COE 301 / ICS 233
Computer Organization

Midterm Exam – Term 172

Saturday, March 24, 2018

10:00 am – 12:00 noon

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

SOLUTION

☐ Dr. Aiman El-Maleh	☐ COE 301	☐ ICS 233
☐ Dr. Marwan Abu-Amara	☐ COE 301	☐ ICS 233
☐ Dr. Muhamed Mudawar		☑ ICS 233

Q1	/ 19	Q2	/ 17
Q3	/ 7	Q4	/ 14
Q5	/ 7	Q6	/ 16
Total	/ 80		

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.

Q1. [19 points] Fill-in the Blanks

a) (1 point) In addition to being space and time efficient, programming in assembly language has the advantage of **providing accessibility to system hardware**.

b) (1 point) In addition to faster program development and maintenance, programming in high-level language has the advantage **that programs are portable**.

c) (1 point) The instruction set architecture of a processor provides an interface between **software and hardware**.

d) (1 point) Assuming **`Array`** is defined as shown below:

```
Array: .word 10, 11, 12, 13, 14
```

The content of register **`$t1`** (in hexadecimal) after executing the following sequence of instructions is **0x0000000b**.

```
la $t0, Array
lw $t1, 4($t0)
```

e) (2 points) Write a minimum sequence of MIPS basic instructions to implement the pseudo instruction: **`bgt $s1,$s2,Next`**

```
slt $at, $s2, $s1
bne $at, $0, Next
```

f) (2 points) Write a minimum sequence of MIPS basic instructions to implement the pseudo instruction: **`andi $t0,$t0,0x12345678`**

```
lui $at, 0x1234
ori $at, $at, 0x5678
and $t0, $t0, $at
```

g) (1 point) Assuming that **`$a0`** contains an Alphabetic character (uppercase or lowercase), write a MIPS instruction that will make the character stored in **`$a0`** always lower case. Note that the ASCII code of character **`'A'`** is **`0x41`** while that of character **`'a'`** is **`0x61`**.

```
ori $a0, $a0, 0x20
```

h) (3 points) The following is a partial MIPS assembly language code:

Address	Label	Instruction
0x00400000		bgtz $a1, loop
		. . .
0x00403000	loop:	addu $a0, $a1, $v0
		. . .
0x00410000		bne $a0, $zero, loop

Calculate the 16-bit immediate value (in hexadecimal) for **loop** in the **bgtz** instruction.
Calculate the 16-bit immediate value (in hexadecimal) for **loop** in the **bne** instruction.

bgtz: (0x00403000-0x00400004)/4 = 0x00002FFC/4 = 0x0BFF

bne: (0x00403000-0x00410004)/4 = -0x0000D004/4 = 0xCBFF

i) (2 points) Given that **Array2** is defined as shown below:

```
Array2: .byte  -1, 2, -3, -4, -5, 6
```

After executing the following sequence of instructions, the content of the two registers (in hexadecimal) is **$t1=** **0xFFFFFFFD** and **$t2=** **0x0000FCFD**.

```
la   $t0, Array2
lb   $t1, 2($t0)
lhu  $t2, 2($t0)
```

j) (2 points) Given the following data definitions, and assuming that the first variable **X** starts at address **0x10010000**, then the addresses for variables **Y** and **Z** will be **0x10010004** and **0x1001000C**.

```
.data
X: .byte 10, 11, 12
Y: .half 13, 14, 15
Z: .word 16, 17, 18
```

k) (3 points) Write a minimum sequence of MIPS basic instructions to multiply the **signed** integer value of register **$t0** by **15.25** without using multiplication instructions. Put the final integer result in **$t0**. For example, if the initial value of **$t0** is **5** then the final value will be **76**. The additional fraction is truncated.

```
sll  $t1, $t0, 4
sub  $t1, $t1, $t0
sra  $t0, $t0, 2
addu $t0, $t0, $t1
```

Q2. Floating-Point [17 points]

a) (3 points) Find the decimal value of the following single-precision float:

S	Exponent	Fraction
1	1000 1000	111 1100 1101 0000 0000 0000

Sign bit = 1 (negative)
Biased Exponent = 1000 1011 = 136
Exponent Value = 136 – 127 = +9
Value = $-(1.111\ 1100\ 1101\ 0000\ 0000\ 0000)_2 \times 2^{+9}$
= $-(1\ 111\ 1100\ 11.01\ 0000\ 0000\ 0000)_2$
Decimal Value = –1011.25

b) (4 points) Find the **normalized** IEEE 754 single-precision representation of **+59.625**.

+59.625 = +111011.101

Normalize: +59.625 = $+1.110\ 1110\ 1000\ 0000\ 0000\ 0000 \times 2^{+5}$

Biased Exponent: E = 5 + 127 = 132

S	Exponent	Fraction
0	1000 0100	110 1110 1000 0000 0000 0000

c) (2 points) Show the IEEE 754 representation of **+Zero** and **-Infinity** for single precision:

+Zero: [0, 00000000, 000 0000 0000 0000 0000 0000]

-Infinity: [1, 11111111, 000 0000 0000 0000 0000 0000]

d) (2 points) Find the approximate decimal value of the largest positive **denormalized** float for single precision.

Exponent = −126

Significand = $(0.111 \ldots 1)_2 \approx 1$

Value in decimal $\approx 1 \times 2^{-126}$, Exact value $=1.1754942 \times 10^{-38}$

e) (6 points) Given that **A** and **B** are single-precision floats, compute the difference **A−B**. Use rounding to **nearest even**. Perform the operation using **guard**, **round** and **sticky** bits.

A = +1.111 0000 0000 1111 1100 0001 × 2^{-4}
B = +1.000 1111 1111 0000 0000 1111 × 2^{+3}

Solution:

+	1.111 0000 0000 1111 1100 0001	x 2^{-4}	
−	1.000 1111 1111 0000 0000 1111	x 2^{+3}	
+	0.000 0001 1110 0000 0001 1111 101	x 2^{+3}	Align ➔ 3 Extra bits
−	1.000 1111 1111 0000 0000 1111	x 2^{+3}	Last bit is sticky
+	00.000 0001 1110 0000 0001 1111 101	x 2^{+3}	
+	10.111 0000 0000 1111 1111 0001 000	x 2^{+3}	2's complement
	10.111 0001 1111 0000 0001 0000 101	x 2^{+3}	Sum is negative
−	1.000 1110 0000 1111 1110 1111 011	x 2^{+3}	2's complement
−	1.000 1110 0000 1111 1110 1111 01	x 2^{+3}	Normalize, RS=01
−	1.000 1110 0000 1111 1110 1111	x 2^{+3}	Round

Q3. [7 points] Translate Nested IF statements

Using minimal number of instructions, translate the following nested IF statements into MIPS assembly code. Assume that variables **a**, **b**, **c**, and **d** are **signed integers** stored in registers **$s0**, **$s1**, **$s2**, and **$s3**, respectively. If needed, **you can use pseudo-branch instructions only**.

```
if ( ((a > b) || (c <= d)) && (a == c) ) {
  if (c != d) a = b + c;
  else c = b – 2;
}
```

Solution:

```
   bgt   $s0, $s1, L1       # if (a > b) skip OR
   bgt   $s2, $s3, skip     # if (c > d), skip IF statement
L1:
   bne   $s0, $s2, skip     # if (a != c) skip IF statement
L2:                         # inner if statement
   beq   $s2, $s3, L3       # if (c==d) goto inner else
   addu  $s0, $s1, $s2      # execute inner if
   j     skip               # skip inner else
L3:
   addiu $s2, $s1, -2       # execute inner else
skip:
```

Q4. [14 points] Translate a Recursive Function

Translate the following high-level recursive function **`freq`** into MIPS assembly code. The **`freq`** function counts the number of times an integer **`i`** appears in an array **`A`** of **`n`** integers. The array **`A`** is already in memory. The function parameters are passed in registers **`$a0`**, **`$a1`** and **`$a2`**, respectively. The **`freq`** function returns the count in register **`$v0`**. Use **MIPS convention** in saving and restoring **only the necessary register(s)** in the recursive function. Your MIPS implementation of the **`freq`** function **must be recursive**. Add comments to explain the utilization of registers.

```
int freq(int A[], int n, int i) {
   if (n == 0) return 0;
   int j = 0;
   if (A[n-1] == i) j = 1;
   return (j + freq(&A[0], n-1, i));
}
```

```
freq:
     bnez   $a1,L1              # skip if (n != 0)
     li     $v0,0               # return 0
     jr     $ra
L1:
     li     $t0,0               # initialize j = 0
     addiu  $a1,$a1,-1          # $a1 = (n-1)
     sll    $t1,$a1,2           # $t1 = 4*(n-1)
     addu   $t1,$t1,$a0         # $t1 = &A[n-1]
     lw     $t2,0($t1)          # load $t2 = A[n-1]
     bne    $t2,$a2,L2          # skip if (A[n-1] != i)
     li     $t0,1               # set j = 1
L2:
     addiu  $sp,$sp,-8          # allocate stack frame = 8 bytes
     sw     $ra,0($sp)          # save $ra
     sw     $t0,4($sp)          # save j
     jal    freq                # freq($a0=&A[0], $a1=n-1, $a2=i)
     lw     $t0,4($sp)          # restore j
     addu   $v0,$t0,$v0         # return (j+freq(&A[0], n-1, i))
     lw     $ra,0($sp)          # restore $ra
     addiu  $sp,$sp,8           # release stack frame
     jr     $ra
```

Q5. [7 points] Compute the Sum of Decimal Digits

Write a MIPS function **`sum_digits`** that computes and returns the **sum of decimal digits** in an **unsigned integer**. For example, the sum of decimal digits for **1536** is **1+5+3+6 = 15**. The function **`sum_digits`** receives the unsigned integer argument **in binary** in register **`$a0`**. For example, **1536 = (0000 … 0110 0000 0000)$_{binary}$**. It should extract the decimal digits, compute, and return their sum, **also in binary**, in register **`$v0`**. Hint: divide the unsigned integer by **10** to extract the decimal digits.

Solution:

```
sum_digits:
li     $v0, 0            # $v0 = sum = 0
li     $t0, 10           # divisor = 10
loop:
divu   $a0, $t0          # divide by 10
mfhi   $t1               # $t1 = remainder
mflo   $a0               # $a0 = quotient
add    $v0, $v0, $t1     # add decimal digit
bne    $a0, $zero, loop  # loop if more digits in $a0
jr     $ra               # return
```

Q6. [16 points] Write Loops to Traverse a Matrix

Given that **M** is a square matrix of $N \times N$ integers (N rows by N columns), which is already read and initialized in memory. The **starting address** of matrix **M** is stored in register **`$a0`**, and N is stored in register **`$a1`**. This is always the case for parts (**a**) and (**b**).

a) (7 points) Write MIPS code to compute the sum of all N elements of row i, where $i < N$. The value of i is stored in register **`$a2`**. The sum should be computed in **`$v0`**. Add comments to explain the utilization of registers. Do NOT use pseudo-instructions.

Solution:

```
add    $v0, $0, $0       # $v0 = sum = 0
add    $t0, $0, $0       # $t0 = j = 0
mul    $t1, $a1, $a2     # $t1 = i * N
sll    $t1, $t1, 2       # $t1 = i * N * 4
addu   $t1, $a0, $t1     # $t1 = &M[i][0]
loop:
lw     $t2, ($t1)        # load $t2 = M[i][j]
addu   $v0, $v0, $t2     # sum = sum + M[i][j]
addiu  $t1, $t1, 4       # point to next row element
addu   $t0, $t0, 1       # j++
bne    $t0, $a1, loop    # branch if (j != N)
```

b) (9 points) Write MIPS code to locate the maximum element in column *j*, where $j < N$. The value of index *j* is stored in **`$a2`**. The **maximum unsigned value** of column *j* should be computed in **`$v0`** and its corresponding row index should be computed in **`$v1`**. Add comments to explain the utilization of registers. You may use pseudo-branch instructions.

Solution:

```
add    $v0, $0, $0       # $v0 = max = 0
add    $v1, $0, $0       # $v1 = row = 0
sll    $t0, $a2, 2       # $t0 = j * 4
addu   $t0, $a0, $t0     # $t0 = &M[0][j]
add    $t1, $0, $0       # $t1 = i = 0
sll    $t2, $a1, 2       # $t2 = N * 4 = number of bytes per row
loop:
lw     $t3, ($t0)        # load $t3 = M[i][j]
bleu   $t3, $v0, skip    # skip if less than or equal
addu   $v0, $t3, $0      # $v0 = new max = M[i][j]
addu   $v1, $t1, $0      # $v1 = row index
skip:
addu   $t0, $t0, $t2     # point to next column element
addu   $t1, $t1, 1       # i++
bne    $t1, $a1, loop    # branch if (i != N)
```