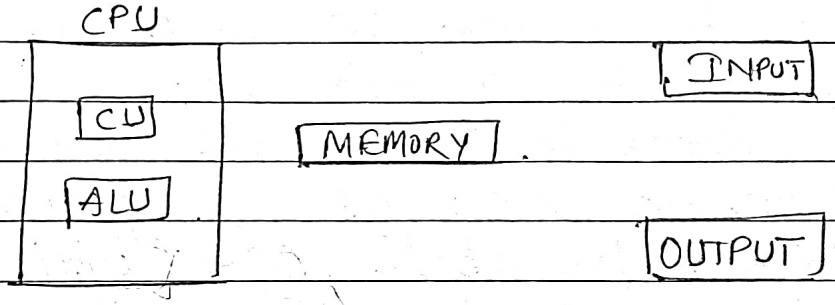
UNIT-1

Camlin Page
Date / /

BASIC STRUCTURE OF COMPUTER

Functional Unit :-



Computer is an eleltronic device that can accept, store and process data under the control of a set of instructions. The instructions called a program are stored in the computer when in use, so they can be executed automatically.

CPU :- The most fundamental part of the mother board is the microprocessor chip or the Central Processing Unit.

It consists of mainly an integrated circuit of semiconductor devices and other electronic components manufactured on a thin sheet of semiconductor material.

CPU executes and store program instructions that process or manipulate data into information. The CPU is often referred as the brain of the computer.

Input-Output Devices :- Input refers to data entered into a computer for processing for example - from a keyboard or from a file stored on disc.

Input includes program instruction that the CPU receives after commands are issued by the

users.

Output refers to the results of processing i.e., information sent to the screen or the printer or to be stored on disc or sent to another computer in a network.

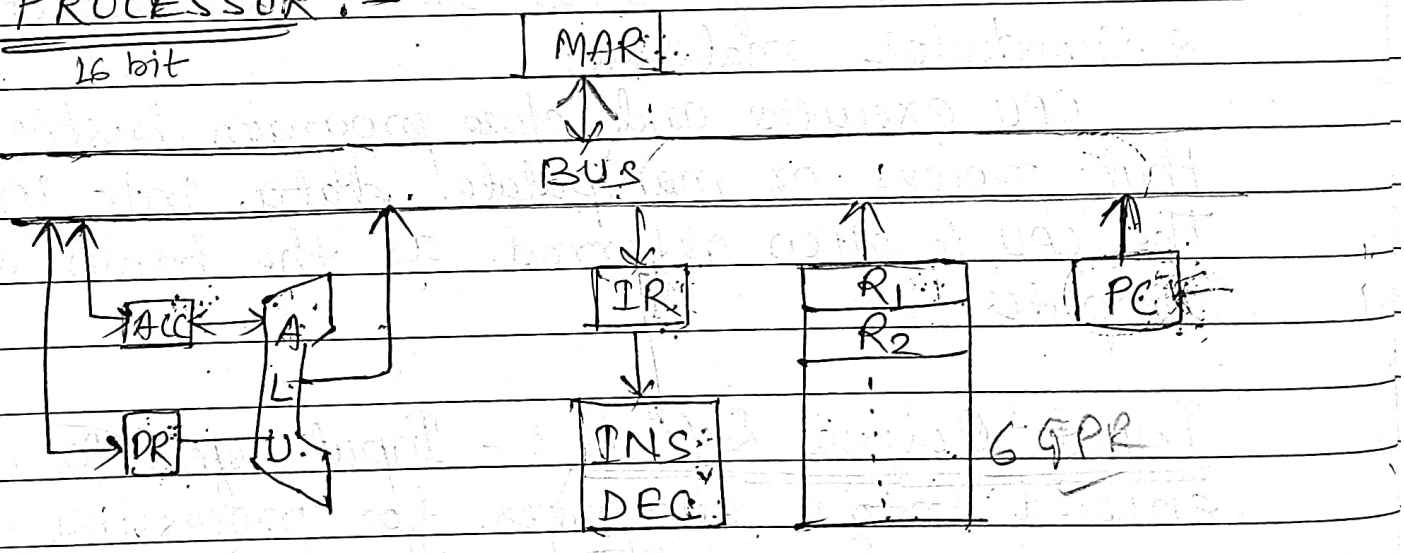
Some input devices are keyboard, mouse, lightpen, touch screens, graphics tablets (digitizer), joystick, trackball, OCR (optical card reader), smart card reader, Bar code reader, web camera, digital camera, scanner, biometric sensor.

Some output devices are monitors, printers, speaker, LCD projectors.

Within the motherboard in the system cabinet, the memory chips are installed and secondary storage devices are connected to it. The memory chips deal only with temporary storage secondary or permanent storage stores your data as long as you want.

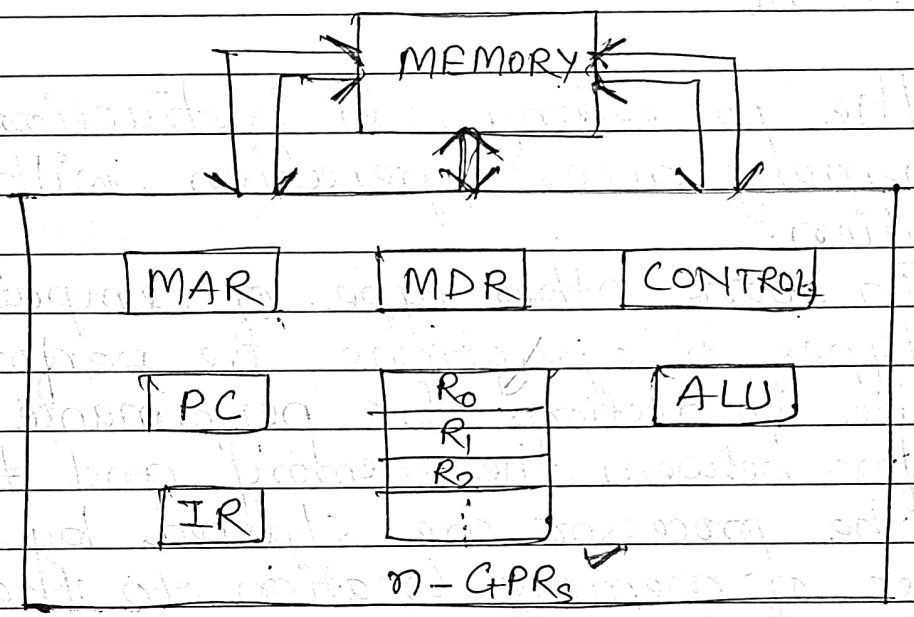
PROCESSOR :-

16 bit



MAR – Memory Address Register
ACC – Accumulator (Special Purpose Register)
DR – Data Register
ALU – Arithmetic Logical Unit
IR – Instruction Register
INS.DEC – Instruction Decoder
R_1, R_2 – Register 1, 2 (General Purpose Register)
PC – Program Counter.

BASIC OPERATIONAL CONCEPT



To perform a given task an appropriate program consisting of list of instructions is stored in the memory.

Individual instructions are brought from the memory into the processor which executes the specific operations.

Add LOCA, R_0

This instruction add the operand at memory location (LOCA) to operant in Register R_0 and

places the sum into the register R0. This instruction requires the performance of several steps —

1) The instruction is fetched from the memory into the processor.

2) The operand at LOCA is fetched and added to the content of R0.

3) Finally, the resulting sum is stored in the register R0.

The preceeding add instruction combines a memory access operation with an ALU operation.

In some other type of computer, these two types of operations are performed by separate instructions for performance reason. Transfer between the memory and ~~the memory~~ ~~and~~ the processor are started by sending the address of memory location to the unit and issuing the appropriate control signal.

IR — It holds the instruction currently being executed. Its output is available for the control circuit which generate the timing signal.

PC — Program counter is another special register that keeps tracks of execution of a program. It contains memory address of the next instructions to be fetched.

Date / /

MAR :- Memory Address Register holds the address of the location to be accessed.

MDR :- Memory Data Register contains the operand used for arithmetic operation.

Operating Steps – (Execution of Complete Instruction)

1) Programs reside in the memory and usually we get this from the input unit.

2) Execution of the program starts when the PC hold the address of the 1st instruction of the program.

3) Content of PC are transferred to MAR and the read control signal is sent to the memory.

4) After the time required to access the memory elapsed the word is read out of memory and loaded into MDR.

5) Now content of MDR are transferred to IR and instruction is ready to decode and execute.

6) If the instruction involves ALU operation it is necessary to obtain the required operand.

7) An operand in the memory is fetched by sending its address to MDR by initiating a read signal.

8) When the operand has been read from the memory to the MDR, it is transferred from MDR to ALU. After one or two such repeated cycle, ALU can perform the desired operation.

9) If the result of this operation is to be stored in the memory, the result is send to the MDR.

10) Address of location where the result is to be stored is sent to MAR and write signal is initiated.

11) The content of PC are incremented so that PC points to the next instruction that is to be executed.

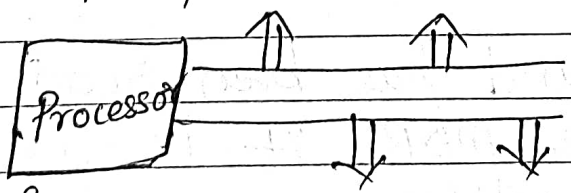
Normal execution of program may be interrupted if some device require urgent service. To do this, the device generate the interrupt signal.

An interupt signal is a request signal from an I/O device for serving the I/O processor. The processor provides the requested service by executing an appropriate interupt service routine.

BUS STRUCTURE:—

Group of wires which carry information from CPU to peripherals or vice versa. There are two types of BUS structure.

1) Single Bus — Common Bus is used communicate between pesipheral and microprocessor.

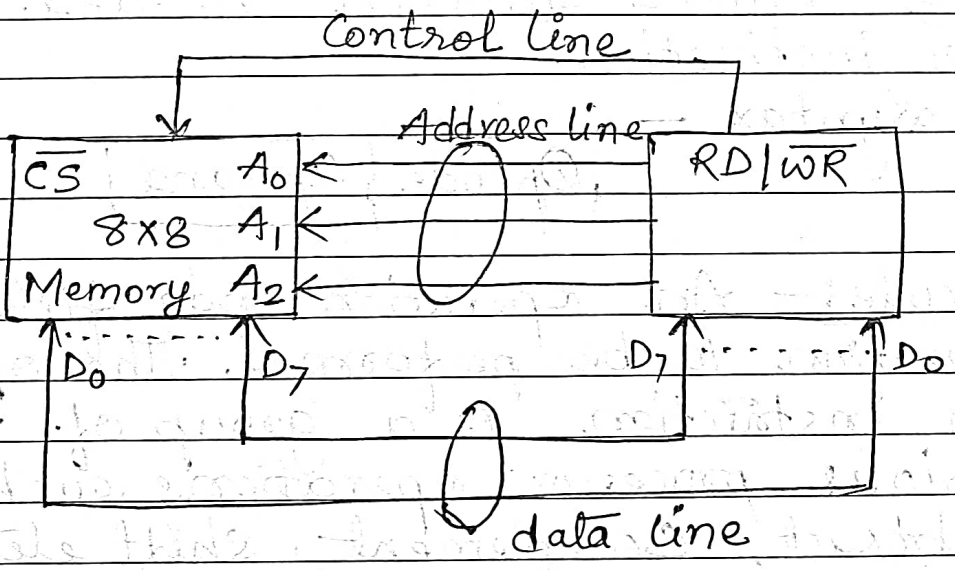


To improve performance multibus structure may be used: for example - In 2 bus

Camlin Page
Date / /

structure, 1 bus can be used to fetch instruction other can used to fetch data required for execution.

Thus, improving the performance but cost is also increased.



2) Address Bus – It is unidirectional group of wires which carries address information bits from processor to peripherals.

3) Data Bus – It is bidirectional group of wires which carries data information bits from processor to peripherals and vice versa.

4) Control Bus – It is bidirectional group of wires which carries control signals from processor to peripherals and vice versa.

Instructions –

1) MOV R_1, R_2
2) MOV R_2, R_1
3) MOV ACC, R_1
4) MOV ACC, R_2
5) MOV ACC, M
6) MOV M, ACC

Arithmetic – ADD
CMP
Logical – AND ⇒ NAND

Date / /

INSTRUCTION FORMAT :-

A computer will usually have a variety of instruction code format. It is the function of control unit within the CPU to interpret each instruction code and provide the necessary control function needed to process the instruction.

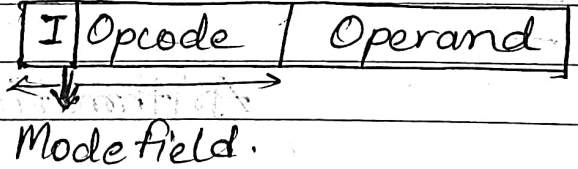
Syntax — RAM

Opcode	Operand

1) Opcode :- An operation code field that specifies operation to be performed. The opcode field of an instruction is a group of that define various processor operations such as add, subtract, complement, shift etc.

2) Operand Address :- An address field that designates memory address or processor register. Operations specified by computer instructions are executed on some data stored in memory or processor registers.

Operands resides in memory are specified by their memory address. Operands residing in processor register are specified with a register address.



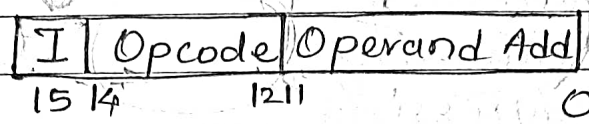
3) Mode field :-

The mode field of an instruction specify a variety of alternatives for choosing the operand from given address. The basic computes has

three instruction code format.

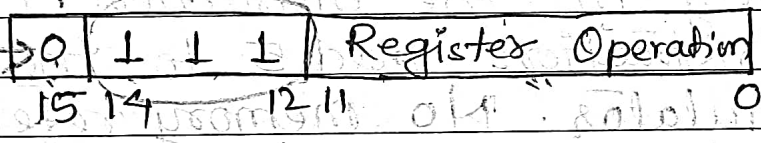
1) Memory Reference
2) Register Reference
3) I/O Instructions.

Memory Reference —



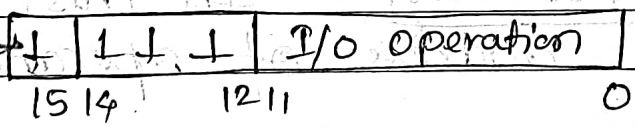
In memory reference instruction, if I = 0 then it is direct addressing and if I = 1 then it is indirect addressing.

Register Reference —



An operand from memory is not needed therefore 12 bits are used to specify operation or test to be executed.

I/O Instructions —



An I/O instruction does not need reference to memory and is recognised by operation code with a one in the left most bit of an instruction. The remaining 12 bits are used to specify the type of I/O operation or test performed.

Camlin | Page
Date / /

ADDRESSING MODES – The different ways in which location of operand is specified in an instruction are referred to as addressing mode.

1) Immediate Addressing
2) Direct Addressing
3) Indirect Addressing
4) Register Addressing
5) Register Indirect Addressing
6) Relative Addressing
7) Indexed Addressing.

① Immediate Addressing - Operand is given explicitly in the instruction. for example Add 5. This instruction add 5 to the content of accumulator. No memory reference to fetch data. So, it is fast.
But in limited range

format – | Opcode | Operand |

② Direct Addressing – In this, address field contains address of operand. Ex - ADD A

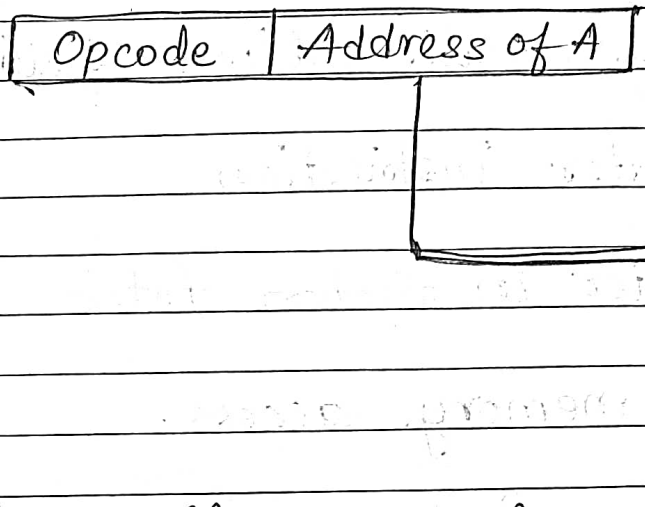
Effective Address (EA) = Address Field of A.
This instruction (ADD A) add contents of A to the accumulator.

Look in memory at address A at for operand.

Single memory reference to access data. No additional calculation require to calculate

effective address.

format –



(3) Indirect Addressing – Memory cell pointed to by address field contains address of operand.

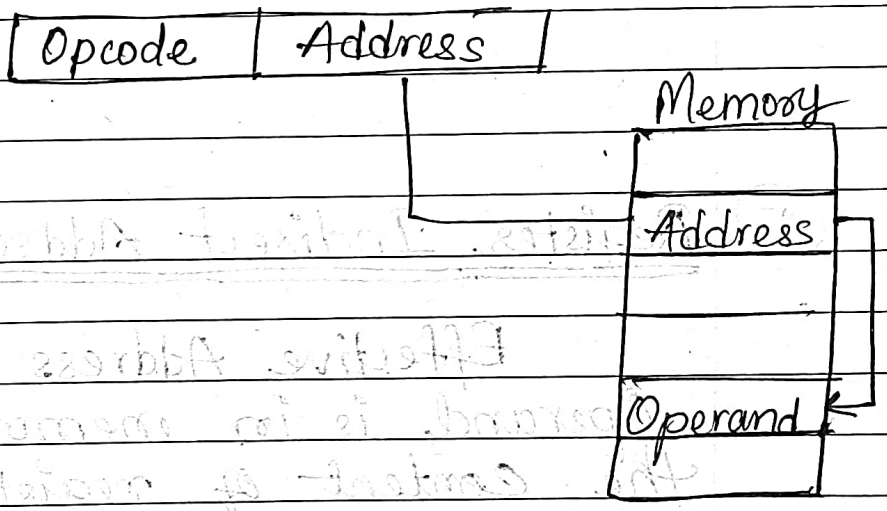
Effective Address (EA) = Address field of A

$EA = [A]$

Look in A, find address of A, Look there for operand.

Eg. ADD A

format –



Large address space require Multiple memory access to find operand. Hence, it is slower.

(4) Register Addressing – Operand is held in register named in address field.

Effective Address = R

$EA = R$

Limited no. of registers.

Very small address field needed

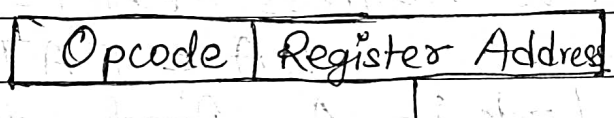
Shorter instruction

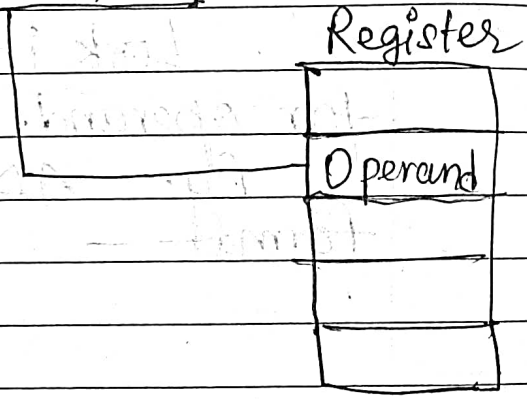
Faster instruction fetch

No memory access.

Very fast execution.

Multiple register help performance.

format – 



(5) Register Indirect Addressing –

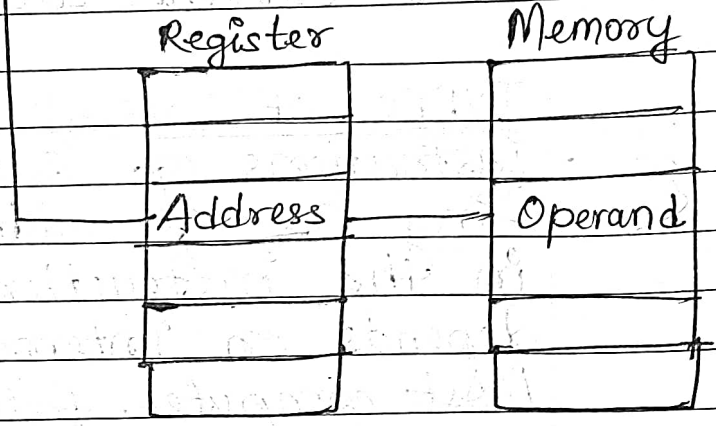
Effective Address = R,
Operand is in memory cell pointed to by the content of register R.

Large address space.

fewer memory access than direct addressing.

~~format~~ –

Format - | Opcode | Register Add. |



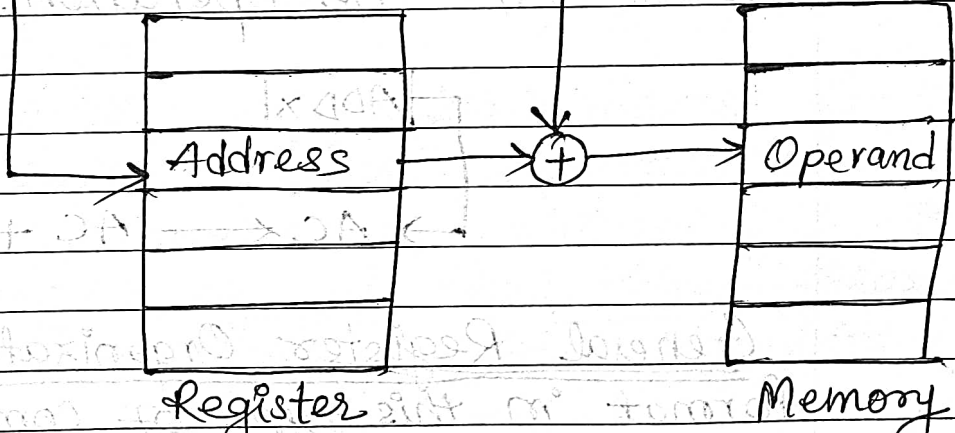
(6) Indexed Addressing :-

Effective Address = X + [R]

Address field holds two value i.e,
X = constant value (offset)
R = Register that holds address of memory location.

Format -

| Opcode | Register R | Const. value |



(7) Relative Addressing :- It is a version of displacement address.

where X is Address part of the instruction

Effective Address = X + P.C (PC)

[R = Program Counter]

Get the operand from X byte away from current location pointed to by PC.

TYPES OF INSTRUCTIONS — Computers may have instructions of several different containing varying no of addresses. The no. of address field in the instruction format of a computer depends on internal organizations of its registers. Most computers fall into one of 3 types of CPU organization —

1) Single Accumulator Organization.
2) General Register Organization
3) Stack Organization.

Single Accumulator Organization — In this type of organization that specifies the arithmetic addition is defined by an assembly language instruction as ADD X, where X is address of operand. The ADD instruction in this case, result in the operation.

ADD X

→ AC ← AC + M(X)

General Register Organization — The instruction format in this type of computer needs 3 register address field. Thus, instruction for an arithmetic addition may be written as,

ADD R_1, R_2, R_3

→ A $R_1 \leftarrow R_2 + R_3$

Camlin Page
Date / /

Stack Organization :- The stack organization CPU uses stack which uses PUSH and POP which requires the address field.

for-ex. — PUSH X

for the above organization, there are 4 types of instructions —

1) Three Address Instruction :- Computers with three address instruction format can use each address field to specify either a processor register or a memory operand.

The program in assembly language that evaluate $X = (A+B) * (C+D)$ as follows-

ADD R_1, A, B $\Rightarrow$ $R_1 \leftarrow M[A] + M[B]$
ADD R_2, C, D $\Rightarrow$ $R_2 \leftarrow M[C] + M[D]$
MUL X, R_1, R_2 $\Rightarrow$ $M[X] \leftarrow R_1 * R_2$

It is assumed that the computer has two processor register. The symbol M[A] denotes the operand at memory Address symbolised by A.

The advantage of three address format is that it results in short program when evaluating arithmetic operations.

The disadvantage is that the binary coded instruction require to many bits to specify three address.

2) Two Address Instruction :- Two address instruction are most common in commercial system. Here again, each address field can specify either the processor register or a memory word.

Date / /

The program to evaluate $X = (A+B) * (C+D)$ are —

```
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
```

(III) One Address Instructions — One Address Instruction use an accumulator register for all data manipulation. for multiplication and division, there is need for second register and assume that the AC contains the result of all operation. The program to evaluate $X = (A+B) * (C+D)$ in one Address Instruction —

```
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
```

(IV) Zero Address Instructions — Step organized computers donot use an address field for the instructions addition and multiplication.

The PUSH and POP instructions however need an address field to specify the operand that communicate with the stack.

The program to evaluate the same operation in

AC

①	2	3	4
000f	0010	0011	0100
0010	0100	0110	1000
01			0000

Camlin | Page
Date / /

zero address Instructions —

PUSH A	TOS ← A
PUSH B	TOS ← B
ADD	TOS ← A+B
PUSH C	TOS ← C
PUSH D	TOS ← D
ADD	TOS ← C+D
MUL	TOS ← (C+D)(A+B)
POP X	M[X] ← TOS

SUBROUTINE — A set of instr which are used repeatedly in a prog.

16 bit

```
        ORG 100
100     LDA  X
101     BSA  SH4
102     STA  X
103     LDA  Y
104     BSA  SH4
105     STA  Y
106     HLT
107     X,  HEX 1234
108     Y,  HEX 4321
109 SH4, HEX 0
10A     CIL
10B     CIL
10C     CIL
10D     CIL
10E     AND  MSK
10F     BUN  SH4 I
110     MSK, HEX FFF0
        END
```

Preprocessor → a.I — expanded code
Compiler → a.asm
Assembler → a.obj
Linker / Loader → a.exe

107 X = 1234 } after op
108 Y = 4321 }

109 → SH4, HEX 0 ← 102, 105

10A–10C implied mode

10D: 0010 0011 0100 XXXX
1111 1111 1111 0000
10E: 0010 0011 0100 0000

LDA — Load In Accumulator
BSA — Branch and save the return Address
MSK — label

STA X - Store the value of Accumulator at the address of X.

HLT - Halt

CIL - Circular left (left shift).

BUN - Branch UN conditionally

for ex. 0001 0010 0011 0100
1 2 3 4

After 4 CIL - 0010 0011 0100 XXXX

& AND operation

HEX 1111 1111 1111 0000
1110 F F F 0

X- 0010 0011 0100 0000

110
010
110

PARAMETER PASSING

```
        ORG 200
200     LDA X       AC = 110
201     BSA OR
202     HEX 3AF6    Suppose = 010
203     STA Y       Y = 110. Ans
204     HLT
205     X, HEX 3AF6 → Suppose 110
206     Y, HEX 0
207     OR, HEX 0   202 ⇒ 203
208     CMA         AC = 001
209     STA TMP     TMP = 001
20A     LDA ORI     AC = 010
20B     CMA         101
20C     AND TMP     001
20D     CMA → 110 ← 001
20E     ISZ OR
20F     BUN OR I
```

16 bit n + k

4 bit opcode 12 bit operand address

Camlin | Page
Date / /

```
    210  TMP ;HEX 0 001
         END
```

Program for logic OR operation.

CMA – Complement the value of Accumulator.

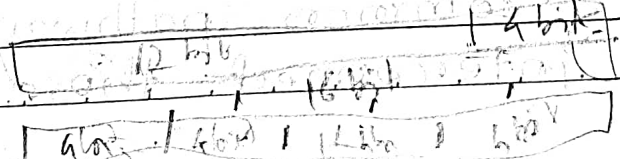
ISZ – Increment M and skip if zero (M – Memory).

EXPANDING OPCODE METHOD :–

Consider an (n+k) bit instruction with a k bit opcode and n bit address. Then, the instruction allows 2^k operations and 2^n addressable memory cells.

Alternatively the same (n+k) bits could be booken up into (k-1) bit opcode and an (n+1) bit address meaning half as many instructions and either twice as much address-able memory or the same amount of memory with twice the resolution.

Ex – ① Consider a machine in which instructions are 16 bit long and addresses are 4 bit long. This might be reasonable on a machine that has 16 registors on which all arithmatic operation take place. One design will be a 4 bit opcode and 3 address in each instructions giving 16 three address instructions.



Q. If designer needs 15 - 3 address instruction 14 - 2 address instruction and 31 - 1 address instruction 16 inst with no address at all opcode / operand

0000	xxxx	yyyy	zzzz	15-3 address
⋮				
1110	xxxx	yyyy	zzzz	
1111	0000	yyyy	zzzz	14-2 address
⋮				
1111	1101	yyyy	zzzz	
1111	1110	0000	zzzz	31-1 address
⋮				
1111	1110	1111	zzzz	
1111	1111	0000 (16)	zzzz	
⋮				
1111	1111	1110	zzzz	
1111	1111	1111	0000	16 - no addr
⋮				
1111	1111	1111	1111	

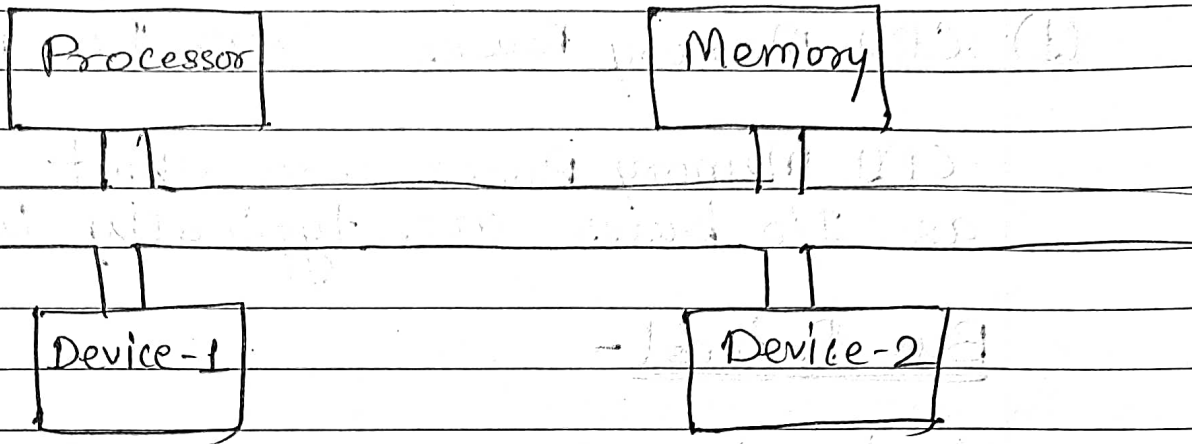
8421

BUS ARCHITECTURE

A typical computer system is composed of several components such as CPU, memory chip and I/o devices.

A Bus is a common pathway of a set of wires that interconnect those various sub

systems. The Bus thus allows different components to communicate with each other.



A Bus in computer language is a channel over which information flows between units or devices. It typically have access points through which the device can use the channel.

Most buses are bidirectional through which devices can sent/receive information.

However, too many devices are connected to the same bus, the bandwidth of the bus can become a bottle-neck problem.

Buses often include wires to carry signals for addresses, data, control, status block etc.

The communication need of different devices in a computer system vary for instance - High bandwidth communication is needed between processor and memory whereas bandwidth requirement on buses to I/O devices is low. This has led to the creation of different kind of buses differing in bandwidth, width and

latency and capability. Typically there are two kind of buses.

(1) CPU Memory Buses (2) I/O Buses.

CPU Memory Buses are short and fast where as I/O buses are typically long and slow.

Bus Protocol-

A bus is a communication channel shared by many devices and hence rules needed to be establish in order to the communication to happen correctly. These rules are called bus protocol.

Design of a bus architecture involves several trade-offs related to the width, data bus, data transfer size, bus protocol, clock-in etc.

Depending on whether the bus transaction are controlled by a clock or not, buses are classified into synchronous and Asynchronous buses.

Depending on whether the data bits are send on parallel wires or multiplex onto one single wire. These are called Parallel and serial Buses.

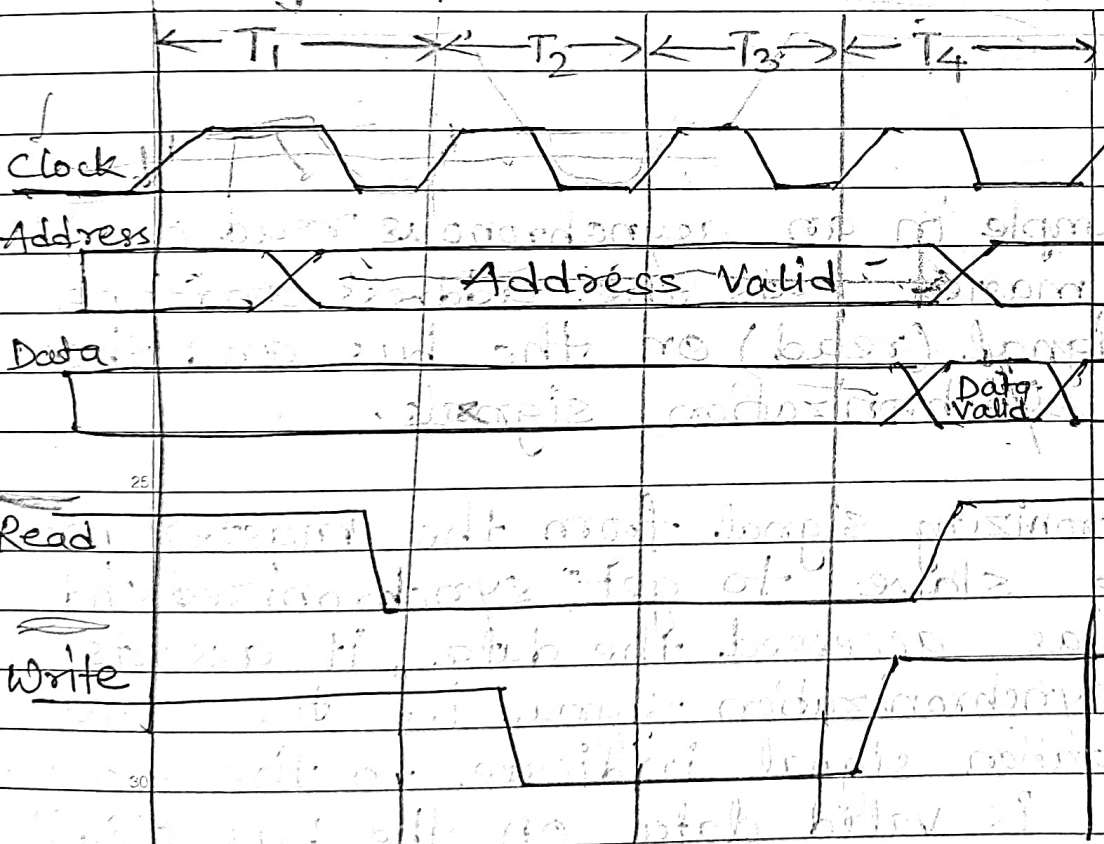
Clock

Camlin | Page
Date / /

SYNCHRONUS AND ASYNCHRONOUS BUSES :-

In a synchronous bus, bus operations are synchronized with reference to a clock signal. The bus clock is generally derived from the computer system clock. However, it is slower than the master clock.

Buses are traditionally slower than processor because memory access times are typically longer than processor clock cycles. A bus transaction often takes several clock cycles although the cycles are collectively referred to by many as a bus cycle.

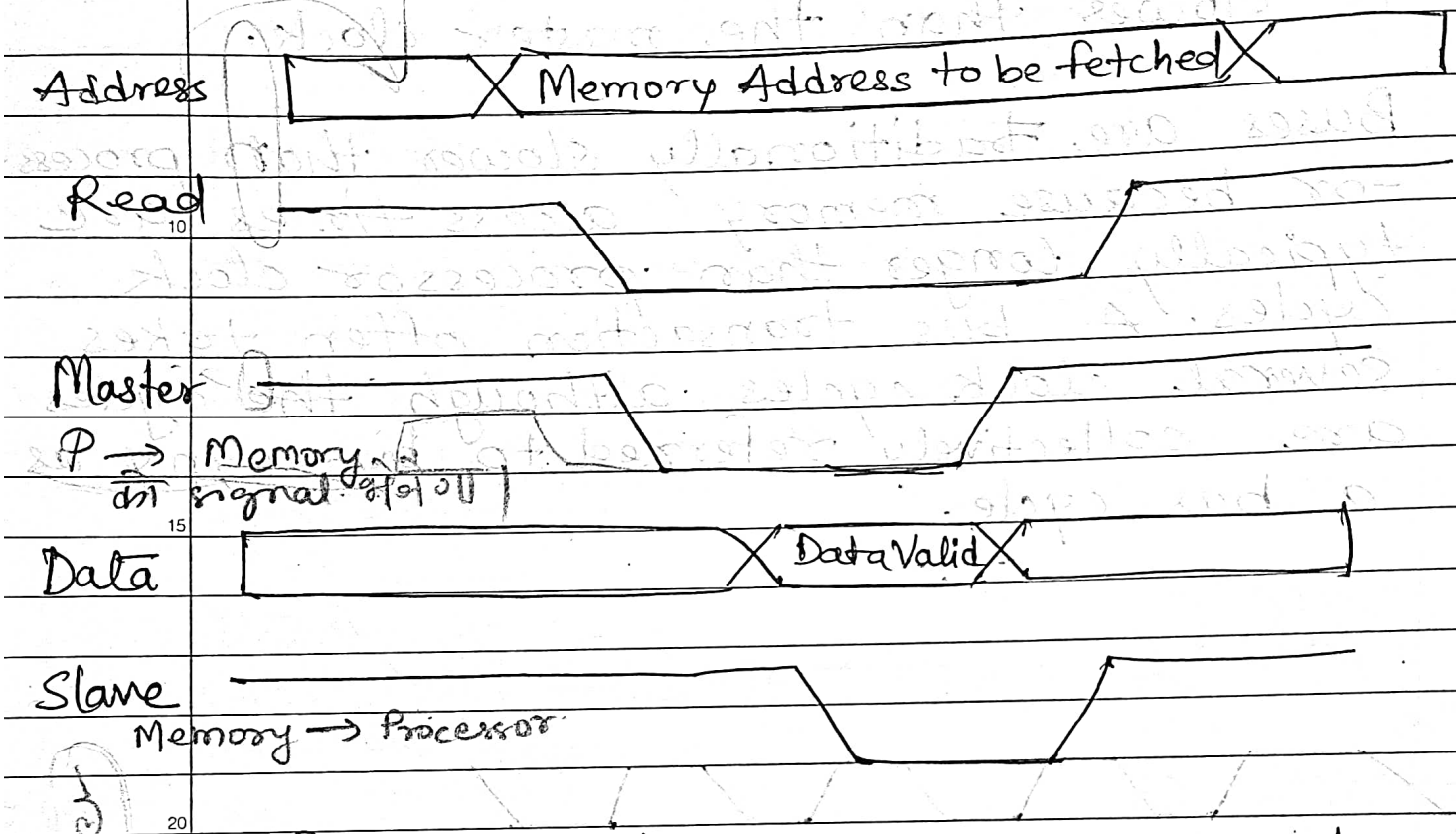


Read Operation On a Synchronous Bus

Handshaking — when one device sends a msg to another device indicating that it wants to establish a comm channel. The process by which two devices initiate communicati

Asynchronous Bus

An asynchronous bus has no system clock. Handshaking is done to properly conduct the transmission of data between sender and receiver.



for example in an asynchronous read operation the bus master puts the address and one control signal (read) on the bus and then asserts a synchronization signals.

The synchronizing signal from the master prom-pts the slave to get synchronized and once it has accessed the data, it asserts its own synchronization signal i.e, the slave synchronization signal indicate to the processor that there is valid data on the bus and it reads the data. The master then deasserts its synchronization signal which indicates to

the slave that the master has read the data. The slave then deasserts its synchronization signal. This method of synchronization is referred to as full handshake.

Serial And Parallel Bus :- Bus that transfer several data bits at the same time are called parallel buses. It is desirable to have wide buses because large chunks can be transfered quickly when multiple lines can be used.

Parallel buses typical have 8, 16, 32 or 64 data lines.

Serial bus use the same line to transfer different data bits of the same byte or word. Typically they have only one data line and the bits are send one after another as a packet.

The universal standard bus (USB) and IEE1394 are bus architecture are example of serial bus.

Serial buses are less expensive than parallel buses however parallel buses have higher through-put.

The execution of program complete in 3 steps —

1) fetch
2) Decode ⟶ Operand fetch
3) Execution

ADD R_1

T_0 : MAR ← PC

T_1 : IR ← M[MAR]; PC ← PC + 1;

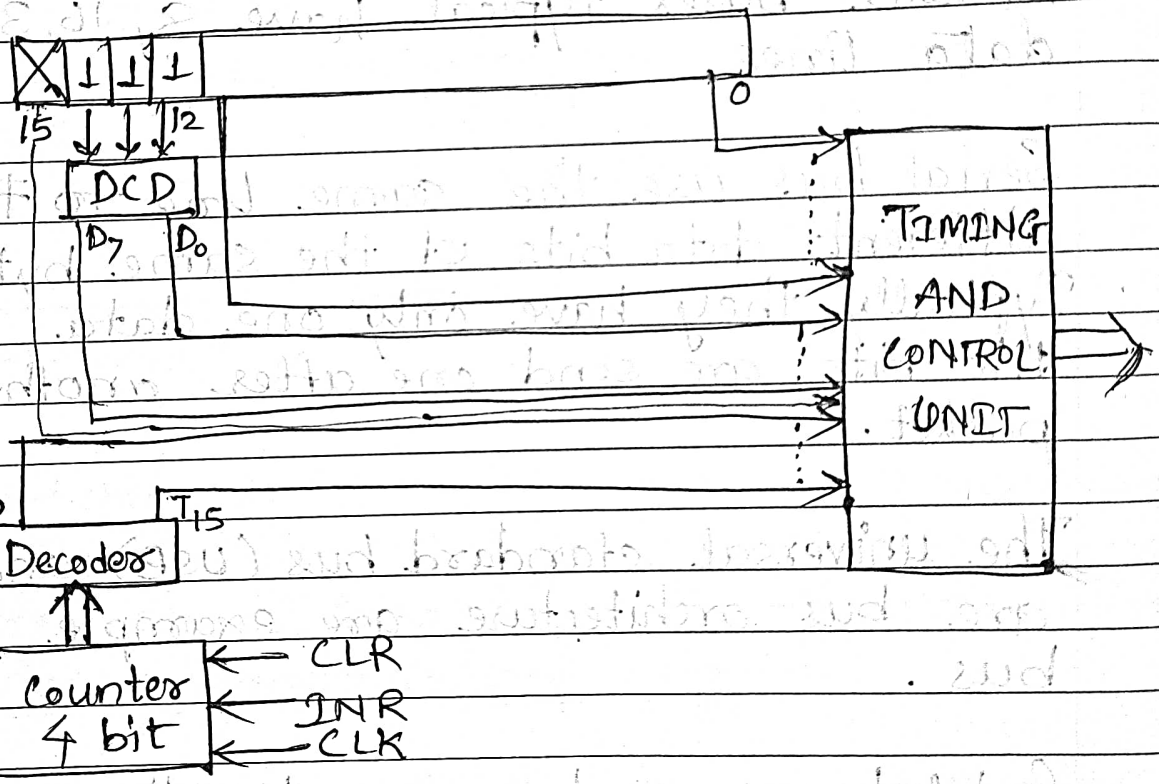
T_2 : DCD(IR); MAR ← IR_{0-11} if ALU op

T_3 : DR ← R_1

T_4 : AC ← ALU_{ADD} (AC, DR)

The ADD is an ALU operation so it requires to transfer the value of R_1 to DR for successfully execution of the desired operation.

TIMING AND CONTROL UNIT (Hardwired Control Unit)



for Register Instructions

Ring counter for Timing seq. ($T_0 - T_4$)

① Shift reg

② Dec with counter

I_0 – ADD R_1

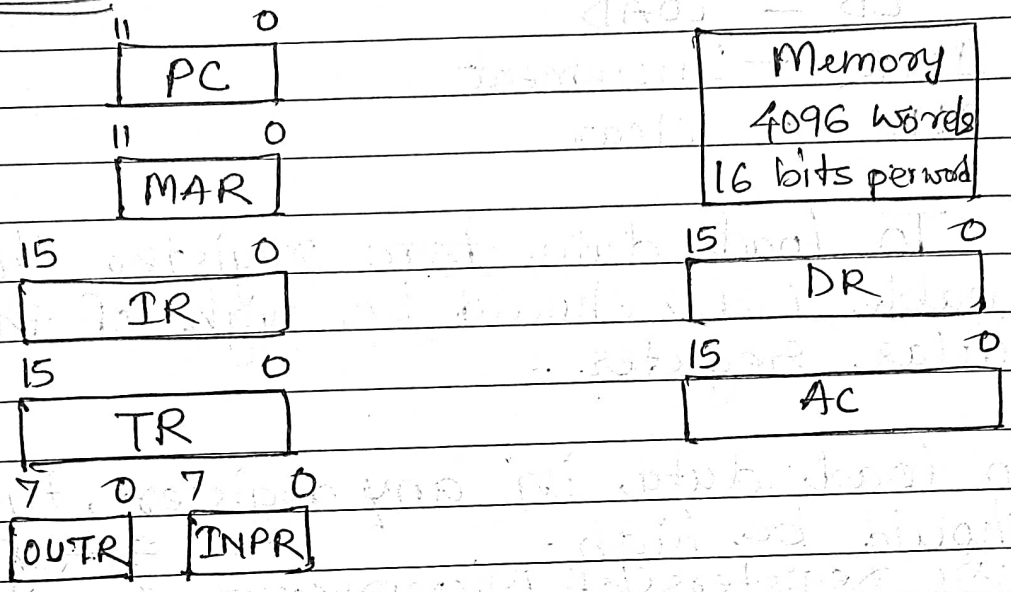
I_1 – CMP

I_2 – AND R_1

I_3 – MOV R_1, R_2

Camlin | Page
Date / /

Registers



Register Symbol	No. of bits	Register Name	function
DR	16	Data Register	Holds memory Operand
AR	12	Address Register	Holds address for memory
AC	16	Accumulator	Processor Register
IR	16	Instruction Register	Holds Instruction code
PC	12	Program Counter	Holds address of instruction
TR	16	Temporary Register	Holds temporary data
INPR	8	Input Register	Holds input character
OUTR	8	Output Register	Holds output character.

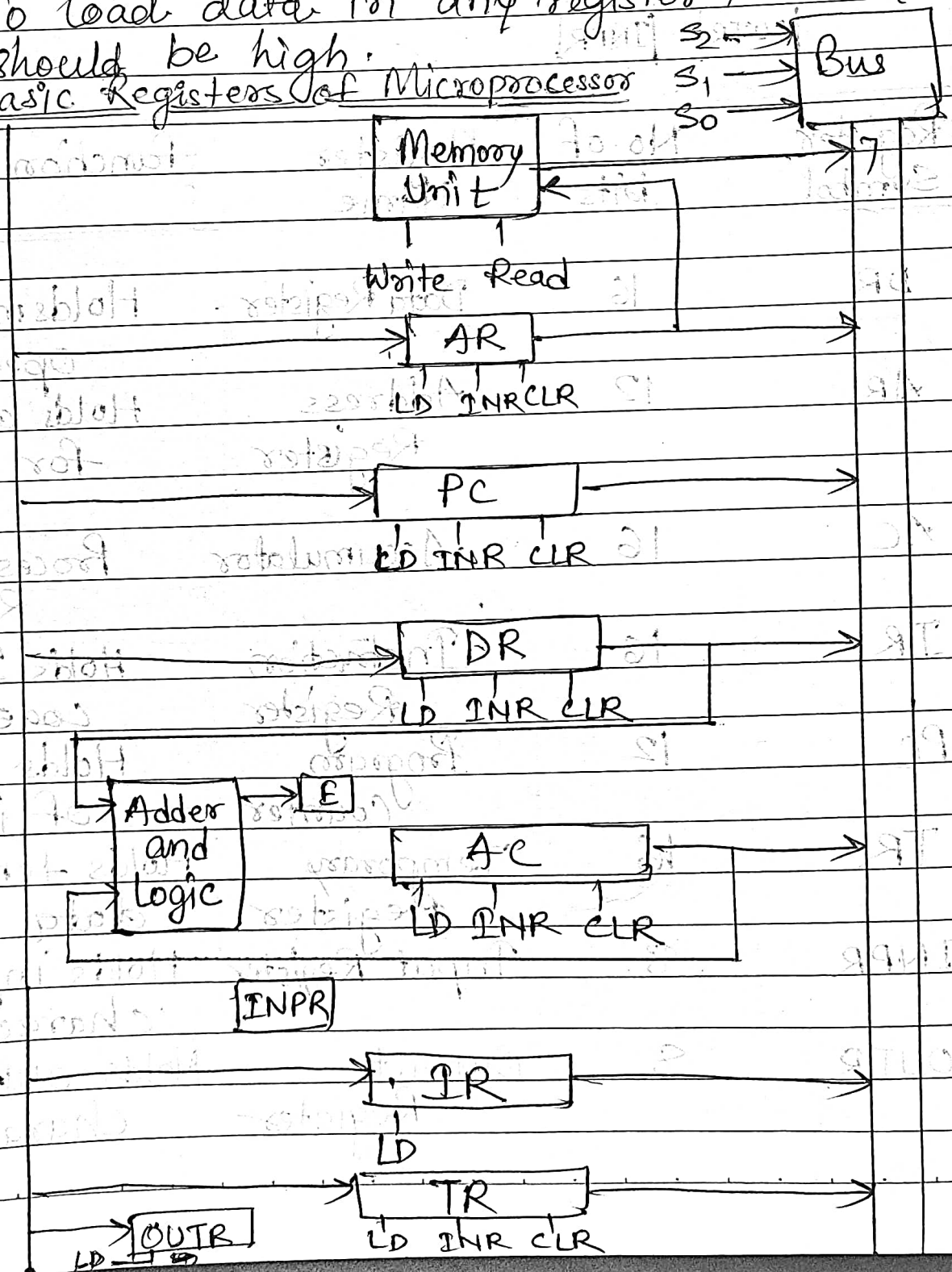
Abbreviation -

1) LD — LOAD
2) INR — Increment
3) CLR — clear

To load data from register, the output Enable (OE) should be high of that particular register.

To load data in any register, the LD (Load) should be high.

Basic Registers of Microprocessor



Camlin Page

Timing signals – Control Signal of ADD R1

$MAR \leftarrow PC$

$PC : OE = T_0$

$MAR : LD = T_0 + T_2$

$IR \leftarrow M[MAR] ; PC \leftarrow PC + 1$

$IR : LD = T_1$

$PC : INR = T_1$

RD ⟶ Read signal that reads data from M[MAR]

$DCD(IR) ; MAR \leftarrow IR_{0-11}$

Combinational Circuit

$IR : OE = T_2$

$MAR : LD = T_2$

$DR \leftarrow IR_1$

$R_1 : OE = T_3$

$DR : LD = T_3$

$AC \leftarrow ALU_{ADD}(AC, DR)$

$AC : LD = T_4 D_7 I_0$

$ALU : ADD = T_4 D_7 I_0$

MICROPROGRAMMED CONTROL UNIT –

Fetch ← MAR ← PC

Decode ← IR ← M[MAR] ; PC ← PC+1

Execute ← DCD(IR) ; MAR ← IR_{0-11}

T_0 : OE_{PC}, LD_{MAR}

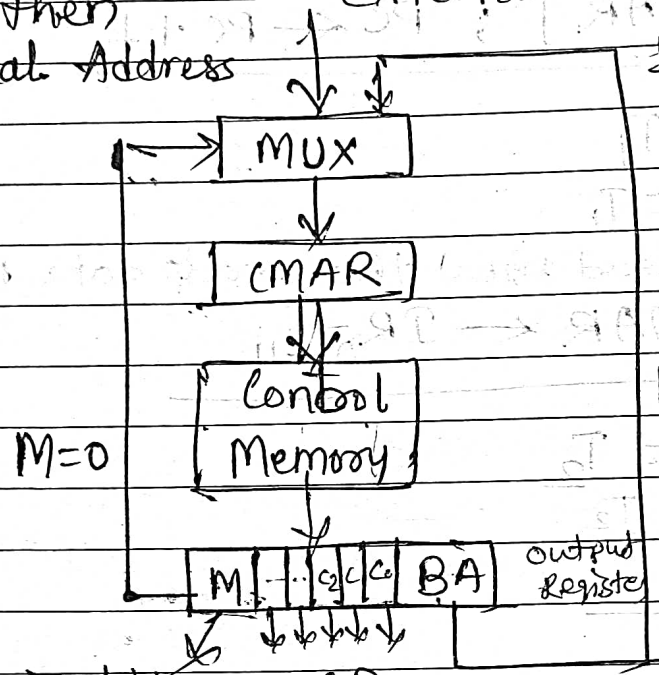
T_1 : LD_{IR}, RD_M, INR_{PC}

T_2 : LD_{MAR}, OE_{IR}

If M=0 then External Address

External Address Generator (M=0)

If M=1 then Branch add



Selection line (M)

M=0

M=1 Branch Address (BA)

M - Mode field (selection line of MUX)

7 bit field

CF

Control field

Control Signal for mov R_1, R_2

C_0 — OE_{PC}

C_1 — LD_{MAR}

C_2 — LD_{IR}

C_3 — RD_M

C_4 — INR_{PC}

C_5 — OE_{IR}

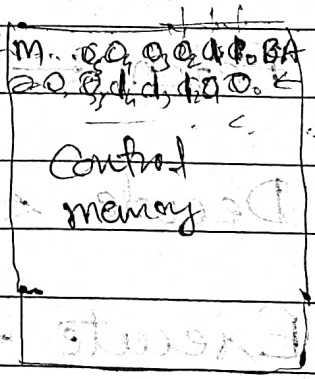
C_6 — OE_{R_2}

C_7 — LD_{R_1}

$R_2 \leftarrow R_1$

OE_{R_2}

LD_{R_1}



Camlin | Page
Date / /

MOV R_1, R_2

Address select field

	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0	BA	M
L →	1	1	0	0	0	0	0	0	00	1

Here this is decimal number

ADD R_1 C_8 C_9

$DR \leftarrow R_1 \Rightarrow OE_{R_1}, LD_{DR}$

$AC \leftarrow ALU_{ADD}(AC, DR) \Rightarrow LD_{AC}, ALU_{OE}, ALU_{ADD}$ C_{10} C_{11} C_{12}

	C_{12}	C_{11}	C_{10}	C_9	C_8	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0	BA	M
L	0	0	0	1	1	0	0	0	0	0	0	0	0	11	1
	1	1	1	0	0	0	0	0	0	0	0	0	0	00	1

V.IMP **Microinstruction format** – Each step in a seq of steps in the execution of a certain m/c instn.

ADD, R_1, X

3	3	3	2	2	7
f_1	f_2	f_3	CD	BR	AD
I			II	III	IV

= 20 bit

f_1, f_2, f_3 : Microinstruction fields – 21 micro operation

CD : Condition for braching

BR : Branch field

AD : Address field

f_1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC → clear AC.
011	$AC \leftarrow AC + 1$	INCAC → Incre. AC.
100	$AC \leftarrow DR$	DRTAC → Data Reg to AC (D)
101	$AR \leftarrow DR(0-10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR → PC to AD
111	$M[AR] \leftarrow DR$	WRITE → DR की value add reg में store होना चाहिए

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

F3	Microoperation	Symbol
000	None	NOP
001	AC ← AC ⊕ DR	XOR
010	AC ← $\overline{AC}$	COM complem
011	AC ← shlAC	SHL
100	AC ← shrAC	SHR
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	Reserved.	

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

BR	Symbol	function
00	JMP	CAR ← AD if cond^n = 1
		CAR ← CAR+1 if cond^n =

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

The microoperations are sub divided into 3 fields of 3 bits each. The 3 bits in each field are encoded to specify seven distinct micro-operations as listed above.

This gives a total of 21 microoperations. No more than 3 microoperations can be chosen for a microinstruction one from each field.

If fewer than 3 microoperations are used one or more of the fields will use the binary code '000'. for example - A microinstruction can specify 2 simultaneous microopperations from f_2 and f_3 and none from f_1.

Eg - $DR \leftarrow M[AR] \longrightarrow f_2$

$PC \leftarrow PC+1 \longrightarrow f_3$

$f_1 \quad f_2 \quad f_3$

∴ Code - 000 100 101

It is important to realise that 2 or more conflicting microopperations cannot be specified simultaneously.

Camlin | Page
Date / /

Eg - 010 001 000
f1 f2 f3
AC ← 0 AC ← AC - DR

CD (Condition) - The condition field consists of 2 bit which are encoded to specify 4 status bit conditions as listed above.
(to select status bit cond^n)

The 1st condition is always 1 so that a reference to CD=00 or the symbol 'U' will always find the condition to be true.

When the condition is used to conjunction with BR (Branch field) it provides an unconditional branch operation.

The indirect bit 'I' is available from bit 15 of DR after an instruction read from memory. The sign bit of AC provides the next status bit. The zero value symbolized by 'z' is a binary variable whose value = 1 if all the bits in AC are equal = zero (0).

(specify the type of Branch)
Branch field - The BR field consists of 2 bits. It is used in conjunction with the address field (AD) to choose the address of the next microinstruction.
for example - When BR=00 the control performs a (JMP) jump operation. When BR=01 it performs a call to subroutine (CALL operation). The return from subroutine is accomplished with a BR field =10. This causes the transfer of the return address from

SBR to CAR. The mapping from the operation code bits of the instruction to an address for CAR is accomplished when BR field is equal 11.

CONDITION

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional Branch
01	DR(15) 15th bit	I	Indirect Add^{r}. Bit
10	AC(15) 15th bit	S	Sign bit of AC
11	AC = 0	Z	Zero value in AC.

Branch field

BR	symbol	functions.
00	JMP	CAR ← AD if cond = 1
		CAR ← CAR+1 if cond = 0
01	CALL	CAR ← AD, SBR ← CAR+1
	call the subroutine	if cond^{n} = 1
		CAR ← CAR + 1 if cond = 0.
10	RET	CAR ← SBR.
	Return	(return from subroutine)
11	MAP	CAR(2,5) ← DR(11-14),

Address field :- next instn to be executed

CAR(0,1,6) ←

7 bit $- 2^7 = 128$ bits.

Bit Slice Concept :- It is a tech for constructing a processor from modules of processors of smaller bit width, for the purpose of increasing the word length; in theory to make an arbitrary n-bit CPU. Each of these component modules processes one bit field or "slice" of an operand.