Department of Electrical and Computer Engineering
The University of Texas at Austin

EE 306, Fall 2011
Yale Patt, Instructor
Faruk Guvenilir, Milad Hashemi, Jennifer Davis, Garrett Galow,
Ben Lin, Taylor Morrow, Stephen Pruett, and Jee Ho Ryoo, TAs
Exam 1, September 28, 2011

Name: Solution

Problem 1 (30 points): 30

Problem 2 (20 points): 20

Problem 3 (15 points): 15

Problem 4 (15 points): 15

Problem 5 (20 points): 20

Total (100 points): 100

Note: Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space provided.

Note: Please be sure your name is recorded on each sheet of the exam.

I will not cheat on this exam.

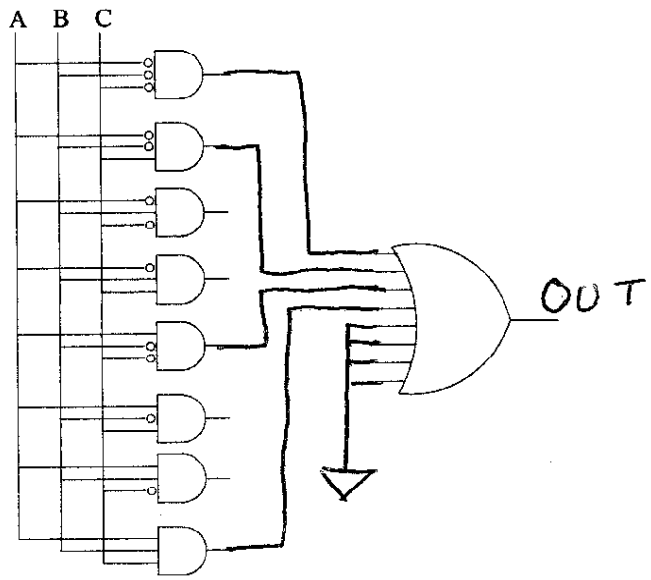
Solution
Signature

GOOD LUCK!

Name: Solution

Problem 1. (30 points):
Part a. (5 points): A function is described by the truth table shown on the left below. Your job: Complete the logic implementation shown on the right by adding the appropriate connections.

A	B	C	Out
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



Part b. (5 points): We have talked about binary, decimal, and hexadecimal. In this problem we are talking about octal, i.e., base-8 representation. Two 3-digit octal numbers are added, and the result, in octal, is 603. One of the numbers is 374. The other number is: 207

$$\begin{array}{r} 374 \\ +\ 207 \\ \hline 603 \end{array}$$

Part c. (5 points): The same 8-bit (one byte) code can represent a number of different values, depending on the data type. For each of the data types in the table below, what is the value of the code 01000000?

ASCII	@
2's Complement Integer	+64
Unsigned Integer	64

Name: Solution

Part d. (5 points): We are just about ready to start writing programs that will execute on the LC-3. One of the instructions the LC-3 can execute is the ADD instruction, which adds two 16-bit 2's complement integers. Suppose the two integers we wish to add are

```
01xxxxx111000xxx
10xxxxx001100xxx
```

where some of the bits have not been identified, and so are represented by x.

Can the sum of the two integers (with x replaced by 0 or 1) ever result in an overflow?

YES NO (circle one)

If yes, give an example. If no, explain why not in 20 words or fewer.

A positive number plus a negative number cannot overflow

Part e. (5 points): Assume a new 8-bit floating point data type, where bit[7] is the sign, bits[6:4] represent the exponent in an excess code, and bits[3:0] represent the fraction bits. The number 3 1/8 is represented exactly as:

```
01001001
```

What is the bias of the excess code? 3

$11.001 = 3\ 1/8$

1.1001×2^1

$1 + \text{Excess} = 4$

$\text{Excess} = 3$

Part f. (5 points): In class we showed the first few states of the finite state machine that is required for processing instructions of a computer program written for LC-3. In the first state, the computer does two things, represented as:

```
MAR <-- PC
PC <-- PC+1
```

Why does the microarchictecture put the contents of the PC into the MAR?

So the instruction that is to be executed can be fetched.

Why does the microarchitecture increment the PC?

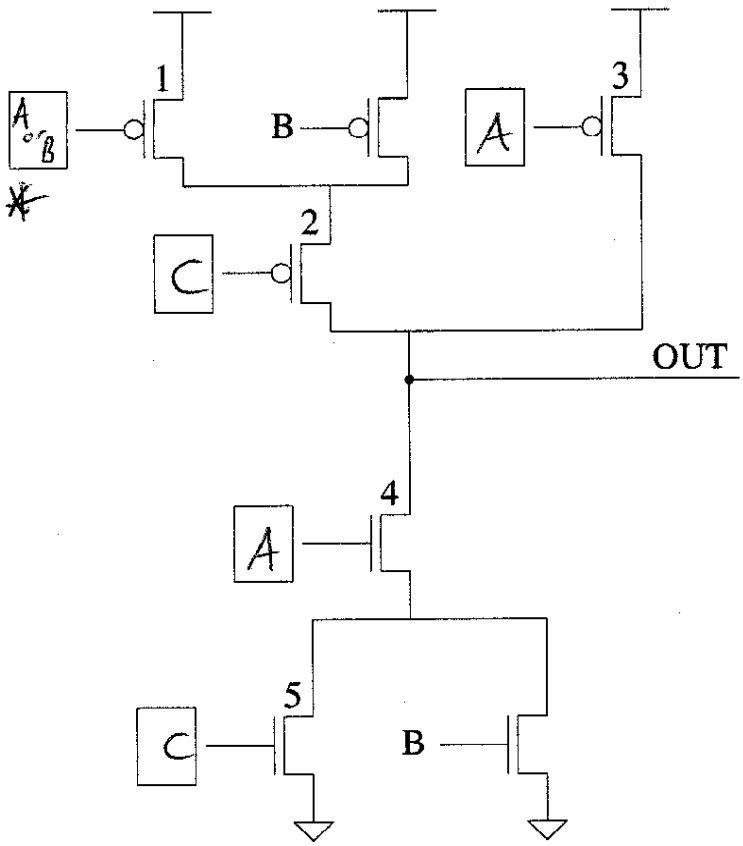
To point to the next instruction to be fetched in preparation for the next instruction cycle

Name: Solution

Problem 2. (20 points): The transistor circuit shown below produces the accompanying truth table. The inputs to some of the gates of the transistors are not specified. Also, the outputs for some of the input combinations of the truth table are not specified.

Your job: Complete both specifications. i.e., all transistors will have their gates properly labeled with either A, B, or C, and all rows of the truth table will have a 0 or 1 specified as the output.

Note that this is not a problematic circuit. For every input combination, either the output is connected to ground (i.e., OUT=0) or to the positive end of the battery (i.e., OUT=1).



A	B	C	OUT
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

* "A", "B", "A or B", all received full credit.

Name: Solution

Problem 3. (15 points): Most word processors will correct simple errors in spelling and grammar. Your job is to specify a finite state machine that will capitalize the personal pronoun I in certain instances if it is entered as a lower case i.

For example, **i think i'm in love** will be corrected to **I think I'm in love.**

Input to your finite state machine will be any sequence of characters from a standard keyboard. Your job is to replace the **i** with an **I** if

the i is the first character input or is preceded by a *space*, and
the i is followed by a *space* or by an *apostrophe*.

Shown below is a finite state machine with some of the inputs and some of the outputs unspecified. Your job is to complete the specification.

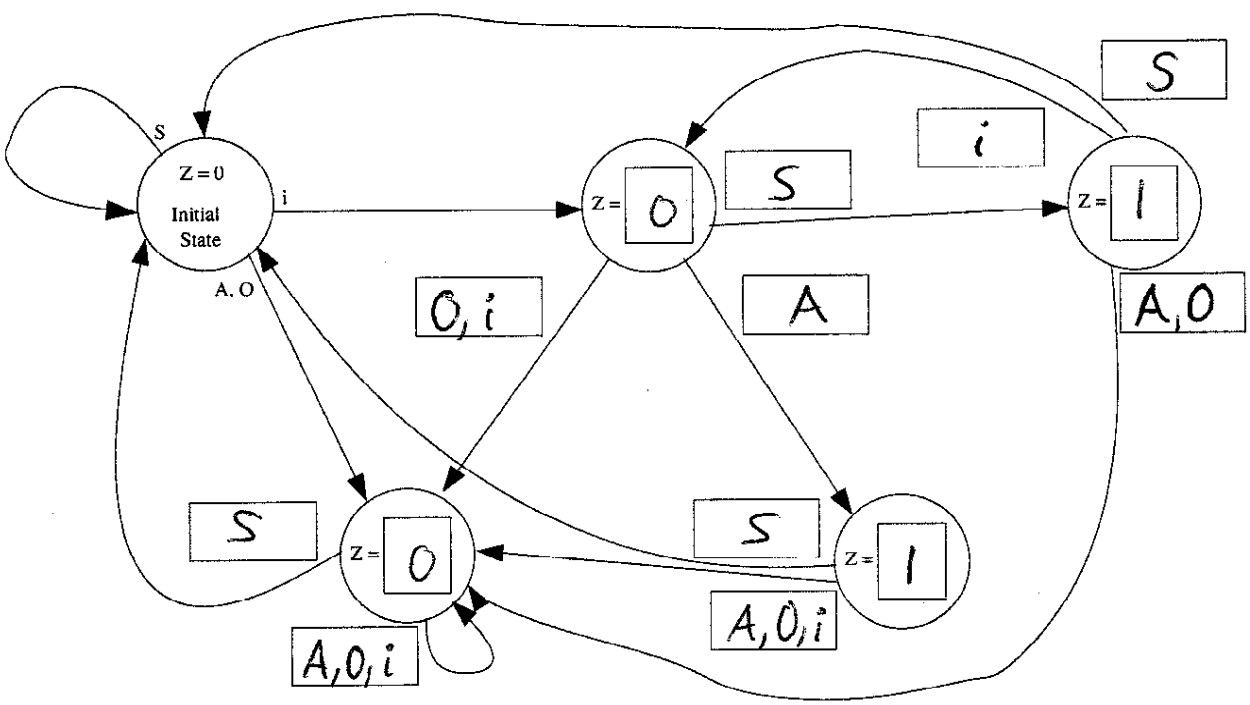
Inputs are from the set {i, A, S, O}, where A represents an apostrophe,
S represents a space,
O represents any character other than i, apostrophe, or *space*.

The output Z corresponding to each state is 0 or 1, where 0 means "do nothing,"
1 means "change the most recent **i** to an **I**."

Note: this exercise in developing a finite state machine word processor is only a first step since a lot of "i to I" will not fix the problem. For example,

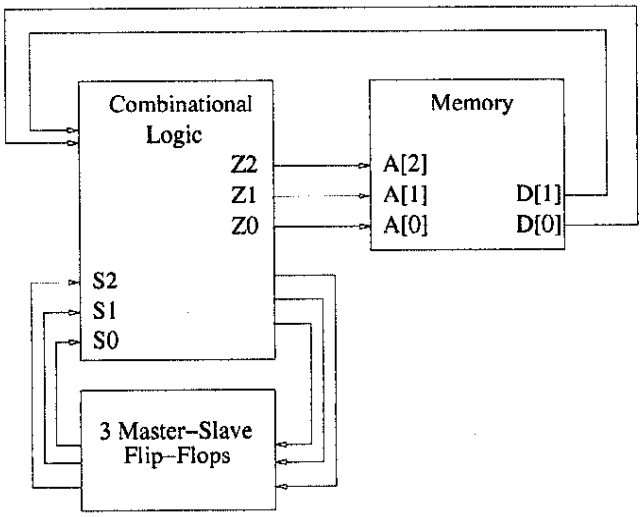
i' am —> I' am, i'abcd —> I'abcd, and i'i —> I'i are all bad!

But it is a first step!



Name: Solution

Problem 4. (15 points): A finite state machine is connected to a 2^3 by 2 bit memory as shown below:



The contents of the memory are shown below to the left. The next state transition table is shown below to the right.

Address A[2:0]	Content D[1:0]
000	11
001	10
010	01
011	10
100	01
101	00
110	00
111	01

Current State S[2:0]	Next State D[1:0] 00	D[1:0] 01	D[1:0] 10	D[1:0] 11
000	001	010	110	100
001	100	000	011	110
010	010	100	111	010
011	001	100	100	010
100	110	011	011	111
101	100	010	100	110
110	001	110	100	010
111	000	101	111	101

The output Z0, Z1, Z2 is the current state of the finite state machine. That is, Z0=S0, Z1=S1, Z2=S2.

The cycle time of the finite state machine is long enough so that during a single cycle, the following happens: the output of the finite state machine accesses the memory and the data supplied by the memory is input to the combinational logic which determines the next state of the machine.

Part a: Complete the table below:

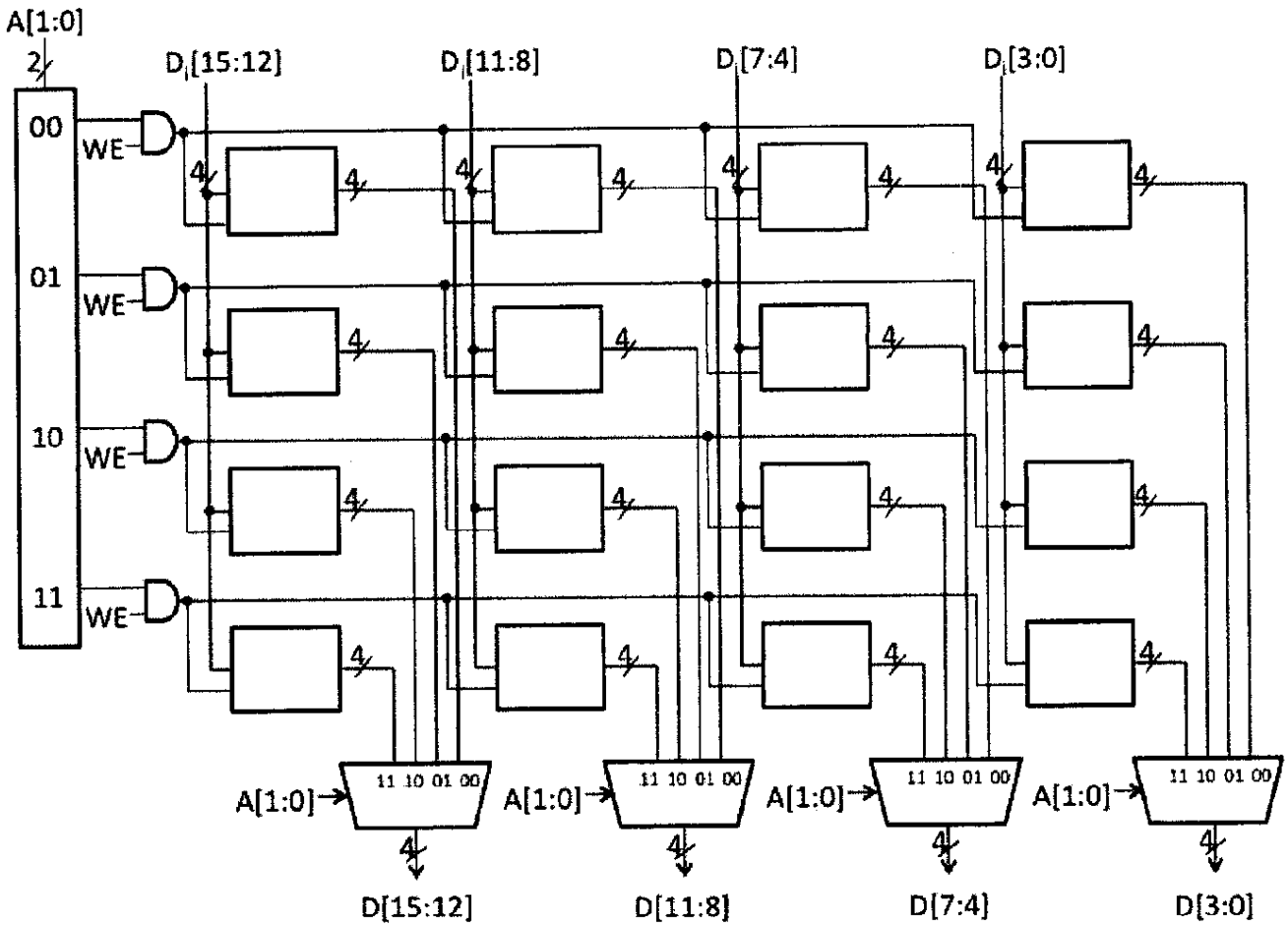
Cycles	State	Data
Cycle 0	000	11
Cycle 1	100	01
Cycle 2	011	10
Cycle 3	100	01

Part b: What will the state of the FSM be just before the end of cycle 100? Why?

011, there is a loop in the states.
Every Even Cycle → State 011
Every Odd Cycle → State 100

Name: Solution

Problem 5. (20 points): Recall the 2^2 by 16-bit memory from problem 6 of problem set 3. It is reproduced below. Recall that each of the four muxes on the diagram have 4-bit input sources and a 4-bit output, and that each 4-bit source is the output of a single 4-bit memory cell.



Name: Solution

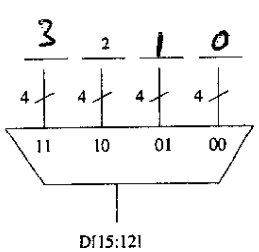
Part a: Unfortunately, the memory was wired by an engineering student from an un-named university, and he got the inputs to some of the muxes mixed up. That is, instead of the 4 bits from a memory cell going to the correct 4-bit input of the mux, the 4 bits all went to one of the other 4-bit sources of that mux. The result was, as you can imagine, a mess.

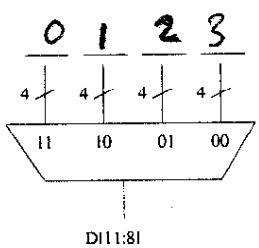
To figure out the mix-up in the wiring, the following sequence of memory accesses was performed:

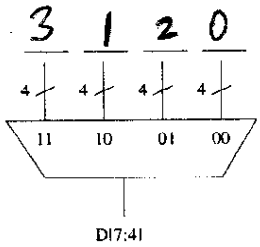
Read/Write	MDR	MAR
Write	x134B	01
Write	xFCA2	10
Write	xBEEF	11
Write	x072A	00
Read	xF34F	10
Read	x1CAB	01
Read	x0E2A	00

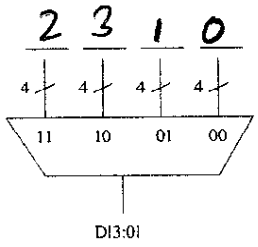
Note: On a write, MDR is loaded before the access. On a read, MDR is loaded as a result of the access.

Your job is to identify the mix-up in the wiring. Show which memory cells were wired to which mux inputs by filling in their corresponding addresses in the blanks provided. Note that one address has already been supplied for you.









Addr.		Data 15:12	11:8	7:4	3:0
0	00	0	7	2	A
1	01	1	3	4	B
2	10	F	C	A	2
3	11	B	E	E	F

Name: Solution

Part b: After rewiring the muxes correctly and initializing all memory cells to **xF**, the following sequence of accesses was performed. Note that some of the information about each access has been left out.

Your job: Fill in the blanks.

Read/Write	MDR	MAR
Write	x72A3	00
Write	x8FAF	11
Read	x72A3	00
Read	xFFFF	10
Write	x732D	11
Read	xFFFF	01
Write	x37A3	01
Read	x37A3	01
Read	x732D	11

Show the contents of the memory cells by putting the hex digit that is stored in each after all the accesses have been performed.

Address	D[15:12]	D[11:8]	D[7:4]	D[3:0]
00	7	2	A	3
01	3	7	A	3
10	F	F	F	F
11	7	3	2	D

The Standard ASCII Table

ASCII			ASCII			ASCII			ASCII		
Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex
nul	0	00	sp	32	20	@	64	40	`	96	60
soh	1	01	!	33	21	A	65	41	a	97	61
stx	2	02	"	34	22	B	66	42	b	98	62
etx	3	03	#	35	23	C	67	43	c	99	63
eot	4	04	$	36	24	D	68	44	d	100	64
enq	5	05	%	37	25	E	69	45	e	101	65
ack	6	06	&	38	26	F	70	46	f	102	66
bel	7	07	'	39	27	G	71	47	g	103	67
bs	8	08	(	40	28	H	72	48	h	104	68
ht	9	09	)	41	29	I	73	49	i	105	69
lf	10	0A	*	42	2A	J	74	4A	j	106	6A
vt	11	0B	+	43	2B	K	75	4B	k	107	6B
ff	12	0C	,	44	2C	L	76	4C	l	108	6C
cr	13	0D	-	45	2D	M	77	4D	m	109	6D
so	14	0E	.	46	2E	N	78	4E	n	110	6E
si	15	0F	/	47	2F	O	79	4F	o	111	6F
dle	16	10	0	48	30	P	80	50	p	112	70
dc1	17	11	1	49	31	Q	81	51	q	113	71
dc2	18	12	2	50	32	R	82	52	r	114	72
dc3	19	13	3	51	33	S	83	53	s	115	73
dc4	20	14	4	52	34	T	84	54	t	116	74
nak	21	15	5	53	35	U	85	55	u	117	75
syn	22	16	6	54	36	V	86	56	v	118	76
etb	23	17	7	55	37	W	87	57	w	119	77
can	24	18	8	56	38	X	88	58	x	120	78
em	25	19	9	57	39	Y	89	59	y	121	79
sub	26	1A	:	58	3A	Z	90	5A	z	122	7A
esc	27	1B	;	59	3B	[	91	5B	{	123	7B
fs	28	1C	<	60	3C	\	92	5C	|	124	7C
gs	29	1D	=	61	3D	]	93	5D	}	125	7D
rs	30	1E	>	62	3E	^	94	5E	~	126	7E
us	31	1F	?	63	3F	_	95	5F	del	127	7F