Computer Architecture

II - CSE
IV - SEM.

UNIT - I
Part B & C.

(1) Explain in detail the components of computer system.
(or)
What are the functional units? Discuss on basic functional units of a computer. (Apl may-18, Nov/dec-16, may/june-16, Nov/dec-15.)

Ans:-

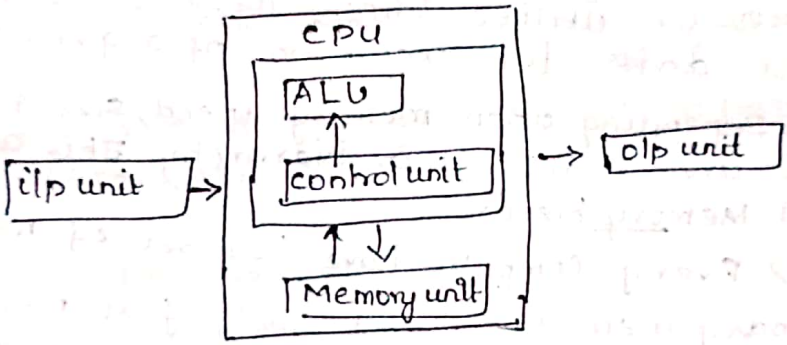
Functional units / components of computer system:

Functional units of a computer are fundamental parts that form a computer.

Every computer is made up of 5 independent fn. units. These are as follows,

(1) Input unit (2) output unit (3) Memory unit (4) Arithmetic and logic unit (5) control unit.

Block Diagram of a computer



(1) Input unit :-

=> Input unit accepts the data from the outside world. eg: commonly used i/p unit is keyboard.

=> keyboard is a device through which user supply data to the computer.

=> other i/p unit include mouse, scanners, microphone, track balls, joysticks, touchscreen etc.

=> These units gets connected to the computer by using an interface.

=> The mouse device which acts on graphical displays & provides fast accessing of different entities.

=> Scanner responsible for scanning various images, texts etc. Microphone take voice signals & manipulate them accordingly.

→ As technology is getting advanced touch screens are replacing other i/ps such as keyboard, mouse, pri: etc

↳ It provides an easy interaction between the user and the computer by allowing the users to use their fingers for accessing the device eg. laptop, smartphones

(2) o/p unit :-

It is a device through which data is supplied to the outside word.

basic o/p device is monitor. others are printer, speaker etc. Printer is used to print text or images. Through speaker sound is heard.

(3) Memory unit :-

⇒ It is a unit which is responsible for storing pgm and large volumes of computer data.

⇒ Basically, whenever a pgm is under execution, the processor utilize these memories to extract required data for execution of a given pgm.

⇒ Depending upon memory speed, size & cost the memories are arranged in hierarchy. This arrangement is called Memory hierarchy.

⇒ Every computer have two set of memories

(i) primary memory ⇒ extremely fast memories. eg RAM.
⇒ These memories are nothing but semiconductor elements capable of storing single bit of infor.
⇒ These elements together store certain length of data called words.
⇒ An extremely fast memory which is provided to support the speed of processor is referred to as cache memory

(ii) Secondary memory :- ⇒ It is less cost and at the same time provide large storage capacity
⇒ eg Magnetic tapes, CDROMs etc.

(4) Arithmetic & logic unit:

→ ALU is responsible for performing arithmetic (or) logical related operations.

→ for eg, if user wants to perform multi operation the processor initially fetches the required data, i.e operands & perform ALU.

→ This unit examines the data & accordingly perform the required computation & then return the computer.

(5) Control Unit:-

=> It is a major component which govern the activities of other functional devices.

=> This is done by transmitting the required control signals to various other units.

=> It inhibits the activities of these units & instruct them to perform their operations in the prescribed timings.

x——x——x

(2) Explain the important Measures of the performance of a computer and derive the basic performance equation. (April/May-17).

Ans:-

Performance Measures of Computer.

The various measures are

(1.) Response Time or Execution Time => It is defined as the total time taken by the computer to complete a task.

(2.) Throughput or Bandwidth => It is defined as the total number of tasks completed per unit time.

(3.) CPU Execution time (or) CPU Time => It is defined as the total time taken by the CPU for executing a particular task.

→ This does not include the time for executing other programs or the time taken in waiting for I/o.

(4) User CPU Time => It is defined as the time taken by the CPU for executing a program.

(5) System CPU Time => It is defined as the time taken by CPU for executing task in operating system.

(6) Clock Cycle as Clock Tick => It is defined as the time taken by the clock to one cycle or rotation.

(7) Clock period => It is defined as the total number of clock ticks in a clock or length of a clock cycle.

(8) Clock cycles per Instruction (CPI) => It is defined as the avg no of clock cycles taken by an instr for pgm execution.

Derivation of Various Performance Equations.

The performance of a system can be increased by decreasing the response time of a task. This can be represented as

$$\text{performance} = \frac{1}{\text{Response time (Exe time)}}$$

Eg :- two computer (C_1 & C_2) if C_1 is greater than C_2. then C_2 is said to have greater response time.

This can be derived as

$$\text{Performance}(C_1) > \text{Performace}(C_2)$$

$$\frac{1}{\text{Response time}(C_1)} > \frac{1}{\text{Response time}(C_2)}$$

$$\text{Response time}(C_2) > \text{Response time}(C_1)$$

Hence it can be said that if computer 1 is faster than Computer 2, then computer 2 takes long time for execution.

CPU Performance :-

CPU performance or CPU time for a program execution can be computed using the number of CPU clock cycles for a program & clock cycle time as follows,

CPU performance = No of CPU clock cycles for a program x clock cycle time.

Since clock cycle time is inverse to clock rate.

$$\therefore \text{CPU performance} = \frac{\text{No of CPU clock cycles}}{\text{clock rate}}$$

=> CPU performan can be maximized by minimizing the total No of clock cycles required for the pgm execution.

Instruction Performance :-

The performance of an instr can be computed using the total No of instrs in a pgm & its CPI as follows,

Instr Performance = No of pgm Instr × CPI of a pgm.

Basic Performance Equation :-

↳ It can be computed using Instr count, (IC) CPI and clock cycle time.

↳ The IC gives the total No of instr that are executed by the pgm

∴ Basic CPU performance equation } = Instruction count × CPI × Clock cycle time.

(3). What is an Addressing Modes? Elaborate different types of addressing Modes with eg. (A|M-17, may|J=16 N|D-16, Nov|Dec-15 ap|may-15)

Ans :-

⇒ An addressing mode is a method of specifying an operand.

↳ A computer pgm is a sequence of instructions.

↳ Each instr specifies the operation (such as, load, add, branch etc) to be performed in the operation code field of the instruction format.

↳ In order to enlarge the capability and ~~diff types of addressing~~ computing power of an instr it is desirable to have different types of addressing capabilities in an instr format.

Types of Addressing Modes :-

1. Immediate Addressing Modes
2. Register Addressing Mode
3. Based or Displacement Addressing Mode
4. PC-relative Addressing Mode
5. Pseudodirect Addressing Mode

(1) Immediate Addressing Mode :-

→ In this mode, operand is part of the instr instead of the content of a register or memory location

⇒ Instr format for Immediate Addressing mode is

opcode	rs	rt	Immediate

⇒ It has operand field rather than address field.

⇒ Since the data are encoded directly into the instruction, Immediate operand normally represent constant data.

Ex :- Add $s0, $s1, 15;

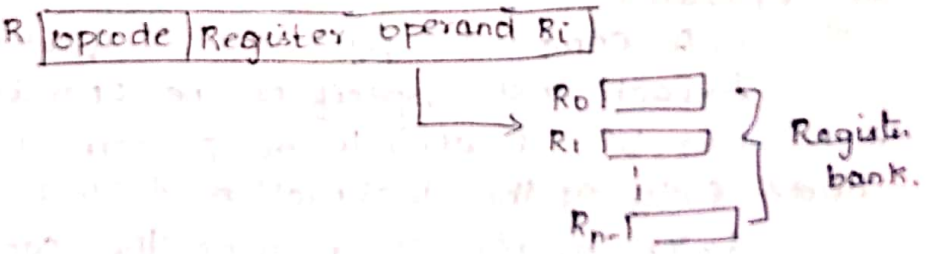
(2) Register Addressing Mode :-

⇒ In this mode, the operand to be accessed is present in CPU register.

⇒ Actually, all computer systems are usually provides with some addressable registers.

⇒ The k bits of at the operand field can denote any one of 2k registers operand holding the data word.

Ex : Add $s3, $s5, $s6.



(3) Based or Displacement Addressing Mode

↳ In this mode, the instr contains an address.

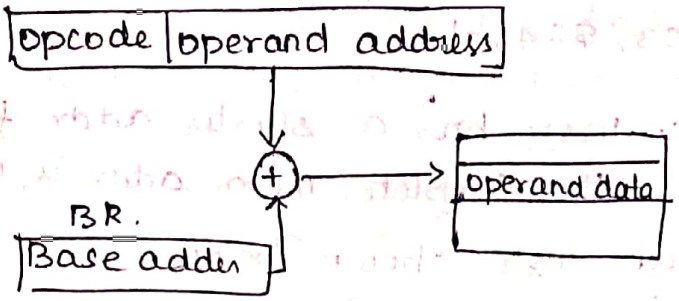
↳ This address is added to the data present in the base register to get the effective address.

Thus,

Effective address = [BR] + Addr specified in the instr

where, [BR] = Data Present in base regist.

⇒ The addr specified in the instr is the difference between the base addr & effective addr whereas, base reg holds base addr.



eg:-

C[BR] = 1

1000	10
1001	20
1002	30
1003	40

LW $t0, 32($s3).

⇒ Here base reg contains starting of data addr 1000. When it is needed to access 10, addr field of instr should contain 0, thus effective addr = 1000+0 = 1000.

⇒ When it is needed to access 40, addr field is 3. ∴ Eeff addr = 1000 + 3 = 1003.

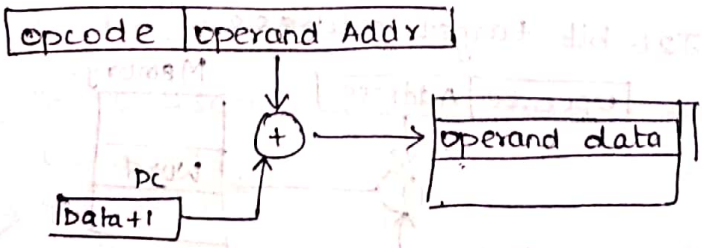
⇒ Thus depending upon data values accept addr field should be changed & base reg content is fixed.

(4) PC-relative Addressing Mode:-

→ In this mode, the instr contains a relative addr which is actually a signed nos (in 2's comp form) which can be +ve or -ve.

→ This addr is added to the contents of the pgm counter to obtain effective addr.

Effective addr = (PC+1) + Addr specified in the Instr

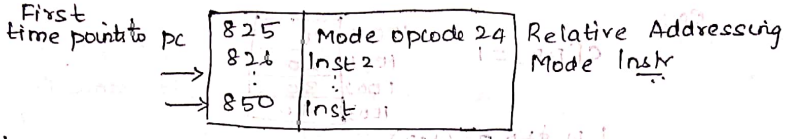


→ This type of addressing mode is commonly used when dealing with branch-type instr where the branch addr is located in the area where the instr word is also stored

Eg:
beq $s3, $s4, L1.

⇒ Adv :- Instr has a shorter addr field as the complete mem addr is not specified.

consider an instr shown below,



⇒ If the instr is present at 825 mem loc. which is a relative addressing mode instr then it is executed as follows,

↳ After fetch cycle, the instr is read and PC is incremented to 826 during execute cycle,

↳ Computer adds addr field of the instr to PC & the result is stored in PC

PC = 826 + 24 = 850.

↳ Thus after executing the instr at 825 memory location, control branches to the instr present at 850, and in between instr are skipped.

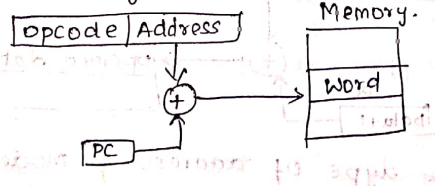
⇒ All branch instr comes under relative addressing mode.

(5) Pseudo direct Addressing Mode :-

⇒ The mem addr is present in the instr itself.

⇒ The effective addr is calculated by concatenating 26 bit immediate value & the upper 4 bit of the PC & lower two bit are set to 00.

Eg :- J26 bit target address.



(4) Discuss about the various techniques to represent instructions in a computer System. (A|M 2015)

Ans :-

→ Instructions in CA refers to set of commands given to a computer processor by a pgm

→ It is represented as nos that are stored in special storage areas called registers

→ Commonly used reg in MIPS assembly language re

$s0 to $s7 => maps to the reg 16 to 23

$t0 to $t7 => " " " " 8 to 15

Eg :-

add $t0, $s1, $s2.

The decimal representation of given instr is

0	17	18	8	0	32

binary rep is :-

6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
000000	10001	10010	01000	00000	100000

→ The sum of all bits (6+5+5+5+5+6) = 32 bits.

So MIPS architecture for this instr takes exactly 32 bits.

→ It is necessary to maintain a constant length throughout the instr as it following design principle of computer h/co.

Design principle 3 : Good design demands good compromises.

=> This design principle diff format containing instr uses 32-bit long strings.

MIPS Instr include following Format

1. R-type / R-format
2. I-type / I-format
3. J-type / J-format

(1) R-type :- It is used when the data values are stored in reg are used as an instr.

→ It contains 6 fields

6bit	5bits	5bits	5bits	5bits	6 bits.
OP	rs	rt	rd	shamt	function

Here.

OP => operation code, tells type of instr. 6 bit long (26 to 31)

rs → first register operand used in instr. (21 to 25 bit)

rt → second " " " " " (16 to 20 bit)

rd → destination operand, holds result of 2 ~~instr~~ operand (11 to 15 bit)

shamt ⇒ used when then theire is shifting needed (Left or right shift) (6 to 10 bit)

funct ⇒ function code that helps to select particular variant, used in op field (0 to 5 bits).

eg:- add $s0, $s1, $s2

op	rs	rt	rd	shamt	funct
opcode	$s1	$s2	$s0	0	add

<-r format

0	17	18	8	0	32

→ decimal representation

000000	10001	10010	01000	000000	100000

→ binary representation

② I-type (I-format):-

⇒ used when the instr are immediate (or) data transfer type.

⇒ Fields:

op	rs	rt	const or addr
6bit	5bit	5bit	6bit.

op ⇒ 6bit long string. same functionality as R-format.

rs, rt ⇒ source & target reg.

const or addr ⇒ 16-bit long field can load any word ranges from -2^{15} to or $+2^{15}$ or 32,768 byte (0 to 5 bit)

eg:-

addi: $s6, $s7, -50

here,

opcode = 8 (addi)

$s7 = rs = 22

$s6 = rt = 21

const or addr = -50 (by default)

opcode	$s7	$s6	const or add

8	22	21	-50

→ deci rep

001000	10110	10101	1111111111 001110

→ binary rep

③ J-type

→ It is used when an instr jump operation is to be performed

→ field :

Opcode	Addr
6bit	26bit

Opcode ⇒ jump command issued.
Addr ⇒ addr of register.
It loads & stores the instr. ($I_{2^{15}}$)

Eg :-

J 10000

opcode	addr

→ J-format Opcode = 3(j)

$(10000)_2 = (2500)_2$.

3	2500

→ decimal representation

⑤ Explain Various Instruction Formats & Illustrate the same with an example. (NID=2017).

Ans :- Instruction ⇒ Instructions in CA refers to set of commands given to computer processor by a program.

It ⇒ It is represented as NO that are stored in special storage areas called registers

Instruction Format :-

1. Three Address Instructions :-

→ It includes three variables in each instr.

→ Each var can specify a processor reg or a memory addr.

→ When these instr form the least steps to execute a given pgm.

Eg :- $Y = (A-B)/(C+D*E)$ using 3 addr instr is given below.

S.No	Pgm Instr
1	SUB Y, A, B
2	MUL TEMP, D, E
3	ADD TEMP, TEMP, C
4	DIV Y, Y, TEMP

An exp of pgm :- 1. In line 1, sub. A & B value is stored in Y.

2. In line 2, introduce TEMP var. Mul D,E store in TEMP.

3. In line 3, value c is added to temp & the result is stored back in TEMP

4. Finally, the pgm is concluded by dividing the value of Y by temp & result store in 'Y'.

② Two Address Instructions :-

→ It includes two variables in each instr.

→ Each variable can specify a processor reg or a memory addr

→ No of steps in 2 addr is less compared to one addr. but more when compare to 3-adar instr.

Eg:- Y=(A-B)/(C+D*E)

S.No	pgm Instr
1	MOV Y, A
2	SUB Y, B
3	MOV TEMP, D
4.	MUL TEMP, E
5.	ADD TEMP, C
6.	DIV Y, TEMP.

An explanation of abv pgm.

→ In line 1, value of A is stored Y & storing result back into y.

→ In line 2, the value of 'B' is subtracted from the value of Y

→ Now, the value of 'D' is stored in a temp var (TEMP).

→ In line 5, the value of 'C' is added to TEMP variable & the result is again stored back into TEMP.

→ Finally, the pgm is concluded by dividing the value of 'Y' by TEMP & storing the result back to Y.

③ One Addr Instruction :-

→ use only one var & uses an implied accumulator (AC) reg for all data manipulation.

→ The result of all operations is stored in the AC reg.

eg:- The pgm to execute Y=(A-B)/(C+D*E) using one-addr instr, is given below.

S.No	pgm Instr
1	LOAD D
2	MUL E
3	ADD C
4.	STORE TEMP
5.	LOAD A
6.	SUB B
7	DIV TEMP
8.	STORE Y.

Steps :-

i) Line 1, transfer value into AC.

ii) Line 2, value of E is multiplied with the value store in AC & result is stored back in AC

iii) Line 3, adds the value of C to the value stored in AC.

iv) In line 4, the value of AC is stored into a temp variable TEMP

v) Further in line 5, 6, & the value of 'A' is added in AC from which B is subtracted. The result is stored in AC

vi) Finally line 7 & 8 the value in AC is divided by value in & result is store AC & later in

④ Zero Address Instr :-

→ In this, the major instr used are Push, POP.

→ These instr correspond to stack related operation. hence, they are implemented in stack-enabled sys.

→ Commands such as SUB, ADD, DIV do not require any addresses (operands).

→ PUSH & POP Instr do maintain single addr var

eg: The pgm to execute $Y=(A-B)/(C+D*E)$ using zero addr instr

S.NO	pgm instr
1	PUSH A
2	PUSH B
3	SUB
4	PUSH D
5	PUSH E
6	MUL
7	PUSHC
8	ADD
9	DIV
10	POP Y.

(b) Explain in detail about the decision making Instruction.

Ans:- The decision making instr are used for selecting the best among the two alternatives.

→ It is used to make certain decision.

→ These instr are generally represented using if stmt along with goto stmt & labels.

→ There are two distinct branches.

(i) conditional branch

(ii) Unconditional branch.

(i) conditional branch:-

=> There are two conditional branch instr in MIPS assembly language that are similar to "if" & "go to" stmts in C

→ Every conditional branch instr compares the values given in an instr & the control shifts the control to new addr

→ based on the comparison result the instr is moved to the next addr location.

→ List of Instr under conditional branch,

Mnemonic	Meaning
beq	branch if equal
bne	branch If not equal
slt	set on less than (signed)
slti	" " " " immediate (signed)
sltiu	" " " " (unsigned)

(ii) Unconditional branch:-

→ It does not requires any comparisons & also there is no need of transfering or shifting the control to new addr given in the instr.

→ It includes jump instr

eg:-

j Exit #goto Exit.

→ List of instr used in unconditional branch

Mnemonic	Meaning
Jr	Jump register
J	Jump
Jal	Jump & link.

Eg :-

(i) ~~bne~~ beq reg1, reg2, L1. => If reg1 & reg2 are equal, then goto labelled L1

(ii) bne reg1, reg2, L2. => if reg1 & reg2 are not equal, then go to stmt labele L2

⇒ Decision making instructions in MIPS architecture are important for

(i) compiling if-then-else into conditional branches
(ii) " looping stmt, such as while loop
(iii) Comparing 2 nos i.e. signed & unsigned comparison

⇒ comparison of signed integers with unsigned integers.

⇒ The binary bit pattern that begins with '1' → rep -ve no & is less that +ve no - begin with '0'.

⇒ Whereas for unsigned integers the pattern that begins with 1 ⇒ is +ve & larger than
" " 0 ⇒ -ve.

⇒ The comparison of signed & unsigned includes slt & slti. for signed . sltu & sltiu for unsigned. integers.

Case/switch statement:

⇒ It provides the programmer with multiple opions or alternatives the pgmmer has to select the best option based on single value.

→ It makes use of jumpaddress table. is a table that stores the addresses of an alternative instr sequences.

↳ In order to select an appropriate sequence.

↳ The pgm must jump to the index of addr of appropriate alternative instr seq..

---

(7) Convert C language Instruction into MIPS Assembly code.

(1) i = j + k;

MIPS code:

```
add i,j,k.  # i = j+k
```

(2) l = i - m;

```
sub l,i,m  # l = i-m
```

(3) $x=(y+z)-(m+n);$

→ where operands x, y, z, m & n are associated to in the reg $s0, $s1, $s2, $s3 & $s4. respectively.

→ To perform add & sub temporary reg are required i.e $t0 & $t1.

```
ADD $t0, $s1, $s2.
ADD $t1, $s3, $s4.
SUB $s0, $t0, $t1.
```

(4) $x=y+A[8]$

$s1 = x
$s2 = y.

Consider temp reg $t0 & reg $s3 for storing the base addr of array.

The array A[8] is in memory, in order to find effective address, the base addr of 'A' is added to nos to select element 8, and result is stored in $t0 in the reg $s3.

```
LW $t0, 8[$s3]    # A[8] store in $t0.
ADD $s1, $s2, $t0.  # $s1=$s2+$t0.
```

(5) A[12] = y + A[8].

y = $s1

```
LW $t0, 32[$s2]   # A[8] from mem loaded in $t0
ADD $t0, $s1, $t0   # $t0 = $s1+$t0.
SW $t0, 48[$s2].   # Now $t0 is stored in $s2. in A[12].
```

(6)
```
if (m==n)
    p=q+r;
else
    p=q-r;
```

$s0=p $s2=r $s4=n.
$s1=q $s3=m

```
MIPS code:- bne $s3, $s4, else
        else:   #goto else if m≠n.
```

if m==n, then perform p=q+r and exit,

```
add $s0, $s1, $s2   # p=q+r (skip if m≠n)
Exit:   #goto exit
```

The mnemonics 'j' (jump) is used to differentiate the conditional branches from unconditional branches.

if m≠n, then perform p=q-r and exit,

```
Else : SUB $s0, $s1, $s2   # p=q-r (skip if m=n)
Exit :
```

(7) while (save[i]==k)
    i+=1;

Given:- var i & k stored in reg $s3 & $s5.
save[] is stored in $s6.

```
LOOP : SLL $t1, $s3, 2     # SLL is the MIPS instr
                             used to shift 2 bits to
                             the left i.e i*4 = i<<2.
```

The label "loop" is ~~used~~ added because, at the end of the loop the user can branch back to the instr.

```
Add $t1, $t1, $s6    # $t1 = i*4 (save[i]=
                       $t1+$s6).
LW $t0, 0($t1)       # $t0 = temp reg
                       $t0 = save[i]
BNE $t0, $s5, exit   # goto exit if save[i]≠k
Add $s3, $s3, 1      # add 1 to i i.e i=i+1
J   Loop             # jump back to the
                       starting of the loop
Exit :
```