28/01/xxx
Tuesday

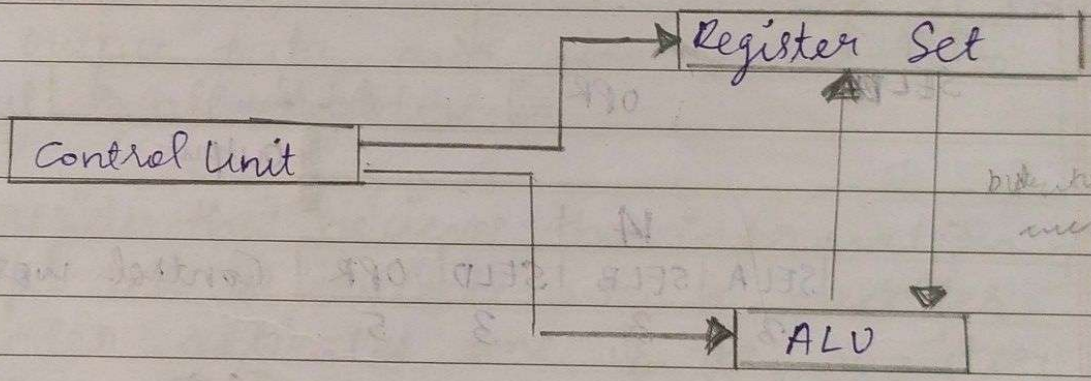
UNIT - II

Central Processing Unit

Introduction

The part of the computer that performs bulk of data processing operations is called a central processing unit. (CPU)

The CPU is made up of three major parts:-
- Control Unit
- Register set
- Arithmetic and Logic Unit (ALU)



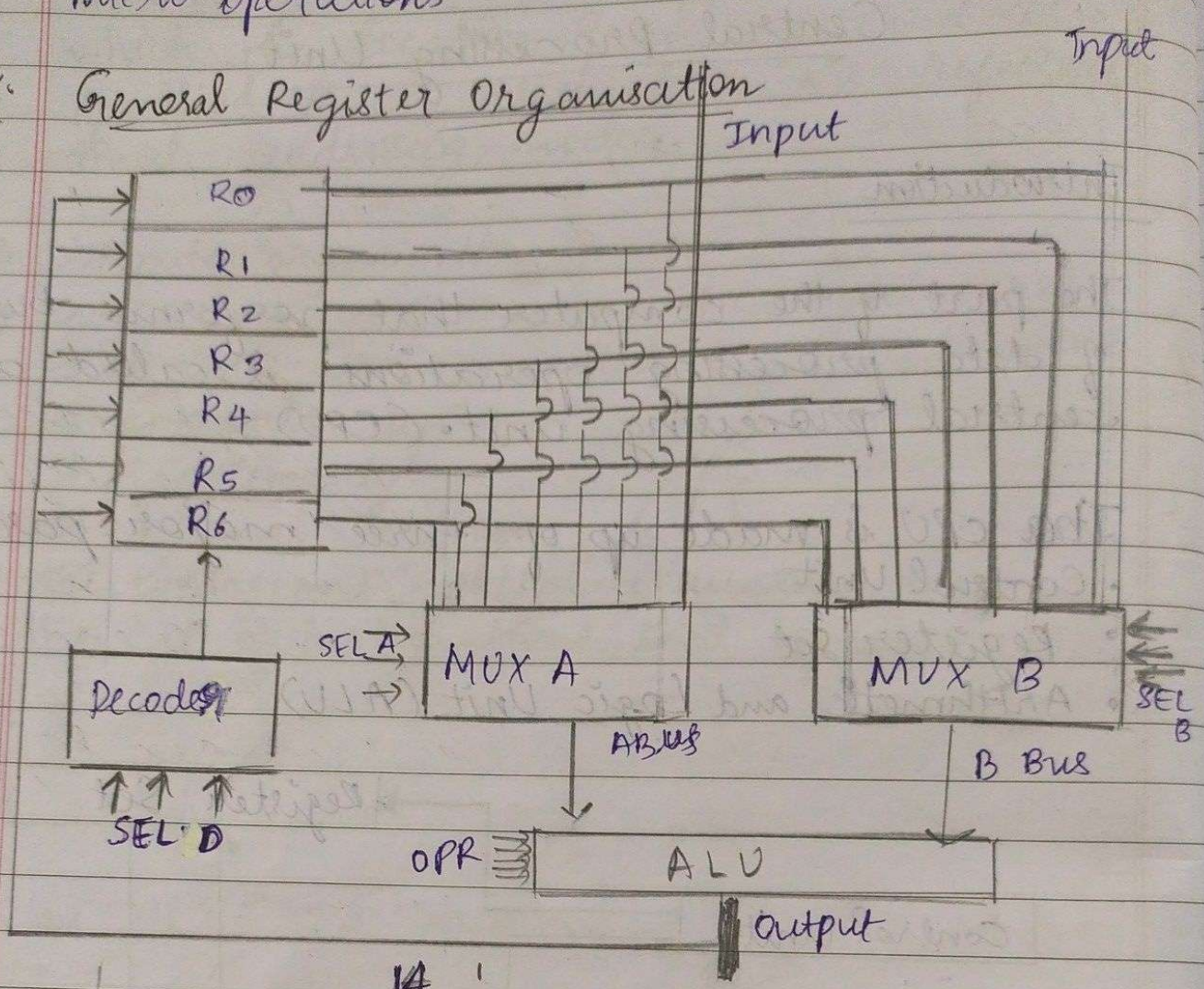
The register set stores the inter-mediate data used during the execution of instructions.

The ALU performs all the arithmetic and logical operations or microoperations for a executing the instructions.

The Control Unit supervises the transfer of information among the registers and instruct the ALU to perform the various

micro operations.

General Register Organisation



14

SEL A	SEL B	SEL D	OPR	Control word
3	3	3	5	

multiplexer

31/01/2020 — The registers communicate with each other not only for direct data transfer but also by performing various operations while Hence it is necessary to provide a common unit that can perform all the arithmetic, logic and shift operations in the processor.

A Bus organisation for a seven register is shown in the above figure.

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

The A and B buses form the input to a common arithmetic unit.
The operation selected in the ALU determines the arithmetic or logic microoperations that is to be performed.

The 5 bit OPR decides the microoperations.
The result of the micro operation is available for output data and also those into the input of all the registers.

The register that recieves the information from output bus is selected by a decoder. The decoder activates one of the register to load input and thus providing a path between the data in the output bus and the input of the selected destination register.

The control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting various components in the system.

The control unit must provide different

binary selection variables for the smooth operations.

The different control signals are:-

① SEL A (MUX A selector)

This is a selection line for the MUX A. It place the content of various registers into Bus A.

② SEL B (MUX B selector)

This is a selection line for the MUX B. It is used to place the content of registers into Bus B.

③ SEL D

It is the decoder destination selector

④ OPR

ALU operation selector :- It provides the arithmetic or logic and shift operations to be performed.

→ The four control selection variables are generated in the control unit and must be available at the beginning of a ~~cycle~~ clock cycle.

The data from the two source registers propagate through the gates in the multiplexers and ALU, to the output bus and to the destination register.

Control Word

There are 14 binary selection inputs and their combined values specifies a control word.
It is a 14 bit word consists of 4 fields - 3 fields contains 3 bits each and 1 field has 5 bits.

A 3 bits of SELA, selects a source register for A input of the ALU.
The 3 bits of SELB select a ~~destinati~~ register for B input of the ALU.
The 3 bits of SELD select a destination register using the decoder and it loads the output.

The 5 bit of the OPR selects one of the operation in the ALU.
The 14 bit control word when applied to the selection input specifies a particular micro operation.

Binary code	SEL A	SEL B	SEL D
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

OPR	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add A + B	ADD
00011		
00100		
00101	Subtract A - B	SUB
00110	Decrement A	DEC A
00111		AND
01000		OR
01001		XOR
01010		
01011		
01100		
01110		
01111		
10000	Shift right A	SHRA

Stack Organisation

A stack is a storage device that stores information in such a manner that the item stored last is the first item retrived.

The operation of a stack can be compared to a stack of tray. The last tray placed on the top of the stack is the first to be taken off.

The stack in a digital computer is essentially a memory unit with an address register. The register that hold

the address of the top element of the stack is known as stack pointer (SP).

The two operations of the stack are:-

① PUSH (insertion of items)

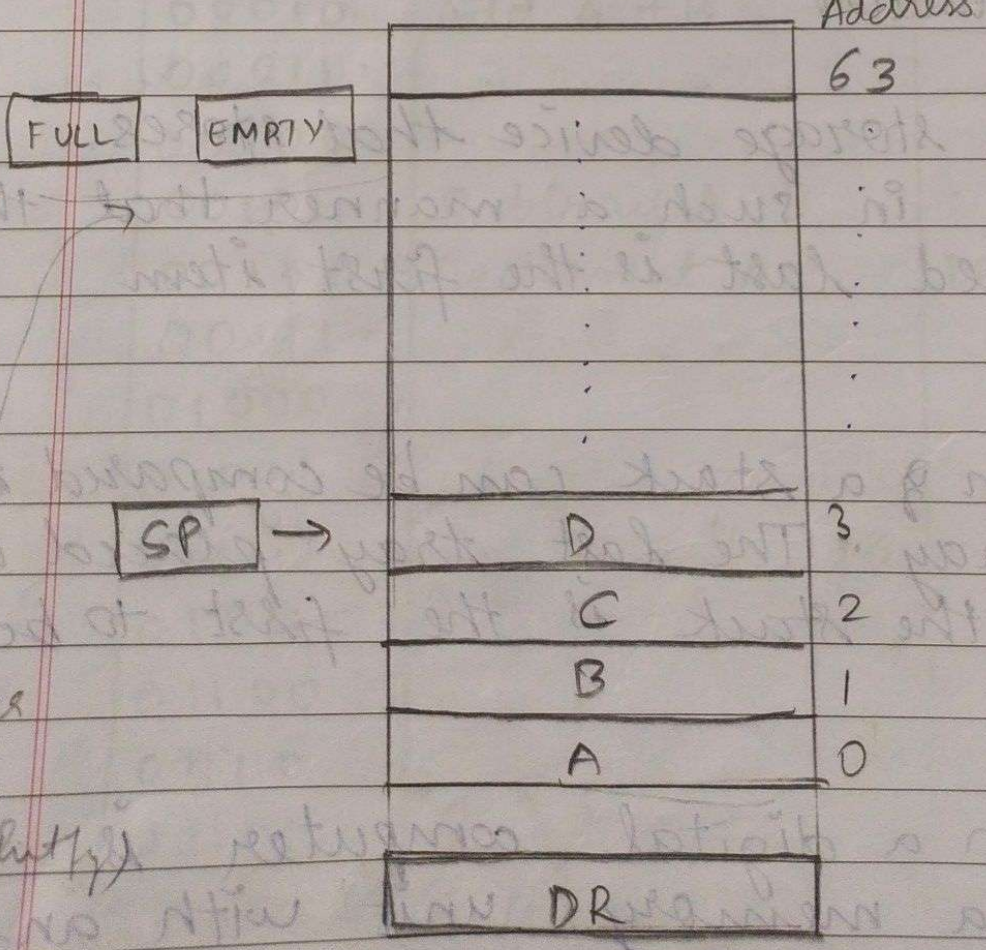
② POP (deletion of items)

The stack pointer register is incremented by 1 during PUSH operation and decremented by 1 during POP operation.

03/02/2020
Monday

Stack register / Register Stack

Block diagram of a 64 word stack



A stack can be placed in a portion of a large memory or it can be organised as a collection of finite number of memory words or registers.

The figure shows the organisation of a 64 word register stack.

- SP (Stack pointer)

The SP register or SP contains a the address of the word that is currently on the top of the stack.

Four items are placed in the stack - A, B, C and D.
The item D is on top of the stack and the content of SP is now 3. To remove the top element, the stack is performed by a pop operation by reading the memory address 3 and taking the item from that address and decrementing the value of SP. Now the top element of the stack is C.

To insert a new item the stack is pushed by incrementing sp and writing a word in the next higher location in the stack

Stack Overflow and Stack Underflow

$2^6 = 64$

In a 64 word stack, the stack pointer contains 6 bit address ($2^6 = 64$). Whenever

the stack is full and we are attempting to push an element, the stack overflow condition occurs.
whenever the stack is empty and we are attempting to pop an element, the stack underflow condition occurs.

- **FULL**

Full is a 1 bit register, which is set to 1 when the stack is full.

- **EMPTY**

It is a 1 bit register, and it is set to 1 when there is no elements in the stack.

- **DR** (data register)

It is the data register that hold the data to be returned into the stack or read out of the stack

Initially SP is cleared to 0, EMPTY is set to 1 and FULL is ~~set to~~ cleared to 0

Now SP points to the word at the address location 0 and the stack is marked as empty and not full.

If the stack is not full (if full = 0), a

new item can be inserted with a PUSH operation

An example of a push operation is given below:-

SP ← SP + 1 (Increment SP)
M(SP) ← DR (write item from DR to top of the stack)

7/02/2020 Wednesday

The stack pointer is incremented so that it points to the address of next higher word. A memory write operation inserts the word from DR into the top of the stack.

SP holds the address of the top of the stack and M[SP] denotes the memory word or data presently available in SP.

If the SP reaches 0 and the stack is full of items, then FULL is set to 1.

POP Operation

A new item can be deleted from the stack if the stack is non-empty. That means EMPTY = 0

The POP operation consists of the following steps

1. DR ← M[SP] (Remove item from stack top)
2. SP ← SP - 1 (Decrement SP)

The top element is read from the stack into DR, the stack pointer is then decremented.
If the value of SP reaches 0 and the stack is empty, the empty register is set to 1.

Applications of Stack

Using stack we can evaluate an arithmetic expression. The arithmetic expression are written in infix notations, where each operators are written in between operands.

eg:- A + B * C - D

We can also represent expressions in prefix notation also known as polish notation. The operators appears before the operands in this type of expression.

eg:- A + (B * C) - D
(A) + (* B C) - (D)
(+ A * B C) - (D)
- + A * B C D

This expression can also can be expressed in postfix notation. It is also known as reverse polish notation. The operator appears after the operands in this type of notations.

eg:- A + B * C) - D

A + (BC *) - D

(A BC * +) - (D)

ABC * + D -

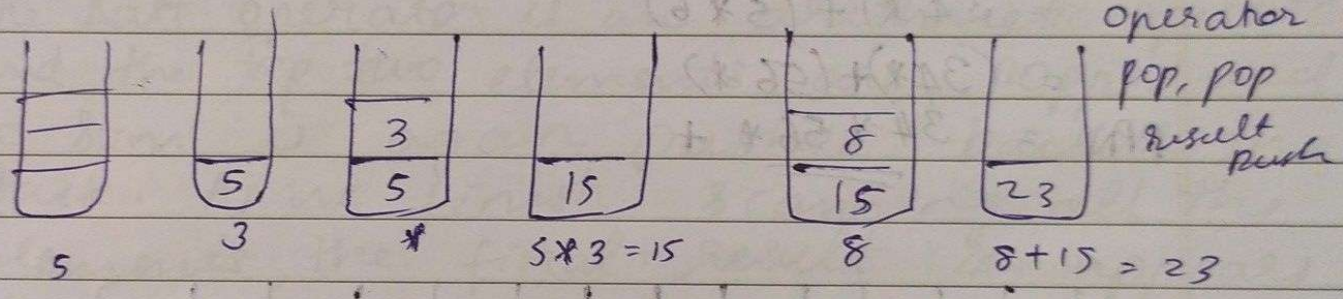
Evaluation of arithmetic expression using reverse polish notation. (RPN)

exp:- 5 * 3 + 8

RPN (post) :- 53 * 8 +

operand - push

operator pop, pop result push



5/02/2020 Thursday The evaluation of arithmetic expression is done by RPN using stack, the following procedure is applied

1. Convert the arithmetic expression into equivalent RPN.
2. The expression is then scanned from left to right.
3. whenever an operand appear, it is pushed into the stack in the order in which they appear.
4. whenever an operator appears the two top

most operands are popped and the corresponding operation is performed. The result of the operation is again pushed into the stack.

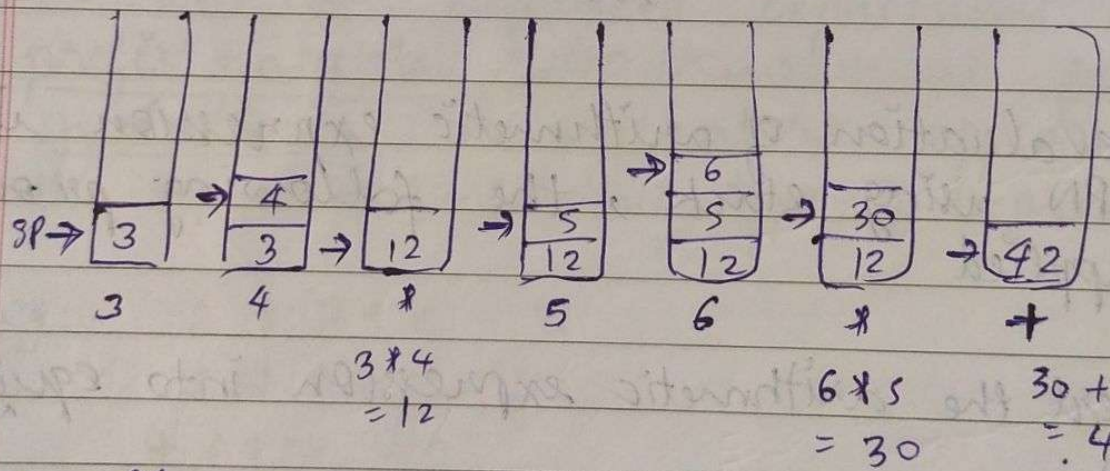
4. By pushing the operands into the stack continuously and performing the operation as defined above, the expression is evaluated in proper order and the final result remains on top of the stack

Eg:- Consider the expression (3*4)+(5*6)

= (34*)+(5*6)

= (34*)+(56*)

RPN = 34*56*+



Result → Top element of stack

= 42

Each box represents one stack and the arrow mark allow points to the top of the stack.

Scanning the expression from left to right and we encountered two operands 3 and 4

the first number 3 is pushed into the stack and then the next number 4.
The next symbol scanned is an operator and the stack is then popped 2 times. As the symbol is * the product of the popped elements is again pushed back into the stack.
Next we again encounter 2 operands 5 and 6 they are pushed into the stack. Next * operator pops the top two elements and the product is again pushed back to the stack.
The last operator is an arithmetic operator and the top two elements are popped and the sum is again pushed back to the stack. As we finish scanning all the elements, the final result appears at the top of the stack.

7/02/2020 eg:- Convert the following expressions into postfix and prefix

1. $(A+B)*[C*(D+E)+F]$

postfix

$= AB+*[C*(DE+)+F]$

$= (AB+)*[(CDE+*)+F]$

$= (AB+)*[CDE+*F+)$

$= AB+CDE+*F+*$

prefix

$= (A+B)*[C*(D+E)+F]$

$= +AB*[C*+DE+F]$

$= +AB * [*CE + DE + F]$

$= +AB * (+*C+DEF)$

$= +AB(+*$

$= *+AB+*C+DEF$

2. $A+B*C/D*(F+E)$

prefix

$= A+B*C/D*(+FE)$

$= A+B*/CD*(+FE)$

$= *A+B/CD$

prefix

$= A+B*C/D*(F+E)$

$= A+B*C/D*(+FE)$

$= A+B*/CD*(+FE)$

$= A+*B/CD*+FE$

$= +A**B/CD+FE$

postfix

$= A+B*C/D*(F+E)$

$= A+B*C/D*(FE+)$

$= A+B*CD/*(FE+)$

$= A+BCD/*FE+*$

$= ABCD/*FE+*+$

3. A/B *C +D -E *(F+G)

postfix

= A/B *C +D-E *FG+
= (AB/) *C +D-E *(FG+)
= AB/C* +D-(EFG+*)
= AB/C*D+ -(EFG+*
= AB/C*D+EFG+*-

prefix

= A/B *C +D-E *(F+G)
= A/B *C +D-E *(+FG)
= (/AB) *C +D-E *(+FG)
= */ABC +D-E *(+FG)
= (*/ABC)+D-*E(+FG)
= (+*/ABCD)-(*E+FG)
= -+*/ABCD*E+FG

4. A*B +C/D

postfix :

= A*B +CD/
= AB* +CD/
= +AB*CD/+

prefix

= A*B +/CD
= *AB +/CD
= +*AB/CD

5. (A*B) + (C*D) + E/F

prefix

(A*B) + (C*D) + /EF
*AB + *CD + /EF
+ *AB *CD + /EF
++ *AB *CD /EF

postfix

(A*B) + (C*D) + EF/
AB* + (C*D) + EF/
AB* + CD* + EF/
AB* CD* + EF/ +

07/02/2020
Friday

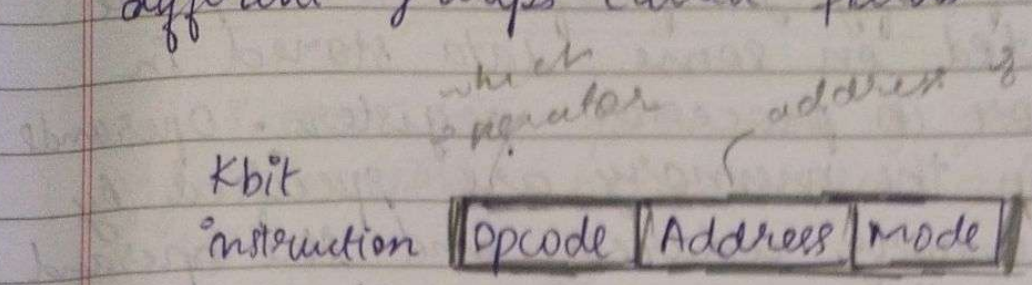
Instruction Formats

The function of control unit within the CPU is used to interpret each instructing code and provide the necessary control function needed to process the instruction.

The format of an instruction is usually represented in a rectangular box symbolising the bit of instructions as they appear in the memory word.

The bit of instructions are divided into 3

different groups called fields.



1. Opcode

It is the operation code field that specifies the operation to be performed.

2. Address field

It specifies a memory address or a processor register.

3. Mode field

It specifies the way the operand or the effective address is determined.

The operation field / opcode field of an instruction is a group of bits that define various processor operations such as ADD, SUB, SHIFT, DIV etc....

The bits that define the mode field of an instruction code specifies a variety of alternatives for choosing the operands from the given address.

Operation specifying specified in the opcode are executed on some data stored in memory or in processor registers. Operands residing in the memory are specified by their memory address and the operands residing in the processor register are specified with a register address.

Different types of CPU organisation

Computer may have instructions of different lengths, containing various number of addresses.
The number of address field in the instruction format of a computer depends upon the internal organisation of its register.
There are three types of CPU organisations

(1) Single accumulator organisation

All operations are performed with an implied accumulator register, regist the instruction formats in this type of computers use only one address field.

eg:- ADD X
This instruction specifies an arithmetic addition operation, where X is the address of the operand and the second operand is within the accumulator itself.
AC ← (X) + (AC)
~~ACC~~

(2) General register organisation

The instruction format of general register organisations needs three address register fields.

eg:- ADD R_1, R_2, R_3

R_3 gets the value added of R_1 added with R_2

$R_3 \leftarrow [R_1] + [R_2]$

The number of address fields in the instruction can be reduced from three to two if the destination register is same as the one of the source register.

ADD R_1, R_2, R_3

$R_1 \leftarrow R_2 + R_3$

is same as

ADD R_2, R_3

$R_2 \leftarrow R_2 + R_3$

Move R_2, R_1

(3) Stack organisation

In stack organization always push and POP instruction require only one address field.

eg:- PUSH A

TOS $\leftarrow$ [A] (TOS - top of the stack)

Classification of instructions

The instruction types are of three types:-

(1) Three address instruction

(2) Two Address instruction
(3) One Address instruction

(2)

(1) Three Address Instruction

The operation of adding two numbers using the statement C = A+B can be explained using all the three categories of instruction types.

In order to add the values of memory location A and B and storing the result in C, you can use the three address instruction as follows:-

ADD A, B, C

Operands A and B are called the source operands and C is called the destination operand.

Add is the operation to be performed on the operands.

The general instruction of this type can be written as:-
operation source 1, source 2, destination.

eg: ADD A, B, C

ie [C] ← [A] + [B]

(2) Two Address Instruction

In this type of instructions each instruction will be having only two operands;

General form is

Operation source, destination

eg:- ADD A , B

ie [B] ← [A] + [B]

MOVE B , C

ie [C] ← [B]

(3) One Address Instruction

When a machine instruction specify only one memory operand along with the operation, it is known as one address instruction.

When a second operand is needed, it is understood implicitly to be in an unique location - a process register called accumulator

The general form is :-
Operation source / destination

TO perform the statement C = A + B using a one address instruction, we need to perform the following sequence of

instructions.

eg:- Load A [ACC] ← [A]

Add B [ACC] ← [ACC] + [B]

Store C [C] ← [ACC]

1.
2.
3.

11/02/2020
Tuesday

* Addressing Modes

The operation field of an instruction specifies the operation to be performed This operation must be executed on some data stored in the computer register or memory

The way the operands are chosen during program execution depends on the addressing modes of the instruction. The addressing modes specifies a rule for ~~interpting the addres~~ interpreting or modifying the address field of the instruction before the operand is actually referred.

Advantage of using addressing mode is

1. to give programming flexibility
2. to reduce the number of bits in the instruction

The control unit of a computer is designed to go through an instruction cycle which is divided into three phases

1. Fetch the instruction from the memory
2. Decode the instruction
3. Execute the instruction.

} Instruction cycle

There is one register in the computer called PC or program counter. This keeps track of the instruction in the program, which is stored in the memory.

PC holds the address of the next instruction to be executed and it is incremented each time after an instruction is fetched from the memory.

The decoding is done in step 2, which determines the operation to be performed. The computer then execute the instruction and return to step 1 to fetch the next instruction in the sequence

Mode field

It is used to locate the operands needed for the operation.
There may or may not be an address field in the instruction. If there is an address field, it maybe a memory address or a processor register.

The following are the different addressing modes used in instruction formats.

1. Implicit mode

In this mode operands are specified implicitly in the instruction itself.

eg:- COMPI ACC
(compliment)

The above example is an implicit mode because the operand in the accumulator is implied in the instruction itself.

POP is also an example of implicit mode.

2. Immediate mode

The operand is specified in the instruction itself. An immediate mode instruction has an operand field rather than an address field.

The operand field contains the actual operand to be used in with the operation specified in the instruction.

These instructions are useful for initializing registers to a constant value

eg:- ADD 5, 8
MUL 2, 10

3. Register mode

The address field of an instruction may specify either a memory or a processor register.
When the address field specify a processor register, the instruction is said to be in register mode.

The operands are in the registers that reside within the CPU

eg:- ADD R_1, R_2.
SUB R0, R7.

4. Register Indirect mode

In this type of mode the instruction specifies a register whose content gives the address of the operand in memory.
OR the selected register gives the address of the operand rather than the operand itself.
Before using the register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register before the execution of the instruction.

eg ADD

EA = (R1)

The advantage of this mode is the address field of the instruction uses fewer bits to select a register than would have being required to specify a memory address directly.

5. Auto Incrementing or Auto Decrementing Mode

This is similar to register indirect mode except that the register is incremented or decremented to obtain the actual operand. eg ADD D, B EA = (R) ± 1

address which contain actual value

Effective address

Effective address is defined to be the memory address obtained from the computation done by the addressing mode. otherwise the actual memory address of an operand is known as effective address.

14/02/2020 Friday

Address of the actual operand value

6. Direct Address Mode

The effective address is equal to the address part of the instruction. The operands resides in memory and its address is given directly by the address field of the instruction. In a branch type instruction the address field specifies the actual branch address.

eg:- ADD A, B EA = A

(actual operand value)

7. Indirect Address Mode

The address field of the instruction gives the address where the effective address is stored in memory. Control fetches the instruction from memory and uses its address part to access main memory again to read the effective address. eg: ADD, A, B EA = [A] OR [B]

The effective address of this mode is obtained by effective address EA = Address part of instruction + content of the CPU register.

8. Relative Address Mode

In this mode the content of the PC (program counter) is added to the address part of the instruction to obtain the effective address. The address part of the instruction gives usually a signed number, which can be either positive or negative.

When this number is added to the content of the PC, the result produce an EA; whose position in memory is relative to the address of the next instruction.

eg:- The number 825 is the content of PC, address part of the instruction contain 24 The instruction and location 825 is read from memory during the fetch phase and the PC is then incremented by 1.

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

The relative address is often used with branch type instruction.

EA = (address) + (PC)

9. Indexed Addressing Mode

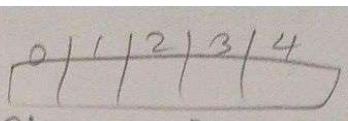
In this mode the content of the indexed register is added to the address part of the instruction to obtain the effective address. The index register is a special CPU register that contains an index value. The address field of the instruction defines the beginning of address of data array in memory.

Each operand in the array is stored in memory relative to the beginning address. The distance between the beginning address and address of operand is the index value stored in the index register. Any operand in the array can be accused with the same instruction provided that the index register contains the correct index value.

eg EA: (Address) + (TR)

10. Base Register Addressing Mode

The content of base register is added to the address part of the instruction to obtain



the EA. It is similar to index addressing mode except that the register is now called the base register instead of an indexed register.

The difference between the two modes is in the way they are used rather than in the way they are computed.

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

This addressing mode is used for the relocation of programs in memory. When programs and datas are moved from one segment of memory to another the address value of instruction must reflect the change of position. EA = (Address) + (BR)

25/02/2020
Tuesday