Procedure Calling

→ parameter placing in registers
→ transfer control to procedure
→ acquire storage for procedure
→ perform procedure's operations
→ place result in register for caller
→ return to place of call

Few Registers

parameter passing → $a0 - $a3
returning results → $v0 - $v1
return address → $ra

Others $t0 - $t7, $s0 - $s7
$gp → global pointer for static data
$sp → stack pointer

Procedure calling in MIPS

jal procedureAddress
↓
Jump-And-Link

link → put the adddress of the following instouction in $ra
jump → jumps to target address

Returning from procedures in MIPS

jr $ra
↓
Jump Register

So, what we do is, (for procedures)

→ put parameters in $a0 - $a3 (caller)

→ jal X to call procedure, X (caller)

→ X computes results and put them in $v0 - $v1 (callee)

→ X then return control to the caller using jr $ra

For all these work to do, procedure calling & jump/branch, we need to have a register to hold the address of the current instruction being executed. This register

PC → program counter

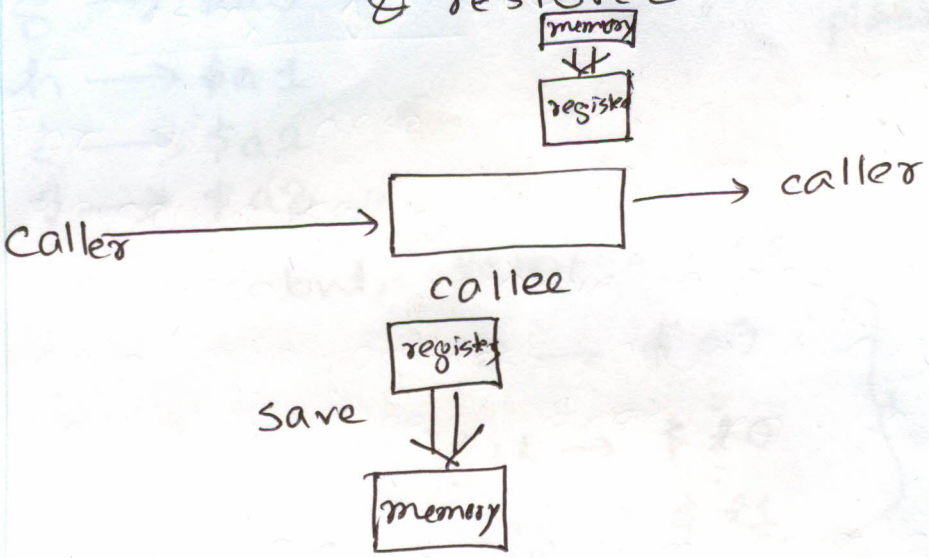
also known as IAR → Instruction Address Register.

jal X → $ra ← PC+4
PC ← X

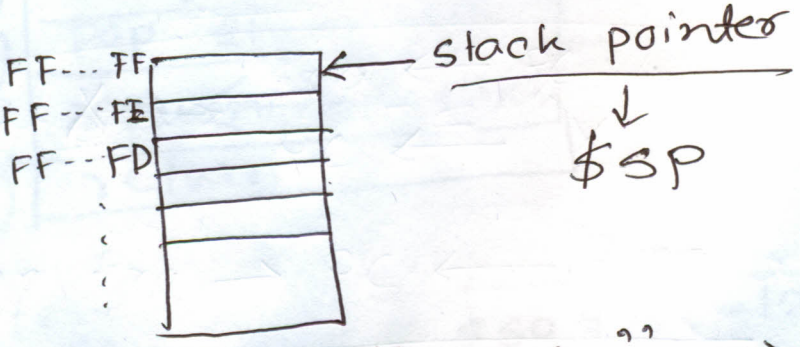
jr $ra → PC ← $ra

⊛ Restoring Registers :

- procedure may need other registers apart from four parameter registers & two return value registers
- if other registers are used, their values must be saved & restored



- for this purpose we use stack (last-in-first-out) memory



"push" → $sp--;

"pop" → $sp++;

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

g → $a0
h → $a1
i → $a2
j → $a3

place result in $v0

but, temporaries,

f → $s0
g+h → $t0
i+j → $t1

} 3 - push to stack

1. sp ← sp - (3×4)
2. push them all
3. compute
4. pop them all
5. ~~return~~ sp + (3×4)
6. return

test:

```
1) add  $sp, $sp, -12
2) sw   $t1, 8($sp)
   sw   $t0, 4($sp)
   sw   $s0, 0($sp)
3) add  $t0, $a0, $a1
   add  $t1, $a2, $a3
   add  $s0, $t0, $t1
   add  $v0, $s0, $zero
```

not completed

(*) Recap:

- Steps in procedure calling:

1. parameter register, $a0 - $a3
2. goto procedure, jal procLabel
3. Acquire storage; stack,
4. procedure operations
5. return result register, $v0 - $v1
6. return to place of call, jr $ra

① ② jal test

C - code

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

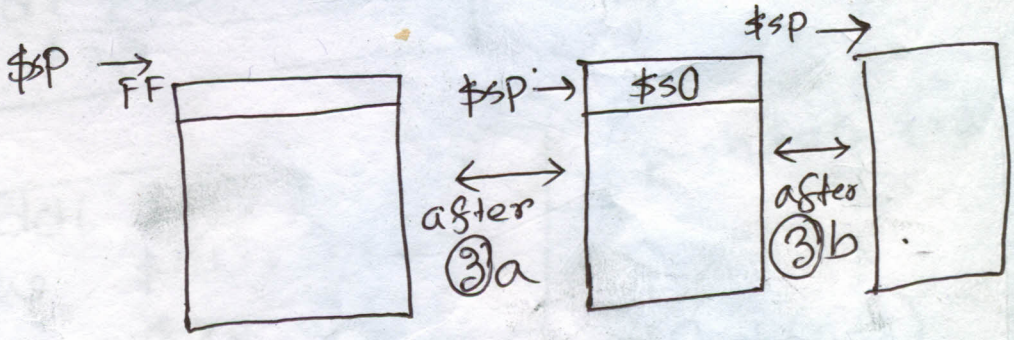
3

g ↔ $a0
⋮
j ↔ $a3

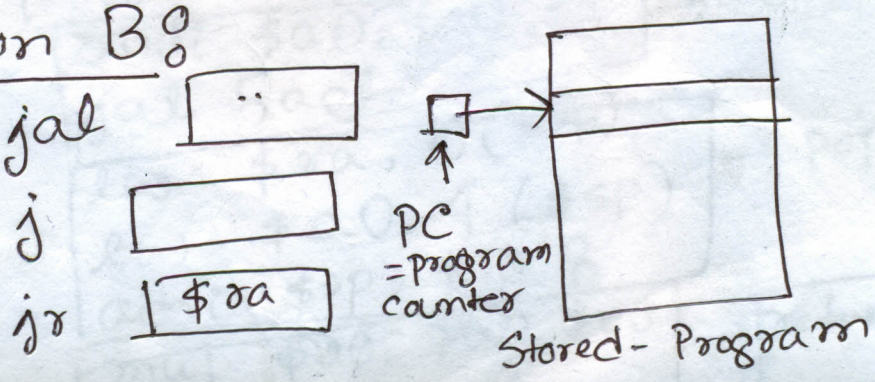
push to stack

f ↔ $s0
g+h ↔ $t0
i+j ↔ $t1

```
test:
③a  addi $sp, $sp, -4
    sw   $s0, 0($sp)
④   add  $t0, $a0, $a1
    add  $t1, $a2, $a3
    sub  $s0, $t0, $t1
⑤   add  $v0, $s0, $zero
③b  lw   $s0, 0($sp)
    addi $sp, $sp, 4
⑥   jr   $ra
```



For Section B:



ted Procedure

main

100: A(3)

$ra = 104
$a0 = 3

A

200: B(7)

$ra = 204
$a0 = 7

B

so we must save
$ra (1) & $a0 (2)

C

| n ↔ $a0

(1) (2)

C

```
int fact(int n){
    if(n<2) return 1;
    else return n*fact(n-1);
}
```

n ⟷ $a0

return result ⟷ $v0

3

fact:

```
slti $t0, $a0, 2
beq  $t0, $zero, Else
addi $v0, $zero, 1
jr   $ra
```

if n<2 : t0=1
else: t0=0
if t0 == 0: goto Else
return 1

Else:

```
addi $sp, $sp, -8
sw   $a0, 4($sp)
sw   $ra, 0($sp)
```

push $a0, $ra

```
addi $a0, $a0, -1
jal  fact
```

fact(n-1)

```
lw   $ra, 0($sp)
lw   $a0, 4($sp)
addi $sp, $sp, 8
```

pop $ra, $a0

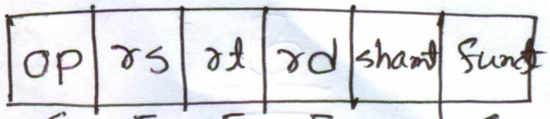
```
mul  $v0, $a0, $v0
jr   $ra
```

return n * fact(n-1)

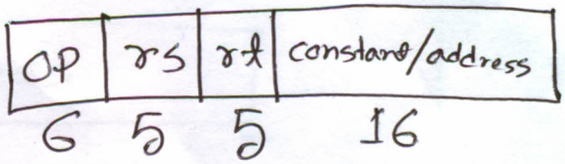
coding Branch Instructions in 32-bit binary

Recap:

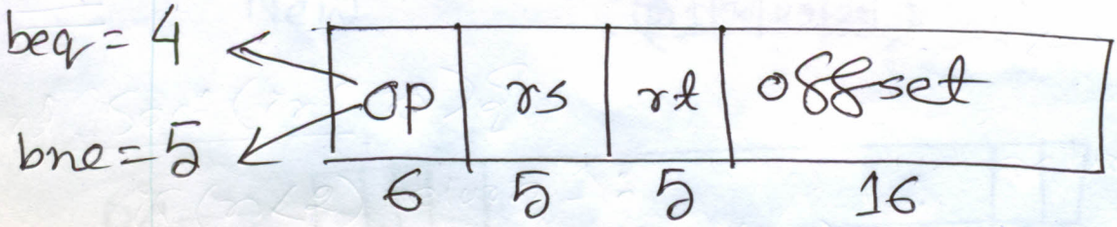
R-format



I-format



→ branch, → beq $r1, $r2, label
→ bne $r1, $r2, label



offset address 16-bit,

X only, 65536 - instructions

$$PC_{New} \leftarrow PC_{Old} + Offset \times 4$$

PC-relative addressing

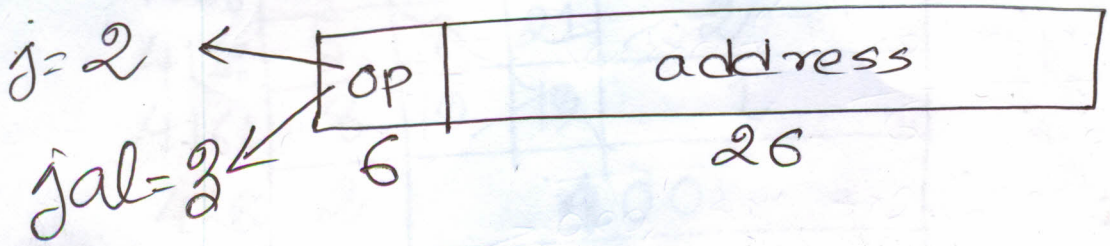
~ PC_{Old} = Next instructions address

branch → loop, if → tend to nearby instructions

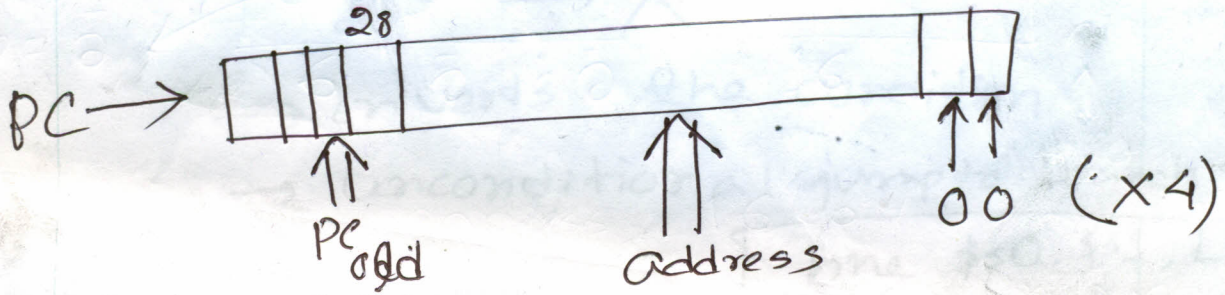
$\frac{1}{2}$ SPEC benchmark branches go to less than 16

→ unconditional jump (j , jal)

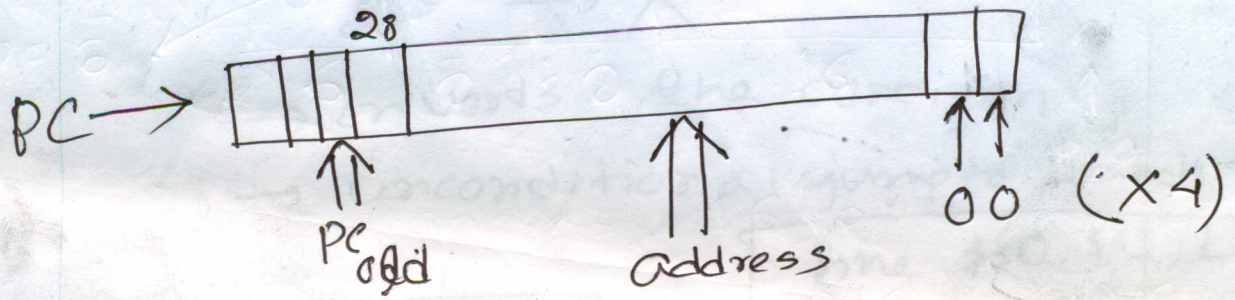
J- type format



$PC_{New} = PC_{Old}(31\,30\,29\,28) : (address \times 4)$



$PC_{New} = PC_{Old}$ (31 | 30 | 29 | 28) : (address × 4)

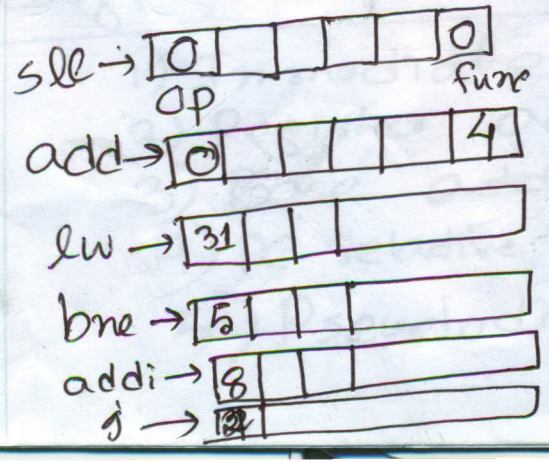


Example:

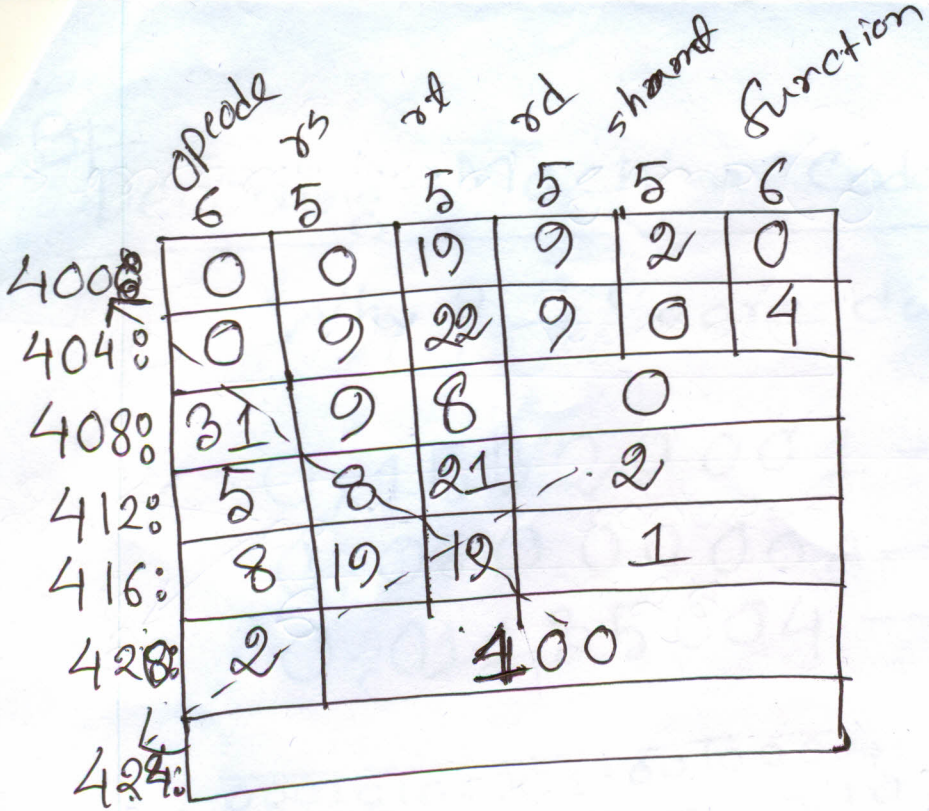
```
while (save[i] == k) i++;
```

i ↔ $s3
k ↔ $s5
save ↔ $s6

```
Loop: sll  $t1, $s3, 2      $t1 ← t1 * 4
      add  $t1, $t1, $s6    t1 ← save + (t1 * 4)
      lw   $t0, 0($t1)
      bne  $t0, $s5, Exit
      addi $s3, $s3, 1
      j Loop
Exit:
```



4000 : ~~sll $t1, $s3, 2~~
4004 : ~~add $t1~~
4008 :
4012 :
4016 :
4020 :
4024 :


opcode
rs
rt
rd
shamt
function
6
5
5
5
5
6
4000:
4004:
4008:
4012:
4016:
4020:
4024:
0 0 19 9 2 0
0 9 22 9 0 4
31 9 8 0
5 8 21 2
8 19 19 1
2 400

(*) Branching far away:

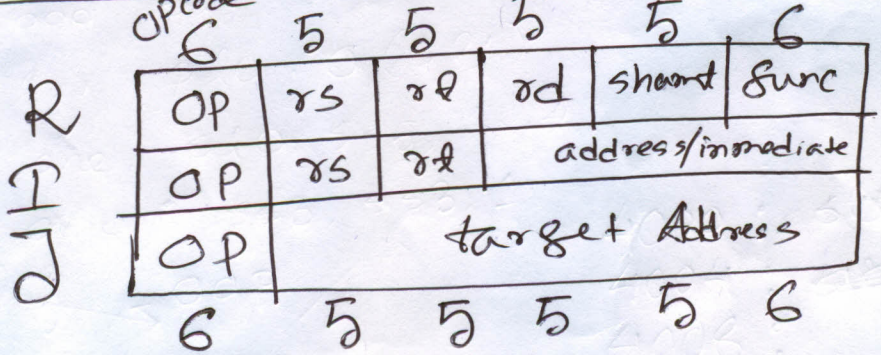
$2^{16} \rightarrow X$

→ inverts the condition

→ unconditional jump to branch target

```
beq  $s0, $s1, L1   <-->   bne $s0, $s1, L2
                           j   L1
                           L2:
```

MIPS instruction format summary:



Addressing Mode Summary:

1) Immediate addressing
2) Register addressing
3) Base addressing
4) PC-relative addressing
5) Pseudodirect addressing

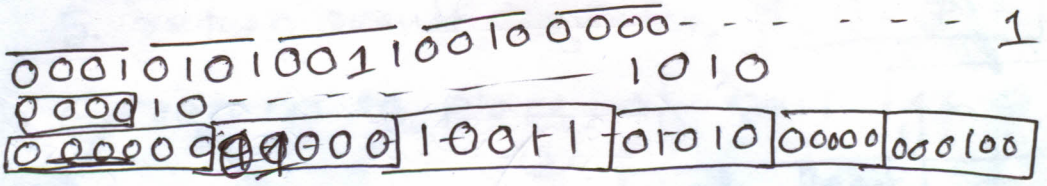
BT Decoding Machine Code:

Why? — "core dump"

0x15320001 → bne $t1, $s2, 52

0x08000000A → j 40

0x01135004 → add $t2, $t0, $s3



40:

44:

48:

52:

44:	5	9	18	1		
48:	2	10				
52:	0	8	19	10	0	4
	6	5	5	5	5	6

bne $t1, $s2, 52

j 40

add $t2, $t0, $s3

0000 0001 0001 0011 0101 0000 0000 0100

0 1 1 3 5 0 0 4