difficult to update or modify data.

This document is available free of charge on studocu
Scanned with CamScanner
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Virtual Memory:

* Virtual memory is a concept that gives an illusion to the user that large memory space is available to the user to execute any program however actually such memory space is not available actually.
* Virtual memory ~~Space~~ system provides a mechanism for translating program-generated addresses into correct main memory locations.
* **Address Space:** An address user by the programmer or generated by CPU is called Virtual address/logical address and the set of such addresses is called Address Space.
* **Memory Space:** An address in main memory is called Physical address/Memory address and the set of such addresses is called memory space.

Conversion from Virtual (Logical) address to Physical address is done by using a mapping table.



Scanned with CamS
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Implementation of virtual memory:

The concept of virtual memory can be implemented using

(1) Address Mapping using pages:

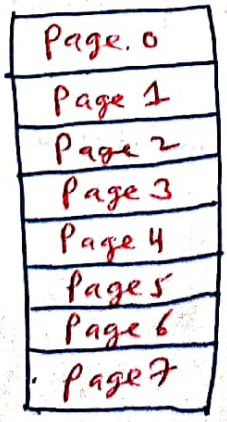
* The logical address space/logical memory is divided into equal size of parts called "Pages".
* The physical address space/Physical memory is divided into equal size of parts called ~~frames/blocks~~.
* Page size is equal to the frame size.
* No. of pages in address space is not equal to the no. of blocks in memory space.
* Ex: If address space = 8K, memory space = 4K and ~~No of~~ Page size = 1K then

~~Block size = Page size~~
~~Frame size~~

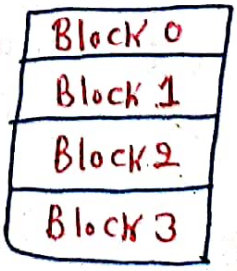
Blocksize = Page size = 1K words

$$\text{No. of pages} = \frac{\text{address space}}{\text{Page size}} = \frac{8K}{1K} = 8$$

$$\text{No. of blocks} = \frac{\text{memory space}}{\text{Block size}} = \frac{4K}{1K} = 4$$

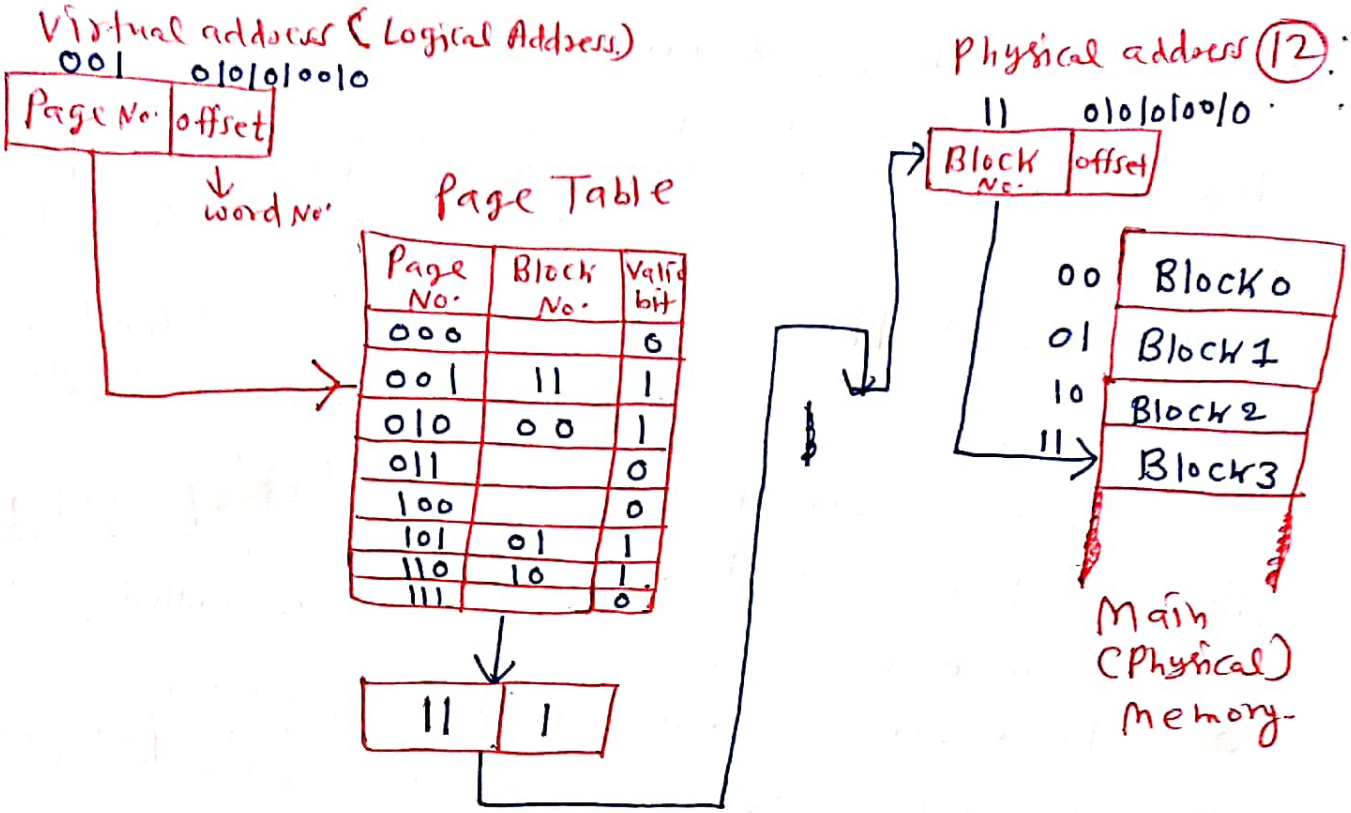


Address Space = 8K



Memory space = 4K

This document is available free of charge on studocu
Scanned with CamScanner
Downloaded by Sangam gupta (sangamg7318@gmail.com)



Memory Mapping using Paging

* Virtual address (Page No., offset) has value

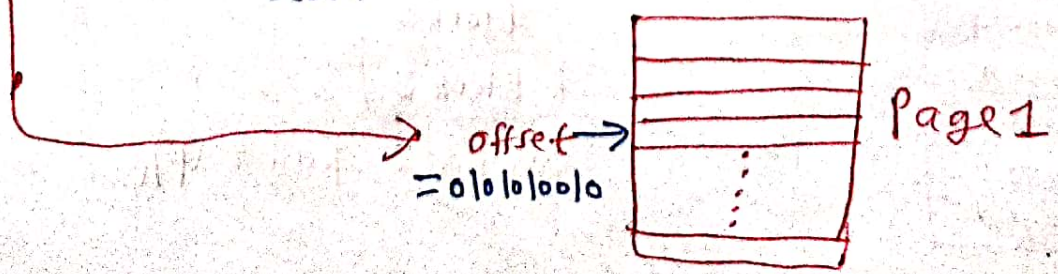
  Page No. = 001 → 3 bits as total Pages = 8

  offset = 0101010010 → 10 bits

Offset specify the address of word within the page/Block. Since Page size = 1K words = 1024 words

So No. of address bits = $2^{10} = 1024$ → 10 bits

* Since Page No. (001) has block No. (11) in the Page table. So Valid bit is said to 1 for it as this page have entry in physical memory.
* Offset value is same in both virtual address and physical address.



Scanned with CamS
Downloaded by Sangam gupta (sangamg7318@gmail.com)

(2) address mapping using Segmented-Paging method:

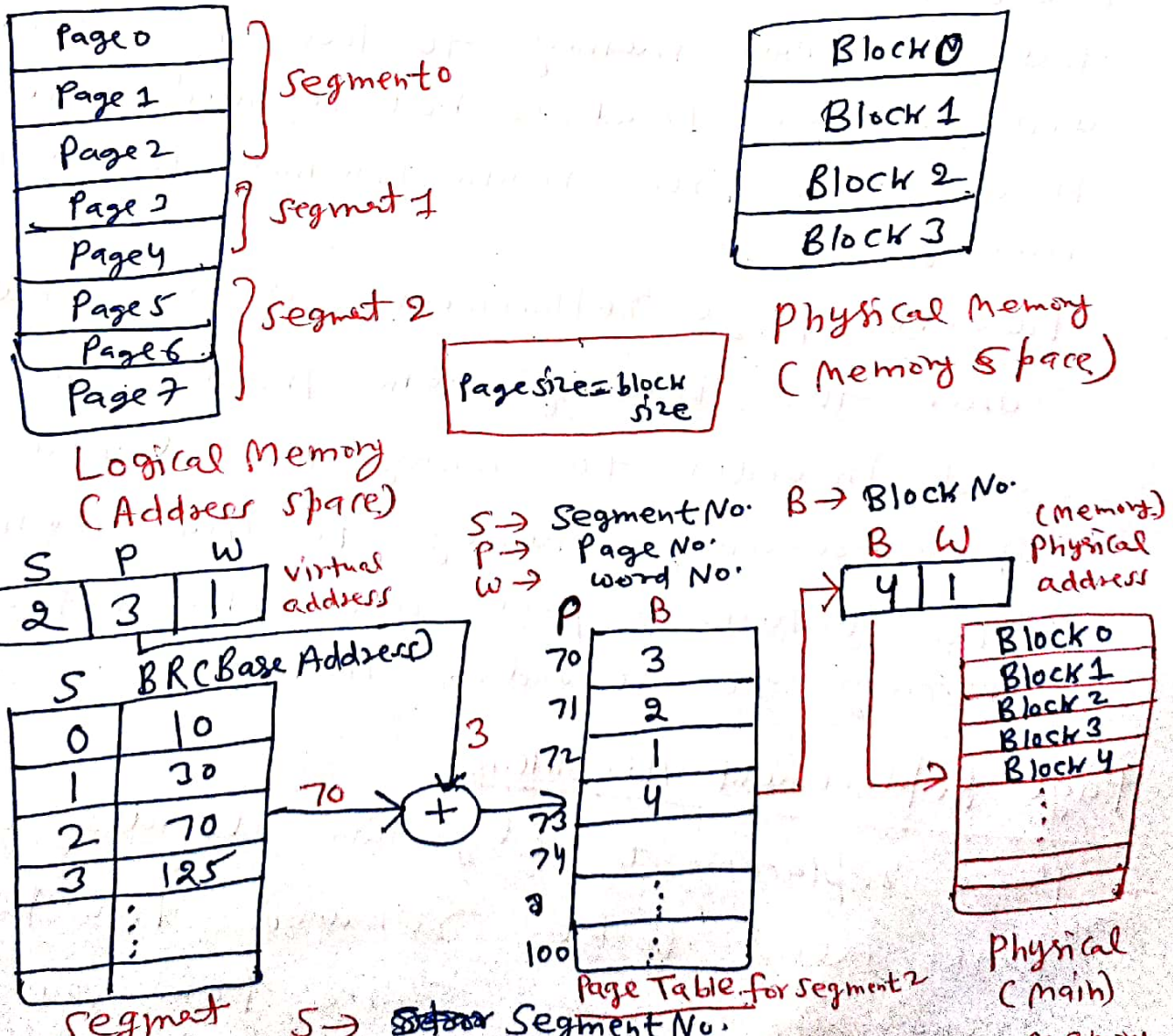
* In segmented paging, each program is considered as a collection of segments. with each segment, is further divided into pages. segments are of variable size.
* The logical address is partitioned into 3 fields:

S	P	W
Segment No.	Page No.	word No. (offset)

* Physical address consists of 2 fields:

Block No.	word No. (offset)

* Each segment has its own page table.
*



This document is available free of charge on studocu
Scanned with CamScanner
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Page Replacement

Before understanding page fault, a few concepts are needed to understand!

Page fault: Page fault occurs, when the desired page is not available in main memory.

Pure Page fault: Pure page fault occurs at the starting of program execution, when no page is available in the main memory.

Page Replacement: Page replacement occurs after page fault in which the required page brings in from the secondary memory to the main memory and the page that is not required have been moved out from main memory to the secondary memory.

However page replacement is only needed when the ~~memory~~ main memory is full and in order to bring the desired page from secondary memory to main memory, the Unnecessary pages are removed from main memory to secondary memory.

Page Replacement Algorithms:

Page replacement algorithm decides which page need to be removed from the main memory to make space for the new pages

Scanned with CamS
Downloaded by Sangam gupta (sangamg7318@gmail.com)

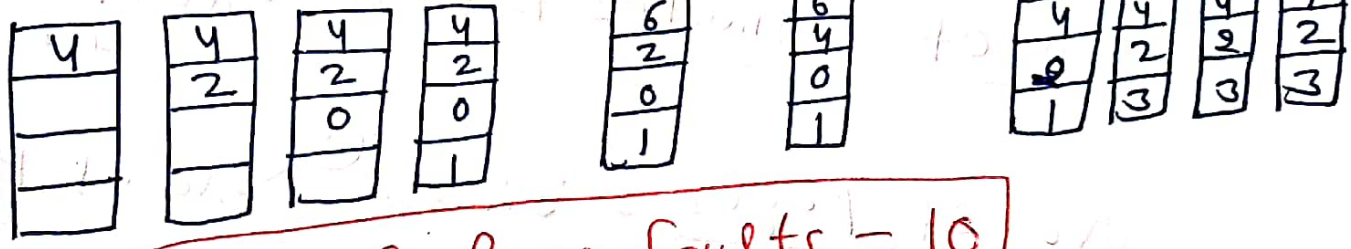
① FIFO (First in First out) Page Replacement algorithm

In this algorithm, The page which comes first in the main memory is to be removed first.

Example: If address space = 8K
memory space = 4K
Page size = 1K
No. of blocks = 4
No. of Pages = 8

(a) If the page reference string sequence is

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



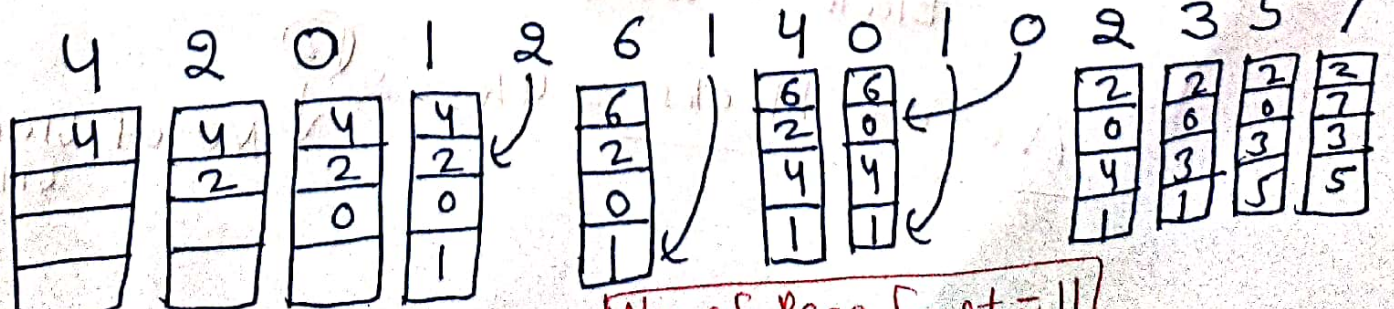
No. of Page faults = 10

② LRU (Least Recently Used)

In this algorithm, the page that has not been used in recent time will be replaced first.

(b) Page Reference string sequence

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



No. of Page fault = 11

This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)
Scanned with CamS

Address Mapping Using Segmented Paging (16)

(Numerical) M. Mano (Problem: 12-23)

The logical address space in a computer system consists of 128 segments. Each segment consists of 32 pages of 4K words in each. Physical memory consists of 4K blocks of 4K words in each. Formulate the logical and physical address formats.

Ans

Segment	Page	Word
7 bits	5 bits	12 bits

Logical address

No. of segments = 128 = 2^{7} → No. of segment bits

No. of Pages in each segment = 32 = 2^{5} → No. of Page bits

No. of Words in each Page = 4K = 2^{12} → No. of word bits

12 bits	12 bits
Block	word

No. of blocks = 4K = 2^{12} → No. of block bits

Scanned with CamS

Downloaded by Sangam gupta (sangamg7318@gmail.com)

No. of address space Numerical

(M. Mano: Problem No: 12-19)

12.19 An address space is specified by 24 bits and the corresponding memory space by 16 bits.

(a) How many words are there in the address space.

(b) How many words are there in the memory space

(c) If a page consists of 2K words, how many pages and blocks are there in the system.

Ans (a) No. of words in address space = 2^{24}

as address space has each address of 24 bits

(b) No. of words in memory space = 2^{16}

as memory space has each address of 16 bits

(c) Page size = 2K words.

So block size = Page size = 2K words

No. of Pages = $\frac{\text{No. of words in address space}}{\text{No. of words in each page}}$

$= \frac{2^{24}}{2K} = \frac{2^{24}}{2^{11}} = 2^{13} = 2^{10} \times 2^3$

$= K \times 8 = 8K$

No. of blocks = $\frac{\text{No. of words in memory space}}{\text{No. of words in each block}}$

$= \frac{2^{16}}{2K}$

$= \frac{2^{16}}{2 \times 2^{10}} = \frac{2^{16}}{2^{11}} = 2^5 = 32$

This document is available free of charge on studocu

Scanned with CamScanner

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Optical disk: An optical disk is any computer disk that uses optical storage techniques and technology to read and write data. It is the computer storage disk that stores data by digitally and uses laser beams to read and write data.

* Non-Volatile.
* laser technology, extreamly fast.
* typical capacity = 700MB.

Advantages of optical disk

* Cheap Media
* CD and DVD can hold lots of media.
* Easy to Carry
* Small in Size.

Disadvantages of optical disk

* Easy to Scrate
* Easy to break
* Can get damaged
* Smaller capacity as compared to Pen drive.

Scanned with CamS
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Numerical on Virtual Memory

A virtual memory has a page size of 1K words. There are 8 pages and 4 blocks. The associative memory page table contains the following entries

Page	Block
0	3
1	1
4	2
6	0

Make a list of all virtual addresses (decimal) that will cause the page fault if used by CPU.

Ans

Address	
0-1023	Page 0
1024-2047	Page 1
2048-3071	Page 2
3072-4095	Page 3
4096-5119	Page 4
5120-6143	Page 5
6144-7167	Page 6
7168-8191	Page 7

Pages

range

Block 0	1K
Block 1	1K
Block 2	1K
Block 3	1K

Blocks

Page size = block size = 1K = 1024 words

↓

(0 to 1023) = 1024 words

This document is available free of charge on studocu

Scanned with CamScanner

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Pages that will not be found in main memory will causes page fault

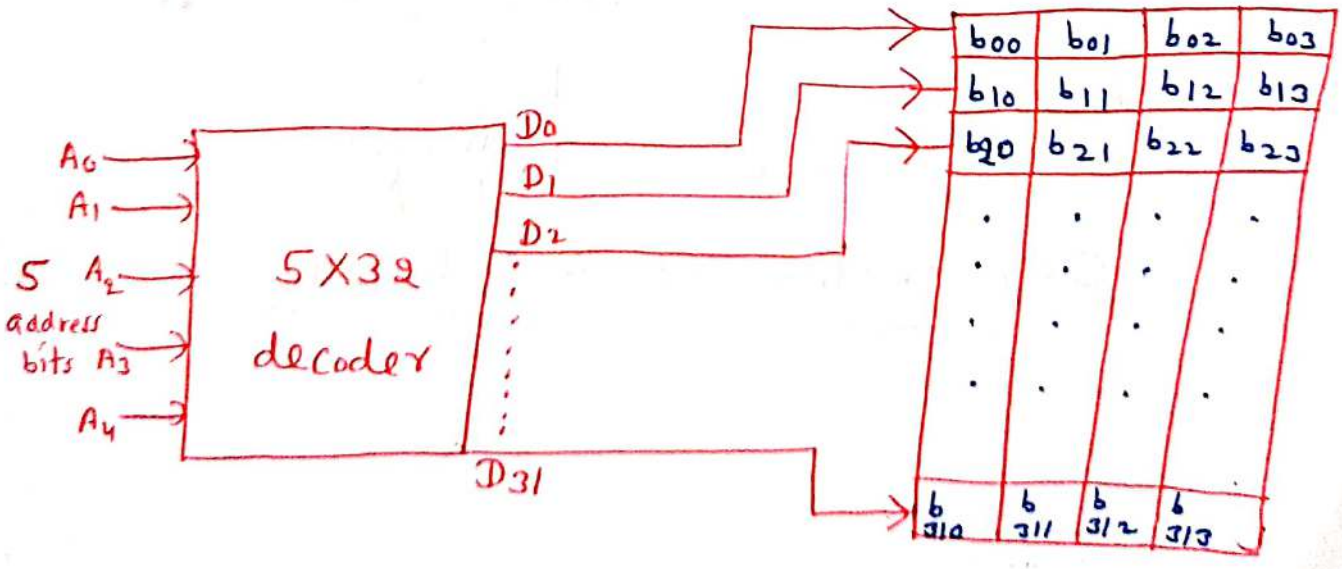
Page No.	Virtual Addresses (decimal)
2	2048-3071
3	3072- 4095
5	5120- 6143
7	7168- 8191

Scanned with CamS
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Difference between 2D RAM & 2.5D RAM

2D RAM:

* In 2D RAM organization, a RAM Consists of $2^n \times m$ Consisting of 2^n memory words of m bits each.
* It has a row decoder of size $n \times 2^n$ to select 1 out of 2^n memory words.
* Each row (memory word) have m Columns (1 column for each individual bit)
* In 2D RAM, hardware is fixed.
* It requires more no. of logic gates.
* 2D RAM organization is more complex relatively.



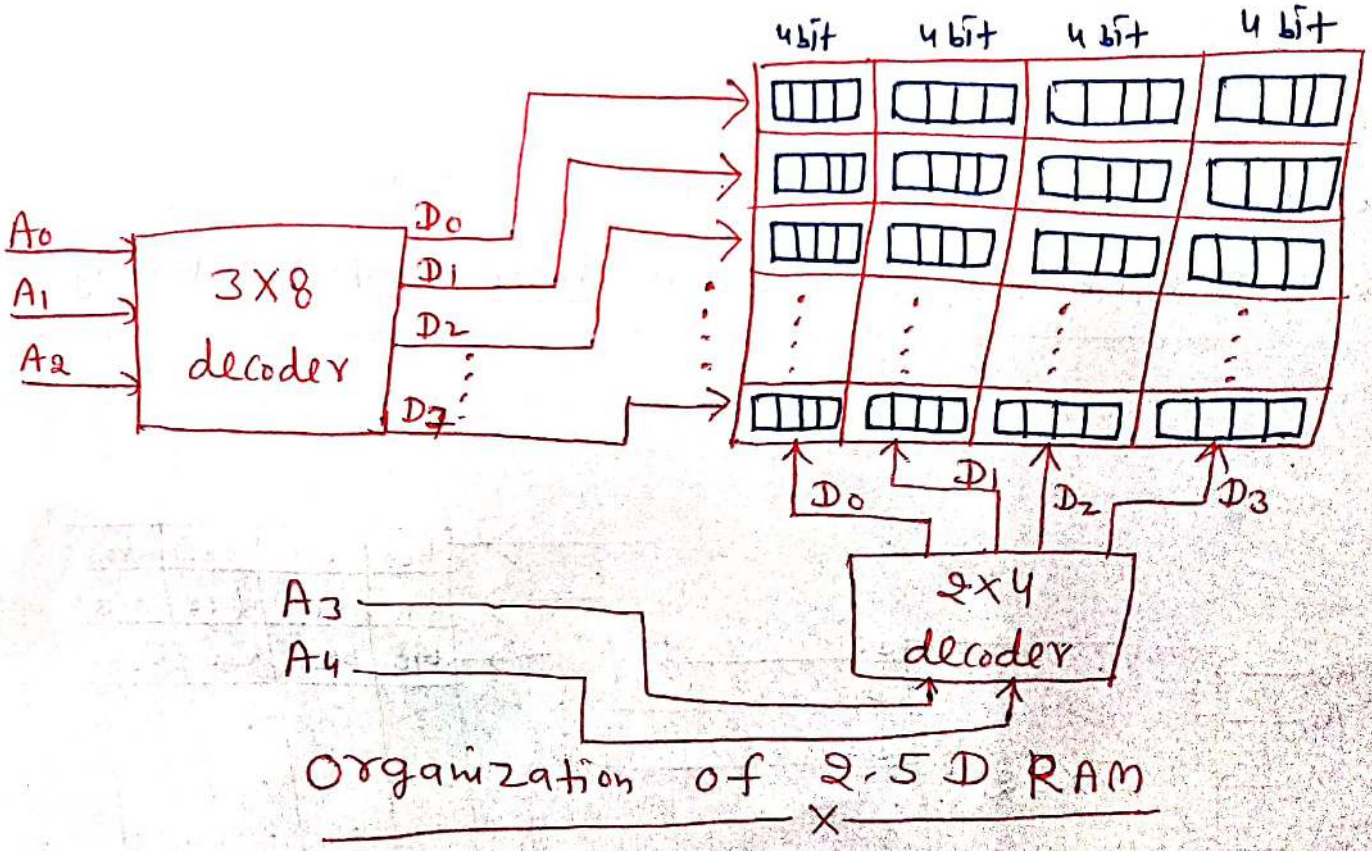
$32 \times 4 \Rightarrow 2^{(5)} \times 4$ → No. of address lines

Organization of 32X4 2D RAM

This document is available free of charge on studocy
Scanned with CamScanner
Downloaded by Sangam gupta (sangamg7318@gmail.com)

2.5D RAM:

* In 2.5D RAM organization, the no. of address lines are divided into approximately equal parts, one for row select decoder and another for column select decoder.
* In 2.5D RAM, hardware is reduable.
* It requires less no. of logic gates.
* 2.5D RAM is less complex than 2D RAM.



Organization of 2.5D RAM

Scanned with CamS
Downloaded by Sangam gupta (sangamg7318@gmail.com)

COA UNit 5 Notes

Computer Organization & Architecture (Dr. A.P.J. Abdul Kalam Technical University)

COA Unit-5 Notes

Peripheral Devices:-

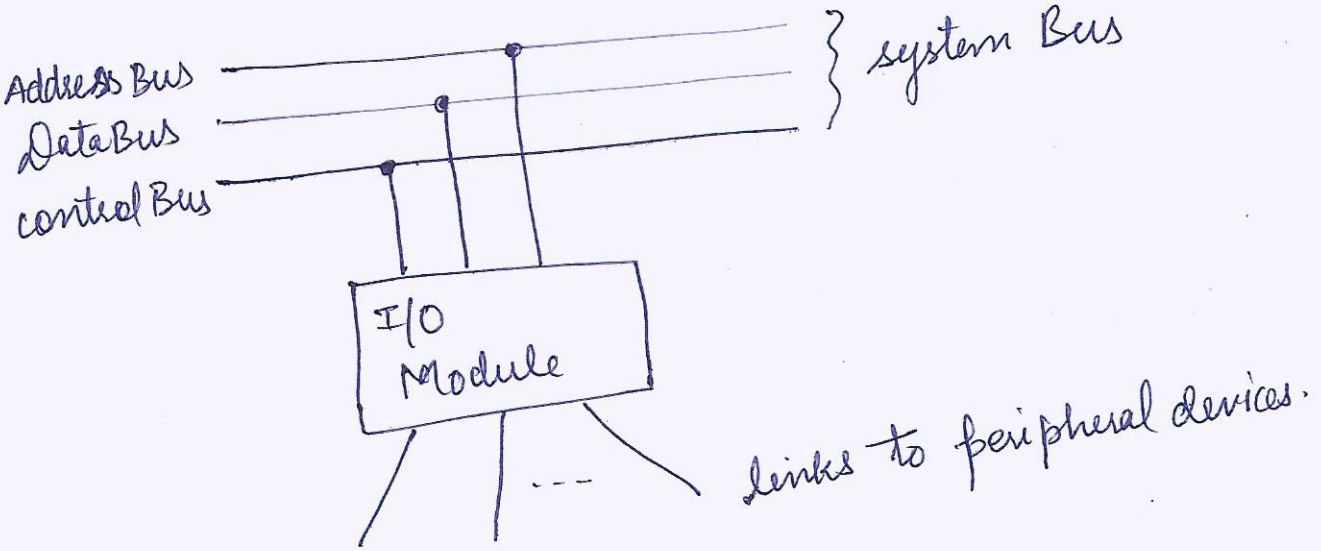
An external device connected to an I/O module is often referred to as a peripheral device or simply a peripheral.

3 categories of external devices:-

(1) Human Readable Device:
suitable for communicating with the computer user (e.g. VDT (Video Display Terminal), Printer, Keyboard, etc).

(2) Machine Readable Devices:-
suitable for communicating with equipment. eg. magnetic disk, sensor etc.

(3) communication Device:
suitable for communicating remote devices.
Eg- The remote device can be the human readable device such as terminal, a machine readable device or another computer.



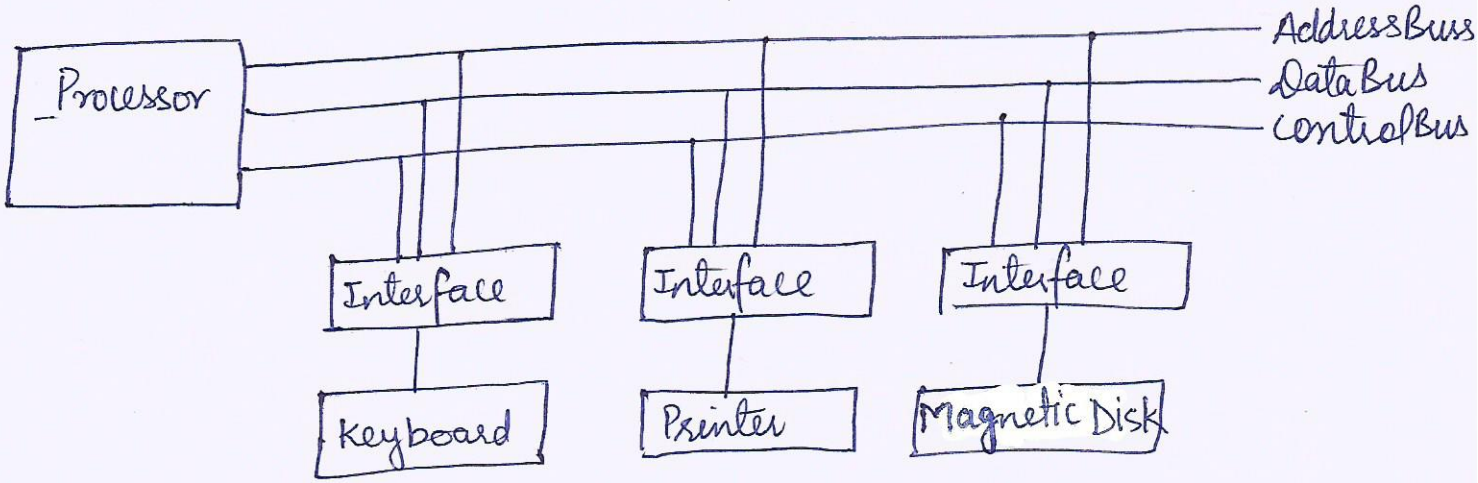
Generic Model of an I/O Module

This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)

I/O Interface:

⇒ I/O interface provide a method for transferring information between CPU and external devices.

⇒ The input output interface resolves the differences that exist between the CPU and each external device.



I/O Bus and Interface Module

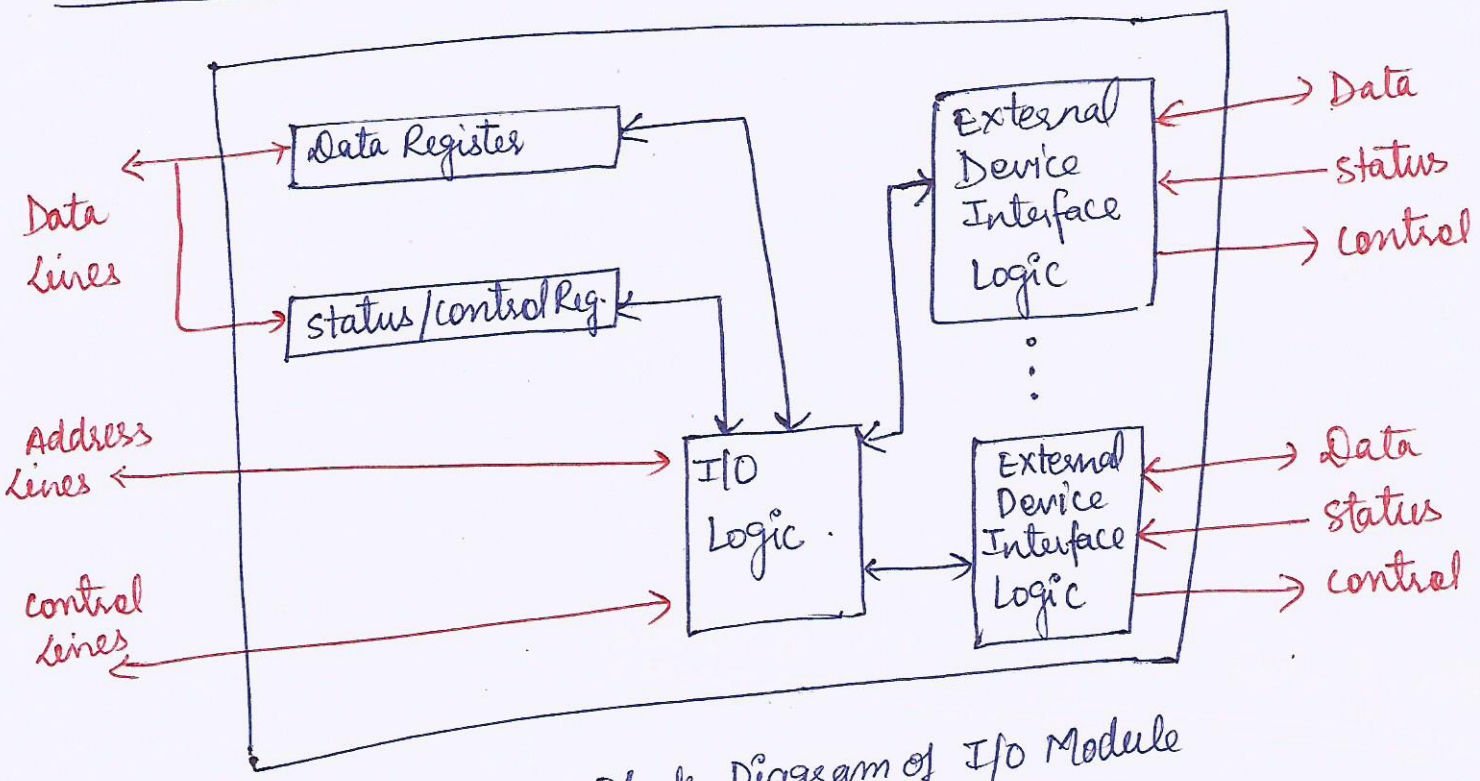
Functions of I/O Module:

→ Major functions for an I/O Module fall into following categories

- control and timing
- Processor Communication
- Device Communication
- Data Buffering
- Error Detection

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Block Diagram of I/O Module



Block Diagram of I/O Module

I/O Port: A connection Point that acts as an interface b/w the computer and external devices like mouse, keyboard, printer, modem etc.

Ports are of two types:-

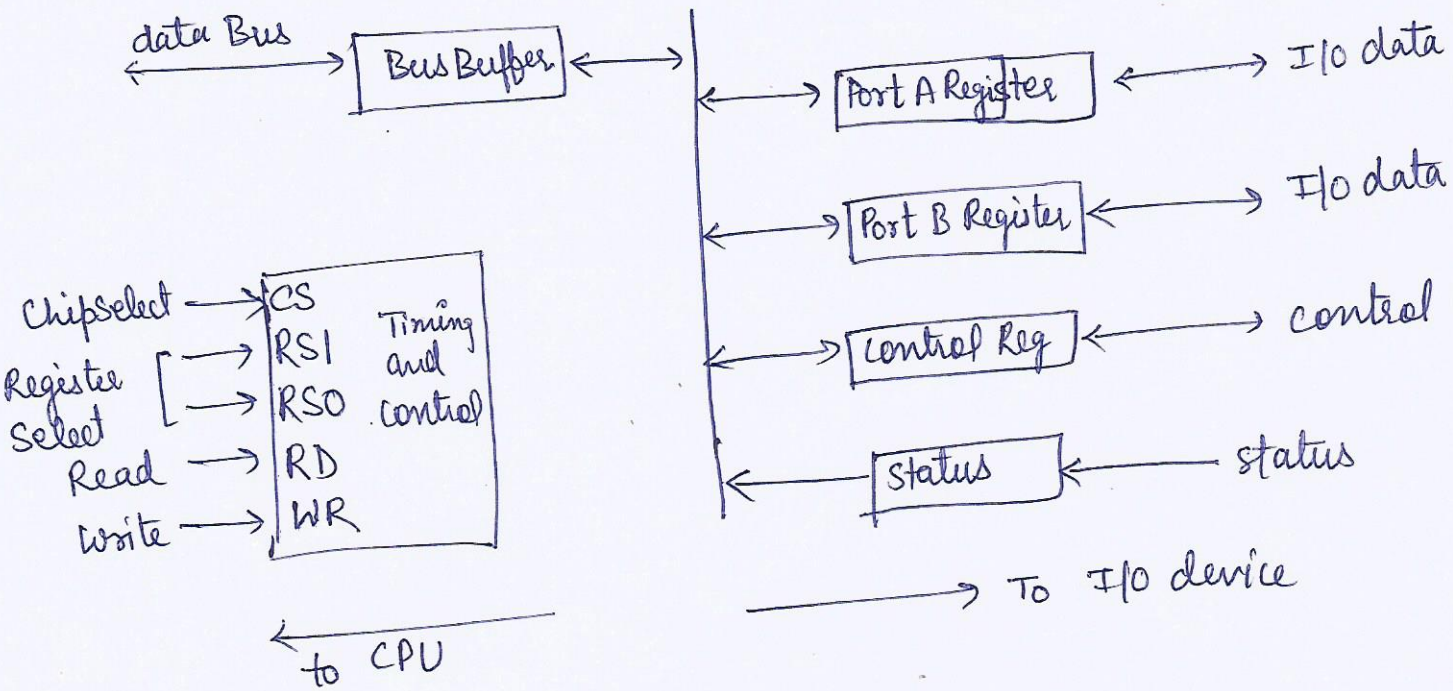
Internal Port: It connects motherboard to internal device like Hard disk drive, internal modem etc.

External Port: It connects the motherboard to external device like, mouse, printer, and keyboard etc.

This document is available free of charge on studocu

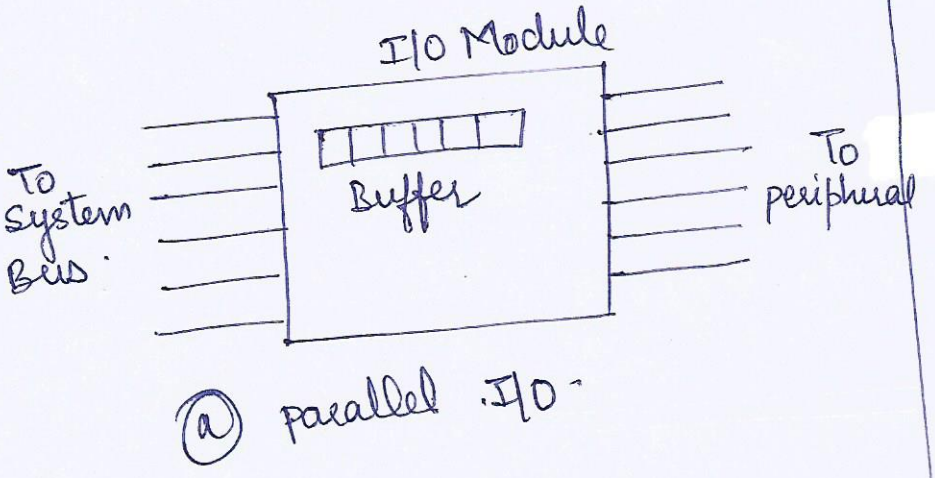
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Example of Interface:

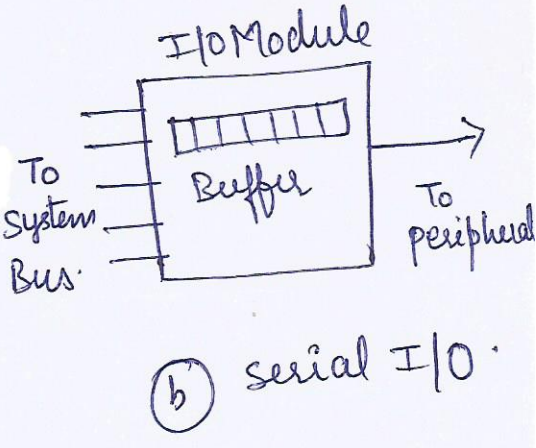


Interfaces
- Serial Interface
- Parallel Interface

* In parallel interface, there are multiple connecting I/O module and the peripheral, and multiple bits are transmitted simultaneously over the data bus. (all of bits of data word transmitted simultaneously)
* In serial interface, there is only one line used to transmit data, and bits must be transmitted one at a time.



(a) parallel I/O.



(b) serial I/O.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

I/O Systems:

User programs never directly interact with I/O devices. All I/O operations are performed with the help of system calls.

There are 4 types of I/O commands:-

→ Control
→ Test Status
→ Read
→ Write

Modes of Data Transfer (I/O Techniques)

- Programmed I/O
- Interrupt Driven I/O
- Direct Memory Access

Programmed I/O

- Input-output instructions written in computer program.
- Each data item transfer is initiated by an instruction in the program
- Transferring data under program control requires a constant monitoring of peripherals by CPU. (once initiated when to transfer)

⇒ ~~Once initiated~~

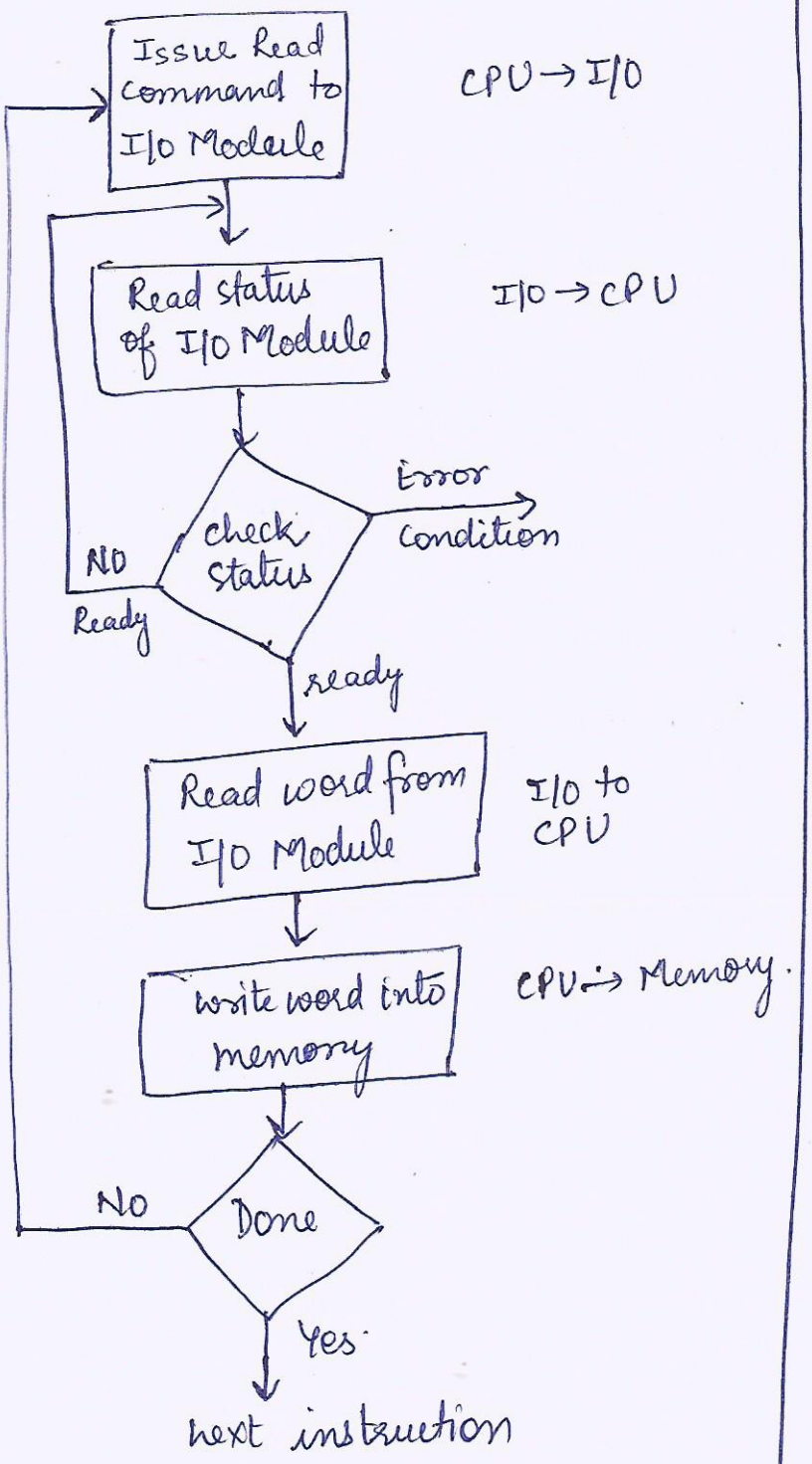
⇒ In this method, the CPU stays in a program loop until I/O unit indicates that it is ready for the data transfer.

⇒ Time consuming

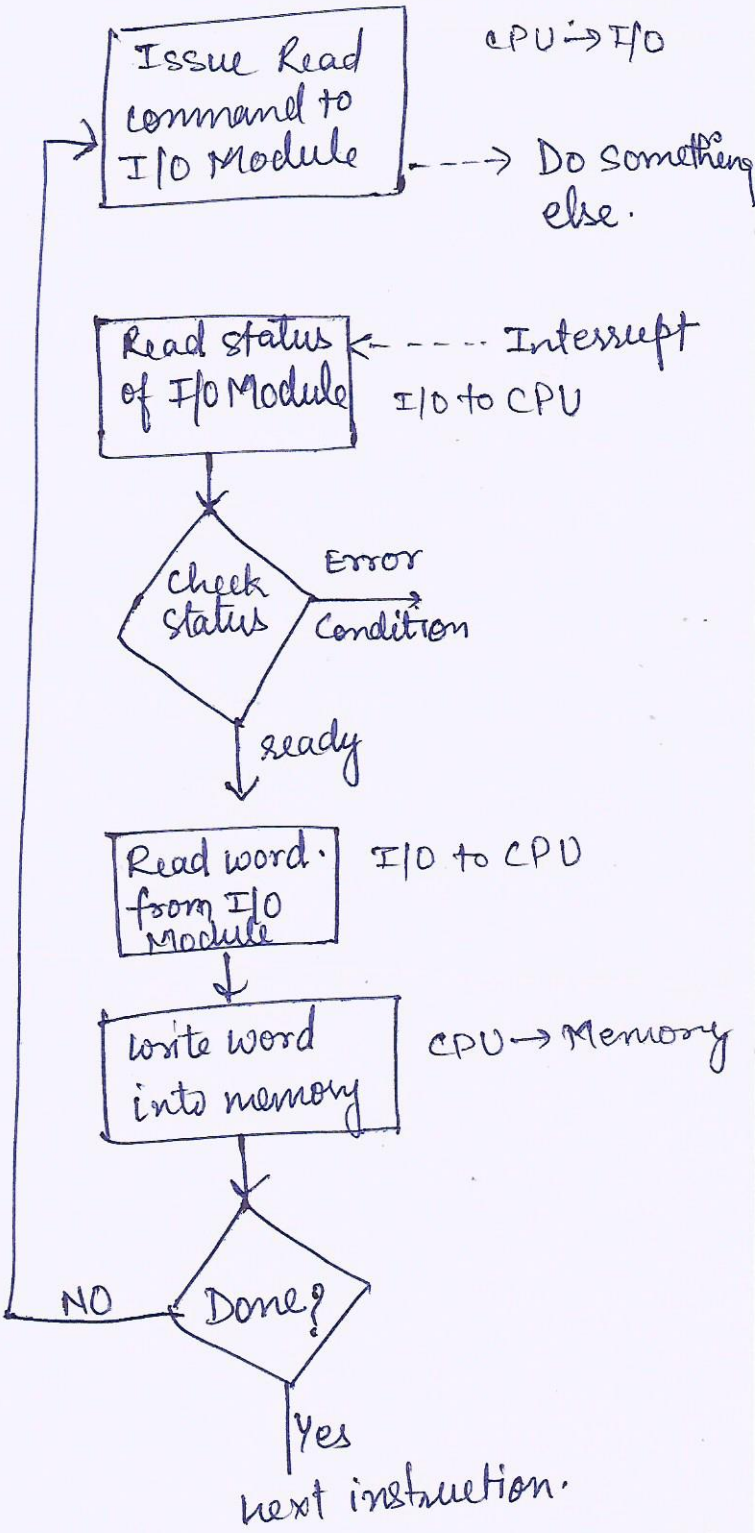
⇒ ~~Less efficient method~~ Less efficiency.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)



Programmed I/O



Interrupt Driven I/O

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Interrupt Driven I/O

→ special commands to inform the interface to issue an interrupt request signal when the data is available from the device. meantime the CPU can proceed to execute another program.

The interface meanwhile keeps monitoring the device.

⇒ when the interface determines the device is ready to data transfer, it generates an interrupt request to the computer.

⇒ upon detecting the interrupt signal, the CPU momentarily stops the task it is processing, branches to a service program to process I/O transfer, and returns to the task it was actually performing.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

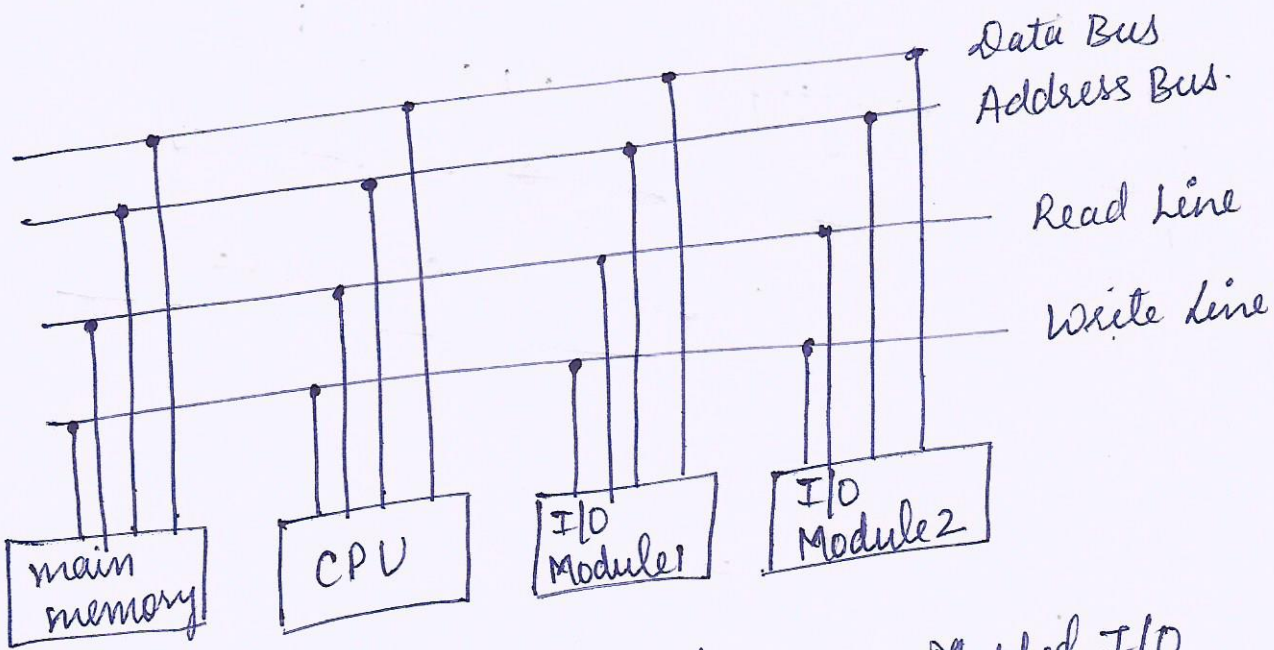
Isolated I/O vs. Memory Mapped I/O

When the processor, main memory and I/O share a common bus, two modes of addressing are possible.

- Memory Mapped I/O
- Isolated I/O (Input-output mapped I/O).

Memory Mapped I/O :-

- There is a single address space for memory locations and I/O devices.
- Processor uses same read instructions to access both the memory and Input-output devices.
- No specific I/O-instructions.
- One set of read and write signals

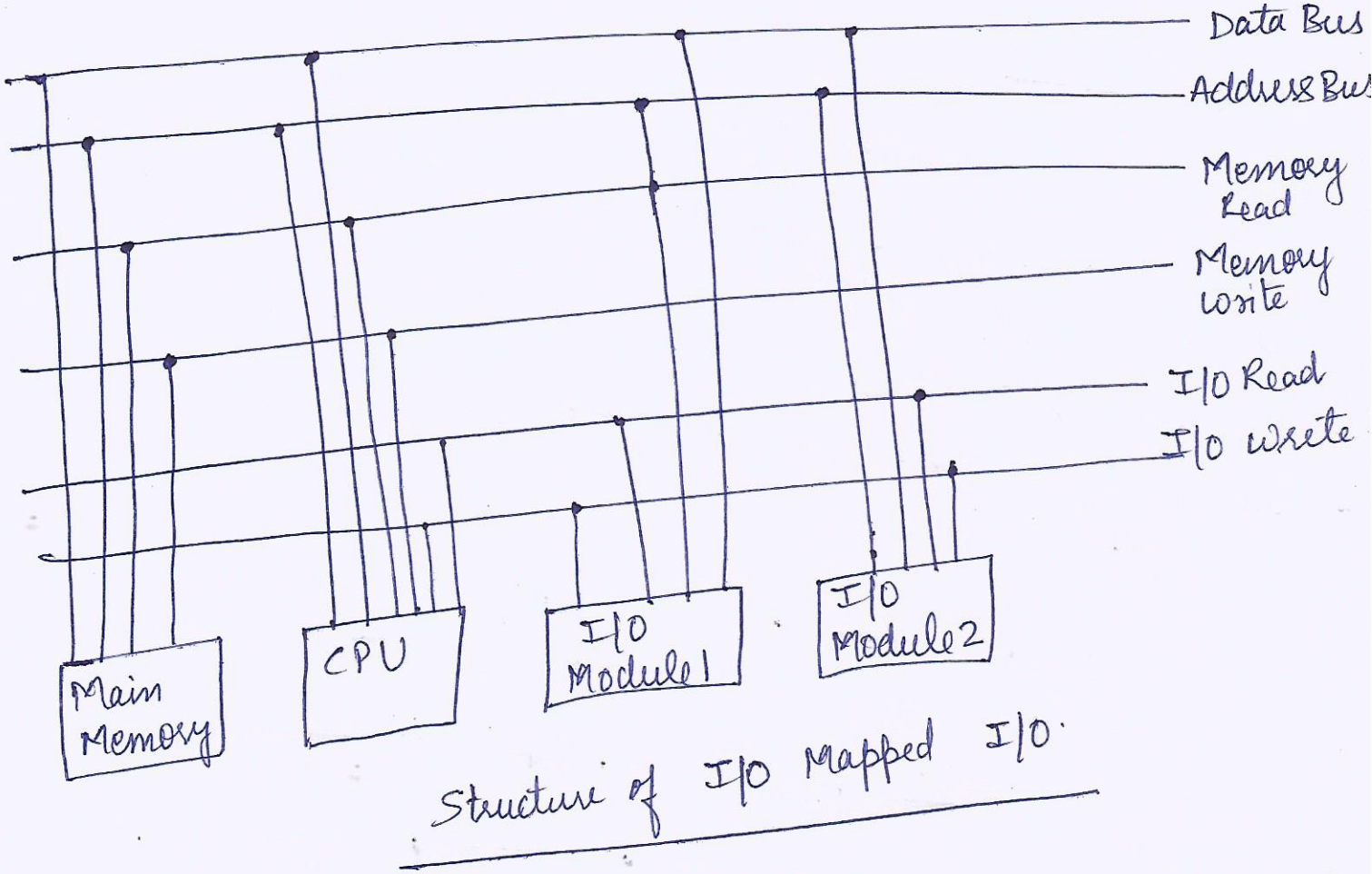


General structure of memory Mapped I/O

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Isolated I/O or Input-output ~~I/O~~ Mapped I/O.

- There are separate address spaces for memory or I/O device.
- CPU specifies that the address on lines is for memory or I/O
- Distinct input output commands.
- ~~separate~~ separate read and write lines.



Structure of I/O Mapped I/O.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Comparison between programmed and Interrupt driven I/O

Programmed I/O	Interrupt Driven I/O
→ Can be implemented without any additional hardware	→ Additional hardware required for interrupt handling
→ based on busy waiting CPU keeps checking the status of I/O device.	→ CPU, switches to another task without waiting
→ Low efficiency	→ higher efficiency than programmed I/O
→ Only one activity can be handled using programmed I/O	→ Multiple I/O activity can be handled in overlapped fashion here

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Comparison between Input-output Mapped (Isolated) and Memory Mapped I/O

Memory Mapped I/O	Peripheral Mapped I/O or Input-output Mapped I/O
– same address space for memory and I/O devices.	→ two (separate) address spaces for memory and I/O devices.
– uses same instructions for both I/O and memory operations	→ separate commands for memory operations and different commands for I/O.
→ one set of read and write signals for memory and I/O	→ separate read and write signals for memory and I/O
– Memory Mapped I/O is less efficient	→ More efficient.

This document is available free of charge on studocu

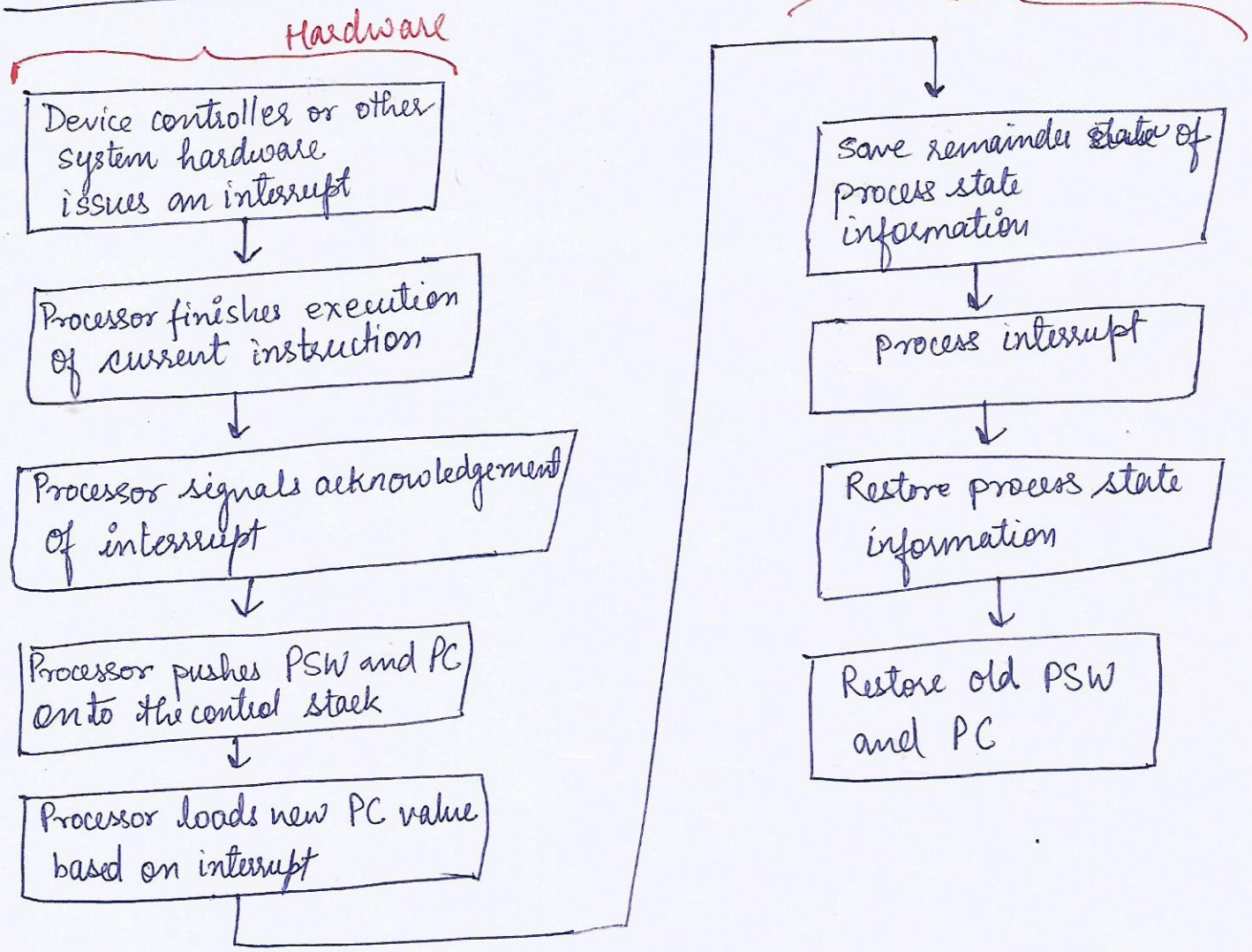
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Interrupt Processing:

The basic method of interrupting the CPU is done by activating a control line that connects the interrupt source to the CPU.

CPU ⇒ * recognizes the presence of interrupt
* executes a specific interrupt handling program.
* determines the source of interrupt
* determines the address (branch address) of the interrupt handling program.

Simple interrupt Processing



Downloaded by Sangam gupta (sangamg7318@gmail.com)

Design issues:

Two design issues in implementing ~~I/O~~ Interrupt I/O.

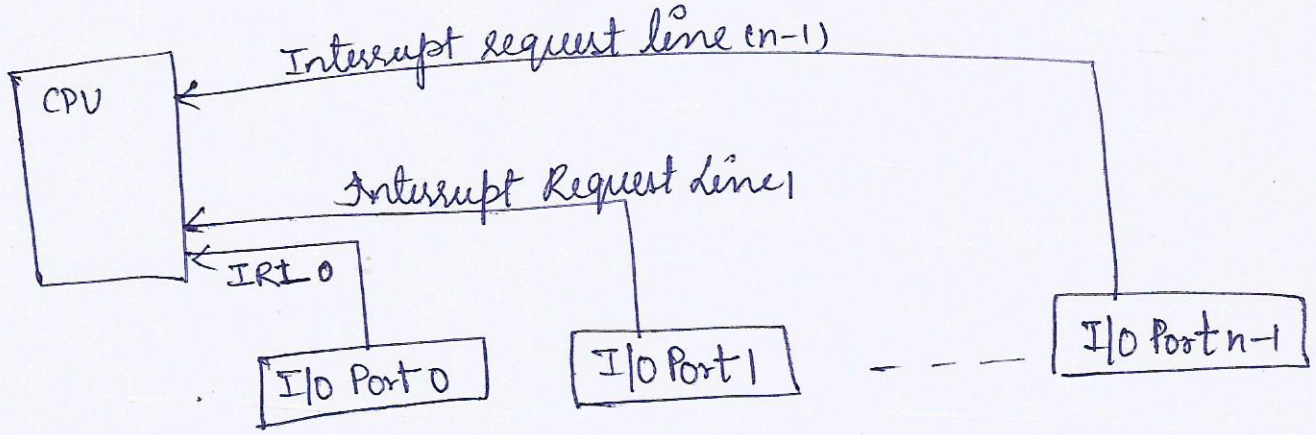
(1) → To determine which device issued the interrupt from multiple devices.

(2) if multiple devices interrupts have occurred, how does the the processor decide which one to process.

Issue: Device identification

4 Techniques exist.
- → Multiple interrupt lines
- → Software Poll
- → Daisy chain (Hardware Poll, vectored)
- → Bus Arbitration (vectored)

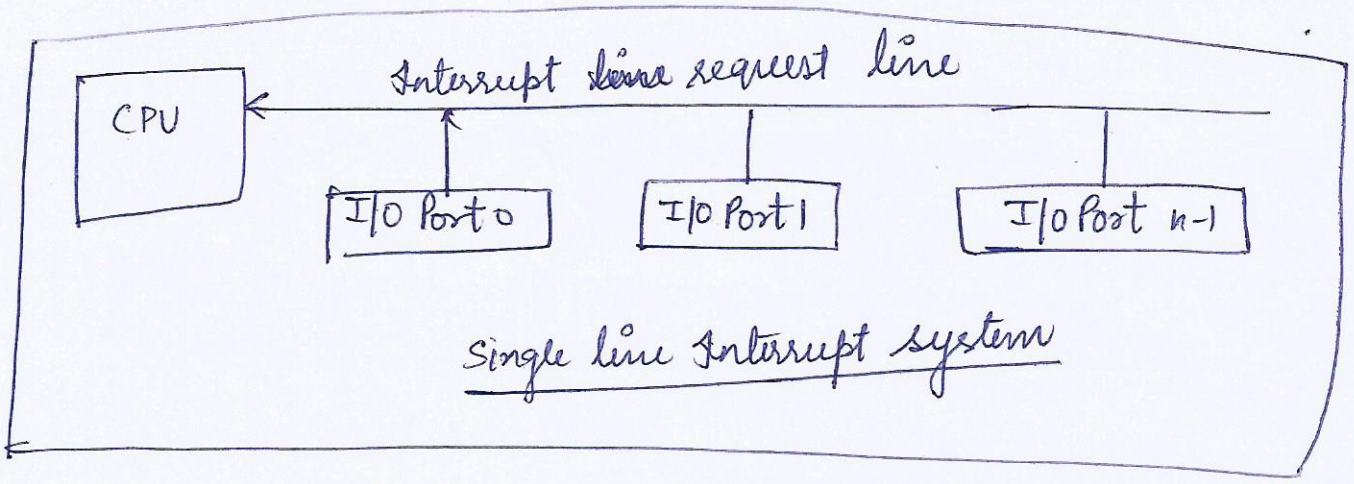
Multiple Interrupt Lines:-



⇒ immediate recognition of device.
⇒ separate Interrupt request line
⇒ impractical approach.

multiple Interrupt line system

This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)



Single line Interrupt system

Software Poll:

⇒ when processor detects the interrupt, it branches to an interrupt service-routine whose job is to poll each I/O module to determine which module caused the interrupt.

⇒ Time consuming

⇒ Priority can be implemented by defining polling sequence.

Vectored Interrupt using Daisy chaining: (Hardware Poll)

⇒ All I/O module shares common interrupt request line

→ The interrupt acknowledgement line is daisychained through the modules.

– when a processor senses the interrupt, it sends out an interrupt acknowledge propagating through a series of I/O module until it gets a requesting module.

– The requesting module responds by placing a word on data lines. This word is referred to as vector.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Bus Arbitration Technique:- (vectored Interrupts)

→ with bus arbitration technique, an I/O module first gain control of bus before it can raise the interrupt request line.

→ Thus only one module can raise the line at a time.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Types of Interrupts:

Program interrupts

These are generated by some condition that occurs as a result of an instruction execution. such as;

- → Arithmetic Overflow
- → Division by zero
- → Execution of illegal machine instruction
- → Segment limit violation
- → Execution of privileged instruction

Timer Interrupts

These are generated within the processor. This allows operating system to perform certain operations on regular basis.

Input-output interrupts:

These are generated for initiation or completion of I/O operations I/O failure or I/O error too can generate an interrupt.

Hardware Failure Interrupts

These are generated by a failure; such as power failure or memory parity error.

Interrupts vs. Exceptions:

An interrupt is generated by a signal from hardware, and it may occur at random times during execution of a program.

An exception is generated from software, and it is provoked by the execution of an instruction.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Hardware and Software interrupts

→ When microprocessor receive interrupts signals through pins (hardware) of microprocessor. These are known as Hardware interrupts

→ Software interrupts are those which are inserted in the between the program which means these are mnemonics of the microprocessor.

Vectored and non vectored interrupts

- In vectored interrupts, the source (or device) that interrupts supplies the branch information to the computer.
- In a non-vectored interrupt, the branch address is assigned to a fixed location in the memory.

Maskable and non-maskable:

- Maskable interrupts are those which can be disabled or ignored by the microprocessor.
- Non-maskable: which can not be disabled or ignored by the microprocessor.

Types of interrupts:

4 categories

→ Program interrupts
→ Timer interrupts
→ Input/output interrupts
→ Hardware failure

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Direct memory Access :-

→ Both interrupt driven and programmed I/O require involvement of CPU for data transfer.

→ CPU can be better utilized for program execution.

→ I/O activities are slow and involvement of CPU will not have a desired effect on the system performance.

→ DMA increases the speed of I/O transfer by eliminating the role played by CPU in I/O operations.

→ when large amount of data is stored/transferred from CPU, a DMA module can be used.

DMA operates in the following ways:

when I/O is requested, the CPU instructs the DMA module about the operation, by providing information:

- → Source address
- → Target address
- → Read / write
- → Number of words to be read or written.

⇒ CPU loads the DMA registers 'Data count' and 'Address Register' with number of bytes to be transferred and starting address memory location respectively.

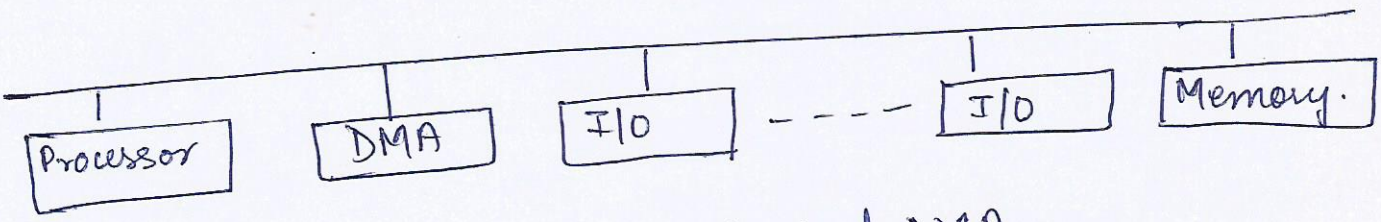
Downloaded by Sangam gupta (sangamg7318@gmail.com)

When the DMA controller is ready to transmit or receive data, it activates the DMA request line to CPU. CPU wait for the next break point. CPU now leaves the control of system bus and activates DMA acknowledge.

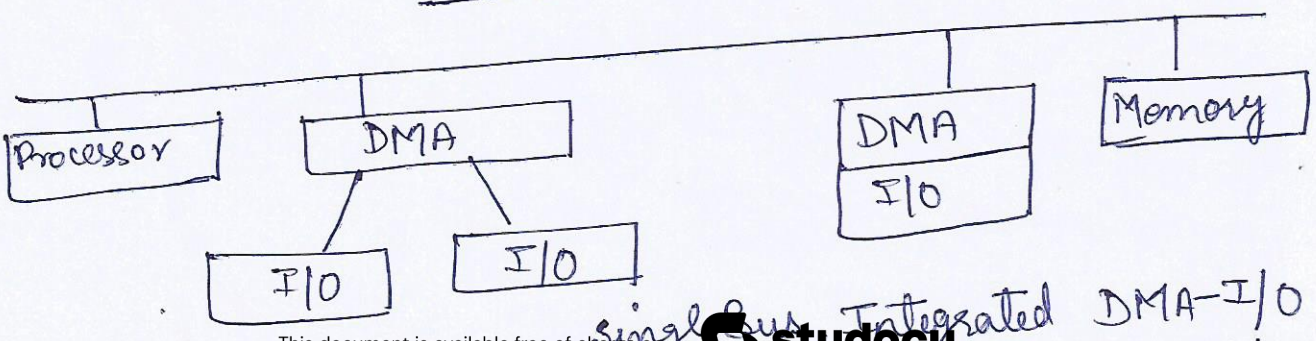
→ The DMA controller transfers the data directly from or to main memory. After a word is transferred, Data count (or word count register) and Address register are updated.

→ If the data count register has not reached zero but the I/O device is not ready to send or receive the data, the DMA controller releases the system bus to CPU by deactivating DMA request line.

- if data count register is decremented to zero, DMA controller finally relinquishs the control of system bus. It may also send an interrupt signal to CPU to indicate the end of data transfer.

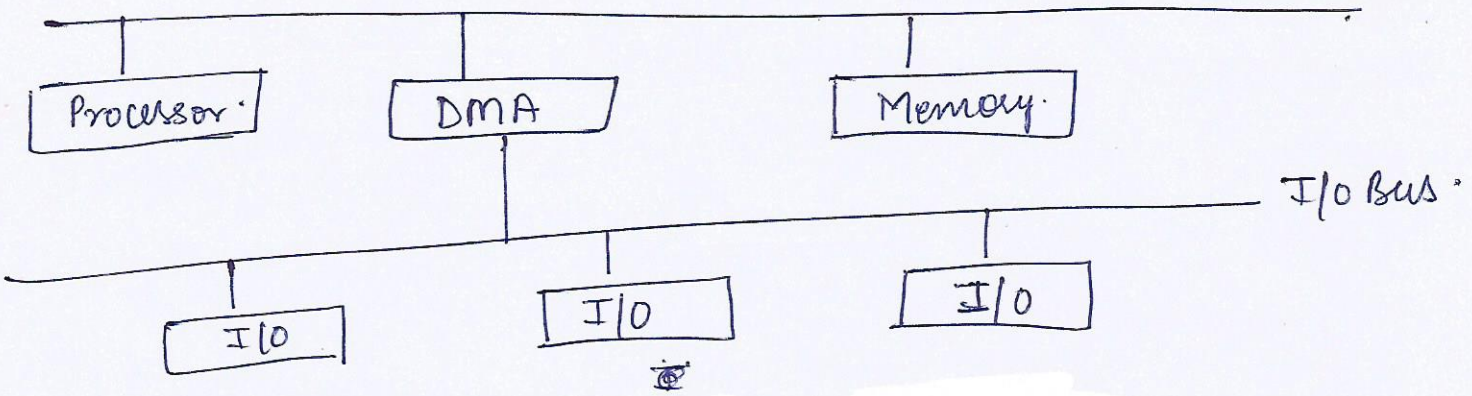


Single Bus Detached DMA



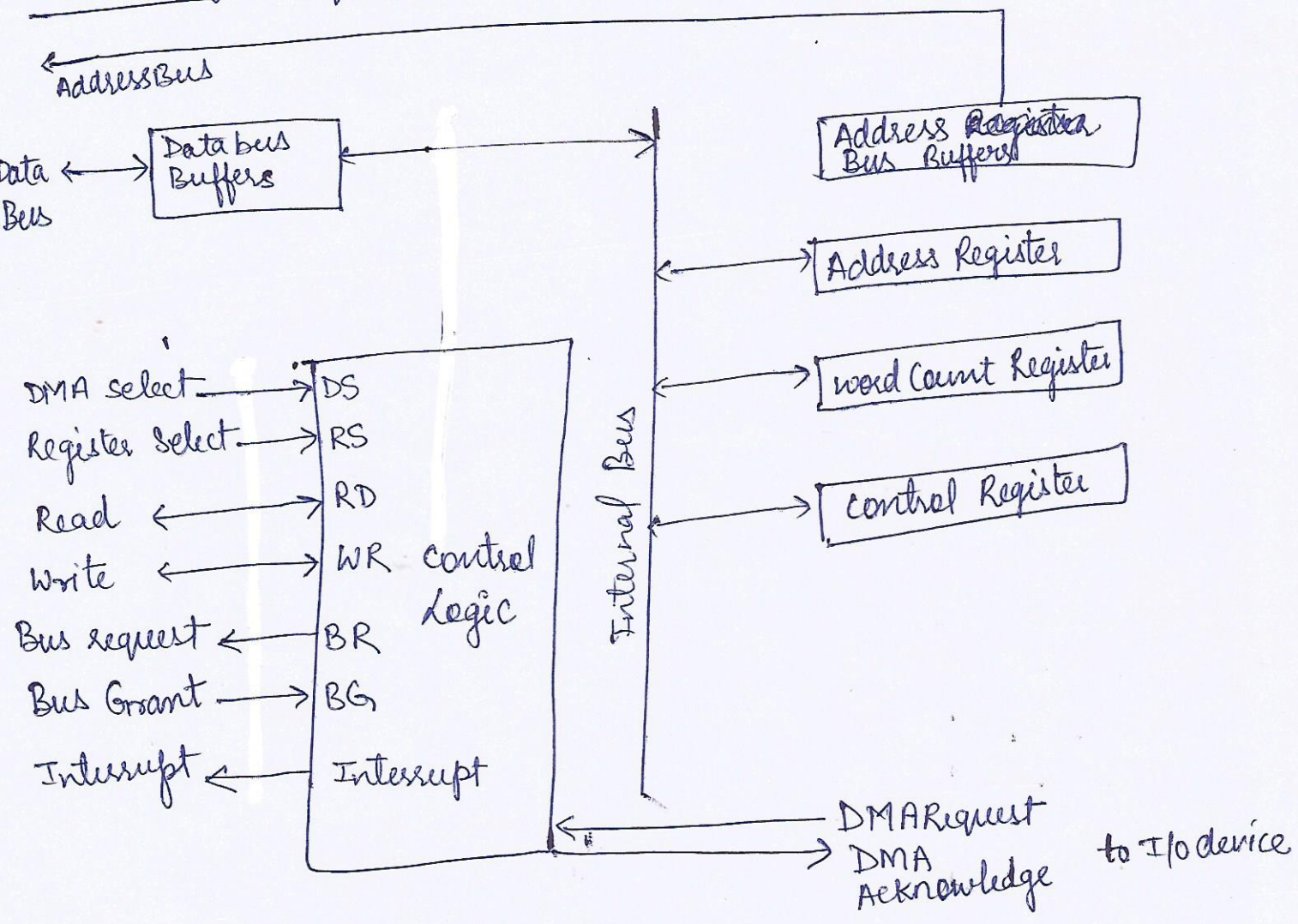
Single Bus Integrated DMA-I/O

This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)



I/O Bus

Block Diagram of DMA



Downloaded by Sangam gupta (sangamg7318@gmail.com)

→ Bus Request (BR) input is used ~~to~~ by the DMA controller to request to CPU to leave the control of system bus.

→ The CPU activates the 'Bus Grant' (BG) output to inform the DMA controller.

→ When DMA terminates the transfer, it disable the bus request line. CPU disables the bus grant; takes control of buses and returns to its normal operation.

DMA Data Transfer modes:

→ DMA block transfer
→ Cycle stealing mode
→ Transparent DMA.

(1) DMA Block Transfer:-

→ In DMA block transfer technique, a block of data of arbitrary length can be transfered in a single burst.
→ This DMA mode is needed for secondary memories like disk drivers, where data transmission can not be stopped or ~~stoused~~ slowed without loss of data.

This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)

(2) cycle steading Mode:

In cycle stealing mode, the DMA controller is allowed to transfer one word at a time, after which it must return the control of the bus to the CPU.

* DMA module must use the bus only when the processor does not need it, or it must force the processor to suspend operation temporarily.

DMA module transfers a word and returns the control to the processor Note that this is not an interrupt; the processor does not save a context and do something else.
Rather, processor pauses for one bus cycle. The overall effect is to cause the processor to execute more slowly. Nevertheless, for a multiple-word I/O transfer, DMA is far more efficient than interrupt driven or programmed I/O.

(3) Transparent DMA:

In transparent DMA, DMA is allowed to steal only those cycles when the CPU is not using the system bus.
CPU does not require to have control of system bus during Decode instruction or execute instruction phase.

* DMA transfering data during transparent DMA does not have any adverse effect on CPU performance.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Input - output channels:

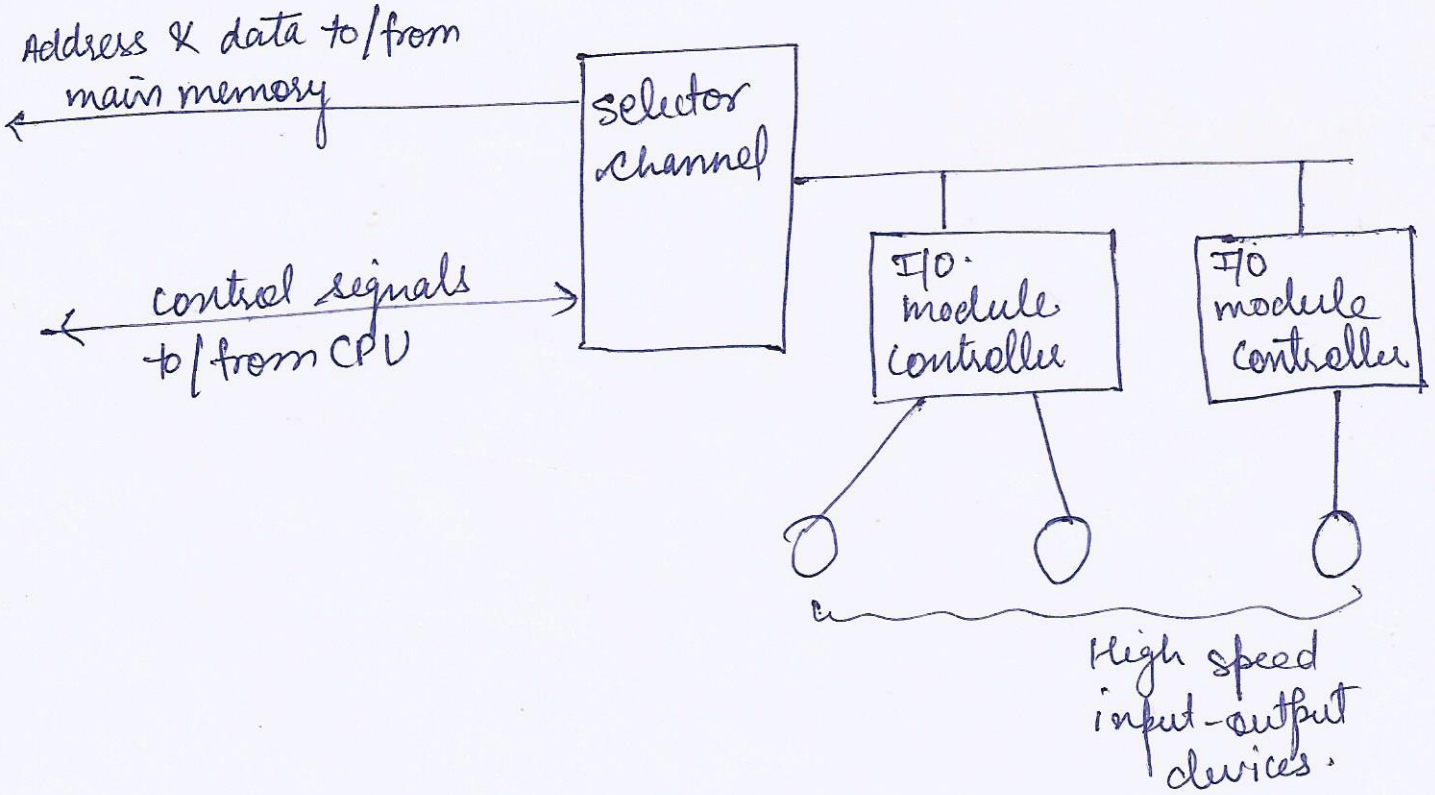
Input-output channels or input-output processors, virtually retrieves CPU from all I/O related activities. It has complete control over device. It can also communicate with CPU.

* I/O processor is a processor like CPU with limited number of instructions. An I/O processor has the ability to execute the I/O instructions It can also carry out the work of data conversion required by the input-output device.

There are two types of I/O channels:

1. Selector channel
2. Multiplexer channel.

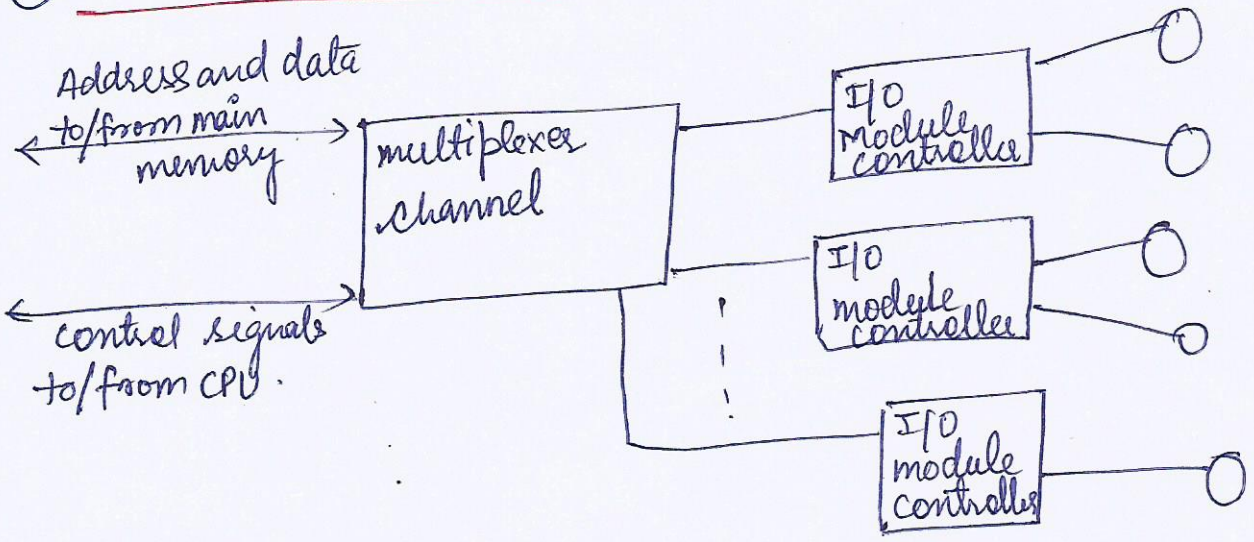
1. Selector channel:



This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)

- A selector channel can handle multiple high speed devices. Only one device is selected at a time for communication.
- A selector channel is useful in high speed transfer of data in burst mode.

(2) A multiplexer channel:



A multiplexer channel can handle I/O with a number of devices at the same time. If the device is slow then byte, multiplexing is used.

Example There are three devices which need to send individual bytes are

B1, B2, ----- B5 → data from 1st device.
X1, X2, ----- X5 → data from 2nd device.
Y1, Y2 - --- Y5 → data from 3rd device.

Then on a byte multiplexer channel may send the ~~data~~ bytes as B1 X1 Y1 B2 X2 Y2 B3 X3 Y3 B4 X4 Y4 - ----.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Data Transfer:

- Synchronous
- Asynchronous

Synchronous: If registers in the interface share a common clock with CPU registers, the transfer between two units is said to be synchronous.

Asynchronous: The internal timing in each unit is independent from the other in that each uses its own private clock for internal registers, In this case the two units are said to be asynchronous.

Asynchronous Data Transfer:

→ two independent units transmit control signals to indicate the time at which data is being transmitted.

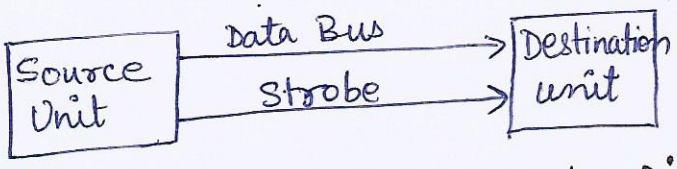
→ Two methods
- Strobe
- Handshaking

Strobed Method (one way) One of communicating device supplies all the control signals.

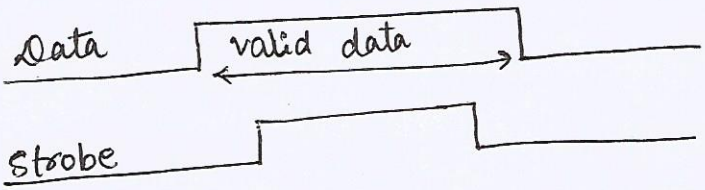
- Strobe can be activated by either source or destination

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)



Source iniated Strobe Diagram



Timing Diagram

The strobe is a single line that informs the destination unit when a valid data word is available in the bus.

* The source unit first places the data on the data bus. After a brief delay to ensure that the data settle a steady value, the source activates the strobe pulse.

* In destination initiated data transfer, destination unit activates the strobe pulse, informing the source to provide the data.

Disadvantage: In strobe method, the source unit that initiates the transfer has no way of knowing whether the destination unit has actually received the data item.

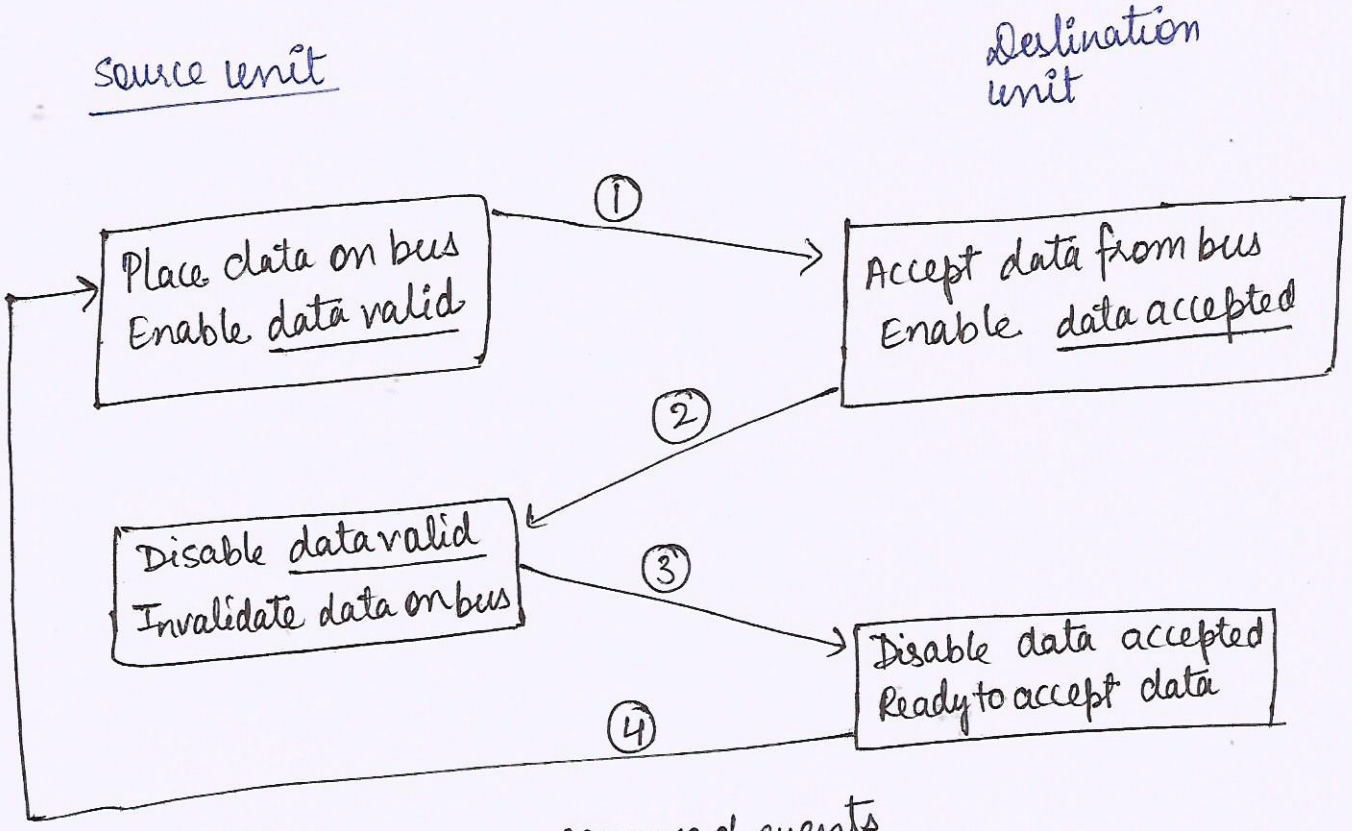
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Handshaking: (Two way)

* The disadvantage of oneway control is that it does not verify that the data transfer has been completed successfully. Data may be lost.

* Control of data transfer is based on the use of handshake between processor and the device selected for input-output.

Block Diagram



sequence of events

This document is available free of charge on studocu
Downloaded by Sangam gupta (sangamg7318@gmail.com)

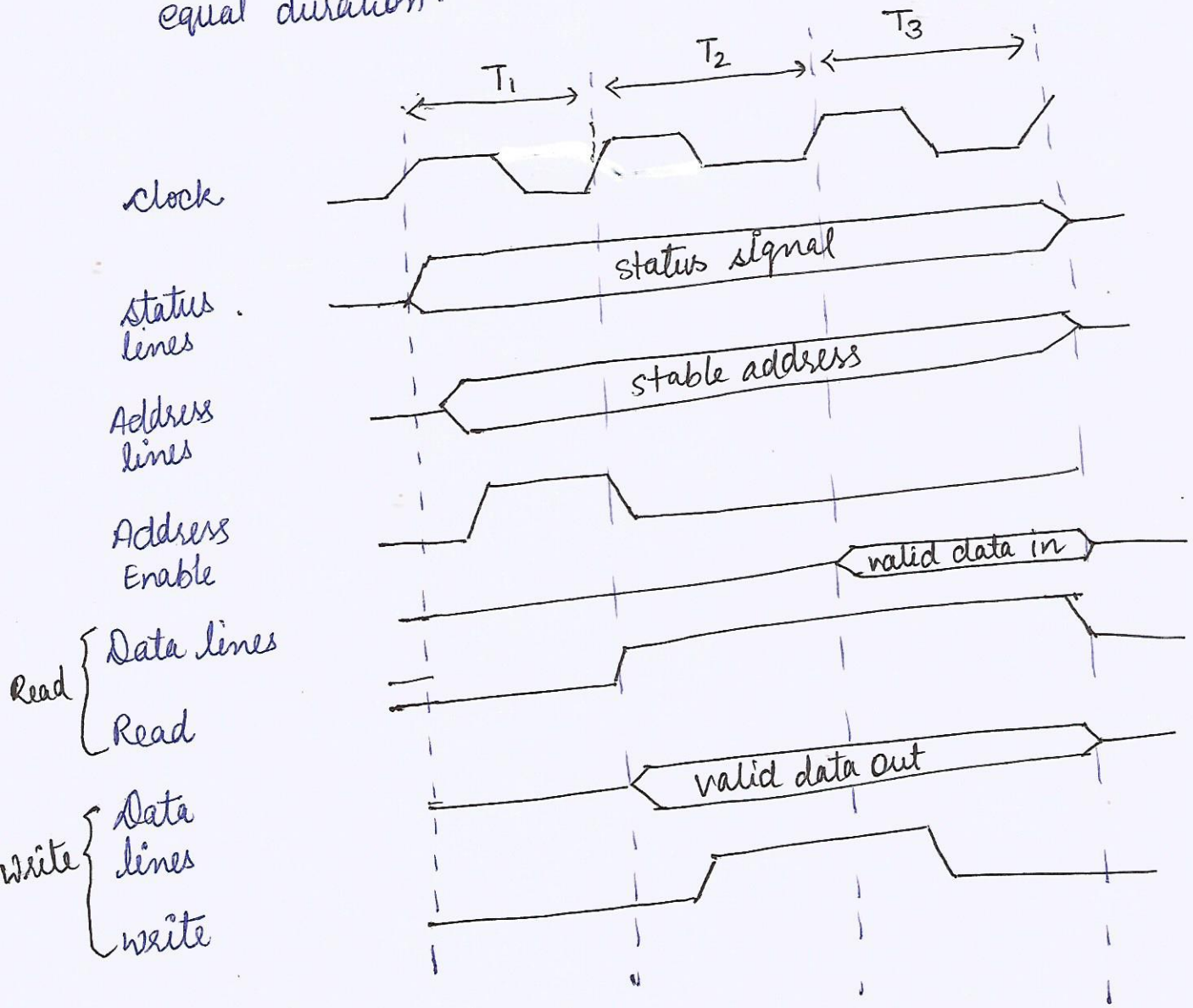
Synchronous Bus:

→ include a clock in the control lines
→ A fixed protocol for communication that is relative to the clock

Advantage: - involves very low logic and can run very fast.
Disadvantage: Every device on the bus must run at the same clock rate.

Synchronous timing

→ The occurrence of events on the bus is determined by a clock
→ A clock is a regular sequence of alternating 1's and 0's of equal duration.



Downloaded by Sangam gupta (sangamg7318@gmail.com)

$T_1 \Rightarrow$ processor places address on address lines and may assert various status lines.

$\Rightarrow$ Once the address lines have stabilized, the processor issues address enable signal.

For Read operation

$T_2 \Rightarrow$ Processor issues a read command.

$T_3 \Rightarrow$ Memory recognizes the address and places the data on the data lines.

For write operation

$T_2 \Rightarrow$ processor puts the data on data lines.

$\Rightarrow$ Once the data on data lines have stablized, the processor issues a write command.

$T_3 \Rightarrow$ Memory copies the information from data lines

This document is available free of charge on studocu

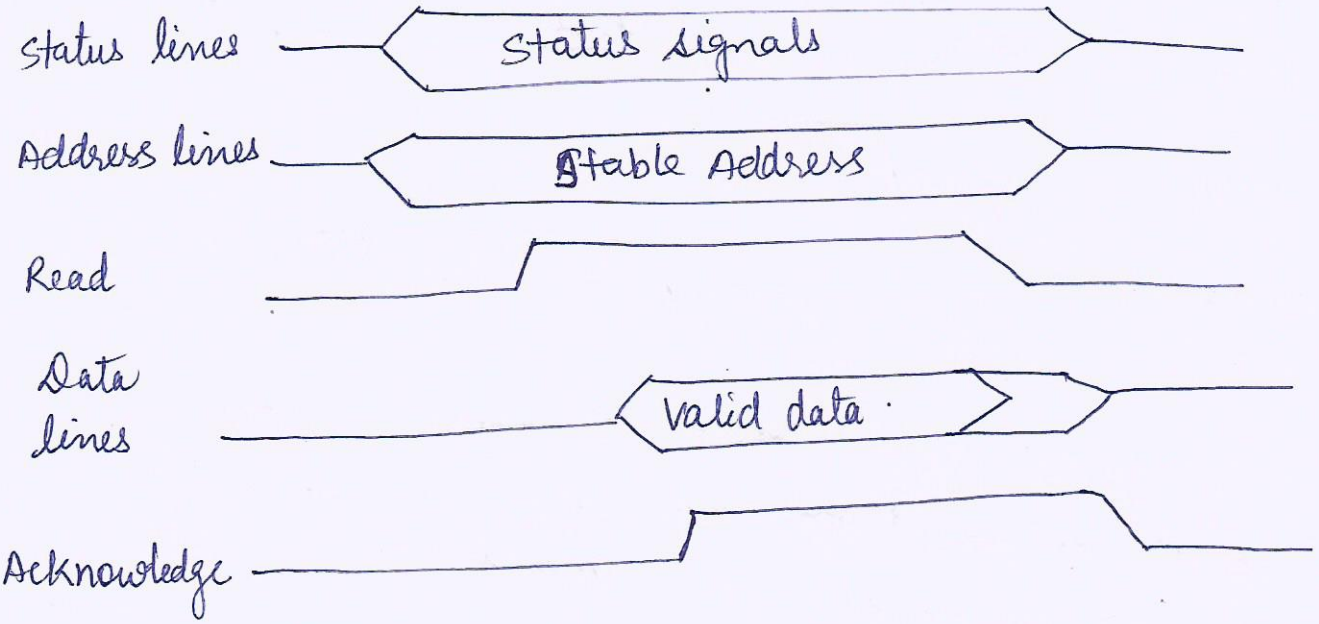
Downloaded by Sangam gupta (sangamg7318@gmail.com)

Asynchronous Bus:

⇒ No clock

⇒ The occurrence of one event on a bus follows and depends on the occurrence of previous event.

Sequence of events for read operation:



- The processor places address and status signals on the bus.
- After signals have stabilized, the processor issues a read command.
- After address decoding, memory places the data on data lines.
- After stabilized data, memory asserts ~~ack~~ acknowledges to the processor.

Synchronous Timing

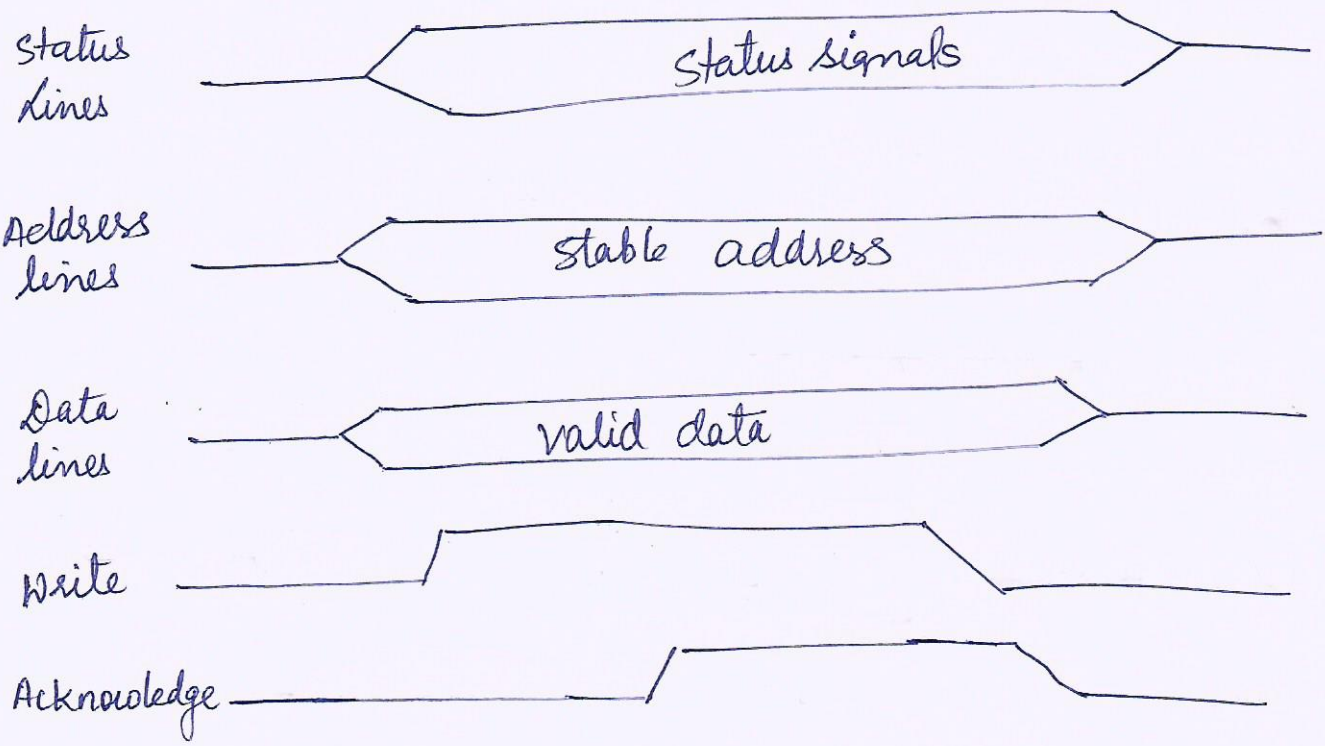
→ Simple to implement and test
→ Less flexible
→ Devices must be on same clock rate

Asynchronous Timing

- Hard to implement and test
- More flexible
- combination of slow and fast devices.

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Sequence of events of Write operation:



- The processor places address and status signals on the bees.
- The master places the data lines at the same time
- The processor issues write command.
- → Memory module responds to the write command by copying the data from the data lines and asserting the acknowledge line.
- The master then drops the write signal and memory module drops the acknowledge signal.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Serial communication

or

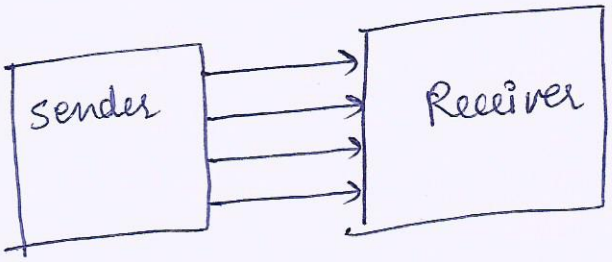
Serial Transmission

⇒ Serial data transmission occurs by transmitting data one bit at a time in sequential order over a bus.

start bit data bits stop bits ⇒ To know the start and end of transmission.

Parallel Transmission

In parallel transmission several bits can be transmitted at the same time.



Serial Communication can be
- Simplex
- Half Duplex
- Full Duplex

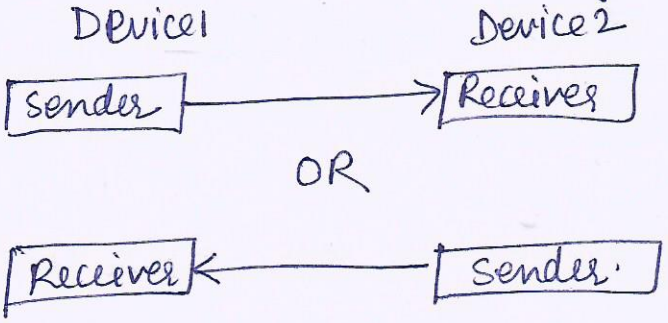
Simplex Transmission: Signals travels in one direction only.
eg. Keyboard

Downloaded by Sangam gupta (sangamg7318@gmail.com)

Half Duplex:

→ capable of sending signals in both the directions, but in only one direction at a time

eg Walkie Talkie.



Full Duplex: (Bidirectional Transmission)

→ This method allows signal transmission in both the directions simultaneously.

eg Telephone IP service.

This document is available free of charge on studocu

Downloaded by Sangam gupta (sangamg7318@gmail.com)