MIPS ARCHITECTURE

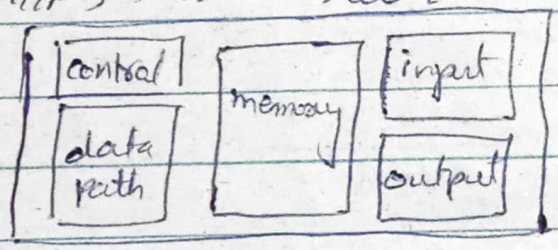
→ It is an cisc architecture like Intel or any other architecture.

→ MIPS Architecture has total 32-registers.

↳ Register #01 is reserved for assembler

↳ Register #26, 27 are reserved for the operating system.

→ MIPS is a load/store architecture



→ main components

⇒ INSTRUCTIONS

→ Simple instr. - all 32-bit wide

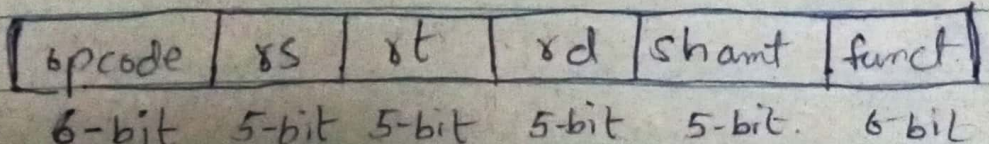
→ Very structured - no unecssary baggage.

→ Only three instr. format.

↳ R-TYPE ↳ I-TYPE ↳ J-TYPE

1) R-TYPE INSTRUCTION:

Instruction-format:



rs : first src operand
rt : second src operand
rd : destination reg
shamt : shift ammount
funct : identifies the diff: type of } both are optional only written if needed.

All arithemetic and logical operations are done in R-type instr.

Arithemetic :

add:	add $s1, $s2, $s3	→ $s1 = $s2 + $s3
Subtract:	sub $s1, $s2, $s3	→ $s1 = $s2 - $s3
~~add immediate:~~	~~addi $s1, $s2, 100~~	~~→ $s1 = $s2 + 100~~

Logical :

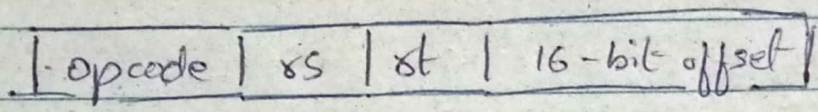
and	and $s1, $2, $s3	$s1 = $s2 & $s3
or	or $s1, $s2, $s3	$s1 = $s2 | $s3
nor	nor " " "	" = " "
and immediate	andi $s1, $s2, 100	" = " & 100
or "	ori " " "	" = " | 1 "
shift left logical	sll $s1, $s2, 10	$s1 = $s2 << 10
" right "	srl " " "	$s1 = $s2 >> 10

add 100 into $S2 and then go to that location at $s1 and save that value in

Data Transfer:

load word	lw $s1, 100($s2)	$S1 = Memory[$S2+100]
store word	sw $s1, 100($s2)	Memory[$S2+100] = $S1

2) I - TYPE INSTRUCTION:

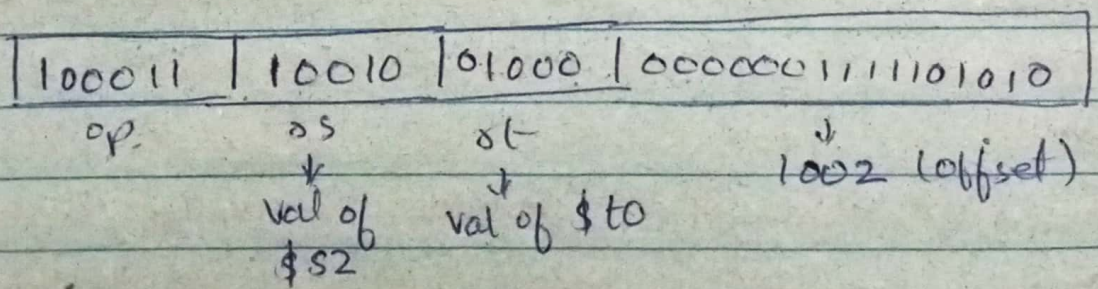


→ The load-word & store word instruction uses I-TYPE instructions only.

→ Commands with immediate values are also I-TYPE.

→ Conditional Branch ~~not equal~~ is also I-type & adds also.

lw $t0, 1002($s2)



Control: Conditional Branch.

- Branch equal (beq)
- Branch not-eq (bne)

beq $t0, $t1, Label

↑ if both val equal then jmp to label (jmp means we are telling that how many words we want to skip from current program counter)

000100	01000	01001	00- - -11001
op	rs	rt	offset
	t1	t0	label

→ offset no. of words main hoga hamesha

3) I - TYPE INSTRUCTION:

opcode	address
6-bit	26-bit

→ J-type are used by unconditional branch instructions.

⇒ j Label # addr. label = 100

convert byte to word 100 byte = 25 word

000010	00- - - -11001
opc	26-bit offset

if (i != j)
  h = i+j

else
  h = i-j

beq $s4, $s5, Lab1
add $s3, $s4, $s5
j Lab2
Lab1: sub $s3, $s4, $s5
Lab2: ----

2) jr $s1 ⇒ go to addr contained in $s1

PHASE OF EXECUTION OF MIPS INSTRUCTION

- Every instruction takes at most 5 clock cycles (phases) as follows:

1) INSTRUCTION FETCH (IF):

→ Program Counter (PC) mein jo content hoga wo ~~memory~~ Instruct. Mem mein dala jayega or current instr ko fetch krega instruct. mem. se.

→ PC ~~doosri inst~~ mein 4 bytes add ho jayenge (bcz each instr. is of 4 bytes) to we next sequential instr. ko point krega.

2) INSTRUCTION DECODE & REGISTER READ (ID):

→ Ab jo bhi instruction mein hai use decode krenge or jo reg mein jo contents hai unhe read krenge.

→ ~~Arg~~ Agr offset field mein sign extension req hai to we bhi hogi.

3) EXECUTION (EX):

Ab ALU un operands par kaam karega jo humein pichle steps mein obtain hue the, depending on the instr. type.

i- Register-Register ALU Instruct.:

ALU executes the specified operation on the operands read from Register File (RF).

ii- Register - Immediate ALU Instruct:

ALU executes the specified operation on the 1st operand read from the RF & the sign-extended immediate operand.

iii- Memory Reference:

ALU adds the base reg & offset to calc. the effective address.

iv- Conditional Branches:

Compare the two register read from RF & compute the possible branch target address by adding the sign-extended offset to the incremented PC.

4) MEMORY ACESS (MIE):

→ Load instr. ko access chahiye hote hai data read krne k liye memory se using the effective addr.

→ Store instr. ko acess chahiye ki a write krne k liye u a address to write data from source reg.

→ Conditional branches can update the content of the PC with the branch target address, if the conditional test returns true.

5) WRITE BACK CYCLE (WB)

→ Load instr. write the data read from memory in destination reg of the RF.

→ ALU instr. write the ALU results into the destination reg of the RF.