M. MANSOUR FACULTY OF ENGINEERING AND ARCHITECTURE
AMERICAN UNIVERSITY OF BEIRUT
SPRING TERM 2004
MIDTERM I EE/CCE 2006

EECE 321 - MICROPROCESSOR SYTEMS

APRIL 8, 2004

NAME: ____________________ ID: ____________________ Section: 9 AM or 10 AM

OPEN BOOK/OPEN NOTES (3 Hours)

WRITE YOUR NAME AND ID NUMBER IN THE SPACE PROVIDED ABOVE.

PROVIDE YOUR ANSWERS IN THE SPACE PROVIDED ON THE QUESTION SHEET.

THE SCRATCH BOOKLET WILL NOT BE CONSIDERED IN GRADING.

BE AS CLEAR AND AS NEAT AS POSSIBLE.

WRITE COMMENTS NEXT TO YOUR MIPS INSTRUCTIONS.

PAGE 2 LISTS SOME COMMON MIPS INSTRUCTIONS YOU CAN USE

CIRCLE YOUR SECTION TIME.

Problem	Points	Over
1		20
2		20
3		20
4		20
5		20
6		20
7		20
8		20
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 any 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	$t0 = $t1 mod $t2 $t0 = $t1 ∕ $t2 $t0 = $t1 × $t2 $t0 = $t1 + 100 $t0 = $t1 – $t2 $t0 = $t1 + $t2
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)	Mem[100 + $t1] = $t0 $t0 = Mem[100 + $t1]

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
	slt $t0, $t1, $t2
		if ($t1 < 100) 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>: (20 points)

Translate the following switch statement in C into MIPS. Try to use as few registers as possible. Assume the following mapping: **`$s1:x, $s2:y, $s3:z, $s0:a`**. <u>Add appropriate comments to your MIPS code</u>.

```
switch(a){
     case 1: x = y + z;
     case 2: x = y - z;
     default: x = z + z;
}
```

Answer:

<u>Problem 2</u>: (20 points)

```
func:
     addi $t0,$a2,1
loop:
     bge  $t0,$a1,exit
     mul  $t1,$t0,4
     add  $t1,$t1,$a0
     lw   $t2,0($t1)
     sub  $t1,$t1,4
     sw   $t2,0($t1)
     addi $t0,$t0,1
     j    loop
exit:
     jr   $ra
```

a) Translate the above MIPS assembly code into C. Your code should be as concise as possible, without any gotos or explicit pointers. Specify how the variables you use in your C code correspond to the above registers. (15 points)

Answer:

b) Describe briefly, in English, what this function does. (5 points)

<u>Problem 3</u>: (20 points)

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. Registers **`$a0`** and **`$a1`** correspond to **`V[]`** and **`n`**. The index of the minimal element is placed in **`$v0`** as the return value.

```c
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
```

Assume that this program is run on a 500MHz processor. The CPIs for different types of MIPS instructions are given below. You can assume that **`mul`** and **`bge`** each count as one instruction.

a) Assume **`minimum`** is passed an array with the numbers from 101 to 1 in descending order (101, 100, 99,… 3, 2, 1). What would be the exact CPU time for the function call, in nanoseconds? (10 points)

Instruction type	CPI
adds	4
mul	10
loads	5
branches and jumps	3

Answer:

b) Assume now that `mul $t2,$t1,4` is replaced by the following two instructions:

```
add  $t2,$t1,$t1
add  $t2,$t2,$t2
```

What would be the exact CPU time in nanoseconds if we made this modification to the original code, and called the **`minimum`** function with the same input array 101, 100, 99, …, 3, 2, 1? What is the performance increase or decrease of the modified code in Part (b) compared to the original unmodified code? (10 points)

Answer:

Problem 4: (20 points)

Consider the following C function **`sum`** which takes an integer argument **`n>0`**. Translate the function as it is into MIPS without doing any modifications to the code. Add comments to your code to make it more readable.

```
int sum(int n){
     if n <= 1
          return 1;
     else
          return n + sum( sum(n-1) – sum(n-2) );
}
```

```
SUM:
















EXIT:
```

<u>Problem 5</u>: (20 points, 5 points each part)

a) Convert **`1 10000011 00010010000000000000000`** from IEEE 754 to decimal:

b) Convert **`2.2`** from decimal to **single** precision IEEE 754 representation. Express your result using hexadecimal numbers.

c) Convert **`2.2`** from decimal to **double** precision IEEE 754 representation. Express your result using hexadecimal numbers.

d) Let $\mathbf{a = 1x2^{0}, b = 1x2^{0}, c = 1x2^{24}}$. Assuming single precision IEEE 754 representation, determine **`(a+b)+c`**, **`(a+c)+b`**, and **`(b+c)+a`**.

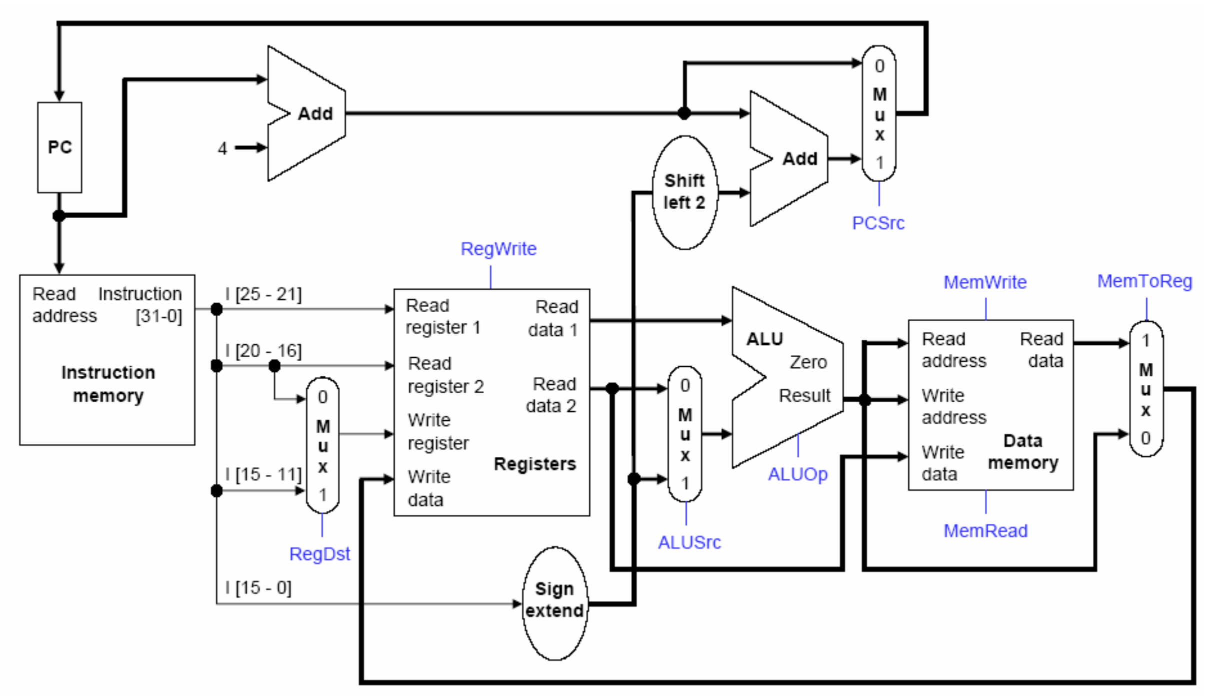
Problem 6: (20 points)

a) Write a minimum-length MIPS instruction sequence that writes an integer value stored in **`$t0`** repeatedly **`$t2`** times in memory starting from the word address pointed to by **`$t1`**. (10 points)

b) Can you write a sequence of MIPS instructions that exchanges the contents of **`$t0`** and **`$t1`** without using any other registers? If so, write a minimal instruction sequence that uses only **`$t0`** and **`$t1`**. If no, justify your answer. (10 points)

<u>Problem 7</u>: (20 points)

Modify the single-cycle datapath shown below to support the **`jal`** instruction. First write down the necessary modifications, and then indicate the modifications on the datapath. You are allowed to modify the existing functional units and/or add extra connections. <u>To receive full credit, all your modifications should be clearly indicated on the figure below</u>.

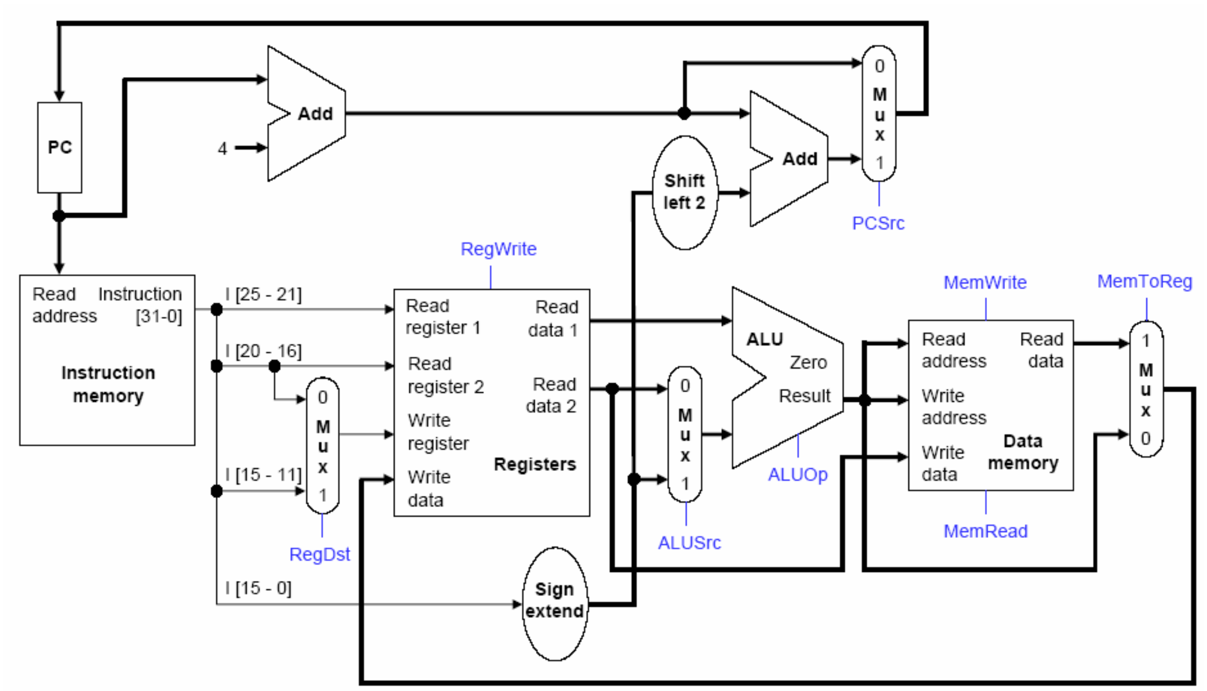


Problem 8: (20 points)

Modify the single-cycle datapath shown below to support the following instruction:

`exchange $s0, $s1`

which exchanges the contents of **`$s0,$s1.`** First write down the necessary modifications, and then indicate the modifications on the datapath. You are allowed to modify the existing functional units and/or add extra connections. To receive full credit, all your modifications should be clearly indicated on the figure below.



PC
Add
4
Add
Shift left 2
0
M
u
x
1
PCSrc
RegWrite
Read address
Instruction [31-0]
Instruction memory
I [25 - 21]
I [20 - 16]
I [15 - 11]
I [15 - 0]
0
M
u
x
1
RegDst
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Sign extend
0
M
u
x
1
ALUSrc
ALU
Zero
Result
ALUOp
MemWrite
Read address
Write address
Write data
Read data
Data memory
MemRead
MemToReg
1
M
u
x
0