

EECS 370
Exam 2
Winter 2006

Name:

University of Michigan username:
(NOT your student ID number!)

Open book, open notes. No calculators, laptops, PDAs, cell phones, etc. This exam has 7 questions, 9 pages, and 100 points. Questions vary in difficulty; it is strongly recommended that you do not spend too much time on any one question. For questions where a box is provided, please put your final answer in the box.

The rules of the Honor Code of the University of Michigan College of Engineering apply for this exam.

Honor code pledge: I have neither given nor received aid on this examination, nor have I concealed any violations of the Honor Code.

Signature:

(examinations without a signed honor pledge will not be graded)

1. Adders (8 points):

(a) Using a **ripple carry** design for an adder, how would the **worst case propagation delay** for a 64 bit adder compare to that of a similar 32 bit adder? (select the **one best choice**)

- (i) The 64 bit adder delay would be approximately half that of the 32 bit adder.
- (ii) The 64 bit adder delay would be several gate delays shorter than the 32 bit adder.
- (iii) The 64 bit adder delay would be exactly the same as the 32 bit adder.
- (iv) The 64 bit adder delay would be several gate delays longer than the 32 bit adder.
- (v) The 64 bit adder delay would be approximately double that of the 32 bit adder.**
- (vi) The 64 bit adder delay would be about four times that of the 32 bit adder.

(b) Using a **carry look-ahead** design for an adder, how would the **worst case propagation delay** for a 64 bit adder compare to that of a similar 32 bit adder? (select the **one best choice**)

- (i) The 64 bit adder delay would be approximately half that of the 32 bit adder.
- (ii) The 64 bit adder delay would be several gate delays shorter than the 32 bit adder.
- (iii) The 64 bit adder delay would be exactly the same as the 32 bit adder.
- (iv) The 64 bit adder delay would be several gate delays longer than the 32 bit adder.**
- (v) The 64 bit adder delay would be approximately double that of the 32 bit adder.
- (vi) The 64 bit adder delay would be about four times that of the 32 bit adder.

2. Pipeline performance (21 points):

An LC2K program consists of 30% loads, 30% arithmetic instructions and 40% branches. 20% of loads are followed immediately by a dependent instruction. 40% of branches are taken. Branches are predicted not taken.

(a) What is the CPI of this program using the original LC2K pipeline (as presented in the lecture)?

One way to solve it, start with baseline $CPI=1$, then account for all stalls or squashes: $1 + 0.3*0.2*1 + 0.4*0.4*3 = 1.54$.

Another way to solve it, account for all cases separately: $0.3*0.8*1 + 0.3*0.2*2 + 0.3*1 + 0.4*0.6*1 + 0.4*0.4*4 = 1.54$

Either way, the bottom line is: **CPI: 1.54**

CPI:	1.54
------	-------------

(b) Theoretically, if you could resolve a hazard between a load and a following dependent instruction without any stalls, what would be the CPI of this program?

$$1 + 0.4*0.4*3 = 1.48$$

CPI:	1.48
------	-------------

(c) Unfortunately, you cannot resolve a hazard between a load and a following dependent instruction without any stalls, but you can improve your branch prediction rate. Assuming that correctly predicted branches can always be handled with no squashing or stalling, what branch prediction rate do you need to achieve the same CPI that you computed in part (b) on this pipeline.

$$1 + 0.3*0.2*1 + 0.4*(1-x)*3 = 1.48$$

Solving gives $x=0.65$

So, branch prediction rate: **0.65 (or 65%)**

Branch prediction rate: 0.65 or 65%
--

3. Single cycle CPUs (16 points):

You are a CPU-architect at Advanced Little Computers, Inc. Your assignment is to design a single-cycle LC2K processor from basic blocks of logic that are given to you. The operations that these blocks perform and the corresponding amounts of time are listed in the table below. All other operations (write-back, register value comparison, update of PC, etc) take a negligible amount of time.

Operation	Time
addition	10ns
nand	7ns
instruction decode and register read	6ns
instruction or data memory access	12ns

(a) How much time is required to execute each of the instructions listed below? That is, how long does it take to do everything that each instruction needs to do, using the logic blocks given. What is the clock cycle period and the frequency at which you can run your single-cycle CPU? (fill in the boxes provided)

Instruction	Time (ns)
add	$12+6+10=28$
nand	$12+6+7=25$
lw	$12+6+10+12=40$
beq (taken)	$12+6+10=28$

Clock cycle period (ns): 40

Clock frequency (MHz): 25

(b) How long does it take for your processor to actually run the following program (assume that initially $r1 = 7$, $r2 = -1$, $r4 = -4$) [please put your answer in nanoseconds in the box provided]:

```
0: nand 2 4 5
1: lw 5 6 0
2: beq 1 6 1
3: .fill 7
4: add 1 4 7
5: halt
```

5 instr * 40ns each = 200ns

4. Pipelined CPUs (16 points):

(a) Your manager tells you that he read in Scientific American that pipelined processors always beat single-cycle processors in terms of performance. You are asked to look into the possibility of making a pipelined LC2K. What is the actual CPI of the pipelined version of LC2K (as discussed in the lecture) for the program below? Use the time graph below as workspace for this question, and write the final CPI answer in the box provided. The first cycle is already given for you. Recall that execution stops when a HALT completes its MEM stage. Thus, the last cycle you show should have "5: HALT" in MEM.

(exactly the same program and initial conditions as in question 3; initially:
 r1 = 7
 r2 = -1
 r4 = -4

0: nand 2 4 5
 1: lw 5 6 0
 2: beq 1 6 1
 3: .fill 7
 4: add 1 4 7
 5: halt

cycle	IF	ID	EX	MEM	WB
1	0: NAND				
2	1: LW	0: NAND			
3	2: BEQ	1: LW	0: NAND		
4	3: .fill	2: BEQ	1: LW	0: NAND	
5	3: .fill	2: BEQ	-	1: LW	0:NAND
6	4: ADD	3: .fill	2: BEQ	-	1:LW
7	-	-	-	2: BEQ	-
8	4: ADD	-	-	-	2:BEQ
9	5: HALT	4: ADD	-	-	-
10	-	5: HALT	4: ADD	-	-
11	-	-	5: HALT	4: ADD	-
12	-	-	-	5: HALT	4:ADD
13					
14					

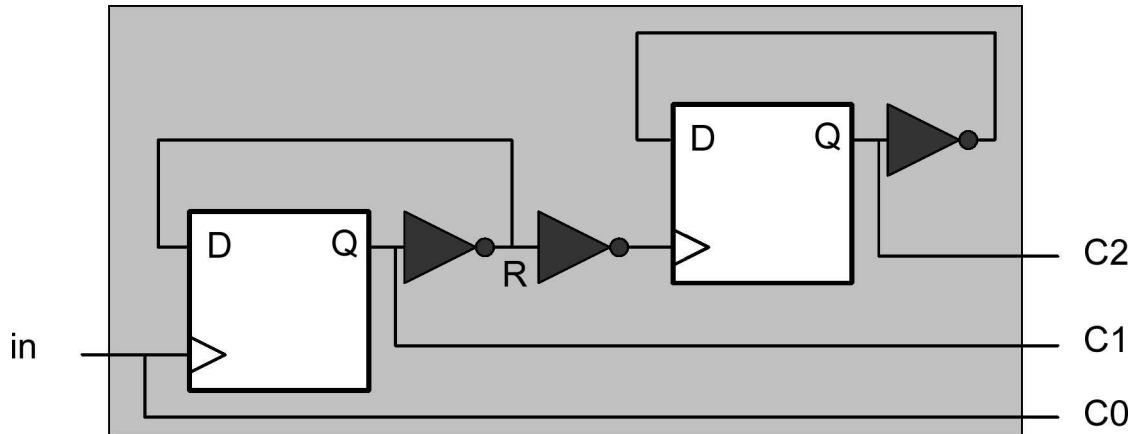
$$\text{CPI} = 12 \text{ cycles} / 5 \text{ instructions} = 2.4$$

(b) You were able to make your pipelined processor run at 60MHz. How long does it take to run this program on the pipelined LC2K processor? (please put your answer in nanoseconds in the box below)

$$(12 \text{ cycles} * 1 / (60 * 10^6)) / 10^{-9} = 200 \text{ ns}$$

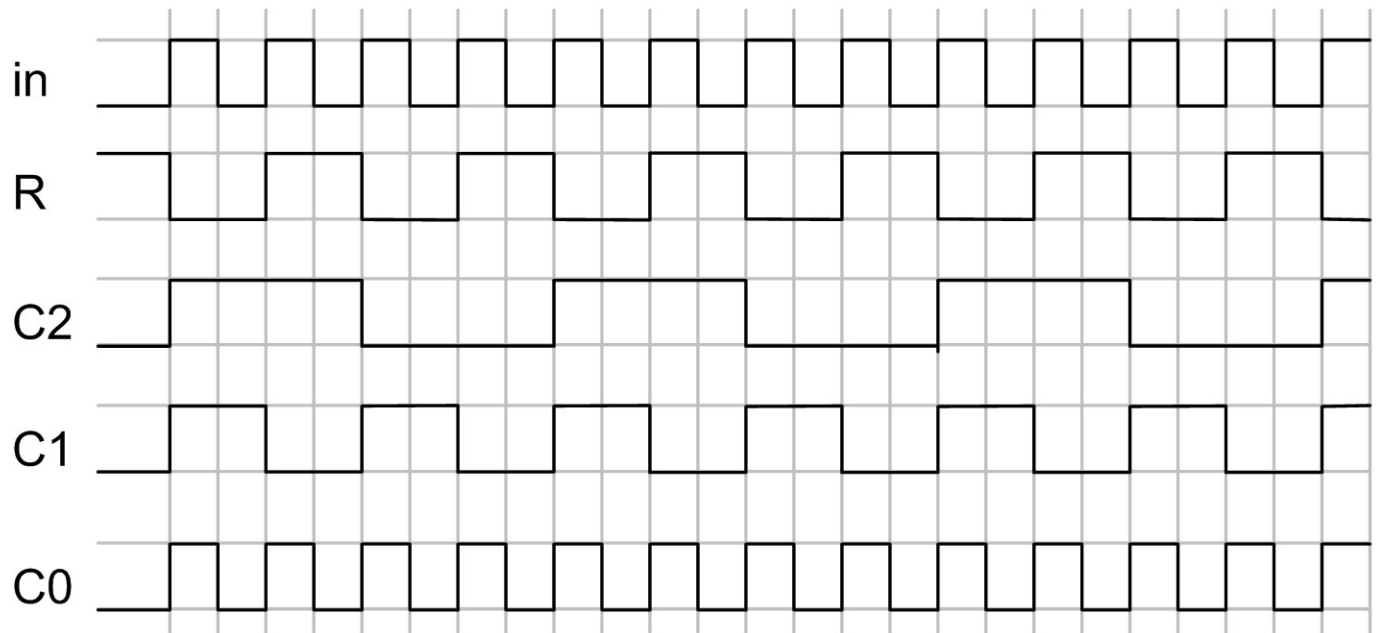
5. Timing diagrams (14 points):

At the end of the semester you are hired at CokeMachines, Inc for a summer internship. When you show up at work, your manager gives you the schematic below, which was left behind by the previous year's intern. The circuit is a central component of the new CM9000, an advanced vending machine which can track when it runs out of sodas. Unfortunately, nobody has been able to understand how the circuit works, which has delayed the completion of the CM9000. The circuit takes an input signal **in** and outputs three signals, **C0**, **C1**, and **C2**. Your job is to discover what this circuit does.



(a) Assume that the input signal is as indicated by the timing diagram below. Draw the waveforms of the signals at nodes **R**, **C0**, **C1**, and **C2**. Assume that the initial value stored in both the storage elements is 0. You can ignore propagation delays in drawing your timing diagrams. (*Hint: it might be easier to compute the diagram for C0 first, then C1 and R, and finally C2*).

Solution 1 - positive-edge triggered flip-flops



(b) Now you need to find what the circuit does. Observe the timing diagram of the output signals that you just computed and, for each transition on any of the output signals, write the value of $\langle C2, C1, C0 \rangle$ as a three bit binary number. (For example, if $C2=1, C1=1, C0=0$, you would write down **110**; in other words, give the sequence of three bit binary numbers that come out of this circuit).

For positive-edge triggered flip-flops the transition sequence is:

000 - 111 - 110 - 101 - 100 - 011 - 010 - 001 - 000 - ...

(c) If the value of $\langle C2, C1, C0 \rangle$ is currently x , write an expression in C language that gives the next value of x in terms of the current value of x .

C expression representing the next value of x is:

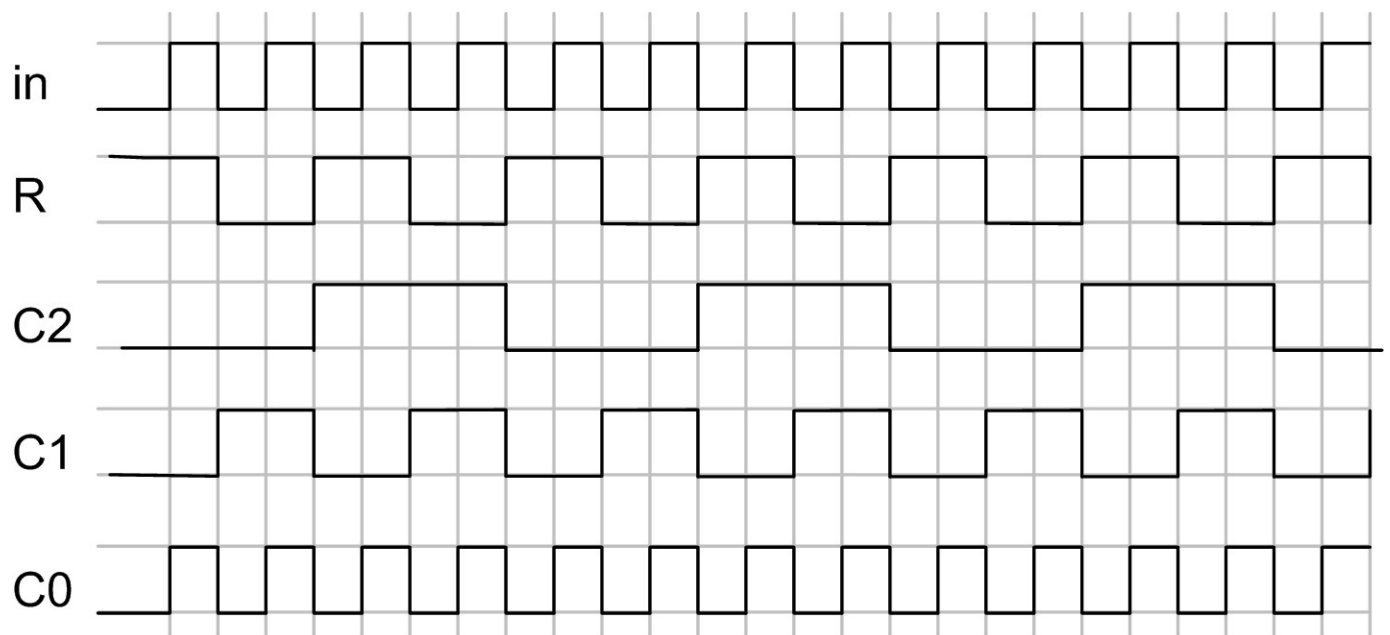
`next_x = (x - 1) % 0x7`

or

`next_x = (x==0) ? 7 : x - 1`

Solution 2 - negedge triggered flip-flops

(a)



(b) the transition sequence is:

000 - 001 - 010 - 011 - 100 - 101 - 110 - 111 - 000 - ...

(c) C expression representing the next value of x is:

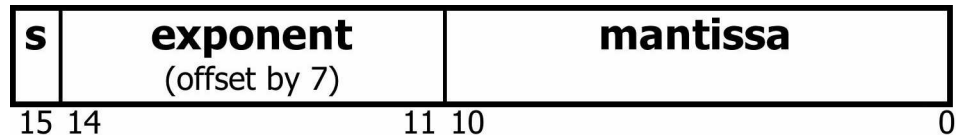
`next_x = (x + 1) % 0x7`

or

`next_x = (x==7) ? 0 : x + 1`

6. Floating point (15 points):

At some point in the future, it would be nice for the LC2K to perform floating-point operations. However, the 32-bit IEEE 754 floating point standard takes more storage space than we wish to use, so we would like to design a 16 bit EECS 370 standard. The EECS 370 standard is similar in form to IEEE 754, it just uses 16 bits instead of 32. The most significant bit is the sign bit, there are 4 bits for the exponent field (with a bias of 7, not 127), and the mantissa (fraction) contains 11 bits.



(a) Assuming that 0x0000 and 0xFFFF are reserved numbers, what is the range, in decimal, of this representation (i.e. what are the largest positive and negative numbers which can be represented in the EECS 370 standard)?

Largest positive number: 511.875

Largest negative number: -511.75

(b) Convert -89.625 to EECS370 floating point.

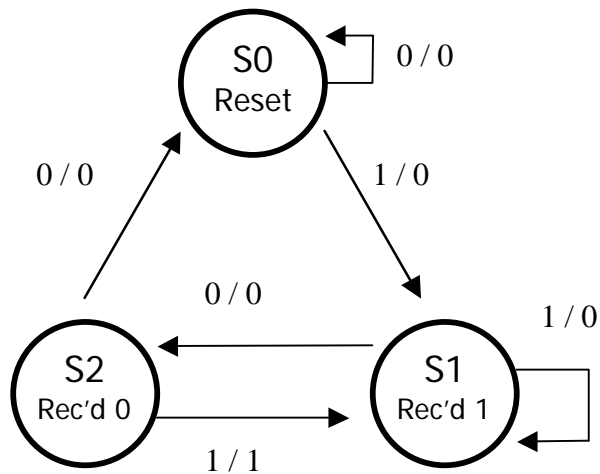
Answer: 1110 1011 0011 0100

7. Finite State Machines (10 points):

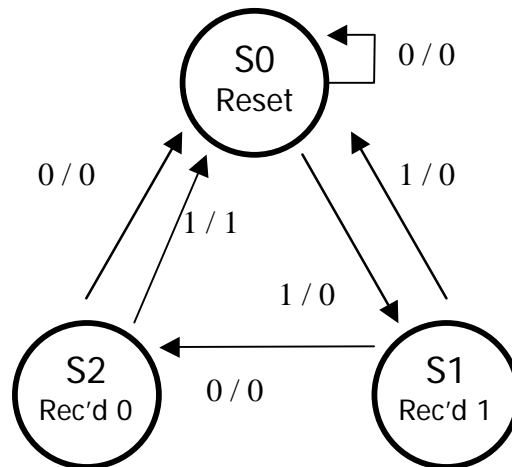
You are responsible for designing control logic of a security keypad for the new CSE building. The keypad has only two numbers that can be pressed: **0** and **1**. The circuit should only send the unlock signal, **U**, any time the correct sequence, **101**, is entered. Assume that the control finite state machine resets any time an incorrect input pattern is entered.

(a) Complete the design of the control FSM by drawing all the necessary transition edges in the graph below. Clearly label the inputs and output signals on each of the edges:

Two solutions are possible depending on the interpretation of the question. If at any point the sequence 101 is entered the door unlocks, the FSM looks like this:



Alternatively, anytime an incorrect sequence is entered, the FSM resets (basically the "correct" input is a 3-digit sequence) and looks like this:



(b) Unfortunately, the building manager realized fairly quickly that this sequence was too simple, so people could easily break into the building. He would like to change the sequence to 1001. How would the finite state machine have to change to accommodate this sequence? (select the correct choice below)

- (i) You need to add 1 more state.
- (ii) You need to add 2 more states.
- (iii) No more states need to be added.
- (iv) The machine can actually be simplified and one state can be removed.