DIGITAL DESIGN
&
COMPUTER ORGANISATION

Dr. Reetinder Sidhu and Dr. Kiran D C
Department of Computer Science and Engineering

PES UNIVERSITY ONLINE

DIGITAL DESIGN & COMPUTER ORGANISATION

Machine Language - 2

Dr. Reetinder Sidhu and Dr. Kiran D C
Department of Computer Science and Engineering

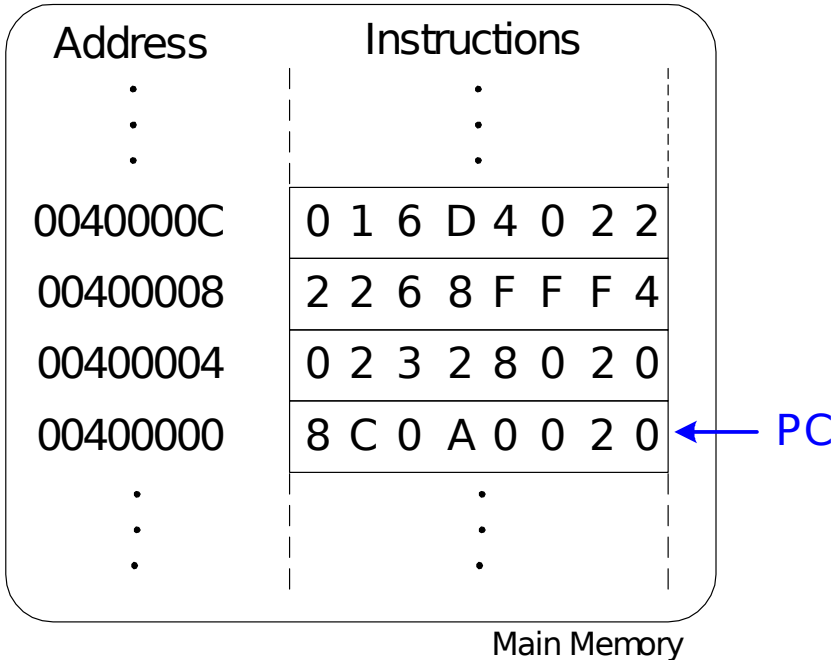
Branching

- Execute instructions out of sequence
- Types of branches:
  - **Conditional**
    - branch if equal (`beq`)
    - branch if not equal (`bne`)
  - **Unconditional**
    - jump (`j`)
    - jump register (`jr`)
    - jump and link (`jal`)

Review: The Stored Program

Assembly Code	Machine Code
`lw $t2, 32($0)`	`0x8C0A0020`
`add $s0, $s1, $s2`	`0x02328020`
`addi $t0, $s3, -12`	`0x2268FFF4`
`sub $t0, $t3, $t5`	`0x016D4022`

Stored Program



Conditional Branching (**beq**)

```
# MIPS assembly
 addi   $s0, $0, 4      # $s0 = 0 + 4 = 4
 addi $s1, $0, 2        # $s1 = 0 + 2 = 2
 add  $s1, $s1, $s1     # $s1 = 2 + 2 = 4
 beq  $s0, $s1, target   # branch is taken
 addi $s1, $s1, 1       # not executed
 sub  $s1, $s1, $s0      # not executed

target:       # label
 add  $s1, $s1, $s0     # $s1 = 4 + 4 = 8
```

Labels indicate instruction location. They can't be reserved words and must be followed by colon (:)

The Branch Not Taken (**bne**)

```
# MIPS assembly
  addi  $s0, $0, 4          # $s0 = 0 + 4 = 4
  addi  $s1, $0, 2          # $s1 = 0 + 2 = 2
  add   $s1, $s1, $s1       # $s1 = 2 + 2 = 4
  bne   $s0, $s1, target    # branch not taken
  addi  $s1, $s1, 1         # $s1 = 4 + 1 = 5
  sub   $s1, $s1, $s0       # $s1 = 5 – 4 = 1

target:
  add   $s1, $s1, $s0       # $s1 = 1 + 4 = 5
```

Unconditional Branching (j)

```
# MIPS assembly
  addi $s0, $0, 4            # $s0 = 4
    addi $s1, $0, 1          # $s1 = 1
    j     target           # jump to target
    sra   $s1, $s1, 2    # not executed
    addi  $s1, $s1, 1    # not executed
    sub   $s1, $s1, $s0     # not executed

  target:
    add   $s1, $s1, $s0     # $s1 = 1 + 4 = 5
```

Unconditional Branching (`jr`)

```
# MIPS assembly
0x00002000        addi $s0, $0, 0x2010
0x00002004        jr   $s0
0x00002008        addi $s1, $0, 1
0x0000200C        sra  $s1, $s1, 2
0x00002010        lw   $s3, 44($s1)
```

`jr` is an **R-type** instruction.

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
```

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
```

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

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:
```

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

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
        add  $s0, $s0, $s0
        addi $s1, $s1, 1
        j    while
done:
```

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:
```

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

```
# $s0 = i, $s1 = sum
```

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
for:    beq  $s0, $t0, done
        add  $s1, $s1, $s0
        addi $s0, $s0, 1
        j    for
done:
```

Think About It

- Assume that register $s1 contains 7 and register $s2 contains 4. Consider the following instruction sequence (the first register is the destination, and the not instruction inverts each bit):

  nor $s2, $s0, $s2

  addi $s2, $s2, 1

  add $s3, $s1, $s2

- What value gets stored in $s3? What arithmetic operation do above instructions perform on $s1 and $s2 contents?