THE UNIVERSITY OF BRITISH COLUMBIA

CPSC 318 Spring 2004

Machine Structure

Midterm

Student Name:______Solution______ OPEN BOOK - 80 minutes

Student ID: __________________________ *Instructor:* Dr. SON VUONG

I. T/F Questions [20%] (2.5 points/each)

Answer True (T) or False (F), fill in, or select the correct answer (a, b, c, d, e, f or g) to the following:

__d___1. The following law states that "chip density doubles every 18 months":
(a) Amdahl's (b) Intel (c) IBM (d) Moore's (e) Cray's (f) Bush's (g) none of those

_____2. Which of the following: (a) Processor speed, (b) Memory capacity, (c) Disk capacity doubles every: (i) 1 year: ___c____ (ii) 1 ½ years: ___a_____ (iii) 2 years___b____

__ F___3. Datapath refers the CPU bus that allows data to be transferred between the CPU components,

__All___4. Which of the following registers may need to be saved by a recursive function?

$s0, $sp,** **$v0, $t0, $a0, $ra

All saved: recursive function=both Caller and Callee

__F___5. MIPS is a good performance measure when comparing machines with the same instruction set. **MIPS = ClockRate / (CPI * 10 **6)** **CPI can be different**

__F___6. Clock rate is also always a good performance measure, just like MIPS, when comparing machines with the same instruction set architecture.

__T__7. Relative performance would be best represented by geometric means of execution times.

__F___8. Incrementing an integer pointer can be realized by an **addi** instruction in MIPS:
addi $s0, $s0, 1 where $s0 contains the pointer (e.g. address of the interger).

II. Benchmarking [15%]

As an expert in computer architecture, you are contracted to develop a new benchmark suite. Extensive market research shows that your users care about performance on only three programs: a word processor called Warp, a spreadsheet program called Expel, and a web browser called Nutscape. The table below shows the runtimes for three different processors running each of the three target programs on a standard workload:

Program	Weight	Processor A	Processor B	Processor C
Warp	x	8	10	23
Expel	y	30	16	24
Nutscape	z	16	10	22
Arithmetic mean		54/3=18	36/3=12	69/3=23

(a) (6%) Compute the arithmetic mean runtime for each processor and enter the result in the last row of the table above? Which machine performs best in terms of arithmetic mean runtime?
Processor B performs best

The manufacturer of Processor C has been falsely advertising their processor as being "faster than A" and up to two times faster than B. The manufacturer of Processors A and B wants you to come up with a benchmark based on these three programs that shows that Processor A is 20% faster (speedup of 1.2) than the competitor's Processor C, and that high-end Processor B is twice as fast as Processor C (not A). The advertising department wants to keep things simple, so they've asked that your benchmark be a weighted arithmetic mean of the runtimes on the three programs.

(b) (9%) Show the weight (percentage) for the three programs (Warp, Expel, Nutscape) in the second column of the table above. (Show your work in the space below)

$$1.2\,(8x + 30y + 16z) = 23x + 24y + 22z \quad (1)$$
$$1.2\,(10x + 16y + 10z) = 23x + 24y + 22z \quad (2)$$

$$12y = 13.4x + 2.8z \quad (3)$$

$8y = 3x + 2z$ (4)

$X + y + z = 1$ (5)

Solve: <u>z = .78, y = .20, x = .02</u>

<u>Note:</u> If B performs twice as fast as A
No workload exists that satisfies this requirement.

III. MIPS Assembly [30%]

(a) **(10%)** Make the following program **smaller** by eliminating (crossing out) unnecessary lines (instructions) in function **f**.

```
__start:                    /#  Main program
      li      $a0, 6        /#  prepare argument $a0=6
      jal     f             /#  call function f:  f(6)
      …                     /#  print or save $v0

f:
      ~~addi    $sp, $sp, -12 /#  adjust stack for 3 items~~
      ~~sw      $ra, 8($sp)   /#  save the return address~~
      ~~sw      $s0, 4($sp)   /#  save the variable $s0~~
      addi    $v0, $0, 1    /#  initialize answer
      addi    $t1, $0, 1    /#  initialize temp $t1
      addi    $t2, $0, 1    /#  initialize temp $t2
loop: slti    $t3, $a0, 1   /#  test for n < 1
      beq     $t3, $0, L1   /#  if n>=1, go to L1
      ~~lw      $s0, 4($sp)   /#  restore the variable $s0~~
      ~~lw      $ra, 8($sp)   /#  restore the return address~~
      ~~addi    $sp, $sp,12   /#  release  stack frame~~
      jr      $ra           /#  return to "after jal"
L1:   add     $v0, $t1, $t2 /# compute result
      add     $t2, $0, $t1  /# update $t2
      add     $t1, $0, $v0  /# update $t1
      addi    $a0, $a0, -1  /#  n >=1; decrement $a0
      j       loop
```

(b) **(10%)** Explain briefly what function **f** does and what value is returned (in $v0) after the call to this function "**jal f**" in the main program.

- **Compute Fibonacci sequence**
- **Return $v0 = Fib(6) = 21**

(c) **(10%)** What would be the execution time of function **f(n)** for large n = 1000**?** Assume the program runs on a 500-MHz MIPS machine with the following CPI for various instruction types:

Instruction Type	# Instructions for each type	CPI
Jump (**j, jr, beq**)	**2n**	1
Compare (**slt, slti**)	**1n**	2
Arithmetic **(add, addi, sub)**	**4n**	2.5
Memory transfer (**lw, sw**)	0	4

ExecTime = (1*2n + 2*n + 2.5*4n)/500 = 14n/500
= 28 microsec

IV. MIPS Assembly (Recursive Function) [35%]

Given the familiarMIPS assembly code for the recursive function to compute the fibonaci sequence **Fib(n)** as shown below. This function is called by a main program, which is partly shown below.

```
            .text   0x00200000     /# Fib at addr 00200000

      Fib:  addi  $sp, $sp, -12  /# Make room stack frame
            sw    $ra, 8($sp)      /#
            sw    $s0, 4($sp)      /#

            addi  $v0, $0, 1       /#
            beq   $a0, $0, fin     /#
            addi  $t0, $0, 1       /#
            beq   $a0, $t0, fin    /#
            addi  $a0, $a0, -1     /#
            sw    $a0, 0($sp)      /#
            jal   Fib              /#

            lw    $a0, 0($sp)      /#
            addi  $a0, $a0, -1     /#
            add   $s0, $v0, $0     /#
            jal   Fib              /#
            add   $v0, $v0, $s0    /#

      fin:  lw    $s0, 4($sp)      /#
            lw    $ra, 8($sp)      /#
            addi  $sp, $sp, 12     /#
            jr    $ra              /#

            .text   0x00400000  /# Main at addr 00400000
     Main:  addi  $a0,$0, 4        /# Prepare argument
            jal   Fib              /# Call function Fib
            . . .
```

(a) **(8%)** List the register(s) used in the MIPS assembly code above, if any, that are saved (in stack) as the recursive function **Fib** above is executed as a Caller, and as a Callee.

As Caller: **$ra, $a0**

As Callee: **$s0 ($sp not explicity saved)**

(b) **(12%)** Show the content of the stack just after the instruction **add $v0, $v0, $s0** at line **16** is executed **the first time**. Assume sp=0x7FFF FFFC at line 21 (before calling function **Fib** from Main). What are the values of of **$v0** and **$s0** at that time (after the add instruction at line 16)?

SP =	
7FFF FFFC	**Ra1= 0040 0008**
7FFF FFF8	**$s0 = 0 don' tcare**
7FFF FFF4	**$a0 = n-1 = 3**
7FFF FFF0	**Ra2= 0020 0028**
7FFF FFEC	**$s0 = 0 don' tcare**
7FFF FFE8	**$a0 = n-1 = 2**
7FFF FFE4	**Ra2= 0020 0028**
7FFF FFE0	**$s0 = 0 don' tcare**
7FFF FFDC	**$a0 = n-1 = 1**
7FFF FFD8	**Ra3= 0020 0038**
. . .	**$s0 = 1**
	$a0 = n-1 = 0

(when $a0=4)

Answer

$s0 = 1

$v0 = 1 + 1 = 2

Pop off upon return

(c) (8%) What is the execution time for function **Fib(n)** for large n = 1000. Assume this program runs on the same MIPS machine as in Question IIIc (with 500MHz clock and the same CPIs for various instruction types).

T(0) = 9 F(0)

T(1) = 11 F(1)

T(2) = T(1) + T(0) + (10+4+5)

= 11 F(1) + 10F(0) + 19

T(3) = T(2) + T(1) + 19

= 11 F(1) + 10F(0) + 19 + 11 F(1) + 19

> 10(F(1) + F(2)) + 2*19 = 10 F(3) + 2*19

T(n) = T(n-1) + T(n-2) + (n-1)*19 (instructions)

T(n) > 10*F(n) + (n-1)*19

F(n) > 2n/2**

T(n) > 10 * (2n/2) instr**

> 2 cycles/intr * 2 ns/cycles *10* (2n/2)**

T(n) > 40 * (2n/2) ns**

T(100) > 40 * (21000/2) = 40 * 2**500**

> 40*(210*50) ~ 40*(10**3*50)**

T(100) > 40*(10150) ns = 40*(10**141) sec**

1 year = 365x 24 x 3600 s = 30,024,000 s

T(100) > O(10135) years = O(10** 132) centuries**

(d) **(7%)** What would be the problem in computing **Fib(n)** for n=1000 using the program above?

- **Expotential Execution Time: O(n**2)**
- **Number exponentially big O(n**2) requiring 500 bits for n = 1000 – can't fit into 32-bit word.**

- Space (stack): 12n bytes = O(n) wasteful, but not a serious problem.