University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Sciences
Computer Science Division

J. Wawrzynek

Spring 2006

Machine Structures
Midterm II

ID Number _____ Your Name: _____

Seat Number:: _____ Your TA: _____

Left Neighbor ID: _____ Right Neighbor ID: _____

This is an open-book exam. You are allowed to use any books and notes that you wish. No calculators or electronic devices of any kind, please. You have 2 hours. Each question is marked with its number of points.

Look over all the questions before you begin. Do the easy ones first!

This exam booklet should have 10 printed pages, plus 4 blank pages at the end. Check to make sure that you have all the pages. Fill out the information above **completely** and put your student ID neatly on each page

Show your answers in the space provided for them. Write neatly and be well organized. If you need extra space to work out your answers, you may use the back of previous questions or the blank sheets attached to the back of your exam booklet. However, only the answers appearing in the proper answer space will be graded.

Good luck!

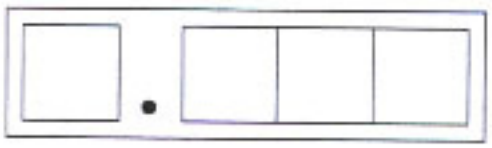
problem	maximum	score
1	8pts	
2	4pts	
3	15pts	
4	6pts	
5	15pts	
6	7pts	
7	5pts	
total	60pts	

1. [8 points] Fixed-point and Floating-point Number Representation.

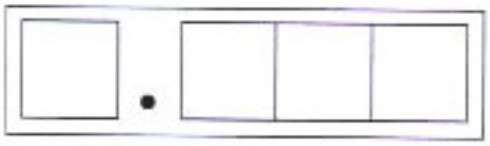
Suppose we have a **2's complement fixed=point** 4-bit number representation with three fractional bits, as follows, $b_3.b_2.b_1.b_0$ (note, the binary point is to the right of the most significant bit).

For each of the following values, write down the closest possible (most accurate) 4-bit fixed-point number representation.

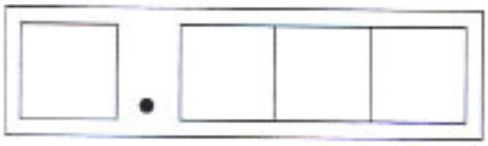
(a) 1/3



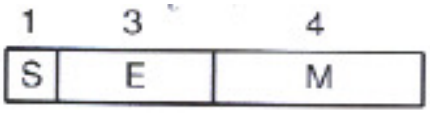
(b) -3/4



(c) 1.0_{ten}



Suppose we have an 8-bit floating point number representation based on the principles of the IEEE 754, with the sign-bit (8), exponent (E), and the mantissa (M) fields as shown below. (You will need to adjust the exponent "bias" accordingly.)



Convert the 8-bit floating-point representation, 01100100, to a decimal number. Show your work.

2. [4 points] Compilers, Interpreters, and Linking.

(a) Multiple choice: circle the 1. or 2. for the best answer.

When comparing compilers to interpreters, in general:

1. compilers take longer to process source code before the program can start running

2. interpreters take longer to process source code before the program can start running

(b) Multiple choice: circle the 1. or 2. for the best answer.

When comparing compiled programs to interpreted programs, in general:

1. compiled programs run faster than interpreted programs

2. interpreted programs run faster than compiled programs

(c) True/False: circle t or F for the best answer.

The technique called "separate compilation" is used to:

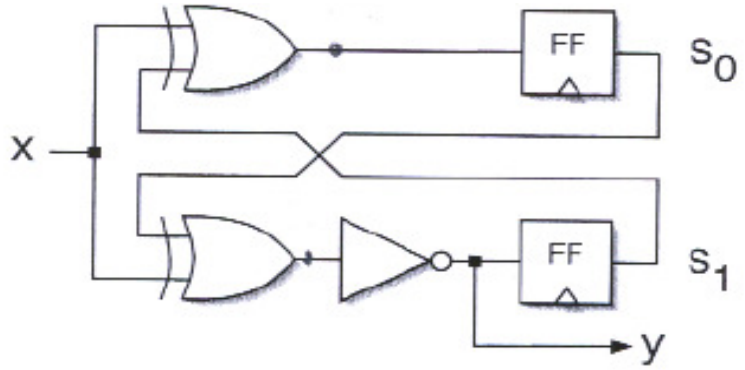
- reduce the overall size of an executable. T/ F
- improve the overall performance of an executable. T / F
- reduce the amount of time required to compile an executable. T/F

(d) For each, put an "X" in the box to indicate the best answer.
Assume that a user has many programs; some of them share libraries with others. The user sometimes runs more than one of these programs at the same time. The user's disk is always slower than the user's main memory (RAM). For this user, the technique called "dynamic linking" (versus a static linking), will:

	always	sometimes	never
increase the time programs take t start up			
require less disk space			

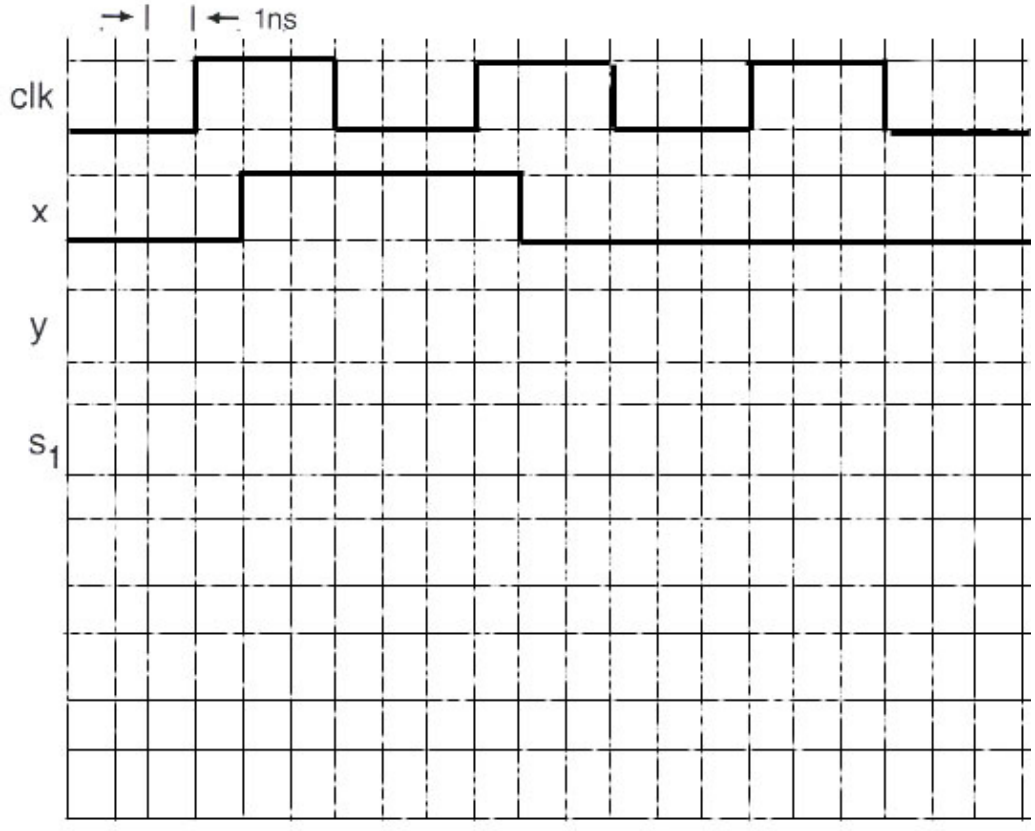
3. [15 points] Synchronous Digital Systems.

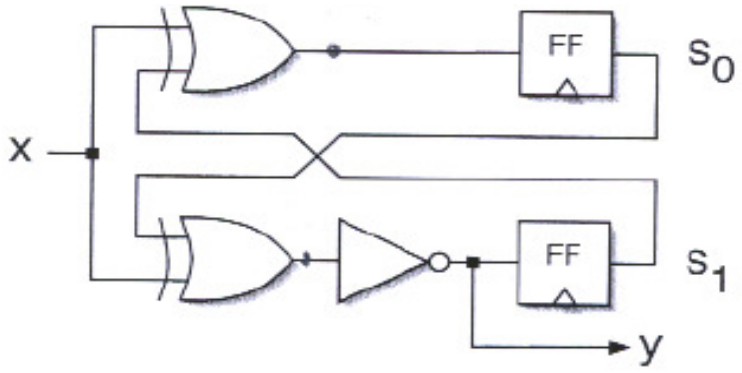
Consider the finite state machine circuit shown below. It has one input, x, and one output, y. A clock signal (not shown in the circuit diagram) is connected to each flip-flop and has a period of 6 ns.



The xor gate and the inverter each have a propagation delay of 1ns. Flip-flops are *positive* edge-triggered and have a set-up time and clock-to-q delay of 1ns each.

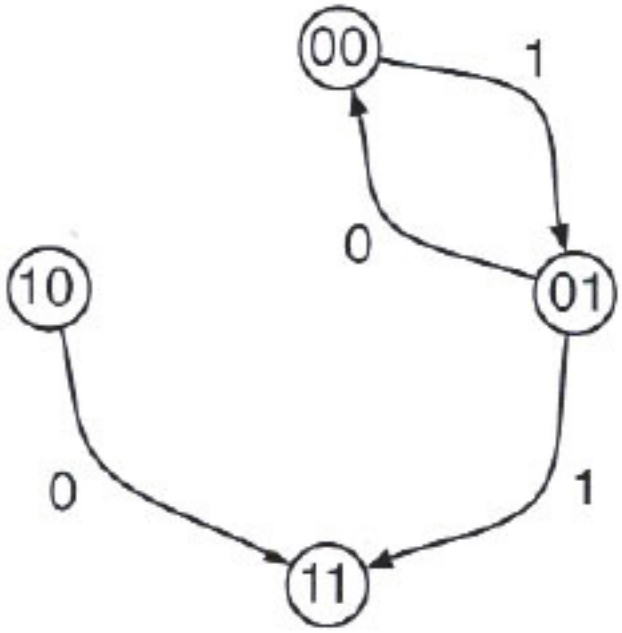
(a) Assume the flip-flips both start out storing a "0" logic value. An input signal is applied as shown below. Neatly, draw the waveforms for the output signal, y, and the signal at the node labeled "s_1". (You might want to practice first).





(b) In the diagram below, finish drawing in the *state transition diagram* for the circuit shown on the previous page (reproduced above). The bubbles below represent the possible states of the circuit as represented by the values held in the flip-flops, s1s0.

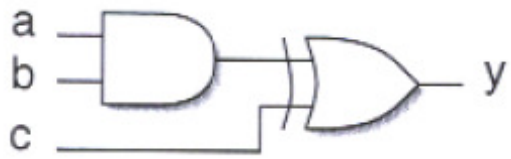
The arcs indicate how the finite state machine transitions from one state to another on each clock cycle, based on the input value. Each arc is labeled with the input value (x) that causes the transition (on the rising edge of the clock). FO instance, if the circuit is in the "00" state and x=1, then on the rising edge of the clock the circuit moves to state "01".



(c) What is the maximum clock *frequency* for the correct operation of this circuit? Assume that on all cycles where the input signal (x), changes, it changes 1ns after the rising edges of the clock.

4. [6 points] Combinational Logic Circuits.

A combinational logic circuit and a truth table are shown below.



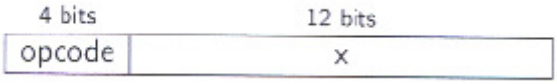
a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

(a) Write the sum-of-products canonical form for the unction represented by the truth-table:

(b) Using only Boolean algebra, prove that the truth-table and the circuit represent the same function. Clearly show your steps.

5. [15 points[Singe-cycle X-processor.

The X-processor is a 16-bit *accumulator-based* architecture instead of a general purpose register file, it has a single register, called the *accumulator* (ACC). All instructions use the same instruction format, shown below:



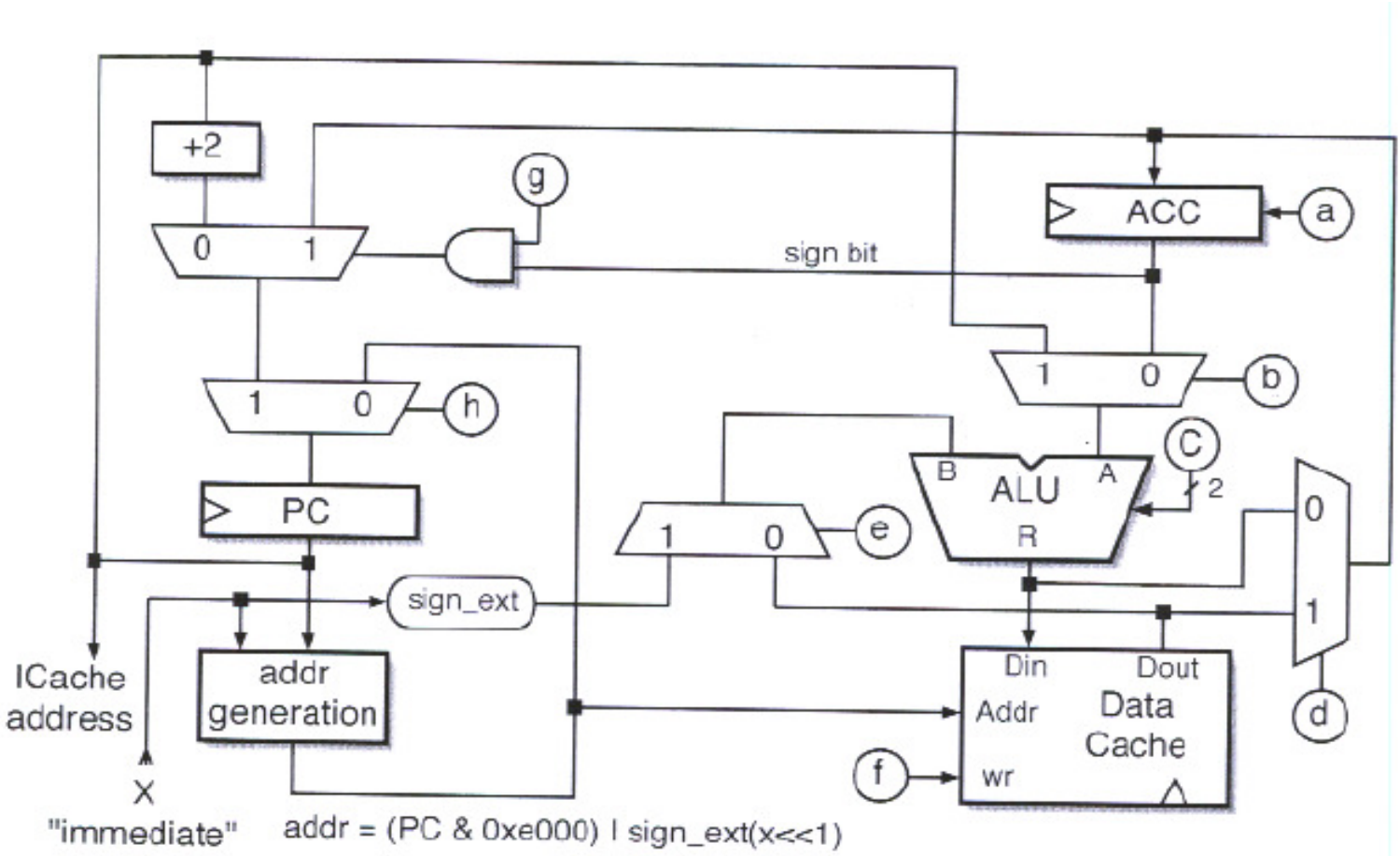
The "x" field is used by each instruction to help specify a memory address, or as an immediate operand. The X- processor instruction are listed below.

instruction	name	operation
load word	`lw`	acc ←MEM[addr]
store word	`sw`	MEM[addr] ←acc
add	`add`	acc ←acc + MEM[addr]
subtract	`sub`	acc ←acc - MEM[addr]
logical nor	`nor`	acc ←acc NOR MEM[addr]
add immediate	`addi`	acc ←acc + sign_ext(x)
branch if less than	`brl`	if acc<0 then PC ←PC + sign_ext(x)
unconditional jump	`jmp`	PC ←addr
addr ≡ (PC & 0xe000) | zero_ext(x<< 1)		

As indicated above, "addr" is a memory address generated from the "x" field by shifting x one place left and appending it to the high three bits of the PC.

The datapath for X-processor is shorn in the next page. The control inputs are labeled with the alphabet, indicated by circles on the diagram. The control signals are described below.

signal	use	meaning
a	ACC write enable	
b	multiplexor control	
C	alu_control	00: R=A+B, 01: R=A-B, 10: R=A NOR B, 11: R=A
d	multiplexor control	
e	multiplexor control	
f	data memory write enable	
g	branch	set to 1 on branch instruction
h	multiplexor control	



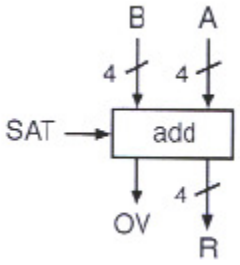
Fill in the table below with the value of each control signal to properly execute each instruction. Write in each square, a "0", "1" or "-" (for don't care).

	lw	**sw**	**add**	**sub**	**addi**	**brl**	**jmp**
a							
b							
C							
d							
e							
f							
g							
h							

6. [7 points[Adder Design.

Consider the design of a 4-bit unsigned *saturating* adder. Saturating addition is like normal addition, except if the result is too large to fit within the number representation the output of the adder is forced to be the largest representable number.

An abstract view of our adder is shown below. It has a special input, SAT, that is used to control if the adder performs normal unsigned addition or saturating unsigned addition. Our adder has a special output signal, OV, used to indicate overflow when performing normal addition.

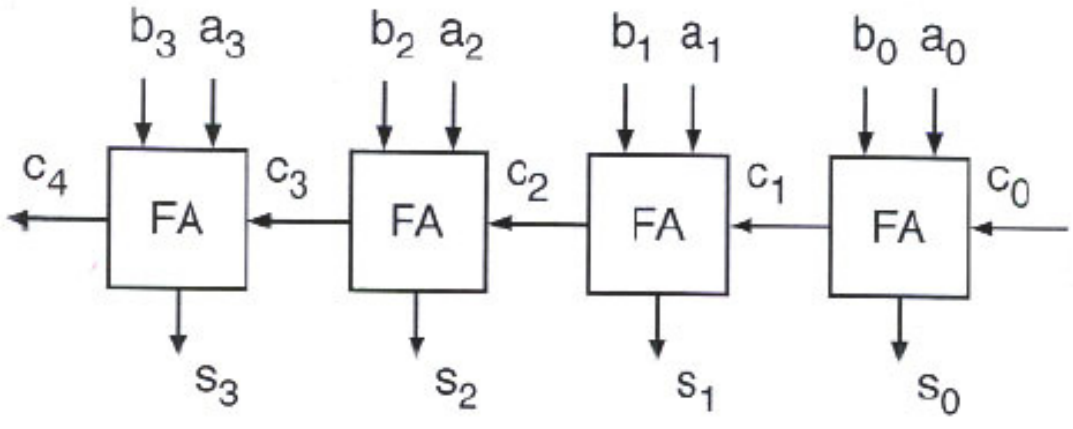


The detailed operation of the adder is as follows:

SAT = 0: Normal unsigned addition with OV=1 if overflow occurred.

SAT = 1: Saturating unsigned addition with OV=0.

Using only AND-gate(s),)R-gates(s), and inverter(s), modify the ripple adder circuit below to implement the above function. Label your inputs and outputs and make sure they match the abstract view of the adder shown above. Try to minimize the amount of extra hardware. (Simplicity and neatness count!)



7. [5 points] Processor Pipelining.

Assume the following two instruction sequences execute on a **pipelined MIPS processor** with the solutions to control and data hazards as discussed in class (branch delay slot, load interlock, register forwarding).

Determine the number of clock cycles needed to execute each sequence. Count from the cycle to fetch the first instruction through the cycle to completely drain the pipeline of the final instruction. (Drain the pipeline between instruction sequences.)

(a)

```
        addi  $1, $0, 2
        add   $0, $0, $0,
loop:   beq   $1, $0, done
        add   $4, $3, $2
        add   $5, $4, $3
        add   $6, $5, $4
        addi  $1, $1, -1
        beq   $0, $0, loop
        addi  $1, $1, -1
done:   beq   $1, $1, exit
        addi  $1, $0, 3
exit:   addi  $1, $0, 1
```

Number of clock cycles:

(b)

```
        lw    $5, 100($0)
        addi  $7, $6, 1
        lw    $7, 200($0)
        addi  $8, $7, 2
        add   $7, $8, $6
```

Number of clock cycles: