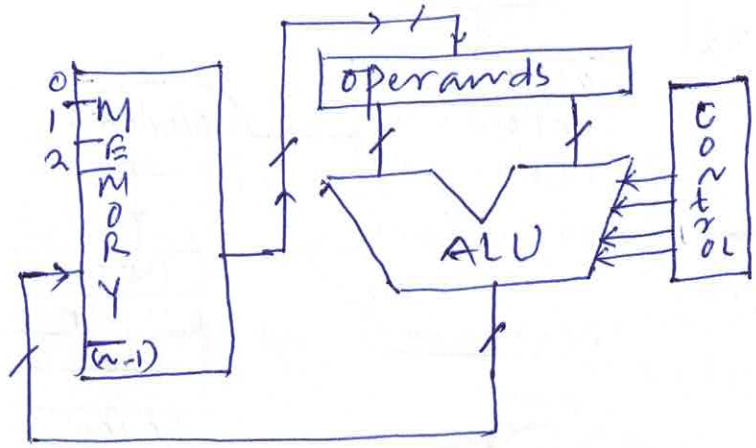
Lecture - 9 | 17th Sep 2019

The language that machine understand
- (Machine language)

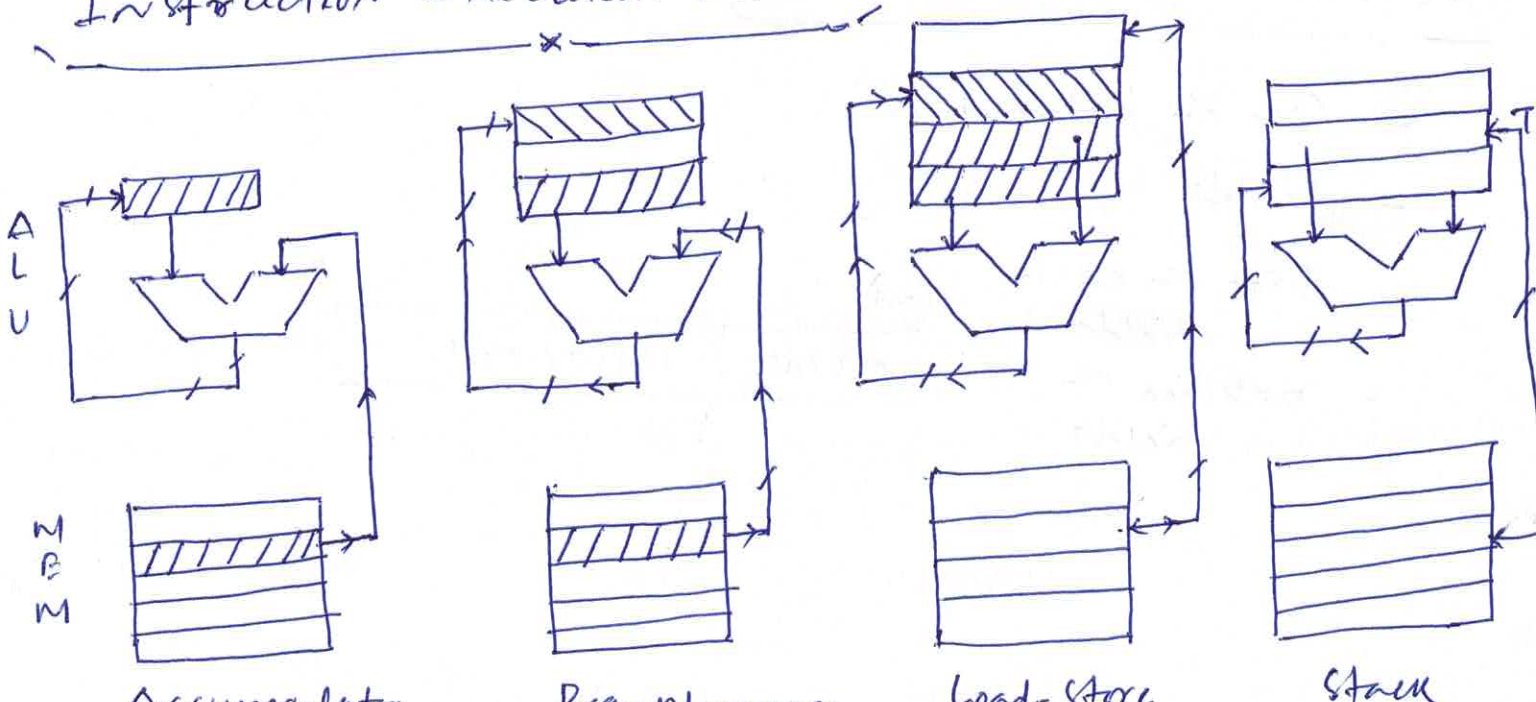
— The Top-level Idea :—



Note :-
- Memory
- operands
- control

Question :- What are the different ways the ALU gets operands ?

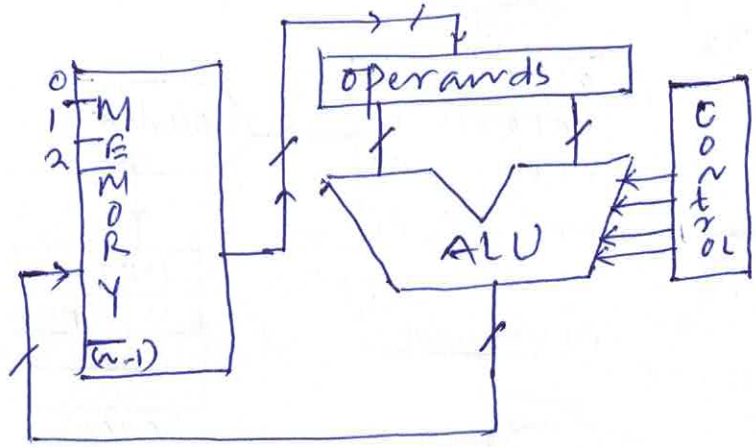
Instruction Execution Model :-



Lecture - 9 | 17th Sep 2019

The language that machine understand
- (Machine language)

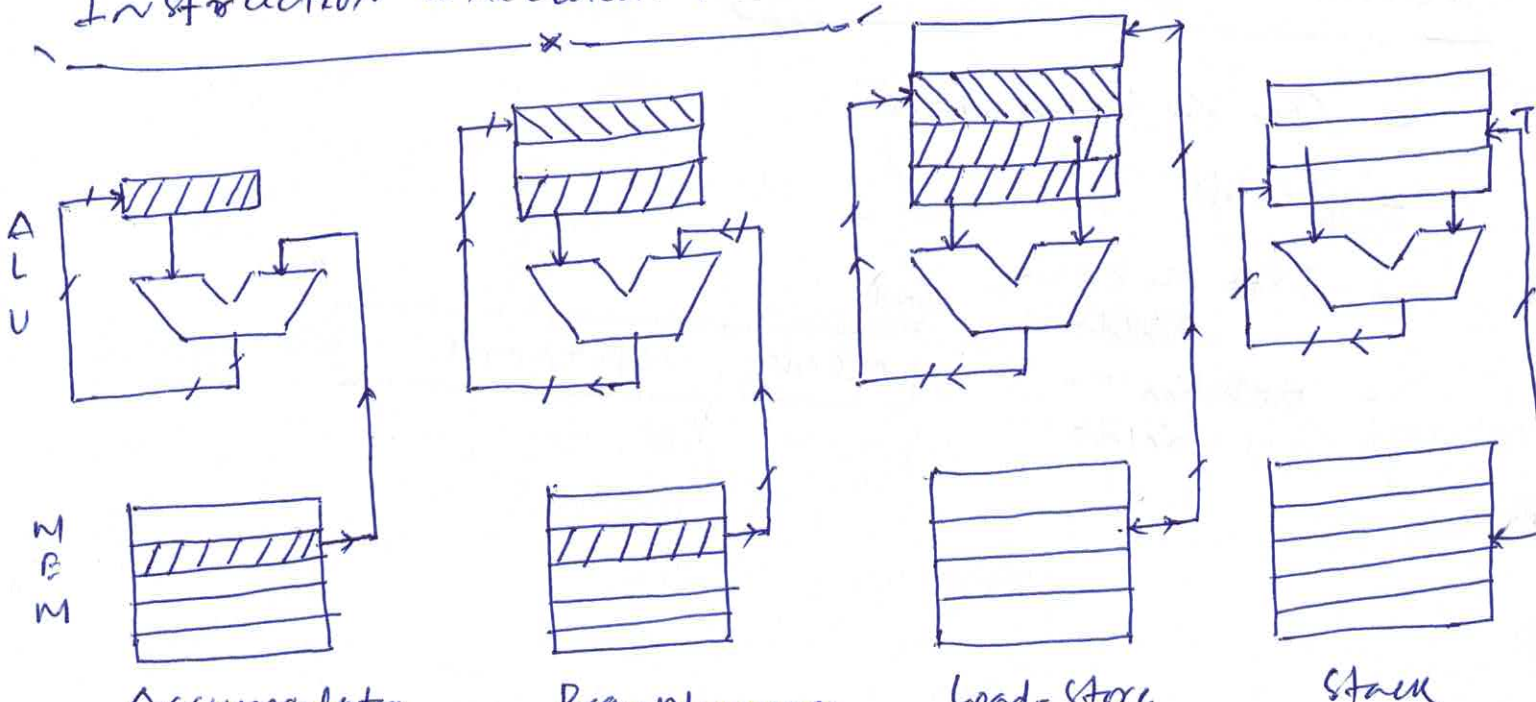
— The Top-level Idea :-



Note :-
- Memory
- operands
- control

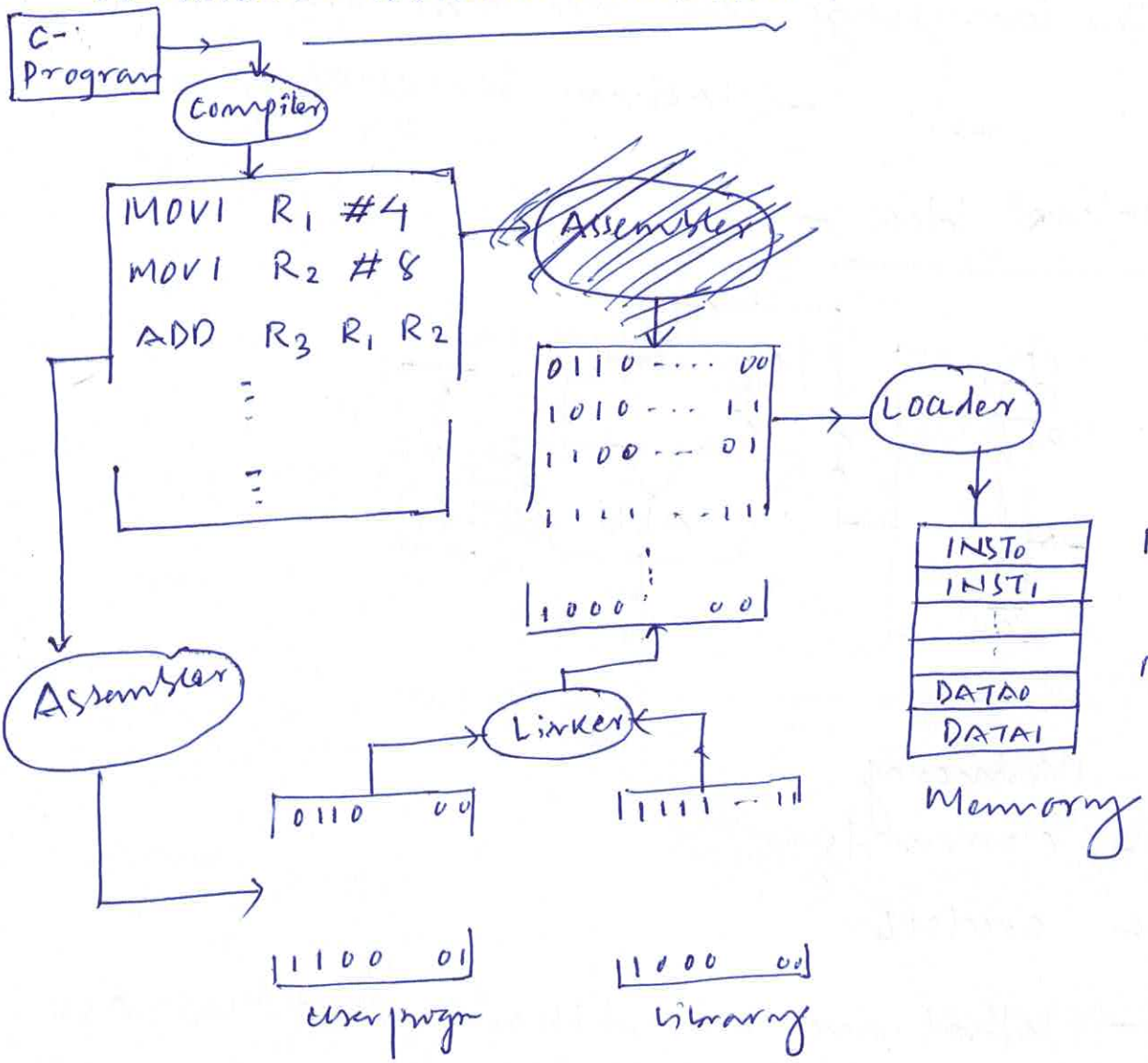
Question :- What are the different ways the ALU gets operands?

Instruction Execution Model :-



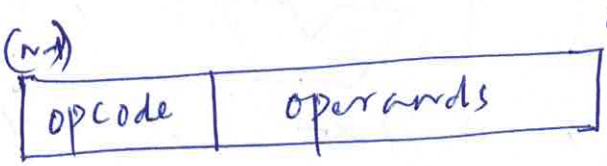
(2)

Hardware - Software Interaction :-



Machine Instruction Format :-

- Size of instruction
- Fields
- Size of each fields
- position of fields



How ~~do~~ decide on opcode & operand? —

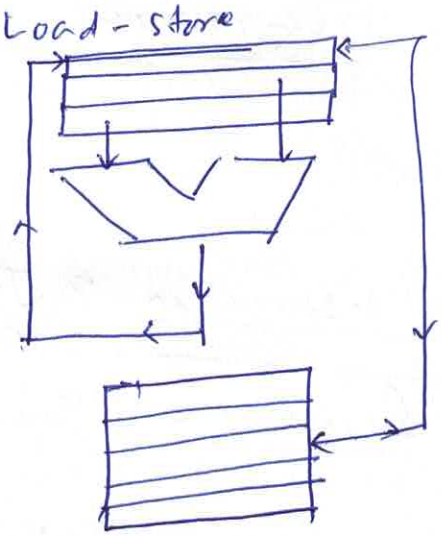
→ ~~Opcode~~ (Opcode) - Total number of instruction

→ (Operand) - Addressing the operand
[Addressing Mode]

Example of MIPS :-

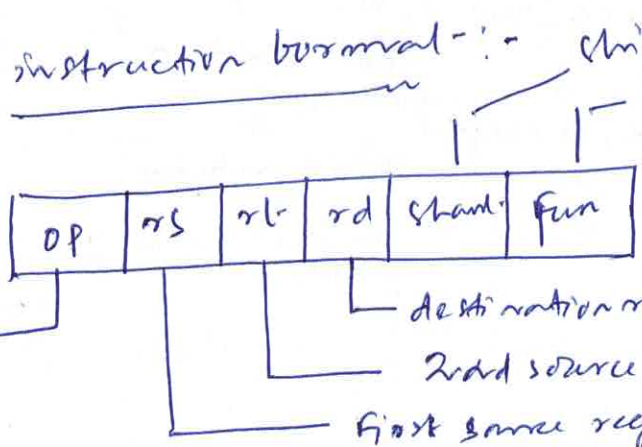
- MIPS is a RISC type of architecture.
- MIPS32 has fixed instruction size of 32-bit (Fixed size)
- Three different formats.

Instruction execution Model -



- Total no. of registers : 32
- Operand type -
  - → two source register.
  - → one destination register
  - → Immediate operand
  - → Memory operand
- Total no. of instruction :- ?

R-type instruction format :-

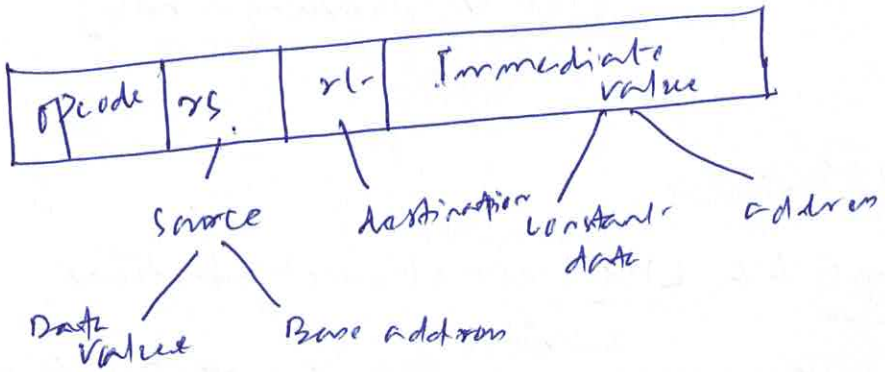


Arithmetic & Logic operation

④ I-type (Immediate instruction):

- Immediate arithmetic operation
- Memory read & write operation (Load & Store).

Immediate & Load store instruction



addi rs, rt, 108

rt ← rs + 108

lw rs, rt, 32

rt ← 32(rs)

rt ← [[rs] + 32]

rt ← MEM[[rs] + 32]

Control instruction & J-type format:

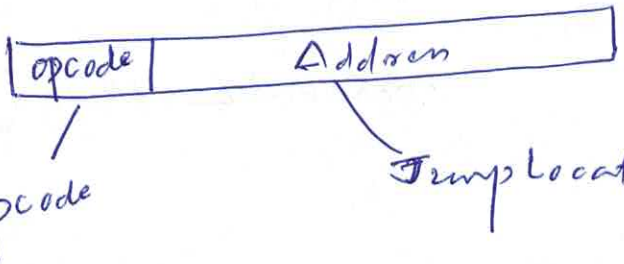
Scenario - Conditional statement - if-else

- Loop - for, while
- Unconditional control switch-case, goto, Jump

- procedure call — call & return
- Interrupt — trap, return.

- J-type format :-

Jump Location (Unconditional Jump)



Conditional branch/Jump

branch on equal

beq rs rt IMD if (rs == rt):
PC ← [PC]+IW

branch not equal

bne rs, rt, IW if (rs != rt):
PC ← [PC] + IW

Addressing Mode :-

- Immediate addressing mode

op | rs | rt | immediate

- Register addressing mode

op | rs | rt | rd | — | fun

→ Register

- Base addressing (Displacement addressing)

op | rs | rt | address Mem[[rs] + address]

⑥

- PC-relative addressing -

op | rs | rd | address Mem[[PC] + address]

operand ← MEM[[PC] + addr]

- pseudo direct addressing

op | Address (26 bit) address ← PC[31,30,29,28] . Whit-2 (Address

address [PC (program counter)
MAR (Memory addr reg

- Register Direct-addressing -

op | rs | ——— | function

PC ← [rs]

address ← [rs] MAR ← [rs]

ARM addressing mode (Similar to MIPS)

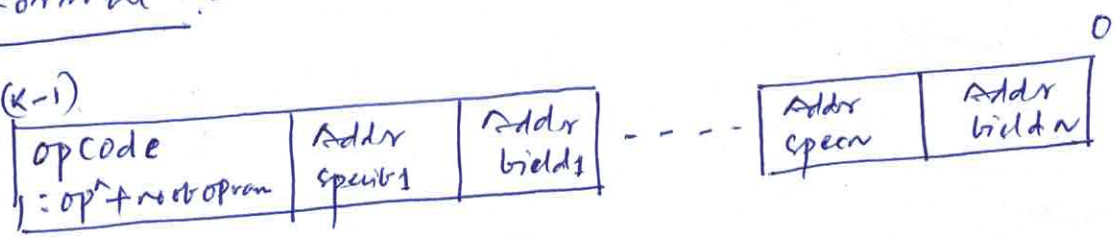
86 addressing mode (variable instruction size).

- Limitation of fixed instruction size

X86 - instruction format :-

- Variable instruction size
- Large number of instruction format (more than three)

Format :-

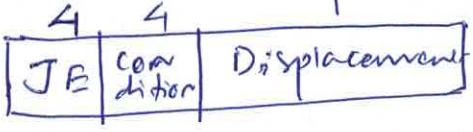


Example :-

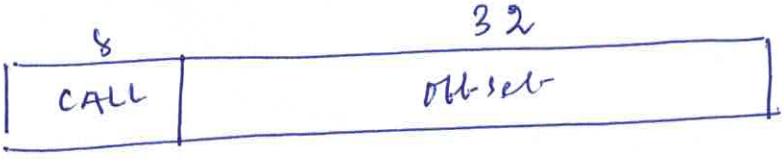
Control Instruction :-

EIP (Extended Instruction pointer (PC))

- JE EIP + displacement-

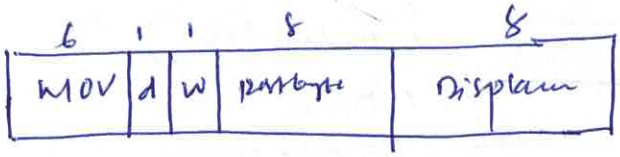


- CALL

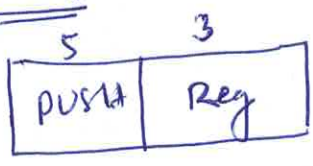


- Data transfer

MOV EBX, [EDI+45]



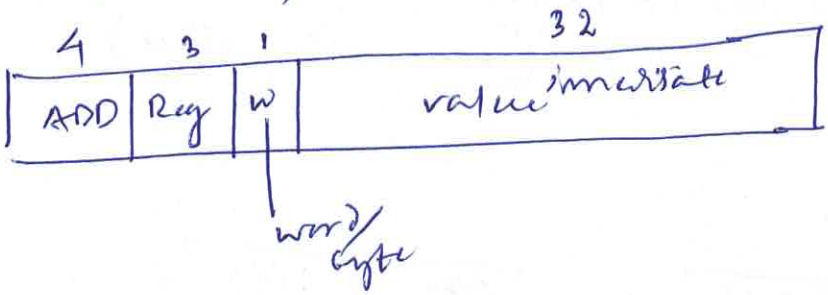
- Stack



(8)

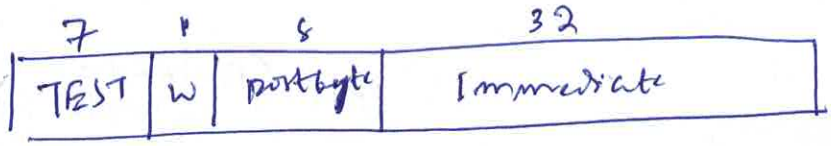
Arithmetic

ADD EAX, #6760



Flag control

TEST EDX, #40



~~Registers~~ Register Set :-

EAX 31 ——— 0

ECX
EDX
EBX
ESP
EBP
ESI
EDI

CS 15 ——— 0 (code segment)
SS (stack segment)
DS (Data seg1)
ES (Data seg2)
FS (Data seg3)
GS (Data seg4)

EIP 31 ——— 0

FLAGS 31 ——— 0