Date: / /

If $A = (-12)_{10}$ & $B = (+10)_{10}$. Perform A−B, A+B, B−A using 2's.

Sol^n:

$A = (+12)_{10} =$ 00001100

$B = (+10)_{10} =$ 00001010

A−B

$A = (-12)_{10} =$ 00001100 → 11110011 → 11110100

$B = (+10)_{10} =$ 00001010 → 11110101 → 11110110

A − B

$= A + (-B) = -12 + (-10) = -12 - 10 = -22$

$=$ 11110100 + 11110110 = 1 11101010 = (00010110)

Discard

B − A = +10 − (−12) = 10 + 12 = 22

00001010

00001100

00010110

Since MSB is '0', result is positive = +(22)

Book question

11010 − 10000 (1's + (−

11010 → 00011010 → 11100110

10000 → 00010000 → 11100000

00011010

+ 11100000

11111010

= (00000110)

Date: 11 Nov. /

Overflow

When we are adding two numbers of n bit each & the result is n+1 bit then there will be overflow. When overflow occurs, the AVF (Added after overflow flag) is set. Overflow may occur when we are adding two numbers with the same sign bit. To detect the overflow the two most n carry are passed through the X-OR Gate and if the result is one then overflow is detected.

Digital Code

1. BCD
2. ASCII
3. Gray Code
4. Excess-3

Date: 12 Nov. /

UNIT 2

REGISTER TRASFER & MICROPROCESSORS

- Digital Modules
- Register
- Micro operation
- RTL

A digital system is an interconnection of digital hardware module that accomplish a specific information processing task. Digital modules are best defined by the register they contain & the operation that are performed on the data stored in them. The operation executed on the data stored in register are called micro operation. The symbolic notation that is used to describe the micro operation transfer among register is called RTL. The internal hardware organization of a digital computer is defined by specifying:-

1. The set of register it contains & their function
2. The sequence of micro operation performed on a binary information stored in the register.
3. The control that initiates the sequence of micro operation

Date: / /

- Bus and Memory Transfer

Bus Transfer

A typical digital computer has many register and a path must be provided to transfer information from one register to another. The number of wires will be excessive if separate lines are used between each register which is practically not feasible. A more efficient scheme is to use a common bus system. A common bus system can be implemented using two different techniques i.e. common bus implementation using multiplexer or using 3 state buffer.

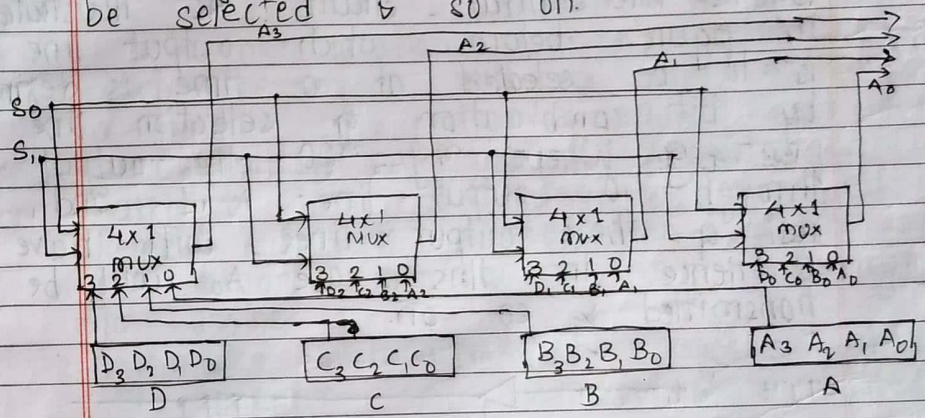
*Common Bus Implementation using Multiplexer

17 Nov

A common bus implementation using multiplexer is illustrated below for 4 register with 4 bit each. 4 multiplexer with 4 input lines i.e. 4x1 mux is taken. In mux 1, the most significant bit of the given registers that is D_3, C_3, B_3 & A_3 are connected, D_2, C_2, B_2 & A_2 are connected to mux 2 and so on. Two selection lines S_0 & S_1 are connected to all the multiplexer. The bit combination of a selection line S_0 & S_1 will decide which register is to be

Date: 17 Nov /

selected. When S_0, S_1 $S_0S_1 = 00$ the 0 input line of every multiplexer is activated. Hence the content of a register A will be transferred through 4 line common bus. Similarly when $S_0S_1 = 01$, register B will be selected & so on.



*Common Bus Implementation using 3-state Buffer

A Bus system can be constructed with three state gates instead of multiplexer. A 3-state gate is a digital circuit that execute ___ three states. Two of the states are signal equivalent to logic 1 & 0 & the third state is high impedance state. The high impedance state behaves like open circuit which means that output is disconnected and doesn't have logic significance. The graphic symbol of 3-state buffer in fig: A below

Date: 17 Nov. /

The construction of Bus system with a 3-state buffer is illustrated in figure. B where 2x4 decoder is used. The figure illustrated ith state for selection of register & we've n bit of data we will require n similar circuit. As illustrated in figure below, which output line is to be selected at a time is defined by bit combination of selection line x & y. When xy = 00, the output through 0 output line is 1 and rest of the output lines will have 0. Hence in this case A_0 will be transmitted & so on.

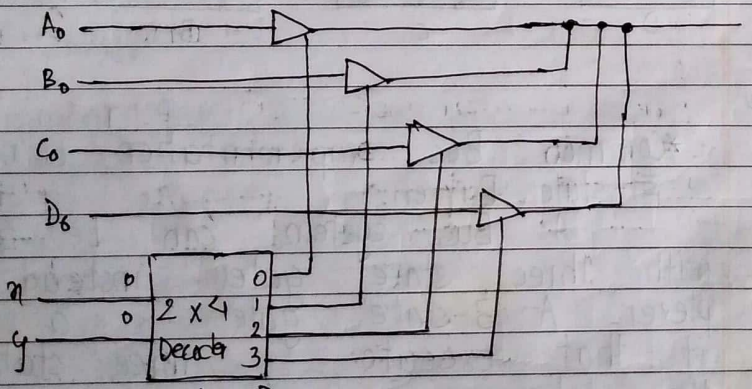


fig: B

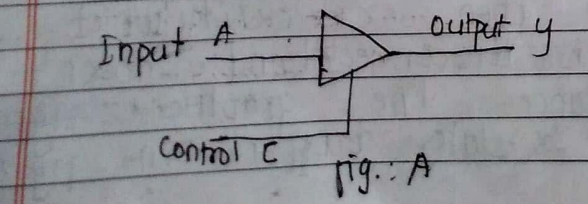


fig: A

Date: / /

Memory Transfer

Read operation : $DR \leftarrow M[AR]$

Write operation : $M[AR] \leftarrow DR$

- Arithmetic Operation

A microoperation is an elementary operation performed with a data stored in a register. The micro operations most often encountered in a digital computer are classified into 4 categories as follows:-

1. Register Transfer microoperation which transfer binary information from one register to another
2. Arithmetic microoperation which performs arithmetic operation on numeric data stored in register.
3. Logic microoperation that perform bit manupulation operation.
4. Shift microoperation that perform shift operation on a data stored on a register.

The list of different airthmetic operations are as follows:-

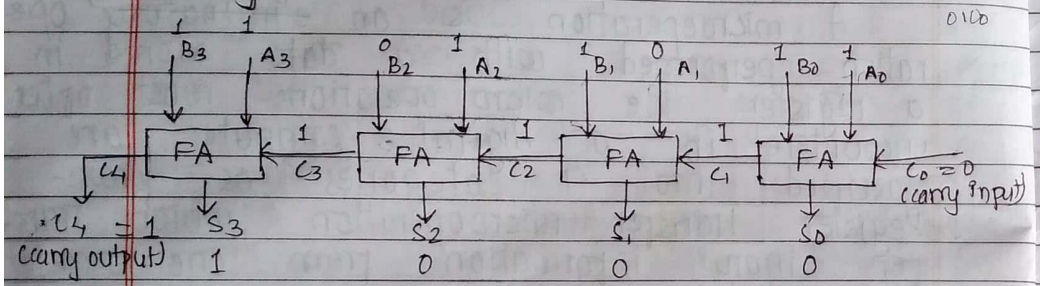
Symbolic Designation	Description
$R_1 \leftarrow R_2 + R_3$	The content of R_2 is added to R_3 & transferred to R_1.
$R_1 \leftarrow R_2 - R_3$	The content of R_3 is minus from R_2 & is transferred to R_1
$R_1 \leftarrow \bar{R}_1$	One's complement
$R_1 \leftarrow \bar{R}_1 + 1$	Two's complement (Negate)
$R_1 \leftarrow R_2 + \bar{R}_1 + 1$	(Subtraction) The content of R_2 is added to 2's complement of R_1.

Date: Nov 18 /

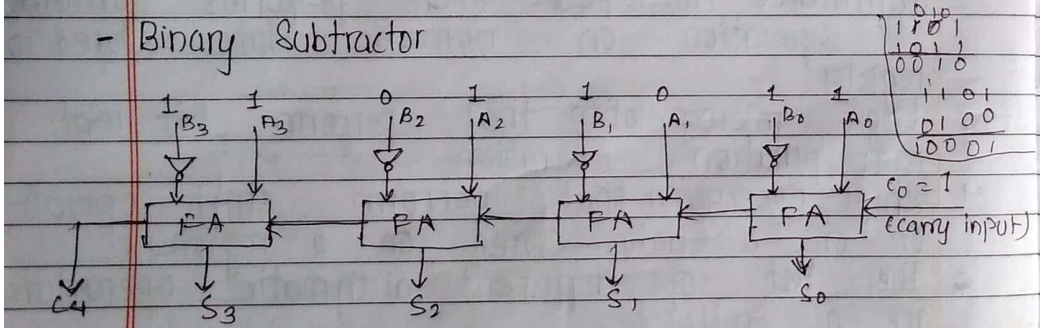
$R_1 \leftarrow R_1 + 1$	Increment
$R_1 \leftarrow R_1 - 1$	Decrement

Nov. 18

- Binary Adder



- Binary Subtractor



$A - B = A_3A_2A_1A_0 - (+B_3B_2B_1B_0)$

$= A_3A_2A_1A_0 - 2$'s complement of $B_3B_2B_1B_0$

$= \overset{C_3\ C_2\ C_1}{A_3\ A_2\ A_1\ A_0}$

$\quad \overline{B_3}\ \overline{B_2}\ \overline{B_1}\ \overline{B_0}$

$\quad + 1$

$C_4\ S_3\ S_2\ S_1\ S_0$

$x \oplus 0 = x$

$x \oplus 1 = \bar{x}$

3 bit add garda FA
2 bit add gard HA

Date: / /

- Binary Adder Subtractor

$x \oplus 0 = x$

$x \oplus 1 = \bar{x}$

Addition

$A_3A_2A_1A_0$

$B_3B_2B_1B_0$

$+ 0$

$C_4S_3S_2S_1S_0$

Subtraction

$A_3A_2A_1A_0$

$\overline{B_3}\ \overline{B_2}\ \overline{B_1}\ \overline{B_0}$

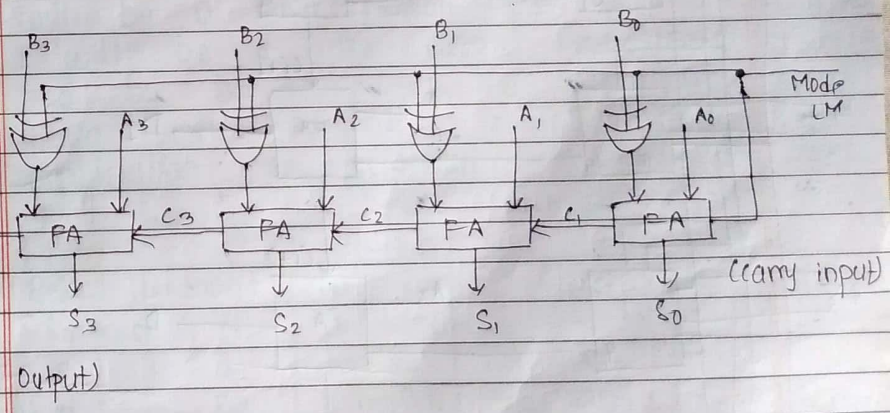
$+ 1$

$C_4S_3\ S_2S_1S_0$

When,

Mode (M) = 0 (Add)

Mode (M) = 1 (Subtract)



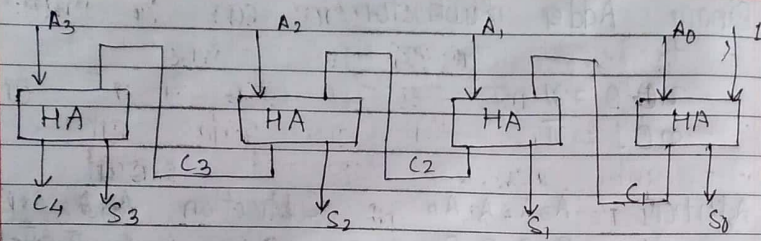
- Binary Incrementor

$\overset{C_3\ C_2\ C_1}{A_3A_2A_1A_0}$

$+ 1$

$C_4S_3S_2S_1S_0$

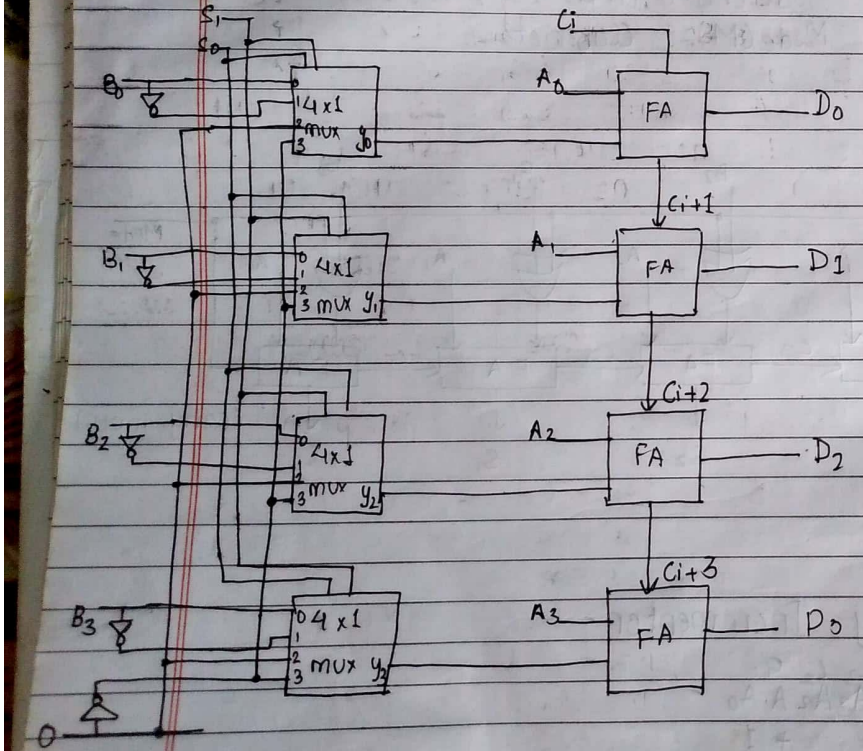
Date: 19 Nov. /



Carry output

Nov. 19

- Arithmetic Circuit



Date: / /

The arithmetic micro operation can be implemented by a circuit which is illustrated in fig. A. In the figure, it describe the 4 bit arithmetic circuit. It has 4 full adder circuit & 4 mux for choosing different operation. There are two 4bit input A & B and 4bit output i.e. D. The input A is directly provided to the full adder whereas the output of the mux i.e. Y is provided as a next input. Which arithmetic is to be performed is defined by the bit combination of a selection line S_1 & S_0. When $S_1S_0 = 00$, the input Y will be B, when $S_1S_0 = 01$, the input Y will be $\bar{B}$, when $S_1S_0 = 10$ the input Y will be 0 and when $S_1S_0 = 11$ the input Y will be 1. The output function is defined by $D = A + Y + C$. The different types of arithmetic possible is based on bit combination of selection line S_1S_0 & carry input Cin which are illustrated in a table below:-

Select $S_1S_0C_{in}$	Input Y	Output $D = A+Y+C$	Description
0 0 0	B	$D = A + B$	Addition
0 0 1	B	$D = A + B + 1$	Addition with carry
0 1 0	$\bar{B}$	$D = A + \bar{B}$	Subtraction with borrow
0 1 1	$\bar{B}$	$D = A + \bar{B} + 1$	Subtraction
1 0 0	0	$D = A$	Transfer
1 0 1	0	$D = A + 1$	Increment
1 1 0	1	$D = A - 1$	Decrement
1 1 1	1	$D = A$	Transfer

Date: / /

- Logic Micro operation

The logic micro operation defines the bit-wise manipulation of binary data. In a logic micro operation, each bit will be treated as a separate logic variable. The major type of logic micro operation are AND, OR, NOT & XOR. While representing AND operation of A & B, in a boolean function as A·B and the OR operation is represented as A+B. These operators are not used in computer since these create ambiguity with multiplication & addition operation respectively. Hence for representing AND operation we use the symbol "∧" & for representing OR operation we use the symbol "∨"

Nov. 26

Half - Adder / Subtractor

a	b	sum	carry	diff.	borrow
0	0	0	0	0	0
0	1	1	0	1	1
1	0	1	0	1	0
1	1	0	1	0	0

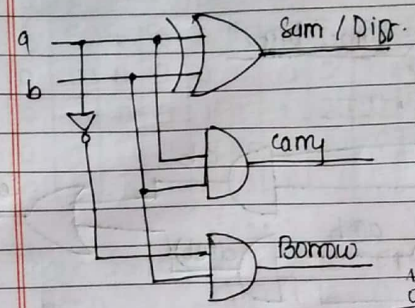
Sum = $\bar{a}b + a\bar{b}$ = $a \oplus b$

Carry = ab

diff. = $\bar{a}b + a\bar{b}$ = $a \oplus b$

borrow = $\bar{a}b$

Date: / /



Full - Adder / Subtractor

a	b	c	Sum	Carry	Diff.	Borrow
0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	0	1	0	1	1
0	1	1	0	1	0	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	1	0	0	1	0	0
1	1	1	1	1	1	1

Sum = $\bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}\bar{c} + abc$

$= \bar{a}(\bar{b}c + b\bar{c}) + a(\bar{b}\bar{c} + bc)$

$= \bar{a}(b \oplus c) + a(\overline{b \oplus c})$

$= a \oplus b \oplus c$

Carry = $\bar{a}bc + a\bar{b}c + ab\bar{c} + abc$

$= c(\bar{a}b + a\bar{b}) + ab(\bar{c} + c)$

$= c(a \oplus b) + ab$

Diff. = $\bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}\bar{c} + abc$

$= \bar{a}(\bar{b}c + b\bar{c}) + a(\bar{b}\bar{c} + bc)$

$= \bar{a}(b \oplus c) + a(\overline{b \oplus c})$

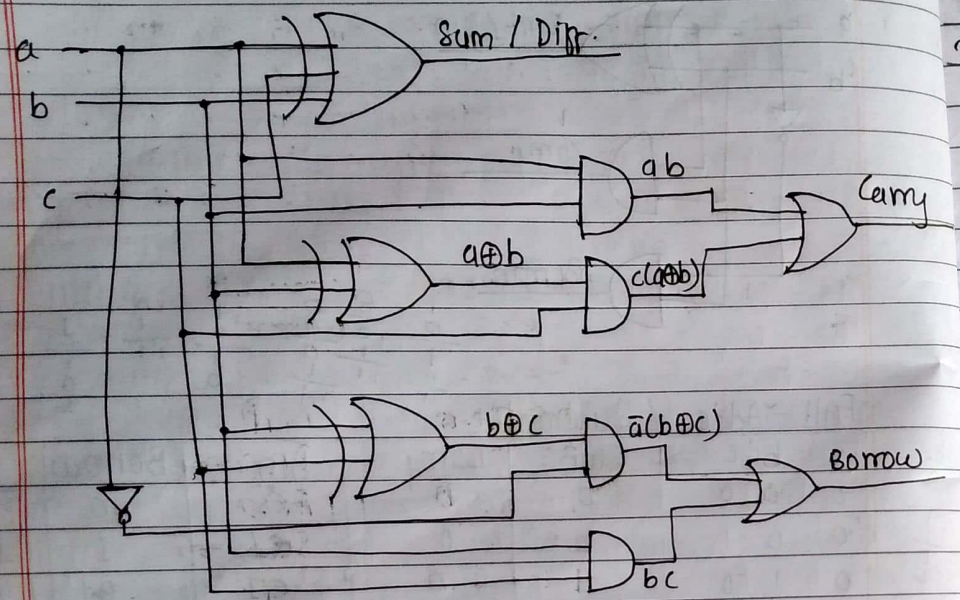
$= a \oplus b \oplus c$

Borrow = $\bar{a}\bar{b}c + \bar{a}b\bar{c} + \bar{a}bc + abc$

$= \bar{a}(\bar{b}c + b\bar{c}) + bc(\bar{a} + a)$

$= \bar{a}(b \oplus c) + bc$

Date: / /



$\bar{x}y + x\bar{y} + xy$

$\bar{x}y + x(\bar{y}+y)$

$\bar{x}y + x$

$(x+y)(x+\bar{x})$

$x+y$

Date: Dec/1 /

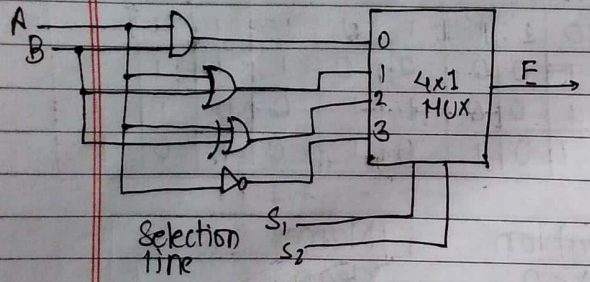
- Logic Microperation

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

Basic Function	Operation	Name
$F_0 = 0$	$A \leftarrow 0$	Clean
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = x\bar{y}$	$F \leftarrow A \wedge \bar{B}$	AND
$F_3 = x\bar{y} + xy = x$	$F \leftarrow A$	Transfer
$F_4 = \bar{x}y$	$F \leftarrow \bar{A} \wedge B$	AND
$F_5 = y$	$F \leftarrow B$	Transfer
$F_6 = \bar{x}y + x\bar{y}$	$F \leftarrow A \oplus B$	XOR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = \bar{x}\bar{y}$	$F \leftarrow \bar{A} \wedge \bar{B}$	AND
$F_9 = x\bar{y} + xy$	$F \leftarrow A \odot B$	X-NOR
$F_{10} = \bar{y}$	$F \leftarrow \bar{B}$	Complement
$F_{11} = x + \bar{y}$	$F \leftarrow A \vee \bar{B}$	OR
$F_{12} = \bar{x}$	$F \leftarrow \bar{A}$	Complement
$F_{13} = x + y$	$F \leftarrow A \vee B$	OR
$F_{14} = \bar{x}\bar{y}$	$F \leftarrow \bar{A}\bar{B}$	NAND
$F_{15} = 1$	$F \leftarrow 1$	Set

Date: / /

- Hardware Implementation Clogical micro operation / Circuit

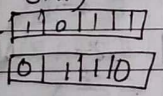


S_1 S_2	Operator	Name
0 0	$E \leftarrow Ai \wedge Bi$	AND
0 1	$E \leftarrow Ai \vee Bi$	OR
1 0	$E \leftarrow Ai \oplus Bi$	XOR
1 1	$E \leftarrow \overline{A}$	Complement

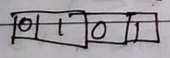
- Shift Microoperation

• Logical Shift (Shl or Shr)

$R_1 \leftarrow Shl\ R_1$

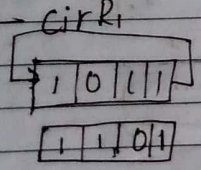


$R_1 \leftarrow Shr\ R_1$

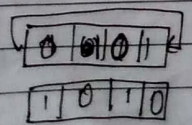


• Circular Shift (Cil or Cir)

$R_1 \leftarrow Cir\ R_1$



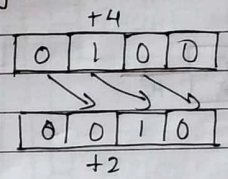
$R_1 \leftarrow Cil\ R_1$

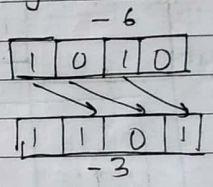


Date: / /

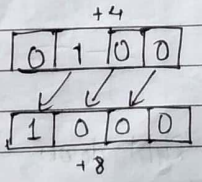
• Arithmetic Shift (

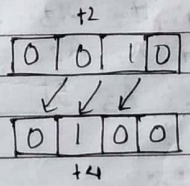
* Right shift: In arithmetic right shift, the content of register is shifted one position right but the signed bit is preserved. After every shifting operation the content of register is divided by 2.





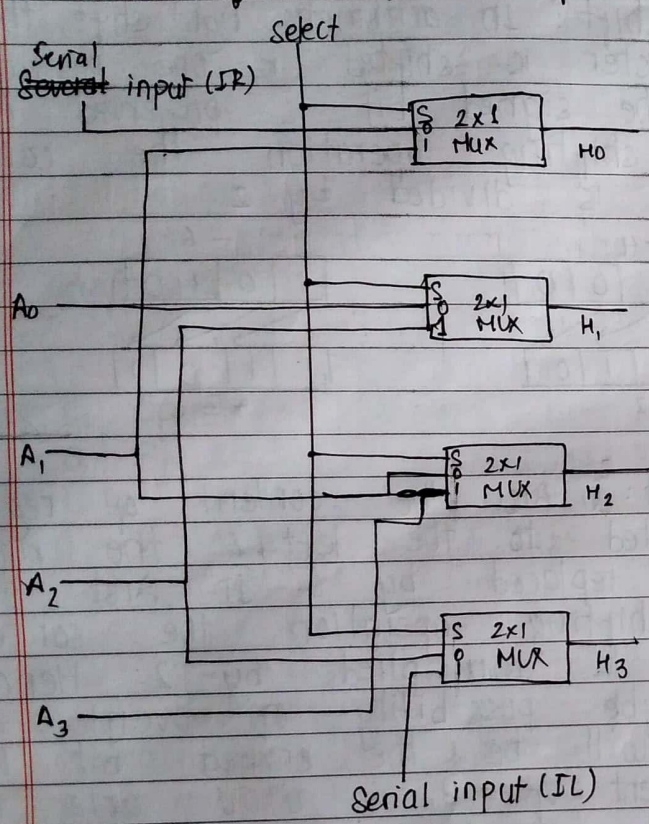
* Left shift: In Arsl the content of register is shifted to the left & the right most bit is replaced by 0. In Arsl after every shifting operation the content of register is multiplied by 2. Hence there maybe possibility of overflow & overflow will be the signed bit. For eg. if content of R_1 is 0100 after arsl, it will be 1000 which means there is overflow & overflow is the signed bit.





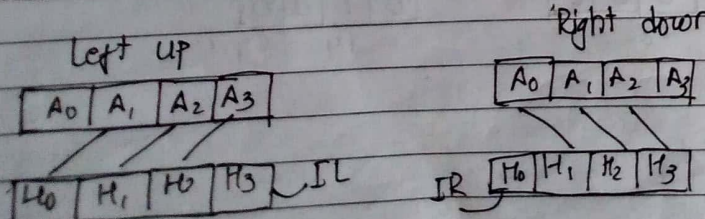
Date: Dec / 8 /

- Hardware for Shift Micro-operation



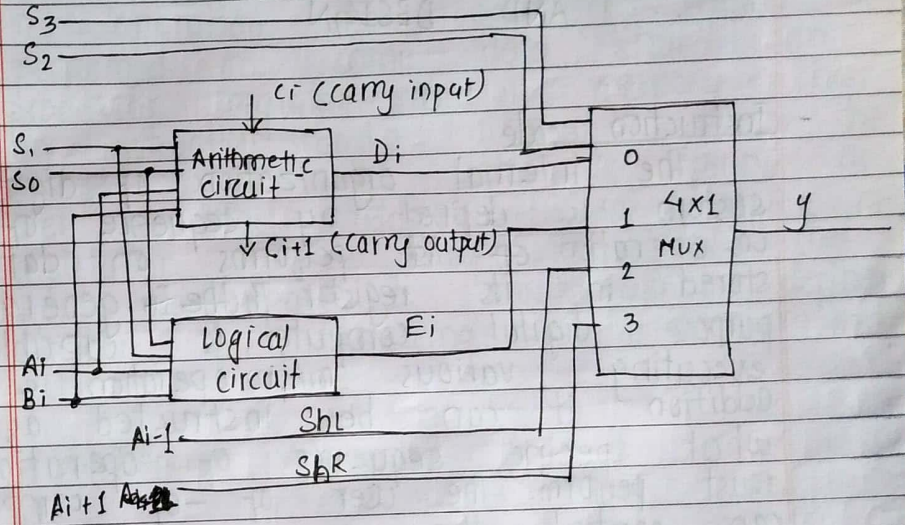
Fundmental table

Select	Output			
S	H_0	H_1	H_2	H_3
0	IR	A_0	A_1	A_2
1	A_1	A_2	A_3	IL



Date: / /

- Arithmetic logic & Shift Unit



Operation Select					Operation	Function
S_3	S_2	S_1	S_0	C_i	~~$y = A + B$~~	~~Addit~~
0	0	0	0	0	$y = A + B$	Addition
0	0	0	0	1	$y = A + B + 1$	Addition with carry
0	0	0	1	0	$y = A + \overline{B}$	Subtraction with borrow
0	0	0	1	1	$y = A + \overline{B} + 1$	Subtraction
0	0	1	0	0	$y = A$	Transfer
0	0	1	0	1	$y = A + 1$	Increment
0	0	1	1	0	$y = A - 1$	Decrement
0	0	1	1	1	$y = A$	Transfer
0	1	0	0	X	$y = Ai \wedge Bi$	AND
0	1	0	1	X	$y = Ai \vee Bi$	OR
0	1	1	0	X	$y = Ai \oplus Bi$	XOR
0	1	1	1	X	$y = \overline{Ai}$	Complement
1	0	X	X	X	$F \leftarrow shl$	Shift Left
1	1	X	X	X	$F \leftarrow shr$	Shift Right

Date: Dec./9 /

UNIT 3

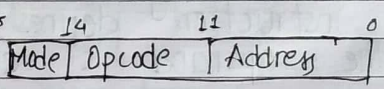
BASIC COMPUTER ORGANIZATION AND DESIGN

- **Instruction code**

The internal organization of digital system is defined by sequence of micro operation ~~on~~ it performs on data stored in its register. ~~The the In~~ general purpose digital computer is capable of executing various micro operation & in addition it can be instructed as to what specific sequence of operation it must perform. The user of ~~-or~~ computer can control the process by means of program & program is set of instruction that specify the operations, operands & the sequence by which the processing is to occur. An instruction code is a group of bit that instruct the computer to perform specific operation. The instruction code is usually divided into 3 basic part:

i) Operation code
ii) Address
iii) Mode

The operation code of an instruction is a group of bit that defines the operation to be performed like addition, subtraction, multiply, etc. If the opcode is of n bit it will define 2^n distinct operation. The operation must be performed on some data stored on processor register or in memory. Thus an instruction code must define the address where the data is found as well as the address where after computation the result is stored. The next imp. part is mode which defines whether the addressing is direct or indirect.



- **Stored Program Organization**

The simplest way to organize a computer is to have one processor register & an instruction code format with 2 parts. The 1 part specifies the operation to perform & second part specifies address. Figure below illustrate stored program organization where the instruction are stored in one section of memory & data in another section. In this org. the control reads 16-bit instruction from programed portion of memory out of which 12 bit address part is used to specify the 16-bit operand from the data portion of a memory. If an

Date: / /

Date: / /

operation code in an instruction code doesn't need an operand from memory, the rest of the bit in instruction code can be used for other operation. For e.g: CMA (Complement Accumulator) In other case, we need to fetch the operand & it can be ~~face phas~~ fetched from two different path i.e. direct & indirect addressing.

(i) Direct Addressing

In a direct addressing, the address part of instruction defines the address where the operand is stored.

(ii) Indirect Addressing

In indirect addressing, the address part of instruction points to the address where the address of a operand is stored.

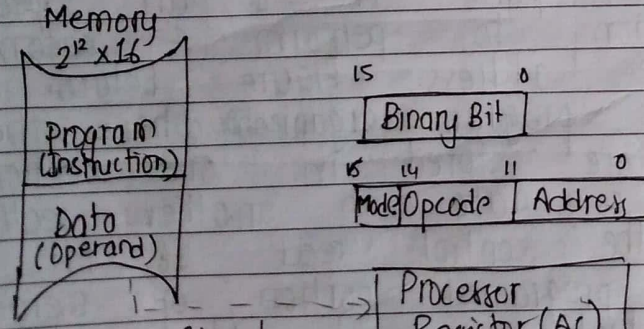


fig: Stored Program Organization

Date: Dec. 10 /

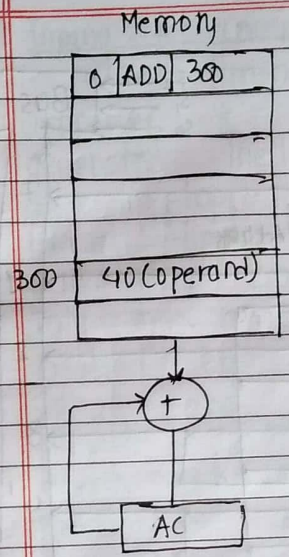


fig: Direct Addresing

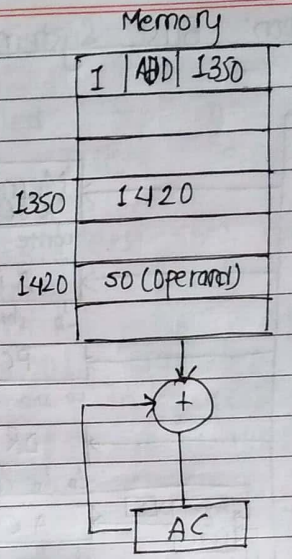


fig: Indirect addressing

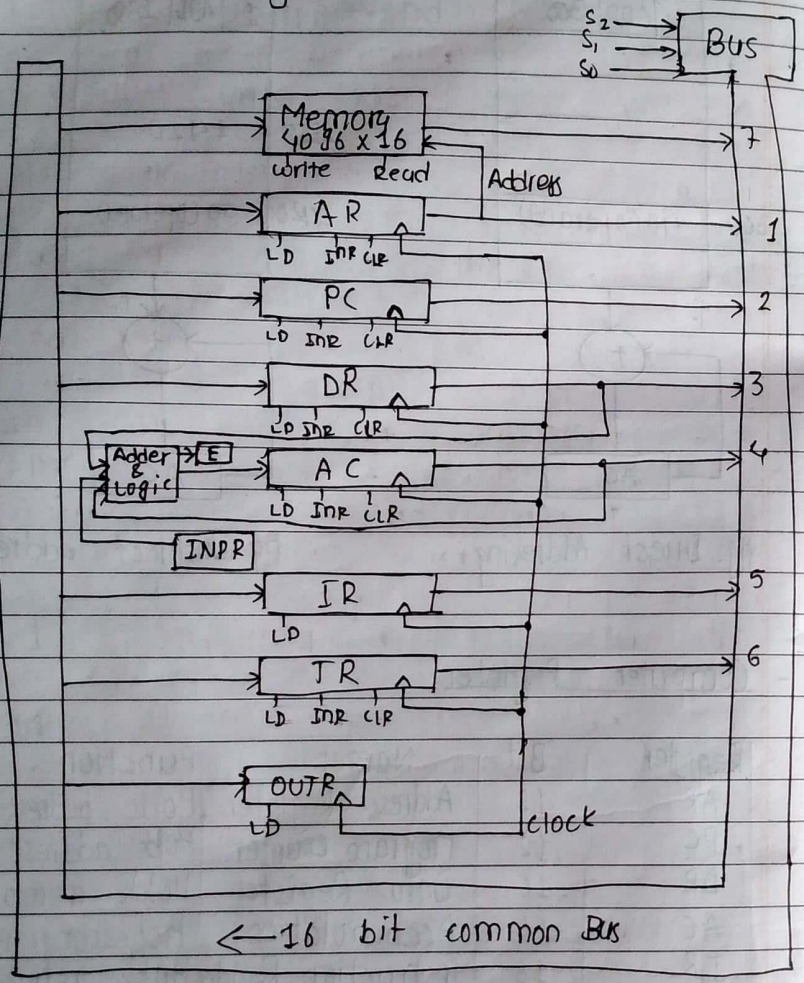
Dec. 10

- Computer Register

Register	Bit	Name	Function
AR	12	Address Register	Holds address for memory
PC	12	Program Counter	Holds address of instruction
DR	16	Data Register	Holds memory operand
AC	16	Accumulator	Processor register
IR	16	Instruction Register	Holds instruction code
TR	16	Temporary Register	Holds temporary data
InPR	8	Input Register	Holds input character
OUTR	8	Output Register	Holds output character

Date: / /

- Common Bus System



Date: Jan 5. /

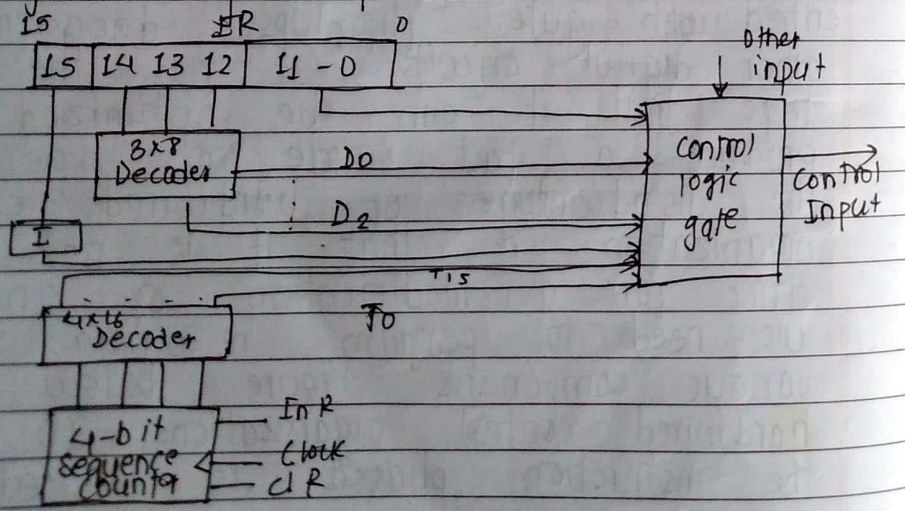
- Timing & Control

The timing for all register in basic computer is controlled by a master clock generator. The clock pulses are applied to all flip flops & registers in system including flipflops & registers in the control unit. The clock pulses don't change the state of register until the register is enabled by a control signal. The control signal are generated in control unit & provide control input for various components such as control input for multiplexer in common bus, control input in processor register & micro operation for the accumulation. There are two major types of control org. i.e. Hardwired control & microprogrammed control.

1. Hardwired control: In ~~thi~~ hardwired organization, the control logic is implemented with gates, flipflops, decoders & other digital circuits. It has the advantage that it can be optimized to produce a fast mode of operation. The disadvantage of hardwired control organization ~~is~~ that it is not flexible since for a modification of anything we need to perform rewiring among various components. Figure below illustrate hardwired control organization. In the fig. the instruction phased is placed in

Date: / /

instruction register. It consists of 2 decoder, a sequence counter & a no. of control logic gates. The operation code in bits 12 through 14 are decoded with 3×8 decoder where the outputs of decoder are designated by the symbol D_0 through D_7. Bit 15 of instruction is transferred to flip flop designated by the symbol I. Bit 0 through 11 are applied to the control logic gate. The 4-bit sequence counter counts value from 0000 to 1111 producing 16 decoded timing signal T_0 through T_{15}. The sequence counter can be incremented or cleared by the respective control signal. Let us suppose the ~~sequen~~ content of data register is transferred to accumulator at the time cycle T_4 when the decoder output D_5 is active, it can be symbolically represented as D_5T_4: AC ← DR



Jan 6

Date: Jan. 6 /

- **Instruction Cycle**

A program residing in the memory unit of a computer consist of sequence of instruction. The program is executed on the computer by going through a cycle for each instruction. Each instruction in turn is sub-divided into a sequence of sub-cycle or phases such as follows:

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

After completion of step 4, control goes back to step 1 & process continues until & unless the halt instruction is encountered.

- **Fetch & Decode**

micro operation for fetch & decode

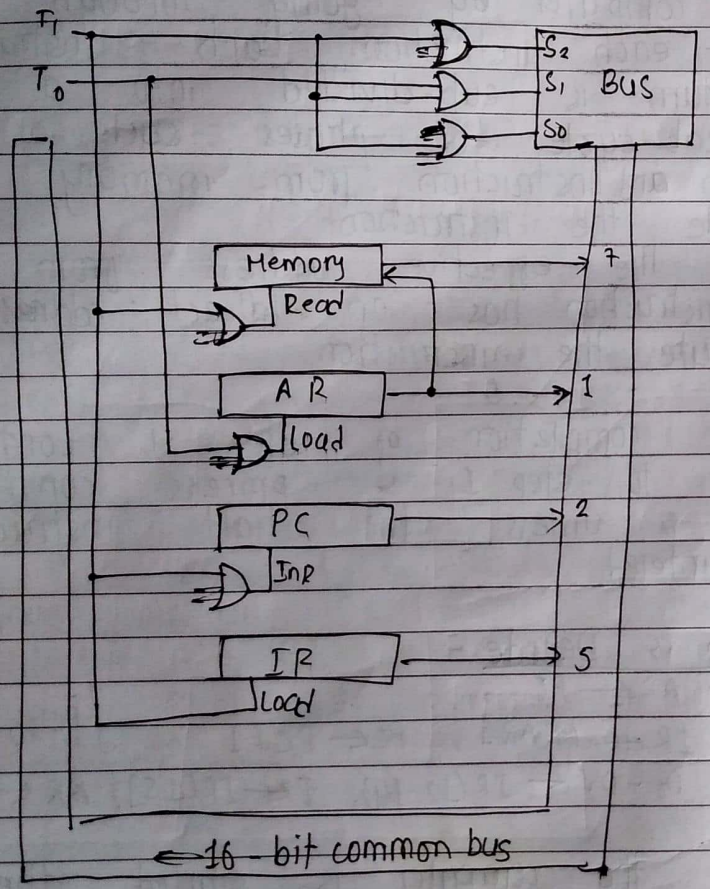
T_0 : AR ← PC
T_1 : IR ← M[AR], PC ← PC + 1 } Fetch
T_2 : $D_0 - D_7$ ← IR(12-14), I ← IR[15], AR ← IR(0-11) } Decode

When the computer is started, the control passes the address of a first instruction to be loaded into the program counter. The sequence counter is cleared to 0, providing decoded timing signal T_0. After each clock pulse SC is incremented by 1 so that timing signal goes through sequ-

Date: / /

nce $T_0, T_1, T_2 \ldots$ & so on the micro operation for the ~~fir~~ fetch & decode operation together with a register transfer for fetch is illustrated below.

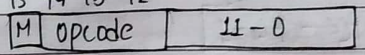
agadi



Date: / /

- Instruction

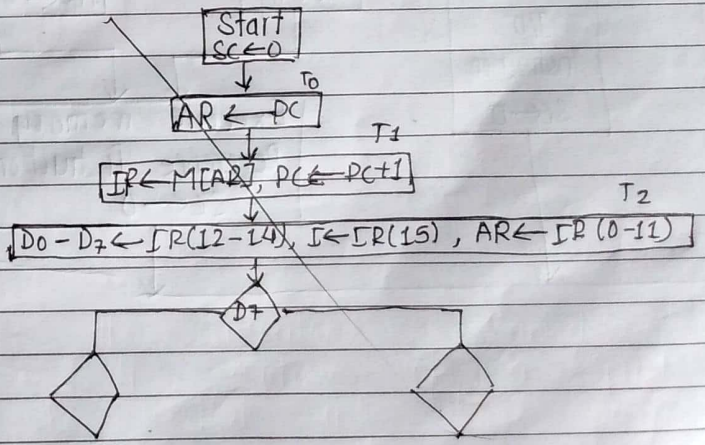
1. Memory Reference Instruction
2. Register Reference Instruction
3. I/O Instruction



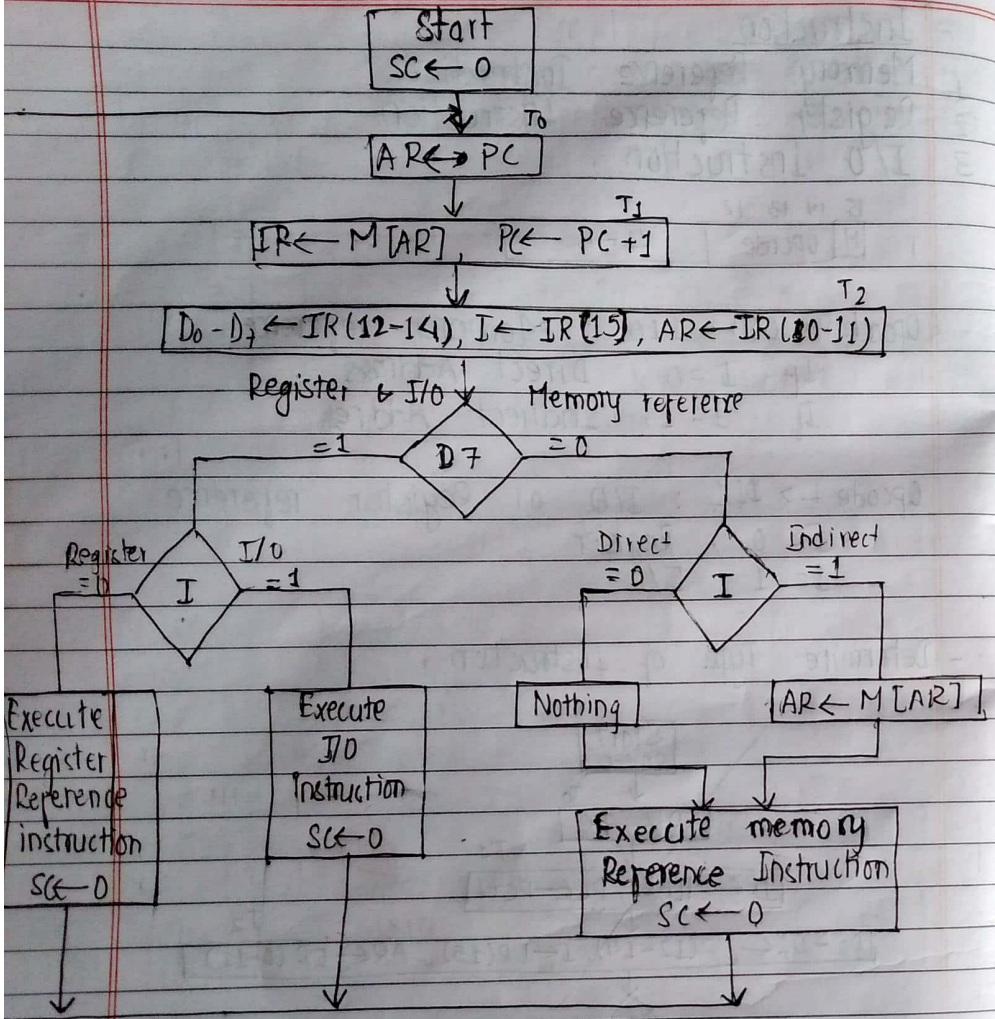
Opcode → 000 - 110 : Memory reference
If I = 0, Direct Address
If I = 1, Indirect Address

Opcode → 111 : I/O or Register reference
I = 0, Register
I = 1, I/O

- Determine Type of Instruction



Date: / /



$T_0 - T_2$: Fetch & decode
D7 : Decision
Rest : Execution

Date: Jan 7./

- Input - Output - Interrupt

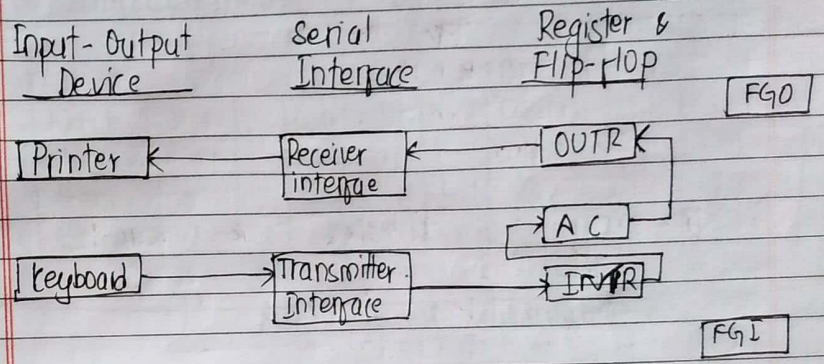
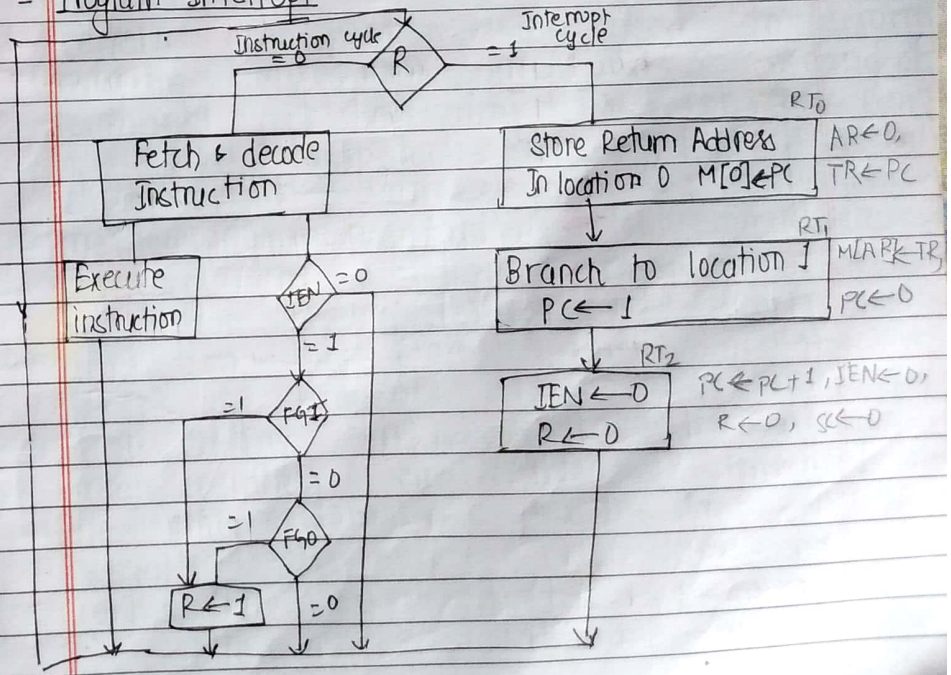


fig: Input - Output Configuration

- Program Interrupt



Date: Jan 12 /

UNIT 5

MICROPROGRAMMED CONTROL ORGANIZATION

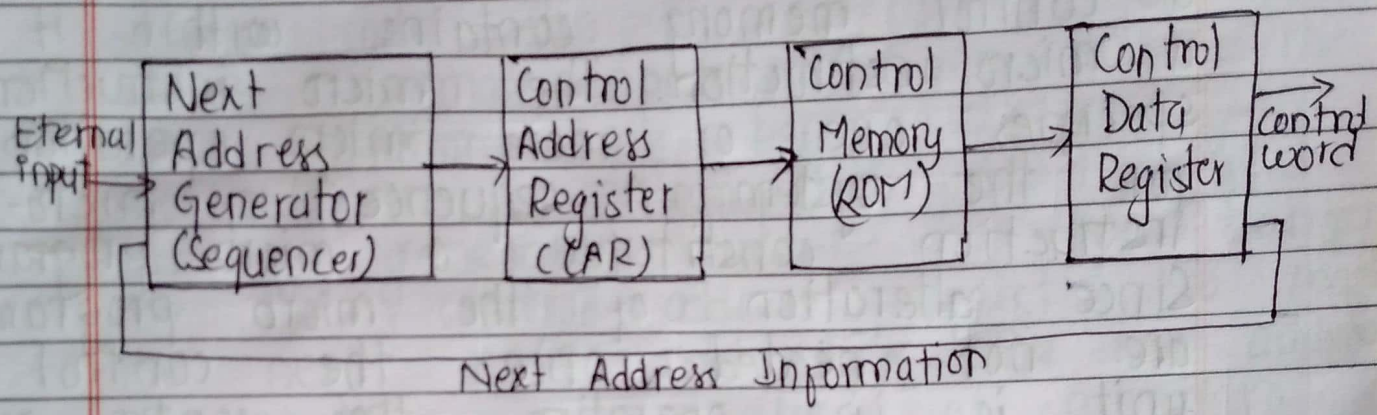


fig: Microprogrammed control organization

The limitation of hardwired control organization is that it isn't flexible since for certain modification, we need to perform rewiring. Microprogramming is a second alternative for designing the control unit of digital computer. The principle of microoperation is an elegant & systematic method for controlling the microoperation sequences in a digital computer. Fig. above illustrate the microprogrammed control organization. The control unit initiates a series of sequential steps of microoperation. During any given time, certain microoperation are to be initiated while other remains idle. The control variable at any given time can be represented by string of 1's & 0's which is called as a control word. A control

Date: / /

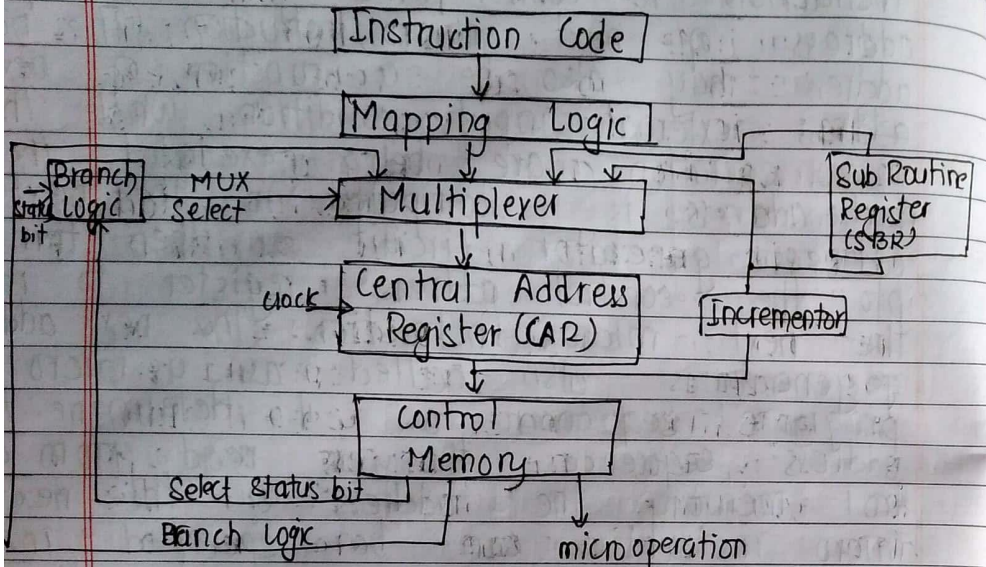
unit whose binary control variables are stored in memory is called a micro-programmed control unit. Each coord in a control memory contains within it a micro instruction. The micro instruction specifies one or more micro operation for the system. A sequence of micro-instruction constitutes a micro program. Since alteration of the micro program are not needed once the control unit is in operation, the control memory can be a ROM. A memory that is part of control unit is referred to as a control memory. A computer that emplo a micro program control unit will have two separate memories. i.e. Main memory & Control memory. The control address register specifies the address of the micro instruction & the control data register holds the micro instructions read from memory. The micro instruction contains a control word that specifies one or more micro operation for the data processor. Once these operation are executed, the control must determine the next address & the location of the next micro instruction maybe the one next in sequence or it may located somewhere else in the control memory. For this reason, it is neces

Date: / /

to use some bits of the present micro-instruction to control the generation of address of next micro instruction. The next address may also be a function of next ~~address~~ external input condition. While the micro operation are being executed, the next address is computed in the next address generator circuit & then transfer into the control address register to read the next micro instruction. The next address ~~ge~~ generator is also called as a micro program sequencer as it determine the address sequence that is read from control memory. The address of the next micro instruction can be specified in several ways such as incrementing the control address register by 1, loading into the control address register address from control memory, transferring an external address or loading an initial address to start the control operation. The control data register holds the present micro instruction while next address is computed & read from memory. For this reason, the data register is sometimes called as a Pipeline Register since it allows the execution of micro operation specified by control word simultaneously with the generation of the next micro instruction

Date: Jan/ 13 /

Address Sequencing



Micro instruction are stored in control memory in groups with each group specifying a routine. The hardware that controls the address sequencing of the control memory must be capable of sequencing the micro instruction within a routine & be able to branch from one routine to another. To understand the addres sequ in microprogram control unit, let us enumerate the state that the control must undergo during the execution of a single computer instruction. An initial address is loaded into the control address register when power is turned o

Date: / /

in the computer. and this address is usually the address of the first micro instruction that activates the instruction fetch routine. The fetch routine maybe sequenced by incrementing CAR through the rest of its micro instruction and at the end of fetch routine, the instruction each in the instruction register of the computer. The control memory next must go through the routine that determine the effective address of the operand & when the effective address computation routine is completed, the address of operand is available in the M[AR]. The next step is to generate the micro operation that execute the instruction fetched from memory. The micro operation step to be generated in processor register depend on the operation code part of the instruction where the instruction code bit is transferred to an address in control memory where the routine is located & is referred to as a mapping process. When the execution of the instruction is completed control must return to the fetch routine & this is accomplished by executing an unconditional branch micro instruction to the first address of the fetch routine. In summary, the address sequencing categories required in a control

Date: / /

memory are:

1. Incrementing of CAR
2. Unconditional branch or conditional branch depending on status bit condition
3. A mapping process from the bits of the instruction to an address for control memory
4. A facility for sub routine call & return

Fig. above shows a block diagram of control memory & the associate hardware needed for selecting the next micro instruction address. The diagram shows 4 different path from which the control address register receives an address. The incrementor increment the content of the control address register by 1 to select the next micro instruction in sequencer Branching is achieved by specifying the branch address in one of the fields of micro instruction. Conditional branching is obtained by using part of micro instruction to select a specific status bit in order to determine its condition. An external address is transferred into control memory via a mapping logic circuit. The return address for a sub routine is stored in a special register ____ whose value is then used when microprogram wishes to return

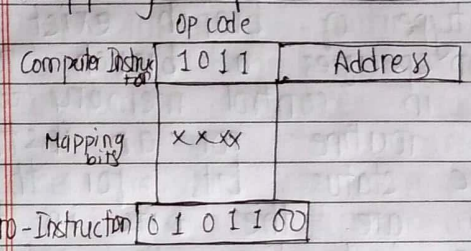
Date: Jan 14/

from the sub routine

Jan. 14

- Conditional Branching

Mapping of Instruction



The branch logic of figure above provides decision making capabilities in the control unit. The status condition are special bits in the system that provide parameter instruction such as carry out of adder, signed bit of number, the mode bits of an instruction and input or output status condition. Information in these bits can be tested & action initiated based on their condition whether their value is one or zero which is defined as condition branching. An unconditional branch micro instruction can be implemented by loading the branch address from control memory in the CAR. This can be accomplished by fixing the value of 1 status bit at the input of the multiplexer. So, it is always equal to 1.

Date: / /

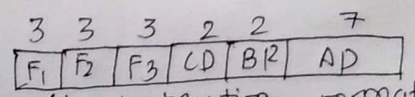
A reference to this bit by the status bit selection line from control memory causes the branch address to be loaded into CAR unconditionally.

Mapping of instruction

A special type of branch exist when a micro instruction specifies a branch to the first word in control memory where a micro program routine for an instruction located. The status bits for this type of branch are the bits in the opcode part of instruction. For each opcode, there exist a micro program routine in control memory that executes the instruction. One simple mapping process that converts 4-bit opcode to 7-bit address for control memory is shown in figure above where the mapping consists of placing a 0 in the most significant bit of the address, transfering the 4 opcode bit & clearing the two least significant bit of the control address register (CAR)

Sub Routine

Sub routines are program that are used by other routines to accomplish a particular task. A sub routine can be called from any point within the main body of micro programm. Micro



Micro instruction format : 20 bits

Date: / /

program that use sub routine must have a provision for storing the return address during a sub routine call & restoring address during a sub routine return. This may be accomplished by placing the incremented output from the control address register into sub-routine register & branching to the begining of sub-routine. The sub-routine register can then become the source for transfering the address for the return to main routine. The best way to structure a register file that stores address for sub routine is to organize the register in a LIFO stack.

Micro Program Example

Computer Configuration

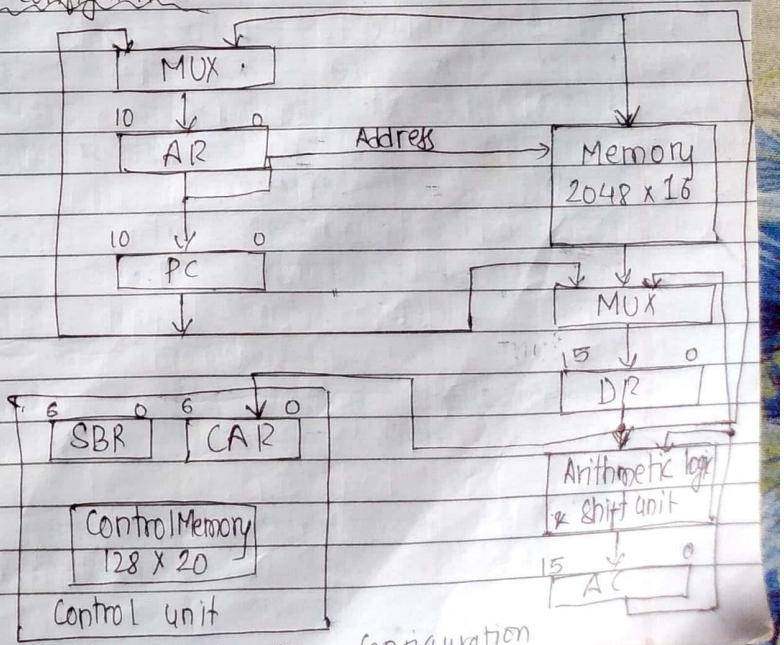
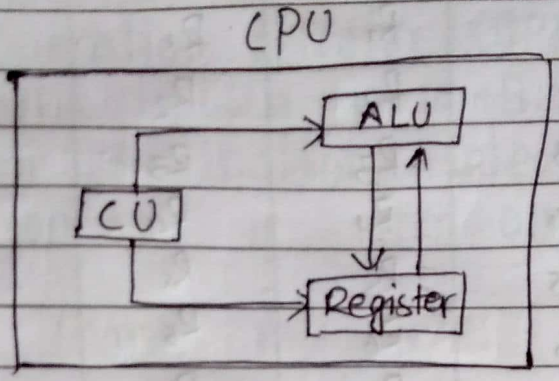


Fig: Computer Hardware Configuration

Unit - 6

Date: Jan 19

Central Processing Unit



- Register Organization

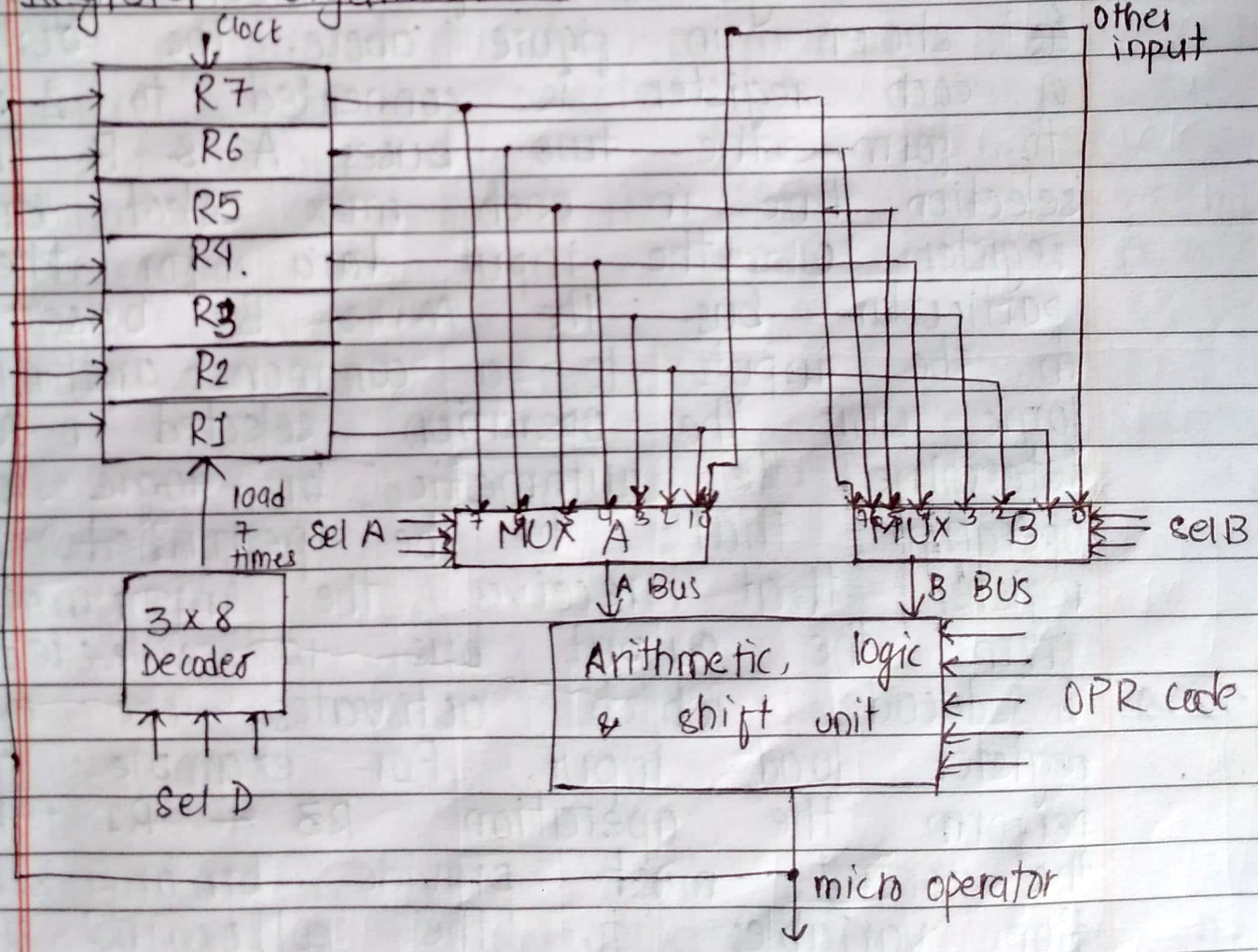


fig: Register set with common ALU

Sel A (3)	Sel B (3)	Sel D (3)	OPR (5)

control word

Date: / /

Binary Code	SeL A	SeL B	SeL D
0 0 0	Input	Input	None
0 0 1	R_1	R_1	R_1
0 1 0	R_2	R_2	R_2
0 1 1	R_3	R_3	R_3
1 0 0	R_4	R_4	R_4
1 0 1	R_5	R_5	R_5
1 1 0	R_6	R_6	R_6
1 1 1	R_7	R_7	R_7

A bus organization for 4 CPU register is shown in figure above. The output of each register is connected to 2 mux to form the two buses A & B. The selection line in each mux select one register or the input data for the particular bus. The A & B buses form the inputs to a common arithmetic logic unit. The operation selected in ALU determine the arithmetic or logic micro operation that's to be performed. The register that receives the information from the output bus is selected by a decoder which activates one of the register load input. For example: To perform the operation R3 ← R1 + R2, the control must provide binary selection variables to the following select input:

1. MUX A selector (SeLA): To place the content of R1 into Bus A.

Date: / /

2. MUX B selector (SeL B): To place the content of R2 into Bus B.
3. ALU operation selector (OPR): To provide the arithmetic addition A+B.
4. Decoder destination selector (SeL D): To transfer the content of the output bus into R3.

control word:

There are 14 binary selection input in the unit & their combine value specifies a control word. The 14-bit control word defines the operation to be performed. Which register is to be selected is defined by 3-bit binary code & these different combination are listed in table above. For performing the above operation, SeL A will have value 001, SeL B will have value 010 & SeL D will have value 011. Let us consider the binary addition operation is defined by opcode 00010, hence the 14-bit control word will be as follows:

Field	: SeL A	SeL B	SeL D	OPR
Symbol	: R1	R2	R3	Addition
Control word	: 001	010	011	00010

- Stack Organization

A useful feature that is included in the CPU of most computer is a stack which is also defined as LIFO organization. A stack is storage device that stores

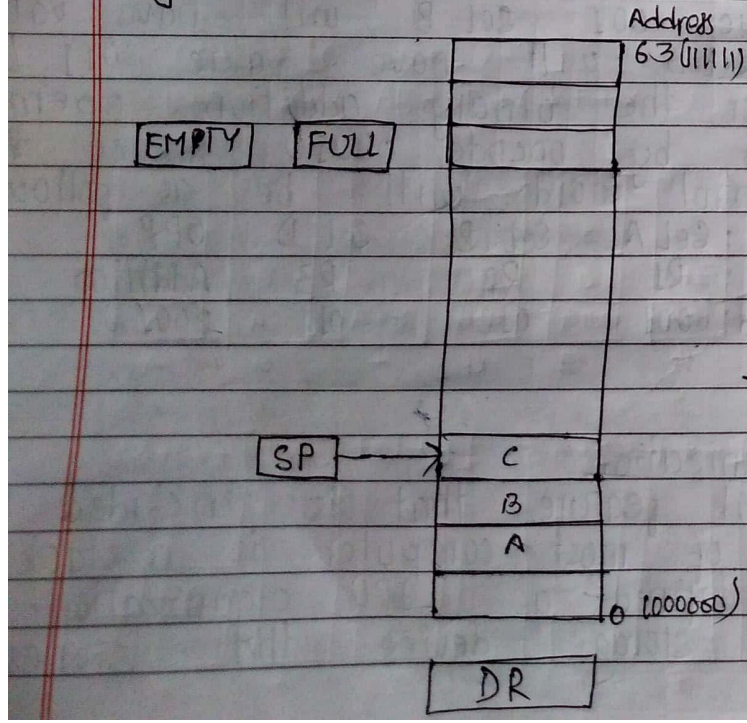
Date: Jan / 20 /

information in such a manner that the items in last is the first item received. The register that holds the address for the stack is called stack pointer which always points to the top of the stack. The two operation of stack are push & pop. The push operation defines the inserting of data item in the stack whereas the pop operation defines the extraction of data item in stack. Stack can be organized in computer using two different approaches :-

1. Register stack
2. Memory stack

20

1. Register Stack



Push operation:

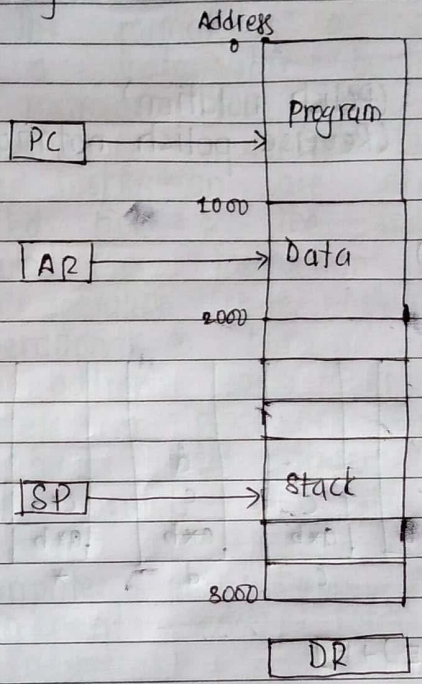
SP ← SP + 1
M[SP] ← DR
If (SP = 0), FULL ← 1
EMTY ← 0

Pop operation:

DR ← M[SP]
SP ← SP - 1
If (SP = 0), EMTY ← 1
FULL ← 0

Date: / /

2. Memory Stack



Push:

SP ← SP - 1
M[SP] ← DR

POP:

DR ← M[SP]
SP ← SP + 1

The limit register, is used in memory stack to define the size limit of the stack, that is, to check full & empty condition of the stack.

Date: / /

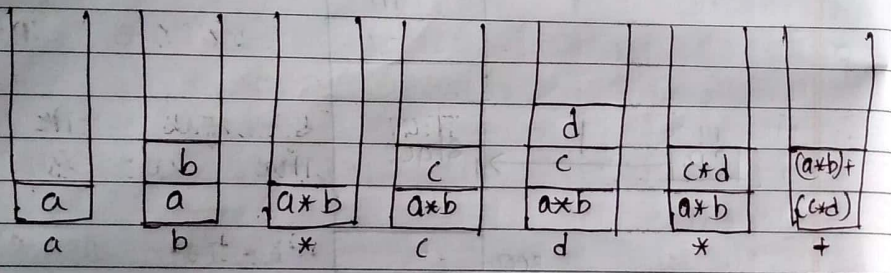
- Reverse Polish Notation (RPN)

1. Infix : a+b
2. Prefix : +ab (Polish notation)
3. Postfix : ab+ (Reverse polish notation)

RPN eg:

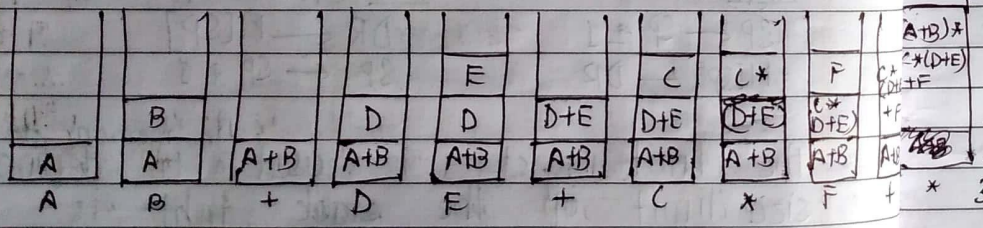
(a*b) + (c*d)

ab* cd* +

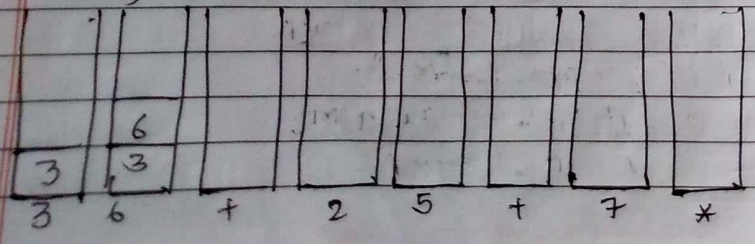


eg: (A+B) * [C * (D+E) + F]

~~BAB*~~ ~~AB+~~ AB+DE+C*F+*



(3+6) * [7 * (2+5) + 9] 3 6 + 2 5 + 7 * 9 + *



Date: Jan 22 /

- Instruction Format

The format of a instruction is illustrated in a rectangular box symbolizing the bits of instruction as they appear in memory word or in control register. The bits of the instruction are divided into groups called field & the most common field found in instruction format are:

1. An operation code field that specifies the operation to perform.
2. An address field that designate memory address or processor register
3. A mode field that specifies the way the operand or effective address are determine.

Computer may have instruction of several different length containing varying number of addresses & the no. of address field in the instruction format of a computer depends on the internal org. of its register. Most computer fall into one or three types of CPU org as follows:

1. Single Accumulator organization (ADD 5)
2. General Register Organization (ADD R1, R2, R3)
3. Stack Organization (Push X, Pop X)

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

One Address Instruction Format

```
load  A   // AC ← M[A]
ADD   B   // AC ← AC + M[B]
STORE T   // M[T] ← AC
```

Date: / /

```
load C // AC ← M[0] + AC
MUL T // AC ← AC * M[T]
STORE X // M[X] ← AC
```

Two Address Instruction

```
MOV   R1, A
ADD   R1, B
MOV   R2, C
ADD   R2, D
MUL   R1, R2
MOV   X, R1
```

Three Address Instruction

```
ADD   R1, A, B
ADD   R2, C, D
MUL   X, R1, R2
```

Zero Address Instruction

```
PUSH  A
PUSH  B
ADD
PUSH  C
PUSH  D
ADD
MUL
POP   X
```

IMP

- Addressing Mode

The operation field of an instruction specifies the operation to be performed & this operation must be executed in some data stored in register or memory word. The way the operands are chosen during program execution is depended on addressing mode of instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referred. There are different types of addressing mode as follows:-

1. Implied Mode
   In this mode the operands are specified implicitly in the definition of the instruction. For e.g. CMA (Complement Accumulation)

2. Immediate Mode
   In this mode, the operand is specified in the instruction itself i.e. the address part of instruction defines the operand. For eg. ADD 5.

3. Register Mode
   In this mode, the operand are in register that reside within the CPU.

4. Register Indirect Mode
   In this mode, the instruction specifies the register in CPU whose content gives address of operand in memory.

5. Auto Increment or Auto Decrement Mode
   This is similar to register indirect

Date: / /

mode except that register is incremented or decremented after or before its value is used to access memory.

6. Direct Addressing Mode
In this mode, the effective address is equal to the address part of instruction. The operand reside in memory & its address is given directly by address field instruction

7. Indirect Mode
In this, the address field of instruction gives the address where effective address is stored in memory.

8. Relative Mode
In this, the content of the PC is added to the address part of the instruction in order to obtain the effective address.

9. Indexed Mode
In this mode, the content of an index register is added to the address part of instruction to obtain the effective address.

10. Base Register Mode
In this mode, the content of base register is added to address part of

Date: Jan. 26 /

instruction to obtain the effective address of operand.

Jan 26 Numerical Example
The two word instruction at address 200 & 201 is a "load to AC" instruction with an address field equal to 500. The first word of the instruction specifies the opcode & mode. And the second word specifies the address part. PC has the value 200 for fetching this instruction. The content of processor register R1 is 400 & the content of an index register XR is 100. AC receives the operand after the instruction is executed. Now, calculate the effective address & the content of accumulator with reference to following addressing mode:

1. Direct addressing
2. Immediate "
3. Indirect "
4. Relative "
5. Register "
6. Register indirect "
7. Indext "
8. Auto-increment "
9. Auto-decrement "

Soln:

⇒

Date: / /

Addressing mode	Effective Address	Content of Register "AC"
Direct	500	650
Immediate	201	500
Indirect	650	47
Register	—	400
Register indirect	400	75
Auto increment	400	75
Auto decrement	399	56
Indexed	600	32
Relative	702	89

		Memory
200	MODE	"Load to AC"
201	Address = 500	
202	Next Instruction	
400		
399		56
400		75
401		84
500		650
600		32
650		47
702		89

[400] R1

[100] XR

[200] PC

Q.1. The three word instruction are stored in memory location 400, 401 & 402. The first word defines the mode, second word defines the opcode & third word defines the address. The instruction is "ADD to AC" where the content of AC is currently 50. The address part have a value 300. The content of processor register is 500. The content of index register is 200.

Date: / /

and the content of base register is 400. Now find the content of accumulator after the instruction is executed with reference to following addressing mode: direct, immediate, indirect, register, register indirect, auto increment, auto decrement, indexed, relative.

Sol:

Addressing Mode	Effective Address	Content of Register 'AC'
Direct	300	650
Immediate	402	300+50
Indirect	600	25+50
Register	—	500+50
Register Indirect	500	32+50
Auto Increment	500	82
Auto Decrement	499	62
Indexed	500	82
Relative	703	40+50

[50] AC

[500] R1

[200] XR

[400] BR

[500] PC

	Memory	
Mode	ADD TO AC	300
400	MODE	
401	ADD TO AC	
402	300	
403	Next address	
499	12	
500	32	
501	35	
600	25	
700	65	
703	40	

Date: Jan 29 /

DATA TRANSFER & MANIPULATION

- Data Transfer Instruction
- Data Manipulation Instruction
- Program control Instruction

- **Data Transfer Instruction**
  - → load → Input
  - → Store → Output
  - → Move → Push
  - → Exchange → Pop

- **Data Manipulation Instruction**
  - → Arithmetic Instruction
  - → Logical & bit manipulation instruction
  - → Shift Instruction

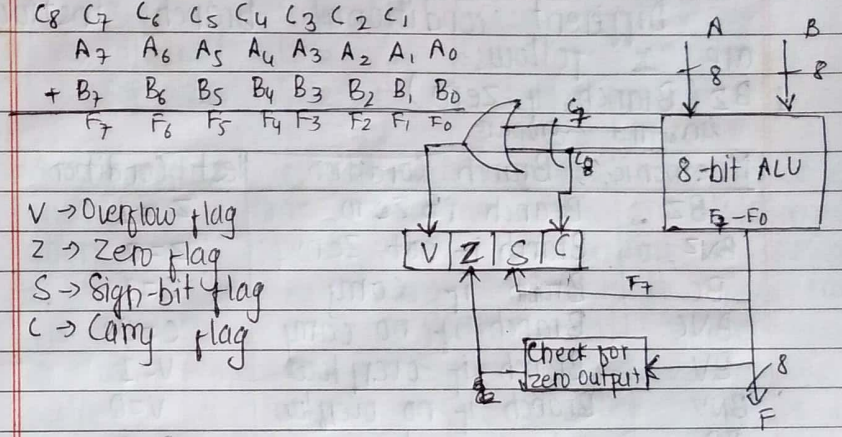
- **Program control Instruction**

Instructions are always stored in a successive memory location and when processed in CPU, the instructions are fetched from consecutive memory location & executed. Each time a instruction is fetched from memory, PC is incremented so that it contains the address of next instruction in sequence. After the execution of data transfer or data manipulation instruction control returns to fetch cycle with the PC containing the address of instruction next in sequence. On the other hand, a programmed control type of instruction

Date: / /

when executed may change address value in PC & cause the flow of control to be altered. Some typical program control instruction are branch, jump, skip, call, return, compare, etc

Status Bit Condition



V → Overflow flag
Z → Zero flag
S → Sign-bit flag
C → Carry flag

Fig. above shows the block diagram of an 8-bit ALU with 4-bit status register. The 4 status-bits are symbolized by C, S, Z & V. The bits are set or cleared as a result of an operation performed in the ALU.

1. Bit C (carry) is set to 1 if the end carry C8 is 1. It is cleared to 0 if the carry bit is 0.
2. Bit S (Sign) is set to 1 if the highest ordered bit F7 is 1. & it indicates the negative number.

Date: / /

3. Bit Z (zero) is set to 1 if the output of ALU contains all zeros.
4. Bit V (overflow) is set to 1 if the exclusive OR of the last two carys is equal to 1 & cleared to 0 otherwise.

Conditional Branch Instruction

Different conditional branch instruction are as follows:

1. BZ (Branch if zero)

Unsigned Compare

Mnemonic	Branch Condition	Test Condition
BZ	Branch if zero	Z = 1
BNZ	Branch if not zero	Z = 0
BC	Branch if cary	C = 1
BNC	Branch if no cary	C = 0
BV	Branch if overflow	V = 1
BNV	Branch if no overflow	V = 0
BP	Branch if plus	S = 0
BM	Branch if minus	S = 1
BHI	Branch if higher	A > B
BHE	Branch if higher or equal	A ≥ B
BLO	Branch if Lower	A < B
BLOE	Branch if lower or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B

Signed Compare

BZT	Branch if greater than	A > B
BZE	Branch if grater or equal	A ≥ B
BLT	Branch if less than	A < B
BLE	Branch if less or equal	A ≤ B

Date: Jan 28 /

BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B

Jan 28

- Subroutine call & return
- Program Interrupt
- Types of Interrupt
  - -> External Interrupt
  - -> Internal Interrupt
  - -> Software Interrupt

A subroutine is a self-contained sequence of instruction that performs a given computation task. During the execution of program, a subroutine may called to perform its functions many times at various point in main program. Each time subroutine is called, a branch is executed to the beginning of subroutine to start executing its set of instruction and after the subroutine has been executed, a branch is made back to the main program. The instruction that transfer program control to a subroutine is known by different names i.e. call subroutine, jump to subroutine, branch to subroutine. A subroutine instruction is executed by performing 2 operation:-

1. The address of next instruction available in PC (the return address) is stored in a temporary location so that the subroutine

Date: / /

knows where to return.
2. Control is transferred to the beginning of the subroutine.

Program Interrupt

Program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request. Control returns to the original program after the service program is executed. The interrupt procedure is in principle quite similar to a subroutine call except for three variation

1. The interrupt is usually initiated by an internal or external signal rather than from the execution of an instruction.
2. The address of an interrupt service program is determined by the hardware rather than from the address field of an instruction.
3. An interrupt procedure usually stores all the information necessary to define the state of the CPU register rather than storing only the PC.

There are three types of interrupt:-

1. External Interrupt

It come from I/O devices, from a timing device, from a circuit monitoring the power supply or from any

Date: / /

other external source. For eg. interrupt from I/O device requesting transfer of data, power failure, elapsed time of an event, etc.

2. Internal Interrupt

It arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called trap. For e.g: internal error condition are register overflow, attempt to divide by 0, an invalid opcode, stack overflow, etc.

3. Software Interrupt

It is initiated by executing an instruction. Software interrupt is a special called instruction that behaves like an interrupt rather than a subroutine call. It can be used by the programmer to initiate an interrupt procedure at any desired point in the program.

- RISC (Reduce Instruction Set Computer)

1. Relatively few instruction
2. Relatively few addressing mode
3. Memory access limited to load & store instruction
4. All operations done within the register of CPU.
5. Fixed length easily decoded instruction format.
6. Single cycle instruction execution.

Date: / /

7. Hardwired rather than microprogrammed control.
8.

- CISC (Complex Instruction Set Computer)

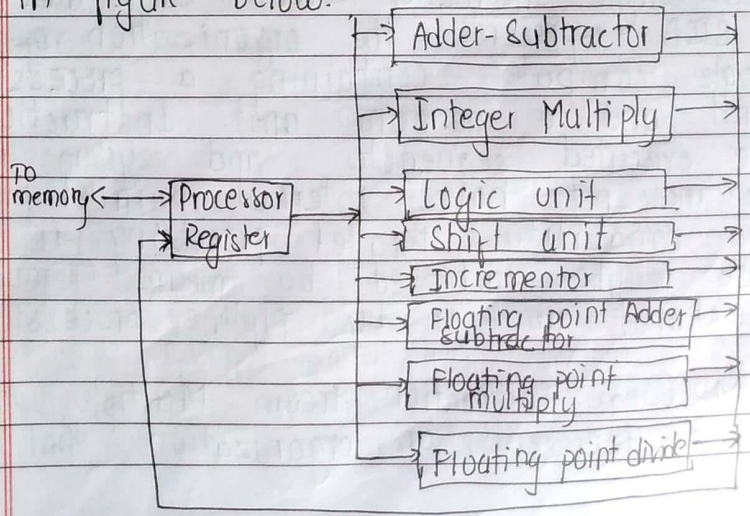
1. A large no. of instruction
2. Some instructions that perform specialized tasks are used in frequently.
3. A large variety of addressing mode.
4. Variable length instruction format.
5. Instruction that manipulates operand in memory.

Date: Feb./2 /

UNIT 7

PARALLEL PROCESSING

Parallel processing is a term used to denote a large class of technique that are used to provide simultaneous data process task for the purpose of increasing computational speed of a computer system. Instead of processing each task sequence as on a conventional computer, a parallel processing system is able to perform concurrent data processing to achieve faster execution time. The purpose of parallel processing is to speed up computer processing capability and increase its throughput. i.e. the amount of processing that can be accomplished during a given interval of time. The amount of hardware increase the cost of computing. For a parallel processing, multiple functional unit are used, as shown in figure below:



Date: / /

- Flynn's Classification

There are variety of ways that parallel processing can be classified. It can be classified from the internal organization of process, from the inter connection structure between or from the flow of information through the system.

One classification introduced by MJ Flynn consider the organization of computer system by the instruction and data that are manipulated simultaneously. The sequence of instruction read from memory constitutes an 'Instruction Stream' and the operation performed in the data processor constitutes a data stream.

On the basis of instruction stream & data stream Flynn's classified the computer into four major categories:-

1. SISD (Single Instruction Stream, Single data Stream)

SISD represents the organization of a single computer containing a processor unit and a memory unit. Instructions are executed sequencly and system may or may not have internal parallel processing capability. Parallel processing in this case maybe achieved by means of multiple functional unit or by pipeline processing.

2. SIMD (Single Instruction Stream Multiple Data stream)

It represents an organization that includes many processing unit under the supervision of common control unit. All processors receive the same instruction from the control unit but operate on different items of data.

3. MISD (Multiple Instruction Single Data stream)

It is only of theoritical interest since no practical system has been constructed using this organization.

4. MIMD (Multiple Instruction Multiple Data stream)

It refers to a computer system capable of processing several program at the same time. Most multiprocessor and multicomputer system can be classified in this category.

Techique of parallel processing

1. Pipelining
2. Vector Processing
3. Array Processing

1. Pipelining

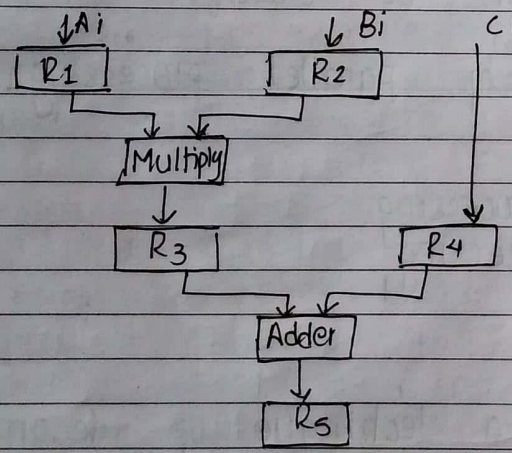
It is a technique of decomposing a sequential process into sub operation with each sub process being executed in a special dedicated segment that operate concurrently with all other segment. Each segment perform partial processing detected by the way the task is partation. The result

Date: / /

obtained from the computation in each segment is transferred to the next segment in the pipeline. The final result is obtained after the data have passed through all segments. The overlapping of computation is made possible by associating a register with each segment in the pipeline. For eg:-

$A_i * B_i + C_i$ for $i = 1, 2, \ldots, 7$

$R_1 \leftarrow A_1$
$R_2 \leftarrow B_1$
$R_3 \leftarrow A_i * B_i$
$R_4 \leftarrow C_i$
$R_5 \leftarrow A_1 * B_i + C_i$



Clock Pulse	Segment 1		Segment 2		Segment 3
	R_1	R_2	R_3	R_4	R_5
1	A_1	B_1	-	-	-
2	A_2	B_2	$A_1 * B_1$	C_1	-

Date: Feb 9 /

3	A_3	B_3	$A_2 + B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 + B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 + B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 + B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 + B_6$	C_6	$A_5 * B_5 + C_5$
8	A_8	B_8	$A_7 + B_7$	C_7	$A_6 * B_6 + C_6$
9	A_9	B_9	$A_8 + B_8$	C_8	$A_7 * B_7 + C_7$

Feb 9 - General Consideration

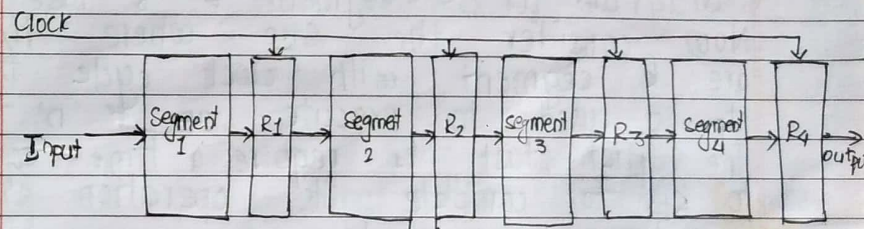


fig: Four Segment Pipeline

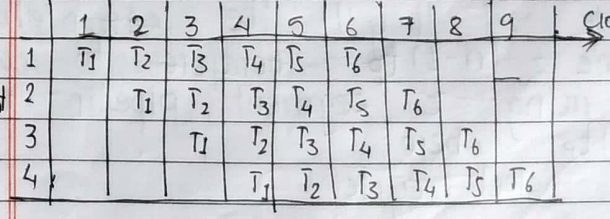


fig: Space Time Diagram

Date: / /

Any operation that can be decompose into sequence of sub-operation of about the same complexity can be implemented by a pipeline operation. The general operation of ~~a~~ pipeline is illustrated in fig. above where the operand pass through all 4 segments in a fixed sequence. Each segment consist of combinational circuit Si that performs a sub operation over the data stream flowing through the pipe. The segment are separated by register Ri - behavi of pipeline is illustrated in space time diagram for 4 segment & 6 task.

Now consider the case where there are k segment with clock cycle time t_p is used to execute 'n' task. The first task t_1 requires a time equal to kt_p to complete its operation since there are k segment.

The remaining n-1 task emerge from the pipe at the rate of one task per clock cycle & they'll be completed after time = $(n-1)t_p$. Therefore to complete 'n' task using k segment pipeline requires $(k+n-1)t_p$, where

t_p = clock cycle

Now consider a non-pipeline unit that performs same operation & takes a time = t_n to complete each task. Therefore the total time required for 'n' task is nt_n.

Date: / /

The speed up of pipeline processing over an equivalent non-pipeline processing is defined by the ratio,

$$S = \frac{\text{non-pipeline}}{\text{pipeline}}$$

$$= \frac{nt_n}{(k+n-1)t_p}$$

As the no. of task increases, 'n' becomes much larger than k-1 and $k+n-1$ approaches the value of n. Under this condition, speed up becomes,

$$S = \frac{nt_n}{nt_p}$$

$$= \frac{t_n}{t_p}$$

If we assume that the time it takes to process a task is the same in pipeline & non-pipeline circuit, we'll have,

$$t_n = kt_p$$

Including this assumption, the speed up reduces to,

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

$$= k$$

This shows that the theoritical maximum speed up that a pipeline can provide is k, where k is no. of segment in the pipeline.

Date: Feb /10

Q. There are 4 segments for computing 100 number of tasks. The time required to complete each sub-tasks in each segment t_p is 20 nanosecond (ns). Let us assume that to complete each task in non-pipeline system $t_n = kt_p$. Now, find out the time required to complete all the task in pipeline system, in non-pipeline system & also find out speed up ratio.

Sol:

no. of tasks (n) = 100
t_p = 20 ns in pipeline system
no. of segments (k) = 4
time taken by each task to complete in non-pipeline system $(t_n) = kt_p$
Total time to complete task in pipeline system = ?
Total time to complete task in non-pipeline system = ?
speedup ratio = ?

here,

Time required in pipeline system = $(k+n-1)t_p$
$= (4+100-1)20$
$= 2060$ ns

Time required in non-pipeline system = nt_n
$= 100 \times 4 \times 20$
$= 8000$ ns

$$\text{Speedup} = \frac{\text{Time required in pipeline}}{\text{Time required in non-pipeline}}$$

$$= \frac{2060}{8000}$$

$$= 3.88$$ //

Date: / /

- Arithmetic Pipeline

Pipeline arithmetic units are usually found in very high speed computer & they're used to implement floating point operation, multiplication of fixed-point no. & similar computation encountered in scientific problem. Floating point operation are easily decomposed into sub-operation. To understand the concept of arithmetic pipeline, let us suppose that the inputs to floating point adder pipeline are two normalize floating point binary number,

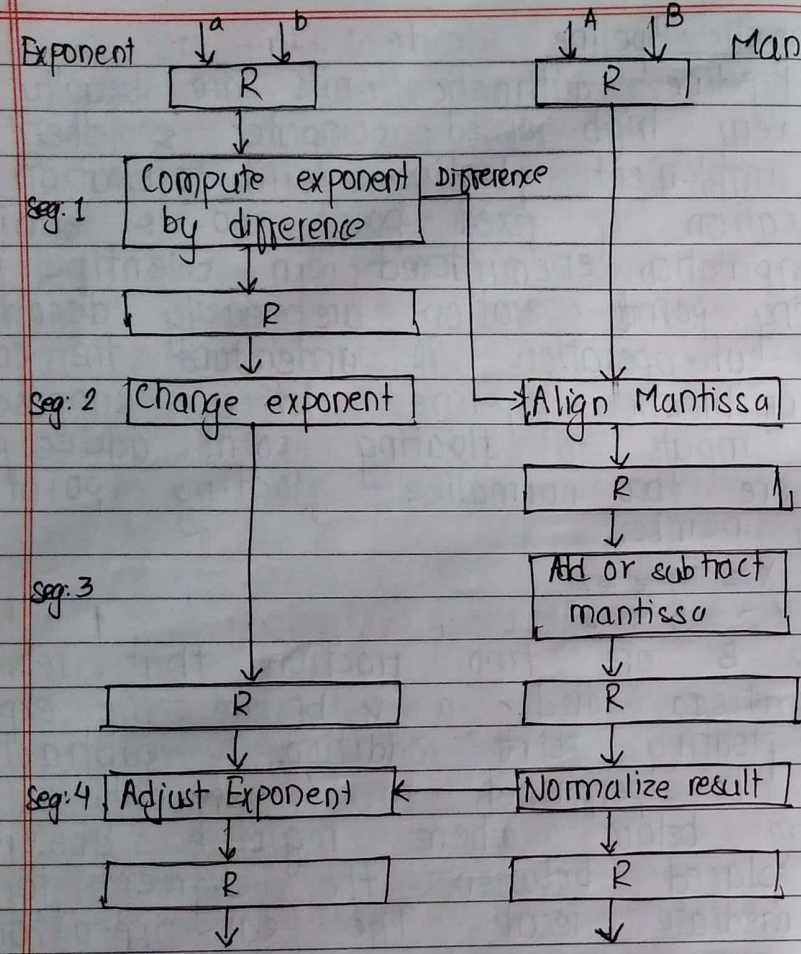
$X = A \times 2^a$
$Y = B \times 2^b$

A & B are two fraction that represent the mantissa and a & b are the exponent. The floating point addition & subtraction can be performed in 4-segment as shown in fig. below, where register labeled "R" are placed between the segment to store intermediate result. The sub operations that are performed in 4-segment are:

1. Compare the exponent
2. Align the mantissa
3. Add or subtract the mantissa
4. Normalize the result

⇒

Date: / /



eg: $X = 0.9504 \times 10^3$
$Y = 0.8200 \times 10^2$

First of all, compute the exponent by subtracting the lower exp^n by higher exp^n i.e. $3 - 2 = 1$. Now, we need to align the mantissa for Y by one position. i.e. Y = .0820 Now, the exponent is chosed to be 3. In next segment, the mantissa will be added i.e.

Date: Feb. 11/

.08200 + .9504 & the result will be

$$\begin{array}{r} .08200 \\ +\ .9504 \\ \hline 1.03240 \end{array}$$

Now, we need to normalize the result. Hence the result will be .10324 and together with this process, the exponent need to be adjusted which will increase by 1. Here total result will be 0.10324×10^4.

Feb. 11

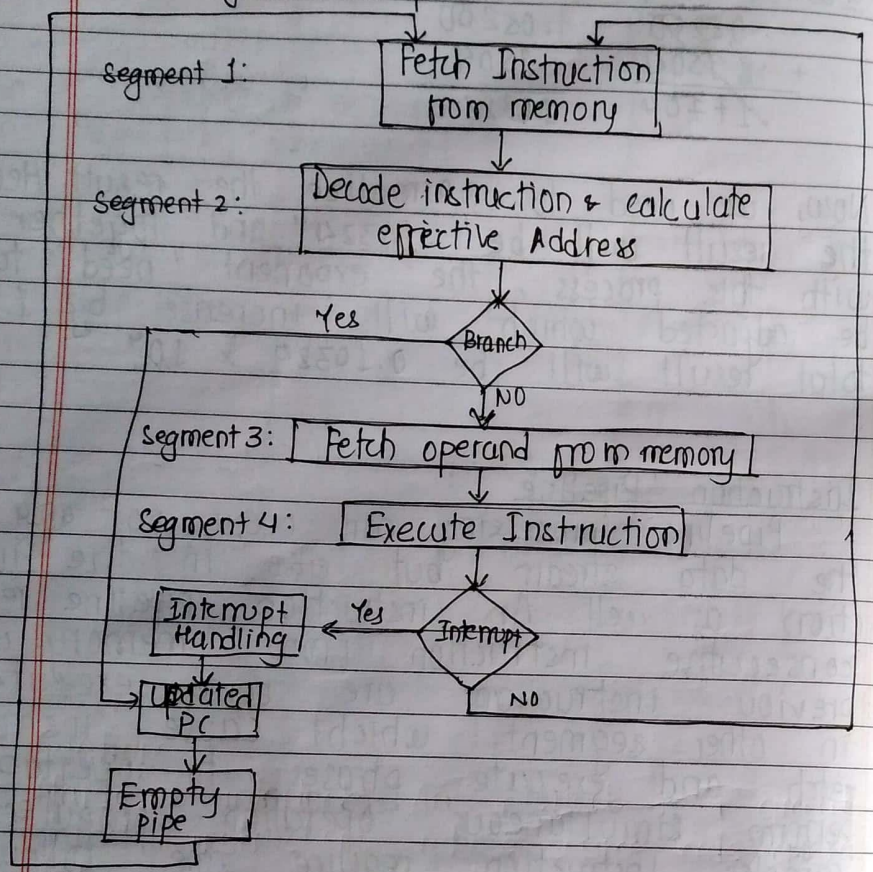
- Instruction Pipeline

Pipeline processing can occur not only in the data stream but also in the instruction as well. An instruction pipeline reads consecutive instruction from memory while previous instructions are being executed in other segment which cause instruction fetch and execute phases to overlap & perform simultaneous operation. Computer with complex instruction require the following sequence of steps:-

1. Fetch the instruction
2. Decode the instruction
3. Calculate the effective address
4. Operand fetch
5. Execute the instruction
6. Store the result in proper place

Date: / /

Four segment Pipeline



Steps	1	2	3	4	5	6	7	8	9	10	11	12	13
1	F_I	DA	F_O	Ex									
2		F_I	DA	F_O	Ex								
Instruction (Branch) 3		F_I	F_I	DA	F_O	Ex							
4				F_I	-	-	F_I	DA	F_O	Ex			
5								F_I	DA	F_O	Ex		
6									F_I	DA	F_O	Ex	
7										F_I	DA	F_O	Ex

Space Time Diagram

Date: / /

Fig. above shows the instruction cycle in a CPU that can be processed with 4 segment. While an instruction is being executed in segment 4, the next intruction in sequence is busy fetching an operand from memory in seg. 3 & so on. The six stages of instruction cycle are computed in 4 segment & they're represented with an abbreviate symbol :-

1. F_I is the segment that fetches an ins
2. DA is " , that decodes the instructio & calculate the effective address
3. F_O is the segment that fetches the operand.
4. Ex is " " " executes the instruction & place the result in proper place.

The space time diagram above illustrate how the 7 diff. instructions are executed using 4 segment pipeline. It is observed that there's branching in instruction 3. and it is noticed only after the DA segment when there is a branch, the PC is update to service the branch & the pipe is made empty. The instruction 4 i.e. fetched in step 4 is again fetched in step 7 after the branch instruction is totally execute

JMP (Group C) Date: / /

- Hazards of instruction pipeline / Pipeline Conflicts (Solution, read from book)

1. Resource Conflict
2. Data Dependency
3. Branch Conflict

1. Caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instructions & data memories.

2. This conflict arises when an instruction depends on the result of previous instruction but this result is not yet available.

3. This conflict arise from branch & other instruction that changes the value of PC.

Solutions

1. Hardware interlocks: An interlock is a circuit that detects instructions whose source ~~codes~~ operands are destinations of instruction in pipeline. Detection of this situation causes the instruction whose source isn't available to be delayed.
2. Operand forwarding: It uses a special hardware that detects a conflict & avoid it by routing the data for e.g: instead of transferring ALU result in destination register, it firstly checks the destination operands if it is needed as a source for next instruction, it directly passes result to ALU, bypassing the register file.
3. Loop buffer: variation of BTB; small very high speed register file maintained by ins. fetch seg. in pipeline. When loop pattern detected, it is stored in loop buffer including all branches.
4. Branch prediction:

Date: Feb / 16 /

UNIT 8 COMPUTER ~~ARCHITECTURE~~ ARITHMETIC

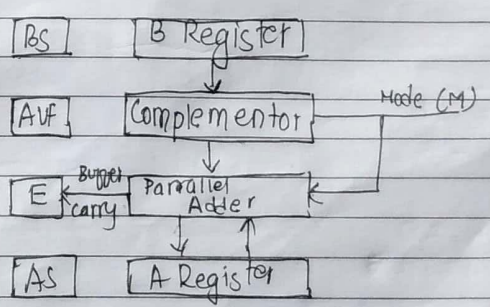
Addition & Subtraction
- Signed magnitude
- 2's complement

Multiplication
- Signed magnitude
- 2's complement (Booth)

- Signed Magnitude Addition & Subtraction:

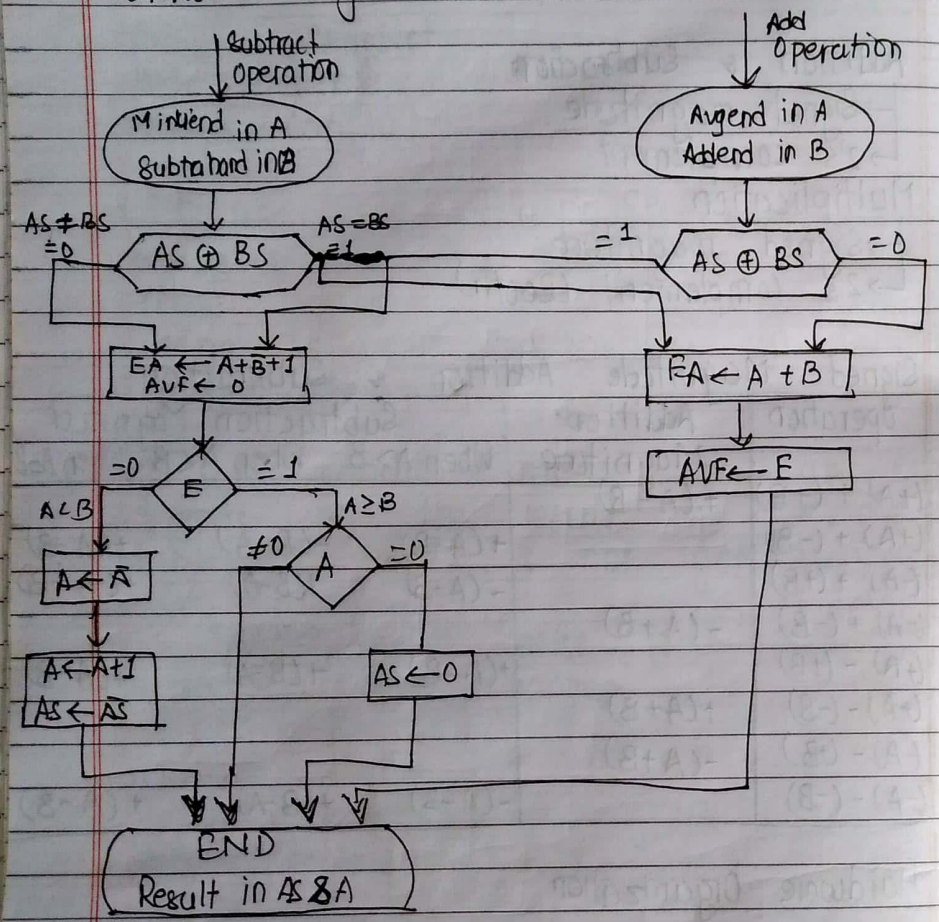
Operation	Addition Magnitude	Subtraction Magnitude When A>B	When A<B	When A=B
(+A) + (+B)	+(A+B)			
(+A) + (-B)		+(A-B)	-(B-A)	+(A-B)
(-A) + (+B)		-(A-B)	+(B-A)	+(A-B)
(-A) + (-B)	-(A+B)			
(+A) - (+B)		+(A-B)	+(B-A)	+(A-B)
(+A) - (-B)	+(A+B)			
(-A) - (+B)	-(A+B)			
(-A) - (-B)		-(A-B)	+(B-A)	+(A-B)

- Hardware Organization



Date: / /

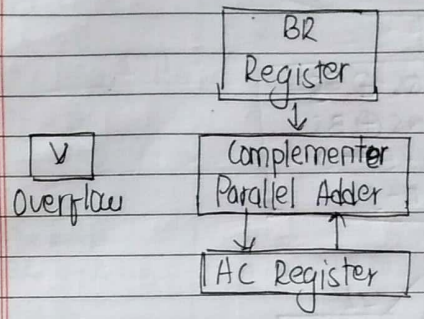
- Hardware Algorithm

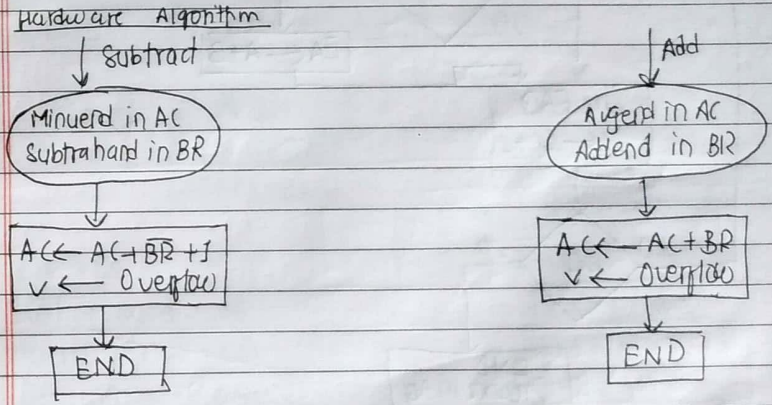


Date: Feb 17

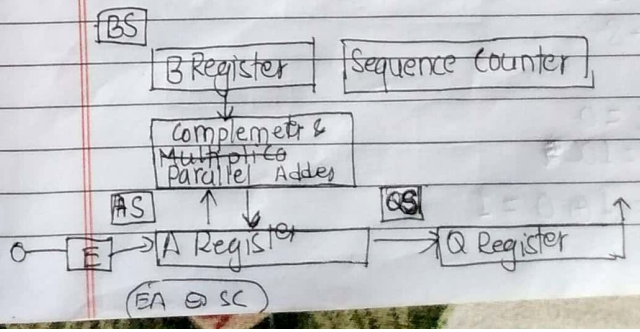
- Addition & subtraction of signed 2's Data

Hardware organization



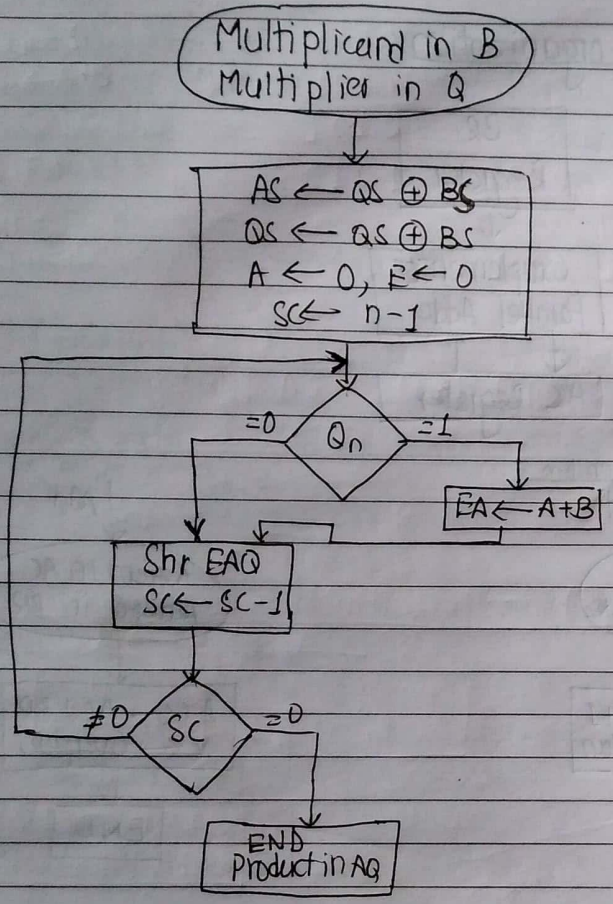


- Signed magnitude multiplication Hardware Organization



Date: / /

- Hardware Algorithm



10. -10 × +8

Multiplicand = -10

Multiplier = +8

BS = 1 QS = 0

B = 1010 Q = 1000

AS = BS ⊕ QS = 1 ⊕ 0 = 1

= QS ⊕ BS

Date: / /

Multiplicand B = 1010	E	A	Q	SC
Qn = 0	0	0000	1000	100
Shr EAQ	0	0000	0100	011
Qn = 0	0			
Shr EAQ	0	0000	0010	010
Qn = 0	0			
Shr EAQ	0	0000	0001	001
Qn = 1		0000		
Shr EAQ	0	+1010		
EA ← A+B	0	1010	0001	
		0101		
Shr EAQ	0	1010	0000	000

Result = AQ

= 10100000

= 01010000

2. +8 × +5

Multiplicand = +8

Multiplier = +5

BS = 0 QS = 0

B = 1000 Q = 0101

AS = QS ⊕ BS = 0 ⊕ 0 = 0

Multiplicand B = 1000	E	A	Q	S
	0	0000	0101	100
Qn = 1	0	0000	0101	100
EA ← A+B	0	0000		
		+1000		
		1000		
ShrEAQ	0	0100	0010	011

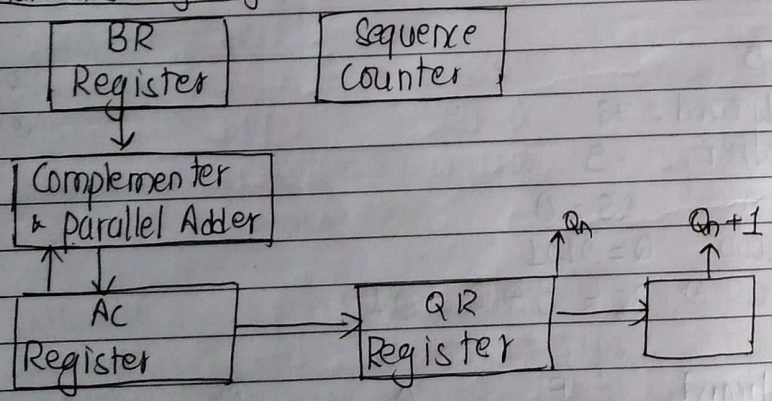
Date: / /

Qn = 0	0	0010	0001	010
shr EAQ				
Qn = 1		0010		
EA ← A+B	0	1000		
		1010		
shr EAQ	0	0101	0000	001
Qn = 0				
shr EAQ	0	0010	1000	000

Result AQ = $000\overset{5}{1}01\overset{3}{0}00$

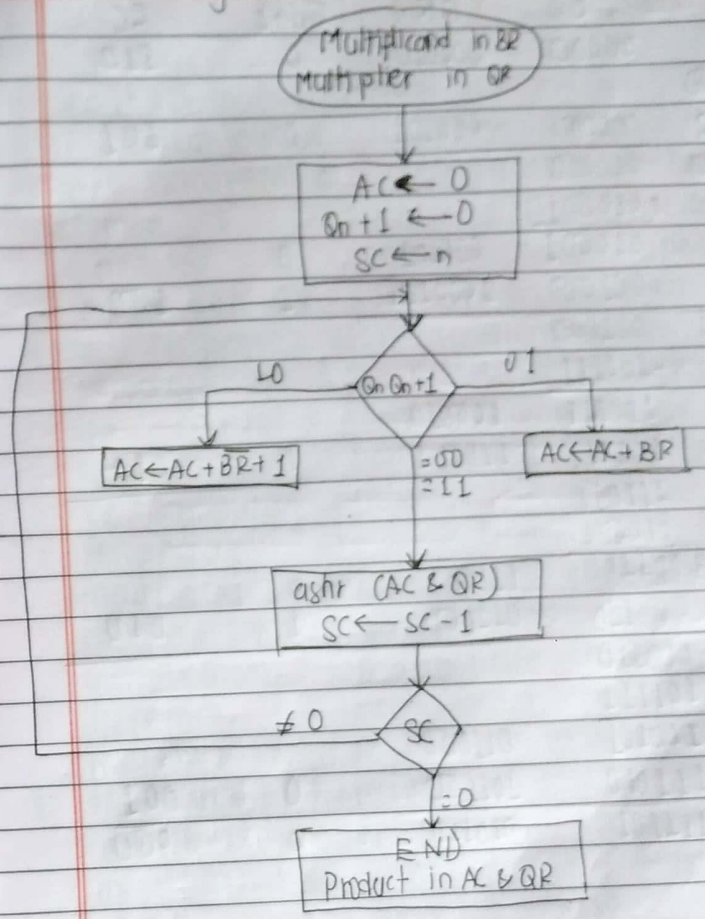
- 2's Complement Multiplication (Booth)

Hardware Org organization



Date: / /

Hardware Algorithm



- -17 x 10
  -20 x -10

+17 = 010001 ..... > 2's complement of +17

BR = 101111 QR = 001010

$\overline{BR} + 1 = 010001$

Date: / /

	AC	QR	Qn+1	SC
	000000	001010	0	110
$Q_n Q_{n+1}=00$				
Ashr ACQR	000000	000101	0	101
$Q_n Q_{n+1}=10$	000000			
$AC \leftarrow AC+\overline{BR}+1$	+ 010001			
	010001	000101	0	
Ashr ACQR	001000	100010	1	100
$Q_n Q_{n+1}=01$	001000			
$AC \leftarrow AC+BR$	+ 101111			
	110111	100010		
Ashr ACQR	111011	110001	0	011
$Q_n Q_{n+1}=10$	111011			
$AC \leftarrow AC+\overline{BR}+1$	010001			
	×(1)001100	110001	0	
Ashr ACQR	000110	011000	1	010
$Q_n Q_{n+1}=01$	000110			
$AC \leftarrow AC+BR$	101111			
	110101	011000		
Ashr ACQR	111010	101100	0	001
$Q_n Q_{n+1}=00$	111101	010110	0	000
Ashr ACQR				

Product = ACQR

Result = 111101010110

2's complement = $-(000010101010) = -(128+32+8+2) = -170$

$= -170$ //

Date: Feb. 17 /

UNIT 9 INPUT- OUTPUT ORGANIZATION

- Input-Output Interface

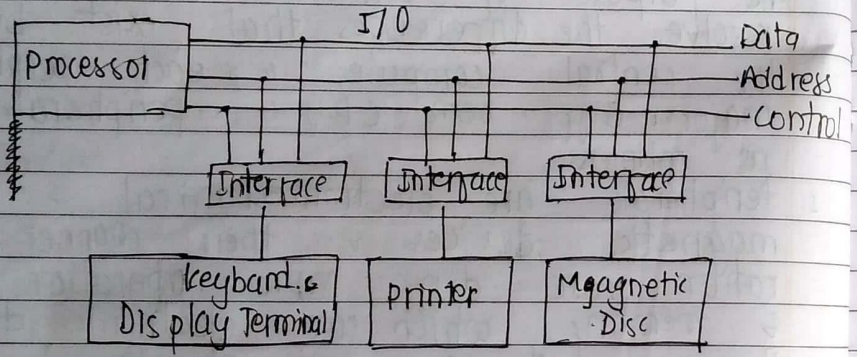
Input-output interface provide a method for transferring information betn internal storage & external I/O devices. Peripherals connected to comp. need special communication link for interfacing them with the CPU. The purpose of communication link is to resolve the differences that exist between the central computer & each peripheral. The major diff. betn C.P.U & peripherals are as follows:-

1. Peripherals are electromechanical & electromagnetic devices & their manner of operation are diff. from operation of CPU & memory which are electronic devices. Therefore, a conversion of signal value may be required.
2. The data transfer rate of peripheral is usually slower than the transfer rate of CPU & consequently a synchronization mechanism may be needed.
3. Data codes & format in peripheral differ from the word format in CPU & memory.
4. The operating mode of peripheral are diff. from each other & each must be controlled so as not to disturb the operation of other.

Date: / /

To resolve these diff., a computer system includes special hardware component betn CPU & peripheral to supervise & sychronize all I/O transfer & this component is ~~compon~~ commonly known as I/O interface.

- I/O Bus & Interface Modules



A typical communication link between the processor & several peripheral shown in figure above. The I/O Bus consists of data line address line & control line. Each peripheral device has associated with an interface unit. Each interface decode the address & control received from I/O bus, interprets them for the peripheral & provide signal for peripheral controller. It also sychronize the data flow & supervises the transfer betn peripheral & processor. To communicate with particular device, the processor places device address on address line, which is monitored by

Date: / /

each interface & when interface detects its own address, it activate the path betn bus line & device that it control. All peripherals whose address don't correspond to address in bus are disabled by their interface. The interface selected respond to function code & proceed to execute it. The function code is also referred to as an I/O command & there're four types of I/O command as follow:- (Function code)

1. Control command: It is used to activate the peripheral & to inform it what to do.

2. Status Command: It is used to test various status condition in interface & peripheral. For e.g: computer may wish to check the status of the peripheral before transfer is initiated.

3. Output Data Command: It causes the interface to respond by transferring data from the bus into one of its register.

4. Input Data Command: It is opposite of the data output command & in this case, the interface receives an item of data from peripheral & places in its buffer register.

Date: Feb/18

- I/O versus Memory Bus

In addition to communicating ~~with~~ I with I/O, the processor must communicate with memory unit. Like the I/O bus, the memory bus contain data, address, read write control lines. There are 3 ways that computer buses can be used to communicate with memory & I/O which are as follows:-

1. Use two separate buses, one for memory & other for I/O.
2. Use one common bus for both memory & I/O but have separate control lines for each.
3. Use one common bus for memory & I/O with common control lines.

In the first method, the computer has independent set of data, address, & control buses, one for accessing memory & the other for I/O. This is done in computer that provides a separate I/O processor in addition to CPU. The memory communicate with both CPU & IOP through a memory bus & the IOP also communicate with I/O device through a separate I/O bus.

Date: / /

- Isolated versus Memory mapped I/O

Many computer use one common bus to transfer information betn memory or I/O & CPU. The distinction betn a memory transfer & I/O transfer is made through separate read & write lines. The I/O read & I/O write control lines are enabled during an I/O transfer whereas memory read & memory write control lines are enabled during memory transfer. This configuration isolates all I/O interface addresses from the addresses assigned to memory & is referred to as Isolated I/O. The other alternative is to use the same address space for both memory & I/O. This is the case in computer that employ only one set of read & write signal & don't distinguish betn memory & I/O addresses. This configuration is known as memory mapped I/O.

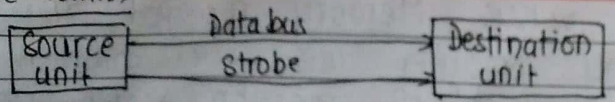
- Asychronous Data Transfer

Asychronous data transfer between two independent unit requires that ~~control~~ control signal be transmitted between the communicating unit to indicate the time at which data is being transmitted. There're two diff means of transmitting a data in asychronous data transfer:

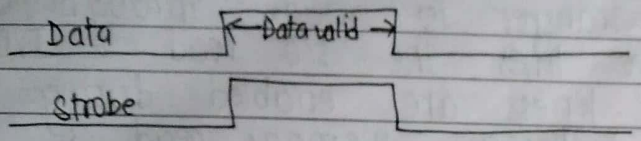
1. Strobe control
2. Handshaking

Date: / /

1. Strobe Control



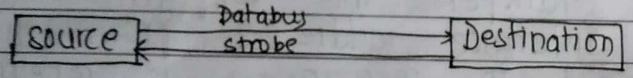
(i) Block diagram



(ii) Timing diagram

fig: Source Initiated

Source Control



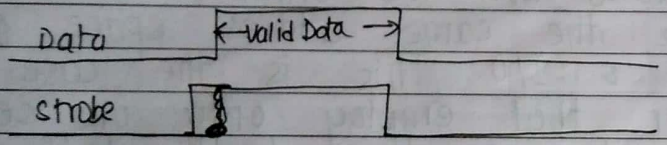


fig: Destination Initiated

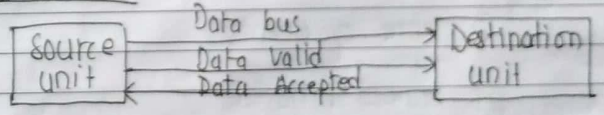
The strobe control method of data transfer employs a single control line to time each other. The strobe maybe activated by the source or destination unit which is illustrate in figur above. The databus carries binary info from source unit to destination unit & the strobe is a single line that informs the destination unit when valid word is available in the bus incase of source initiated data transfer whereas for distinated initiated data transfer, the strobe is activated by the destination after which the source will place valid data in data bus.

Date: / /

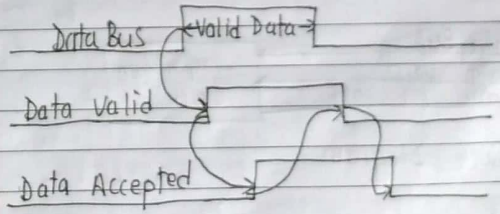
2. Handshaking

The disadvantage of strobe method is that the source unit that initiate the transfer has no way of knowing whether the destination unit has actually received data item that was placed in bus. similarly a destination unit that initiate the transfer has no way of knowing whether the source unit has actually placed data on the bus. The handshake method solve this problem by introducing a second control signal that provides a reply to the unit that initiate the transfer. The principle of handshaking is illustrate below:-

source initiated



(i) Block Diagram

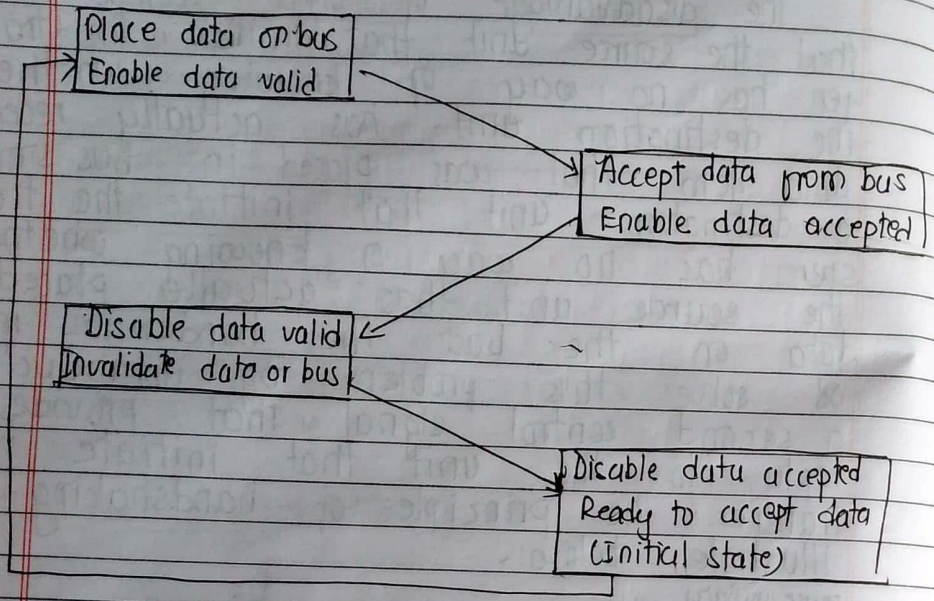


(ii) Timing Diagram

fig: source initiated

Date: / /

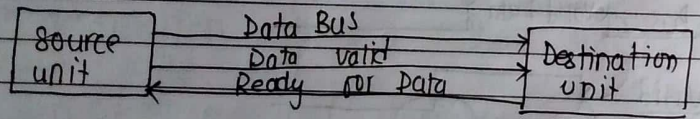
Source unit | Destination unit



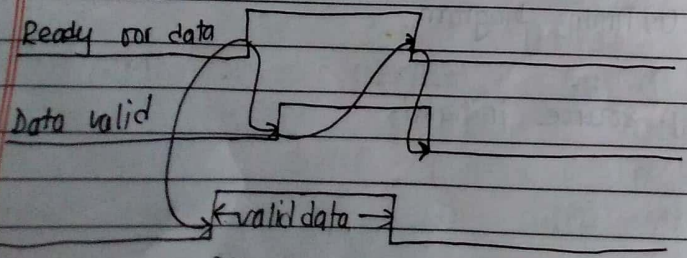
Sequence of events

fig. Source initiated

Destination initiated



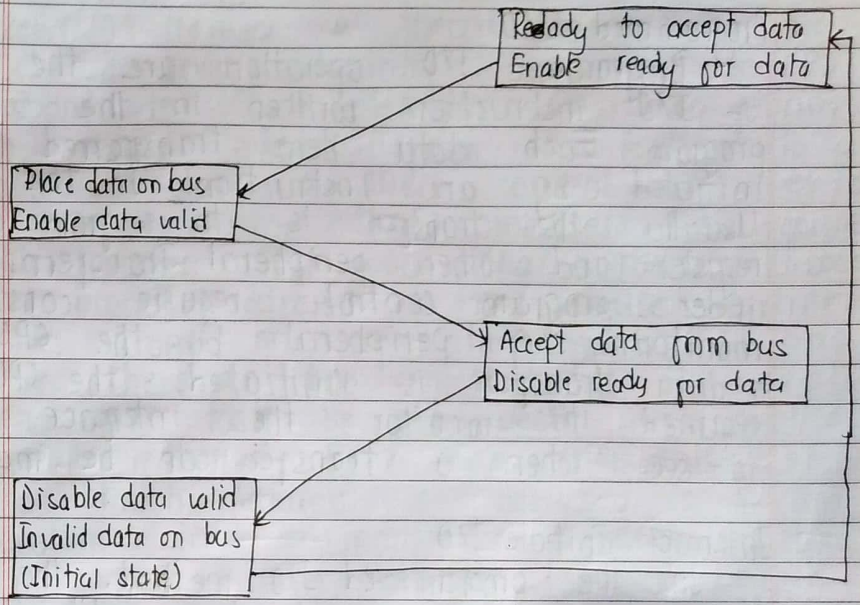
(i) Block Diagram



(ii) Timing diagram

Date: / /

Source unit | Destination unit



Sequence of Events

fig: Destination initiated

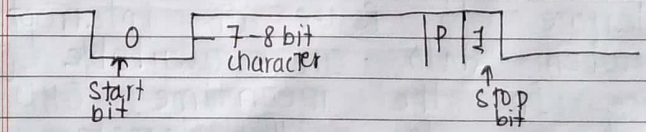


fig: ~~Asynchr~~ Asynchronous Data Transfer

Date: / /

- Mode of Transfer

1. Programmed I/O

Programmed I/O operation are the result of I/O instruction written in the computer program. Each data item transferred is initiated by an instruction in the program. Usually the transfer is to & from a CPU register and other peripheral. Transferring data under program control require constant monitoring of peripheral by the CPU. Once a data transfer is initiated, the CPU is required to monitor the interface, ~~when~~ to see when a transfer can be made again.

2. Interrupt Initiate I/O

In the programmed I/O method, the CPU stays in a programmed loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process since it keeps the processor busy needlessly. It can be avoided by using an interrupt facility and special command to inform the interface to issue an interrupt request signal when data are available from the device. In the meantime, CPU can proceed to execute another program. In this method, when the interface determine that the device is ready for data transfer, it generates an interrupt request to the computer. Upon detecting the external interrupt signal, the CPU momentarily stops the task it is processing, branches to service program to process I/O transfer & they returns to the task it was originally performing. The CPU respond to the interrupt signal by storing the return address from the PC into a memory stack & then control branches to a service routine that processes the required I/O transfer. The way that processor chooses the branch address of the service routine varies from one unit to another. In principle, there're two methods for accomplishing this task i.e. non-vector interrupt & vector interrupt. In a non-vector interrupt, the branch address is assigned to a fixed location in memory. In a vector interrupt, the source that interrupts supplies branch info. to the computer & this info. is called interrupt vector.

Date: / /

- Priority Interrupt

In a typical application, a no. of I/O devices are attached to computer with each device being able to originated an interrupt request. The first task of interrupt system is to identify the source of interrupt. There's also the possibility that, several sources will request service simultaneously (at the same time). In this case, the system must also decide which device to service first. Priority Interrupt is a system that establishes a priority over the various

Date: / /

sources to determine which condition is to be serviced when two or more request arrives simultaneously. Higher priority interrupt levels are assigned to request which, if deleted or interrupted could have serious consequences. Establishing the priority of simultaneous interrupt can be done by software or hardware as follows:-

(i) Polling: A polling procedure is used to identify the highest priority source by software means. In this method, there's one common branch address for all interrupt, the program that takes care of interrupt begins at branch address & polls interrupt sources in sequence. The order in which they're ~~tasted~~ tested determine the priority of each interrupt. The highest priority source is tested first & if its interrupt signal is on, control branches to service routine for this source otherwise the next lower priority source is tested & so on. The disadvantage of software (polling) method is that, if there're many interrupts, the time required to poll them can exceed the time available to service I/O device. In this situation, the hardware priority interrupt can be advantage to speed up the operation

Date: Mar. 3 /

- Hardware Priority Interrupt

This unit functions as an overall manager in an interrupt system environment. It accepts interrupt request from many sources determine which of the incoming request has the highest priority & issues an interrupt request to computer based on this determination. This function can be established by either a serial or parallel connection of interrupt lines on the basis of which it is categorized as

(i) Daisy Chaining Priority
(ii) Parallel Priority Interrupt.

Mar 3

(i) Daisy Chaining Priority

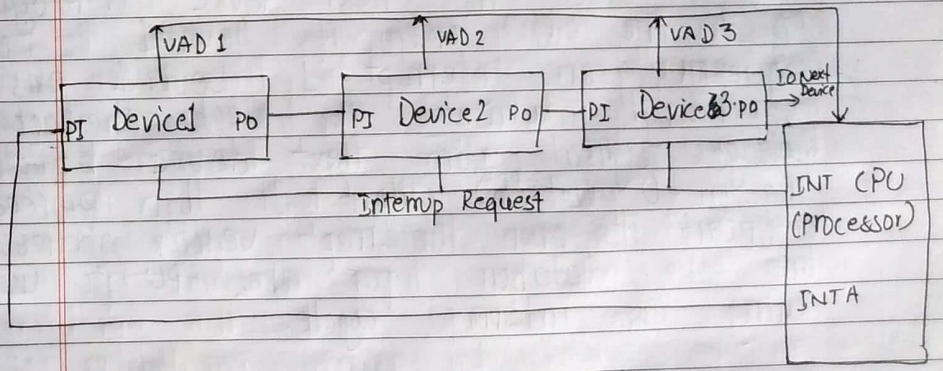


fig: Daisy Chaining Priority

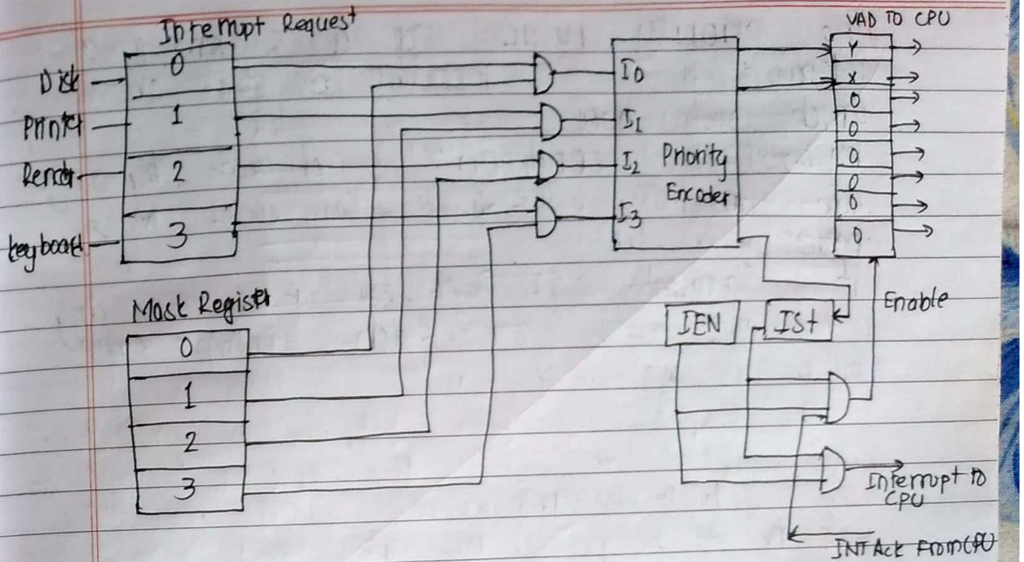
The daisy chaining method of establishing priority consist of serial connection of all device that request an interrupt. The device

Date: / /

with highest priority is placed at first position followed by lowered priority device upto the device with the lowest priority which is placed last in the chain. Fig. above illustrate daisy chaining priority where 3 devices are connected. The interrupt request line is common to all devices & forms a wired logic connection. If any device has its interrupt signal in low level state, the interrupt line goes to low level state & enables the interrupt input in CPU. The CPU responds to an interrupt request by enabling the interrupt acknowledge line. The signal is received by Device 1 at its PI (Priority Input input), the acknowledge ~~system~~ signal passes the next device through PO (Priority Out) only if Device 1 is not requesting an interrupt. If Device 1 has a pending interrupt, it blocks the acknowledge signal from next device by placing a 0 in PO output. It then proceeds to insert its own interrupt vector address into the databus for the CPU to use during an interrupt cycle.

ii) Parallel Priority Interrupt

Date: / /



In daisy chaining, the highest priority is given to the device that receives inter acknowledge signal from CPU. The further the device is its proprity. The lower priority device has to wait for long period of time to get the resources and this problem can be resolved by using the parallel priority interrupt. Parallel priority uses a reger whose each [illegible] by the interrupt device. Priority is enabled according to in the position

a mask ~~reg~~ register whose to contro & the status of an

It can also [illegible] a highest device to

Date: / /

The priority logic for a system of 4 interrupt is show in fig. above which construction by external condition & cleared by program instruction. The mask has same bit as in the.
it is possible, to set of any each interrupt bit is 0.

In this way, an re is set to 1.

The two bits of Vector address.

In the daisy chaining priority the highest priority is given to the device that receives the interrupt aknowledge signal from the CPU, the farther the device is from the first position, the lower is its priority that means the lower priority device has to wait for long interval of time to get the resources and this problem can be solved by using parallel priority interrupt. The parallel priority interrupt method uses a register whose bits are set separately by the interrupt signal from each device. Priority is established according to the position of the bits in the register. In addition to

Date: 4 Mar /

the interrupt register, the circuit may include a mask register whose purpose is to control the status of each interrupt request. The mask register can be programmed to disable lower priority interrupt while a higher priority device is being serviced. It can also provide a facility that allows a high priority device to interrupt the CPU while a lower priority device is being serviced. The priority logic for a system of a interrupt source is shown in figure above which consist of an interrupt register whose individual bits are set by external condition & cleared by a program instruction. The mask register has the same number of bits as in the interrupt register. By means of program instruction it is possible to set or reset any bit in the mask register. Each interrupt bit and its corresponding mask bit are applied to an AND gate to produce the Y inputs to a priority encoder. In this way, an interrupt is recognised only if its corresponding mask bit is set to 1 by the program. The priority encoder generates n-bit of vector address which is transferred to the CPU.

4 Mar

- DMA (Direct Memory Access)

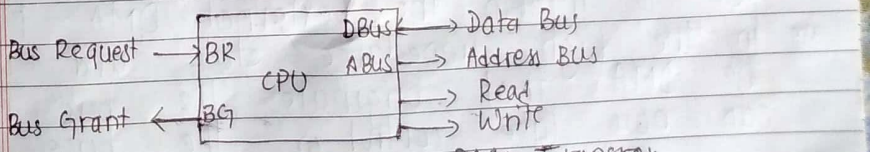


fig: CPU signal for DMA Transfer

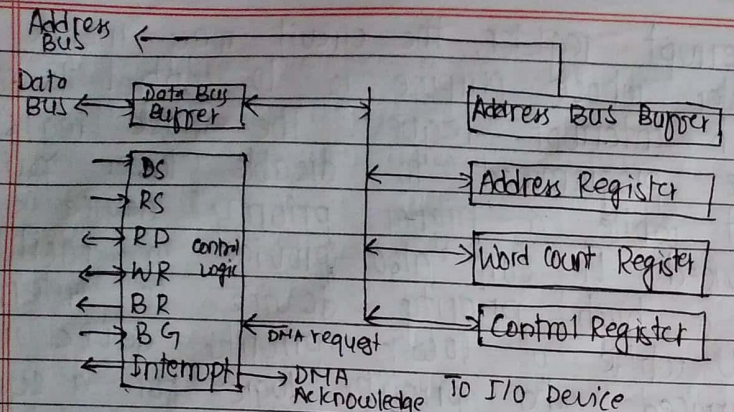


fig: DMA Controller

The transfer of data between a fast storage device such as magnetic disc and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called Direct Memory Access. During DMA transfer, the CPU is idle and has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between I/O devices and memory. Fig above shows two control signal in the CPU that facilitate the DMA transfer. The Bus Request (BR) input is used by the DMA controller to request the CPU to relinquish control of the buses. The CPU activates the Bus Grant (BG) output to inform the external DMA that the buses are in the high impedance state and then after the DMA that originate the bus request can now take control of the buses to conduct memory transfer without processor intervention. When the DMA terminates the transfer, it disables the bus request line. The CPU disable the be BG, takes control of the buses & returns to its normal operation. When the DMA takes control of the bus, it communicates directly with the memory & the transfer can be made in two different approach ie. ~~Brust~~ Burst Transfer & Cycle stealing. In a burst transfer, a block sequence consisting of a number of memory word is transferred in a continuous burst. while the DMA controller is master of the memory buses. In cycle stealing, the DMA controller transfer one data word at a time after which it must return control of the buses to the CPU.

(i) DMA Controller

Fig. above shows the block diagram of typical DMA controller. The DMA controller needs the usual circuits of an interface to communicate with the CPU & I/O devices. In addition, it needs an AR, a word count register & a set of address line. The AR & address lines are used for direct communication with the memory. The word count register specifies the number of words that must be transferred. The DMA controller communicate with CPU via data bus & control lines. The registers in DMA are selected by the CPU through the address bus by enabling the DS

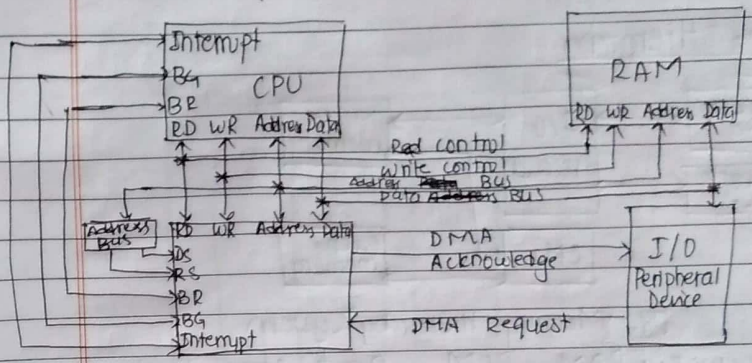
Date: / /

(DMA select) & RS (Register select) inputs. The RD & WR inputs are bidirectional. When the BG input is zero the CPU can communicate with the DMA Reg through the data bus to read from or write DMA register. When BG = 1, the CPU has relinquished the buses & the DMA can communicate directly with the memory by specifying an address in the address bus & activating the RD or WR control lines. The DMA controller has 3 registers i.e. address, word count & control registers. The address register contains an address to specify the desired location in memory. The word count register hold the number of words to be transferred & the control register specifies the mode of transfer. In DMA transfer, the DMA is first initialized by the CPU after which the DMA starts & continues to transfer data between memory & peripheral unit until an entire block is transferred. The CPU initializes the DMA by sending the following information through the data bus..

1. The starting address of the memory block where data are available for read & write where data are to be stored for write.
2. The word count which is the number of words in the memory block.
3. Control to specify the mode of transfer such as read / write.
4. A control to start the DMA transfer.

Date: March 5 /

(ii) DMA Transfer



- Input-Output Processor (IOP)

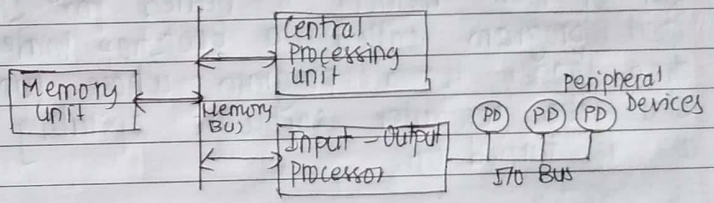


fig: Block Diagram of computer with IO processor

(Baki Cache initialization paxi cha.)

Date: 5th/March

MEMORY ORGANIZATION

- Memory Hierarchy

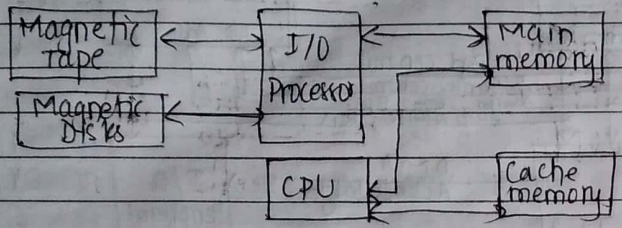
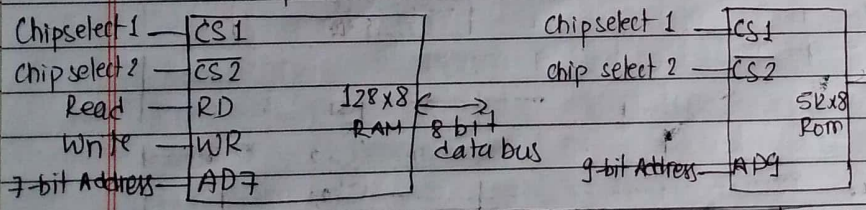


fig. Memory Hierarchy System

- Main memory : RAM, ROM, Boot strap loader

i) Boot Strap loader : Among other things the ROM p
tion of main memory is needed for starting an i
itial program called a bootstrap loader. The boo
strap loader is a program whose function is t
start the computer software operating when po-
wer is turned on.

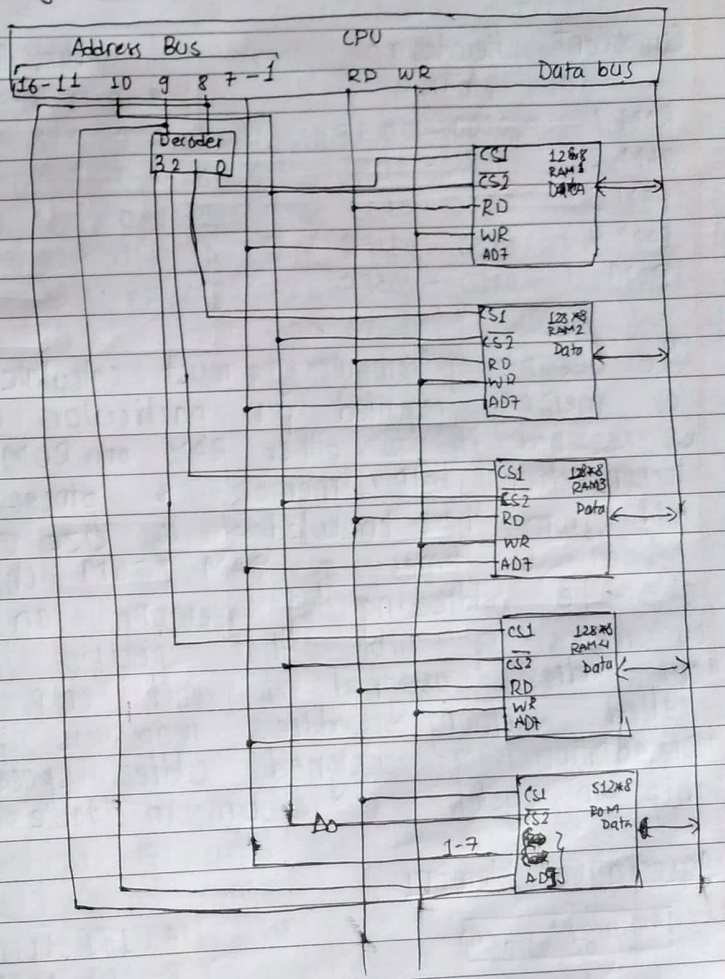
- RAM chip & ROM Chip



CS1	CS2	RD	WR	Memory Function	State of the databus
0	0	X	X	Inhibit	High Impedence
0	1	X	X	Inhibit	High Impedence
1	0	0	0	Inhibit	High Impedence
1	0	0	1	Write	Input data to RAM
1	0	1	X	Read	Output data from RAM
1	1	X	X	Inhibit	High Impedence

Date: / /

Memory Connection To CPU



Date: / /

- Memory Address Map

Component	Hexadecimal Address	Address Bus 10	9	8	7	6	5	4	3	2
RAM 1	0000 - 007F	0	0	0	x	x	x	x	x	x
RAM 2	0080 - 00FF	0	0	1	x	x	x	x	x	x
RAM 3	0100 - 017F	0	1	0	x	x	x	x	x	x
RAM 4	0180 - 01FF	0	1	1	x	x	x	x	x	x
ROM	0200 - 03FF	1	x	x	x	x	x	x	x	x

The designer of computer system must calculate amou of memory required for particular applicat assigned it to either RAM or ROM. The in terconnection betn memory & processor is the estd. from the knowledge of size of memo needed & types of RAM / ROM chips ava able. The addressing of memory can be esta by means of table that specifies the mem ory address assigned to each chip. A tabl called memory address map is pictorial representation of assigned address space for ea chip in system as shown in table above.

- Associative Memory

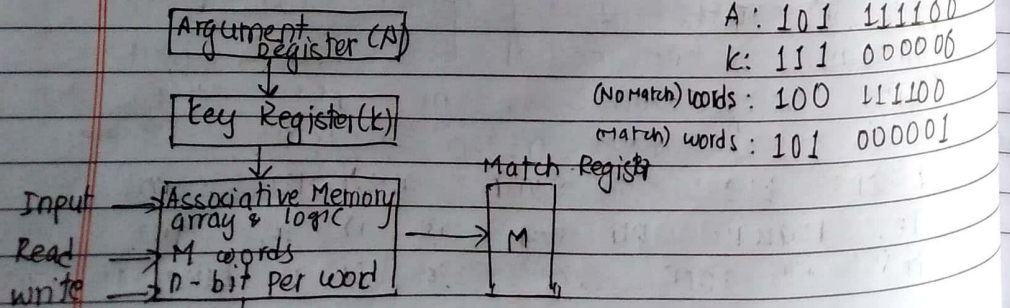


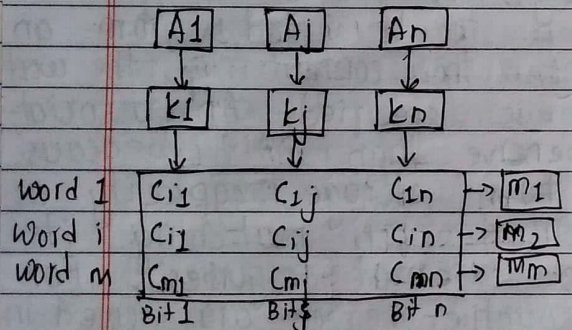
fig: Block Diagram of Associative Memory

Date: / /

The time required to find an item stored in memory can be reduced considerably if stored data can be identified for access by the content of data itself rather than by an address. A memory unit accessed by content is called an associative memory or it may also be referred as content addressable memory (CAM). This type of memory is accessed simultaneously & in parallel on the basis of data content rather than by specific address or location. When a word is written in memory, no address is given. When a word is to be read from an associative memory, the content of the word or part of word is specified. An associative is more expensive than RAM beacaus each cell must have storage capacity as well as logic circuit for matching its content with an external argument. For this reason, associative me are used in app where the search time is very critical & must be very short. The block diagram of an ass memory is shown above which consist of memory array & logic for M words with n-bits per word. The argument register A & key register k, each have n bits, one for each bit of a word. The match register M has n-bits, one for each memory word. The key register provides a mask for choosing a particular field or a key in arg-

Date: / /

ument word. The entire argument is compared with each memory word if key register contains all 1s. Otherwise, only those bits in argument that have 1's in their corresponding position of key register are compared. In eg. above, there're 3 1's in the key register. Hence we'll only compare the 3 bits of argument register with word in asso. memory & in the given eg. word 2 is matched.



Match logic:

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

When $x_j = 1$, the pair of bits in position j are equal otherwise $x_j = 0$

The match logic for each word can be derived from the comparison algorithm for two binary numbers. First we neglect the

Date: / /

key bits & compare the argument in A with the bits stored in cells of the word. Word i is equal to argument in A if $A_j = F_{ij}$ for $j = 1, 2, 3 \ldots n$. Two bits are equal if they're both 1 or both 0. The equality of two bits can be expressed logically by the Boolean function $x_j = A_j F_{ij} + A_j' F_{ij}'$, where $x_j = 1$ if the pair of bits in position are equal otherwise $x_j = 0$. For a word i to be equal to the argument in A, we must have all x_j variables equal to 1. This condition for setting the corresponding match bit M_i to 1. The Boolean function for this condition, $M_i = x_1 x_2 x_3 \ldots x_n$ and constitutes the AND operation of all pairs of matched bit in a word. We now include the key bit k_j in comparison logic. The requirement is that if $k_j = 0$, the corresponding bit A_j & F_{ij} need no comparison. Only when $k_j = 1$, they must be compared & this requirement is achieved by ORing each term with k_j complement. Thus, $x_j + k_j = \begin{cases} x_j & \text{if } k_j = 1 \\ 1 & \text{if } k_j = 0 \end{cases}$

When $k_j = 1$, we have $k_j' = 0$ and $x_j + 0 = x_j$ & similarly when $k_j = 0$ then $k_j' = 1$ & $x_j + 1 = 1$. A term $x_j + k_j'$ will be in the one state if its pair of bit are not compared. This is necessary because each term is ended with all

Date: March 6

other term so that output of J will have no effect. The comparison of bit have an effect only when $k_j = 1$. The match logic for word n in an associative memory can now be accessed by following boolean function,

$M_i = (x_1 + x_1')(x_2 + x_2') \ldots (x_n + x_n')$ which is equivalent to $M_i = \prod_{j=1}^{n} (A_j + F_{ij}' + A_i F_{ij}' + k_1')$

Mar 6

- Cache Memory

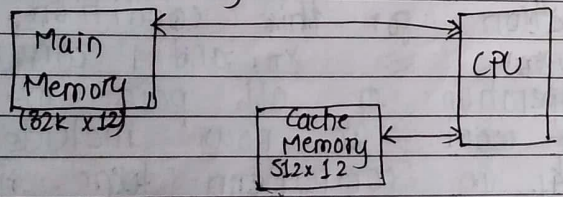


fig: Cache Memory

Analysis of large no. of typical prog. has shown that the reference to memory at any given interval of time tend to be confined within a few localized area in memory. This phenomena is known as the property of locality of reference. If the active portion of prog & data are placed in fast small memory the average memory access time can be reduced. Thus reducing the total execution time of the program. Such a fast small memory is referred to as a

hierarchy
hierarchy

Date: / /

cache memory. It is placed between the CPU & main memory as illustrated in fig. above. The cache is the fastest component in memory hierarchy and approaches the speed of CPU components. The basic operation of cache is as follows:

(i) When the CPU needs to access memory, the cache is examined. If the word is found in the cache, it is read from the cache memory & the process is known as cache hit. If the word is not found in cache, it is defined as cache miss. When a cache miss occur, the required word is accessed from main memory & process is known as cache mapping. If there's space available in cache memory, the block where a required word is located is transferred from main memory to cache but if cache is full, we need to replace the block in cache which will be based on diff. replacement algorithm such as least recently used, FIFO, etc. The performance of cache memory is frequently measured in terms of quantity called the hit ratio. The hit ratio is the no. of hit divided by the total CPU references to a memory (hit + miss). Example

eg: The cache assistant of a computer is 100 ns a main memory assistant is of 1000 ns & a hit ratio is 0.9. Now find out the average memory access ti

Date: / /

Sol:

Cache access time = 100 ns
Main memory access time = 1000 ns
hit ratio = 0.9
Average access time = ?

Now,

Miss ratio = 1 - hit ratio
= 1 - 0.9
= .1

Average access time = hit ratio x Cache access time + miss ratio + (cache access time + main memory access time)
= 0.9 x 100 + .1 (100 + 1000)
= 90 + .1 (1100)
= 200 ns //

hit ratio = $\frac{\text{cache access time}}{\text{(cache + memory) access time}}$

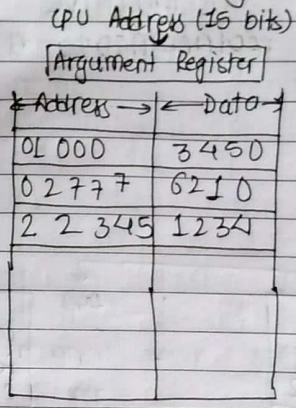
2. - Cache Mapping

The basic characteristic of cash memory is its fast access time. Therefore very little or no time must be wasted when searching for words in the cache. The transformation of data from main memory to cache memory is defined as cache mapping process. There're 3 types of cache mapping as follows:

(i) Associative mapping
(ii) Direct mapping
(iii) Set Associative mapping

Date: / /

(i) Associative Mapping



The fastest & most flexible cache org. uses an associative memory & this org. is illustrated in fig. above. The associative memory stores both the address & content (data) of memory word which permits any location in cash to store any word from main memory. Fig. above shows 3 words presently stored in the cash. The address value of 15 bit is shown as a 5 digit octal no. & its corresponding 12 bit word is shown as a 4 digit octal no. A CPU address of 15 bit is placed in argument reg. & the associative memory is searched for a matching address. If the address is found, corresponding 12 bit data is read & sent to CPU & if no match occurs, the main memory is accessed for the word. The address data pair is then transferred to association mem. If space is availble,

Date: / /

the address data pair will be placed otherwise data from associative mem- is replaced using replacement algorithm

(ii) Direct Mapping

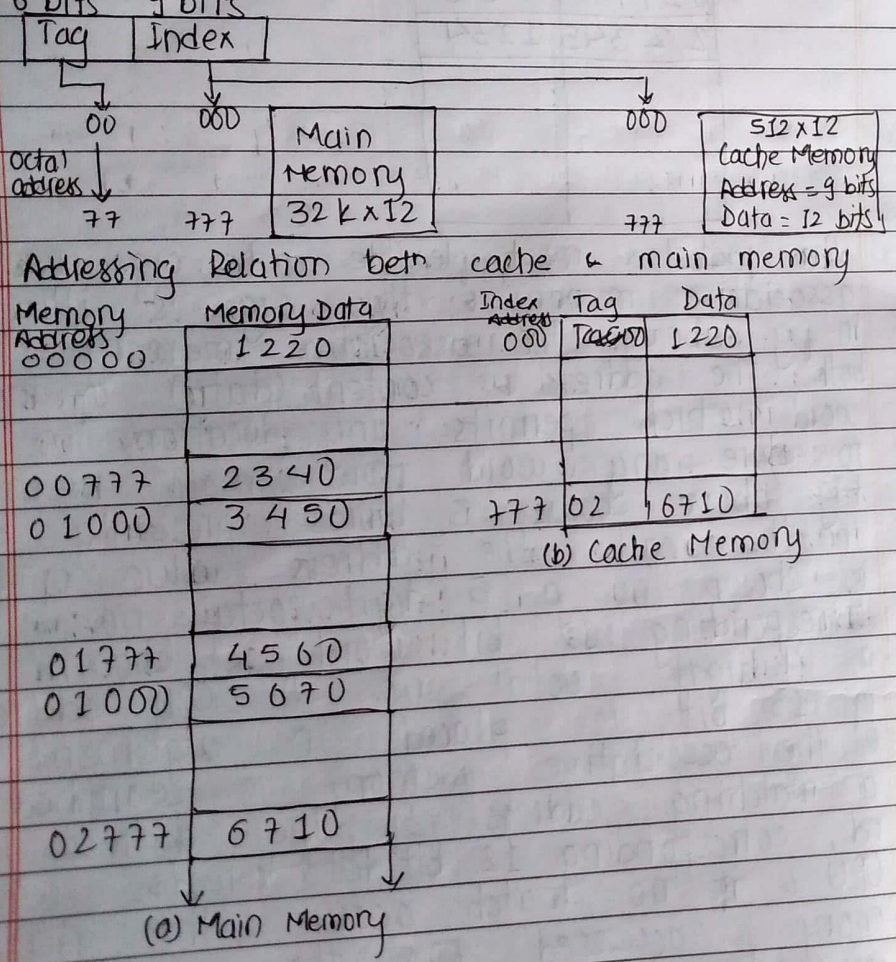


fig: Direct Mapping

Date: / /

Associative memories are expensive compared to RAM because of added logic associated with each cell. As a alternative, we can use direct mapping which is illustrated in fig. above. The CPU address of 15 bit is divided into two field. The 9th least significant bit constitutes the index field & remaining 6 bit form tag field. In general case 2^k words in cache memory & 2^n in main memory. The n-bit memory address is divided into two field i.e. k-bit for index & (n-k)-bit for tag field. The direct mapping cache org uses n-bit address to access the main memory & k-bit address to access the cache as illustrated in fig. above. The disadvantage of direct mapping i.e. two words with same index in their address but with diff. tag values can't reside in cache memory at same time which can drop the hit ratio if two or more words whose addresses have the same index but diff. tags are accessed repeatatively.

(iii)

Date: / /

(iii) Set Associative Mapping

Index	Tag	Data	Tag	Data
000	01	3450	02	5670
777	02	6710	00	2340

Two way set Associative Mapping Cache

A set associative mapping is an improvement over direct mapping org. in which each word of a cache can store two or more words of memory under the same index address. Each data word is stored together with its tag & no. of tag data item in one word of cache is ~~that set~~ said to form a set. An ex. of set associative cache org. for set size of two is shown in fig. above where each index address refers to two data word & their associated tag. In general a set associative cache of set size k will accumulate k words of main memory in each word of a cache.

- **Writing into Cache**

When the CPU finds a word in a cache during read operation, the main memory

Date: / /

is'nt involved in transfer. However if the operation is write, there're two ways that a system can proceed:

(i) Write Through
(ii) Write Back

(i) The simplest & most commonly used procedure is to update main memory with every memory write operation with cache memory being updated in parallel if it contains the word at specified address. This is called write through method. Its advantage is that the memory will always contain the same data as the cache.

(ii) In this method, only the cache location is updated during a write operation & location is then marked by a flag so that when the word is removed from the cache it is copied into main memory.

- **Cache Initialization**

One more aspect of cache org. that must be taken into consideration is the problem of initialization. The cache is initialized when power is applied to the computer or when the main memory is loaded with a complete set of program from axulliary memory. After initialization, the cache is considered to be empty but in effect it contains some non-valid

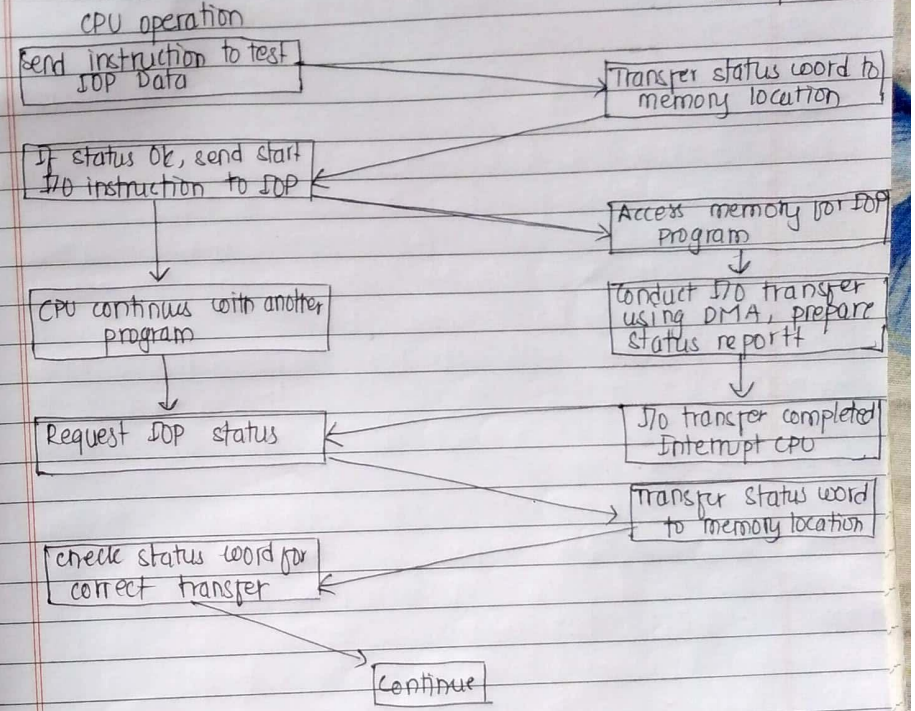
Date: / /

data. It is customary to include with each word in cache a valid bit to indicate whether or not the word contains valid data. The cache is initialized by clearing all the valid bits to zero.

Date: March 4

IOP

CPU IOP communication:



Practical

Date: Feb. 5

Addition

```
        ORG   10
        LDA   XY1
        ADD   XY2
        STA   Res
        HLT
XY1,    DEC   5
XY2,    DEC   4
Res,
        END
```

Subtraction

```
        ORG   10
        LDA   A1
        CMA   A1
        INC   A1
        ADD   A2
        STA   S
        HLT
A1,  DEC 2
A2,  DEC 6
S,
        END
```

1. Write a program in assembly language to add two numbers i.e. decimal value 7 & decimal value 2. The program starts from memory address 17.

=)

Date: / /

```
    ORG  17
    LDA  XY1
    ADD  XY2
    STA  Res
    HLT
XY1, DEC 7
XY2, DEC 2
Res,
    END
```

1. Write a program to subtract two numbers 7 & 3. The memory address starts from 101.

```
    ORG  101
    LDA  A1
    CMA  A1
    INC  A1
    ADD  A2
    STA  a
    HLT
A1, DEC 3
A2, DEC 7
a,
    END
```

2. Write a program to add two numbers 5 & 4. Assign the memory address of your choice

```
    ORG  23
    LDA  S1
    ADD  S2
```

Date: Feb/. 12 /

```
    STA  S
    HLT
S1, DEC 5
S2, DEC 4
S,
    END
```

Feb. 12

ISZ → Memory word +1 & check if memory word is zero and
CLA → clear AC
BUN → Branch unconditional (goto)
CMA →
INC →

Multiply (7x5)

```
    LDA  XY2
    CMA
    INC
    STA  Tem      Tem = -5
    CLA           AC = 0
L1, ADD  XY1      AC = 0+7 = 7
    ISZ  Tem      Tem = -4 = ?   AC = 7+7 = 14
    BUN  L1
    STA  Res
    HLT
Tem,
XY1, DEC 7
XY2, DEC 5
Res,
    END
```

qw

Date: / /

1. Write a program to add 2, 4 & 1.

```
      LDA   A1
      ADD   A2
      STA   S1
      ADD   A3
      STA   S
      HLT
A1,   DEC   2
A2,   DEC   4
A3,   DEC   1
S,
      END
```

* multiply 2x4

```
      LDA   A1      (4)
      CMA           (4')
      INC           (-4)
      STA   Tem     (Tem = -4)
      CLA           AC = 0
T1,   ADD   A2      0 + 2 = 2
      ISZ   Tem     -4 = 0 ? NO / yes   2+2=4
      BUN   T1      goto T1
      STA   Res
      HLT
Tem,
A1,   DEC   4
A2,   DEC   2
T1,
      END
```

CO Practical

Date: Mar 15/

Input

```
Lo, SKI
    BUN  Lo
    INP
```

Output

```
Lo, SKO
    BUN  Lo
    OUT
```

Input

1 character Input

```
    ORG  10
    LDA  XYZ
    STA  CTR
Lo, SKI
    BUN  LO
    INP
L1, SKO
    BUN  L1
    OUT
    ISZ  CTR
    BUN  LO
    HLT
```

Lab Addition, sub, multip, 1 charac. input/output