ECE 369A

HW1

**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 20 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).

1. How many times is the “**slt**” instruction executed? Justify your answer to receive credit. **(6pts)**
2. How many times is the “**lw**” instruction executed? Justify your answer to receive credit. **(6 pts)?**

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

**Part (c)** What is the execution time in **milliseconds** for the tomato(myarray, 5, 23) function if the processor operates at 60KHz?Show your work **(6 points)**

**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)**

**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.

****

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)

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)

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

**Problem 3:**

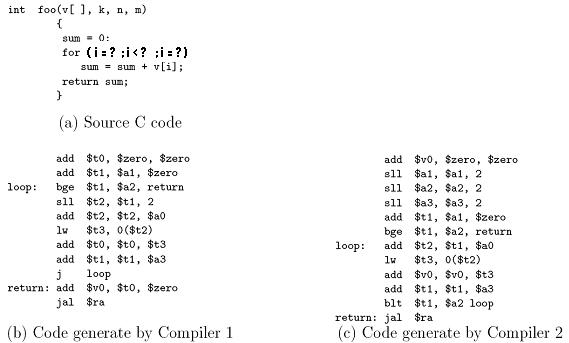


Figure 1: Assembly code generated by two compilers for the same segment of C code.

|  |  |
| --- | --- |
| 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

1. (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 |  |  |
| Logic |  |  |
| Loads |  |  |
| conditional branches |  |  |
| unconditional jumps |  |  |

1. (10 points) Compute the average number of cycles per instruction (CPI) for each version of the program (call them: P1 and P2 respectively).

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?

**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)?**

**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)?**

**Part (c)** Calculate the MIPS for this program if the processor is operating at 2GHz? Show your work **(5 points)**

**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)**

**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.

1. 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)**

7bits 4bits 4bits 4bits 0bits 0bits

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| op | rs | rt | rd | shamt | funct |

For op: 2^7 = 128, so 7 bits to be able to represent 100 different R-Type operations.

For rs, rt & rd: 2^4 = 16, so 4 bits to represent any of the 16 registers.

For shamt: since the shift amount field is not used for R-type operations it can be set to zero to find the minimum number of bits.

For funct: since any of the 100 different R-Type operations can be specified in the op field which is 7 bits, then funct can also be set to zero to find the minimum. Optionally op the 7 bits from op can be split between op and funct, as this wouldn’t affect the ability to represent the 100 R-type operations.

In total, the minimum number of bits needed to support 100 different R-type operations is 19 bits.

1. Reducing the number of registers reduces the code size.
   * 1. Give an argument that shows that this statement is false. **(4pts)**

With a fewer number of registers it becomes much more likely that the program will have to saving and reading from memory very often, leading to more instructions being needed to access and save all the necessary values to run the program and overall a larger code.

* + 1. Give an argument that shows that this statement is true. **(4pts)**

Having a fewer number of registers can simplify register allocation, by having less registers available, programmers are forced to be more selective of how many registers they use and are more likely to reuse registers as opposed to allocating a new register for every temporary variable, which can overall lead to less code.

**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?

**S = 1 / ((f/n) + (1 – f)) => 2 = 1 / ((f / 10) + 1 – f) => .5 = f/10 + 1 – f => -.5 = -(9/10)f**

**f = 10/18 => f= 5/9 \* 100% = 55.555556%**

**To achieve an overall speedup of 2 the percentage of media enhancement must be approximately 55.555556%.**

**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.

**Enhancement X:**

**Sx = (0.5) \* (3) + 0.5 = 2**

**So, the overall speedup of enhancement x will be 2.**

**Enhancement Y:**

**Sy = Sx = 2 = (% with y) \* (1.75) + (% without y) => 200 = P\*1.75 + 100 – P => P = 133.3333%**

**So, for enhancement Y to have the same speedup as enhancement x, enhancement Y would need to optimize approximately 133.33% of the dynamically executed instructions. This means that it is not possible for enhancement Y to achieve the same overall speedup as obtained using enhancement X.**

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

func:

bne $a1, 1, foo #if a1!= 1 go to foo

move $v0, $a0 #v0 = a0

j $ra #return to return address

foo:

addi $sp, $sp, -8 #Make space in the stack for 2 words

sw $ra, 0($sp) #Save return address on stack

sw $a0, 4($sp) #Save a0 on stack

addi $a1, $a1, -1 #a1 -= 1

jal func #jump to func and save return address

lw $a0, 4($sp) #a0 from stack

mult $v0, $v0, $a0 #v0 \*= a0

lw $ra, 0 ($sp) #get return address from stack

addi $sp, $sp, 8 #Clear stack

j $ra #jump to return address

foo and func translated to C pseudo-code:

foo( n, m){

if(m != 1){

Return n \* foo(n, m – 1);

}

Else{

Return n;

}

}

Foo and func work together as a recursive function that output n to the power of m. Foo is the represents the recursive function itself, while func represents the if else statements which returns back to foo until m == 1, where func will return n to end the recursive cycle.