ECE 369A

HW1 Solutions (Revised 9/20) Fixed Problem 1 parts b,c, d)

20 Points

Group # ?????			
Name	**Last Name**	**% Effort**	**Lab Section** **A -** 2:00-3:15pm / **B-** 3:30:4:45pm / **C-** 5:00-6:15pm
?	**?**	**?**	**?**
?	**?**	**?**	**?**
?	**?**	**?**	**?**

-2 pt per missing "Group #", "Name", "Last Name", "% Effort", "Lab Section"

Show your work in order to receive full credit.

No credit if you write the final result only (without showing how you derived it)

Include your answers in this document and submit.

Grading:

- Will randomly pick a subset of the questions and scale the overall score to 30 points.
- 8pts per day late penalty

Problem 1: Performance

```
 tomato:
        slt   $t0, $a1, $a2
        beq   $t0, $zero, orange
        sll   $t1, $a1, 2
        add   $t1, $a0, $t1
        sll   $t2, $a2, 2
        add   $t2, $a0, $t2
        add   $t5, $a2, $zero
        andi  $t5, $t5, 1
        bne   $t5, $zero, potato
        lw    $t3, 0($t1)
        add   $t4, $t3 $t3
        sw    $t3, 0($t2)
        sw    $t4, 0($t1)
potato: addi  $a1, $a1, 2
        addi  $a2, $a2, -1
        j     tomato
orange: jr    $ra
```

Instruction Type	Cycles
Arithmetic	2
Logical	1
Loads	8
Stores	6
Conditional branches	3
Unconditional jumps	1

Table 1. Number of clock cycles for each type of instruction

Part (a) Above is the assembly code for the function declared as: tomato(int array[], int x, int y). Assume that "myarray" is an array of 500 integers, and the function is called as tomato(myarray, 5, 23).

(i) How many times is the "**slt**" instruction executed? Justify your answer to receive credit. **(6pts)**

Tracing the values in each iteration
- 5 23
+ 7 22
- 9 21
+ 11 20
- 13 19
+ 15 18
17 17
Loop executes 6 times and last case (17,17) is when the loop exits and slt is executed one last time totaling 7

(ii) How many times is the "**lw**" instruction executed? Justify your answer to receive credit. **(6 pts)?**

Above + indicates cases when lw is executed (when a2 is an even number, 22,20,18)
Note that
add $t5, $a2, $zero
andi $t5, $t5, 1
bne $t5, $zero, potato
checks if the least significant bit of a2 is 1 or not (checking if it is an odd value or not)

Part (b) Calculate the total number of cycles it takes to execute tomato(myarray, 5, 23). Show your work **(10 points)?**

1 slt $t0, $a1, $a2
3 beq $t0, $zero, orange
1 sll $t1, $a1, 2
2 add $t1, $a0, $t1
1 sll $t2, $a2, 2
2 add $t2, $a0, $t2
2 add $t5, $a1, $zero
1 andi $t5, $t5, 1
3 beq $t5, $zero, potato
8 lw $t3, 0($t1)
2 add $t4, $t3, $t3

6 sw $t3, 0($t2)
6 sw $t4, 0($t1)
potato :
2 addi $a1, $a1, 1
2 addi $a2, $a2, -1
1 j tomato
orange: 1 jr $ra

lw, add, sw, sw sequence is executed 3 times, 22 cycles each : 66 cycles
remaining part of the code from slt (first instruction to j tomato excluding the lw, add, sw, sw sequence is executed 6 times for a total of 21*6 = 126 cycles
last slt (7th) + last beq + jr = 5 cycles
Total = 66+126+5 = 197 cycles (Original solution missed 66)

Part (c) What is the execution time in **milliseconds** for the tomato(myarray, 5, 21) function if the processor operates at 60KHz? Show your work **(6 points)**

= 197/60,000 = 3.28ms

Part (d) Your manager claims that it is possible to achieve an overall speedup of 1.5x by optimizing the data memory access time. Is this claim correct? If so how much speed up is necessary for the **data memory accesses** to achieve an overall speedup of **1.5x**? Show your work to get credit. **(10 points)**

You can use execution time and find the new time after 1.5x speedup and solve for speedup on data memory.

Alternative is Amdahl's law.
Calculate the ratio of cycles spent for memory transactions to total number of cycles (f)
Then apply Amdahl's law for an overall speedup of 1.5

You need to take the total number of memory related cycles :
lw(8) + sw(6) + sw(6) = 20 cycles per iteration
executed 3 times
total of 60 cycles

f = 60/197 = 0.305 Speedup= 1/(f/n+ (1-f))
1.5= 1/(0.305/n+ (1-0.695)) =
0.6667 = 0.305/n + 0.695

It turns out that n is a negative value. that means even if we reduce the number of cycles it takes to execute memory type of instructions to 0 we can not get to 1.5X speedup.

Problem 2 (22pts)

Below is a C function that returns the index of the smallest element in an integer array V[], which contains n items. A translation of this function into MIPS assembly language is shown to the right. The index of the minimal element is placed in $v0 as the return value.

```
int minimum(int V[], int n)
{
    int min, i;

    min = V[0];
    for (i = 1; i < n; i++)
        if (V[i] < min)
           min = V[i];
    return min;
}
```

```
minimum:
        lw    $t0, 0($a0)      # min = V[0]
        addi  $t1, $0, 1       # i = 1
loop:
        bge   $t1, $a1, exit   # i >= n ?
        mul   $t2, $t1, 4
        add   $t2, $t2, $a0
        lw    $t2, 0($t2)      # $t2 = V[i]
        bge   $t2, $t0, next   # V[i] >= min ?
        add   $t0, $t2, $0     # min = V[i]
next:
        addi  $t1, $t1, 1      # i++
        j     loop
exit:
        add   $v0, $t0, $0     # return min
        jr    $ra
```

Say that we run this program on a 250MHz processor. The CPIs for different types of MIPS instructions are given below.

Type	CPI
adds	3
mul	12
loads	5
branch and jumps	2

a) Assume that function is called as: mimimum(my_array, 101), and my_array stores 101 integers in descending order (101, 100, 99, …, 3, 2, 1). What would be the exact CPU time for the minimum function in nanoseconds? (10 points)

For an input array 101 elements, the loop body executes 100 times. Since array elements are in descending order, add $t0, $t2, $0 will be executed in each iteration. The bge instruction will fail on the 101st iteration and the function will exit
5 + 3 + 100 (2 + 12 + 3 + 5 + 2 + 3 + 3 + 2) + 2 + 3 + 2 = 3215 cycles
lw+addi+ 100(bge + mul + add + lw + bge + add + addi + j) + bge + add + jr
with a 4ns cycle time , execution time is 12,864 ns.

b) What would be the exact CPU time in nanoseconds if we replaced
mul $t2,$t1,4" with

add $t2, $t1, $t1
add $t2, $t2, $t2
(6 points)
saves 6 cycles per iteration , total of 600 cycles , resulting with 2616 cycles

= 10, 464 ns.

c) Compare the execution times and MIPS values for the original and revised codes. (6pts)

revised version is 12, 864/10,464 = 1.23X better

CPIold = 3216 / (1 + 1 + 100*8 + 1 + 1 + 1) = 3.99

CPInew = 2616 / (1 + 1 + 100*9 + 1 + 1 + 1) = 2.89

MIPSold = 250 / 3.99 = 62.65

MIPSnew = 250 / 2.89 = 86.5

Problem 3:

```
int  foo(v[ ], k, n, m)
       {
        sum = 0:
        for (i=? ;i<? ;i=?)
           sum = sum + v[i];
        return sum;
       }
```

(a) Source C code

```
        add  $t0, $zero, $zero
        add  $t1, $a1, $zero
loop:   bge  $t1, $a2, return
        sll  $t2, $t1, 2
        add  $t2, $t2, $a0
        lw   $t3, 0($t2)
        add  $t0, $t0, $t3
        add  $t1, $t1, $a3
        j    loop
return: add  $v0, $t0, $zero
        jal  $ra
```

(b) Code generate by Compiler 1

```
        add  $v0, $zero, $zero
        sll  $a1, $a1, 2
        sll  $a2, $a2, 2
        sll  $a3, $a3, 2
        add  $t1, $a1, $zero
        bge  $t1, $a2, return
loop:   add  $t2, $t1, $a0
        lw   $t3, 0($t2)
        add  $v0, $v0, $t3
        add  $t1, $t1, $a3
        blt  $t1, $a2 loop
return: jal  $ra
```

(c) Code generate by Compiler 2

Figure 1: Assembly code generated by two compilers for the same segment of C code.

Instr. Type	Instruction Cycles
Arithmetic	4
Logical	2
Loads/Stores	5
conditional branches	3
unconditional jumps	1

Table 1: Number of clock cycles for each type of instruction.

You generated MIPS assembly code for the C code shown in Figure 1(a) using two different compilers. Compiler 1 generated the code in Figure 1(b) while compiler 2 generated the code in Figure 1(c). For the scope of this question only, you should assume that all the instructions used by the compilers are real instructions (as opposed to pseudoinstructions) in the MIPS architecture.

Notes:

bge : branch if greater than or equal to

blt : branch if less than

a) (10 points) Fill the table below with the number of instructions of each type that will be executed by the code generated by compiler 1 and by compiler 2 when the function **foo(v, 16, 100, 16)** is invoked.

Instr. Type	Compiler 1 Code	Compiler 2 Code
Arithmetic	**3N+3**	**3N+2**
Logic	**N**	**3**
Loads	**N**	**N**
conditional branches	**N+1**	**N+1**
unconditional jumps	**N+1**	**1**

b) (10 points) Compute the average number of cycles per instruction (CPI) for each version of the program (call them: P1 and P2 respectively).

N is 6
Cycles P1 = 21*4 + 6*2 + 6*5 + 7*3 + 7*1 = 154
IC P1 = 47
CPI P1 = 3.28

Cycles P2 = 20*4 + 3*2 + 6*5 + 7*3 + 1 = 138
IC P2 = 37
CPI P2 = 3.7

c) (6 points) If the machine that you are using has a processor operating at 1 GHz, Which version of foo(), P1 or P2, is faster? By how much?

=154/138 = 1.12x P2 is faster

Problem 4: Performance

```
tomato:
        slt   $t0, $a1, $a2
        beq   $t0, $zero, orange
        sll   $t1, $a1, 2
        add   $t1, $a0, $t1
        sll   $t2, $a2, 2
        add   $t2, $a0, $t2
        lw    $t3, 0($t1)
        lw    $t4, 0($t2)
        sw    $t3, 0($t2)
        sw    $t4, 0($t1)
        addi  $a1, $a1, 2
        addi  $a2, $a2, -2
        j     tomato
orange: jr    $ra
```

Instruction Type	Cycles
Arithmetic	2
Logical	1
Loads	6
Stores	4
Conditional branches	3
Unconditional jumps	1

Table 1. Number of clock cycles for each type of instruction

Part (a) Above is the assembly code for the function declared as: tomato(int array[], int x, int y). This function involves a loop. How many times is this loop executed if the function is called as **tomato (my_array, 4, 103)**? Assume that "my_array" is an array of 250 integers. Show your work **(5 points)?**

Registers a1 and a2 values are changing by 2, therefore 25 iterations.

Part (b) Based on your answer to part(a), calculate the total number of cycles it takes to execute the tomato function. Show your work **(10 points)?**

Instructions	Operation	Cycle
1	slt	2
2	beq	3
3	sll	1
4	add	2
5	sll	1
6	add	2
7	lw	6
8	lw	6
9	sw	4
10	sw	4
11	addi	2
12	addi	2
13	j	1
14	jr	1

Each iteration executes instructions 1-13, total of 36 cycles per iteration.

Cycle count = 25 iterations* 36 cycles/iteration + 2 (last slt before exiting the loop) + 3 (last beq) + 1 (jr) = 906 cycles

Part (c) Calculate the MIPS for this program if the processor is operating at 2GHz? Show your work **(5 points)**
Time = 906cycles*0.5ns/cycle = 453ns

IC = 25*13 instructions in the loop + 1 (last slt) + 1 (last beq) + 1(jr) = 328

MIPS = 328/ (453*10^-9*10^6) = 724

Part (d) If we find a way to accelerate only the **load word (lw)** operations, how much speed up is necessary for the **lw**, to observe an overall speedup of **1.45x** in the "tomato()" function? Show your work to get credit. **(10points)**

Lw = 25 iterations, 2 lw per iteration, 6 cycles per lw => 300 cycles
Ratio = 300/906 = 0.331

1.45 = 1/ ((0.3/n) + 1-0.331) - Amdahl's law => N=15

Or

Lw = 100 iterations, 2 lw per iteration, 6 cycles per load = 1200 cycles

Ratio = 1200/3606 = 0.333 => 1.45 = 1/ ((0.3/n) + 1-0.331) => N=15

Problem 5 ISA Design
Assume that we would like to reduce the MIPS register file to 16 registers and expand the instruction set to support 100 different R-type of operations.

(a) What should be the minimum number of bits needed by R-type of instructions to support these changes? Show your work by indicating the bitwidth of each field for R-type of instructions **(6pts)**

In MIPS we have 32 registers. To index any of the registers we need 5 bits. To index 16 registers we need 4 bits of address.

R-type of instructions are defined by the func field of the R-type of instruction. MIPS has 6 bits offering 64 different R-type of operations. If we have 100 operations to support, then we need 7 bits (7 bits gives 128 different operations to represent)

Problem is not making changes to the opcpode field (no new type of instruction), therefore remains as 6 bits

Problem is not changing the architecture to any other bitwidth. Therefore we are still working in 32 bit ISA. In this case registers are still 32 bits. For shifting we still 32 bits.

opcode	rs	rt	rd	shmt	func
6	4	4	4	5	7

(b) Reducing the number of registers reduces the code size.

i. Give an argument that shows that this statement is false. **(4pts)**

With reduced number of registers, there is more chance to use stack. This increases the number of push and pop types of operations, resulting with increase in code size for applications that require large number of registers.

ii. Give an argument that shows that this statement is true. **(4pts)**

Each instruction requires 30 bits instead of 32 bits. This reduces the code size for applications that do not stress the register file.

Problem 6: Performance (10pts)
Assume that a design team is considering enhancing a machine by adding MMX (multimedia extension instruction) hardware to a processor. When a computation is run in MMX mode on the MMX hardware, it is 10 times faster than the normal mode of execution. Call the percentage of time that could be spent using the MMX mode the percentage of media enhancement. What percentage of media enhancement is needed to achieve an overall speedup of 2?

We will use Amdahl's Law for this question.

Execution time with Media Enhancement =

(Execution time improved by Media enhancement)/(Amount of Improvement) +

Execution time unaffected

Let x be the percent of media enhancement needed for achieving an overall speedup of 2. Then,

(100)/2 = (x)/10 + (100-x)

Solving for x, we have x = 55.55

Problem 7. Performance (10pts)
A designer wants to improve the overall performance of a given machine with respect to a target benchmark suite and is considering an enhancement X that applies to 50% of the original dynamically-executed instructions, and speeds each of them up by a factor of 3. The designer's manager has some concerns about the complexity and the cost-effectiveness of X and suggests that the designer should consider an alternative enhancement Y. Enhancement Y, if applied only to some (as yet unknown) fraction of the original dynamically-executed instructions, would make them only 75% faster. Determine what percentage of all dynamically-executed instructions should be optimized using enhancement Y in order to achieve the same overall speedup as obtained using enhancement X.

We will use Amdahl's Law for this problem.

Execution time after improvement =

(Execution time affected by improvement)/(Amount of Improvement) +

Execution time unaffected

Execution Time using X = (50)/3 + (100-50) = 66.67

The speedup is given by = (100)/66.67 = 1.5

Let the percentage of dynamically executed instructions to which Y is to be applied be x.

Execution Time using Y = (x)/1.75 + (100-x)

SpeedUp = (100)/(Execution Time using Y) = 1.5

Solving for x, we get x = 77.78

Problem 8. MIPS (8pts) What does func(n,m) do? Write as a function of n and m.

```
func:
        bne     $a1, 1, foo
        move    $v0, $a0
        j       $ra
foo:
        addi    $sp, $sp, -8
        sw      $ra, 0($sp)
        sw      $a0, 4($sp)
        addi    $a1, $a1, -1
        jal     func
        lw      $a0, 4($sp)
        mult    $v0, $v0, $a0
        lw      $ra, 0 ($sp)
        addi    $sp, $sp, 8
        j       $ra
```

$f = n^m$