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

Spring 1999 John Kubiatowicz

Prerequisite Quiz

February 1, 1999
CS152 Computer Architecture and Engineering

This prerequisite quiz will be used in determining class admissions. The use of calculators or notes is not allowed during this quiz. Good Luck!

Your Name:	
SID Number:	
Discussion Section:	

1	
2	
3	
4	
Total	

[This page left for π]

3.14159265358979323846264338327950288419716939937510582097494 4

1) Assume that we have a 16 bit system that uses signed, two's-complement integers. Perform the following conversions:

-8_{10} to base 2: $1111\ 1111\ 1111\ 1000_2$

$D458_{16}$ to base 2: $1101\ 0100\ 0101\ 1000_2$

24_{10} to base 16: 0018_{16}

$(1111110110101100)_2$ to base 16: $FDAC_{16}$

$FC34_{16}$ to base 10:

$$1111\ 1100\ 0011\ 0100_2 =$$
$$-(0000\ 0011\ 1100\ 1011_2 + 1) =$$
$$-(0000\ 0011\ 1100\ 1100_2) =$$

$$-(512+256+128+64+8+4) = -972_{10}$$

2) Suppose that we start with the following C program for factorial:

```
int fact (int n) {
   if ( n < 1 ) {
      return 1;
   } else  {
      return n * fact (n-1);
   }
}
```

Here is a translation of the program above into MIPS assembly language.

```
fact:
   slti  $s0,$a0,1      ⇐ Callie saves reg!  (Use $t0, not $s0)
   bne   $s0,$zero,L1   ⇐ Must be "beq". (Also use $t0):
                           beq $t0, $zero, L1
   addi  $v0,$zero,1
   jr    $ra
L1:subi  $sp,$sp,2      ⇐ Need to adjust by 8 (2 words)
   sw    $a0,0($sp)
                        ⇐ Must also save $ra: Insert instruction
                           sw   $ra,4($sp)
   subi  $a0,$a0,1
   jal   fact
                        ⇐ Must restore argument. Insert:
                           lw   $ra,0($sp)
   lw    $ra,4($sp)
   addi  $sp,$sp,2      ⇐ Need to adjust by 8 (2 words)
   mult  $v0,$a0,$v0    ⇐ Pseudo-op should be called "mul" or,
                           remove dest register and add
                           mflo $v0
   jr    $ra
```

a) There are 6 mistakes in this translation. Fix them in the above listing. *Partial hint: Assume that MIPS register conventions must be maintained throughout the execution.*

See fixed code above. There were 7 bugs if you count "mult" typo: Use of $t0 instead of $s0, adjusting of stack pointer by 8 instead of 2, 2 missed instructions. Note that the "mult" instruction is actually wrong as well.

b) Assume that the initial n=2, the initial sp address = 0x80000000. Also, assume that the instruction in the main program that called `fact` is at address 0x1020, and the fact routine starts at address 0x1200. Draw the memory used for the stack, *after the execution of the program.* Show the position of the stack pointer, as well as the contents of any memory that was modified during execution.

Address	Value
0x800000000	< Unknown>
0x7fffffffc	0x1024
0x7fffffff8	2
0x7fffffff4	0x1224
0x7fffffff0	1

Note that in "bare MIPS", the return values would be 0x1028, 0x1228.

2) Given the following series of memory references: 1, 0, 4, 33, 36, 5, 10, 13, 14, 17, 45, 77, 48, 24, 30, 9, 15, 32, 8, 76. Assume a two-word block with direct-mapped cache with total cache size of 32 words that's initially empty.

a) Label each memory reference address as a Hit (H) or a Miss (M). Show the final contents of the cache in the table at the right. Use the notation "M[i]" to mean the word in memory at address i.

Reference Address	Hit or Miss
1	M
0	H
4	M
33	M
36	M
5	M
10	M
13	M
14	M
17	M
45	M
77	M
48	M
24	M
30	M
9	M
15	H
32	H
8	H
76	H

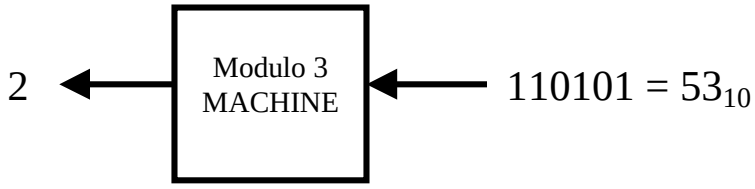
Index	Word 1	Word 0
0	M[33]	M[32]
1		
2	M[5]	M[4]
3		
4	M[9]	M[8]
5	M[11]	M[10]
6	M[77]	M[76]
7	M[15]	M[14]
8	M[49]	M[48]
9		
10		
11		
12	M[25]	M[24]
13		
14		
15	M[31]	M[30]

b) Calculate the miss rate and hit rate.

Hit rate: 5/20 = 25%. Miss rate: 15/20 = 75%

[This page left for scratch]

4) In this problem, we are going to design a Finite State Machine (FSM) that will take an arbitrary-sized integer as input, one bit at a time (starting from most significant bit), and return the remainder after this integer is divided by 3. Here is an example of the machine taking 53_{10} as input and returning 2:



Let the state itself represent the answer. This means we will have 3 states. We need state transition equations. After we have shifted in the first n bits, the machine will be in a state that represents the remainder after dividing those first n bits by three. So:

Assume: K_n is the n-bit number that we have already shifted in,
R_n is the reminder after K_n is divided by 3,
Q_n is the quotient after K_n is divided by 3, and
X_{n+1} is the next bit.

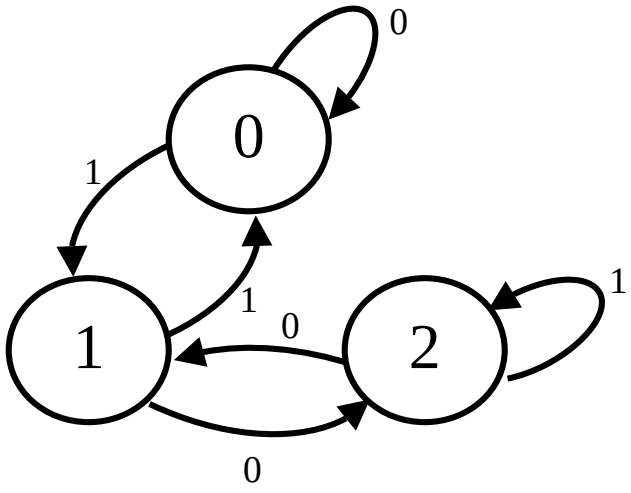
Then, we know that, for all n: $K_n = 3 \times Q_n + R_n$.
Also, when we shift in a new bit, $K_{n+1} = 2 \times K_n + X_{n+1}$

Substituting for K_n and equating: $2\times (3 \times Q_n + R_n) + X_{n+1} = K_{n+1} = 3 \times Q_{n+1} + R_{n+1}$
So, by definition of R_{n+1}: $R_{n+1} = (6 \times Q_n + 2 \times R_n + X_{n+1}) \bmod 3$

Since 6 times anything is clearly divisible by 3, the first term is zero. So, this leads us to our final state update:

$$R_{n+1} \leftarrow (2 \times R_n + X_{n+1}) \bmod 3$$

a) So, R_{n+1} is our state variable (2 bits). Complete the Following State Transition Diagram. (The numbers in the circles represent R_n). Label transition arcs with the value of the new input bit (X_{n+1}). Assume that the machine resets to state 0.



b) Draw a State Transition Table for this FSM.

Cur State	In	New State
00	0	00
00	1	01
01	0	10
01	1	00
10	0	01
10	1	10
11	X	XX

c) Derive Next-State Logic Equations given the state transition table. Make sure to include a signal called "RESET" that is asserted high on reset.

$$NS_1 = \overline{RESET} \cdot \left(\overline{CS_1} \cdot CS_0 \cdot \overline{Input} + CS_1 \cdot \overline{CS_0} \cdot Input \right)$$

$$NS_0 = \overline{RESET} \cdot \left(CS_1 \cdot \overline{CS_0} \cdot \overline{Input} + \overline{CS_1} \cdot \overline{CS_0} \cdot Input \right)$$

d) Implement this FSM using falling-edge triggered flip flops, inverters, and 2, 3, or 4-input NAND gates. Try to minimize the number of gates that you use. Clearly label input and output signals.

NS_0:

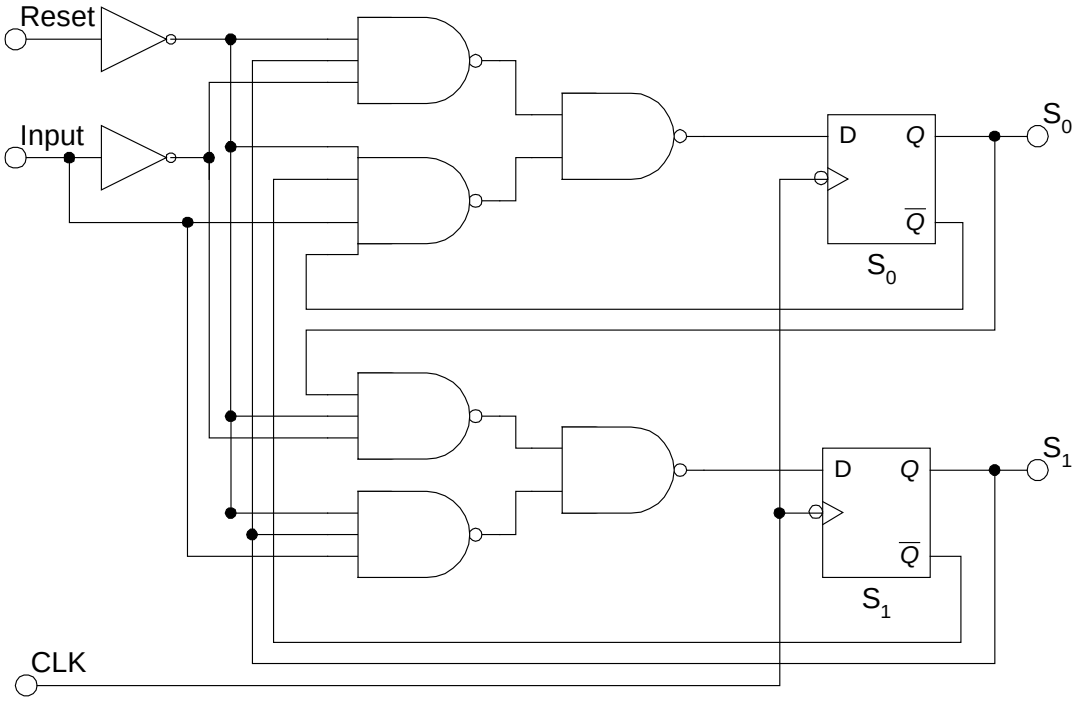
	00	01	11	10
0	0	0	X	1
1	1	0	X	0

NS_1:

	00	01	11	10
0	0	1	X	0
1	0	0	X	1

$$NS_0 = \overline{RESET} \cdot \left(CS_1 \cdot \overline{Input} + \overline{CS_1} \cdot \overline{CS_0} \cdot Input \right)$$

$$NS_1 = \overline{RESET} \cdot \left(CS_0 \cdot \overline{Input} + CS_1 \cdot Input \right)$$



[The End]