ELE 375 / COS471B, COS 471A Midterm
Fall 2003

Prof: David August
TAs: Neil Vachharajani
David Penry

For grading
↓

Question	Score	
I	20 /20	DA
II	20 /20	NV
III	20 /20	DA
IV	20 /20	DP
TOTAL	80 /80	

Please write your answers clearly in the space provided. For partial credit, show all work. State all assumptions. You have 1 hour and 20 minutes for this exam. This midterm is closed book. Only one two-sided, handwritten 8.5x11 sheet is allowed. Put your name on every page. Write out and sign the Honor Code pledge before turning in the test. "*I pledge my honor that I have not violated the Honor Code during this examination.*"

NAME: Sol Ution

COURSE (circle one): COS471B/ELE375 COS471A

HONOR CODE:

QUESTION I:

Answer the following questions. Circle your final answer. Consider the following assembly code for parts 1 and 2. (Assume no branch delay slot in this architecture.)

```
     r1 = 99
  Loop:
     r1 = r1 - 1
     branch r1 > 0, Loop
     halt
```

1. During the execution of the above code, how many dynamic instructions are executed?

r1 = 99 is executed once

$\left.\begin{array}{l}\text{r1 = r1 - 1}\\ \text{branch r1 > 0, Loop}\end{array}\right\}$ executed 99 times

halt is executed once

1
99
99
1
$\boxed{\text{200 insts}}$

2. Assuming a standard unicycle machine running at 100KHz, how long will the above code take to complete?

$$200 \text{ inst} \times \frac{1 \text{ cycle}}{\text{inst}} \times \frac{1 \text{ Sec}}{100{,}000 \text{ cycles}} = \frac{1}{500} \text{ Sec} = \boxed{2 \text{ ms}}$$

3. Using our 8-bit IEEE 754 wimpy precision floating point with three (3) exponent bits and four (4) mantissa bits, show the representation of -11/16 (-0.6875).

$$\text{Bias} = 2^{n-1}-1 = 2^2-1 = 3$$

$$\frac{+11}{16} = \frac{8}{16} + \frac{2}{16} + \frac{1}{16} = \frac{1}{2} + \frac{1}{8} + \frac{1}{16} \Rightarrow 0.\overset{\frac{1}{2}}{1}\underset{\frac{1}{4}}{0}\overset{\frac{1}{8}}{1}\underset{\frac{1}{16}}{1}{}_2$$

implied

$$1.011 \times 2^{-1}$$

$$\text{Exp} = 3-1 = 2$$

$$\boxed{10100110} \leftarrow \underbrace{1}_{\text{Sign}}\,\underbrace{0\,1\,0}_{\text{Exp}}\,\underbrace{0\,1\,1\,0}_{\text{Mant}}$$

negative = 1

4. What is the smallest positive (not including +0) representable number in our 8-bit IEEE 754 wimpy precision floating point? Show the bit encoding and the value in base 10 (fraction or decimal OK).

Smallest Denorm. not +0

positive Denorm Smallest mant.

$$\boxed{0\ 000\ 0001}$$

Bias = 3 Denorm exp is same as $001 \Rightarrow 1_{10}$

$\text{Exp} = 1 - 3 = -2_{10}$

implied 0 in denorms

$0.0001 \times 2^{-2} \Rightarrow 2^{-6} = \boxed{\frac{1}{64}}$

5. Consider a unicycle machine implementation in which 40% of every cycle is spent performing instruction memory fetch. You invent and implement a technique which cuts fetch time in half. **Quantitatively**, what effect did this optimization have on latency and throughput?

Latency is $0.80 \left(1 - \frac{40\%}{2}\right)$ or $\boxed{20\% \text{ shorter}}$

Throughput is $\frac{1}{1 - 0.4 + \frac{0.4}{2}} = \frac{1}{0.8} = \frac{5}{4} = \boxed{1.25\text{x more}}$

$$\text{Speedup} = \frac{1}{1 - f + \frac{f}{s}}$$

6. In the same machine, what is the maximum speedup possible by optimizing only instruction memory fetch?

$$\text{Speedup} = \frac{1}{1 - 0.4 + \frac{0.4}{\infty}} = \frac{1}{0.6} = \frac{5}{3} = \boxed{1.67\text{x}}$$

QUESTION II:

Convert the C function on the next page to MIPS assembly language. Make sure that your assembly language code could be called from a standard C program (that is to say, make sure you follow the MIPS calling conventions).

This machine has no delay slots. The stack grows downward (toward lower memory addresses). The following registers are used in the calling convention:

Register Name	Register Number	Usage
\$zero	0	Constant 0
\$at	1	Reserved for assembler
\$v0, \$v1	2, 3	Function return values
\$a0 - \$a3	4 – 7	Function argument values
\$t0 - \$t7	8 – 15	Temporary (caller saved)
\$s0 - \$s7	16 – 23	Temporary (callee saved)
\$t8, \$t9	24, 25	Temporary (caller saved)
\$k0, \$k1	26, 27	Reserved for OS Kernel
\$gp	28	Pointer to Global Area
\$sp	29	Stack Pointer
\$fp	30	Frame Pointer
\$ra	31	Return Address

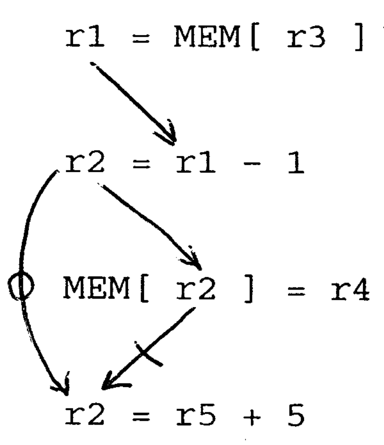
This is the function you should convert:

```
unsigned int sum(unsigned int n)
{
    if (n == 0) return 0;
    else return n + sum(n-1);
}
```

```
sum:    add    $v0, $zero, $zero    ; Initialize return value
                                    ;   to 0
        beq    $a0, $zero, return   ; If n == 0 goto
                                    ;   return
        addiu  $sp, $sp, -8         ; Make room on stack
        sw     $a0, 0($sp)          ; Save n
        sw     $ra, 4($sp)          ; Save return address
        addiu  $a0, $a0, -1         ; n = n-1
        jal    sum                  ; $v0 = sum(n-1)
        lw     $a0, 0($sp)          ; restore n
        lw     $ra, 4($sp)          ; restore return addr.
        addiu  $sp, $sp, 8          ; Discard stack frame
        add    $v0, $v0, $a0        ; return value =
                                    ;   n + sum(n-1)
return: jr     $ra                  ; Return
```

QUESTION III:

1. Draw the register dependence arcs that exist in the following code segment. Be sure to properly label each arc.



```
r1 = MEM[ r3 ]
r2 = r1 - 1
MEM[ r2 ] = r4
r2 = r5 + 5
```

$\longrightarrow$ flow RAW

$\overset{+}{\longrightarrow}$ anti WAR

$\overset{o}{\longrightarrow}$ output WAW

(r2 is a source!!)

2. The architecture respects all dependences. The implementation of the machine you are targeting has a 3-cycle LOAD-USE data hazard. No other hazards exist.) The code segment above is a small part of a much larger program which you cannot change. Registers r1-r9 are used after this code segment. Registers r10-r20 are not used in this program at all. Write a more efficient version of the code segment (leave room in your answer for part 3). The resulting program which includes your new code segment must have equivalent functionality.

let's put something → here to fill hazard at least a bit.

```
r1 = MEM[r3]
r2 = r1 - 1
MEM[r2] = r4
r2 = r5 + 5
```

move this here since it has no <u>real</u> dependence on the load.

We need to remove false deps to perform the move ⇒ rename.

```
r1 = MEM[r3]
r2 = r5 + 5
r12 = r1 - 1
MEM[r12] = r4
```

rename this r2 since other r2 may be used in other parts of the program.

3. Draw all the register dependence arcs that exist in your optimized code segment above.

(This r2 is not available to the rest of the program because r2 = r5+5 overwrites it.)

QUESTION IV:

Consider the datapath on the last page. This machine does not support code with branch delay slots. (It predicts not-taken with a 1-cycle penalty on taken branches.)

For each control signal listed in the table on the next page, determine its value at cycles 3 through 9, inclusive. Also, show the instruction occupying each stage of the pipeline in all cycles. (Assume the IF/ID write-enable line is set to the inverse of the Stall signal.)

The initial state of the machine is:

PC = 0
All pipeline registers contain 0s
All registers in the register file contain 0s.
The data memory contains 0s in all locations
The instruction memory contains:

```
00: addiu $3, $zero, 4
04: lw $4, 100($3)
08: addu $2, $4, $3
0C: beq $4, $zero, 0x14
10: addiu $3, $3, 1
14: addiu $2, $2, $3
```

all other locations contain 0

Use data forwarding whenever possible. All mux inputs are numbered vertically from "top" to "bottom" starting at 0 as you look at the datapath in the proper landscape orientation. Also, the values for ALUOp are:

Value	Desired ALU Action
00	Add
01	Subtract
10	Determine by decoding funct field

Instruction formats:

	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
R:	op	rs	rt	rd	shamt	funct
I:	op	rs	rt	address / immediate		
J:	op	target address				

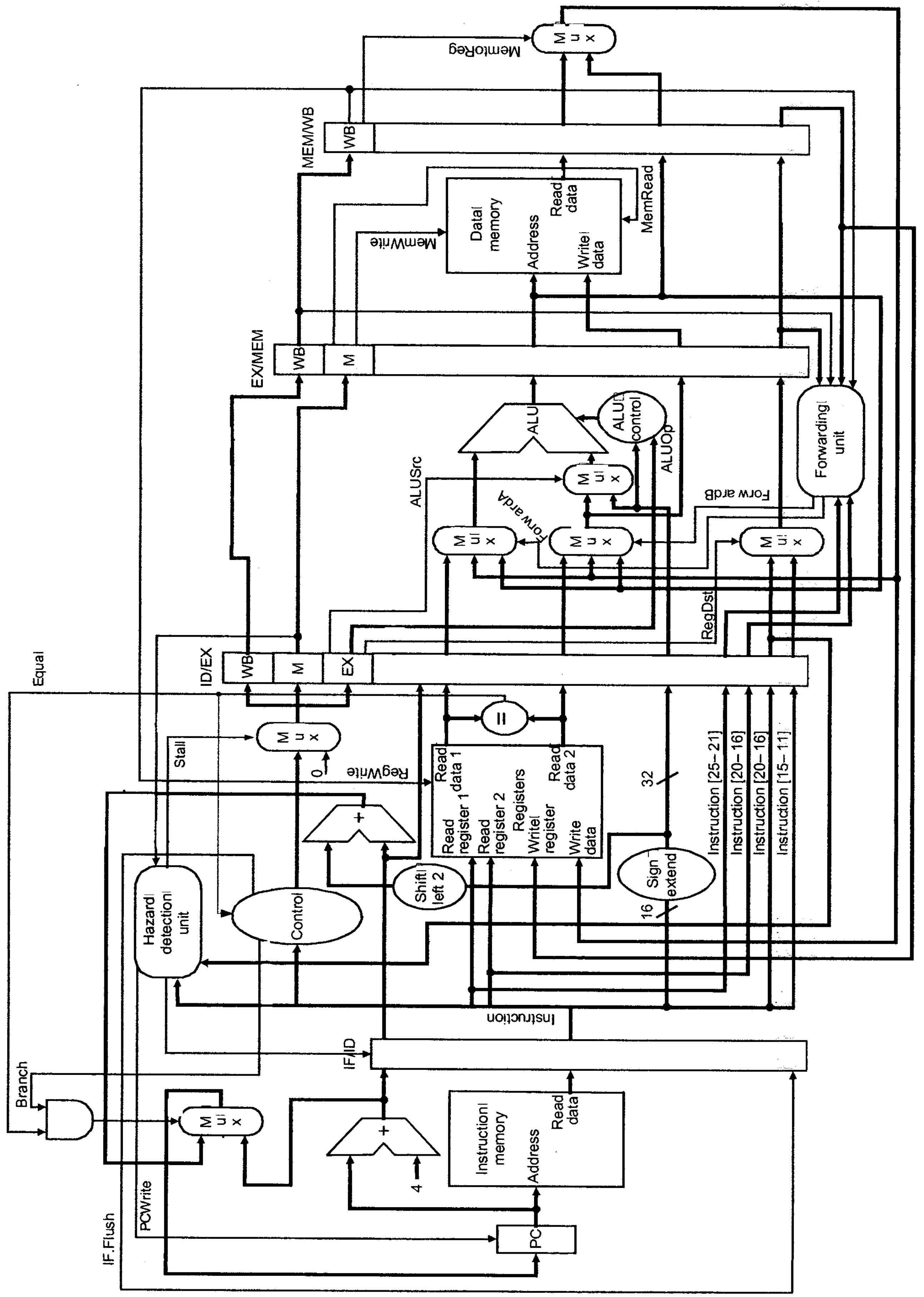
Time	PCWrite	IF.Flush	Branch	Stall	ALUSrc	ALUOp	RegDst	ForwardA	ForwardB	MemRead	MemWrite	MemtoReg	RegWrite
	IF		ID		EX					M		WB	
0	00: addiu		--		--					--		--	
	1	0	0	0	0	00	0	00	00	0	0	0	0
1	04: lw		00: addiu		--					--		--	
	1	0	0	0	0	00	0	00	00	0	0	0	0
2	08: addu		04: lw		00: addiu					--		--	
	1	0	0	0	1	00	0	00	00	0	0	0	0
3	0C: beq		08: addu		04: lw			BYPASS M TO EX		00: addiu		--	
	0	0	0	1	1	00	0	10	00	X	0	X	0
4	0C: beq		08: addu		LOAD-USE STALL (NOP)					04: lw		00: addiu	
	1	0	0	0	X	X	X	X	X	1	0	1	1
5	10: addiu		0C: beq		08: addu			BYPASS WB TO EX		— (NOP)		04: lw	
	1	1	1	0	0	10	1	01	00	X	0	0	1
6	14: addu		TAKEN BRANCH PENALTY — (nop)		0C: beq					08: addu		— (NOP)	
	1	0	0	0	X	X	X	X	X	X	0	X	0
7	18: nop		14: addu		— (nop)					0C: beq		08: addu	
	1	0	0	0	X	X	X	X	X	X	0	1	1
8	1C: nop		18: nop		14: addu					— (NOP)		0C: beq	
	1	0	0	0	0	10	1	00	00	X	0	X	0
9	20: nop		1C: nop		18: nop					14: addiu		— (NOP)	
	1	0	0	0	X	X	X	X	X	X	0	X	0

STALL FOR LOAD-USE; assert STALL to insert bubble after ID and deassert PCWrite

predicted not-taken. branch resolves; should have been taken, so Flush asserted to insert bubble after IF (removing 0x10)

```
00: addiu $3, $zero, 4
04: lw $4, 100($3)
08: addu $2, $4, $3
0C: beq $4, $zero, 0x14
10: addiu $3, $3, 1
14: addiu $2, $2, $3
```

"X" means don't care


IF.Flush
PCWrite
Branch
Equal
Hazard detection unit
Stall
Control
0
M u x
ID/EX
WB
M
EX
RegWrite
IF/ID
Instruction
PC
Instruction memory
Address
Read data
4
+
Shift left 2
Registers
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
=
Sign-extend
16
32
Instruction [25–21]
Instruction [20–16]
Instruction [20–16]
Instruction [15–11]
RegDst
ALUSrc
Forw ardA
Forw ardB
ALU
ALU control
ALUOp
Forwarding unit
EX/MEM
MemWrite
MemRead
Data memory
Address
Write data
Read data
MEM/WB
MemtoReg