COMPUTER ARCHITECTURE CS2010

BK
TP.HCM

Faculty of Computer Science and Engineering
Department of Computer Engineering

Nam Ho

Instructions:
Language of the Computer

BK
TP.HCM

Exercise 1

- Translate the C statements below into MIPS assembly code

  a. `f = g - A[B[4]]`

  b. `f = g - A[B[k]]`

- Assume that the variables f, g, k and the base address of the arrays A, B are in registers $s0, $s1, $s2, $s3, $s4 respectively and they are declared as 32-bit integers.

Exercise 1

a.

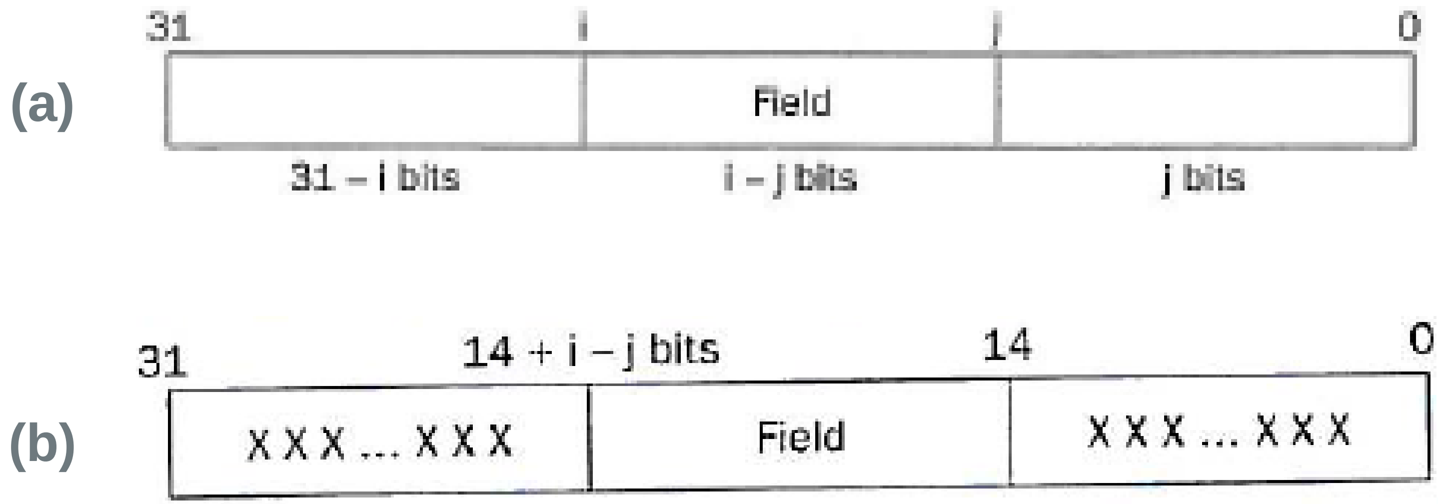
```
lw $s0, 16($s7)          #$s0 = B[4]
sll $s0, $s0, 2          #$s0 = B[4] * 4
add $s0, $s6, $s0        #$s0 = A + B[4] * 4
lw $s0, 0($s0)           #$s0 = A[B[4]]
sub $s0, $s1, $s0        #f = g – A[B[4]]
```

b.

```
sll $s4, $s4, 2          #k * 4
add $s0, $s7, S4         #$s0 = B + k * 4
sll $s0, $s0, 2          #$s0 = B[4] * 4
add $s0, $s6, $s0        #$s0 = A + B[4] * 4
lw $s0, 0($s0)           #$s0 = A[B[4]]
sub $s0, $s1, $s0        #f = g – A[B[4]]
```

Exercise 2

- Write the MIPS code to extract the bits "Field" from register \$t0 shown in the *figure (a)* and place them into register \$t1 at the location indicated in the *figure (b)*. Assume that i = 17, j = 11.



Exercise 2

```
move $s0, $t0
sll  $s0, $s0, 3
andi $s0, $s0, 0x000FC000
and  $t1, $t1, 0xFFF03FFF
ori  $t1, $t1, $s0
```

Exercise 3

- Implement a function `int extract_field(int i, int j)` in C to provide functionality as described in exercise 2. This function returns 1 if `i` or `j` is not satisfy else it returns 0.
- Translate this function from C to MIPS assembly code.

Exercise 4

- What is the total number of MIPS code instruction executed?
- Translate the loop into C. Assume that the C-level integer i is held in register $t1, $s2 holds the C-level integer called result, and $s0 holds the base address of the integer MemArray.

```
        addi   $t1, $0, 100
LOOP:   lw     $s1, 0($s0)
        add    $s2, $s2, $s1
        addi   $s0, $s0, 4
        subi   $t1, $t1, 1
        bne    $t1, $0, LOOP
```

(a)

```
        addi   $t1, $s0, 400
LOOP:   lw     $s1, 0($s0)
        add    $s2, $s2, $s1
        lw     $s1, 4($s0)
        add    $s2, $s2, $s1
        addi   $s0, $s0, 8
        bne    $t1, $s0, LOOP
```

(b)

Exercise 4

a.

```
for(i=100; i>0; i--)
{
  result = result + *(int*)MemArray++;
}

i = (int) MemArray + 400;
```

b.

```
do
{
  result = result + *(int*)MemArray++;
  result = result + *(int*)MemArray++;
}
while( (int)MemArray == i)
```

Exercise 5

- C code:

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

  – Argument **n** in **`$a0`**
  – Result in **`$v0`**

Exercise 5

```
fact: addi  $sp, $sp, -8      #adjust stack pointer
      sw    $ra, 4($sp)       #save return address
      sw    $a0, 0($sp)       #save argument n
      slti  $t0, $a0, 1       #test for n < 1
      beq   $t0, $zero, L1    #if n >=1, go to L1
      addi  $v0, $zero, 1     #else return 1 in $v0
      addi  $sp, $sp, 8       #adjust stack pointer
      jr    $ra               #return to caller (1st)

L1:   addi  $a0, $a0, -1      #n >=1, so decrement n
      jal   fact              #call fact with (n-1)
      #this is where fact returns

bk_f: lw    $a0, 0($sp)       #restore argument n
      lw    $ra, 4($sp)       #restore return address
      addi  $sp, $sp, 8       #adjust stack pointer
      mul   $v0, $a0, $v0     #$v0 = n * fact(n-1)
      jr    $ra               #return to caller (2nd)
```

Exercise 5

- Show the stack's state and the values in register `$ra, $a0, $v0` at the encounter instructions `jal, jr` when we have a calling `fact(3)` from main function.

Exercise 6

- Assume that the PC is at address 0x00000000. How many branch and jump instructions are required to get the target address bellow ?
  - 0x0000 1000
  - 0xFFFC 0000

Exercise 6

39. ***Branch on Equal*: BEQ**

```
                  +--------+-------+-------+----------------------+
I-type format:    | 000100 |  Rs   |  Rt   |        offset        |
                  +--------+-------+-------+----------------------+
```

Effects: if $[R_s]=[R_t]$ then PC <-- $[PC]+4+([I_{15}]^{14} \,||\, [I_{15..0}] \,||\, 0^2)$
else PC <-- [PC] + 4

Address Range =

- **0x00000000 + 0x04 + 0x1FFFC = 0x00020000**
- **0x00000000 + 0x04 - 0x20000 (0x8000||00_2, 2's complement) = 0xFFFE0004**

- **PC + 4 * n + X1 + .. +Xn = $T_{Address}$**
- **4 *n + Xmax (n – 1) + Xn = $T_{Address}$ - PC**
- **(4 – Xmax)*n + Xn – Xmax = $T_{Address}$ - PC**

a. 0x00001000 div 0x00020000 = 0

b. 0xFFFC0000 div 0xFFFE004 = 2

Exercise 6

```
44. Jump: J
                +--------+---------------------------------------
+
J-type format|000010  |               jump_target
|
                +--------+---------------------------------------
+
```

Effects: PC <-- $[PC_{31..28}]$ || $[I_{25..0}]$ || 0^2

Address Range =

- 0000_2 || 0x3FFFFFF||00_2 = 0x0FFFFFFC

Exercise 7

- Exercise 2.28 from the textbook.

MIPS Assembler

- Program Layout

```
        .text               #code section
        .globl main         #starting point: must be global
main:
        # user program code
        .data               #data section
label:  .data_type list_of_data
                            #data loc + data type + data
        .text               #code section
label:                      #function label
        #user functions
```

MIPS Assembler

- Data Types
  - **.word, .half** - 32/16 bit integer
  - **.byte** - 8 bit integer (similar to ‘char’ type in C)
  - **.ascii, .asciiz** - string (asciiz is null terminated)
    - Strings are enclosed in double-quotas(”)
    - Special characters in strings follow the C convention
      - newline(\n), tab(\t), quote(\”)
  - **.double, .float** - floating point

MIPS Assembler

- Other Directives
  - **.text** - Indicates that following items are stored in the user text segment
  - **.data** - Indicates that following data items are stored in the data segment
  - **.globl *sym*** - Declare that symbol *sym* is global and can be referenced from other files

MIPS Assembler - Array

- A Program with Procedure Call

```
# sample example 'swap two numbers'
.text
.globl   main
main:
         la       $a0, array
         addi     $a1, $0, 0

         addi     $sp, $sp, -4
         sw       $ra, 0($sp)

         jal      swap

         lw       $ra, 0($sp)
         addi     $sp, $sp, 4

         jr       $ra

.data
array:   .word 5, 4, 3, 2, 1
```

```
#        swap(int v[], int k)
#        {
#                 int temp;
#                 temp = v[k];
#                 v[k] = v[k+1];
#                 v[k+1] = temp;
#        }

swap:    add      $t1, $a1, $a1
         add      $t1, $t1, $t1
         add      $t1, $a0, $t1
         lw       $t0, 0($t1)
         lw       $t2, 4($t1)
         sw       $t2, 0($t1)
         sw       $t0, 4($t1)
         jr       $ra
```