COE 301 – Computer Organization

Midterm Exam – Spring 2015

Wednesday, April 1, 2015
6:30 – 8:30 pm

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

Student Name: **SOLUTION**

Student ID: ______________________

Q1	/ 20	Q2	/ 15
Q3	/ 10	Q4	/ 15
Q5	/ 25	Q6	/ 20
Total	/ 105		

Important Reminder on Academic Honesty

Using unauthorized information or notes 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.

Prepared by Dr. Muhamed Mudawar

Q1. (20 pts) Fill in the blanks

a) (4 pts) **`0xAFEC3A15 + 0xB5182C90`** equals to **`0x650466A5`** in hexadecimal. The addition produces a [✓ CARRY] / NO CARRY (circle one) and it produces [✓ OVERFLOW] / NO OVERFLOW (circle one). Show addition for full mark.

b) (4 pts) **`0xAFEC3A15 - 0xB5182C90`** equals to **`0xFAD40D85`** in hexadecimal. The subtraction produces a [✓ BORROW] / NO BORROW (circle one) and it produces OVERFLOW / [✓ NO OVERFLOW] (circle one). Show subtraction for full mark.

c) (2 pts) Assume that you are in a company that will market a certain IC chip. The cost per wafer is \$5000, and each wafer has 2000 dies. If the cost of a good die is \$4, then the yield of this manufacturing process is **(\$5000/\$4) / 2000 = 0.625 = 62.5%**.

d) (2 pts) The smallest 32-bit negative number that can be represented using 2's complement representation in hexadecimal is **`0x80000000`** and the largest positive number in hexadecimal is **`0x7FFFFFFF`**.

e) (2 pts) Assume that the instruction **`j NEXT`** is at address **`0x00401030`**, and the label **`NEXT`** is at address **`0x00400A18`**. Then, the **26-bit immediate** stored in the jump instruction for the label **`NEXT`** is **`0x00400A18 >> 2 = 0x100286`**.

f) (2 pts) Assume that the instruction **`beq $t0,$t1,NEXT`** is at address **`0x00401030`**, and the label **`NEXT`** is at address **`0x00402A18`**. Then, the **16-bit immediate** stored in the branch instruction is **`(0x00402A18-0x00401034)>>2 = 0x0679`**.

g) (4 pts) Given the following data definitions, the address of the first variable **`X`** is given at **`0x10010000`** **(hexadecimal)**, Construct a symbol table showing the symbols **X**, **Y**, **Z**, **S** and their corresponding addresses in hexadecimal.

```
.data
  X: .byte   'A', 'B', 'C'
  Y: .half   1, -2, 100
  Z: .word   7, 8, 0x123
  S: .asciiz  "STRING"
```

Symbol	Address
X	0x10010000
Y	0x10010004
Z	0x1001000C
S	0x10010018

Q2. (15 pts) Integer Multiplication

a) (10 pts) Show the binary multiplication of the following two 16-bit unsigned integers. The product should be a 32-bit unsigned integer.

```
                                1 0 1 1 0 0 1 0 0 0 0 0 1 0 0 1
                              × 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0
                              ---------------------------------
              1     1 1 1 1 1 1   1     1
                              1 0 1 1 0 0 1 0 0 0 0 0 1 0 0 1 0
                      1 0 1 1 0 0 1 0 0 0 0 0 1 0 0 1
            1 0 1 1 0 0 1 0 0 0 0 0 1 0 0 1
    1 0 1 1 0 0 1 0 0 0 0 0 1 0 0 1
    -------------------------------------------------------------
0 0 1 0 1 1 1 1 0 1 1 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 1 0
```

b) (5 pts) To implement a 16-bit tree multiplier in hardware, how many AND gates are used? How many carry-save adders are needed? How many carry-propagate adders are needed? Explain your answer.

Number of AND gates = 16 × 16 = 256

The 16-bit tree multiplier has 16 intermediate product results

Total number of adders = 15

Number of Carry-Save adders = 14

Number of Carry-Propagate adders = 1

Q3. (10 pts) Floating-Point Number Representation

a) (5 pts) Given that *x* is a single-precision IEEE 754 floating-point number:

x = **`1 10000101 100 1010 0001 1010 0000 0011`**$_2$

What is the decimal value of *x*?

Sign bit = 1 (negative)

Biased Exponent = 10000101 = 133

Exponent Value = 133 – 127 = +6

$$\text{Value} = -\,(1.10010100001101000000011)_2 \times 2^6$$
$$= -\,(1100101.00001101000000011)_2$$

Decimal Value = - 101.0508

b) (5 pts) Convert **`-6.25`** from decimal to the IEEE 754 single-precision floating point format. Show all your work for each step in the solution.

$0.25 \times 2 = 0.5$
$0.50 \times 2 = 1.0$

6.25 (decimal) = 110.01 (binary)

Normalize:

110.01 (binary) $= 1.1001 \times 2^2$

Biased Exponent = 2 + 127 = 129 = 10000001 (binary)

IEEE 754 Single-Precision Representation:

1 10000001 100 1000 0000 0000 0000 0000

Q4. (15 pts) Tracing the Execution of Assembly Language Code

a) (7 pts) Given that `Array` is defined as shown below, determine the content of register **`$v0` and `$v1`** after executing the following code. Show your steps.

```
Array: .word 15, -19, 17, 20, -10, 12, 100, -5

         la   $a0, Array     # $a0 = 0x10010000
         addi $a1, $a0, 28
         move $v0, $a0
         lw   $v1, 0($v0)
         move $t0, $a0
loop:    addi $t0, $t0, 4
         lw   $t1, 0($t0)
         bge  $t1, $v1, skip
         move $v0, $t0
         move $v1, $t1
skip:    bne  $t0, $a1, loop
```

```
$v0 = 0x10010004 (address of minimum element)

$v1 = -19 (minimum value)
```

b) (8 pts) Given that `Array` is defined as shown below, determine the content of `Array` after executing the following code. Show your steps.

```
Array: .half 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

     la   $a0, Array
     li   $a1, 6
     move $t0, $a0
     addi $t1, $a0, 12

loop: lh   $t3, ($t0)
     lh   $t4, ($t1)
     sh   $t3, ($t1)
     sh   $t4, ($t0)
     addi $t0, $t0, 2
     addi $t1, $t1, 2
     addi $a1, $a1, -1
     bne  $a1, $zero, loop
```

```
New Array Content:

7, 8, 9, 10, 11, 12, 1, 2, 3, 4, 5, 6

(swapping the first six elements with the last six)
```

Q5. (25 pts) Writing Assembly Language Functions

a) (12 pts) Write a MIPS function named **`count1s`** to count the number of 1's in register **`$a0`** and put the result in register **`$v0`**. For example, if **`$a0 = 0xffff0000`** then the number of 1's will be **`$v0 = 16`**.

```
count1s:
       li   $v0, 0              # initialize $v0 = 0
loop:  andi $t0, $a0, 1         # $t1 = bit 0 of $a0
       add  $v0, $v0, $t0       # $v0 = count bit in $t0
       srl  $a0, $a0, 1
       bne  $a0, $zero, loop    # loop until ($a0 == 0)
       jr   $ra                 # return to caller
```

b) (13 pts) Write a function **`gcd`** to compute the greatest common divisor of two unsigned integers as follows:

`gcd(a,0) = a`

`gcd(a,b) = gcd(b,a%b) // a%b is the remainder of division`

For example: **`gcd(8,12)=gcd(12,8)=gcd(8,4)=gcd(4,0)=4`**.

The arguments are passed in registers **`$a0`** and **`$a1`** and the result is returned in **`$v0`**.

```
gcd:
       bne  $a1, $0, else      # branch if (b != 0) else
       move $v0, $a0           # $v0 = a
       jr   $ra                # return to caller
else:  divu $a0, $a1           # divide a by b
       move $a0, $a1           # $a0 = b
       mfhi $a1                # $a1 = remainder a%b
       j    gcd                # jump to gcd
```

Q6. (20 pts) Translating a Function into MIPS Assembly Language

The function **BinarySearch** searches an array of integers for a given item. Each element in the array is a 4-byte signed integer. The procedure receives three parameters: **$a0** = **address** of the array to be searched, **$a1** = number **n** of elements in the array, and **$a2** = **item** to be searched for. If **item** is found then **BinarySearch** returns the index of **item** in register **$v0**. Otherwise, **$v0 = -1**. Translate this function into MIPS assembly language and insert comments to explain the use of registers.

```
int BinarySearch (int array[], int n, int item) {
  int lower = 0;
  int upper = n-1;
  while (lower <= upper) {
    middle = (lower + upper)/2;
    if (item == array[middle])
      return middle;
    else if (item < array[middle])
      upper = middle–1;
    else
      lower = middle+1;
  }
  return -1;
}
```

Solution:

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