FACULTY OF ENGINEERING AND ARCHITECTURE
AMERICAN UNIVERSITY OF BEIRUT
EECE 321 – COMPUTER ORGANIZATION

SPRING 2005

PROF. MANSOUR | **MIDTERM 1** | MARCH 29, 2005

NAME: ________________________ ID: __________________________

INSTRUCTIONS:
• **OPEN BOOK/OPEN NOTES/COMPUTERS ALLOWED**
• **TIME: 3 HOURS**
• **WRITE YOUR NAME AND ID NUMBER IN THE SPACE PROVIDED ABOVE.**
• **WRITE YOUR ANSWERS ON THE QUESTION SHEET.**
• **THE SCRATCH BOOKLET WILL NOT BE CONSIDERED IN GRADING.**
• **WRITE COMMENTS NEXT TO YOUR MIPS INSTRUCTIONS.**
• **PAGE 2 LISTS SOME COMMON MIPS INSTRUCTIONS YOU CAN USE**
• **BE AS CLEAR AND AS NEAT AS POSSIBLE.**
• **WRITE DOWN ANY ASSUMPTIONS YOU USE IN SOLVING ANY PROBLEM.**

PROBLEM	MAX POINTS	GRADE
1	20	
2	20	
3	20	
4	25	
5	20	
6	20	
7	10	
8	25	
TOTAL	160	

MIPS Instructions

These are some of the most common MIPS instructions and pseudo-instructions, and should be all you need. However, you are free to use <u>any</u> valid MIPS instructions or pseudo-instruction in your programs.

Category	Example Instruction	Meaning

Arithmetic	`rem $t0, $t1, $t2`
	`div $t0, $t1, $t2`
	`mul $t0, $t1, $t2`
	`addi $t0, $t1, 100`
	`sub $t0, $t1, $t2`
	`add $t0, $t1, $t2`
	`addu $t0, $t1, $t2`
	`addiu $t0, $t1, 100`
		`$t0 = $t1 mod $t2`
		`$t0 = $t1 ⁄ $t2`
		`$t0 = $t1 × $t2`
		`$t0 = $t1 + 100`
		`$t0 = $t1 – $t2`
		`$t0 = $t1 + $t2`
		`$t0 = $t1 + $t2 # unsigned`
		`$t0 = $t1 + 100 # unsigned`

Logic	`or $t0, $t1, $t2`
	`and $t0, $t1, $t2`
	`nor $t0, $t1, $t2`
	`xor $t0, $t1, $t2`
		`$t0 = $t1 or $t2`
		`$t0 = $t1 and $t2`
		`$t0 = $t1 nor $t2`
		`$t0 = $t1 xor $t2`

Register Setting	`li $t0, 100` `move $t0, $t1`	`$t0 = 100` `$t0 = $t1`

Data Transfer	`sw $t0, 100($t1)`
	`lw $t0, 100($t1)`
	`l.s $f0, 100($t1)`
	`s.s $f0, 100($t1)`
	`l.d $f0, 100($t1)`
	`s.d $f0, 100($t1)`
		`Mem[100 + $t1] = $t0`
		`$t0 = Mem[100 + $t1]`
		`$f0 = Mem[100 + $t1]`
		`Mem[100 + $t1] = $f0`
		`$f0 = Mem[100 + $t1], $f1 = Mem[104 + $t1]`
		`Mem[100 + $t1] = $f0, Mem[104 + $t1] = $f1`

Branch	`blt $t0, $t1, Label`
	`ble $t0, $t1, Label`
	`bgt $t0, $t1, Label`
	`bge $t0, $t1, Label`
	`bne $t0, $t1, Label`
	`beq $t0, $t1, Label`
		`if ($t0 < $t1) go to Label`
		`if ($t0 ≤ $t1) go to Label`
		`if ($t0 > $t1) go to Label`
		`if ($t0 ≥ $t1) go to Label`
		`if ($t0 ≠ $t1) go to Label`
		`if ($t0 = $t1) go to Label`

Set	`slti $t0, $t1, 100`
	`sltiu $t0, $t1, 100`
	`slt $t0, $t1, $t2`
	`sltu $t0, $t1, $t2`
		`if ($t1 < 100) then $t0 = 1 else $t0 = 0`
		`if ($t1 < 100) then $t0 = 1 else $t0 = 0`
		`if ($t1 < $t2) then $t0 = 1 else $t0 = 0`
		`if ($t1 < $t2) then $t0 = 1 else $t0 = 0`

Jump	`jal Label`
	`jr $ra`
	`j Label`
		`$ra = PC + 4; go to Label`
		`go to address in $ra`
		`go to Label`

<u>Problem 1</u>: General Questions [20 points]

a) Let the number ***y*** = **`0x4CDD0000`** represent a single precision IEEE 754 number. Represent ***y*** in double-precision IEEE 754 format. Write your result in hexadecimal [**5 points**]

<u>Answer</u>:

b) A frequently used instruction in multimedia applications is the **`swap $t1,$t2`** instruction which swaps the upper and lower16 bits of **`$t2`** and stores the result in **`$t1`**. Implement the **`swap`** instruction in MIPS using no more than 3 instructions (no pseudo-instructions allowed). [**5 points**]

<u>Answer</u>:

c) Assume variables **`u`** and **`v`** are two integers represented in 2's complement notation and that **`v`** is positive. Implement the following C expression using exactly <u>three</u> instructions (no pseudo-instructions): [**5 points**]

```
if ( 0 ≤ u < v )  u = u + v;
```

<u>Answer</u>:

d) Represent the <u>decimal</u> number -8.875 x 2^3 in IEEE 754 single precision format. Give your result in hexadecimal format. [**5 points**]

<u>Answer</u>:

<u>Problem 2</u>: Set-Less-Than [20 points]

Suppose the instructions **`slt`**, **`sltu`**, **`slti`**, **`sltiu`** were removed from the MIPS instruction set. Show how to perform **`slt $s0,$s1,$s2`** using the modified instruction set in which **`slt`** is not available (no pseudo-instructions allowed). Beware to account for overflow.

<u>Solution</u>:

<u>Problem 3</u>: Understanding MIPS Programs [20 points]

Translate the function AMD into C. You should follow all MIPS function conventions. Your code should be as concise as possible, without any gotos or pointers [1**5 points**]. Describe what the function does. [**5 points**]

```
AMD:        li $t0, 0
            li $t1, 0
            li $t2, 0x3fffffff

ALPHA:      bge $t0, $a1, POWERPC
            mul $t3, $t0, 4
            add $t3, $a0, $t3
            lw  $t3, 0($t3)
            ble $t3, $t1, PS2
            move $t1, $t3

PS2:        bge $t3, $t2, SPARC
            move $t2, $t3

SPARC:      addi $t0, $t0, 1
            j ALPHA

POWERPC:    sw $t1, 0($a2)
            sw $t2, 0($a3)
            jr $ra
```

<u>Solution</u>:

<u>Problem 4</u>: MIPS Programming [25 points]

The goal of this problem is to write a MIPS <u>function</u> *flipimage* which flips an image horizontally. For example, a simple image is shown on the left, and its flip is shown on the right.

A picture is composed of individual dots, or pixels, each of which will be represented by a <u>single byte</u>. The entire two-dimensional image is then stored in memory row by row. For example, we can store a 4 x 6 picture in memory starting at address 1000 as follows:

- The first row (consisting of 6 pixels) is stored at addresses 1000-1005.
- The second row is stored at addresses 1006-1011.
- The third row is stored at 1012-1017
- The last row is stored at addresses 1018-1023.

a) Write a MIPS function *fliprow* to flip a *single* row of pixels. The function has two arguments, passed in **`$a0`** and **`$a1`**: the address of the row and the number of pixels in that row. There is no return value. Be sure to follow all MIPS calling conventions. [**10 points**]

MIPS Code	Comments

b) Using the *fliprow* function, you should now be able to write *flipimage*. The arguments will be:

- The memory address where the image is stored
- The number of rows in the image
- The number of columns in the image

Again, there is no return value, and you should follow normal MIPS calling conventions. [**15 points**]

MIPS Code	Comments

<u>Problem 5</u>: Recursive Functions [20 points]

Compile the following function into MIPS. You must follow all MIPS conventions. Add comments to your code. You will be graded on the efficiency of your code.

```
int function(int x){
      if (x == 0)
            return 0;
      int temp = x & 1;
      return temp + function(x >> 1);
}
```

MIPS Code	Comments

Problem 6: Maximum [20 points]

Implement the instruction **`max $t1,$t2,$t3`** in MIPS, which returns the maximum of **`$t2`** and **`$t3`** in **`$t1`**, without using any conditional branch instructions (**`beq`**, **`bne`**). Also, you are not allowed to use pseudo-instructions.

Solution:

Problem 7: Speedup [10 points]

We are interested in determining the speedup (S) gained from a certain improvement made to enhance a computer. The improvement targets two classes of instructions which roughly constitute a fraction p_1 and p_2, respectively, of the overall instructions used. The first class of instructions runs x_1 times faster with the improvement, while the second runs x_2 times faster. Determine S in terms of p_1, p_2. x_1, x_2.

Solution:

<u>Problem 8</u>: Square Function [20 points]

Let **`A`** be a 30x20 array of single-precision floating point numbers whose base address in memory is stored in register **`$a0`**. The elements of **`A`** are stored in <u>column-major</u> form in memory. Write a function in MIPS that squares the elements of **`A`** and stores the resulting array at memory address **`$a1`** in <u>column-major</u> form. Be sure to follow all MIPS calling conventions. Comment your code.

MIPS Code	Comments