Northeastern University
College of Engineering
Department of Electrical & Computer Engineering

EECE2322: Fundamentals of Digital Design and Computer Organization

Summer II 2022 – Homework 3

Submission Instructions

a. For the programming problems (MIPS assembly and Verilog HDL):
   1. Your code must be well commented by explaining what the lines of your program do. Have at least one comment for every 4 lines of code.
   2. At the beginning of your source code files write your full name, students ID, and any special assembling/running instruction (if any).
   3. If required, test your code using the MARS assembler or the Vivado Simulator.

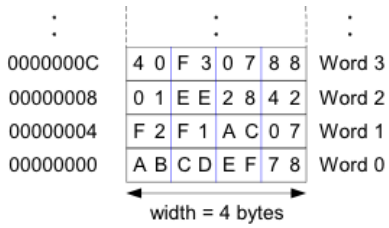
b. For non-programming problems, include explanation of all steps of your answers/calculations not only the final results.

c. Submit the following to the homework assignment page on Canvas:
   1. Your homework report developed by a word processor and submitted as <u>one</u> PDF file. For answers that require drawing and if it is difficult on you to use a drawing application, you can neatly hand draw the answer, scan it, and include it into your report. The report includes the following (depending on the assignment contents):
      a. Answers to the non-programming problems that show all the details of the steps you follow to reach these answers.
      b. A summary of your approach to solve the programming problems.
      c. If required, the screen shots of the sample runs of your programs.

   2. Your well-commented programs <u>source code files</u> (i.e., the .asm or .v files). Do not upload any whole project folders/files that are usually created by Vivado.

   Do NOT submit any files (e.g., the PDF report file and the source code files) as a compressed (zipped) package. Rather, upload each file individually.

<u>Note</u>: You can submit multiple attempts for this homework, however, only what you submit in the last attempt will be graded (i.e., all required reports and files must be included in this last attempt).



Q1 (16 Points)

Find the machine code, represented in hexadecimal, of each of the MIPS instructions in the following program. Explain the detailed steps followed to find each machine code by showing the code of the segments within each instruction (e.g., the operation code, the registers code, shift value, immediate value, etc.)

```
      addi $t0, $t1, -10
      lw $s0, -3($t0)
L1:   beq $t0, $s0, L2
      sw $v0, 84($0)
L2:   addi $t0, $sp, -4
      beq $t0, $s0, L1
```

Q2 (18 Points)

The following are the 32-bit hexadecimal numbers that represent the machine code of six different MIPS instructions. Find the assembly code of each instruction and, for each instruction, determine its source and destination registers, if any exist. Represent these registers by their name (e.g., $t5) not their code number. Explain your answer by segmenting each code to its instruction machine code components (e.g., the operation code, the registers code, shift value, immediate value, etc.)

a. 0x02519020
b. 0x00129080
c. 0xae4a0000
d. 0x22100001
e. 0x1608fff3
f. 0x8e4a0005

Q3 (20 Points)

For the following C code, assume that:

- The $s1 and $s2 registers are used to represent the integer variables i and sum respectively.
- L is an array of integers that has been already initialized with ten integers.
- The base address of L in the memory is 0x55552222.
- Each integer in this problem is stored in a word of 4 bytes.

Convert the following C code to MIPS instructions. The purpose of the code is to calculate the sum of the ten integers stored in L. No need to run your code on MARS.

```
sum = 0;
for (i=0; i<10; i++) {
      sum = sum + L[i];
}
```

```
lui $s0, 0x5555
ori $s0, $s0, 0x2222
addi $s2, $0, 0
addi $s1, $0, 0
addi $t0, $0, 10
L6: slt $t1, $s1, $t0
beq $t1, $0, done
sll $t2, $s1, 2
add $t2, $t2, $s0
add $s2, $s2, $t2
j done L6
```

(Q2) a) 0000 00 10010 10001 10010

op = 0 | rs: $s2 | rt: $s1 | rd: $s2

R type 000000 10000 } add $s2, $s2, $s1

func: add

b) 0000 00 00000 10010 01001

rs: $s0 | rt: $s1 | rd: $s2

R type 00001 000000 } SLL $s1, $s2, 1

shamt: 1 | func: sll

c)

I type 1010 11 10010 01010 00000

op: sw | rs: $s1 | rt $s2

0000 00 0006

immediate

} store word $t1, 0($s1)

d)

I type 0010 00 10000 10000 00000

op: addi | rs: $s1 | rt $s1

0000 00 0001

immediate: 1

} Addi $s1, $s1, 1

e)

I type 0001 01 10000 01000 11111

op: BNE | rs: $s1 | rt: $t1

1111 11 0011

immediate: -13

} BNE $s1, $t1, -13

f)

I type 1000 11 10010 01010 00000

op: LW | rs: $t1 | rt: $s1

0000 00 0101

immediate: 5

} LW $t1, 5($s1)

★ question: how to identify register # so guessed eg: $s0 vs $s1

Q4 (15 Points)

Assume that the variables in the following C code are integers that are stored in the following registers: `$t0 = c, $s0 = x, $s1 = y, $s2 = z`

Convert the following C code to MIPS instructions. No need to run your code on MARS.

```
switch (c) {
  case 1:  x = y + z;
           break;
  case 2:  x = y - z;
           break;
  default: x++;
           break;
}
```

```
beq $t0, $0, L0   // if zero go case 0
lw $t1, 1         // load 1 for comparison
beq $t0, $t1, L1
L0: add $s0, $s1, $s2
    J exit
L1: sub $s0, $s1, $s2
    J exit
default: addi $s0, $s0, 1
exit:
```

Q5 (15 Points)

Assume register `$t0` has been initialized to any random 32-bit binary number. Write the MIPS assembly code that counts the number of 1s in `$t0` and stores that count in `$s0`. No need to run your code on MARS.

```
L0: addi $s0, $0, 0   // setting $s0 to 0
    beq  $t0, $t0, -1 // t0-1
    and  $t0, $t0, $t1,
    addi $s0, $s0, 1
    J L0              // Jumps back to the loop start
```

Q6 (16 Points)

Assume $s1 has been initialized to a signed integer and hence it might have a negative value.

a) The following MIPS code increments $s1 by 1 only if the current value in $s1 is within a specific range. What is this range?

```
        slt  $t1, $s1, $0
        bne  $t1, $0, Done
        slti $t2, $s1, 8
        beq  $t2, $0, Done
        addi $s1, $s1, 1
Done: ....
```

if $s1 < 0, $t1 = 1
$t1 != 0 → Done
if $s1 < 8 $t2 = 1
$t2 = 0 → done
$s1 = $s1 + 1

$0 \leq S1 \leq 7$ or $[0, 8]$ not including 8

b) Explain why the following MIPS code does the same task as the code in part (a) above. Support your explanation with examples.

```
        sltiu $t1, $s1, 8
        beq   $t1, $0, Done
        addi  $s1, $s1, 1
Done: ....
```

it works the same because now set less than is for unsigned integers, whose range is 0 – 4294967295 but our $s1 range is still $0 \leq \$s1 \leq 7$ or $[0, 8]$

while ($s1 >= 0 && < 8)
$s1 = $s1 + 1