Sem-1

CO

Digital Computers

Ch-1

0-1 The digital computer is a digital system that performs various computations. The first electronic digital computers, developed in 1940s. Digital computers use the binary number system, which has two digits 0 and 1. A binary digit is called a bit. Information is represented in digital computers in group of bits. By using various coding techniques, group of bits can be made to represent not only binary numbers but also symbols, such as letters, decimal digits

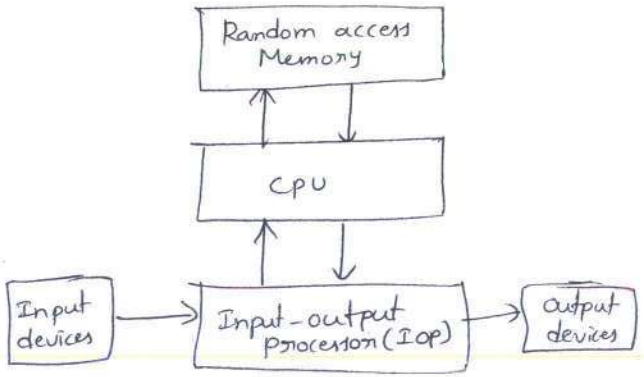
Decimal digits use a base 10 system and binary numbers use base 2 system with two digits: 0 and 1. For example binary number 1001011 represents a quantity that can be converted to a decimal number by multiplying each bit by the base 2 raised to an integer power as follows

$1\times2^6+0\times2^5+0\times2^4+1\times2^3+0\times2^2+1\times2^1+1\times2^0=75$

A computer system is sometimes subdivided into 2 functional entities: hardware and software. The hardware of computer consists of all the electronic components and devices. Computer software consists of the instructions and data the computer manipulates to perform various tasks. The system software of a computer consists of a collection of programs whose purpose is to make more effective use of the computer.

The hardware of the computer is divided into 3 major parts

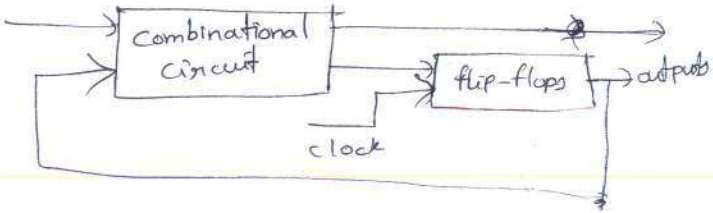
1. The central processing unit (CPU)
2. Memory
3. Input-output processor



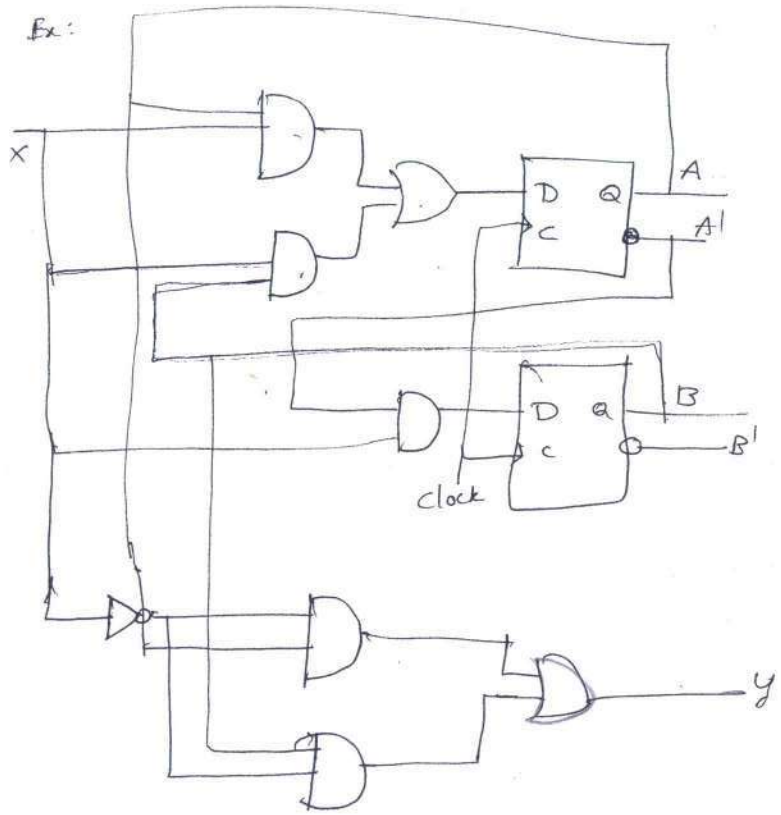
CPU contains an arithmetic and logic unit for manipulating data, a number of registers for storing data, control circuits for fetching and executing instructions. The memory contains storage for instructions and data. It is called Random access memory (RAM) because the CPU can access any location within a fixed interval of time. The input-output processor (IOP) contains electronic circuits for communicating and controlling the transfer of information between computer and the outside world.

Sequential Circuits D-2

A sequential circuit is an interconnection of ~~fit~~ flip-flops and gates. The gates themselves constitute a combinational circuit but when included with the flip-flops, the overall circuit is classified as a sequential circuit



Ex:



$D_A = Ax + Bx$

$D_B = A'x$

$y = Ax' + Bx'$

The output of OR gate is connected to the D input of flip-flop A, we write the first input equation as

$$D_A = Ax + Bx$$

Where A and B are the outputs of two flip-flops and x is the external input.

The second equation is derived from the single AND gate whose output is connected to the D input of flip-flop B

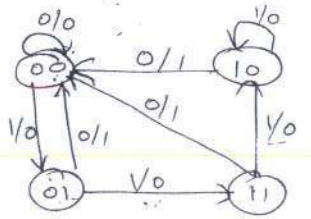
$$D_B = A'x$$

The sequential circuit also has an external output

$$y = Ax' + Bx'$$

State table

Present state		Input	Next state		output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

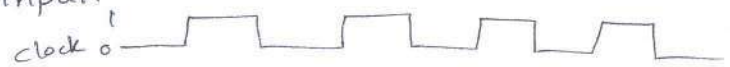


State diagram

There is no difference between a state table and state diagram except in the manner of representation. The state table is easier to derive from a given logic diagram and the state diagram follows directly from the state table.

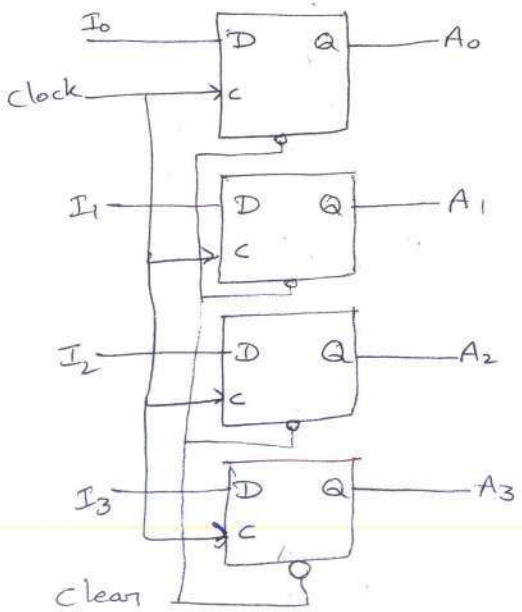
The same clock transition may change the input value. If the input changes to 0, the output becomes 1, but the input remains at 1, the output stays at 0. This information is obtained from the state diagram along the two directed lines emanating from the circle representing state 01.

The most fundamental sequential component are the latch and ~~flip~~ flip-flop. They store one bit of data and make available to other components. Latches are level triggered flip flops are edge triggered, both have a clock input.



Registers

A register is a group of flip-flops capable of storing one bit of information. An n bit register has a group of n flip flops and is capable of storing any binary information of n bits. In addition to the flip-flops, a register may have combinational gates that perform ~~certain~~ certain data processing tasks. The flip-flop holds the binary information and the gates control when and how new information is transferred into the register. The following figure shows a register constructed with four D flip-flops.



gates
cache memory
virtual memory
DMA
Interrupts
asynchronous transmission
Microsequencer design
CPU design
RTL and microoperations
Memory (RAM, ROM)
Addressing modes
Instruction set Architecture

30%

1617904707289 46

Digital Logic fundamentals

Digital logic is the basis for computer design. Digital logic gates, latches, memories and other components are used to design computer systems, including CPU/microprocessor. There are two classes of digital logic combinatorial logic and sequential logic. The output values of a combinatorial logic circuit depend on its current inputs. When input values are changed the output values will change correspond to the new input values. The current outputs depend solely on the current inputs.

The outputs of a sequential circuit depend on both the current inputs and on previous inputs and outputs of the circuit. The state and inputs also combine to generate new state of the circuit. The same inputs to a sequential circuit may generate different outputs, and different new states

Boolean Algebra (logic gates) - 39

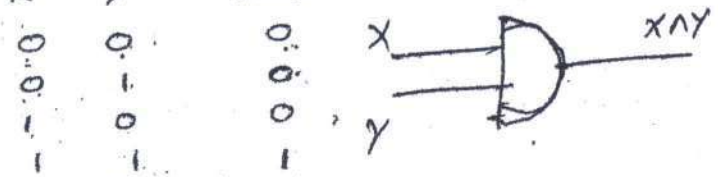
A boolean value can be either true or false. 1 is usually used to represent true and 0 to represent false.

1. AND
2. OR
3. NOT
4. XOR
5. XNOR
6. NAND
7. NOR

AND

The AND function output is 1 if and only if every input has a value of 1. In expression form the AND of boolean values x and y could be represented as $x \wedge y$, $x \cdot y$ or simply xy

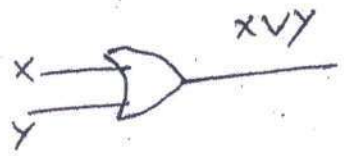
X	Y	X∧Y
0	0	0
0	1	0
1	0	0
1	1	1



OR

The OR function produces an output of 1 if any of its inputs is equal to 1. The logical OR is represented as $x \vee y$ or $x + y$

x	y	x∨y
0	0	0
0	1	1
1	0	1
1	1	1



NOT

NOT function operates on a single boolean value. An input of 1 it produces an output of 0. An input of 0 it produces an output of 1

X	X^1
0	1
1	0

EXCLUSIVE OR (XOR)

This function is represented

If two inputs are equal to 1, their XOR

If two inputs are equal to 0, their XOR is

X	Y	X⊕Y
0	0	0
0	1	1
1	0	1
1	1	0

XNOR

If two inputs are equal to 0 the

XNOR is 1. If two inputs are equal to 1 the

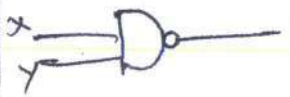
XNOR is 1

X	Y	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

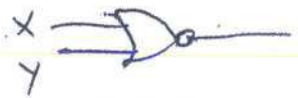
NAND, NOR

NAND function outputs a 1 if not all its inputs are 1. The NOR function produces 1 no inputs are equal to 1. These are called universal gates

X	Y	NAND
0	0	1
0	1	1
1	0	1
1	1	0



X	Y	NOR
0	0	1
0	1	0
1	0	0
1	1	0



Manipulating Boolean functions

Truth table for the function $xy' + yz$

x	y	z	y'	xy'	yz	xy'+yz
0	0	0	1	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	0	1	1	1	0	1
1	1	0	0	0	0	0
1	1	1	0	0	1	1

$(x+y')(y+z)$

x	y	z	x+y'	y+z	(x+y')(y+z)
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Demorgan's Law

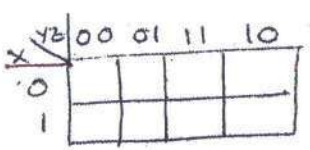
Demorgan's Law allows a digital designer to convert an and function to an equivalent or function and vice versa

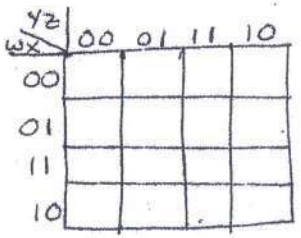
$(ab)' = a' + b'$

$(a+b)' = a'b'$).39

Karnaugh Map - 40

A karnaugh map is set up as a grid and is used to represent the values of a function for different input values. Each cell in the k-map represents a minterm. Each possible set of input values is called a minterm. Consider 3 input karnaugh map and 4 input karnaugh map





There are 3 inputs x, y and z and each input can have the value 0 or 1. There are $2^3 = 8$ possible combinations of input value and $2^4 = 16$ possible combinations of input value

In these karnaugh maps that the input values do not follow the linear progression 00, 01, 10, 11. Rather, they are ordered 00, 01, 11, 10. This ordering is called the Gray code order

A Gray code is a reflected code whose order depends on the number of bits in its value.

(10) Gray code sequence generation ✓

0	0 ⇒ 0 ⇒	00	000
1	1	01	001
	1	11	011
(a)	0	10	010
			110
	(b)		111
			101
			100
			(c)

We start by listing the 1 bit sequence. Then we list the 1 bit sequence again but in reverse order. We append a leading bit of 0 to the values in the original 1 bit sequence and a leading 1 to the values in the reflected sequence.

000 → 001 → 011 → 010 → 110 → 111 → 101 → 100 → 000 ...

Karnaugh map for $(xy' + yz)'$

x \ yz	00	01	11	10
0	1	1	0	1
1	0	0	0	1

without grouping

x \ yz	00	01	11	10
0	1	1	0	1
1	0	0	0	1

with grouping

x	y	z	y'	xy'	yz	xy'+yz	(xy'+yz)'
0	0	0	1	0	0	0	1
0	1	0	0	0	0	0	1
0	1	1	0	0	1	0	0
1	0	0	1	1	0	1	0
1	0	1	1	1	0	1	0
1	1	0	0	0	0	0	1
1	1	1	0	0	1	1	0

(from demorgan)

$[(ab)' = a' + b']$

$[(a+b)' = a'b']$ ~~$xy' + yz'$~~

x	y	z	$(xy')'$	$(yz)'$	$(xy')'.(yz)'$	$(x'+y)$	$(y'+z')$
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	1	0	0	1	0
1	0	0	0	1	0	1	1
1	0	1	0	1	0	0	1
1	1	0	1	1	1	1	1
1	1	1	1	0	0	1	0

$(x'+y)(y'+z')$	$x'y'$	$x'z'$	yz'	$x'y' + x'z' + yz'$
1	1	1	0	1
1	1	0	0	1
1	0	1	1	1
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	0	0	1	1
0	0	0	0	0

end) 4Q

Ex-

wx \ yz	00	01	11	10
00	1	0	0	1
01	0	0	1	0
11	0	0	0	0
10	1	1	1	1

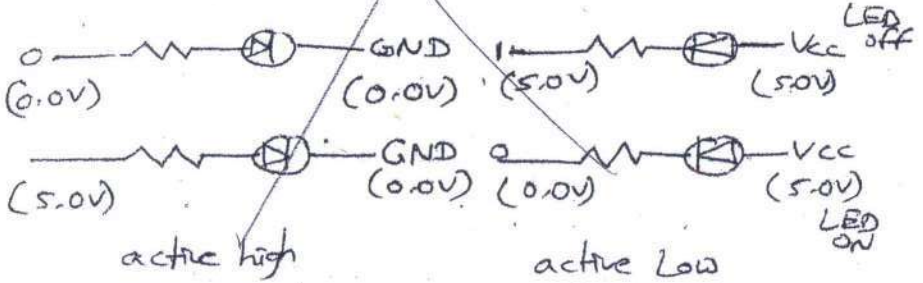
⇒ $w'x'y'z' + w'x'yz' + w'xyz + wx'y'z' + wx'y'z + wx'yz + wx'yz'$

⇒ $x'y'z'(w+w') + wx'y(z+z') + w'y(xz+x'z')$

$\pm wx'y'z$

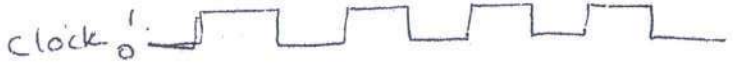
LEDs

LEDs can be wired so they are lit by a logic 1 (active high) or by a logic 0 (active low). A logic 1 typically corresponds to a voltage of about 5V for TTL digital logic. a logic 0 represents a voltage of ou.



Basic Sequential components (Flip-flops) - SQ

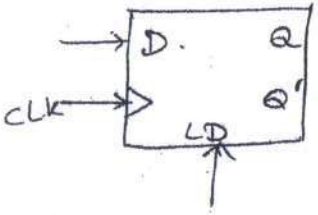
The most fundamental sequential component are the latch and flip-flop. They store one bit of data and make available to other components. Lathes are level triggered. flip flops are edge triggered. both have a clock input.



1. D Flip Flop
2. JK flip flop
3. T flip-flop

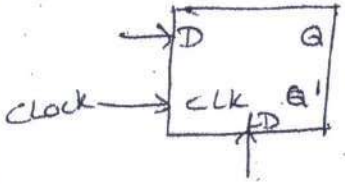
D - Flip Flop:-

The flip-flop has one data input D and a clock input. when the clock input changes from 0 to 1, the data on the D input is loaded into the flip-flop. The data is made available via output Q, and its complements via Q'.



D	LD	clock	Q
X	0	X	Q_0
0	1	↑	0
1	1	↑	1
X	1	not↑	Q_0

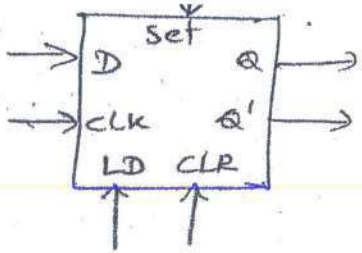
(a) edge triggered



D	LD	CLK	Q
X	0	X	Q_0
X	1	0	Q_0
0	1	1	0
1	1	1	1

(b) level triggered

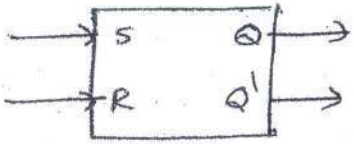
The second figure loads data as long as both its clock and load signals are 1. If both are 1, the value of the D input is passed to the Q output. If D changes while the load and clock signals are 1, Q also changes.



D	LD	CLR	SET	CLK	Q
X	X	X	1	X	1
X	X	X	0	1	0
X	0	X	0	0	Q_0
X	1	0	0	0	Q_0
0	1	1	0	0	0
1	1	1	0	0	1

A positive level triggered D latch.

SR latch

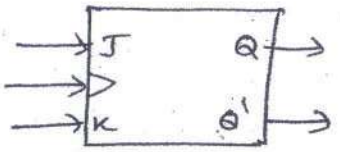


S	R	Q
0	0	Q_0
0	1	0
1	0	1
1	1	undefined

SR latch does not have a clock input. Its S and R inputs are always active. The output of the latch is undefined when SR = 11.

JK flip-flop

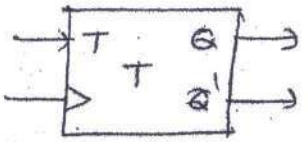
JK flip flop resolves the problem of undefined outputs associated with the SR latch. As with the SR latch, J=1 sets the output to 1 and K=1 resets the output to 0. However JK=11 inverts the current value.



J	K	Clock	Q
X	X	not ↑	Q_0
0	0	↑	Q_0
0	1	↑	0
1	0	↑	1
1	1	↑	Q_0

T Flip-flop (Toggle)

The T input does not specify a value for this flip-flop's output, only whether or not it should be changed. On the rising edge of the clock, if T=0, the output of the flip-flop is unchanged. However, if T=1, the output is inverted.

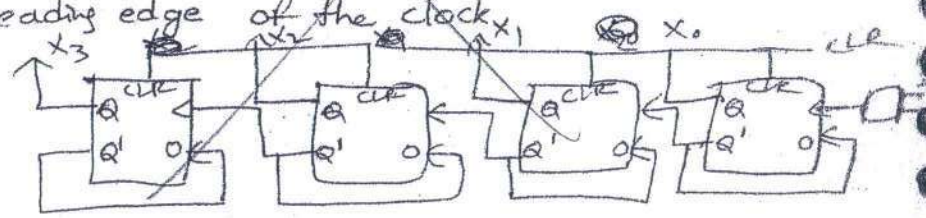


T	Q
0	Q_0
1	Q_0'

) SQ

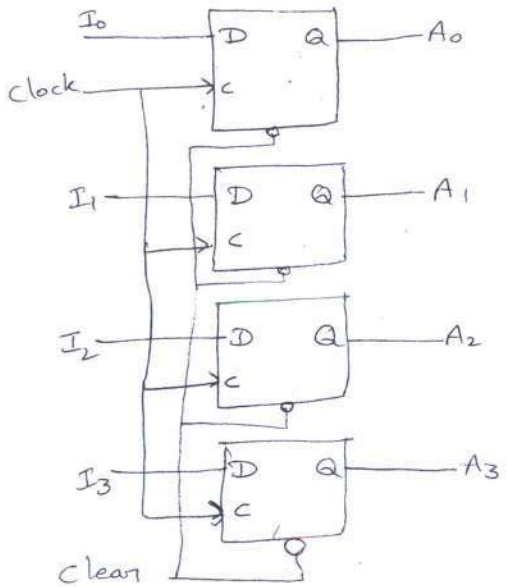
Counters

A counter stores a binary value, arithmetically increments and decrements the value. It uses the D flip-flop to store the 4 bit value. The CLR control signal clears the counter. Setting INC=1 increments the counter on the leading edge of the clock.



Registers Ch-2 Digital components

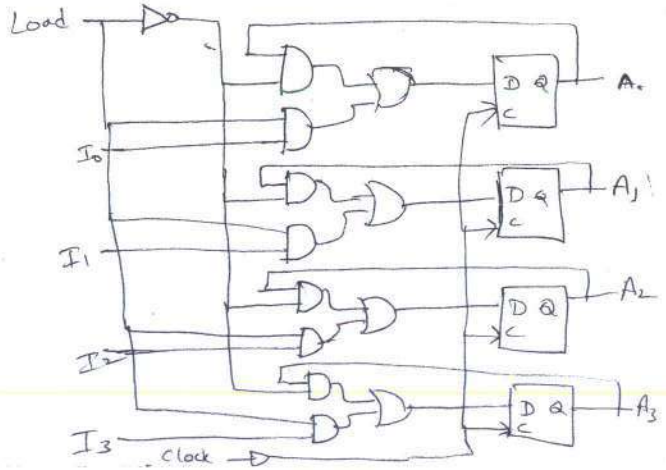
A register is a group of flip-flops capable of storing one bit of information. An n bit register has a group of n flip flops and is capable of storing any binary information of n bits. In addition to the flip-flops, a register may have combinational gates that perform ~~certain~~ certain data processing tasks. The flip-flop holds the binary information and the gates control when and how new information is transferred into the register. The following figure shows a register constructed with four D flip-flops.



The clear input goes to a special terminal in each flip-flop. It is useful for clearing the register to all 0's. The clock signal enables the D input but the clear input is independent of clock.

Register with parallel load

Most digital systems have a master clock generator that supplies a continuous train of clock pulses. The clock pulses are applied to all flip-flops and registers in the system. The master clock acts like a pump that supplies a constant beat to all parts of the system. A Seperate control signal must be used to decide which specific clock pulse will have an effect on a particular register



The load input in the register determines the action to be taken with each clock pulse. When the load input is 1, the data in the 4 inputs are transferred into register. When load input is 0 then D inputs of the flip-flops are connected to their outputs. and data inputs are inhibited. The D input determines the next state of the output.

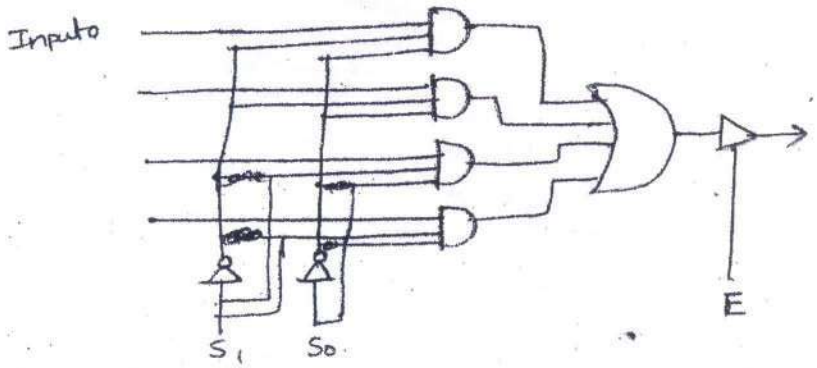
Combinatorial components

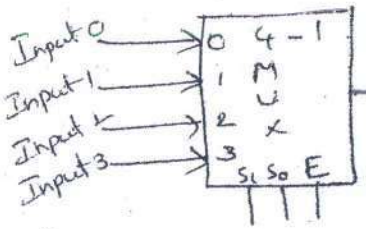
1. Multiplexer
2. Decoders
3. Encoders
4. Comparators
5. Adders and subtracters
6. Memory.

Multiplexer — 3Q

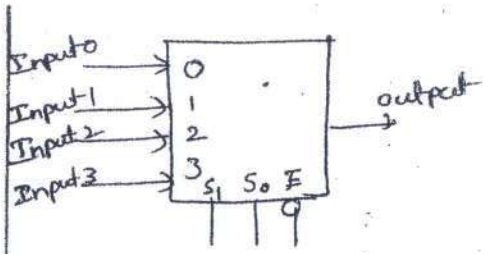
A Multiplexer, or MUX is a selector. It chooses one of its data inputs and passes it through to its output.

Ex:- 4 to 1 Multiplexer



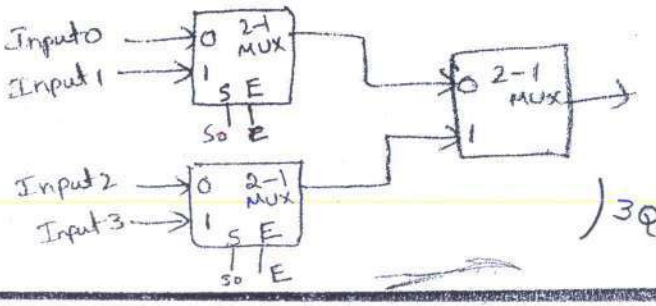


S_1	S_0	E	output
X	X	0	Z
0	0	1	Input 0
0	1	1	Input 1
1	0	1	Input 2
1	1	1	Input 3



S_1	S_0	E	output
X	X	1	Z
0	0	0	Input 0
0	1	0	Input 1
1	0	0	Input 2
1	1	0	Input 3

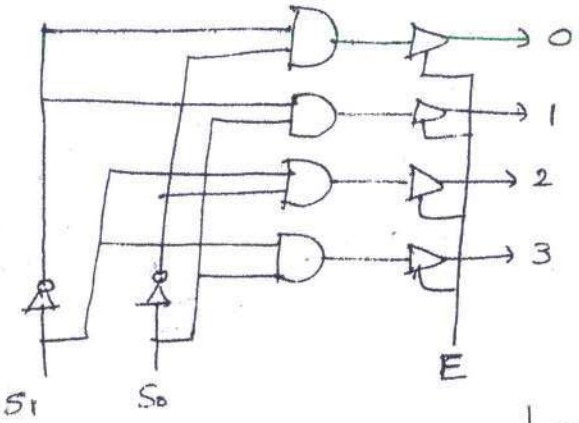
Four binary data values are passed to the inputs of the multiplexer. Two select signals determine which of the four inputs is passed through to the output. If $S_1 = 0$ and $S_0 = 0$ the inputs to the TOP AND gate are 0, $S_1' = 1$ and $S_0' = 1$; the output is the value of input 0. The other three gates each input either S_1 or S_0 (or both). Since both are 0, these AND gates produce 0 results. Setting S_1 and S_0 to 01, 10 or 11 produces outputs of the value of Input 1, 2 and 3. Finally the output passes to a tri state buffer. If the buffer is enabled (E = 0) the value is passed as the output of the multiplexer. Otherwise, the output is the high impedance value Z. Multiplexers can be used to select from a large number of inputs.

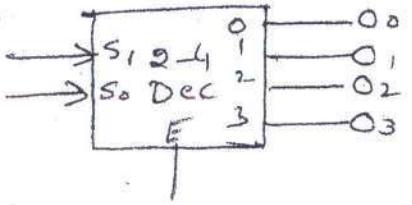


3Q

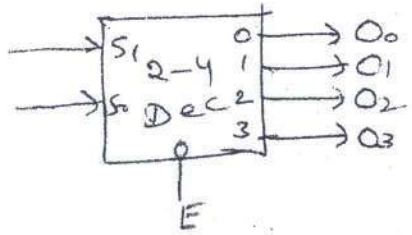
Decoders – 2Q

A decoder, accepts a value and decodes it. It has n inputs and 2^n outputs, numbered from 0 to $2^n - 1$. For example, a decoder with three inputs and eight outputs will activate output 6 when the input values are 110. Decoders can have active high or active low enables.





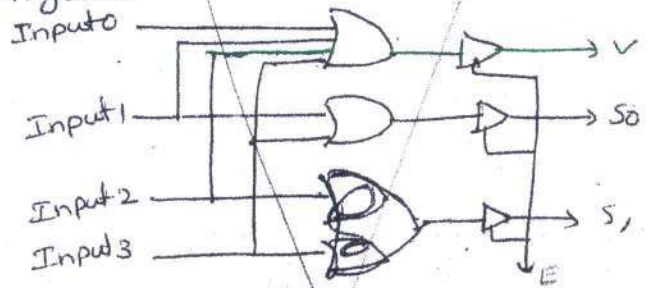
S_1	S_0	E	O_0	O_1	O_2	O_3
X	X	0	0	0	0	0
0	0	1	1	0	0	0
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	1

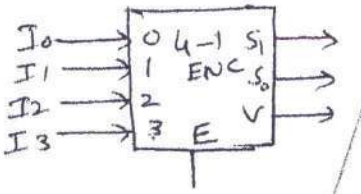


S_1	S_0	E	O_0	O_1	O_2	O_3
X	X	1	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	1	0	0
1	0	0	0	0	1	0
1	1	0	0	0	0	1

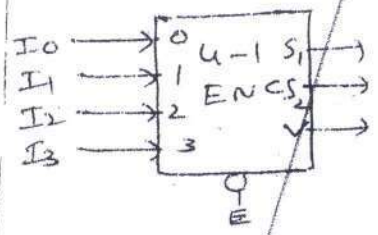
Encoders

The encoder is the exact opposite of the decoder. It receives 2^n inputs and outputs an n bit value corresponding to the one input that has a value of 1. The 4 to 2 encoder and its schematic representation are shown in figure.



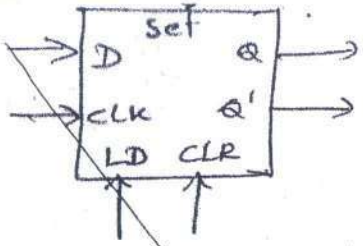


I_0	I_1	I_2	I_3	E	S_1	S_2	V
X	X	X	X	0	Z	Z	Z
0	0	0	0	1	0	0	0
1	0	0	0	1	0	0	1
0	1	0	0	1	0	1	1
0	0	1	0	1	1	0	1
0	0	0	1	1	1	1	1



I_0	I_1	I_2	I_3	E	S_1	S_2	V
X	X	X	X	1	Z	Z	Z
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
0	1	0	0	0	0	1	1
0	0	1	0	0	1	0	1
0	0	0	1	0	1	1	1

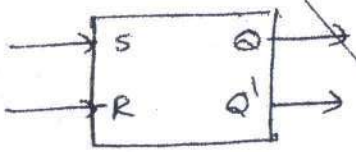
Here there is a third output V. The value indicates whether any of the inputs are active. The encoder will output $S_1 S_0 = 00$ if either input 0 is active or no input is active



D					
X	X	X	1	X	1
X	X	X	0	1	0
X	0	X	0	0	Q_0
X	1	0	0	0	Q_0
0	1	1	0	0	0
1	1	1	0	0	1

A positive level triggered D latch.

SR latch

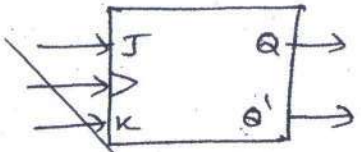


S	R	Q
0	0	Q_0
0	1	0
1	0	1
1	1	undefined

SR latch does not have a clock input. Its s and R inputs are always active. The output of the latch is undefined when SR = 11.

JK flip-flop

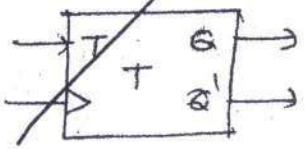
JK flip flop resolves the problem of undefined outputs associated with the SR latch. As with the SR latch, J=1 sets the output to 1 and K=1 resets the output to 0. However JK=11 inverts the current value



J	K	Clock	Q
X	X	not ↑	Q_0
0	0	↑	Q_0
0	1	↑	0
1	0	↑	1
1	1	↑	Q_0

T Flip-flop (Toggle)

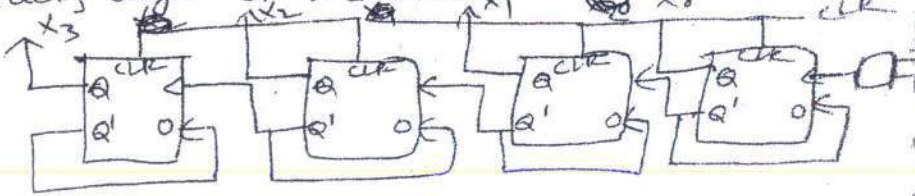
The T input does not specify a value for this flip-flop's output, only whether or not it should be changed. On the rising edge of the clock, if T=0, the output of the flip-flop is unchanged. However, if T=1, the output is inverted.



T	Q
0	Q_0
1	$\overline{Q_0}$

Counters - 4Q

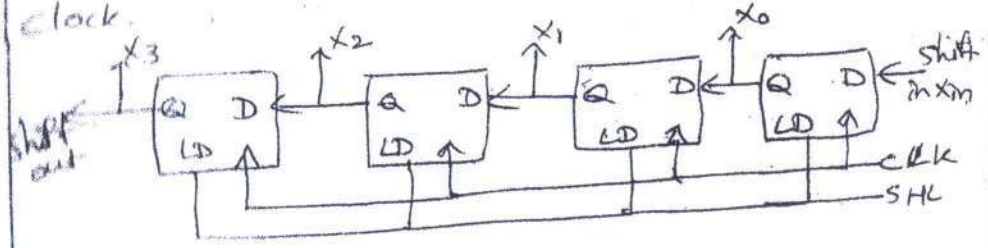
A counter stores a binary value, arithmetically increments and decrements the value. It uses the D flip-flop to store the 4 bit value. The CLR control signal clears the counter. Setting INC=1 increments the counter on the leading edge of the clock



CLK	INC	CLK	X_{3-0}
1	x	x	0
0	0	x	X_{3-0}
0	1	not↑	X_{3-0}
0	1	↑	$X_{3-0}+1$) 4Q

Shift Registers - 5Q

A shift register can shift its data one bit position to the right or left. It is particularly useful for certain hardware multipliers and dividers, which use a shift-add or shift-subtract methodology. When the shift Left (SHL) signal is high, data is shifted one position to the left on the raising edge of the clock.

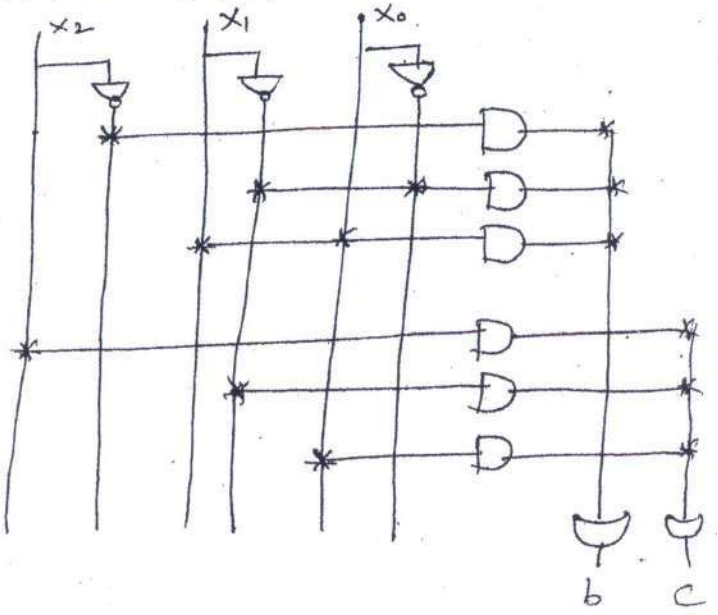


SHL	CLK	X_{3-0}
0	x	X_{3-0}
1	not↑	X_{3-0}
1	↑	$X_2 X_1 X_0 X_{in}$

Programmable logic devices

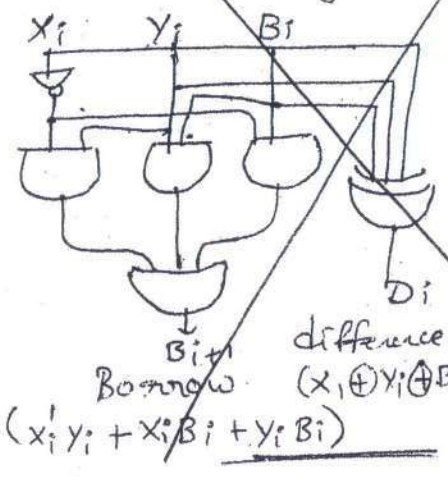
Programmable logic array contains one or more AND-OR arrays. In the array, the inputs and their complements are made available to several AND gates.

Ex:- $b = x_2' + x_1' x_0' + x_1 x_0$ and $c = x_2 + x_1' + x_0$



) 5Q

A full subtracter has 3 inputs, two data inputs and a borrow input. It is possible to use 2's complement addition to implem subtraction. It is generated by complementing the value and adding 1.



Xi	Yi	Bi	Bi+1	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Borrow diffe

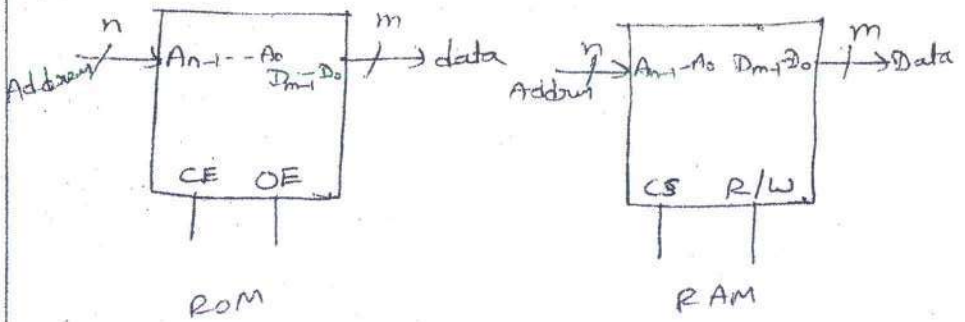
Memory - 6Q

Memory is a group of circuits use to store data. A memory component has some number of memory locations, each of which stores a binary value of some fixed length. The numb of locations and the size of each location vary from memory chip to memory chip. The size of the chip is denoted as the number of locations times the number of bits in each location. For example, a memory chip of size 512 x 8 has 512 memory locations, each of which has 8 bits.

The address inputs of a memory chip choose one of its locations. A memory chip with 2^n locations requires n address inputs, usually labeled $A_{n-1}\ A_{n-2} - - - A_0$. The 512×8 memory has address lines $A_8\ A_7 - - - A_0$

The data pins on a memory chip are used to access the data. There is one pin per bit in each location. For chips with m bits per location, these pins are $D_{m-1}\ D_{m-2} - - D_0$. The 512×8 memory chip has eight data pins $D_7, D_6 - - - D_0$.

There are two primary classes of chips. ROM and RAM. ROM stands for read only memory. It is also called non-volatile memory. We can read the data from ROM, we can not write new data to the ROM. ROM does not erase its contents when the power is turned off.



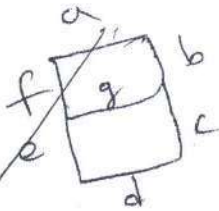
RAM stands for Random Access memory

This is often referred to as read/write memory. RAM erases its contents when the power is turned off. It is called as volatile memory. It stores the data temporarily.) 6Q.

Combinatorial Circuit Designs

* BCD to 7 segment decoder
* Data sorter.

BCD to 7 segment decoder.



It performs a common function. It converts the binary representation of a digit decimal digit, 0000 to 1001.

x_3	x_2	x_1	x_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1

Data sorter

A simple data sorter will input four 4 bit values and output them in descending order.

Chapter. 3

Data Representation

3.1 Data Types

Binary information in digital computers is stored in memory or processor registers. Registers contain either data or control information. Control information is a bit or group of bits used to specify the sequence of command signals needed for manipulation of the data in other registers. There are different types of datatypes

* numbers used in arithmetic computations
* letters of the alphabet used in data processing
* other symbols used for specific purpose.

Number Systems

A number system of base, or radix, r is a system that uses distinct symbols for r digits. There are different types of number systems

* Decimal
* binary
* octal
* hexadecimal

Decimal number system in everyday use employs the radix 10 system. The 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. The string of digits 724.5 is interpreted to represent the quantity

$$7 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 + 5 \times 10^{-1}$$

that is, 7 hundreds, plus 2 tens, plus 4 units, plus 5 tenths.

The binary number system uses the radix 2. The two digit symbols used are 0 and 1. The string of digits 101101 is interpreted to represent the quantity

$$1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 45$$

For example to distinguish the equality between decimal and binary forty-five we will write

$$(101101)_2 = (45)_{10}.$$

The octal system symbols are 0, 1, 2, 3, 4, 5, 6, 7. For example, octal 736.4 is converted to decimal as follows

$$(736.4)_8 = 7 \times 8^2 + 3 \times 8^1 + 6 \times 8^0 + 4 \times 8^{-1}$$
$$= 7 \times 64 + 3 \times 8 + 6 \times 1 + 4/8$$
$$= (478.5)_{10}$$

The Hexa decimal system symbols are 16. They are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

$$\text{Ex:-}\ (F3)_{16} = F \times 16^1 + 3 \times 16^0$$
$$= 15 \times 16 + 3$$
$$\Rightarrow (243)_{10}$$

Octal and Hexadecimal numbers

The conversion from binary to octal is easily accomplished by partitioning the binary number into groups of 3 bits each.

ex:- coversion of decimal 41.6875 into binary

Integer = 41

2 | 41	
2 | 20 – 1	
2 | 10 – 0	
2 | 5 – 0	
2 | 2 – 1	
1 – 0	

101001

Fraction = 0.6875

0.6875
× 2
1.3750
× 2
0.7500
× 2
1.5000
× 2
1.0000

$(41.6875)_{10} = (101001.1011)_2$

Ex :-

1	2	7	5	4	3	octal
00 1 010	111	101	100	011		Binary
A	F	6	3			Hexadecimal

Conversion of binary to Hexadecimal is the bits are divided into groups of four. The below tables represent binary coded octal numbers and binary coded hexa decimal numbers. For example decimal number 99, when converted to binary becomes 1100011. The binary coded octal equivalent of decimal 99 is 001 100 011, and binary coded hexa decimal is 0110 0011.

Binary coded octal Numbers

octal number	Binary coded octal	Decimal equivalent
0	000	0
1	001	1
2	010	2
3	011	3
4	100	4
5	101	5
6	110	6
7	111	7

ex: $(143)_8 \rightarrow$ 001 100 011 $\rightarrow$ $(99)_{10}$

Binary coded Hexadecimal

Hexadecimal number	Binary coded hexadecimal	Decimal equivalent
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

ex: 63 $\rightarrow$ 0110 0011 $\rightarrow$ 99

Decimal representation

The binary number system is most natural system for a computer, but people are accustomed to decimal system. One way to solve the problem is to convert all input decimal numbers into binary numbers, let the computer perform all arithmetic operations in binary and then convert the binary result to decimal.

A binary code is a group of n bits that assume upto 2^n distinct combinations of 1's and 0's. Numerous different codes can be obtained by arranging four bits in 10 distinct combinations.

Decimal number	Binary coded decimal (BCD) number
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Alpha Numeric representation

Many applications of digital computers require the handling of data that consist not only of numbers, but also letters of the alphabet and certain special characters. An alphanumeric character set is a set of elements that includes 10 digits, the 26 letters of the alphabet and special characters. The standard alphanumeric code is ASCII (American Standard Code for Information Interchange) which uses seven bits to code 128 characters.

ex:- 0 011 0000
1 011 0001
2 011 0010
- - -
9 011 1001

Complements - ② - 2

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation. There are 2 types of complements for each base r system. The r's complement and (r-1)'s complement.

(r-1)'s complement

For decimal numbers r = 10 and r-1 = 9.

So the 9's complement of N is $(10^n - 1) = N$.

$(10^n - 1) - N$. $10^n - 1$ is a number represented by n 9's.

ex: 9's complement of 546700 is

999999 - 546700 = 453299

For binary numbers $r = 2$ and $r - 1 = 1$. So the 1's complement of N is $(2^n - 1) - N$. $2^n - 1$ is a binary number represented by n 1's. for example 1's complement of 0001111 is 1110000.

r's complement

The r's complement of an n digit number N in base r is defined as $r^n - N$ for $N \neq 0$ and 0 for $N = 0$. The 10's complement of the decimal 2389 is $7610 + 1 = 7611$, and is obtained by adding 1 to the 9's complement. The 2's complement of binary 101100 is $010011 + 1 = 010100$ and is obtained by adding 1 to the 1's complement value.

Subtraction of Unsigned Numbers

The subtraction of two n digits unsigned numbers $M - N$ $(N \neq 0)$ in the base r can be done as follows

1. Perform $M + (r^n - N)$
2. If $M \geq N$, the sum will produce an end carry, which is discarded
3. If $M < N$, the sum does not produce an end carry

Ex:- $72532^{M} - 13250^{N}$

$M = 72532$

10's complement of $N = +86750$

159282

discard end carry 59282

Subtraction with complements is done with binary numbers in a similar manner using the same procedure

$X = 1010100$, $Y = 1000011$

$X = 1010100$

2's complement of $Y = 0111101$

10010001

discard carry 1, sum = 0010001

Fixed Point Representation

To represent negitive integers, we need a notation for negitive values. A negitive number is indicated by a minus sign and positive number is indicated by a plus sign. Computers must represent everything with 1's and 0's, including sign of a number. In addition to the sign, a number may have a binary (or decimal) point. The position of the binary point needed to represent fractions, integers, or mixed integer fraction numbers.

Integer Representation

When an integer binary number is positive, the sign is represented by 0 and the magnitude by a positive binary number. When the number is negitive sign is represented by 1 but the rest of the number may be represented in one of the possible ways

* Signed magnitude
* Signed 1's complement
* Signed 2's complement

For example +14 is reprented by a sign bit of 0 in the leftmost position followed by the binary equivalent of 14: 00001110. But there are three different ways to represent -14

Signed magnitude	1 0001110
signed 1's complement	1 1110001
signed 2's complement	1 1110010

Signed magnitude reprentation of -14 is obtained from +14 by complementing only the sign bit. The signed 1's complement representation is obtained by complementing all the bits of +14. including sign bit. Signed 2's complement representation is obtained by taking 2's complement of positive number, including its sign bit.

Arithmetic Addition

In Arithmetic addition the procedure is very simple and can be stated as follows. Add the two numbers, including their sign bits, discard carry out of the sign (leftmost) bit position

ex:-	+6	00000110	-6	11111010
	+13	00001101	+13	00001101
		00010011	+7	00000111
	+6	00000110	-6	11111010
	-13	11110011	-13	11110011
	-7	11111001	-19	11101101

Note that negative numbers must initially be in 2's complement and that if the sum obtained after the addition is negative, it is in 2's complement form. In each of 4 cases, the operation performed is always addition, including sign bits. Any carry out the sign bit position is discarded, and negative results are automatically in 2's complement form.

Arithmetic Subtraction

Subtraction of two signed binary numbers when negative numbers are in 2's complement form is very simple.

Take the 2's complement of the subtrahend (including sign bit) and add it to the minuend (including sign bit). A carryout of the sign bit position is discarded.

$$(\pm A) - (+B) = (\pm A) + (-B)$$
$$(\pm A) - (-B) = (\pm A) + (+B)$$

Ex: $(-6) - (-13) = +7$.

11111010 − 11110011

The subtraction is changed to addition by taking 2's complement of subtrahend (−13) to give (+13). In binary this is

11111010 + 00001101 = 100000111

Removing end carry, answer is 00000111 (+7).

Overflow:

An overflow is a problem in digital computers because width of registers is finite, and when it occurs, a corresponding flip-flop is set which can then be checked by the user.

Ex:-	+70	0	1000110
	+80	0	1010000
	+150	1	0010110

carries 0 1

Ex:-	−70	1	0111010
	−80	1	0110000
	−150	0	1101010

Carries 1 0

In the above example 8 bit result that should have been positive has a negative sign bit and the 8 bit result that should have been negitive has a positive sign. The carryout of sign bit is taken as sign bit of the result. The 9 bit answer so obtained will be correct. Since the answer can not be accomodated within 8 bits, we say an overflow occurred.

Decimal fixed point representation

A 4 bit decimal code requires 4 flip-flops for each decimal digit. The representation of 4385 in BCD requires 16 flip-flops, four flip-flops for each digit.

4385 → 0100 0011 1000 0101

Some applications, such as business data processing, require small amounts of arithmetic computations compared to the amount required for input and output of decimal data. Some computers have hardware for arithmetic calculations with both binary and decimal data.

Ex: (+375) + (−240) = +135 is done in the Signed 10's complement system

0 375	(0000 0011 0111 0101)$_{BCD}$	
+9 760	(1001 0111 0110 0000)$_{BCD}$	
0 135	(0000 0001 0011 0101)$_{BCD}$	

9760 is the 10's complement of 0240

Floating-Point Representation

The floating point representation has two parts. The first part represents a signed fixed point number called the mantissa. The second part designates the position of the decimal (or binary) point and is called the exponent.

	fraction	exponent
Ex:	+0.6132789	+04

↓

$+0.6132789 \times 10^{+4}$

Floating point is always interpreted to represent a number in the following form

$m \times r^e$

Only mantissa m and exponent e are physically represented in the register (including their signs). The radix r of the mantissa are always assumed

Ex: +1001.11 is represented with an 8 bit fraction and 6 bit exponent

fraction	exponent
01001110	000100

The fraction has a 0 in the leftmost position to denote positive. The exponent has the equivalent binary number +4

$$m \times 2^e = +(.1001110)_2 \times 2^{+4}$$

Arithmetic operations with floating point numbers are more complicated than arithmetic operations with fixed point numbers and their execution takes longer and requires more complex hardware.

Register Transfer Language

Chapter-4

Arithmetic Microoperations

A Microoperation is an elementary operation performed with the data stored in registers.

* Register transfer microoperations transfer binary information from one register to another
* Arithmetic Microoperations perform arithmetic operations on numeric data stored in registers
* Logic Microoperations perform bit manipulation operations on non numeric data stored in registers.
* Shift Microoperations perform shift operations on data stored in registers

Ex:- $R_3 \leftarrow R_1 + R_2$

The contents of R_1 plus R_2 transfered to R_3

Ex:- $R_3 \leftarrow R_1 - R_2$

The contents of R_1 minus R_2 transferred to R_3

Ex:- $R_2 \leftarrow \bar{R}_2$

Complements the contents of R_2

Ex:- $R_2 \leftarrow \bar{R}_2 + 1$

2's complement the contents of R_2

Ex:- $R_3 \leftarrow R_1 + \bar{R}_2 + 1$

R_1 plus the 2's complement of R_2

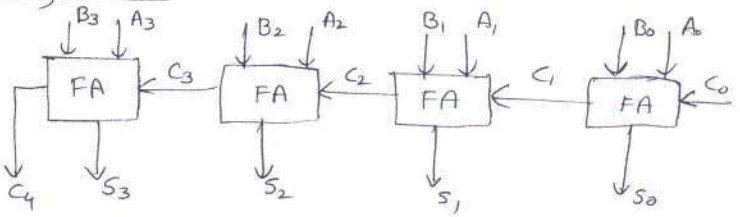
Ex: $R_1 \leftarrow R_1 + 1$

Increment the contents of R_1 by one

Ex: $R_1 \leftarrow R_1 - 1$

Decrement the contents of R_1 by one.

Binary adder

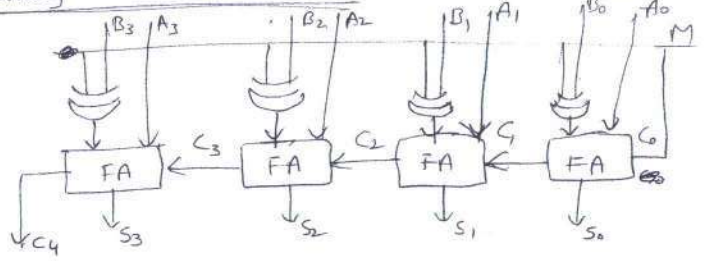


4 bit binary adder

The figure shows the interconnections of full-adders (FA) to provide a 4 bit binary adder. The carries are connected in a chain through the full adders. The input carry to the binary adder is C_0 and the output carry is C_4. The S outputs of the full adders generate the required sum bits.

n data bits for the A inputs come from one register (such as R_1), and n data bits for the B inputs come from another register (such as R_2). The sum transffered to third register.

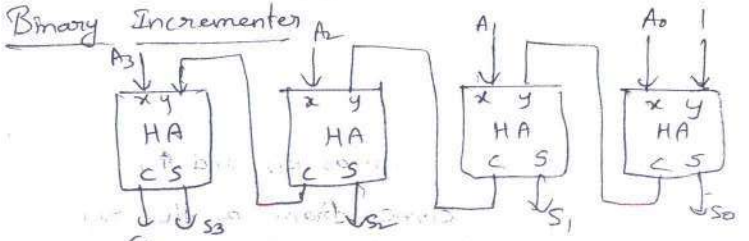
Binary adder - subtractor



The subtraction can be done by taking 2's complement of B and adding it to A. The addition and subtraction operations can be done combined into one common circuit by including exclusive OR gate with each full adder. The mode input M controls the operation. When M=0 the circuit is an adder and M=1 the circuit becomes a subtractor.

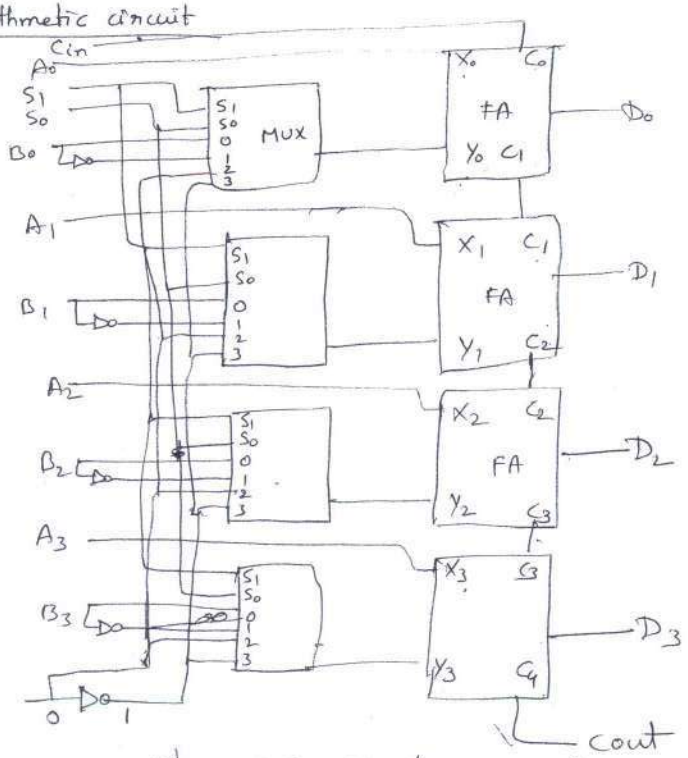
When M=0, we have $B \oplus 0 = B$. The full adder receive the value of B, The input carry is 0, and the circuit performs A plus B. when M=1 we have $B \oplus 1 = B'$ and $C_0 = 1$. The circuit performs the operation A plus 2's complement of B

Binary Incrementer



For example, if a 4 bit register has a binary value 0110, it will goto 0111 after it is incremented. This microoperation is easily implemented with a binary counter. Every time the count enable is active, the clock pulse transition increments the content of the register by one.

Arithmetic circuit



When $S_1 S_0 = 00$, the value of B is applied to the Y inputs of the adder. If $c_{in} = 0$, then $D = A + B$, if $C_{in} = 1$ output $D = A + B + 1$

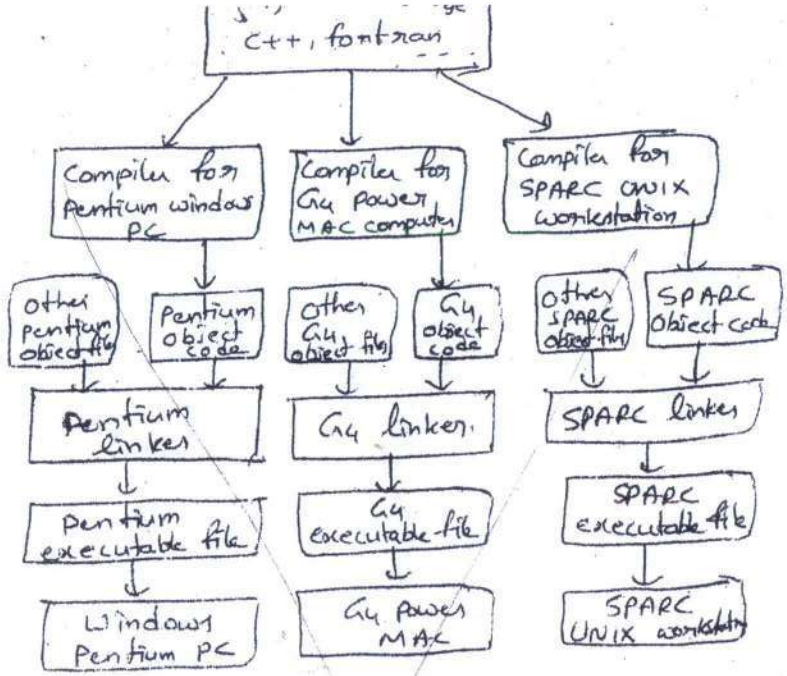
When $S_1 S_0 = 01$, the complement of B is applied to Y inputs of adder. If $c_{in} = 1$ then $D = A + \bar{B} + 1$, when $c_{in} = 0$ then $D = A + \bar{B}$.

When $S_1 S_0 = 10$, the inputs from B are neglected, all 0's are inserted into Y inputs.

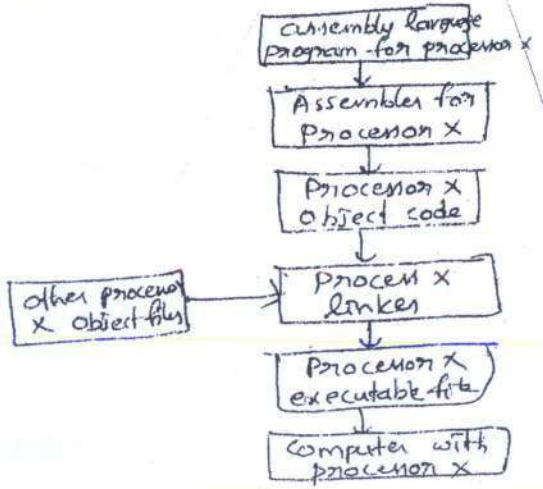
$D = A$ when $C_{in} = 0$, $D = A + 1$ when $C_{in} = 1$.

$(D = A + 0 + C_{in})$

When $S_1 S_0 = 11$ all 1's are inserted into the Y inputs of the adder to produce decrement operation $D = A - 1$ when $C_{in} = 0$, when $C_{in} = 1$ then $D = A - 1 + 1 = A$, which means direct transfer from input A to output D.



A high level language statement is usually converted to a sequence of several machine code instructions. The below diagram shows the process of converting an assembly language program to an executable form.



its source code code to object code. It follows the linking and loading procedure that was used for compiled code.

(2) Assembly language Instructions - 2Q

A microprocessor's assembly language instruction set is an important part of its instruction set architecture.

Instruction types: Assembly language instructions can be grouped together based on the types of operations they perform.

1. Data transfer instructions
2. Data operation instructions.
3. program control instructions

Data transfer instructions

The most common operation a microprocessor performs is to move data from one place to another

* Load data from memory into the Microprocessor
* Store data from the microprocessor into memory
* Move data within the microprocessor
* Input data to the Microprocessor
* Output data from the Microprocessor

A Microprocessor usually contains several registers to store data that it is using. These instructions copy data from memory into a microprocessor register. Store data from the microprocessor into memory is similar to loading data, except data is copied in the opposite direction, from a microprocessor register to memory. We can move data from one register to another register. The microprocessor inputs data from the input device, the keypad. The microprocessor copies data from one of its registers to an output device.

Data Operation Instructions

Data operation instructions do modify their data values. Arithmetic instructions make up a large part of the data operation instructions. Instructions that add, subtract, multiply or divide values fall into this category.

Logic instructions perform basic logical operations on data. They AND, OR or XOR two data values, or complement a single value. These operations are done in a bit wise manner. They are $A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0$ and $B_7, B_6, B_5, B_4, B_3, B_2, B_1, B_0$ and operation results $A_7 \wedge B_7$, $A_6 \wedge B_6$, $A_5 \wedge B_5$, $A_4 \wedge B_4$, $A_3 \wedge B_3$, $A_2 \wedge B_2$, $A_1 \wedge B_1$, $A_0 \wedge B_0$.

Program control instructions

High level languages have constructs, such as for.. Do and Do.. while to control the flow of a program without resorting to the dreaded goto statement. Assembly languages do not have this option.

Assembly language programs may include subroutines, just as high level language programs may have subroutines, procedures, and functions. A Microprocessor can be designed to accept interrupts. An interrupt basically causes the microprocessor to stop what it is doing and start executing other instructions.

An assembly language instruction set may include specific instructions to generate interrupt, these are called software interrupts. There are also Interrupts that are not part of the instruction set. Hardware interrupts are triggered by devices outside of the microprocessor.

Data Types

A Microprocessor may have to operate on more than one type of data. As a result, its instruction set may include different instructions to perform the same operation on different data types.

Instruction	1st Loop	2nd Loop	3rd Loop	4th Loop	5th Loop
CLAC	AC=0				
STAC total	total=0				
STAC i	i=0				
LDAC i	AC=0	AC=1	AC=2	AC=3	AC=4
INAC	AC=1	AC=2	AC=3	AC=4	AC=5
STAC i	i=1	i=2	i=3	i=4	i=5
MVAC	R=1	R=2	R=3	R=4	R=5
LDAC total	AC=0	AC=1	AC=3	AC=6	AC=10
ADD	AC=1	AC=3	AC=6	AC=10	AC=15
STAC total	total=1	total=3	total=6	total=10	total=15
LDAC n	AC=5	AC=5	AC=5	AC=5	AC=5
SUB	AC=4, z=0	AC=3 z=0	AC=2 z=0	AC=1 z=0	AC=0 z=1
JPNZ Loop	JUMP	JUMP	JUMP	JUMP	NOJUMP

(3) 8085 Microprocessor instruction set architecture - 3Q

The 8085 microprocessor contains several general purpose data registers. As in the relatively simple C.P.U the 8085 has an accumulator register. This register always receives the result of an 8 bit arithmetic or logical instruction. The 8085 has six general purpose registers.

registers, named B, C, D, E, H and L. The B and C are always a pair, as are D and E, and H and L. SP contains the address of the top of the stack.

The 8085 microprocessor has five flags

Sign flag : 'S' indicates the sign of a value calculated by an arithmetic or logical instruction

zero flag : 'Z', works the same as in the relatively simple CPU. It is set to 1 if an arithmetic or logical instruction produces a result of 0, it is set to 0, otherwise

parity flag : 'P' is set to 1 if the result of an arithmetic or logical operation has an even no of 1's, otherwise it is set to 0.

Carry flag : 'C y' is set when an arithmetic operation generates a carry out.

Auxiliary carry flag : AC, is similar to the carry flag. Instead of indicating a carry out, it denotes a carry from the lower half of the result to the upper half.

The 8085 Microprocessor Instruction Set

The 8085 instruction set contains a total of 74 instructions. We break the instruction set into three parts. data movement instructions, data operation instructions, and program control instructions.

* r, r1 and r2 indicate any 8 bit register A, B, C, D, E, H or L.
* M indicates a memory location. When a location is represented as M[HL], the address of the memory location is stored in register pair HL
* rp indicates register pair BC, DE, HL or Stack pointer SP.
* r is a 16-bit address or data value. As in the relatively simple CPU, the low order 8 bits of r are stored in the memory location
* n is an 8 bit address or data value stored in memory immediately after the opcode.
* cond is a condition for conditional instructions. It can have the following values

NZ (Z = 0)	Z (Z = 1)
P (S = 0)	N (S = 1)
PO (P = 0)	PE (P = 1)
NC (CY = 0)	C (CY = 1)

Data Movement instruction for the 8085 Microprocessor

instruction	operation
NOP	no operation
MOV r1, r2	r1 = r2
MOV r, M	r = M[HL]
MOV M, r	M[HL] = r
MVI r, n	r = n
MVI M, n	M[HL] = n
LDA r	A = M[r]
STA r	M[r] = A
LHLD r	HL = M[r], M[r+1]
SHDL r	M[r], M[r+1] = HL
LDAX rp	A = M[rp] (rp = BC, DE)
STAX rp	M[rp] = A (rp = BC, DE)
XCHG	DE ⇔ HL
PUSH rp	Stack = rp (rp ≠ SP)
push PSW	Stack = A, Flag register
POP rp	rp = Stack (rp ≠ SP)
POP PSW	A, Flag register = stack
XTHL	HL ⇔ Stack
SPHL	SP = HL
IN n	A = Input port n
Out n	Output n = A.

The MOV instructions move data from one place to another either between registers or between a register and memory. When MOV moves data to or from memory, it uses the 16 bit value in the register pair HL as the memory address. For example if HL = 1234H, the instruction MVI M, 56 Moves the data value 56 to memory location 1234H. The LXI is a 16 bit immediate move. LDA, STA are direct mode instructions, they transfers the data between the memory location at address r and the accumulator register A

data between register pair HL and memory. The LDAX and STAX instructions are similar to the MOV r, M and MOV M, r instructions, except they only move data to and from register A. The XCHG instruction swaps the contents of register pairs DE and HL.

The push rp operation pushes the contents of a register pair onto the stack. POP instruction is the opposite of push, read data off of the stack and write it into the register. The XTHL instruction exchanges the top 2 elements of the stack. SPHL copies the contents of register pair HL to the stack pointer SP. The IN and OUT instructions access I/O devices for ~~this~~ microprocessor.

Ex:-

A simple 8085 program

Instruction	operation
JUMP r	Goto r
Jcond r	If condition is true then Goto r
PCHL	Goto address in HL
Call r	call subroutine at r
C cond r	If condition is true then call subroutine at r
RET	return from subroutine

RSTn	call subroutine at 8*n (n = 5.5, 6.5, 7.5)
RIM	A = IM
SIM	IM = A) X
DI	disable interrupts
EI	enable interrupts
HLT	Halt the CPU

Ex:-

1: i = n, sum = 0
2: sum = sum + i, i = i - 1
3: if i ≠ 0 then goto 2
4: total = sum

```
      LDA n   } i = n
      MOV B,A }
      XRA A   } sum = A ⊕ A = 0
Loop: ADD B   } sum = sum + i
      DCR B   } i = i - 1
      JNZ loop} if i ≠ 0 then goto Loop
      STA total} total = sum
```

Instruction	1st Loop	2nd Loop	3rd Loop	4th Loop	5th Loop
LDA n MOV B,A	B = 5				
XRA A	A = 0				
ADD B	A = 5	A = 9	A = 12	A = 14	A = 15
DCR B	B = 4 Z = 0	B = 3 Z = 0	B = 2 Z = 0	B = 1 Z = 0	B = 0 Z = 1
JNZ Loop	JUMP	JUMP	JUMP	JUMP	NO JUMP
STA total					total = 15

This reduces the size of the circuit board needed by the computer and wiring between the components, both of which reduce the cost of manufacturing the system. This system has 2k of EPROM starting at location 0. Its address range is 0 to 2k-1, or 0000 0000 0000 0000 to 0000 0111 1111 1111 in binary. The system also has 256bytes of RAM starting at address 4k. Its range is 0001 0000 0000 0000 to 0001 0000 1111 1111. It has 4 8 bit I/O ports at addresses 00H, 01H, 19H and 1AH and a 6 bit I/O port

Register Transfer Languages-4Q (2)

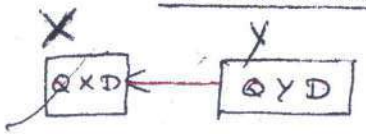
Micro Operations

Micro-operations are the basis for most sequential digital systems. Microoperations are much simpler actions: the movement of data from one register, memory location, or I/o device to another, modify stored values, such as incrementing or clearing a register perform arithmetic or logical functions, such as adding two values and storing the result in a register and otherwise modify a stored value.

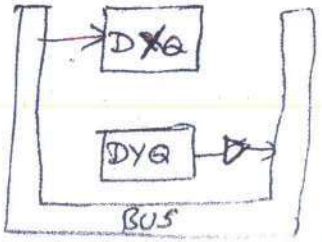
Microoperations and Register transfer language

A Microoperation, specifies an operation whose result is stored, typically in a register or memory location. Consider a digital syste with two 1 bit registers, x and y. The micro operation that copies the contents of register y to register x can be expressed as $X \leftarrow Y$. The micro operation may be implemented via a direct connection,

a direct connection



bus connection

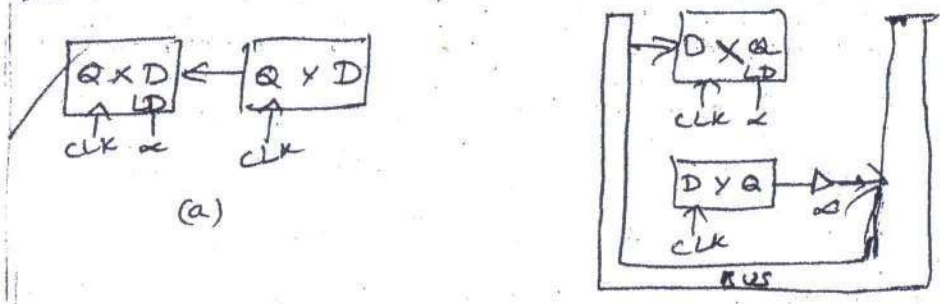


both signs provide a path for data to flow from register Y to register X, but neither specifies when X should load this data. Assume that the transfer should occur when control input α is high. The transfer could be written as

If α then $X \leftarrow Y$

$\alpha : X \leftarrow Y$

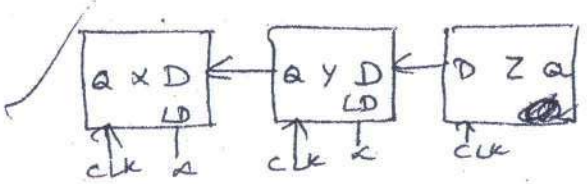
Ex-2



Here α is used to load register X and, in the bus based implementation, to enable the tri-state buffer so that the contents of register Y are placed on the bus.

$\alpha : X \leftarrow Y, Y \leftarrow Z$

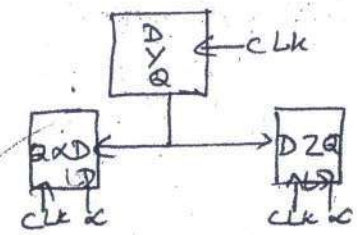
Ex-3



A single bus can not be used here because a bus can hold only one value at a time.

$\alpha : X \leftarrow Y, Y \leftarrow Z$

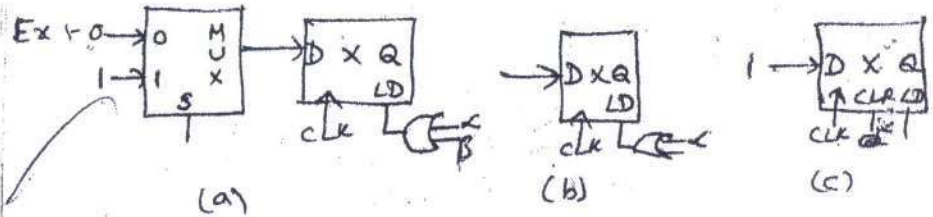
Ex-4



Register Y can be read by many other registers simultaneously. both Microoperations can be performed concurrently. Some times it is necessary to move a constant value into a register, rather than data from another register.

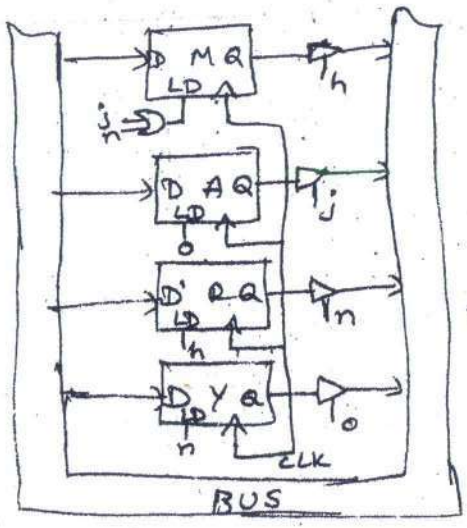
$\alpha : X \leftarrow 0$

$\beta : X \leftarrow 1$



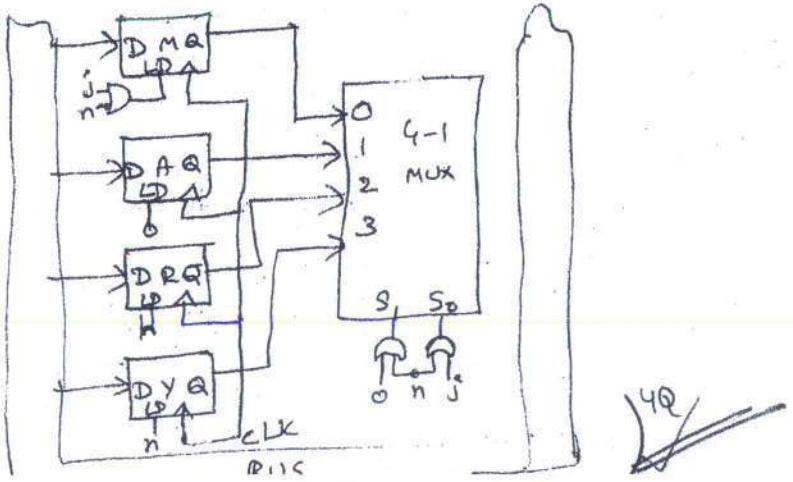
The figure shows three different ways to perform ~~those~~ transfers. The first circuit sets x by loading data for both transfers. Either α or β can assert the register's load signal. If $\alpha=0$ and $\beta=0$, the multiplexer still outputs data. In the case of second circuit when $\alpha=1$, $\beta=0$ register X loads in the data value 0, when $\beta=1$ and $\alpha=0$, it loads in the value 1. When both $\alpha=0$, $\beta=0$ the input data value 0 is ignored by the register because its load signal is not asserted. In the case of The final circuit when $\alpha=1$, register x is set to 0 by being cleared. when $\beta=1$, the register is loaded with value 1.

RTL code using a bus and tristate buffer



The output of each register must pass through a tri-state buffer before it is placed onto the bus. only one buffer may be enabled at any time. The system includes the RTL statement j: M ← A therefore, j must enable the buffer that places the contents of register A onto the bus.

RTL code using a bus and multiplexer



4Q

More complex digital systems and RTL

Modulo 6 counter:-

The modulo 6 counter is a 3 bit counter that counts through the sequence.

$000 \to 001 \to 010 \to 011 \to 100 \to 101 \to 000 - - -$

$(0 \to 1 \to 2 \to 3 \to 4 \to 5 \to 0 \to - - - -)$

Its input U controls the counter. When U=1, the counter increments its value on the rising edge of the clock. when U=0, it retains its current value regardless of the value of the clock. The value of the count is represented as the 3 bit output $V_2 V_1 V_0$. An additional output, C is 1 when going from 5 to 0, and 0 otherwise. In this example the c output remains at 1 until the counter goes from 0 to 1.

The finite state machine for this counter must have six states. S_0, S_1, S_2, S_3, S_4 and S_5

$S_0 \to S_1 \to S_2 \to S_3 \to S_4 \to S_5 \to S_0 \to - - -$

Present state	U	Next state	C	$V_2 V_1 V_0$
S_0	0	S_0	1	000
S_0	1	S_1	0	001
S_1	0	S_1	0	001
S_1	1	S_2	0	010
S_2	0	S_2	0	010
S_2	1	S_3	0	011
S_3	0	S_3	0	011
S_3	1	S_4	0	100
S_4	0	S_4	0	100
S_4	1	S_5	0	101

when $S_1 S_0 = 10$, the inputs from B are neglected, all 0's are inserted into Y inputs.

$D = A$ when $C_{in} = 0$, $D = A + 1$ when $C_{in} = 1$.
$(D = A + 0 + C_{in})$

When $S_1 S_0 = 11$ all 1's are inserted into the Y inputs of the adder to produce decrement operation $D = A - 1$ when $C_{in} = 0$, when $C_{in} = 1$ then $D = A - 1 + 1 = A$, which means direct transfer from input A to output D.

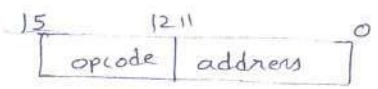
Chapter - 5

Instruction codes – Q-1

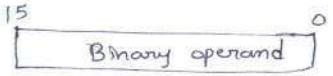
The organization of computer is defined by its internal registers, the timing and control structure, and the set of instructions that it uses. A computer instruction is a binary code that specifies a sequence of microoperations for the computer. Instruction codes together with data are stored in memory. The computer reads each instruction from memory and places it in a control register. The control then interprets the binary code of the instruction and proceed to execute by issuing sequence of microoperation.

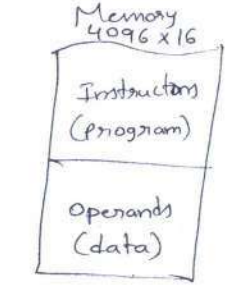
An instruction code is a group of bits that instruct the computer to perform a specific operation. It is usually divided into parts. The most basic part of an instruction code is its operation part. The operation code of an instruction is a group of bits that define such as add, subtract, multiply, shift and complement. When this operation code is decoded in the control unit, the computer issues control signals ~~to~~ to read the operand from memory.

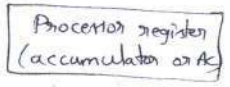
Stored program organization



Instruction format





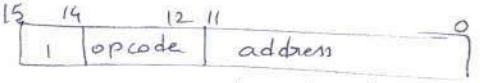


An instruction code format with 2 parts. The first part specifies the operation to be performed and the second specifies an address. The memory address tells the control where to find an operand in memory. This operand is read from memory. For a memory unit with 4096 words we need 12 bits to specify an address since $2^{12} = 4096$. If we store each instruction code in one 16 bit memory word, we have available 4 bits for the operation code.

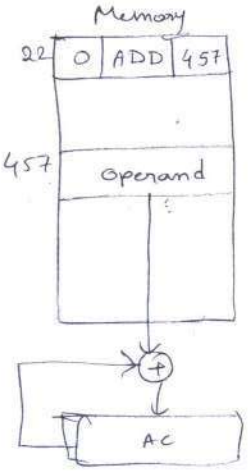
Computers that have a single-processor register usually assign to it the name accumulator. The operation is performed with the memory operand and the content of AC.

Indirect Address

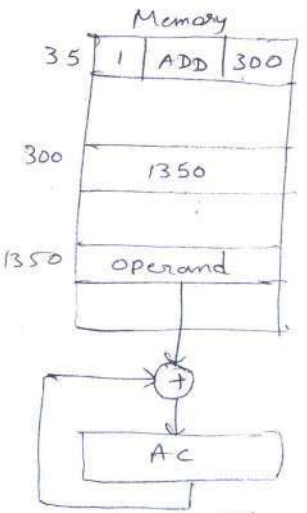
When the second part of an instruction code specifies an operand, the instruction is said to have an immediate operand, when the second part specifies the address of an operand, the instruction is said to have a direct address. When the second part of the instruction designate an address of a memory word in which the address of the operand is found, it is called indirect address. The indirect address instruction needs two references to memory to fetch operand. The first reference is needed to read the address of the operand. The second is for the operand itself. We define the effective address is the target address in a branch type instruction.



Instruction format



Direct address



Indirect Address

Instruction set Completeness

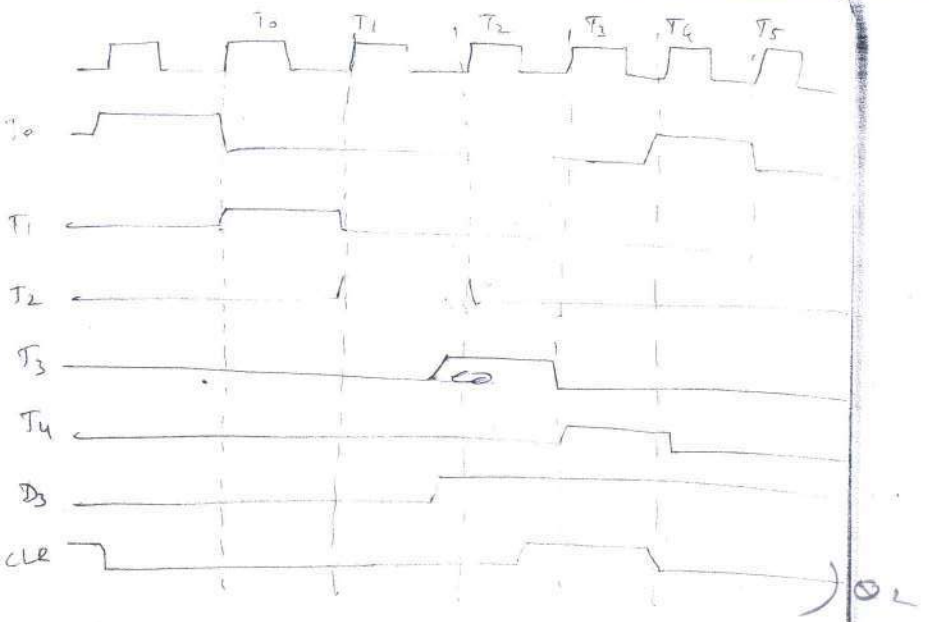
The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories

* Arithmetic, logical and shift instructions
* Instructions for moving information to and from memory and processor registers
* Program control instructions together with instructions.
* Input and output instructions) ①1

Timing and control - 2①

The timing for all registers in the basic computer is controlled by a master clock generator. The clock pulses are applied to all flip flops and registers.

There are 2 types of control organization: hardwired control and Microprogrammed control. In the hardwired control the control logic is implemented with gates, flip-flops, decoders and other digital circuits. In the microprogrammed organization, the control memory is programmed to initiate the required sequence of microoperations



①2

Memory Reference Instructions (Computer instructions) ①3

The data must be read from a memory to a register where there can be operated on with logic circuits.

Symbol	operation decoder	description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR]$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC$, $PC \leftarrow AR+1$
ISZ	D_6	$M[AR] \leftarrow M[AR]+1$ If $M[AR]+1 = 0$ then $PC = PC+1$

AND to AC

This is an instruction that performs the logic AND operation on pairs of bits in AC The result of the operation ~~transffe~~ transferred to AC.

$DR \leftarrow M[AR]$
$AC \leftarrow AC \wedge DR$

ADD to AC

The instruction adds the contents of the memory word to the accumulator.

$DR \leftarrow M[AR]$
$AC \leftarrow AC + DR$

LDA

This instruction transfers the memory word to the Accumulator.

$DR \leftarrow M[AR]$
$AC \leftarrow DR$

STA

This instruction stores the content of Ac into the memory.

$M[AR] \leftarrow AC$

BUN

This instruction transfers the program to the instruction specified by the effective address. PC is the program counter which stores the address of the next instruction to be executed

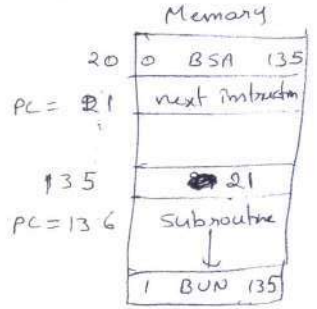
$PC \leftarrow AR$,

BSA

This instruction is useful for branching a Portion of the program called a subroutine or Procedure.

$M[AR] \leftarrow PC$,
$PC \leftarrow$ ~~PC~~ $AR + 1$

When executed, the BSA instruction stores the address of the next instruction in sequence into a memory location specified by the effective address

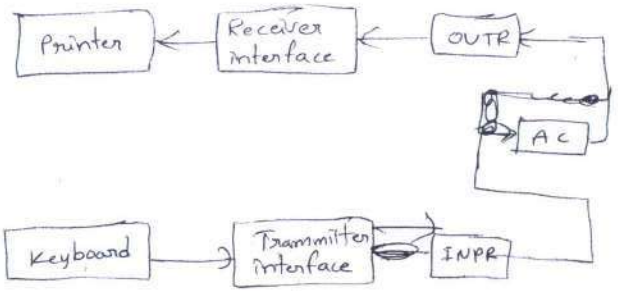


ISZ : Increment and skip if zero

This instruction increments the word specified by the effective ~~add~~ address. If the incremented value is equal to 0, PC is incremented by 1.

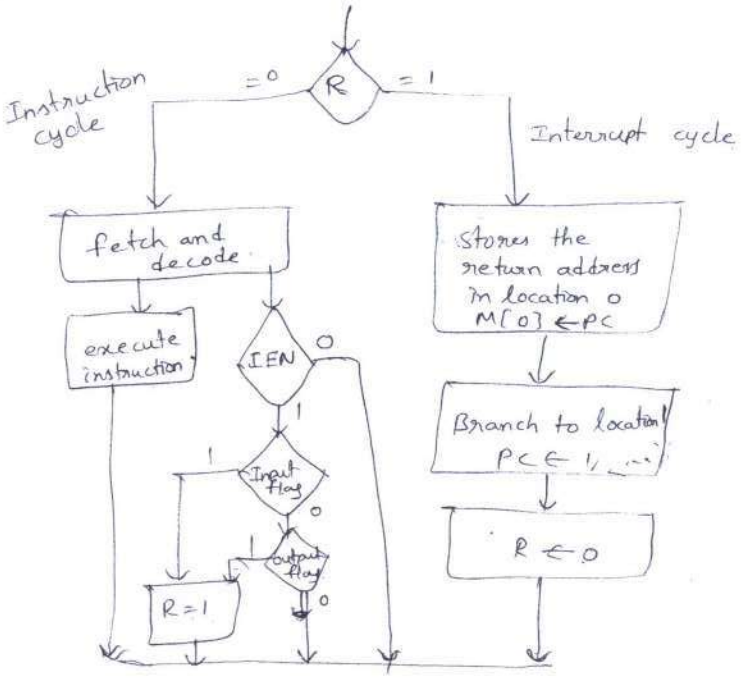
Input-output and Interrupt

Instructions and data stored in memory must come from some input input device. Computational results must be transmitted to the user through some output device.

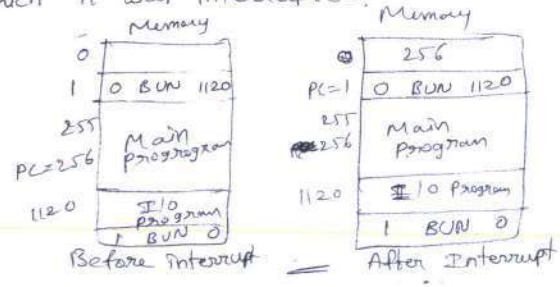


The serial information from the keyboard is shifted in into the input register INPR. The serial information for the printer is stored in the output register OUTR. These two registers communicate with a communication interface serially and with the AC in parallel. The transmitter interface receives serial information from the keyboard and transmits it to INPR. The receiver interface receives the information from OUTR and sends it to the Printer serially.

INP	Input character
OUT	Output character
SKI	skip on input flag
SKO	Skip on output flag
ION	Interrupt enable on
IOF	Interrupt enable off



When R=0, the computer goes to the instruction cycle. when R=1 the computer goes to the interrupt cycle. Interrupt cycle is the hardware implementation of a branch and save return address operation. The return address stored in a specific location where it can be found later when the program returns to the instruction at which it was interrupted.



blank

ch-5 (2) This relatively simple CPU contains several registers.

* A 16 bit address register (AR)
* A 16 bit program counter (PC)
* An 8 bit data register (DR)
* An 8 bit instruction register (IR)
* An 8 bit temporary register (TR)

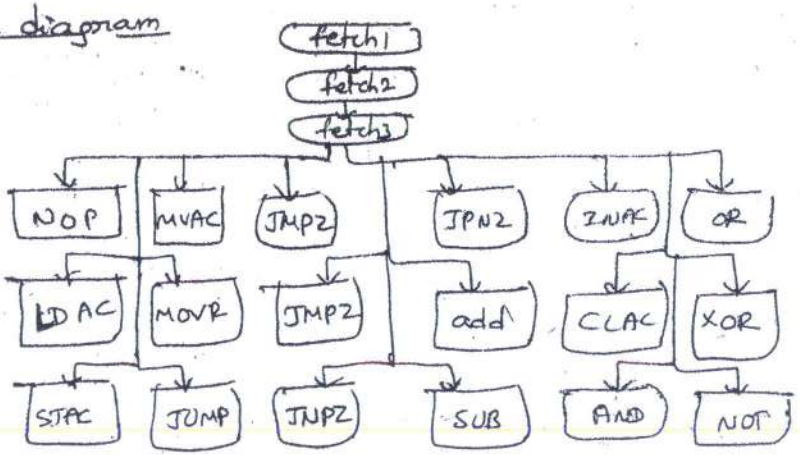
Input register-
output register-OU

Fetching and decoding

This CPU fetches instructions from memory in exactly the same way as the very simple CPU does, except at the end of the fetch cycle. Here IR is 8 bits and receives the entire contents of DR.

fetch1: AR ← PC
fetch2: DR ← M, PC ← PC+1
fetch3: IR ← DR, AR ← PC

state diagram



The final task in creating the state diagram for this CPU is to prepare the state diagrams for the execute routines.

NOP

Nop means NO operation.

LDAC

LDAC contains three words the opcode, the low order half of the address, and the high order half of the address.

LDAC1: DR←M, PC←PC+1, AR←AR+1
LDAC2: TR←DR, DR←M, PC←PC+1
LDAC3: AR←DR, TR
LDAC4: DR←M
LDAC5: AC←DR

STAC:

STAC performs the opposite operation of LDAC

STAC1: DR←M, PC←PC+1, AR←AR+1
STAC2: TR←DR, DR←M, PC←PC+1
STAC3: AR←DR, TR
STAC4: DR←AC
STAC5: M←DR.

MVAC and MOVR

MVAC1: R←AC
MOVR1: AC←R

JUMP:

JUMP 1: DR←M, AR←AR+1
JUMP 2: TR←DR, DR←M
JUMP 3: PC←DR, TR
JMPZ Y_1: "
JPNZ Y_1: "
JMPZ N_1: PC←PC+1
PC←PC+1
JPNZ N_1: PC←PC+1
PC←PC+1

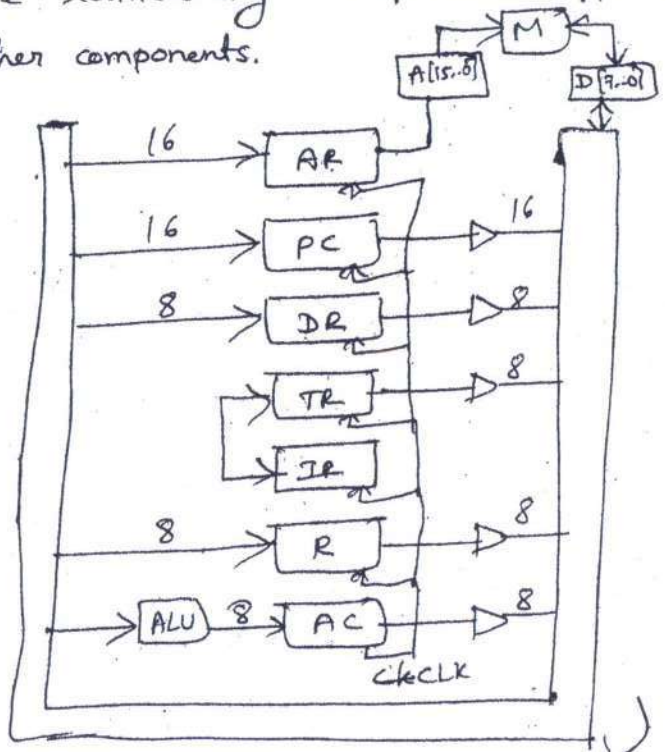
ADDI : AC←AC+R if (AC+R = 0) then Z←1 else "
SUBI : AC←AC−R if (AC−R=0) "
INAC1 : AC←AC+1 If (AC+1=0) "
CLAC1 : AC←0, Z←1
ANDI : AC←AC∧R, If (AC∧R=0) "
ORI : AC←AC∨R, if (AC∨R=0) "
XORI : AC←AC⊕R, If (AC⊕R=0) "
NOT1 : AC←AC′, If (AC′=0) "

Establishing data paths

We regroup the data transfers by destination.

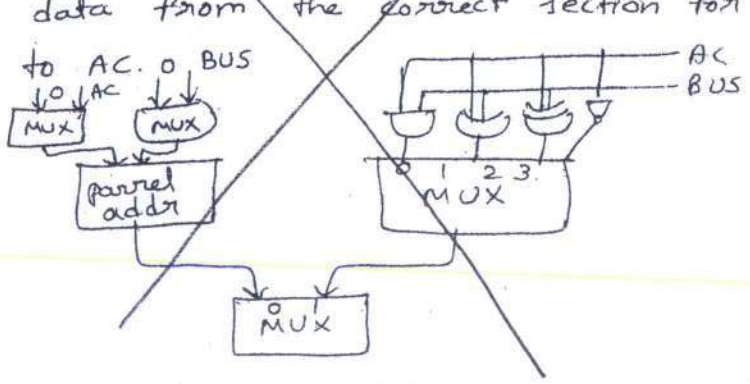
AR : AR←PC, AR←AR+1, AR←DR, TR
PC : PC←PC+1, PC←DR, TR
DR : DR←M, DR←AC
IR : IR←DR
R : R←AC
TR : TR←DR
AC : AC←DR, AC←R, AC←AC+R, ~~AC←AC−R,~~
AC←AC+1, AC←0, AC←AC∧R, ~~AC←~~
AC←AC∨R, AC←AC⊕R, AC←AC′
Z : Z←1, Z←0. (both conditions)

1. AR, IR does not supply data to other compone
2. The ramaining components supplies data to the other components.



5Q

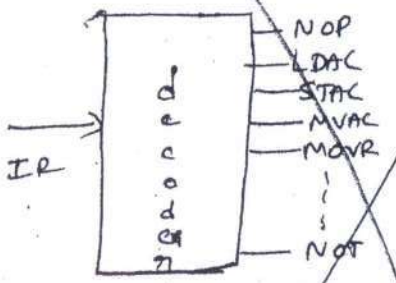
An arithmetic/logic unit can be (ALU) designed just as its name implies. A multiplexer selects data from the correct section for output to AC. 0 BUS



The first input ...

either the contents of AC or 0. The ALU can use a multiplexer to select one of these two values. Similarly ALU uses a multiplexer to send ~~BS~~ BUS or 0 the second input. There are 4 logical operations. They are AC∧BUS, ~~AV~~ ACVBUS, AC⊕BUS, and AC'.

Control unit



~~Control signal values~~ The CPU has a total of 37 states, making it too complex to implement efficiently using the same design. Instead of using one register to generate the state, this control unit uses two registers and combines their outputs.

PC BUS	fetch1 V fetch3
DR bus	LDAC3 V STAC3 V JUMP3 V JMP2Y.
~~DR~~	V JPN2Y3 V LDAC5 V STAC5
TR bus	LDAC3 V STAC3 V JUMP3 V JMP2Y3 V
	JPN2Y3 V JUMP3
AC bus	STAC4 V MVAC1
Membus	fetch2 V LDAC1 V LDAC2 V LDAC4 V STAC1 V STAC2 V JUMP1 V JUMP2 V JMP2Y1 V JMP2Y2 V JPN2Y1 V JPN2Y2

The control bus is [illegible]

the other two buses. The address bus consists of n lines, which combine to transmit one n bit address value. Similarly, the lines of the data bus work together to transmit a single, multibit value. Control bus is a collection of individual control signals. These signals indicate whether the data is to be read into or written out of the CPU.

Instruction Cycles -6Q

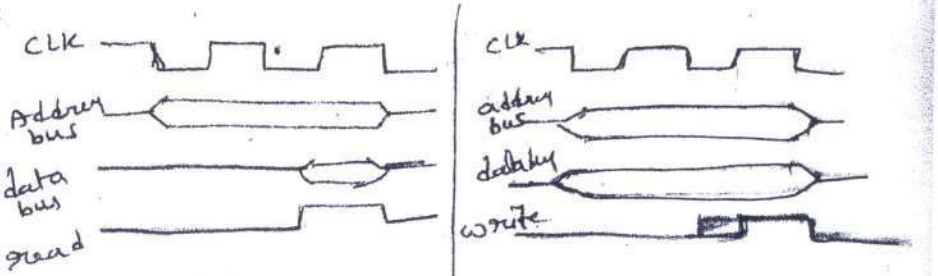
Instruction cycle is the procedure a microprocessor goes through to process an instruction. First the microprocessor fetches, or reads, the instruction from memory. Then it decodes the instruction, determining which instruction it has fetched. Finally it performs the operations necessary to execute the instruction.

After microprocessor allows sufficient time for memory to decode the address and access the requested memory location, the microprocessor asserts read control signal. Read signal is a signal on the control bus which the microprocessor asserts. when it is ready to read data from memory or an I/o device.

ready to read data from memory

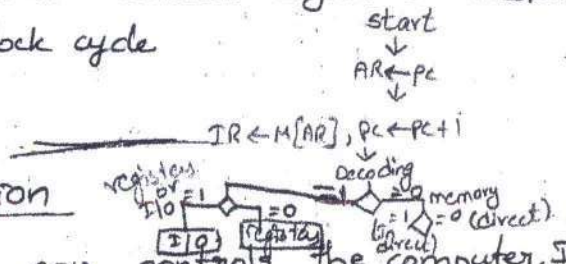
When the read signal is asserted, the memory subsystem places the instruction code to be fetched on to the computer system's data bus. The microprocessor then inputs this data from the bus and stores it in one of its internal registers. At this point the microprocessor has fetched the instruction.

Next, the microprocessor decodes the instruction. After it determines which instruction it is in order to select the correct sequence of operations to perform. This is done entirely within the microprocessor. It does not use the system buses.



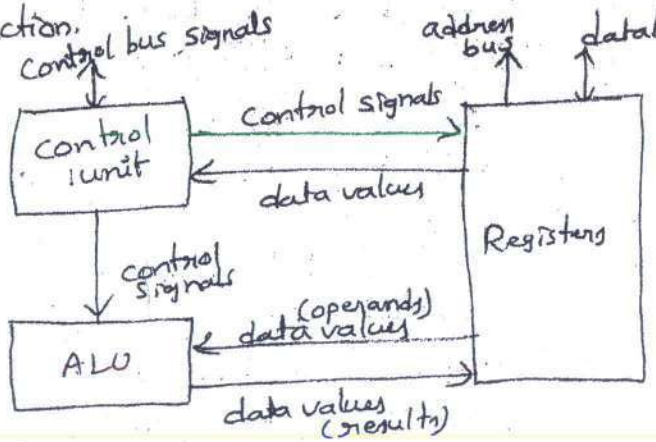
To read data from memory, the microprocessor performs the same sequence of operations it uses to fetch an instruction from memory. The microprocessor uses the system clock to synchronize its operations. The Microprocessor places the address onto the bus at the beginning of a clock cycle.

The timining of the memory write operation is shown in the figure. The processor places the address and data onto the system buses during the first clock cycle. The microprocessor then asserts a write control signal at the start of the second clock cycle



CPU organization

The CPU controls the computer. It fetches instructions from memory. The CPU decodes the instructions and controls the execution procedure. It performs some operations internally and supplies the address, data and control signals needed by memory and I/O device to execute the instruction.



CPU has a register section, includes a set of registers and a bus. The processor has a register called the program counter. The CPU keeps the address of the next instruction to be fetched in this register. Before the CPU outputs the address onto the system's address bus, it retrieves the address from the program counter register. At the end of the instruction fetch, the CPU reads the instruction code from the system data bus. It stores this value in a internal register, usually called the instruction register.

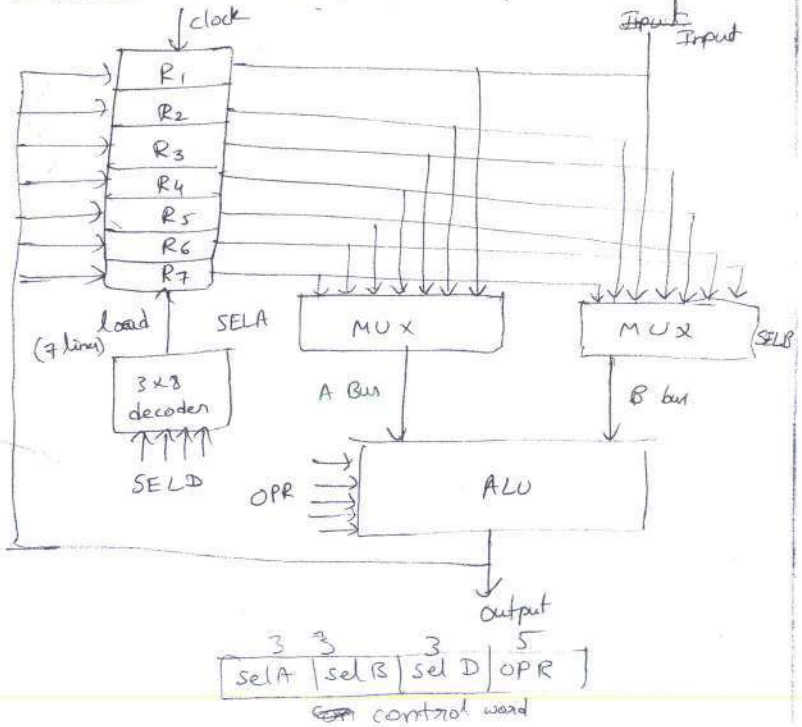
ALU performs most arithmetic and logical operations, such as adding or ANDing values. It receives its operands from the register section of the CPU and stores its results back in the register section. ALU must complete its operations within a single clock cycle.

The control unit controls the CPU. This unit generates the internal control signals that cause registers to load data, increment or clear their contents, and output their contents, as well as cause the ALU to perform the correct function.) 6Q

Chapter - 6

6.1 General Register Organization

Memory locations are needed for storing pointers, counters, return addresses, temporary results. Having to refer to memory locations for such applications is time consuming because memory access is the most time-consuming operation in a computer. It is more convenient and more efficient to store these intermediate values in processor registers. When a large number of registers are included in the CPU, it is efficient to connect them through a common bus system.



3	3	3	5
SelA	SelB	Sel D	OPR

control word

In the above figure the outputs of each register is connected to two multiplexers to form the two buses A and B. The select lines in each multiplexer select one register or the input data for the particular bus. The A and B buses form the input to a common ALU. The operation selected in the ALU determines the arithmetic or logic microoperation that is to be performed. The result of the microoperation is available for output data and also goes into the inputs of all the registers. The register that receives the information from the output bus is selected by a decoder. The decoder activates one of the register load inputs, providing a transfer path between the data in the output bus and the inputs of the selected destination register.

The 14 bit control word is defined in the above figure. It consists of 4 fields. Three fields contain three fields each, one field has five bits. The three bits selA select a source register for the A input of the ALU. Sel B, sel D performs the same action like sel A. The five bits of OPR select one of the operations in the ALU.

encoding of ALU operations

OPR select	operation	symbol
00000	transfer A	TSFA
00001	increment A	INCA
00010	add A+B	ADD
00101	Subtract A-B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

The binary control word for the Subtract microoperation is 010 011 001 00101 and is obtained as follows

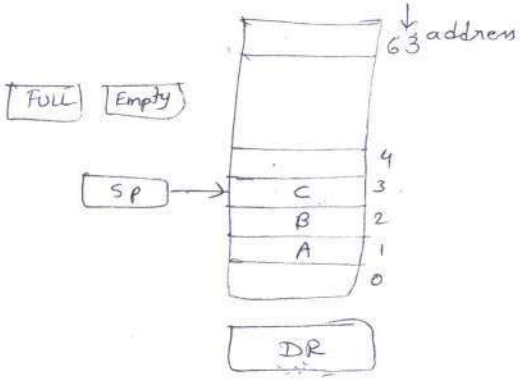
Field	sel A	sel B	sel D	OPR
Symbol	R_2	R_3	R_1	sub
Control word	010	011	001	00101

examples of Microoperations for the CPU

Micro operation	SELA	SELB	SELD	OPR	Control word
$R_1 \leftarrow R_2 - R_3$	R_2	R_3	R_1	SUB	010 011 001 00101
$R_4 \leftarrow R_4 \vee R_5$	R_4	R_5	R_4	OR	100 101 100 01010
$R_6 \leftarrow R_6 + 1$	R_6	—	R_6	INCA	110 000 110 00001
$R_7 \leftarrow R_1$	R_1	—	R_7	TSFA	001 000 111 00000
output $\leftarrow R_2$	R_2	—	none	TSFA	010 000 000 00000
output $\leftarrow$ input	input	—	none	TSFA	000 000 000 00000
$R_4 \leftarrow$ shl R_4	R_4	—	R_4	SHLA	100 000 100 11000
$R_5 \leftarrow 0$	R_5	R_5	R_5	XOR	101 101 101 01100

Stack Organization - 9.2

A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers. The stack pointer register sp contains a binary number whose value is equal to the address of the word that is currently on the top of the stack. Three items are placed in the stack A, B and C. Item C is on the top, so that the contents of sp is now 3. To remove the top item, the stack is poped by reading the memory word at address 3 and decrementing the content of SP. Now item B is on the top



In 64 word stack the stack pointer contains 6 bits because $2^6 = 64$. Since sp has only 6 bits, It can not exceed a number greater than 63, when 63 is incremented by 1 the result is 0. When 0000000 is decremented by 1, the result is 111111. The one bit register full is set to 1.

Ex:- A * B + C * D

is written in reverse polish notation as AB * CD * +

Scan the expression from left to right. When an operator is reached, perform the operation with the two operands found on the left side of the operator.

(A*B) CD * +

(A * B)(C * D) +

(A * B) + (C * D)

Reverse polish notation

Ex:- (A + B) * (C * (D+E) + F)

To evaluate the expression we must first perform the arithmetic inside the parentheses (A+B) and (D+E). Next we must calculate the expression inside the square brackets.

AB + DE + C * F + *.

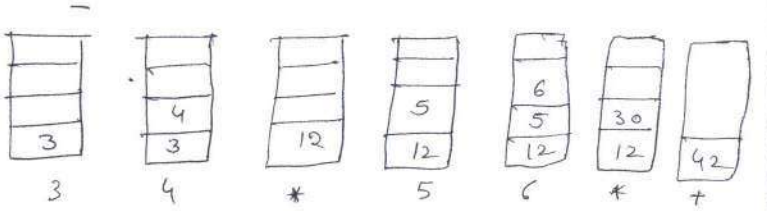
Proceeding from left to right, we first add A and B, then add D and E.

(A+B)(D+E) C * F + *

Evaluation of Arithmetic expressions

Ex:- (3 * 4) + (5 * 6)

In reverse polish notation, it is expressed as 34 * 56 * +. Now consider the stack operations.



First the number 3 is pushed into the stack, then the number 4. The next symbol is multiplication operator *. This causes a multiplication of the two topmost items in the stack. The stack is then poped and the product is placed on the top of the stack. Next we encounter the operands 5 and 6, then * replaces the numbers by product. The last operation causes addition of two operands and the final result is 42.

DR is the data Register that holds the binary data to be written into or read out of the stack. The push operation is implemented with the following Microoperations

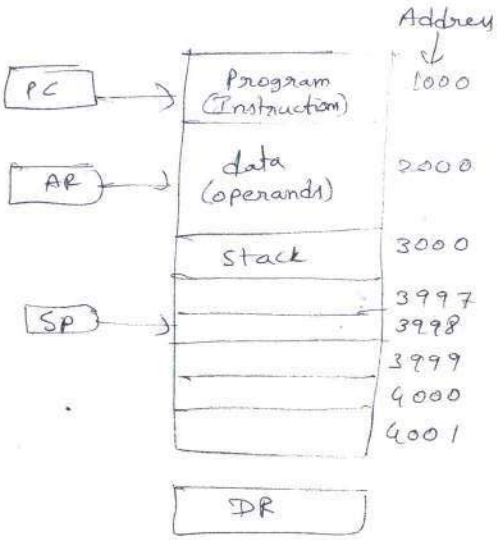
$SP \leftarrow SP+1, M[SP] \leftarrow DR$ (write item on top of stack)
If $(SP=0)$ then $full \leftarrow 1$ (check if stack full)
$empty \leftarrow 0$ (Mark the stack not empty)

A new item is deleted from the stack if stack is not empty. The pop operation consists of the following Microoperations

$DR \leftarrow M[SP]$ (Read item from top of stack)
$SP \leftarrow SP-1$
If $(SP=0)$ then $empty \leftarrow 1$ (check if stack is empty)
$full \leftarrow 0$ (Mark the stack not full)

Memory Stack

The following figure shows a portion of computer memory partitioned into 3 segments. Program, data, stack. The program counter PC points at the address of the next instruction in the program. The address register AR points at an array of data. The SP points the top of stack.



The three registers are connected to a common address bus, and either one can provide an address for memory. PC is used during the fetch phase to read an instruction. AR is used during the execute phase to read the operand. SP is used to push and pop into or from the stack.

Reverse polish Notation

The polish mathematician Lukasiewicz showed that arithmetic expressions can be represented in prefix notation. This representation, often referred to as polish notation.

A+B	Infix notation
+AB	prefix or polish notation
AB+	postfix or reverse polish notation

Instruction formats

The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in control register. The most common fields found in instruction format are

* An operation code field that specifies the operation to be performed
* An address field that designates a memory address or a processor register
* A mode field that specifies the way the operand or the effective address is determined

Computers fall into 3 types of CPU organizations.

Single accumulator organization

Ex :- ADD X

Where X is the address of operand.

The add instruction result the operation $AC \leftarrow AC + M[X]$ AC is the accumulator register and $M[X]$ is the memory word located at address X.

Ex :- Add R_1, R_2, R_3

Adds the contents of registers R_1, R_2, R_3 and stores the result in R_1

Ex :- Add R_1, R_2

Adds the contents of R_1, R_2 and stores the result in R_1

Ex :- MOV R_1, R_2

Moves the contents of R_2 to R_1.

General Register organization

Ex :- Add R_1, X

Would specify the operation $R_1 \leftarrow R_1 + M[X]$

Stack organization

Ex :- Push X

Will push the word at address X to the top of the stack. The stack pointer is updated automatically.

Ex :- ADD.

This operation adding the numbers, and pushing the sum into the stack.

Three address instruction

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

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

The advantage of three address instruction is it results in short programs. The disadvantage is it requires too many bits to specify three addresses.

Two-address instructions

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

The Move instruction moves or transfers the operands to and from memory and processor registers.

One-address Instructions

LOAD	A	$AC \leftarrow M[A]$
ADD	B	$AC \leftarrow AC + M[B]$
STORE	T	$M[T] \leftarrow AC$
LOAD	C	$AC \leftarrow M[C]$
ADD	D	$AC \leftarrow AC + M[D]$
MUL	T	$AC \leftarrow AC * M[T]$
STORE	X	$M[X] \leftarrow AC$

All operations are done between Accumulator and a memory operand. T is the address of a temporary memory location.

Zero-address Instructions

push A	$TOS \leftarrow A$
push B	$TOS \leftarrow B$
add	$TOS \leftarrow (A+B)$
push C	$TOS \leftarrow C$
push D	$TOS \leftarrow D$
add	$TOS \leftarrow (C+D)$
MUL	$TOS \leftarrow (C+D) * (A+B)$
POP X	$M[X] \leftarrow TOS$

The name zero address is given to this type of computer because of the absence of an address field in the computational instruction.

RISC instructions

A program for a RISC type CPU consists of LOAD and store instructions that have one memory and one register address.

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

Load	R_1, A	$R_1 \leftarrow M[A]$
Load	R_2, B	$R_2 \leftarrow M[B]$
Load	R_3, C	$R_3 \leftarrow M[C]$
Load	R_4, D	$R_4 \leftarrow M[D]$
Add	R_1, R_1, R_2	$R_1 \leftarrow R_1 + R_2$
Add	R_3, R_3, R_4	$R_3 \leftarrow R_3 + R_4$
MUL	R_1, R_1, R_3	$R_1 \leftarrow R_1 * R_3$
STORE	X, R_1	$M[X] \leftarrow R_1$

} 4-3

Numeric data can often be represented as integers. In unsigned integers, an n bit value can range from 0 to $2^n - 1$. An n bit signed integer can have any value between -2^{n-1} and $2^{n-1} - 1$

Some numeric data can not be represented as integers. These values, which typically include fractional portions, are represented in floating point format. The boolean values True and false are used often enough to warrant having their own data type, boolean, and assembly language instructions.

Computers must also deal with character data. The characters are stored as binary values encoded using ASCII, EBCDIC, UNICODE - - - .

Addressing Modes – 4Q

When a microprocessor access memory, to either read or write data, it must specify the memory address it needs to access. An assembly language instruction may use one of several addressing modes to generate the address

Direct Mode : The instruction includes a memory address. The CPU accesses that location in memory for example the instruction LDAC 5 reads the data from memory location 5 and stores the data in the CPU's accumulator

Indirect mode :-

In indirect mode the address specified in the instruction is not the address of the operand It is the address of a memory location that contains the address of the operand. For example LDAC @5 or LDAC (5), first retrieves the contents of location 5, say 10. Then CPU goes to location 10, reads the contents of that location.

Register direct and Register Indirect Mode

Register modes work the same as direct and indirect modes, except they do not specify a memory address. Instead, they specify a register. If register R contains the value 5, the LDAC R instruction copies the value 5 from register R into CPU's accumulator. The indirect instruction, LDAC (R) or LDAC@R, it reads the address from the register instead of the first memory access.

Immediate Mode

Here the operand specified is not an address. It is the actual data to be used. The instruction LDAC #5 moves the data value 5 into the accumulator.

Implicit Mode

Implicit mode does not explicitly specify an operand, This is not usually used for load instructions, clears the accumulator (CLAC)

Relative Mode

In this mode, the operand supplied in an offset, not the actual address. It is added to the contents of the CPU's program counter register to generate the required address. For example LDAC $5 is located at memory location 10, and it takes up two memory locations, the next instruction is at location 12.

Index Mode and Base Address Mode

Index mode works like relative mode, except the address supplied by the instruction is added to the contents of an index register instead of the program counter. If the index register contains the value 10, the instruction LDAC 5(X) reads data from location (5+10=)15 and stores it in the accumulator.

a) 0: LDAC 5

5 : 10 ⟶ stores value in CPU

b) 0: LDAC @5

5 : 10

10 : 20 ⟶ stores value in CPU

c) 0: LDAC R

R : 5 ⟶ stores value in CPU

d) 0: LDAC R

R : 5

5 : 10 ⟶ stores value in CPU

e) 0: LDAC #5 ⟶ stores value from instruction in CPU

f) 0: LDAC (Implicit)

Stack: ⟶ stores value in CPU

g) 0: LDAC $5

1

5

6 : 12 ⟶ stores value in CPU

h) 0: LDAC 5(X)

X : 10

15 : 30 ⟶ stores value in CPU

Instruction Formats

When an assembly language instruction is converted to its equivalent machine code, it is represented as a binary value called the instruction code. This value is in a specific format. Different group of bits represent different parts of the instruction one group may represent the operation to be performed (the opcode), while other groups select the operands of the operation.

Ex: A = B + C

This instruction has one operation, addition two operands B and C.

Ch-7 Input – Output Organization Chapter-7

1Q Peripheral Devices

Input or output devices attached to the computer are called peripherals. The most common peripherals are keyboard, display units and printers. Peripherals that provide auxiliary storage for the system are magnetic disks and tapes.

Monitor and keyboard

Video monitors are the most commonly used peripherals. They consist of a keyboard as the input device and a display unit as the output device. There are different types of video monitor. The most popular use a cathode beam to a screen in front of the tube. The CRT receives a voltage that causes the beam to hit the screen. A characteristic feature of display device is a cursor that marks the position in the screen to a single character, the beginning of a word, or to any line.

Printers

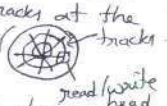
Printers provide a permanent record on paper of computer output data or text. There are 3 types of character printers. daisywheel, dot matrix and laser printer. The daisywheel printer contains a wheel with the characters placed along circumference. The dot matrix printer contains a set of dots along the printing mechanism. The laser printer uses a rotating photographic drum that is used to imprint the character images.

Magnetic tape

Magnetic tapes are used mostly for storing files of data: For example A company's payroll record. Access is sequential and consists of records that can be accessed one after another as the tape moves along a stationary read-write mechanism. It is one of the cheapest and slowest methods for storage.

Magnetic disk

Magnetic disks have high speed rotational surfaces coated with magnetic material. Access is achieved by moving a read-write mechanism to a track in the magnetized surface. Disks are used mostly for bulk storage of programs and data. Disks may have multiple heads and simultaneous transfer of bits from several tracks at the same time



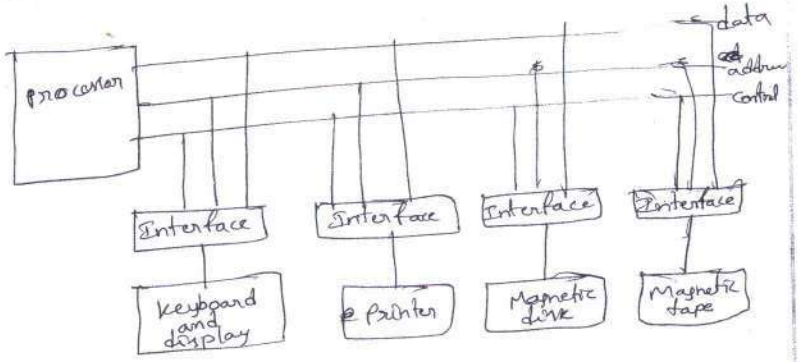
Q-1

2Q Input – Output Interface

1. The peripherals are electromechanical devices and electromagnetic devices and their manner of operation is different from the operation of CPU, and memory, which are electronic devices. So signal conversion may be required
2. The data transfer rate of peripherals is usually slower than the transfer rate of the CPU, a synchronization may be needed

* Data codes and formats in pto peripherals differ from the word format in the cpu and memory
* The operating modes of peripheral are different from each other.

The components between the cpu and peripherals to supervise and synchronize all input and output transfers. These components are ~~calle~~ called interface units.



The I/o bus from the processor is attached to all peripheral interfaces. To communicate with a particular device, the processor places a device address on the address lines. Each interface attached to the I/o bus contains an address decoder that monitor the address lines. When the interface detects its own address, it activates the path between the bus lines and device that it controls.

A control command is issued to activate the peripheral and to inform it what to do. A status command is used to test various status conditions. During the transfer, one or more errors may occur which are detected by the interface. A data output command causes the interface to respond by transferring data from the bus into one of its registers. The data ~~input~~ output command is the opposite of the data input. Here the interface receives an item of data from the peripheral and places it in its buffer register.

There are 3 ways that computer buses can be used to communicate with memory and I/o

* Use two seperate buses, one for memory and the other for I/o
* Use one common bus for both memory and I/o but have seperate control lines for each
* Use one common bus for memory and I/o with common control lines.

Many computer use one common bus to transfer information between memory or I/o and the cpu. The distinction between a memory transfer and I/o transfer is made through seperate read and write lines. The cpu specifies whether the address on the address lines is for a memory word or for an

Interface register by enabling one of two possible read or write lines. The I/o read and I/o write control lines are enabled during an I/o transfer. The memory read and write control lines are enabled during a memory transfer.

In the Isolated I/o configuration, CPU has distinct input and output instructions. Each of these instructions is associated with the address of an interface register. When the CPU fetches and decodes the operation code, it places the address into the address lines. It enables the I/o read or I/o write. This inform the external components that are attached to common but that the address lines is for an interface register and not for a memory word. On the other hand, when CPU is fetching an instruction it places the ~~ad~~ memory address on the address lines and enables the memory read or memory write. This inform external components that the address is for a memory and not for I/o

In the memory mapped I/o the same address space for both memory and I/o. ~~So one~~ This is the case in computers that employ only one set of read and write signals do not distinguish between memory and I/o.

Here there are no specific input or output instructions. It allows the computer to use the same instructions for either I/o or memory transfer. The advantage is that the load and store instructions used for reading and writing from memory can be used to input and output data from I/o registers.

9-2

Integrated Circuits

Digital circuits are ~~connected~~ constructed with integrated circuits. An integrated circuit (IC) is a small silicon semiconductor crystal, called a chip, containing the electronic components for the digital gates. The various gates are interconnected inside the chip to form the required circuit. The number of pins may range from 14 in a small IC package to 100 or more in a larger package. The package is being either a small, medium, large scale integration device.

Small scale integration (SSI) devices contain several independent gates in a small package. The inputs and outputs of the gates are connected directly to the pins in a package.

Medium-scale integration (MSI) devices have a complexity of approximately 10 to 200 gates in a single package. They usually perform specific elementary digital functions such as decoders, adders, and registers.

Large Scale Integration (LSI) devices contain between 200 and a few thousand gates in a single package. They include digital systems, such as processors, memory chips, and programmable modules.

Very large scale integration (VLSI) devices contains thousands of gates within a single package. Examples are large memory arrays and complex microcomputer chips. Because of their small size and low cost.

Many different logic families of Integrated circuits have been introduced commercially. The following are the most popular.

TTL	Transistor transistor logic
ECL	Emitter-coupled logic
MOS	Metal oxide semiconductor
CMOS	Complementary metal oxide semiconductor

TTL is a widespread logic family that has been in operation for many years. This technology used in diodes and transistor for the nand gate. The technology was called DTL

ECL family provides the highest speed digital circuits in integrated ~~form~~ form-ECL is used in systems such as supercomputers.

The MOS is a unipolar transistor that depends on the flow of only one type of carrier. Bipolar transistor used in TTL and ECL gates.

The CMOS technology ~~is used~~ in the high package density of circuits and lower power consumption.

Integrated circuits are used exclusively to provide various digital components needed in the design of computer systems.

Ch-7 Input / Output Organization

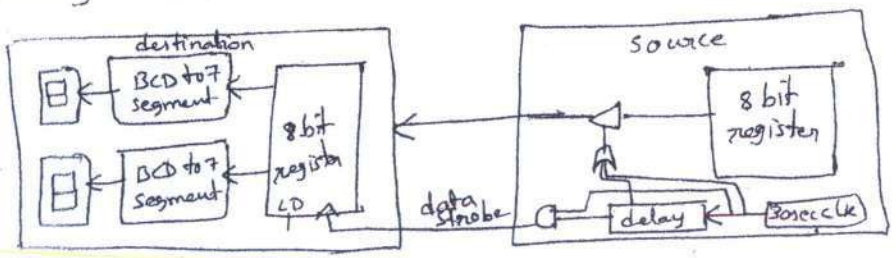
Asynchronous Data Transfers - 3Q

Asynchronous transfers use control signals and their associated hardware to coordinate the movement of data. These data transfers do not require that the source and destination use the same system clock. There are 4 types of asychronous data transfers.

* Source initiated data transfer.
* Destination initiated data transfer
* Source initiated data transfer with handshaking
* Destination initiated data transfer with handshaking.

Source initiated data transfer

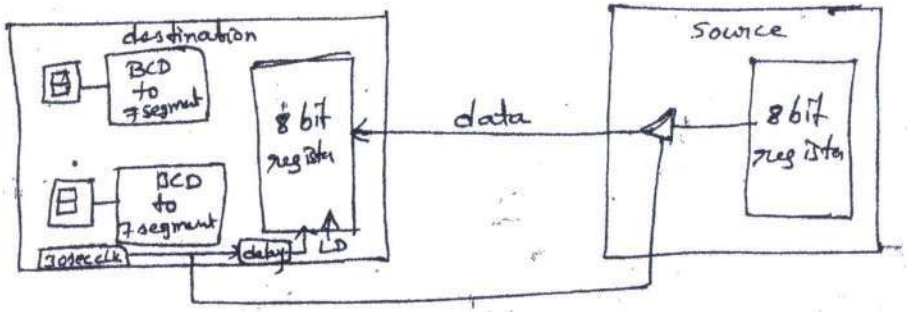
In this mode, the source outputs its data, then strobes a control signal for a set of amount time. The destination device reads in the data during this time. The source device deasserts the strobe and stops outputting the data.



value to the output device. A clock with a period of 3 seconds enables the tri-state buffers, which causes valid data to be made available to the output module.

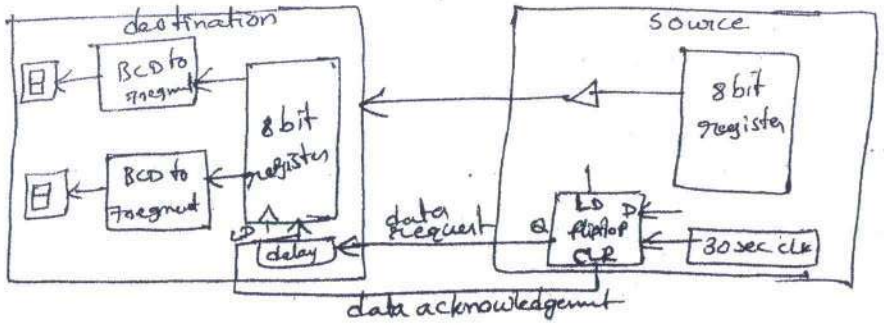
Destination-Initiated Data Transfer

Here the destination device transmits a data strobe signal to the source device. After a set delay to ensure that valid data is ready, the destination device reads in this data and deasserts the data strobe. This in turn causes the source to stop transmitting valid data.



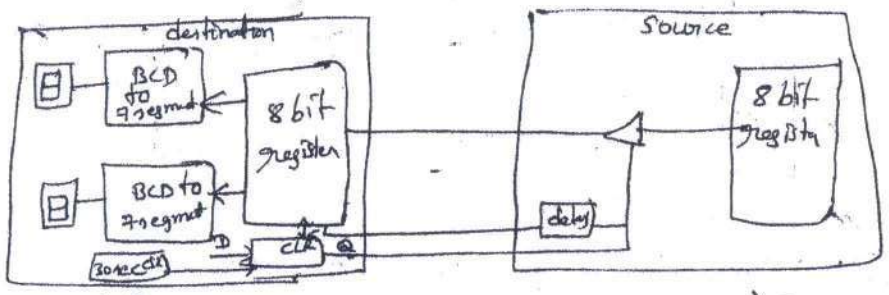
Source initiated transfer with handshaking

The source sets the data request signal high and then makes valid data available to the destination device. After a delay, the destination reads the data. Once the destination device has read the data it sends a data acknowledge signal to the source. This tells the source that the destination has read it and no longer needs this data



Destination initiated data transfer with handshaking

It is similar to that of source initiated data transfer using handshaking, except that the data acknowledge signal is replaced by data ready signal.



Modes of I/O

4Q

Programmed I/O - 4Q

Consider the design of relatively simple CPU. The CPU reads in the current temparature from the memoryе mapped input port at address FFFFH and the thermostat setting from the port address FFFE.H. The CPU controls the heating and air conditioning systems. Writing one of the following 4 values to the output port at address FFFD.

01 = turn on air conditioning
02 = turn off air conditioning
03 = turn on heat
04 = turn off heat.

The current status is stored in memory location 1000H. It has the following values

FF = heat on
FE = air conditioning is on

Algorithm

Start : Read current status
Heaton : If heat is on then
read current setting and temparature
If setting = temparature then
turn off heat
update status
goto start
Aironn : If air conditioning is on then
read current setting and temparature
If setting = temparature then
turn off airconditioning
update status
goto start.

start : LDAC 1000
INAC (set z=1 if status = FF)
JMPZ heaton
INAC (set z=1 if status = FE)
JMPZ aironn

```
Wheaton : LDAC FFFF
          MOVR
          LDAC FFFE
          SUB
          JPNZ start (If setting ≠ temparature)
          INAC
          INAC
          INAC
          INAC
          STAC FFFD
          CLAC
          STAC 1000
          JUMP start

Akron :   LDAC FFFF
          MOVR
          LDAC FFFE
          SUB
          JPNZ start (If setting ≠ temparature)
          INAC
          INAC
          STAC FFFD
          CLAC
          STAC 1000
          JUMP start
```

* New instructions

For this design, we add new two instructions in the instruction set.

INPT	AC ← Inputport
OUPT	Outputport ← AC

* New control signals

we can set IO=1 for Input/output operations and IO=0 for memory reference operations.

* New states

INPT 1 : DR ← M, PC ← PC+1, AR ← AR+1
INPT 2 : TR ← DR, DR ← M, PC ← PC+1
INPT 3 : AR ← DR, TR
INPT 4 : DR ← Input port
INPT 5 : AC ← DR

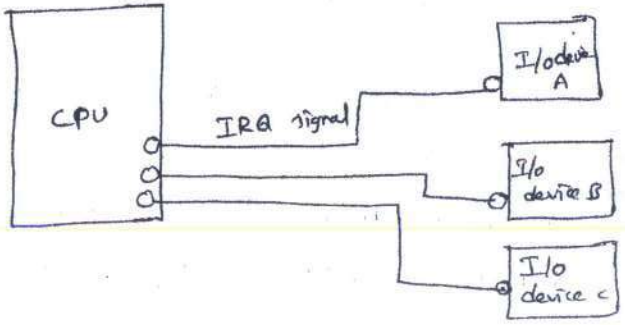
* Modify the cpu hardware for new instructions

1. Modify the register section
2. Modify the ALU
3. Modify the CU

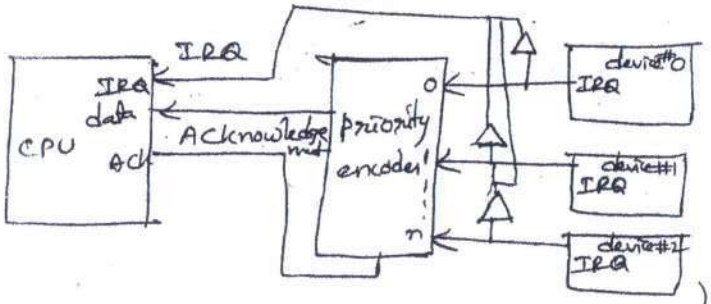
* Make sure the other instruction still work

Interrupts - 5Q

Interrupt is a signal from the hardware. Interrupt is a signal that tells the computer to stop what it is doing and to execute some other piece of code. When the I/O device is ready to receive the data, it sends an interrupt request signal to the CPU. The CPU then acknowledges the interrupt, by asserting interrupt acknowledge signal and completes the data transfer.



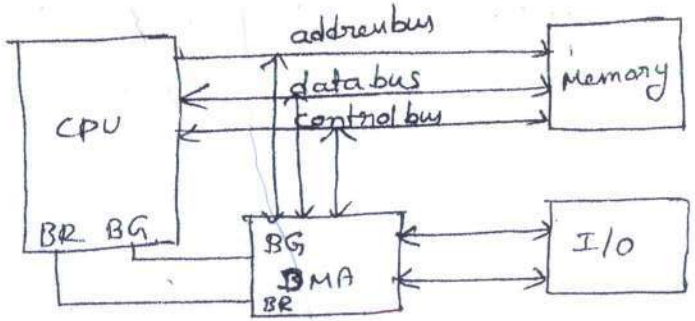
The output of the encoder is the value of the highest priority device requesting an interrupt. This value is placed on the data bus as the interrupt vector and is read by the CPU



)5Q.

Direct Memory Access (DMA) 6Q - 6Q

DMA is circuit which transfers data from I/o device to memory or memory to I/o devices using address bus, data bus, control bus.

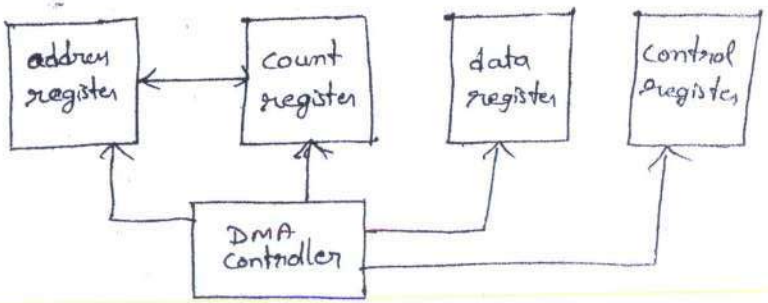


When the I/o device wants to transmits the data to memory it sends ~~an~~ a DMA request to DMA. DMA requests CPU for bus that means it sends bus request signal (BR = 1) to CPU. If the CPU is ready to give up the bus it send BG signal (bus grant BG = 1) to DMA. and tristate address, data, control buses. and DMA sends acknowledge to the I/o device, and receives the data from I/o and Stores internal registers.

The DMA places the address of the memo on to the address bus and places the data on to the data bus and activates the write signal in the contro bus completes the transmission. Then DMA sets BR = 0 and sends the signal to CPU and CPU sen sets BG = 0 and sends the acknowledgement to the DMA to stop the transmission.

DMA has four internal registers

1. address register
2. count register
3. Data register
4. control register



First the CPU places the memory address in the address register. For example DMA must move 4 bytes of data from I/o to memory starting at location

The interrupt request output pins are in the I/O device are connected to interrupt request input pins in the CPU. There are different types of interrupts.

1. External Interrupts
2. Internal Interrupts
3. Software Interrupts.

External Interrupts

External Interrupts are used by the CPU to interact with Input/output devices. External Interrupts improve the system performance.

Internal Interrupts

Internal Interrupts occur entirely within the CPU. no input/output devices play any role in these interrupts. For example a timer built into the CPU may generate an interrupt at a predetermined interval. These are used to handle exceptions (divide by zero, overflow, page faults - - -)

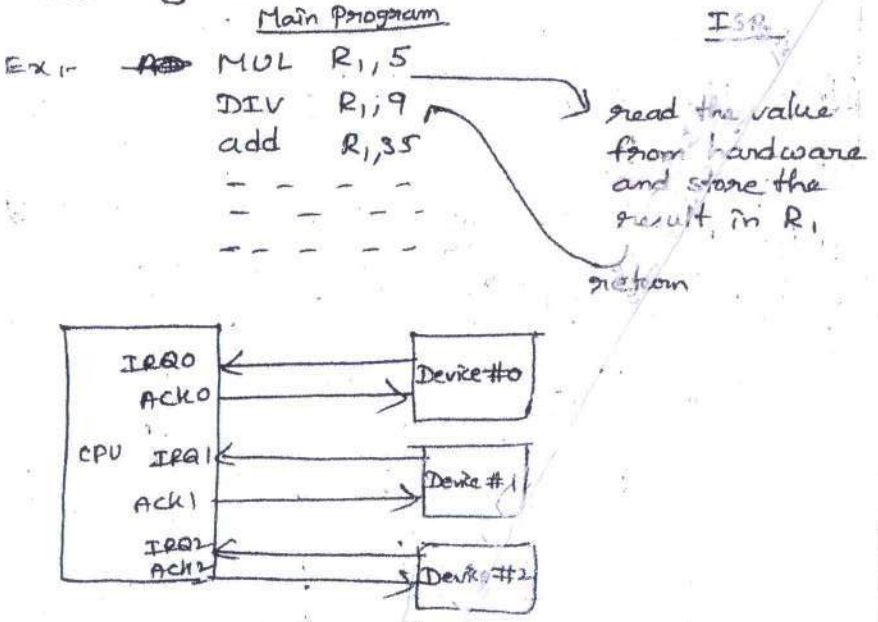
Software interrupts

Software interrupts are generated by specific interrupt instructions. They act like subroutine call statements except they always go to a specific address.

When the I/O requests data device sends a request to the CPU then the CPU stops executes the instruction and jumps to the Interrupt Service routine (ISR). It is also called interrupt handler. After completing the interrupt Service routine it returns back to the original program and continues the execution. Before going to ISR it stores all the values of the registers into a stack and Before returning to the main program all the register values can be deleted from the stack and stores them in the registers.

Push AC : SP ← SP+1, M[SP] ← AC POP AC : AC ← M[SP], SP ← SP-1

Pushing all the registers at the beginning of interrupt routine is called Saving context. Poping all the registers at the end of interrupt routine is called restoring context.



When more than one device request the CPU at the same time it will execute the highest priority interrupt signal request first, using some priority encoder.

The CPU writes 1000 to the ...

and 4 to the count register. Then it outputs a value to the DMA control register, which causes it to initiate a transfer from I/o to memory. The DMA reads the data and places the data in data register and places the address on the address bus, data on the data bus and activates write signal. The count register value is decremented. If the count register value is 0 then the transmission will be stopped.

DMA transfer modes

1. Burst mode
2. Cycle stealing mode
3. Transparent mode

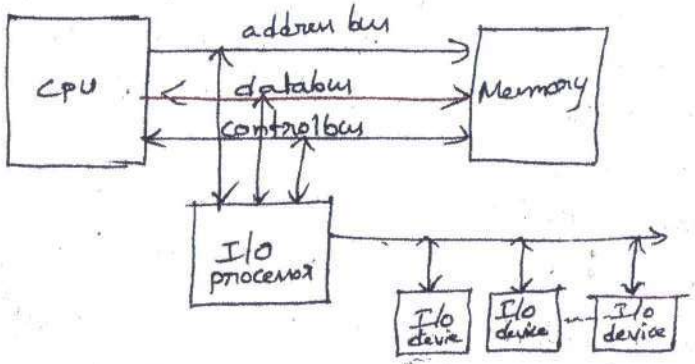
In burstmode the entire block of data is transferred from I/o to memory. In cycle stealy mode only one byte is transmitted and the DMA will give the bus to CPU. After some time it transfers the second byte by receiving the bus from CPU and then give the bus. But in transparent mode the DMA only transfers data when CPU is ~~not~~ performing operations that do not use the system buses.

Ex: AC ← DR
PC ← DR, TR

I/O processors — 14 — 79

I/O processor:-

I/o processors are sometimes called as I/o controllers, or pheripheral processing units (PPUs). The I/o processor is situated between the I/o devices and the rest of the system. The I/o processor connects to more than one I/o device. The I/o devices are grouped together on an I/o bus. I/o processors handle all of the interactions between the I/o devices and CPU.



The CPU instructs the I/o processors to perform transfer between an I/o device and memory Instead of loading values into registers, the CPU issues a series of I/o instructions to the I/o processor. The CPU stores these instructions in memory and sends a pointer to the instructions to the I/o processor. There are 3 ~~categ~~ categories.

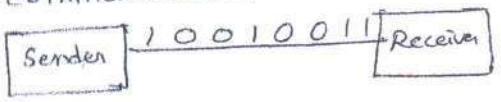
* block transfer commands
* arithmetic, logic and branch operations
* control commands.

A block transfer command might tell the I/o processor to read 32 bytes of data from I/o with port address 220H and write the data to memory starting at location 1000H. In the second type of arithmetic and logic transfer commands, for example CPU has 32 bit data bus, If the CPU reads the data from Input port an 8 bit, it would have to read four seperate values and concatnate them into a single 32 bit value. Instead CPU can instruct the I/O processor to read 4 values, using ~~A bb~~ ALU commands The third type of command is control commands. for example

* Move 247 bytes from diskdrive at port address 9000H to memory starting at address 1000H.
* read 1 byte of data from the input port at address 9001H into AC of the CPU
* Write the contents of memory locations 2000H through 207FH to the printer at I/o address 9002H)

Serial communication

In parallel transmission we can transmit more than one bit at a time. Some devices can not handle more than one bit of data. They utilize serial communication.

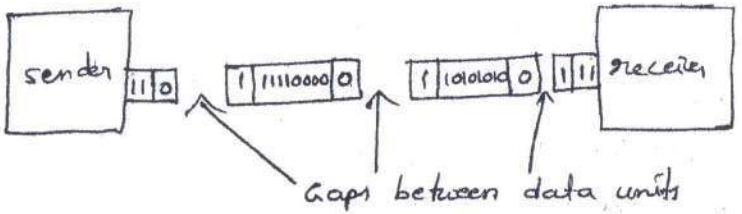


There are 2 types of Serial communication

* Asynchronous Serial communication
* Synchronous Serial communication

Asynchronous Serial communication

In asynchronous serial communication we send 1 start bit at beginning and 1 or more stop bits at the end of each byte. There may be a gap between each byte.

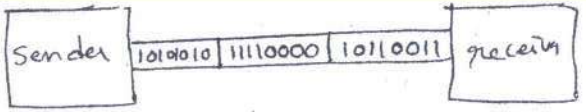


The addition of start and stop bits and the insertion of gaps into the bit stream make asynchronous transmission ~~slower than~~ slow. But it is cheap and effective. This is useful for low speed communication.

Ex:- keyboard to a computer

Synchronous Serial communication

In synchronous serial communication the bit stream, we send bits one after another without start/stop bits or gaps. It is the responsibility of the receiver to group the bits.



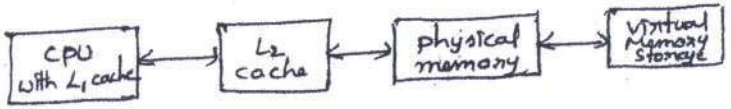
Ch-8

Hierarchical Memory Systems

(Cache Memory) – 1Q

Hierarchical memory system question – 4Q

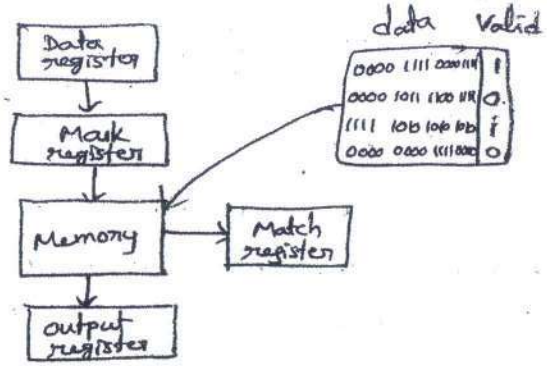
(A typical memory hierarchy is shown in the figure. The most well known element of the memory subsystem is the physical memory, which is constructed using dynamic random access memory (DRAM) chips.



Cache Memory is the intermediate between CPU and main memory. Cache memory itself can be arranged hierarchically, usually two levels. The first level, the level 1 cache, is incorporated directly into the Microprocessor. The level 2 cache, is usually outside of the Microprocessor.) Hierarchical memory system question – 4Q

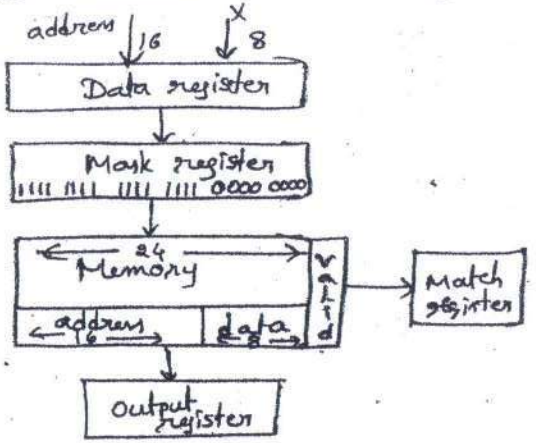
Associative memory. – 3Q

Cache memory can be constructed using either static RAM or associative memory (sometimes called content addressable memory). Static RAM is accessed just like most other types of memory: It receives an address and accesses the data at that address. But associative memory is accessed differently. To access data in associative memory, a portion of the data is specified. It searches all its locations and identifies the locations that match the specified data input.

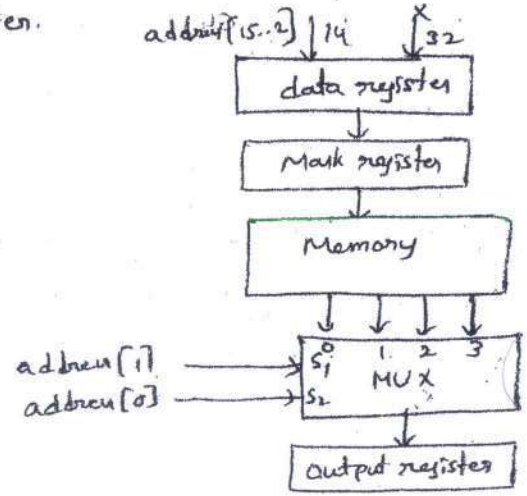


In addition to the data bits, each word has one additional bit labeled V. This is the valid bit. a 1 in this position indicates that the word contains valid data and 0 shows that the data is not valid. To read the value from associative memory, the CPU must specify the data value to be matched and the bits of this data value that are to be checked. For example CPU wishes to access data in the associative memory that has 1010 as its 4 high order bits. The CPU would load the value 1111 0000 0000 0000 into the mask register. The CPU also loads the value 1010 xxxx xxxx xxxx into the data register. The 4 leading bits are the value to be matched, and the remaining 12 bits can have any value. The associative memory contains a match register, which has one bit for each location in the associative memory. If a location generates a match, its bit in the match register is set to 1. otherwise it is set to 0.

Cache Memory with associative Mapping



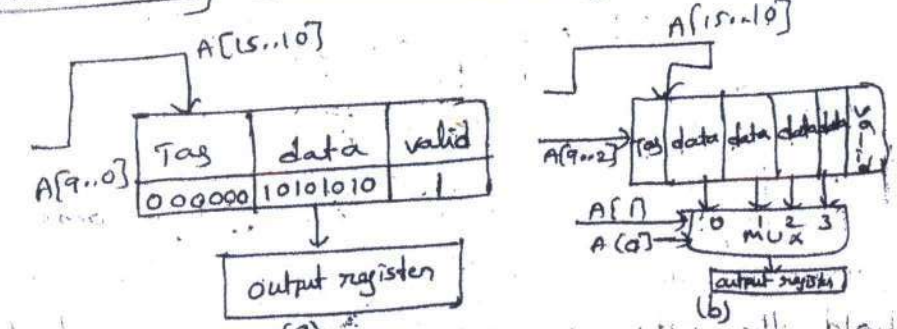
The first 16 bits of each location would consist of a memory address and the last 8 bits would hold the data stored at that address in physical memory. The processor would output the address it wishes to access to the data register of the associative memory and the value 1111 1111 1111 1111 0000 0000 to its mask register.



This memory stores 4 bytes of data

each location, each of which has the same address in physical memory, except for the two low order bits. The bottom location contains the data for physical memory address 0000H to 0003H. only upper 14 bits are used to generate a match.) 39

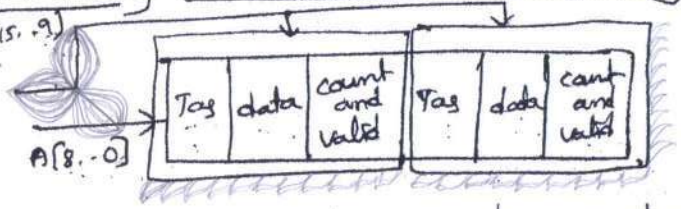
Cache Memory with direct Mapping



Cache Mapping scheme that can use standard SRAM can be much larger than an associative cache and still cost less. one such mapping is called direct mapping. The CPU accesses 64K, 8 bit bytes of data using 16 address bits. The 10 low order address bits, the index, select one specific location in the cache. The location contains 3 fields. A valid bit is used for checking valid data or not, a tag field contains the high-order bits of the original address. 6 high orders bits are stored in the tag field.

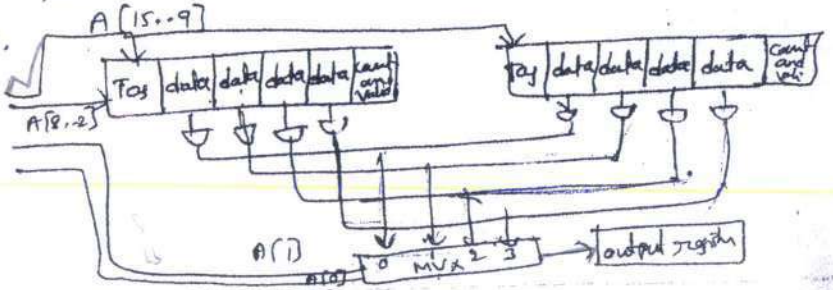
Consider location 0000 0011 1111 1111 of physical memory, which contains data 1010 1010. This data can be stored in only one location in cache, with address bits as the original address, or 11 1111 1111. However, any address of the form xxxx xx11 1111 1111 would map to this same cache location. The tag value for this location is 00 0000. This means that the data stored at location 11 1111 1111 is actually the data from physical memory location 0000 0011 1111 1111. In the second diagram the two low order bits are used to select one byte from among the four in the line.

Cache Memory with Set-associative Mapping



A set associative cache makes use of relatively low cost SRAM. A cache in which each location can contain n bytes or words of data is called n-way set associative cache.

Each location contains 2 groups of fields, one for each way of cache. The tag field is same as in direct mapping. Nine address bits select the cache location and the remaining 7 specify the tag value. You can use multiplexer to select particular data.



Replacing data in the cache

There are a number of replacement policies to select location to be used.

FIFO: This method fills the memory from its top to its bottom location. When it copies data to its last location, the cach is full. Then it goes back to top and replaces the data.

LRU: This method keeps track of the relative order in which each location is accessed and replaces the least recently used

Random: Random method randomly selects a location to replace the data.

Writing data to the cache

Two ways are used to write data to the cache. 1. write through 2. write back. In write through, every time a value is written from the CPU into a location in the cache, it is also written to the corresponding location in physical memory. In write-back the value is written to physical memory only once, when the data is removed from the cache.

Cache performance

cache hits: Every time the CPU accesses memory, it checks the cache. If the request data is in the cache then it is called as cache hit.

cache miss: If the requested data is not in cache then it is called as cache miss.

hit ratio: The hit ratio is the percentage of memory accesses that are saved from the cache. The average memory access time $T_M = h T_C + (1-h) T_p$

T_p = access time for physical memory

T_C = access time for cache memory

h = hit ratio.

hitratio	T_M
0.0	60ns
0.1	55ns
0.2	50ns
0.3	45ns
0.4	40ns
0.5	35ns
0.6	30ns
0.7	25ns
0.8	20ns
0.9	15ns
1.0	10ns

T_C is much less than T_p, increasing the hit ratio reduces the average memory access time.)19.

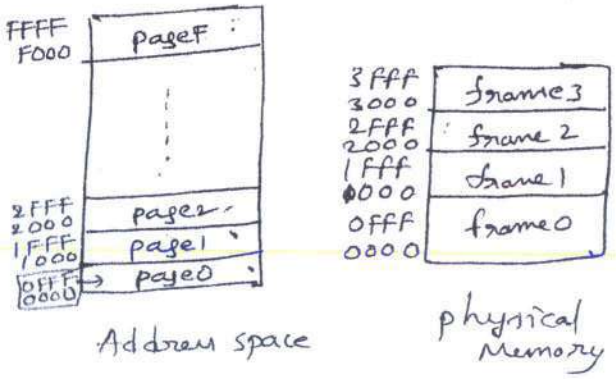
Virtual Memory GQ — 2Q

Virtual Memory uses auxiliary storage, such as disk, to expand the memory space available to the processor. It is much less costly than adding physical memory to a system. A memory management unit (MMU) moves data between physical memory and some slower storage device, usually disk. It is necessary to translate the CPU logical address to its corresponding physical memory address.

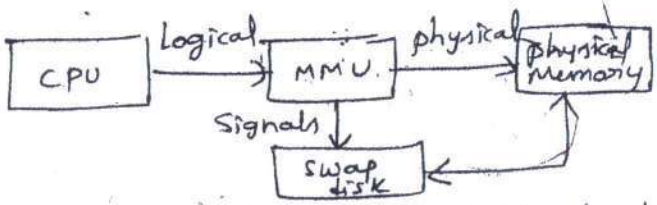
Two primary methods are used for implementing virtual memory: paging and segmentation.

Paging:

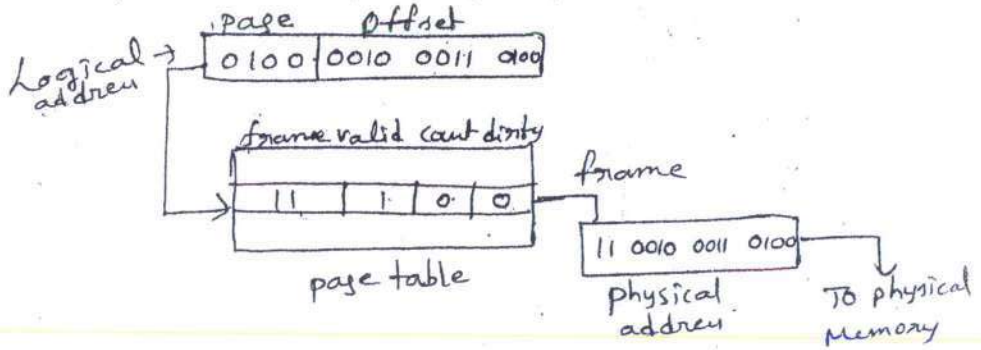
In paging, the entire range of logical addresses, those addresses that can be divided into continuous blocks called pages. Each page can store only one logical address. The physical memory is divided into frames.



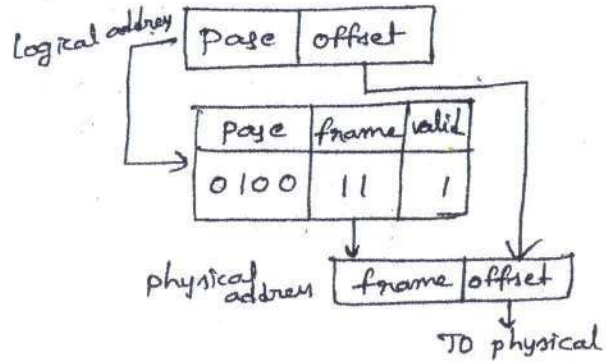
Paging moves pages from the swap disk to frames of the physical memory. So data can be accessed by the processor. Any page can occupy any frame. MMU converts logical address to physical address. For example LDAC 4234H means, which is actually stored in physical memory location 3234H.



MMU uses a page table to keep track track of which pages are stored in which frames. The page table contain 16 locations, one for each page in logical address space. Each location contains a 2 bit field, which denotes the frame in which that page is stored. each location contains a valid bit and a dirty bit. when a page is loaded into a frame dirty bit is set to 0. A memory write operation causes dirty bit to 1.

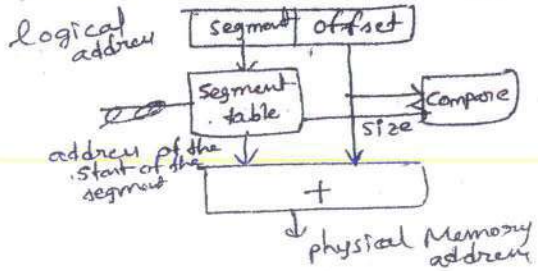


But there is a time delay associated with sending the frame number from the page table. MMU usually contain a translation lookaside buffer, or TLB. TLB does not necessarily contain count nor dirty bits.



Segmentation

Segmentation is another method of allocating memory that can be used instead of paging. Segments can vary in size. The logical address is partitioned into a segment number and an offset. The segment number is input to the segment table, outputs the address of the start of the segment. The offset is compared to segment size, If the offset is greater than or equal to the segment size, an error is generated.



It is possible to combine both segmentation and paging by constructing a segment from pages. A logical address is broken into 3 parts. The segment no, the page no, and the offset.