University of California, Berkeley
College of Engineering
Computer Science Division — EECS

Spring 1999 John Kubiatowicz

Midterm I

March 3, 1999
CS152 Computer Architecture and Engineering

Your Name:	
SID Number:	
Discussion Section:	

Problem	**Possible**	**Score**
1	15	
2	15	
3	20	
4	20	
5	30	
Total		

[This page left for π]

3.141592653589793238462643383279502884197169399375105820974944

Problem 1: Performance

Problem 1a:
Name the three principle components of runtime that we discussed in class. How do they combine to yield runtime?

Now, you have analyzed a benchmark that runs on your company's processor. This processor runs at 300MHz and has the following characteristics:

Instruction Type	Frequency (%)	Cycles
Arithmetic and logical	40	1
Load and Store	30	2
Branches	20	3
Floating Point	10	5

Your company is considering a cheaper, lower-performance version of the processor. Their plan is to remove some of the floating-point hardware to reduce the die size.

The wafer on which the chip is produced has a diameter of 10cm, a cost of \$2000, and a defect rate of 1 / (cm^2). The manufacturing process has an 80% wafer yield and a value of 2 for α. Here are some equations that you may find useful:

$$\text{dies/wafer} = \frac{\pi \times (\text{wafer diameter}/2)^2}{\text{die area}} - \frac{\pi \times \text{wafer diameter}}{\sqrt{2 \times \text{die area}}}$$

$$\text{die yield} = \text{wafer yield} \times \left(1 + \frac{\text{defects per unit area} \times \text{die area}}{\alpha}\right)^{-\alpha}$$

The current procesor has a die size of 12mm $\times$ 12mm. The new chip has a die size of 10mm $\times$10mm, and floating point instructions will take 12 cycles to execute.

Problem 1b:
What is the CPI and MIPS rating of the original processor?

Problem 1c:
What is the CPI and MIPS rating of the new processor?

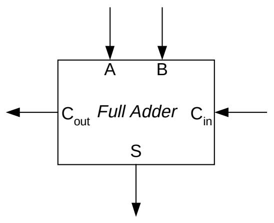
Problem 1d:
What is the original cost per (working) processor?

Problem 1e:
What is the new cost per (working) processor?

Problem 1f:
What is the improvement (if any) in price per performance?

Problem 2: Delay For a Full Adder

A key component of an ALU is a full adder. A symbol for a full adder is:



Problem 2a:
Implement a full adder using as few 2-input AND, OR, and XOR gates as possible. Keep in mind that the Carry In signal may arrive much later than the A or B inputs. Thus, optimize your design (if possible) to have as few gates between Carry In and the two outputs as possible:

Assume the following characteristics for the gates:

AND:	Input load: 150fF,
	Propagation delay: TPlh=0.2ns, TPhl=0.5ns,
	Load-Dependent delay: TPlhf=.0020ns, TPhlf=.0021ns
OR:	Input load: 100fF
	Propagation delay: TPlh=0.5ns, TPhl=0.2ns
	Load-Dependent delay: TPlhf=.0020ns, TPhlf=.0021ns
XOR:	Input load: 200fF,
	Propagation delay: TPlh=.8ns, TPhl=.8ns
	Load-Dependent delay: TPlhf=.0040ns,TPhlf=.0042ns

Problem 2b:
Compute the input load for each of the 3 inputs to your full adder:

Problem 2c:
Identify two critical paths from the inputs to the Sum and the Carry Out signal. Compute the propagation delays for these critical paths based on the information given above. (You will have 2 numbers for each of these two paths):

Problem 2d:
Compute the Load Dependent delay for your two outputs.

Problem 3: Division

Here is pseudo-code for an ***unsigned*** division algorithm. It is essentially the last divider (#3) that we developed in class. Assume that **quotient** and **remainder** are 32-bit global values, and the inputs **divisor** and **dividend** are also 32 bits.

```
divide(dividend, divisor)
{  int count;

   /* Missing initialization instructions. */

   ROL64(remainder,quotient,0);
   while (count > 0) {
      count--;
      if (remainder ≥ divisor) {
         remainder = remainder – divisor;
         temp = 1;
      } else {
         temp = 0;
      }
      ROL64(remainder,quotient,temp);
   }
   /* Something missing here */
}
```

The `ROL64(hi,lo,inbit)` pseudo-instruction treats `hi` and `lo` together as a 64-bit register. Assume that `inbit` contains *only* 0 or 1. ROL64 shifts the combined register left by one position, filling in the single bit from `inbit` into the lowest bit of lo.

Problem 3a:
Implement ROL64 as 5 MIPS instructions. Assume that \$t0, \$t1, and \$t2 are the arguments.
Hint: what happens if you use signed slt on unsigned numbers?

Problem 3b:
This divide algorithm is incomplete. It is missing some initialization and some final code. What is missing?

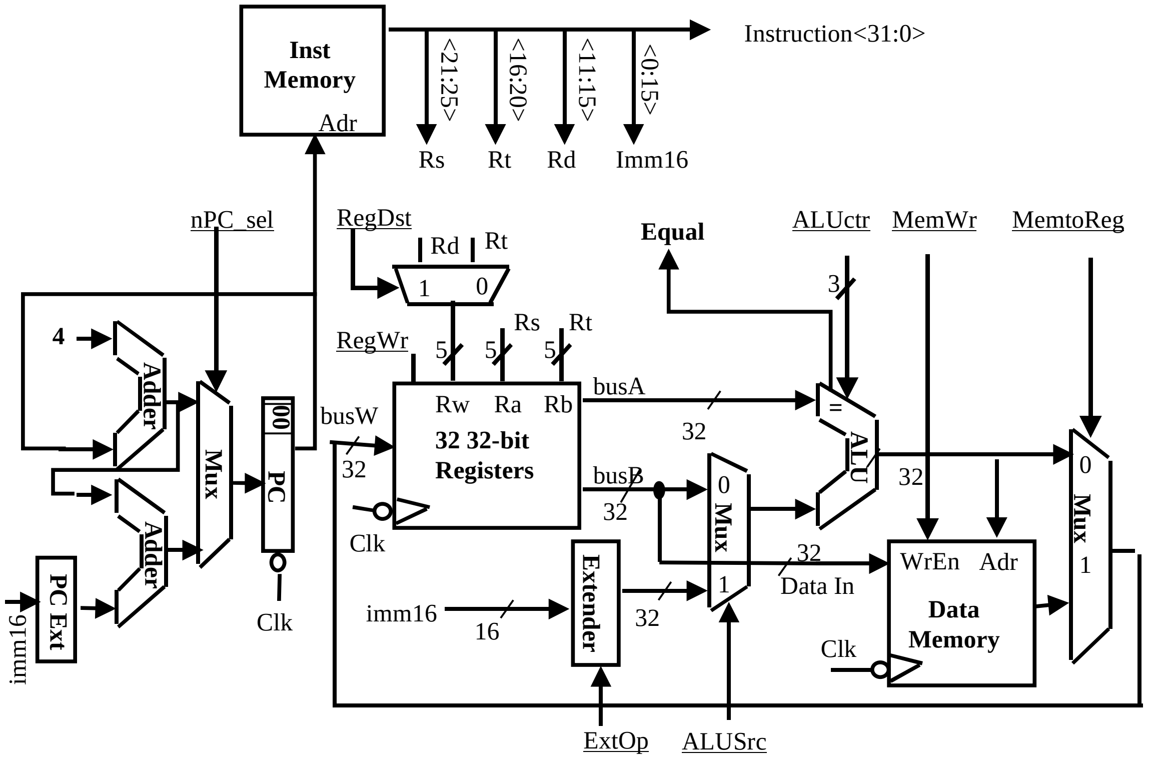
Problem 3c:
Assume that you have a MIPS processor that is missing the divide instruction. Implement the above divide operation as a procedure. Assume **dividend** and **divisor** are in \$a0 and \$a1 respectively, and that **remainder** and **quotient** are returned in registers \$v0 and \$v1 respectively. You can use ROL64 as a pseudo-instruction that takes 3 registers. Don't use any other pseudo-instructions, however. Make sure to adher to MIPS register conventions, and optimize the loop as much as possible.

Problem 3d:
What is the "CPI" of your divide procedure (i.e. what is the total number of cycles to perform a divide)? Assume that each MIPS instruction takes 1 cycle.

[This page left for scratch]

Problem 4: New instructions for a single-cycle data path

The Single-Cycle datapath developed in class is shown below (similar to the one in the book):



It supports the following instructions. (Note that, as in the virtual machine, the branch is not delayed.)

op | rs | rt | rd | shamt | funct = MEM[PC]
op | rs | rt | Imm16 = MEM[PC]

INST	Register Transfers	
ADDU	R[rd] ← R[rs] + R[rt];	PC ← PC + 4
SUBU	R[rd] ← R[rs] - R[rt];	PC ← PC + 4
ORI	R[rt] ← R[rs] + zero_ext(Imm16);	PC ← PC + 4
LW	R[rt] ← MEM[R[rs] + sign_ext(Imm16)];	PC ← PC + 4
SW	MEM[R[rs] + sign_ext(Imm16)] ← R[rs];	PC ← PC + 4
BEQ	**if** (R[rs] == R[rt]) **then**	PC ← PC + sign_ext(Imm16)∪ 00
	else	PC ← PC + 4

Consider adding the following instructions: **ADDIU, XOR, JAL**, and **BGEZAL** (branch on greater than or equal to zero and link). This last instruction branches if the value of register **rs** ≥ 0; further, it saves the address of the next instruction in $ra (like **JAL).** Remember that the JAL format has a 6-bit opcode + 26 bits used as an offset...

Problem 4a:
Describe/sketch the modifications needed to the datapath for each instruction. Try to add as little hardware as possible. Make sure that you are very clear about your changes. Also assume that the original datapath had only enough functionality to implement the original 5 instructions:

Problem 4b:
Specify control for each of the new instructions. Make sure to include values (possibly "x") for all of the control points.

Problem 5: Multiplication

Problem 5a:
Draw the datapath for a multi-cycle, 32-bit x 32-bit *unsigned* multiplier. Try to minimize hardware resources. Assume a single 32-bit adder. Further, to be consistent with MIPS, call the two 32-bit registers that contain the result "hi" and "lo". Draw the control for this multiplier as a round oval with all control signals labeled.

Problem 5b:
Describe the algorithm for performing a multiplication with this hardware. You can use pseudo-code. Make sure to include any initialization that might be required. Also, make sure that any loops are labeled with the number of iterations.

[Problem 5 continued]
Single-bit Booth encoding results from noticing that a sequence of ones can be represented by two non-zero values at the endpoints:

$$\mathbf{111111 = 1000000 - 1 = 10000\bar{1}}$$

The encoding uses three symbols, namely: $\mathbf{\bar{1}, 0, \text{and } 1}$. (The $\bar{1}$ stands for "-1"). A more complicated example of Booth encoding, used on a two's-compliment number is the following:

$$\mathbf{1111111100111011 = 000000\bar{1}0100\bar{1}10\bar{1}}$$

To perform Booth encoding, we build a circuit that is able to recognize the beginning, middle, and end of a string of ones. When encoding a string of bits, we start at the far right. For each new bit, we base our decision of how to encode it on both the current bit and the previous bit (to the right).

Problem 5c:
Write a table describing the this encoding. It should have 2 input bits (current and previous) and should output a 2 bit value which is the two's compliment value of the encoded digit (representing $\bar{1}, 0, \text{or } 1$):

Problem 5d:
Modify your datapath to do 32x32 bit *signed* multiplication by Booth-encoding the multiplier (the operand which is shifted during multiplication). Draw the Booth-encoder as a black-box in your design that takes two bits as input and produces a 2-bit, two's complement encoding on the output. Assume that you have a 32-bit ALU that can either add or subtract. *(Hint: Be careful about the sign-bit of the result during shifts. Also, be careful about the initial value of the "previous bit".)* Explain how how your algorithm is different from the previous multiplier.

Problem 5e: By encoding two bits at a time, we could potentially speed up multiplication a lot. To do this, you simply use your table from problem 5c on two successive bits at the same time. So, we are going to make a table that has *three* input bits (namely the 2 bits for encoding and one previous bit) and which has a single radix-2 symbol as an output. This output will be one of: $\mathbf{\bar{3},\bar{2},\bar{1},0,1,2,3}$.

To help in this, you will first list the two encoded bits that result from using the table in (5c) to encode bits In1/In0 and bits In0/Prev. Don't bother with two's complement, just use the symbols: $\mathbf{\bar{1},0,\text{ and }1}$. Call these results Enc1 and Enc0. Then, treat these two encoded bits as representing a composite number, with the left digit being in the "2s" place (call this the "output-2" column). So, a sample line from your table will be:

Example:

In1	In0	Prev	Enc1	Enc0	Output
1	0	1	$\bar{1}$	1	$\bar{1}$ $\rightarrow$(this is 2x(-1) + 1 = -1)

Write the complete 8-entry table (note again that you are leaving the output symbols as one of $\mathbf{\bar{3},\bar{2},\bar{1},0,1,2,3}$):

Problem 5f: Notice that 3 and $\bar{3}$ never show up in the output column. Explain why this is true. Also, explain why this is good for implementing radix-4 (two-bit at a time) multiplication.

Problem 5g [Extra Credit]:
Draw a datapath that does signed multiplication, two bits at a time and include the multiplication algorithm. Draw the two-bit Booth encoder as a black box which implements output from the table in problem 5f . Make sure that you describe how the 5 possible output symbols (i.e. $\bar{2}, \bar{1}, 0, 1,$ and 2) are encoded *(hint: two's complement is not the best solution here).* As before, assume that you have a 32-bit ALU that can add and subtract: