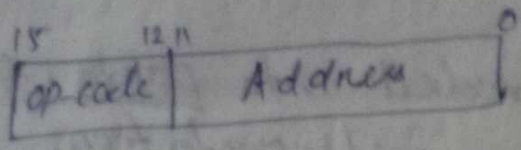
Instruction code: An instruction code is a group of bits that instruct the computer to perform a specific operation.

* It is usually divided into parts, each having its own particular interpretation.
* One part of an instruction code specify the operation code and other part specify the registers or the memory words (Address of operand) where the operands are to be found, as well as the register or memory word where the result is to be ~~found~~ Stored.

* The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift and complement.
* The number of bits required for the operation code of an instruction depends on the total number of operation available in the computer.
* There are many variations for arranging the binary code of instructions and each computer has its own particular instruction code format.
* The simplest way to organize a computer is to have one Processor register and an instruction code format with two parts.

* The first part specifies the operation to be performed and the second specifies an address.



A Simple Instruction format

INSTRUCTION FORMATS:

Computers may have instructions of several different lengths containing varying number of addresses. The no of address fields in the instruction format of a computer depends on the internal organization of its registers. Most computers fall into one of three types of CPU organizations.

1. Single accumulator organization
2. General register organization
3. Stack Organization.

To illustrate the influence of the number of addresses on computer programs, we will evaluate the arithmetic statement

$X = (A+B) * (C+D)$ using three, two, one, zero address instructions.

(1) Three-Address Instructions: Computers with three-Address instruction formats can use

each address field to specify either a Processor register or a memory operand.

* The assembly language program for $X = (A+B) * (C+D)$ is

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	$M[X] \leftarrow R_1 * R_2$

* It is assumed that the computer has two Processor registers R_1 and R_2 and M[A] denote the content of memory location A.

* Advantage: Short programs
* Disadvantage: Requires too many bits to specify three addresses

Ex. cyber 170.

(ii) Two-Address Instruction:

Here each address field can specify either a Processor register or a memory word.

The program for $X = (A+B) * (C+D)$

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	$M[X] \leftarrow R_1$

(iii) One-Address Instructions :

* One address instructions use an implied accumulator (AC) register for all data manipulation.
* Accumulator contains the result of all operations. All operations are done between the AC register and a memory operand. T is the address of a temporary memory location required for storing the intermediate resu

Program

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

(iv) Zero-Address Instructions :

* A stack-organized computer doesn't use an address field for the instruction such as ADD and MUL etc.
* The PUSH and POP instructions need an address field to specify the operand that communicates with the stack.

Program

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 catagories.

1. Arithmetic, Logical and Shift instructions
2. Instructions for moving information to and from memory and Processor registers.
3. Program Control instructions.
4. Input and Output instructions.

Most computer instructions can be classified into ~~three~~ three catagories.

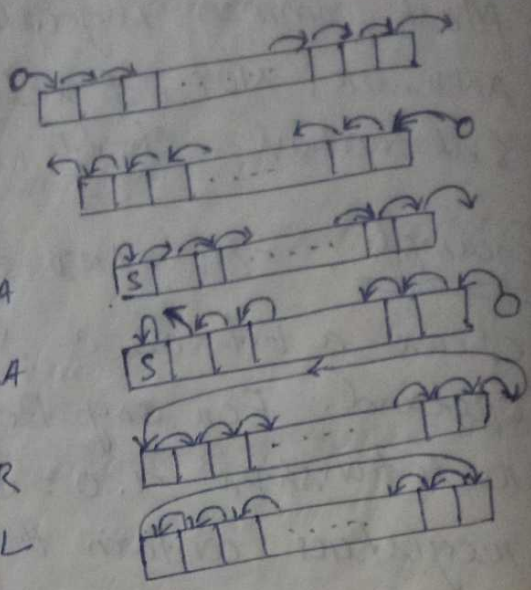
1. Data transfer instructions
2. Data manipulation instructions
3. Program Control instructions.

1. Data Transfer Instructions

* Data transfer instructions move data from one place in the computer to another without changing the data content.
* The most common transfers are between memory and Processor registers, between processor registers and input or output and between the processor registers themselves.
* Data transfer instruction must specify
  (i) the location of the source and destination operand

The shift instructions are

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic " right	SHRA
Arithmetic " left	SHLA
Rotate right	ROR
Rotate left	ROL



* The logical shift inserts 0 to the end bit position. The position is the leftmost bit for shift right and the rightmost bit position for the shift left.
* The arithmetic shift-right instruction must preserve the sign bit in the leftmost position. The sign bit is shifted to the right together with the rest of the number but the sign bit itself remains unchanged. The arithmetic shift-left instruction inserts 0 to the position and is identical to the logical shift-left instruction.
* The rotate instructions produce a circular shift. Bits shifted out at one end of the word are not lost as in a logical shift but are circulated back into the other end.

Instruction set completeness (continued)

Program control instructions: A Program control type of instruction, when executed, may change the address value in the Program counter and cause the flow of control to be altered. Program control instructions specify conditions for altering the contents of the Program counter, when this instruction get executed it changes the value of the Program counter as a result this causes a break in the sequence of instruction execution.

System control: These instructions are reserved for the use of operating system.

Ex:- A system control instruction may read or alter a control register.

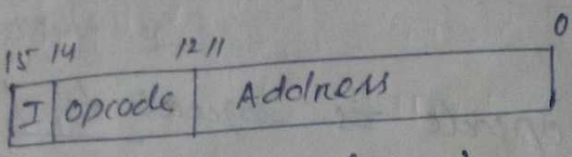
Transfer of control: We usually assume that, the next instruction to be performed is the one that immediately follows, in memory, the current instruction.

* However, a significant number of instruction in any Program change the sequence of instruction execution.
* For these instruction, the operation performed by the CPU is to update the Program counter to contain the address of some instruction in memory.
* There are a number of reasons why transfer of control operations are required. So they are (i) Looping (ii) Decision making (iii) Subroutine (function

Instruction Types

The basic computer instruction format has 16-bits, and is divided into three different fields.

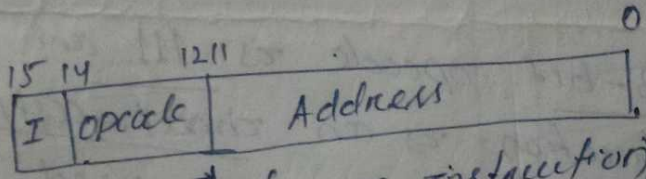
(i) Addressing mode (bit no-15)
(ii) opcode (bit no-12 to 14)
(iii) Address / Register type operation / I/o type operation (bit no:- 0 to 11)



(Instruction format)

The type of instruction is recognized by the computer control from the four bits in positions 12 through 15 of the instruction.

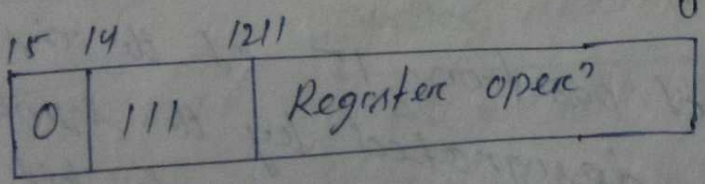
(i)



(Memory-Reference instruction)

[op-code:- 000 through 110 and I is 0 or 1
I is '0' for direct Addressing
and I is 1 for indirect Addressing]

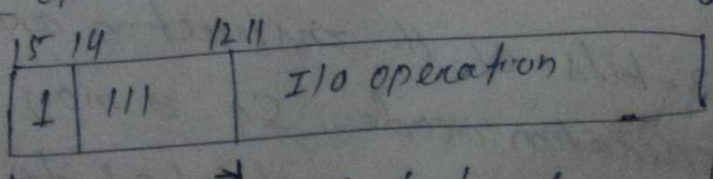
(ii)



(Register - Reference Instruction)

[op-code:- 111 , I = 0]

(iii)



Input - output Instruction

[opcode:- 111
I = 1]

(i) If the three opcode bits in positions 12 through 14 are not equal to 111, the instruction is a memory-Reference type and bit position 15 is taken as the addressing mode I. I is equal to 0 for direct address and to 1 for indirect addressing mode.

other 12-bits (0-11) to specify an address

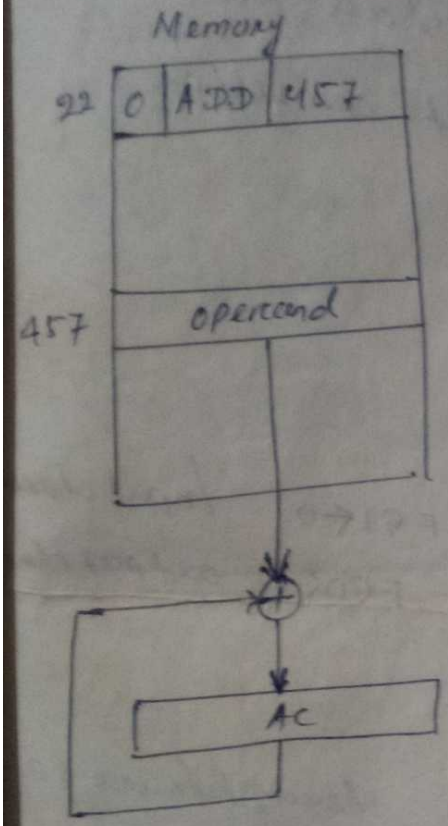
(ii) If the 3-bit opcode is equal to 111 and I bit is 0, the instruction is a Register-Reference type and other 12-bits are used to specify the operation or test to be executed on AC register.

(iii) If the 3-bit opcode is 111 and I bit is 1, the instruction is an input-output type and remaining 12-bits are used to specify the type of input-output operation or test performed.

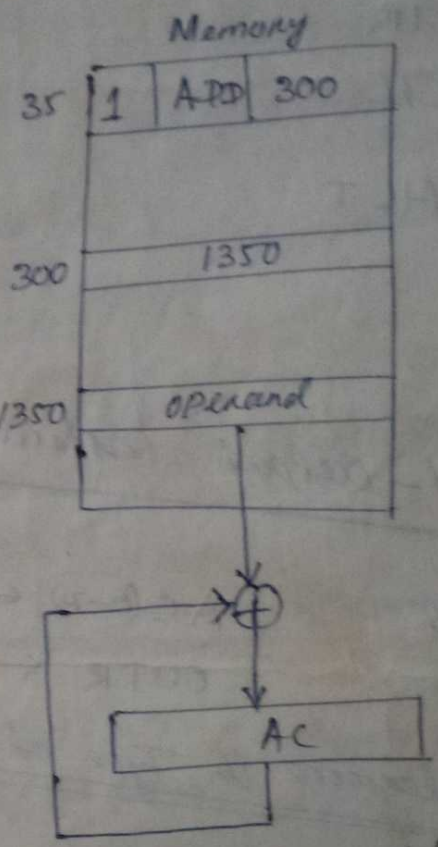
* Bit position 15 of the instruction code is designated by the symbol I but is not used as a mode bit when operation code is equal to 111.

* Only 3-bits of the instruction are used for the operation code. So it may seem that the computer is restricted to maximum

of eight distinct operations. Best
Register-Reference and Input-Output instructions
use the remaining 12-bits as part of the
operation code, so the total no of instructions
will exceed eight.



Direct Address



Indirect Address.

Some of the Instructions are given below.

Memory-Reference Instructions :-

Symbol	Symbol description
AND	$AC \leftarrow AC \wedge M[MAR]$
ADD	$AC \leftarrow AC + M[MAR]$
LDA	$AC \leftarrow M[MAR]$
STA	$M[MAR] \leftarrow AC$
BUN	$PC \leftarrow AR$ (Branch unconditionally)

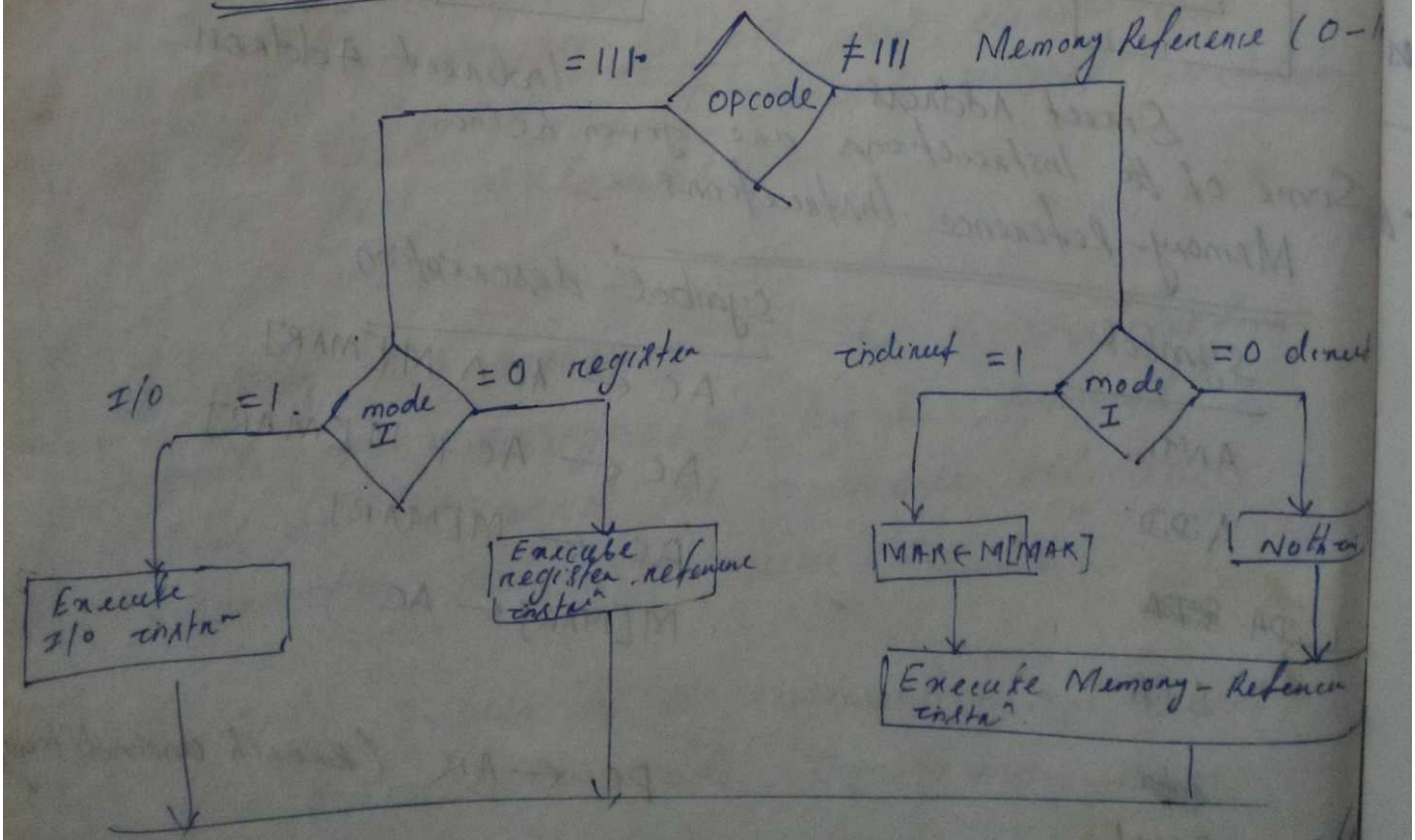
Register - Reference Instructions

CLA	Clear AC
CMA	$AC \leftarrow \overline{AC}$
CIR	Circulate Right
CIL	Circulate Left.
HLT	Halt computer

Input - output Instructions

INP	$AC(0-7) \leftarrow INPR$	$FGI \leftarrow 0$	Input character
OUT	$OUTR \leftarrow AC(0-7)$	$FGO \leftarrow 0$	output char

Determine the Type of Instruction



Flowchart for instruction cycle

Register Reference instructions

[IR(i) = Bi [bit in IR (0-11) that specifies the register operation]

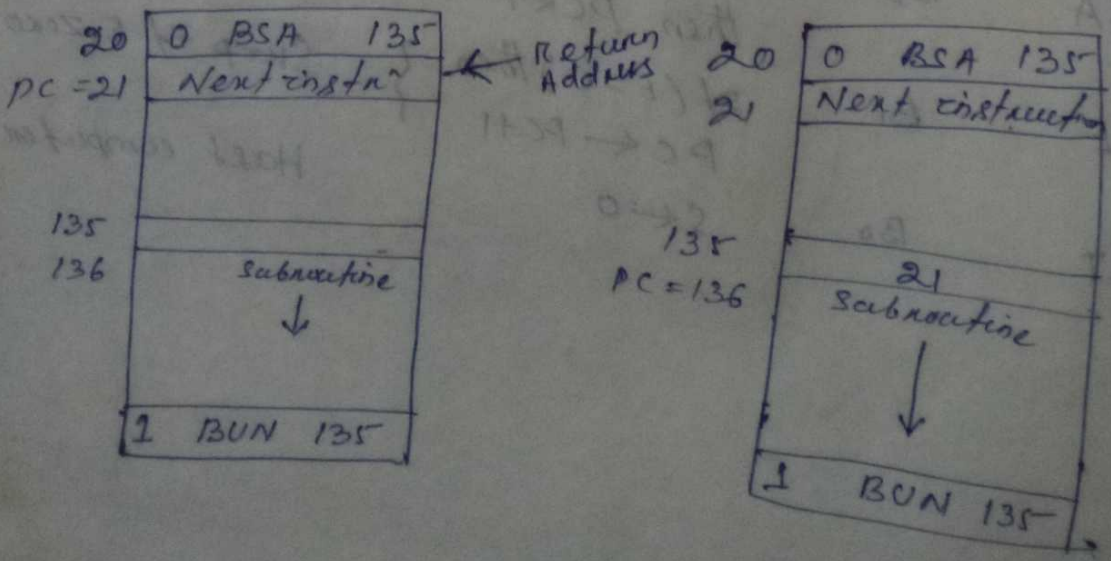
Symbol	bit		
CLA	B_{11}	$AC \leftarrow 0$	clear AC
CLE	B_{10}	$E \leftarrow 0$	clear E
CMA	B_9	$AC \leftarrow \overline{AC}$	complement AC
CME	B_8	$E \leftarrow \overline{E}$	complement E
CIR	B_7	$AC \leftarrow shr\ AC$, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$	Circulate right
CIL	B_6	$AC \leftarrow shl\ AC$, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$	Circulate left
INC	B_5	$AC \leftarrow AC + 1$	Increment AC
SPA	B_4	if $(AC(15) = 0)$ then $PC \leftarrow PC + 1$	Skip if +ve
SNA	B_3	if $(AC(15) = 1)$ then $PC \leftarrow PC + 1$	Skip if -ve
SZA	B_2	if $(AC = 0)$ then $PC \leftarrow PC + 1$	Skip if Zero
SZE	B_1	if $(E = 0)$ then $PC \leftarrow PC + 1$	Skip if E zero
HLT	B_0	$S \leftarrow 0$	Halt computer

Memory - Reference Instructions

Symbol	operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[MAR]$
ADD	D_1	$AC \leftarrow AC + M[MAR]$, $E \leftarrow Cout$
LDA (Load)	D_2	$AC \leftarrow M[MAR]$
STA (Store)	D_3	$M[MAR] \leftarrow AC$
BUN (Branch unconditionally)	D_4	$PC \leftarrow MAR$
BSA	D_5	$M[MAR] \leftarrow PC$, $PC \leftarrow MAR + 1$
ISZ	D_6	$M[MAR] \leftarrow M[MAR] + 1$, if $M[MAR] + 1 = 0$ then $PC \leftarrow PC + 1$

BSA (D_5) :- Branch and Save Return Address

The BSA instruction is assumed to be in memory at address 20. The I-bit is 0 and the address part of the instruction is 135.



After fetch and decode the PC is incremented to 21 which is the return address.

* MAR holds the effective address 135. The BSA instruction performs the following numerical operation.

M[135] ← 21 , PC ← 135+1 = 136

* The return address 21 is stored in memory location 135 and control continues with the subroutine program starting from address 136.

* The return to the original program (at address 21) is accomplished by means of an indirect BUN instruction placed at the end of the subroutine.

Input-Output Instructions