Computer Architecture
Fall, 2017
Week 4
2017.10.02

[group14]

1. 請說明為何 R-Type Instruction 中其指令格式的 shamt 僅要 5-bits? (已知 shamt 用於 sll, srl)

Ans:

我們都知道在 MIPS 裡面一個 register 總共也就只有 32 個 bit，當我們要 shift 的數量超過 31 個 bit 其實是沒有任何意義的，因為所有的資料都 shift 光了，這也就是為什麼，我們總共只需要用 5 個 bit 就可以明確的去描述我們所需要 shift 的個數。

[group3]

2. 有一陣列 A[i]，請寫出指令

if (A[i] > 10) goto More ;

的 MIPS code，其中 i in $s3，address of A in $s4。

Ans:

```
sll     $t1 , $s3 , 2
add    $t2 , $t1 , $s4
lw       $t0 , 0($t1)
addi    $t4 , $zero , 10
slt      $t2 , $t4 , $t0
addi    $t5 , $zero , 1
beq    $t3 , $5 , More
```

[group6]

3. R-format 中 opcode 與 funct 是用來辨別是何 instruction 用的
為何不將兩者合併為一個 12-bit field?

Ans:

為了讓 R-format 與 I-format 越接近越好，opcode 與 funct 分開能讓 R-format 與 I-format 都有 opcode、rs、rt 三個相同的 field

[group12]

4. 在 MIPS 中，下列 instructions 分別為何種 format(R-format, I-format, J-format)?

(a) add
(b) addi
(c) bne
(d) j
(e) sw
(f) sll
(g) slt
(h) sra

Ans:

R-format: a, f, g, h
I-format: b, c, e
J-format: d

[group7]

5. 請將下列數字轉成 assembly code

00100001000010011111111111110110

Ans:

先轉成 10 進位制得到 8,8,9,-10
根據查表知道答案為 addi $8, $9, -10

[group10]

6. 令 $s0=1023，考慮執行以下 MIPS code:

```
sra $t0 $s0 5
sll $s0 $t0 4
and $t1 $s0 $t0
```

請問執行完以上 code 後，$t1, $s0, $t0 的值分別為何?

Ans:

初始值: $s0 = 0000 0000 0000 0011 1111 1111

```
sra $t0 $s0 5        # $t0 = 0000 0000 0000 0000 0001 1111
sll $s0 $t0 4        # $s0 = 0000 0000 0000 0001 1111 0000
```

and $t1 $s0 $t0　　# $t1 = 0000 0000 0000 0000 0001 0000

故$t0= 31，$s0= 496，$t1=16

[group11]

7. 小王將以下組合語言翻譯為 c 語言時 ,高階語言某些部份空格__想不出來，請大家幫他完成:

（i in $s0, address of A in $s1, address of B in $s2）

```
loop:   sll     $t0,    $s0,    2           # $t0= i*4
        add     $t1,    $t0,    $s1         # $t1= address of A[i]
        lw      $t2,    0($t1)              # $t2= A[i]
        beq     $t2,    $zreo, Exit         #小王不會,快幫他想想 beq
        add     $t3,    $t0,    $s2         # $t3 = address of B[i]
        lw      $t4,    0($t3)              # $t4 =B[i]
        sll     $t4,    $t4,    4           #小王不會,要被當了
        sw      $t4,    0($t3)
        addi    $s0,    $s0,    1           # i=i+1
        j       loop
Exit:
```

Ans:

(1)判斷回推的能力

(2)段是否懂往左位移為乘 2 的意思

```
while( A[i] != 0 )
{
    B[i] *= 16;
    i += 1;
}
```

[group1]

8. 我要取數字 a 的前 16 個 bit 跟數字 b 的後 16 個 bit 請問我該怎麼做呢請用 mips 指令解答? (a 為$s0 b 為$s1 結果存在$s2)

Ans:

(Ans 1)

```
addi $t0 $zero -1
sll $t1 $t0 16
srl $t2 $t0 16
```

and $s0 $s0 $t1
and $s1 $t2 $s1
add $s2 $s0 $s1

(Ans 2)
srl $s2, $s0, 16
sll $s2, $s2, 16
andi $t0, $s1, 0xFFFF
or $s2, $s2, $t0

(Ans 3)
srl $t0, $s0, 16
sll $t0, $t0, 16
sll $t1, $s1, 16
srl $t1, $t1, 16
or $s2, $t0, $t1

(Ans 4)
srl $t0, $s0, 16
sll $t0, $t0, 16
sll $t1, $s1, 16
srl $t1, $t1, 16
add $s2, $t0, $t1

[group2]

9. 寫出下列的 MIPS instruction format，以 binary representation 表示

g = h + k

variable g,h,k are assigned to registers $s1, $s2, $s4
and R-format op code = 0
funt code of add = 32
$s1:17, $s2:18, $s3:19…
$t0:8, $t1:9, $t3:10…

Ans:

add $s1, $s2, $s4
=> 0 | $s1 | $s2 | $s4 | 0 | 32
=> 0 | 17 | 18 | 20 | 0 | 32 (decimal)

=> 000000 | 10001 | 10010 | 10100 | 00000 | 100000

[group13]

10. 請解釋何謂 stored program，何謂 binary compatibility。

Ans:

Stored program 是一種程式執行的概念，其方法是在程式執行時，先將程式儲存在 Memory 中，再由中央處理單元執行。

好處是只需將程式載入即可，無須改變所需的硬體線路。

而 binary compatibility 是說由一台電腦編譯出的 machine code，能讓另一台機器執行。

[group5]

11. w = x - Y[Z[8]]

w: $s0, x: $s1, Y: $s2, Z: $s3

For the C statement above, what is the corresponding MIPS assembly code?

Ans:

```
lw $s0, 32($s3)        //$s0 = Z[8]
sll $s0, $s0, 2        //$s0 = Z[8] * 4
add $s0, $s0, $s2      //$s0 = address of Y[Z[8]]
lw $s0, 0($s0)         //$s0 = Y[Z[8]]
sub $s0, $s1, $s0      //$s0 = x - Y[Z[8]]
```

[group8]

12. Write the equivalent MIPS assembly code, where i is in $s0 and sum is in $s1:

```
        i = 10;
        sum = 0;
WHILE:
        if (i < 0) goto END_WHILE;
        sum = sum + i;
        i = i – 1;
        goto WHILE;
END_WHILE:
```

Ans:

```
        addi        $s0 $0 10
        addi        $s1 $0 0
WHILE:
        slti        $t0 $s0 0
        bne       $t1 $0 END_WHILE
        add       $s1 $s1 $s0
        subi       $s0 $s0 1
        j       WHILE
END_WHILE:
```