COE 308 – Computer Architecture

Exam I – Spring 2008

Tuesday, April 1st, 2008
7:00 pm – 9:00 pm

Computer Engineering Department
College of Computer Sciences & Engineering
King Fahd University of Petroleum & Minerals

Student Name: **SOLUTION**

Student ID: ____________________

Q1	/ 15	Q2	/ 15
Q3	/ 15	Q4	/ 10
Q5	/ 10	Q6	/ 20
Q7	/ 20		
Total	/ 105		

Important Reminder on Academic Honesty

Using unauthorized information on an exam, peeking at others work, or altering graded exams to claim more credit are severe violations of academic honesty. Detected cases will receive a failing grade in the course.

Q1. (15 pts) Given the bit pattern:

`1100 0110 1101 0100 0000 0000 0000 0000` (binary)

What is the decimal value of the above number, assuming it is

a) (2 pts) Unsigned integer?

$$2^{31} + 2^{30} + 2^{26} + 2^{25} + 2^{23} + 2^{22} + 2^{20} + 2^{18} = 3{,}335{,}782{,}400$$

b) (2 pts) Signed integer?

$$3{,}335{,}782{,}400 - 2^{32} = -959{,}184{,}896$$

c) (5 pts) Single-precision floating-point number?

Sign = negative

Exponent value = 10001101_2 **– Bias = 141 – 127 = 14**

Decimal Value = $-1.10101_2 \times 2^{14} = -1.65625 \times 2^{14} = -27136$

d) (6 pts) Show the **Single precision** IEEE 754 representation for **-0.05**, rounded to the nearest even.

```
0.05 * 2 = 0.1
0.1 * 2 = 0.2
0.2 * 2 = 0.4
0.4 * 2 = 0.8
0.8 * 2 = 1.6
0.6 * 2 = 1.2
0.2 * 2 = 0.4
```

$$\boxed{\begin{array}{l} 0.05 = 0.000011001\overline{1001}_2 = 1.1001\overline{1001}_2 \times 2^{-5} \\ \text{Exponent} = -5 + 127 = 122 = 01111010_2 \end{array}}$$

Single Precision Representation:

1 01111010 10011001100110011001101 (rounded)

Q2. (15 pts) Consider the following data definitions:

```
.data
var1:    .byte    3, -2, 'A'
var2:    .half    1, 256, 0xffff
var3:    .word    0x3de1c74, 0xff
.align 3
str1:    .asciiz  "COE308"
```

a) Show the content of each byte of the allocated memory, **in hexadecimal** for the above data definitions. The **Little Endian** byte ordering is used to order the bytes within words and halfwords. Fill the symbol table showing **all labels** and their **starting address**. The ASCII code of character 'A' is 0x41, and '0' is 0x30. Indicate which bytes are skipped or unused in the data segment.

Data Segment

Address	Byte 0	Byte 1	Byte 2	Byte 3
0x10010000	0x03	0xfe	0x41	--
0x10010004	0x01	0x00	0x00	0x01
0x10010008	0xff	0xff	--	--
0x1001000C	0x74	0x1c	0xde	0x03
0x10010010	0xff	0x00	0x00	0x00
0x10010014	--	--	--	--
0x10010018	0x43	0x4F	0x45	0x33
0x1001001C	0x30	0x38	0x00	
0x10010020				
0x10010024				
0x10010028				
0x1001002C				

← **Unused**

Symbol Table

Label	Address
var1	**0x10010000**
var2	**0x10010004**
var3	**0x1001000C**
str1	**0x10010018**

b) How many bytes are allocated in the data segment including the skipped bytes?

31 Bytes including the skipped ones

Q3. (15 pts) For each of the following pseudo-instructions, produce a **minimal** sequence of real MIPS instructions to accomplish the same thing. You may use the **`$at`** register only as a temporary register.

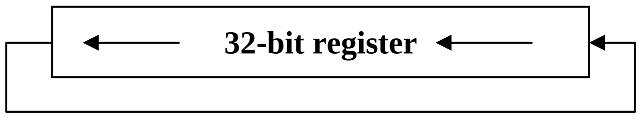
a)
```
abs   $s1, $s2

addu $s1, $zero, $s2
bgez $s2, next
subu $s1, $zero, $s2
next:
```

b)
```
addiu $s1, $s2, imm32   # imm32 is a 32-bit immediate


lui  $at, upper16
ori  $at, $at, lower16
addu $s1, $s2, $at
```

c)
```
bleu $s1, $s2, Label    # branch less than or equal unsigned

sltu $at, $s2, $s1
beq  $at, $zero, Label
```

d)
```
bge $s1, imm32, Label   # imm32 is a 32-bit immediate


lui $at, upper16
ori $at, $at, lower16
slt $at, $s1, $at
beq $at, $zero, Label
```

e)
```
rol $s1, $s2, 5         # rol = rotate left $s2 by 5 bits

srl $at, $s2, 27
sll $s1, $s2, 5
or  $s1, $s1, $at
```



Q4. (10 pts) Translate the following loop into assembly language where **a** and **b** are integer arrays whose base addresses are in **$a0** and **$a1** respectively. The value of **n** is in **$a2**.

```
for (i=0; i<n; i++) {
  if (i > 2) {
    a[i] = a[i-2] + a[i-1] + b[i];
  }
  else {
    a[i] = b[i]
  }
}
```

```
         li     $t0, 0         # $t0 = i = 0
         beq    $a2, $0, skip  # skip loop if n is zero
loop:    lw     $t1, 0($a1)    # $t1 = b[i]
         bgt    $t0, 2, else   # if (i>2) goto else
         lw     $t2, -8($a0)   # $t2 = a[i-2]
         lw     $t3, -4($a0)   # $t3 = a[i-1]
         addu   $t2, $t2, $t3  # $t2 = a[i-2]+a[i-1]
         addu   $t1, $t2, $t1  # $t1 = a[i-2]+a[i-1]+b[i]
else:    sw     $t1, 0($a0)    # a[i] = $t1
         addiu  $a0, $a0, 4    # advance array a pointer
         addiu  $a1, $a1, 4    # advance array b pointer
         addiu  $t0, $t0, 1    # i++
         bne    $t0, $a2, loop
skip:
```

Q5. (10 pts) Translate the following **if-else** statement into assembly language:

```
if (($t0 >= '0') && ($t0 <= '9')) {$t1 = $t0 – '0';}
else if (($t0 >= 'A') && ($t0 <= 'F')) {$t1 = $t0+10-'A';}
else if (($t0 >= 'a') && ($t0 <= 'f')) {$t1 = $t0+10-'a';}
```

```
    blt   $t0, '0', else1
    bgt   $t0, '9', else1
    addiu $t1, $t0, -48      # '0' = 48
    j     next
else1:
    blt   $t0, 'A', else2
    bgt   $t0, 'F', else2
    addiu $t1, $t0, -55      # 10-'A' = 10-65=-55
    j     next
else2:
    blt   $t0, 'a', next
    bgt   $t0, 'f', next
    addiu $t1, $t0, -87      # 10-'a' = 10-97=-87
next:
```

Q6. (20 pts) Given that x = **1 10000101 10110000000000000000001**$_2$ and y = **1 01111111 01000000000000001100000**$_2$ are single precision IEEE 754 floating-point numbers. Perform the following operations showing all the intermediate steps and final result in binary. Round to the nearest even.

a) (10 pts) $x + y$
b) (10 pts) $x * y$

a)

- 1.101 1000 0000 0000 0000 0001$_2$ **×** 2^6
- 1.010 0000 0000 0000 1100 0000$_2$ **×** 2^0

- 1.101 1000 0000 0000 0000 0001$_2$ **×** 2^6
- 0.000 0010 1000 0000 0000 0011 000000$_2$ **×** 2^6 **(shift)**

- 1.101 1010 1000 0000 0000 0100 000000$_2$ **×** 2^6 **(add)**

- 1.101 1010 1000 0000 0000 0100 × 2^6 **(round to nearest)**

1 10000101 101 1010 1000 0000 0000 0100

b) **Biased exponent = 10000101**$_2$ **+ 01111111**$_2$ **– 127 = 10000101**$_2$
Result sign = 0 (positive)

```
                              1.10110000000000000000001₂
                            × 1.01000000000000001100000₂
    ────────────────────────────────────────────────────
                         110110000000000000000001000000₂
                        1101100000000000000000010000000₂
          110110000000000000000001000000000000000000000₂
        1.1011000000000000000000100000000000000000000000₂
    ────────────────────────────────────────────────────
       10.0001110000000010100010101000000000000011000000₂
```

Normalize and adjust exponent:

1.00001110000000010100010 1 01000000000000011000000$_2$

Biased exponent = 10000101$_2$ **+ 1 = 10000110**$_2$

Round to nearest even:

Round bit = 1, Sticky bit = 1 (OR of remaining bits)

Rounded Significand = 1.00001110000000010100010$_2$ **+ 1**
= 1.00001110000000010100011$_2$

Result:

0 10000110 00001110000000010100011$_2$

Q7. (20 Pts) Write MIPS assembly code for the procedure **BinarySearch** to search an array which has been previously sorted. Each element in the array is a 32-bit signed integer. The procedure receives three parameters: register **$a0** = **address of array** to be searched, **$a1** = **size** (number of elements) in the array, and **$a2** = **item** to be searched. If found then **BinarySearch** returns in register **$v0** = **address** of the array element where **item** is found. Otherwise, **$v0** = **0**.

```
BinarySearch ($a0=array, $a1=size, $a2=item) {
  lower = 0;
  upper = size-1;
  while (lower <= upper) {
    middle = (lower + upper)/2;
    if (item == array[middle])
      return $v0 = ADDRESS OF array[middle];
    else if (item < array[middle])
      upper = middle–1;
    else
      lower = middle+1;
  }
  return $v0=0;
}
```

```
BinarySearch:
    li      $t0, 0            # $t0 = lower index
    addiu   $t1, $a1, -1      # $t1 = upper index
loop:
    bgt     $t0, $t1, ret
    addu    $t2, $t0, $t1     # $t2 = lower+upper
    srl     $t2, $t2, 1       # $t2 = (lower+upper)/2
    sll     $v0, $t2, 2       # $v0 = middle*4
    addu    $v0, $a0, $v0     # $v0 = address array[middle]
    lw      $t3, 0($v0)       # $t3 = value array[middle]
    bne     $a2, $t3, else1   # (item == array[middle])?
    jr      $ra               # return
else1:
    bgt     $a2, $t3, else2   # (item < array[middle])?
    addiu   $t1, $t2, -1      # upper = middle-1
    j       loop
else2:
    addiu   $t0, $t2, 1       # lower = middle+1
    j       loop
ret:
    andi    $v0, $v0, 0       # $v0 = 0
    jr      $ra               # return
```