UCCD1133

Introduction to Computer Organisation and Architecture

Chapter 4

Computer Architecture and Organisation Fundamental

Slides adapted from http://booksite.elsevier.com

Disclaimer

- This slide may contain copyrighted material of which has not been specifically authorized by the copyright owner. The use of copyrighted materials are solely for educational purpose. If you wish to use this copyrighted material for other purposes, you must first obtain permission from the original copyright owner.

Chapter 4 - 3

MIPS PROGRAMMING (PART 1)

Programming

- High-level languages:
  - e.g., C, Java, Python
  - Written at higher level of abstraction
- Common high-level software constructs:
  - if/else statements
  - for loops
  - while loops
  - arrays
  - function calls

Instructions: Logical

- Instructions for bitwise manipulation

Operation	C	Java	MIPS
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bitwise AND	&	&	and, andi
Bitwise OR	|	|	or, ori
Bitwise NOT	~	~	nor

Instructions: Logical

- **`and, or, xor, nor`**
  - `and`: useful for masking bits

    Masking all but the least significant byte of a value:

    0xF234012F AND 0x000000FF = 0x0000002F
  - `or`: useful for combining bit fields

    Combine 0xF2340000 with 0x000012BC:

    0xF2340000 OR 0x000012BC = 0xF23412BC
  - `nor`: useful for inverting bits:

    A NOR $0 = NOT A

- **`andi, ori, xori`**
  - 16-bit immediate is zero-extended (*not* sign-extended)
  - nori not needed

Example 1

Source Registers

$s1	1111	1111	1111	1111	0000	0000	0000	0000
$s2	0100	0110	1010	0001	1111	0000	1011	0111

Assembly Code	Result							
`and $s3, $s1, $s2`	$s3							
`or $s4, $s1, $s2`	$s4							
`xor $s5, $s1, $s2`	$s5							
`nor $s6, $s1, $s2`	$s6							

Example 1

Source Registers

$s1	1111	1111	1111	1111	0000	0000	0000	0000
$s2	0100	0110	1010	0001	1111	0000	1011	0111

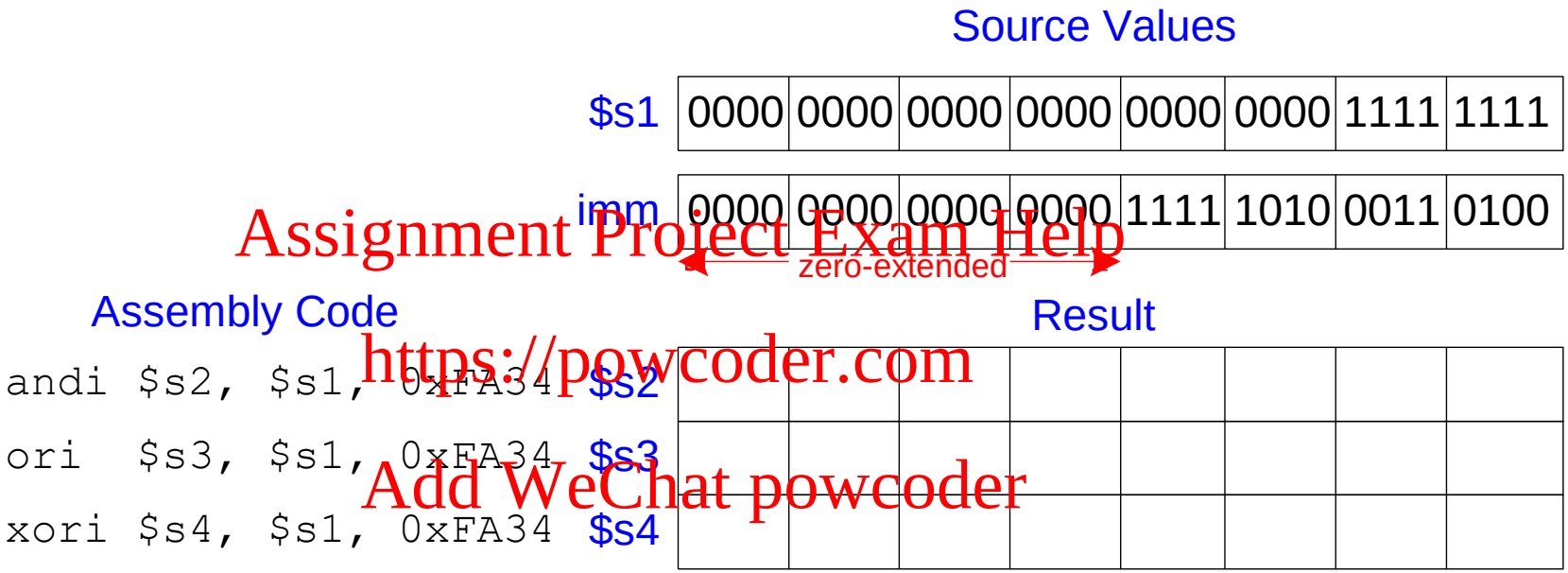
Assembly Code

```
and $s3, $s1, $s2
or  $s4, $s1, $s2
xor $s5, $s1, $s2
nor $s6, $s1, $s2
```

Result

$s3	0100	0110	1010	0001	0000	0000	0000	0000
$s4	1111	1111	1111	1111	1111	0000	1011	0111
$s5	1011	1001	0101	1110	1111	0000	1011	0111
$s6	0000	0000	0000	0000	0000	1111	0100	1000

Example 2



Example 2

Source Values

$s1	0000	0000	0000	0000	0000	0000	1111	1111
imm	0000	0000	0000	0000	1111	1010	0011	0100

zero-extended

Assembly Code

```
andi $s2, $s1, 0xFA34
ori  $s3, $s1, 0xFA34
xori $s4, $s1, 0xFA34
```

Result

$s2	0000	0000	0000	0000	0000	0000	0011	0100
$s3	0000	0000	0000	0000	1111	1010	1111	1111
$s4	0000	0000	0000	0000	1111	1010	1100	1011

Shift Instructions

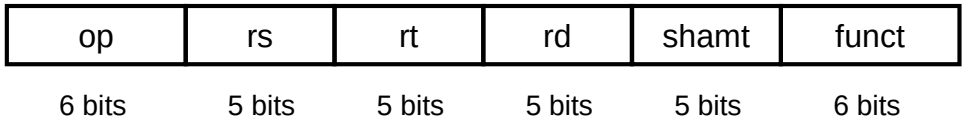
- `sll` (shift left logical)
  - Shift left and fill with 0 bits
  - `sll` by *i* bits multiplies by 2^i
  - Example: `sll $t0, $t1, 5  # $t0 <= $t1 << 5`

- `Srl` (shift right logical)
  - Shift right and fill with 0 bits
  - `srl` by *i* bits divides by 2^i (unsigned only)
  - Example: `srl $t0, $t1, 5  # $t0 <= $t1 >> 5`

- `sra` (shift right arithmetic)

  Example: `sra $t0, $t1, 5  # $t0 <= $t1 >>> 5`

- shamt: how many positions to shift



Shift Instructions

Assembly Code

```
sll $t0, $s1, 2
srl $s2, $s1, 2
sra $s3, $s1, 2
```

Field Values

	op	rs	rt	rd	shamt	funct
sll $t0, $s1, 2	0	0	17	8	2	0
srl $s2, $s1, 2	0	0	17	18	2	2
sra $s3, $s1, 2	0	0	17	19	2	3
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Machine Code

op	rs	rt	rd	shamt	funct	
000000	00000	10001	01000	00010	000000	(0x00114080)
000000	00000	10001	10010	00010	000010	(0x00119082)
000000	00000	10001	10011	00010	000011	(0x00119883)
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

Generating Constants

- 16-bit constants
  - use `addi`:

C Code

```
/* int is a 32-bit
   signed word */
   int a = 0x4f3c;
```

MIPS assembly code

```
# $s0 = a
addi $s0, $0, 0x4f3c
```


- 32-bit constants
  - use load upper immediate (`lui`) and `ori`:

C Code

```
int a = 0xFEDC8765;
```

MIPS assembly code

```
# $s0 = a
lui $s0, 0xFEDC
ori $s0, $s0, 0x8765
```

Multiplication, Division

- Special registers: `lo`, `hi`

- 32 × 32 multiplication, 64 bit result
  - `mult $s0, $s1`
  - Result in {`hi, lo`}

- 32-bit division, 32-bit quotient, remainder
  - `div $s0, $s1`
  - Quotient in `lo`
  - Remainder in `hi`

- Moves from lo/hi special registers
  - `mflo $s2`
  - `mfhi $s3`

Branching

- Execute instructions out of sequence
- Types of branches:
  - **Conditional**
    - Branch to a labeled instruction if a condition is true. Otherwise, continue sequentially.
    - **branch if equal** (`beq rs rt L1`)
      - If (rs == rt) branch to instruction labeled L1
    - **branch if not equal** (`bne rs rt L1`)
      - If (rs != rt) branch to instruction labeled L1
  - **Unconditional**
    - jump (`j`)
    - jump register (`jr`)
    - jump and link (`jal`)

Instructions: Conditional Branch

Branch on equal (beq)

```
# MIPS assembly
 addi    $s0, $0, 4          # $s0 = 0 + 4 = 4
 addi $s1, $0, 1          # $s1 = 0 + 1 = 1
 sll  $s1, $s1, 2         # $s1 = 1 << 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 (:)

Instructions: Conditional Branch

Branch on not equal (bne)

```
# MIPS assembly
  addi   $s0, $0, 4          # $s0 = 0 + 4 = 4
  addi   $s1, $0, 1          # $s1 = 0 + 1 = 1
  sll    $s1, $s1, 2         # $s1 = 1 << 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
```

Instructions: Unconditional Jump

Jump (j L1)

- Unconditional jump to instruction labeled L1

```
# 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
```

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, Else
        add $s0, $s1, $s2
        j Exit
Else:   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, Else
      add $s0, $s1, $s2
      j   Exit
Else: sub $s0, $s0, $s3
Exit:
```

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

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, Exit
        sll  $s0, $s0, 1
        addi $s1, $s1, 1
        j    while
Exit:
```


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

High-level 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

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:
```

More conditional operations

- Set result to 1 if a condition is true
  - Otherwise, set to 0

- `slt rd, rs, rt`
  - if (rs < rt) rd = 1; else rd = 0

- `slti rt, rs, constant`
  - if (rs < constant) rt = 1; else rt = 0

- Use in combination with `beq, bne`

```
slt $t0, $s1, $s2  # if ($s1 < $s2)
bne $t0, $zero, L  #   branch to L
```

Less Than Comparison

C Code

```
// add the powers of 2 from 1
// to 100
int sum = 0;
int i;

for (i=1; i < 101; i = i*2)
  sum = sum + i;
}
```

MIPS assembly code

Less Than Comparison

C Code

```
// add the powers of 2 from 1
// to 100
int sum = 0;
int i;

for (i=1; i < 101; i = i*2)
  sum = sum + i;
}
```

MIPS assembly code

```
# $s0 = i, $s1 = sum
        addi $s1, $0, 0
        addi $s0, $0, 1
        addi $t0, $0, 101
loop:   slt  $t1, $s0, $t0
        beq  $t1, $0, done
        add  $s1, $s1, $s0
        sll  $s0, $s0, 1
        j    loop
done:
```

$t1 = 1 if i < 101