Chapter 6 :: Architecture

Quiz 4 → opcode
→ 16 bits
→ 5 bits

6.9-6.10 MIPS instructions

Digital Design and Computer Architecture

David Money Harris and Sarah L. Harris

High-Level Code Constructs

- `if` statements
- `if/else` statements
- `while` loops
- `for` loops

If Statement

C Code

```
if (i == j)
   f = g + h;

f = f - i;
```

MIPS assembly code

```
# $s0 = f, $s1 = g, $s2 = h
# $s3 = i, $s4 = j
```

beq / bne

by itself

If Statement

C Code

```
if (i == j)
  f = g + h;

f = f - i;
```

MIPS assembly code

```
# $s0 = f, $s1 = g, $s2 = h
# $s3 = i, $s4 = j
    bne $s3, $s4, L1
    add $s0, $s1, $s2

L1: sub $s0, $s0, $s3
```

Assembly tests opposite case (`i != j`) of high-level code (`i == j`)

If/Else Statement

C Code

```
if (i == j)
  f = g + h;
else
  f = f - i;
```

MIPS assembly code

beq / bne
followed by
j that jumps
forward

If/Else Statement

C Code

```
if (i == j)
  f = g + h;
else
  f = f - i;
```

MIPS assembly code

```
# $s0 = f, $s1 = g, $s2 = h
# $s3 = i, $s4 = j
      bne $s3, $s4, L1
      add $s0, $s1, $s2
      j   done
L1:   sub $s0, $s0, $s3
done:
```

(opposite condition)

While Loops

C Code

```
// determines the power
// of x such that 2^x = 128
int pow = 1;
int x   = 0;

while (pow != 128) {
  pow = pow * 2;
  x = x + 1;
}
```

MIPS assembly code

beq / bne followed by a j that jumps backward

Assembly tests for the opposite case (`pow == 128`) of the C code (`pow != 128`).

While Loops

C Code

```
// determines the power
// of x such that 2^x = 128
int pow = 1;
int x   = 0;

while (pow != 128) {
  pow = pow * 2;
  x = x + 1;
}
```

MIPS assembly code

```
# $s0 = pow, $s1 = x

       addi $s0, $0, 1
       add  $s1, $0, $0
       addi $t0, $0, 128
while: beq  $s0, $t0, done
       sll  $s0, $s0, 1
       addi $s1, $s1, 1
       j    while
done:
```

(extra)

repeat the loop

Assembly tests for the opposite case (`pow == 128`) of the C code (`pow != 128`).

For Loops

```
for (initialization; condition; loop operation)
  statement
```

- `initialization:` executes before the loop begins
- `condition:` is tested at the beginning of each iteration
- `loop operation:` executes at the end of each iteration
- `statement:` executes each time the condition is met

For Loops

C Code

```
// add the numbers from 0 to 9
int sum = 0;
int i;

for (i=0; i!=10; i = i+1) {
  sum = sum + i;
}
```

MIPS assembly code

branch followed by jump back

For Loops

C Code

```
// add the numbers from 0 to 9
int sum = 0;
int i;

for (i=0; i!=10; i = i+1) {
  sum = sum + i;
}
```

MIPS assembly code

```
# $s0 = i, $s1 = sum
        addi $s1, $0, 0
        add  $s0, $0, $0
        addi $t0, $0, 10    constant 10
for:    beq  $s0, $t0, done
        add  $s1, $s1, $s0
        addi $s0, $s0, 1
        j    for
done:
```

stop here 9/22/25

Addressing Modes

(or refer to)

How do we address the operands?

- Register Only
- Immediate
- Base Addressing
- PC-Relative
- Pseudo Direct

Addressing Modes

Register Only → r-type

- Operands found in registers
  - **Example:** `add $s0, $t2, $t3`
  - **Example:** `sub $t8, $s1, $0`

Immediate → most i-type

- 16-bit immediate used as an operand
  - **Example:** `addi $s4, $t5, -73`
  - **Example:** `ori  $t3, $t7, 0xFF`

Addressing Modes

Base Addressing

(base + offset)

- Address of operand is:

memory instructions

`base address + sign-extended immediate`

  - **Example:** `lw   $s4, 72($0)`
    - address = `$0 + 72`
  - **Example:** `sw   $t2, -25($t1)`
    - address = `$t1 - 25`

Addressing Modes

program counter

PC-Relative Addressing

branch

instruction addresses

PC → 0x10		beq	$t0, $0, else
PC+4 → 0x14		addi	$v0, $0, 1 #else=3: (0x20-0x14)/4
0x18		addi	$sp, $sp, i
0x1C		jr	$ra
0x20	else:	addi	$a0, $a0, -1
0x24		jal	factorial

1 2 3

Assembly Code

beq $t0, $0, else

(beq $t0, $0, 3)

Field Values

op	rs	rt	imm		
4	8	0	3		
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Addressing Modes

Pseudo-direct Addressing

j-type

instruction address

want to specify address directly

```
0x0040005C          jal    sum
...
0x004000A0   sum:   add    $v0, $a0, $a1
```

JTA → jump target address

JTA ~~0000~~ 0000 0100 0000 0000 0000 1010 00~~00~~ (0x004000A0)

26-bit addr 0000 $\underbrace{00}_{0}\underbrace{00\ 01}_{1}\underbrace{00\ 00}_{0}\underbrace{00\ 00}_{0}\underbrace{00\ 00}_{0}\underbrace{00\ 10}_{2}\underbrace{10\ 00}_{8}$00 (0x0100028)

borrow from PC+4

Field Values

op	imm
3	0x0100028
6 bits	26 bits

Machine Code

op	addr	
000011	00 0001 0000 0000 0000 0010 1000	(0x0C100028)
6 bits	26 bits	

How to Compile & Run a Program