# COS 471A,COS 471B/ELE 375 Midterm

Prof: David August
TAs : Jonathan Chang
Junwen Lai
Neil Vachharajani

Fall 2004

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.*"

| Question | Score |
|---|---|
| 1 | /20 |
| 2 | /20 |
| 3 | /20 |
| 4 | /20 |
| Total | /80 |

Name:

Course (circle one): COS471B/ELE375 COS471A

Honor Code:

## 1 Binary Arithmetic

Consider 2's complement 4-bit signed integer addition and subtraction.

### 1.1

Since the operands can be negative or positive and the operator can be subtraction or addition, there are 8 possible combinations of inputs. For example, a positive number could be added to a negative number, or a negative number could be subtracted from a negative number, etc. (I just did a quarter of your thinking for you!). For each of them, describe how the overflow can be computed from the sign of the input operands and the carry out and sign of the output. Fill in the table below:

| Sign(Input 1) | Sign(Input 2) | Operation | Sign(Output) | Overflow(Y/N) |
|---|---|---|---|---|
| + | + | + | + | |
| + | + | + | - | |
| + | + | - | + | |
| + | + | - | - | |
| + | - | + | + | |
| + | - | + | - | |
| + | - | - | + | |
| + | - | - | - | |
| - | + | + | + | |
| - | + | + | - | |
| - | + | - | + | |
| - | + | - | - | |
| - | - | + | + | |
| - | - | + | - | |
| - | - | - | + | |
| - | - | - | - | |

### 1.2

Define the WiMPY precision IEEE 754 floating point format to be:

$$\underbrace{X}_{Sign}\ \underbrace{XXX}_{Exponent}\ \underbrace{XXXX}_{Mantissa}$$

where each 'X' represents one bit. Convert each of the following WiMPY floating point numbers to decimal:

1. 00000000

2. 11011010

3. 01110000

## 2 Dependence Detection

### 2.1

This question covers your understanding of dependencies between instructions. Using the code below, list all of the dependence types (RAW, WAR, WAW). List the dependencies in the respective table by writing the instruction numbers involved with the dependence. You may not need to fill the entire table.

```
I0:    r1 = r2 + r3;
I1:    r3 = r1 - r2;
I2:    r4 = r1 + r3;
I3:    r1 = r2 * r3;
I4:    r3 = r5 / r4;
```

| RAW Dependence | | WAR Dependence | | WAW Dependence | |
|---|---|---|---|---|---|
| From Instr | To Instr | From Instr | To Instr | From Instr | To Instr |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

### 2.2

Given four instructions, how many unique comparisons (between register sources and destinations) are necessary to find all of the RAW, WAR and WAW dependencies (find a tight upper bound). Answer for the case of four instructions, and then derive a general equation for N instructions. Assume that all instructions have one register destination and two register sources.

## 3 Loop Unrolling and Fibonacci

Consider the following pseudo-C code to compute the fifth Fibonacci number ($F(5)$).

```
int a,b,i,t;
a=b=1;      /* Set a and b to F(2) and F(1) respectively */
for(i=0;i<2;i++)
{
  t=a;      /* save F(n-1) to a temporary location */
  a+=b;     /* F(n) = F(n-1) + F(n-2) */
  b=t;      /* set b to F(n-1) */
}
```

One observation that a compiler might make is that the loop construction is somewhat unnecessary. Since the the range of the loop indices is fixed, one can *unroll* the loop by simply writing three iterations of the loop one after the other without the intervening increment/comparison on `i`. For example, the above could be written as:

```
int a,b,t;
a=b=1;
t=a;
a+=b;
b=t;
t=a;
a+=b;
b=t;
```

### 3.1

Convert the pseudo-C code for both of the snippets above into reasonably efficient MIPS code. Represent each variable of the pseudo-C program with a register. Try to follow the pseudo-C code as closely as possible (i.e. the first snippet should have a loop in it, while the second should not).

### 3.2

Now suppose that instead of the fifth Fibonacci number we decided to compute the 20th. How many static instructions would there be in the first version and how many would there be in the unrolled version? What about dynamic instructions? *You do not need to write out the assembly for this part.*

### 3.3

What are the advantages of unrolled code on a deeply pipelined system?

## 3.4

Assuming a four-instruction-deep, single-issue pipe with no branch prediction how many cycles will the unrolled and non-unrolled versions of $F(5)$ take? *State any assumptions you make.*

## 3.5 (Optional Extra Credit)

How might a *really* smart compiler further optimize the unrolled code for $F(5)$?

## 4 Pipelining and Bypass

In this question we will explore how bypassing affects program execution performance. To begin consider the standard MIPS 5 stage pipeline (presented in lecture and the book). For your reference, refer to the figure on the last page. For this question, we will use the following code to evaluate the pipeline's performance:

```
1   add  $t2, $s1, $sp
2   lw   $t1, $t1, 0
3   addi $t2, $t1, 7
4   add  $t1, $s2, $sp
5   lw   $t1, $t1, 0
6   addi $t1, $t1, 9
7   sub  $t1, $t1, $t2
```

...

### 4.1

What is the load-use latency for the standard MIPS 5-stage pipeline?

### 4.2

Once again, using the standard MIPS pipeline, identify whether the value for each register operand is coming from the bypass or from the register file. For clarity, please write REG or BYPASS in each box.

| Instruction | Src Operand 1 | Src Operand 2 |
|---|---|---|
| 1 | | |
| 2 | | N/A |
| 3 | | N/A |
| 4 | | |
| 5 | | N/A |
| 6 | | N/A |
| 7 | | |

### 4.3

How many cycles will the program take to execute on the standard MIPS pipeline?

## 4.4

Assume, due to circuit constraints, that the bypass wire from the memory stage back to the execute stage is omitted from the pipeline. What is the load-use latency for this modified pipeline?

## 4.5

Identify whether the value for each register operand is coming from the bypass or from the register file for the modified pipeline. For clarity, please write REG or BYPASS in each box.

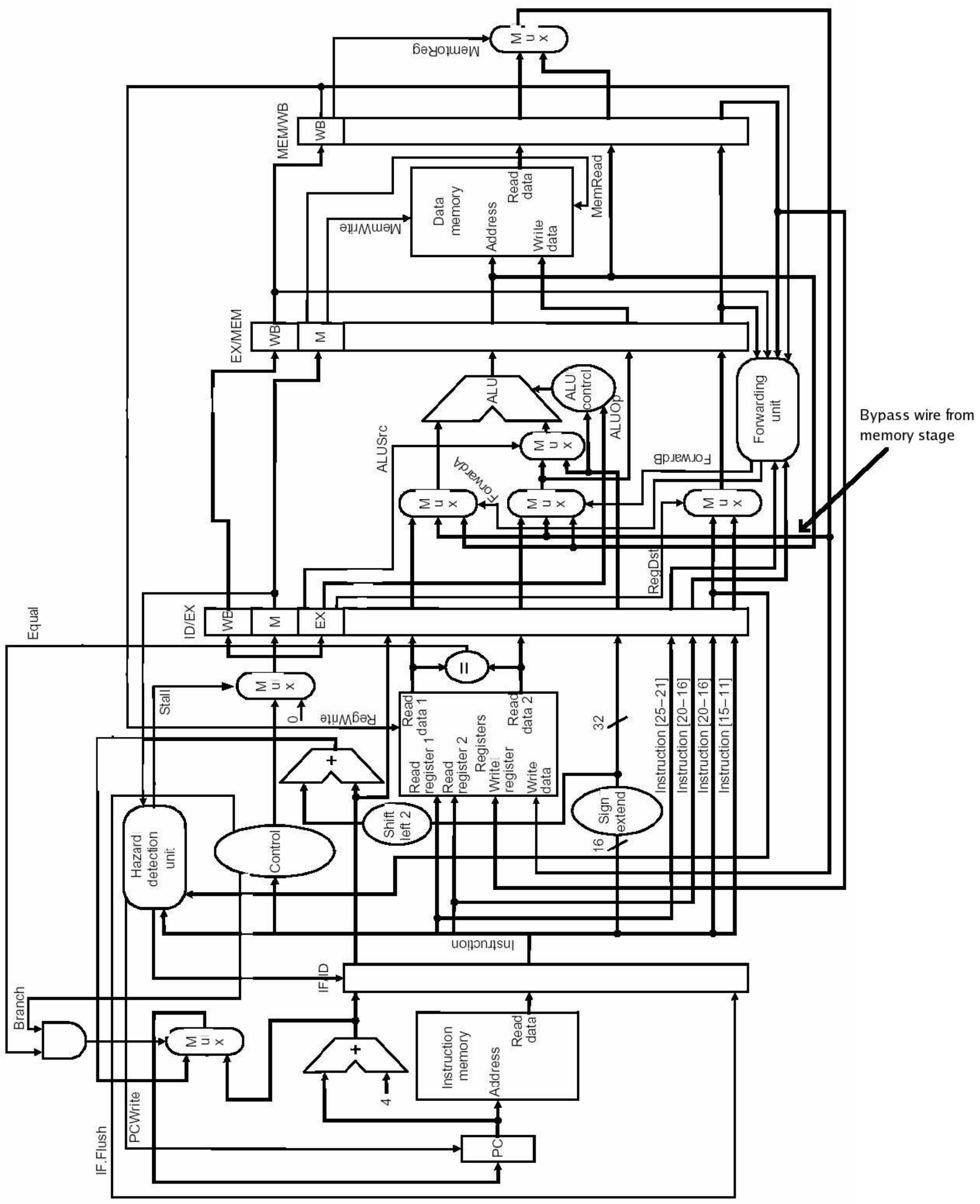
| Instruction | Src Operand 1 | Src Operand 2 |
|---|---|---|
| 1 | | |
| 2 | | N/A |
| 3 | | N/A |
| 4 | | |
| 5 | | N/A |
| 6 | | N/A |
| 7 | | |

## 4.6

How long does the program take to execute on the modified pipeline?

## 4.7

Could the code be transformed to make the program perform better on the modified pipeline? If so, show the transformed program. You are free to use any register not used in the code. Also, you can assume that only register `$t1` is used by the code following this code. How many cycles does it take to execute on the modified pipeline? How about on the original pipeline?



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