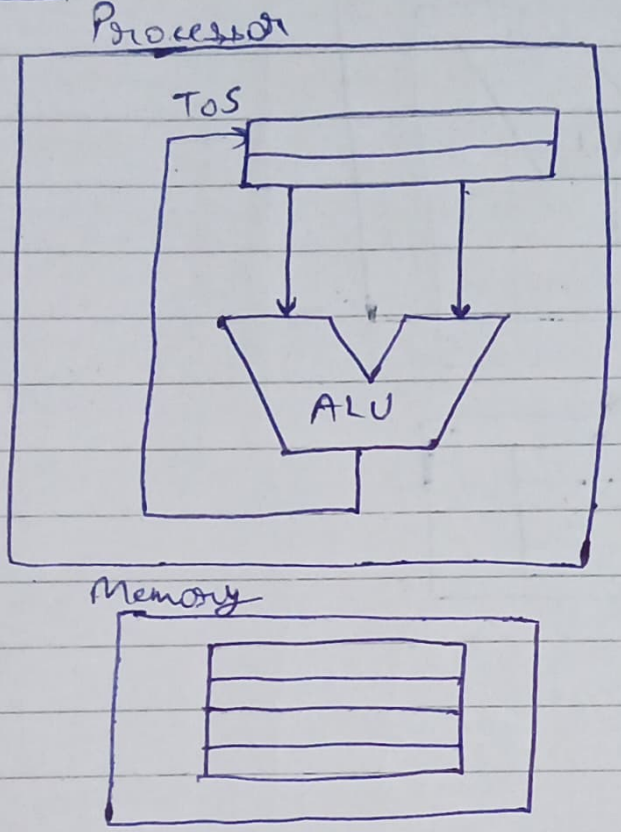
Computer Architecture

(1) Hennessy & Patterson - H/S
(2) P & H - a quantitative approach
5th edition

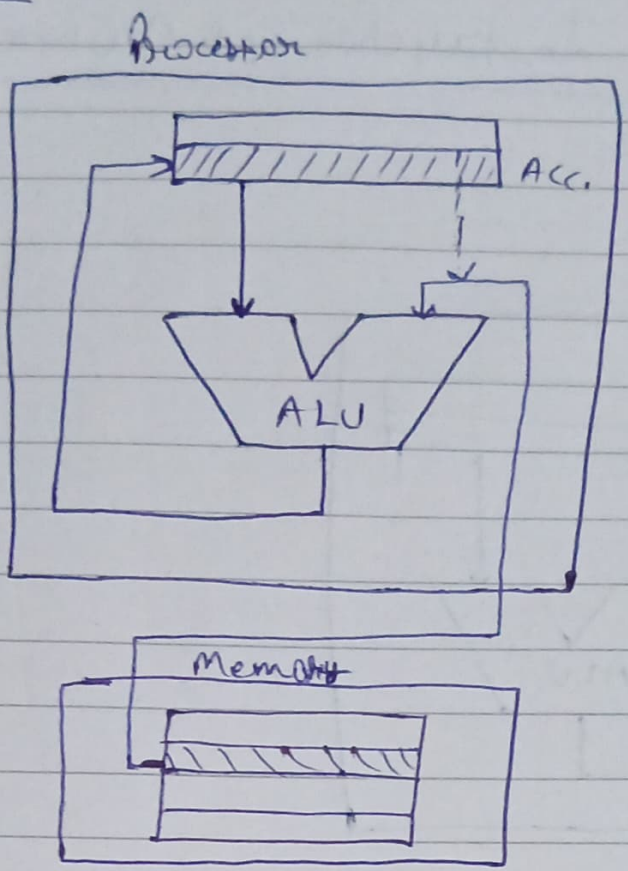
Classifying Instruction Set Architecture

(a) Stack



Push A
Push B
ADD
Pop C

(b) Accumulator

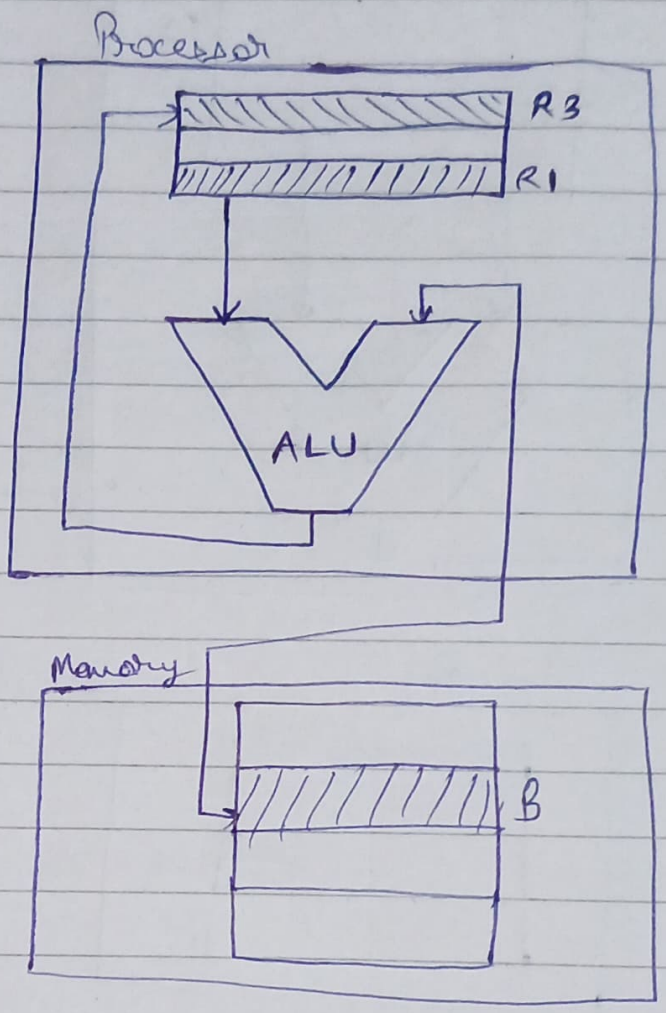


Load A

Add B

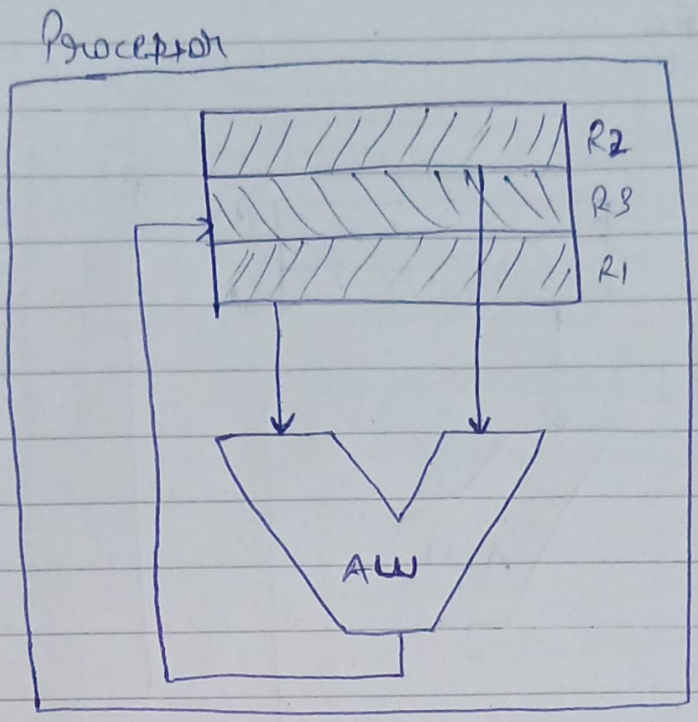
Store C

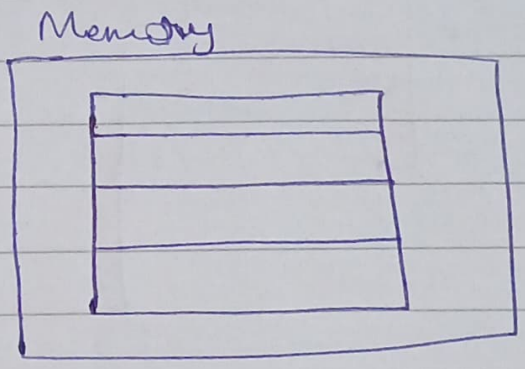
(c) Register - Memory



Load R1, A
Add R3, R1, B
Store R3, C

(d)





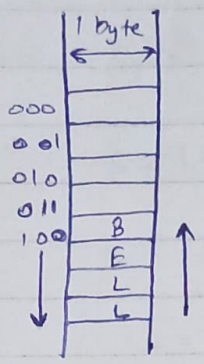
Load R1, A
Load R2, B
Add R3, R1, R2
Store R3, C

Appendix A

Memory Addressing

How memory addresses are specified? → We have addressing modes

" " " " interpreted? – Byte Ordering



BELL

LLEB

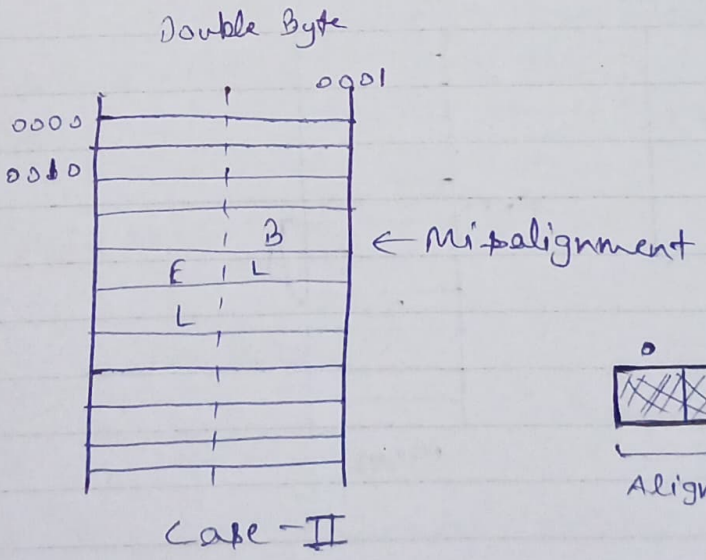
Case I

Little Endian

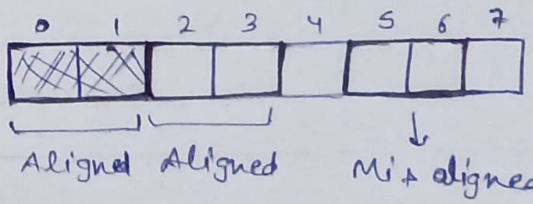
Put the initial address at least significant position → LLEB

Big Endian

Put the initial address at most significant position → BELL



Case – II



Addressing Modes

1. Register Addressing Mode

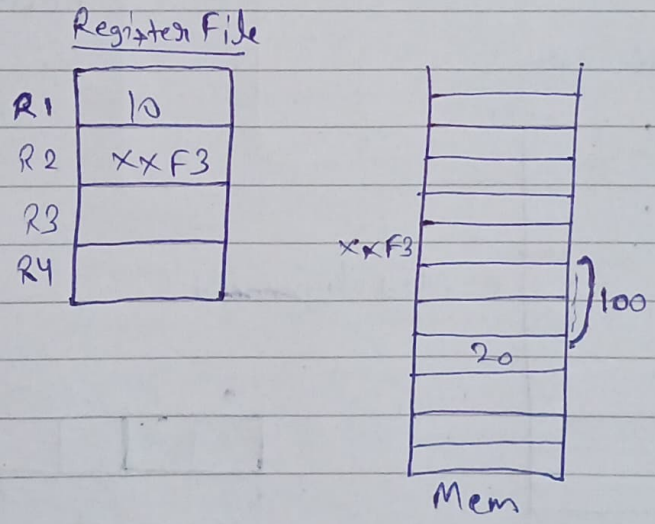
Add R1, R2 $\quad\quad$ R1 ⟵ R1+R2

2. Immediate A.M.

Add R1, #3 $\quad\quad$ R1 ⟵ R1+3

3. Displacement A.M.

Add R1, 100(R2) $\quad\quad$ R1 ⟵ R1+Mem[100+R2]



4. Indexed A.M.

Add R1, (R_x + R2) $\quad\quad$ R1 ⟵ R1 + Mem[R_x + R1]

5. Direct A.M.

Add R1, (1000) R1 ← R1 + Mem[1000]

6. Indirect A.M.

Add R1, @(R2)

7. Auto-Increment/Decrement A.M. | 8. Scaled A.M

MIPS

x86
ARM

Operations performed by processor:

1. Branch (conditional, Uncond.)
2. Load & store
3. ALU operation
4. Control "
5. Floating pt. "

a = b + c

add a, b, c
(a: result; b, c: input operands)

MIPS architecture :

1. It has 32 registers,
2. " " 30 bit memory address

Registers : $s0 - $s7
$t0 - $t9

add $s0, $s1, $s2

$f = (g+h) - (i+j)$

$\$s0$ $\$s1$ $\$s2$ $\$s3$

```
add $t0, $s0, $s1
add $t1, $s2, $s3
sub $t2, $t0, $t1
```

spilling of registers

MIPS Instructions

ADD, SUB, AND, OR, NOT

LOAD half word (16 bit), lw → load word (32 bit)
sw → store "

B.A

$g = h + A[8]$

$\$s1$ $\$s2$ $\$s3$

offset

```
lw  $t0, 32($s3)
add $s1, $s2, $t0
sw  $s1, 48($s3)
```

↓

Data is stored at A[12] now

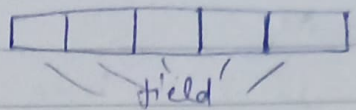
```
addi $s0, $s1, 3   ← immediate addressing
```

↑

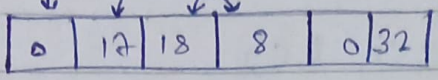
$a = a + 3$

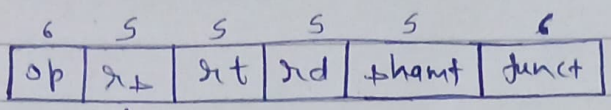
add $s0, $s1, $s2 → 5 bit

addi $s0, $s1, #300 → 8 bit

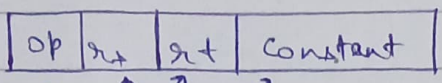


add $t0, $s0, $s1





} - R-type



} I-type

lw $t0, 32 ($s3)

Logical Instructions

and
andi
or
ori
sll → shift logical left
slr → " . " right

Control Instructions

1. beq r1, r2, L1 → cond^n r1, r2 is equal, then branch to label L1.
2. bnq, r1, r2, L2 → cond^n r1 ≠ r2, then branch to label L2.
3. j address → relative address

~~Practice~~

```
if (i==j)          bne $s3, $s4, else
   f = g+h;        # add $s2, $s1, $s2
else
   f = g-h;        # sub, $s2, $s1, $s2
```

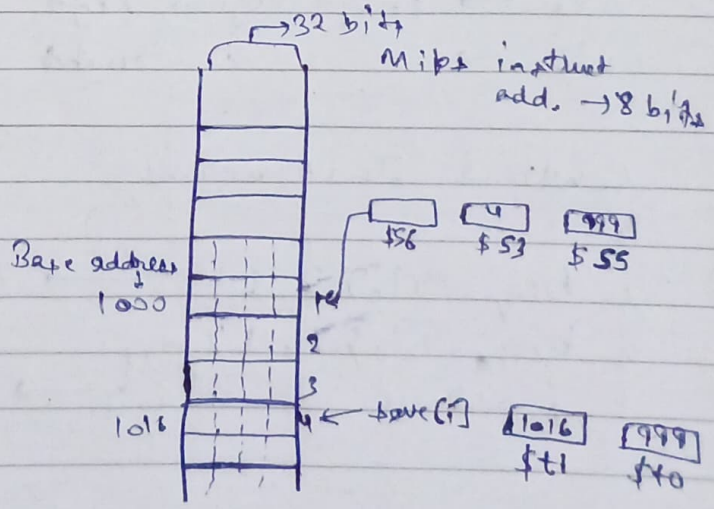
Assembly

```
        bne  $s3, $s4, else
        add  $s0, $s1, $s2
          j Exit
else :  sub  $s0, $s1, $s2
Exit:
```

```
        beq  $s3, $s4, else
        sub  $s0, $s1, $s2
          j Exit
else : add  $s0, $s1, $s2
Exit:
```

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

```
Loop:
    sll  $t1, $s3, 2
    add  $t1, $t1, $s6
    lw   $t3, 0($t1)
    bne  $t0, $s5, Exit
    addi $s3, $s3, i
Exit:
```



```
while (j==k) {
    i+=1
```

i → $s0
j → $s1
k → $s2

```
Loop:
    bne $s1, $s2, Exit
    addi $s0, $s0, 1
    j Loop
Exit:
```

jump

```
while (save[i] < k)
    i+=1
```

i → $s3
k → $s5
save → $s6

```
Loop:
    sll $t1, $s3, 2
    add $t1, $t1, $s6
    lw $t0, 0($t1)
    add $t3, $zero, $zero
    slt $t3, $t0, $s5
    beq $t3, $zero, Exit
    addi $s3, $s3, 1
    j Loop
Exit:
```

false = 0 ← $t3 → True = 1

$a0 – $a3 → argument registers

$v0 – $v1

$ra → return address

return value registers

jal → jump & link, jal add
jr → jump & register, jr $ra

$sp → stack pointer

slt $t0, $t1, $t2
set on less than

if $t1 < $t2
$t0 = 1

$zero

```
sum (int a, int b)
{
    c = a+b;
    return c;
}
```

```
int leaf_example (int g, int h, int i, int j)
{
    int f;
    f = (g+h) - (i+j);
    return f;
}
```

($a0 → g, $a1 → h, $a2 → i, $a3 → j)

Write the compiled MIPS code.

```
add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
jal Procedure address
addi $sp, $sp, -12
sw   $t0, 0($sp)
sw   $t1, 4($sp)
sw   $s0, 8($sp)
add  $v0, $s0, $zero
lw   $s0, 8($sp)
lw   $t1, 4($sp)
lw   $t0, 0($sp)
addi $sp, $sp, 12
jr $ra
```

PC
PC+4
PC+4 = 100
↓
if initial add. is 100

$a0 - $a3 - argument registers
$v0 - $v1 - return value registers
$ra - return address registers

↓
PC+4

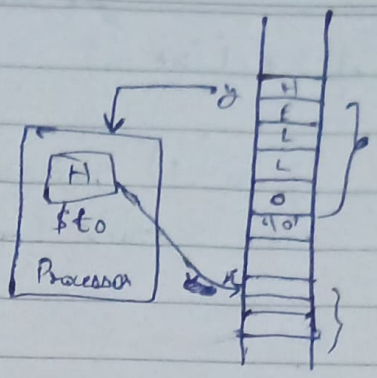
```
                    → $a0
int fact (int n)
 {
     if (n < 1) return 1;
     else return (n * fact(n-1))
}
```

```
fact : addi $sp, $sp, -8
       sw   $ra, 0($sp)
       sw   $a0, 4($sp)
       slti $t0, $a0, 1
       beq  $t0, $zero, L1
       addi $v0, $zero, 1
       addi $sp, $sp, 8
       jr   $ra

L1:  addi $a0, $a0, -1
     jal fact
     lw $a0 4($sp)
     lw $ra 0($sp)
     addi $sp, $sp, 8
     mul $v0, $a0, $v0
     jr $ra
```

```
strcpy (char x[], char y[]) {
  int i;
  i=0;
  while((x[i]=y[i])!='\0')
        i+=1;
}
```

Ans

$a0 - y
$a1 - x
$s0 - i

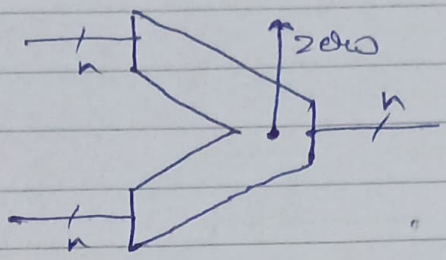



```
Exit: lw   $s0, 0($sp)
      addi $sp, $sp, 4
      jr   $ra

      addi $sp, $sp, -4
      sw   $s0, 0($sp)

      add  $s0, $zero, $zero
while: add  $t1, $s0, $a0
       lb   $t2, 0($t1)

       add  $t3, $s0, $a1
       sb   $t2, 0($t3)
       beq  $t2, $zero, Exit
       addi $s0, $s0, 1
       j while
```

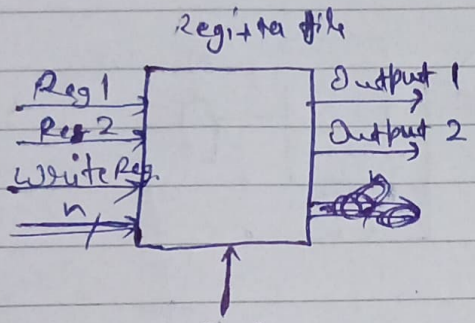
Unit-2

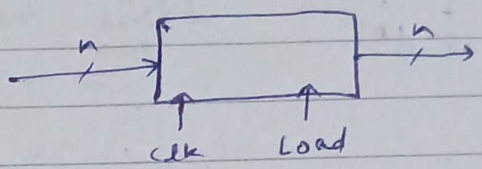
Processor Design

MIPS
32

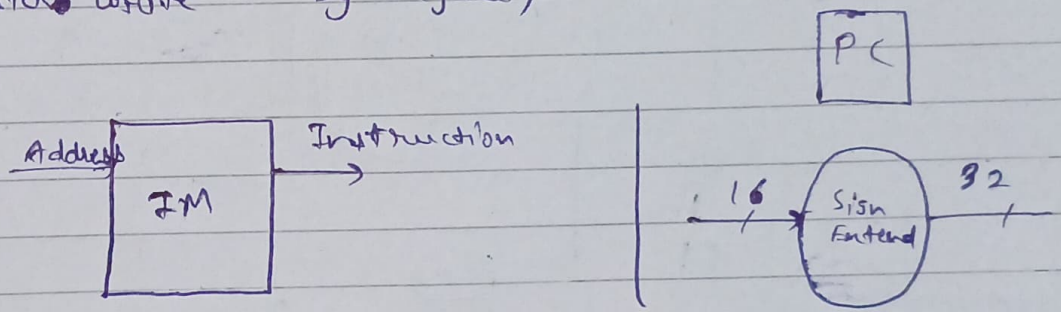


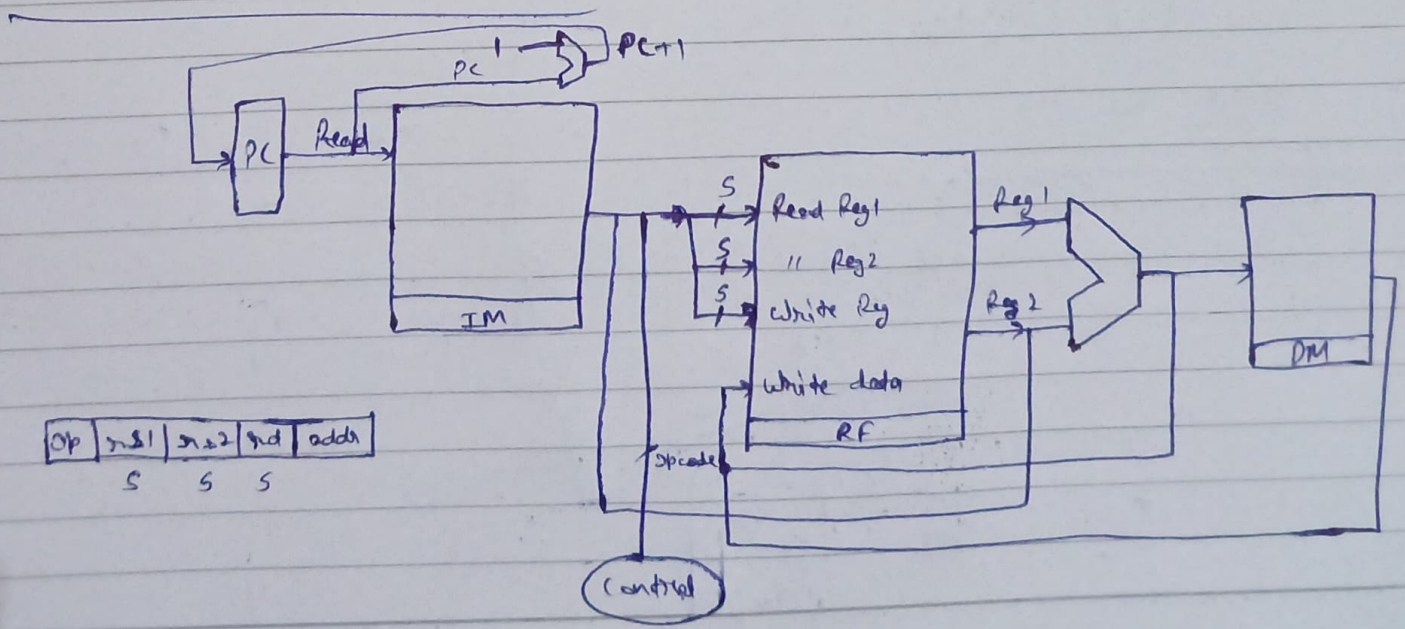
- ALU
- Registers
- control unit
- Memory
- Bus
- Clock
- I/O
- Decoder
- Multiplexer



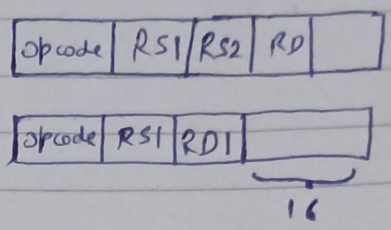


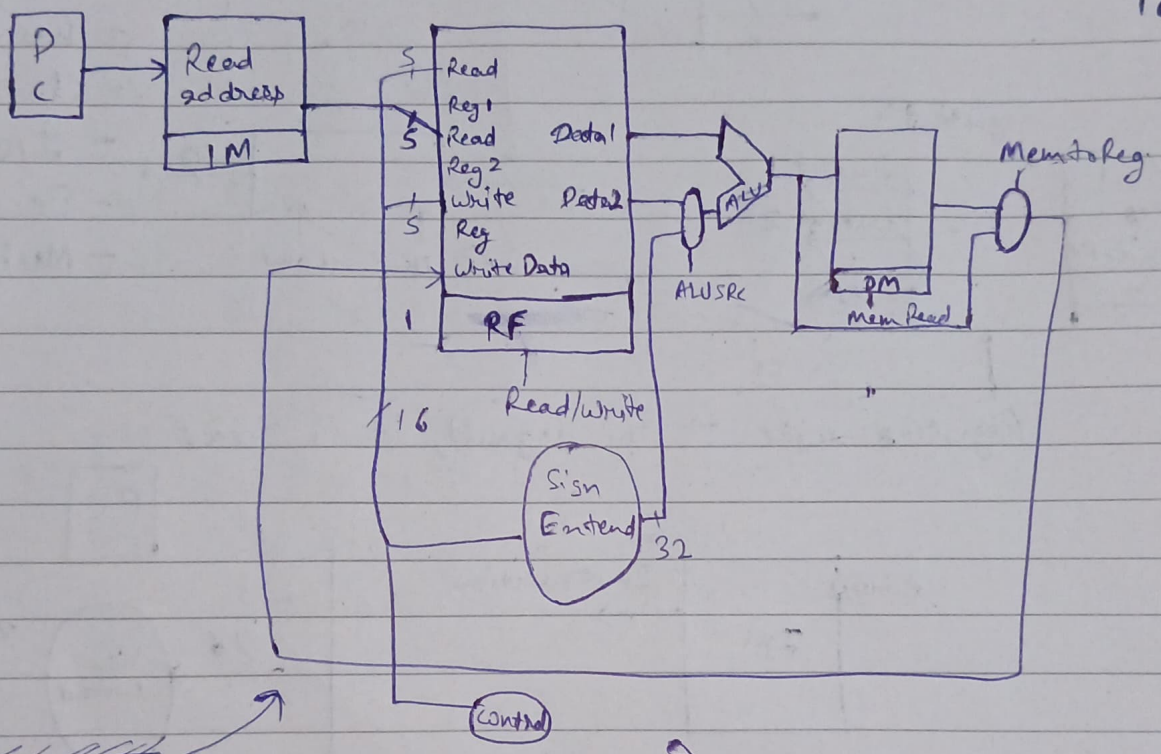
Register write → By default, it is read.





lw $t0, offset($t0)



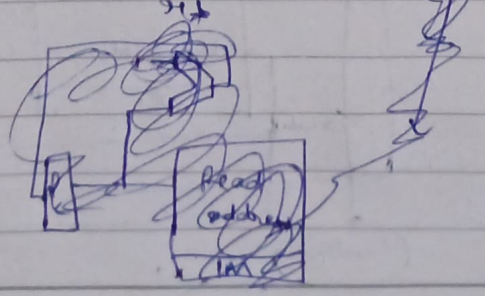


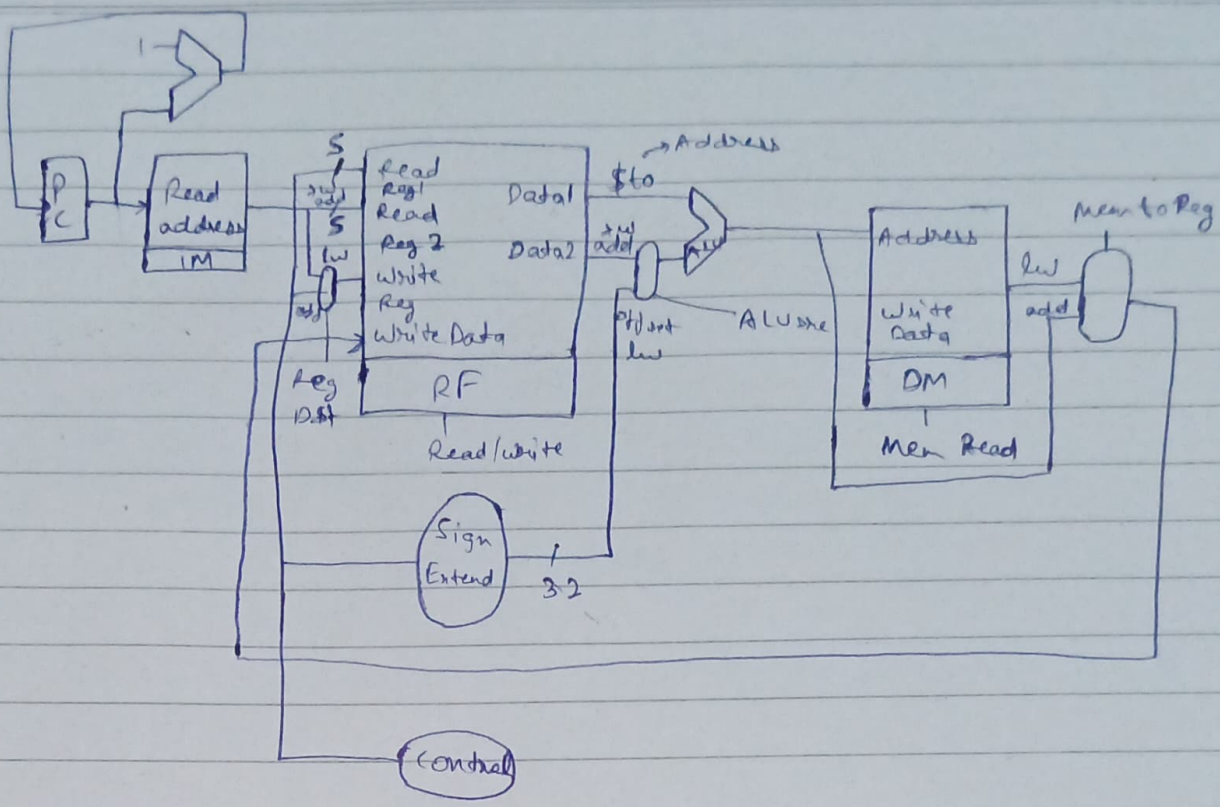
add $s0, $t0, $t1

opcode rd rs rt

lw $s0, offset ($t0)

rt rs




1
PC
Read address
IM
5
5
lw
Read Reg 1
Read Reg 2
Write Reg
Write Data
Data1
Data2
RF
Read/write
Reg Dst
$to
Address
add
ALUSrc
Branch
lw
Address
Write Data
DM
Mem Read
lw
add
MemtoReg
Sign Extend
32
Control

for add, sw & lw