CSC390 (Computer Org. & Arch.)
Exam#1- Spring 2018
SOLUTION

Name:

Student ID:

Q1. Suppose a program requires the execution of 110 million Floating Point (FP) instructions, 200 million Integer (INT) instructions, 80 million Load/Store (L/S) instructions, and 30 million Branch instructions. The CPI for each type of instruction is 4, 3, 12, and 8, respectively. Also assume that the processor executing the program has a 2.8 GHz clock rate.

(i) Find the execution time of the program. [4 pts]

(ii) By how much is the execution time of the program improved if the CPI of INT and FP instruction is reduced by 30%, the CPI of L/S instructions is reduced by 40% and the CPI of Branch instructions is reduced by 20%. [4 pts]

(iii) By how much must we improve the CPI of only Branch instructions if we want the program to run two times faster? [5 pts]

(iv) Suppose we are trying to reduce the execution time by 25% but this leads to an increase of 15% in the CPI of all instructions. What clock rate should we have to get this time reduction? [5 pts]

SOLUTION:

FP Instruction		INT Instruction		L/S Instruction		Branch Instruction		frequency
IC	CPI	IC	CPI	IC	CPI	IC	CPI	2.80E+09
1.10E+08	4.00	2.00E+08	3.00	8.00E+07	12.00	3.00E+07	8.00	
i) EXECUTION TIME	:			8.00E-01	sec			
ii) With all the mentioned CPI reduction, new execution time :						5.34E-01	sec	
thus excution time improved by:	2.66E-01	sec						
iii) Required CPI of L/S to improve the execution time twice:							-2.93E+01	sec
CPI cannot be negative. So it is not possible.								
iv) New frequency:	4.29E+09	Hz						

Q2. Assume that the program in Q1 is parallelized to run over 3 cores, the number of FP, INT, and L/S instructions per processor is divided by 2.5 but the branch instruction per processor remains the same. Find the relative speed up of the 3 processors result relative to the single processor result. [10 pts]

SOLUTION:

With 3 core processors each core will have the following number of instructions:

FP Instruction		INT Instruction		L/S Instruction		Branch Instruction		frequency
IC	CPI	IC	CPI	IC	CPI	IC	CPI	2.80E+09
4.00E+07	4.00	8.80E+07	3.00	3.20E+07	12.00	3.00E+07	8.00	

So, the new execution time after parallelization: 3.74E-01 sec

speed up: 2.14 times

Q3. Consider the following MIPS assembly code which calls a leaf_procedure to perform an arithmetic series operation, n*(n-1)*(n-2)*…..*1, where n is a positive number. The result of this operation will be stored into the variable "fact", as defined in line 6. [42 pts]

Q3_test1_sp18*

```
1  # Q3 (leaf_Procedure)
2  # Call a function to perform fact=n*(n-1)*(n-2)*.....*1
3  # where n is a positive number
4  .data
5  n: .word 5
6  fact: .word   # store the result
7  .text
8  # put the data in the argument registers.
9  # arguments resisters are used to pass-parameters in the procedure (function)
10 lw $a0, n
11 la $s0, fact # load the address of fact in $s0 register
12 # Call the leaf_procedure
13 jal leaf_procedure
14 sw $v0, 0($s0) # store the return value of the procedure in to fact location.
15 j halt
16 #the leaf_procedure
17 leaf_procedure: addi $sp, $sp, -4  # reserve space in the stack to store $s0
18 sw   $s0, 0($sp)  # save $s0 into the stack
19 add  $s0, $a0, $zero #load the value of $a0 to $so
20 loop: addi $a0, $a0, -1  #decrementing the value of $a0 by 1
21 slti $t0, $a0,1    #check if $a0 < 1
22 bne  $t0,$zero, exit #reached end of the loop
23 mul  $s0, $s0, $a0 #Performing n*(n-1)*(n-2)*.....*1
24 j loop #looping
25 exit:add  $v0, $s0, $zero  # put the result in the return resister $v0
26 lw   $s0, 0($sp)  #restore $s0 for the caller
27 addi $sp, $sp, 4  #free-up stack space
28 jr   $ra     #jump back to the calling program
29 halt: nop
```

Figure 1

After assembling the program, the MIPS simulator shows the necessary information, as shown in figure 2, regarding system resources (data segment, text segment, stack-pointer and other registers) used in the program. Observe figure 2 carefully and answer the following question:

i) What would be content of registers $a0 and $s0 after executing line 10 and 11 of figure 1.

$a0 will have the value of n, i.e. 5
$s0 will have the address of "fact" variable in the data segment, which is 0x10010004

ii) What would be the content of program-counter (pc) and the register $ra, after executing line 13 of figure 1.

The $ra register will have the address of the next instruction of line 14, i.e. $ra=0x00400014
Program-counter(pc) will have the starting address of the "leaf_procedure", i.e. 0x0040001c

iii) Observe that the instruction in line 17 of figure 1 (i.e. addi $sp, $sp, -4) is converted to "addi $29, $29,0xfffffffc" as shown in the Basic Column of figure 2. Explain what do these numbers 29 and 0xfffffffc represent?

$29 represents the corresponding register number of $sp and 0xfffffffc represents 2's complement representation of -4

iv) Observe the current value of the stack-pointer ($sp), as shown in figure 2, and tell me what would be the new value of $sp after executing line 17 of figure 1.

The new value of the $sp would be 4 less than the current value (0x7fffeffc-4 = 0x7fffeff8) after executing line 17

v) Why is it necessary to decrement the $sp before you store something into the stack memory location? Explain.

At the beginning when you call a procedure, $sp indicate the highest location of the stack memory location. We cannot store anything above that location. Thus, to store anything into stack memory location, we need to allocate space in the stack memory location by decrementing the $sp. (I.e. if we want to store two words, we need to decrement $sp by 8).

vi) After executing line 18 of figure 1, the content of $s0 will be stored in the stack memory location. In which memory location the value of $s0 will be stored?

$s0 will be stored in to the memory location specified by $sp in line 17, i.e. 0x7fffeff8 to 7fffeffc

vii) Is it necessary to store $s0 into the stack for this program? What would happen if you do not do that operation?

The "leaf_procedure" and the main program both are using $s0. The main program in line 14 uses $s0 to store the result in to "fact" location. Whereas, the "leaf_procedure" is using $s0 to calculate the "mul" in line 23. Thus, when the program execution is transferred to the "leaf_procedure" the value of $s0 will be changed. That is why, it is absolutely necessary to store the value of $s0 in to the stack for this program. Otherwise, the main program will not be able to store the result in to the desired location ("fact").

viii) Write down the machine code of the instruction in line 19 of figure 1. Show all the MIPS fields of the instruction clearly and verify your machine with figure 2.

The MIPS fields of the instruction: add $s0, $a0, $zero
Opcode: 000000 in binary
rs= $a0 (which is numbered as $4). Thus rs= 00100 in binary
rt=$zero (which is numbered as $0). Thus, rt=00000 in binary
rd=$s0 (which is numbered as $16). Thus rd=10000 in binary
shamt = 00000 in binary (for add, sub instruction it is considered as zero)
funct=100000 in binary (for addition operation funct=32 in decimal)
Thus combining all the fields, we get:
0000 0000 1000 0000 1000 0000 0010 0000= **00808020** in hexadecimal

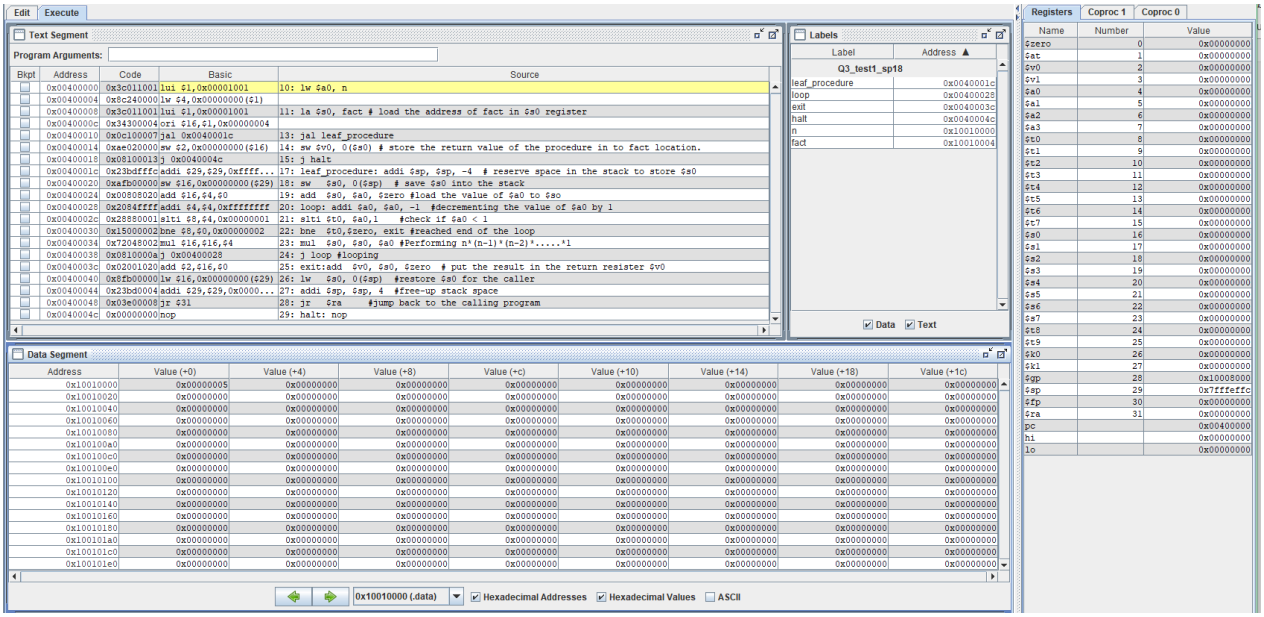

Figure 2 (the larger version of this picture is also attached with this exam)

ix) The leaf_procedure starts a loop operation in line 20 of figure 1. Given the value of n=5, how many times the loop will be executed?

The loop will be executed 5 times.

x) Observe that the instruction in line 21 of figure 1 (i.e. slti $t0, $a0, 1) is checking the value of n (i.e $a0) if it is less than 1. Tell me what would be the value of $t0 after the 2nd iteration and after the 5th iteration?

The initial value of $a0 is 5. In Line 20, $a0 is decremented by 1. Thus after 2nd iteration in line 21, $a0 is not less than 1, i.e. “slti” will not be satisfied. So, $t0 will not be set (i.e. $t0 will 0).

After 5th iteration $a0 becomes less than 1 (i.e. the "slti" condition is satisfied). So the value of $t0 will be set (i.e. $t0 will be 1).

xi) Observe that the branch instruction (bne $t0, $zero, exit) in line 22 of figure 1 is converted to "bne $8, $0, 0x00000002" as shown in Basic column of figure 2. Explain what do these numbers 8, 0 and 0x00000002 represent? Explain.

The number 8 and 0 represent the corresponding register numbers of $t0 and $zero, respectively. 0x00000002 represents, the offset of the label "exit" from the next instruction (i.e. line 23). Observe that the label "exit" is at line 25. Thus, the offset is (25-23=2).

xii) What would be the content of the program-counter (pc), if the condition in line 22 of figure 1 **is satisfied**?

If the branch instruction (bne $t0, $zero, exit) in line 22 of figure 1 **is satisfied**, the value of $pc will be the address of the label, exit, i.e. 0x0040003c

xiii) What would be the content of the program-counter (pc), if the condition in line 22 of figure 1 is **not satisfied**?

If the branch instruction (bne $t0, $zero, exit) in line 22 of figure 1 is **not satisfied**, the value of $pc will be the address of the next instruction, i.e. 0x00400034

xiv) If the condition in line 22 is satisfied, then the instructions in line 23 will be skipped. Can you guess the value of $s0 when the "bne" condition in line 22 is satisfied?

When the "bne" condition in line 22 is satisfied, the program has completed the 5th iteration. That means the loop has calculated the desired value of the factorial operation (i.e. n(n-1)(n-2)….1). Thus $s0 will have the results of 5! When the "bne" condition is satisfied.

xv) Observe that the instruction in line 24 (i.e. J loop) is converted to "j 0x00400028" as shown in Basic column of Fig 2. Explain what does the number 0x00400028 represent?

0x00400028 represents the physical address of the label "loop". Observe that the line 20 (I.e. loop: addi $a0, $a0, -1) has the physical address (address column of figure 2) of 0x00400028. Note that the instruction format of a jump instruction has 26 bits for the physical address for the label in the jump instruction (here, "J loop").

xvi) What would be the value of $v0 after executing line 25 of Figure 1?

$v0 will have the result of the factorial operation (i.e. 5!, for our case).

xvii) What would be the content of register, $s0, **before and after** executing line 26 of figure 1?

Before executing line 26 the $s0 will have the result of the factorial operation (i.e. 5!).
After executing line 26, $s0 will have the address of "fact" variable in the data segment, which is 0x10010004.

xviii) What is purpose of incrementing stack-pointer in line 27? What would happen if you do not do that operation?

The purpose is to free-up the stack space. It is always a good practice to free-up the stack after you use it. Otherwise, stack memory spaces might not be available for other programs.

xix) What would be the content of the program-counter(pc) after executing line 28?

The program-counter(pc) will have the content of $ra i.e. the return address to the main program (0x00400014)

xx) Calculate the physical location of the data segment where the result ($v0) will be stored.

The physical location of the data segment where $v0 will be stored is the location of variable "fact", i.e. 0x10010004

xxi) What would be the content of the program-counter after executing line 15 of figure 1?

The last line of the code segment, 0x0040004c