ECE 198JL Third Midterm Exam
Spring 2013

Tuesday, April 9th, 2013

Name: ____ NetID: ________________

Discussion Section:

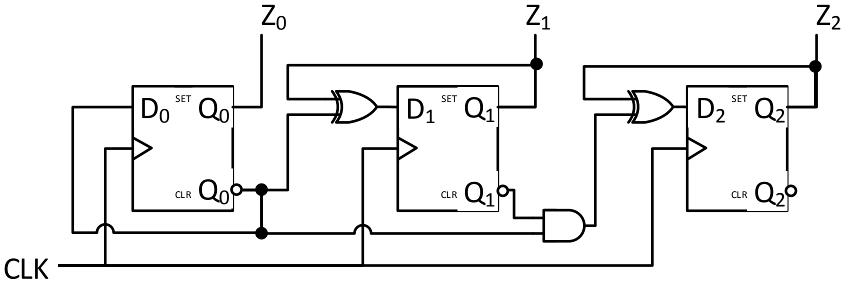
10:00 AM	[ ] JD1	
11:00 AM	[ ] JD2	
2:00 PM	[ ] JD3	
4:00 PM	[ ] JD4	

- **Be sure your exam booklet has 11 pages.**
- **Be sure to write your name and lab section on the first page.**
- **Do not tear the exam booklet apart; you can only detach the last page.**
- **We have provided LC-3 instructions set at the back.**
- **Use backs of pages for scratch work if needed.**
- **This is a closed book exam. You may not use a calculator.**
- **You are allowed one handwritten 8.5 x 11" sheet of notes.**
- **Absolutely no interaction between students is allowed.**
- **Be sure to clearly indicate any assumptions that you make.**
- **The questions are not weighted equally. Budget your time accordingly.**
- **Don't panic, and good luck!**

Problem 1	8 points:	___________
Problem 2	6 points:	___________
Problem 3	16 points:	___________
Problem 4	11 points:	___________
Problem 5	12 points:	___________
Problem 6	23 points:	___________
Problem 7	9 points:	___________
Problem 8	15 points:	___________
Total	100 points:	___________

Problem 1 (8 pts): Sequential circuit analysis

Consider the Finite State Machine (FSM) shown below that has three internal state bits, $Q_2Q_1Q_0$, and three outputs, $Z_2Z_1Z_0$, that match the internal state bits, that is, $Z_i = Q_i$.



1. Write Boolean expressions for the flip-flop inputs D_2, D_1 and D_0 (flip-flop excitation equations) as functions of current state $Q_2Q_1Q_0$:

D_2 = __________________________________

D_1 = __________________________________

D_0 = __________________________________

2. Fill in the next-state table and output table

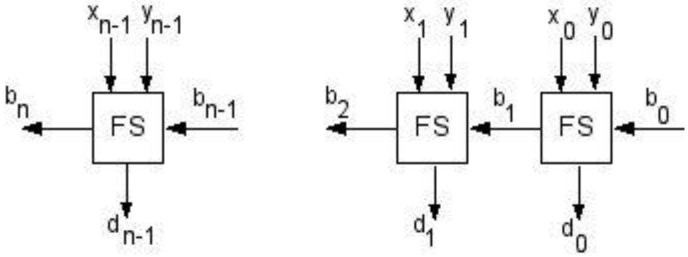
Current state			Next state		
Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

3. Draw the state transition diagram for this FSM.

4. Explain the function of this FSM.

Problem 2 (6 pts): Serial design

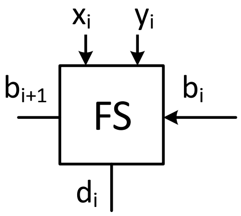
In Midterm 2 you designed a bit-sliced *ripple borrow* subtractor circuit consisting of *n* Full Subtractor (FS) bit slice cells:



1. How many flip-flops are needed to construct a *serial* subtractor circuit using a Mealy machine model?

Answer: ___________________

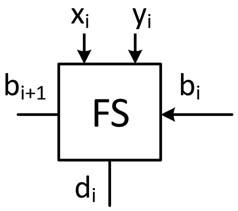
2. Re-draw the bit-sliced design shown above as a serial design using a Mealy machine model



3. How many flip-flops are needed to construct a *serial* subtractor circuit using a Moore machine model?

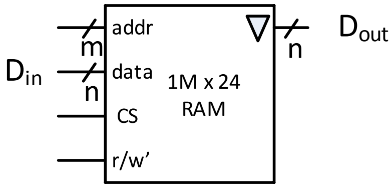
Answer: ___________________

4. Re-draw the bit-sliced design shown above as a serial design using a Moore machine model



Problem 3 (16 pts): RAM

Shown to the right is a 1M x 24 RAM. (1M = 2^{20}.)

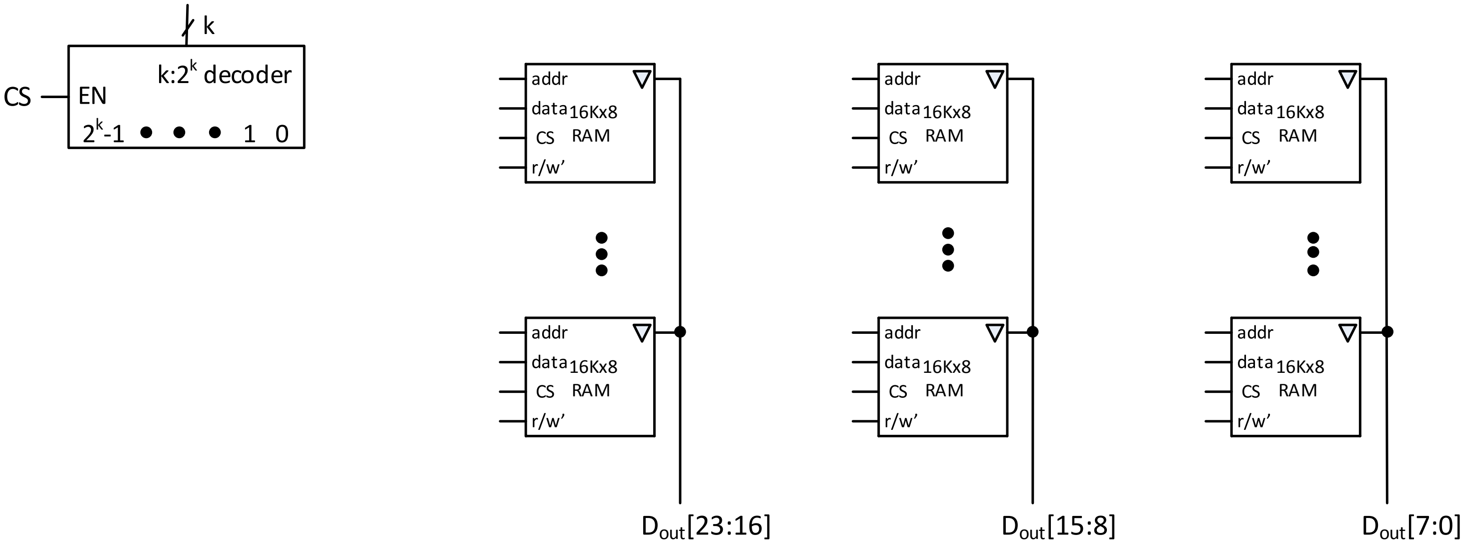


1. How many bits **addr**, **data**, **CS**, and **r/w'** ports require?

Answer: addr: __________ CS: _____

data: __________ r/w': _____

2. Implement this RAM using 16Kx8 RAM chips. Each 16Kx8 RAM chip has inputs **data**, **addr**, **CS**, and **r/w'**, and output gated by a tri-state buffer. Finish the implementation by drawing the missing connections and labeling all newly added wires. (You do not need to draw all the rows, they are shown as " ... ", but be sure the pattern is clear.) The RAM output wires and CS are already drawn for you.



3. How many rows of 16Kx8 RAM chips are needed and what is the value of *k*?

Number of rows: ________________ *k* = ______________?

4. Suppose you wish to store the decimal value -4 in memory at the address 16,416. In which 16Kx8 RAM chip(s) will this value be stored?

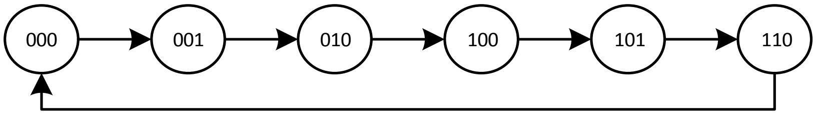
Answer: ____

5. What input values must be provided to the corresponding 16Kx8 RAM chip(s) in order to properly store this value? Give the **addr** and **data** values *for each chip* in hexadecimal.

addr: ______________ data: ____

Problem 4 (11 pts): Counter design

Using conventional sequential circuit design technique, implement a 3-bit synchronous counter with D flip-flops that counts in the following $Q_2Q_1Q_0$ sequence:



1. Show K-maps for D_2, D_1, D_0:

Q_1Q_0

		00	01	11	10
Q_2	0				
	1				

D_2

Q_1Q_0

		00	01	11	10
Q_2	0				
	1				

D_1

Q_1Q_0

		00	01	11	10
Q_2	0				
	1				

D_0

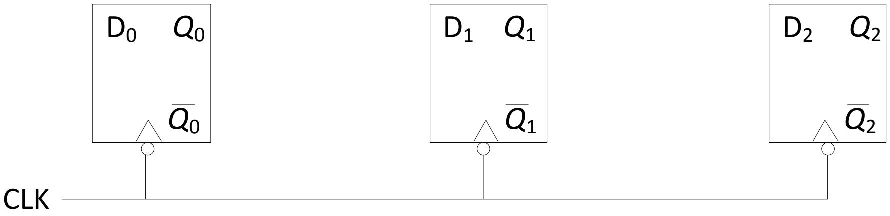
2. Write Boolean expressions for D_2, D_1, D_0:

D_2 = ______________________________________

D_1 = ______________________________________

D_0 = ______________________________________

3. Draw the circuit using only **2** additional gates.



4. Is your counter *self-starting*? Explain - and be specific to *your solution*. (Self-starting means that no matter in what state the counter starts, it always converges to produce the correct counting sequence.)

Problem 5 (12 pts): Sequence recognizer design

Design a sequence recognizing FSM with input *x* and output *z* such that the output is 1 if and only if pattern **01** or **110** has been detected in the input stream. Make sure to detect overlapping patterns.

Example:

Input:	0	0	1	0	1	0	1	1	0	1	1	0	1	0	1	1	0	1	0. . .
Output:	0	0	1	0	1	0	1	0	1	1	0	1	1	0	1	0	1	1	0. . .

1. Draw the Moore state diagram, using as few states as possible. (Solutions with more than 6 states are not acceptable.) Let A be the start state. Each edge must be labeled with *x* and each node must be labeled with *name*/*z*.

2. Write down *state meaning* and choose their binary *representations* $S_2S_1S_0$:

Problem 6 (23 pts): Two-floor elevator controller FSM

You are tasked with designing a two-floor elevator electronic control system. The elevator moves between just two floors, floor #1 and floor #2. It is controlled by 2 buttons: UP and DOWN. When a user presses button UP, input signal *U* becomes 1, otherwise it is 0. When user presses button DOWN, input signal *D* becomes 1, otherwise it is 0. The control unit generates output signals G_u (go up) and G_d (go down) to control the electric motor that moves the elevator one floor up (when G_u=1) or down (when G_d=1). When user presses both buttons at the same time, the elevator should not move.

1. Design a Moore state machine implementing this system. Determine the states you need, giving them meaningful names so that we can comprehend your design intent, and sketch the complete state transition diagram for your FSM, specifying inputs *U* and *D* for each transition and outputs G_u and G_d for each state. (*You do not need more than 4 states.*)

2. Fill in the truth table below by assigning internal state bit representations S_1S_0 and output values G_d and G_u for each of your states from Part 1 and then filling in the values for the next-state variables S_1^+ and S_0^+ for each combination of S_1S_0UD.

State name and meaning	Current state		External Inputs		Next state		External outputs	
	S_1	S_0	U	D	S_1^+	S_0^+	G_u	G_d
			0	0				
			0	1				
			1	0				
			1	1				
			0	0				
			0	1				
			1	0				
			1	1				
			0	0				
			0	1				
			1	0				
			1	1				
			0	0				
			0	1				
			1	0				
			1	1				

3. Using the truth table that you wrote in Part 2, fill in the K-maps for S_1^+, S_0^+, G_d, and G_u and derive Boolean expressions for these variables.

		S_1	
		0	1
S_0	0		
	1		

G_u = ________________________________

		S_1	
		0	1
S_0	0		
	1		

G_d = ________________________________

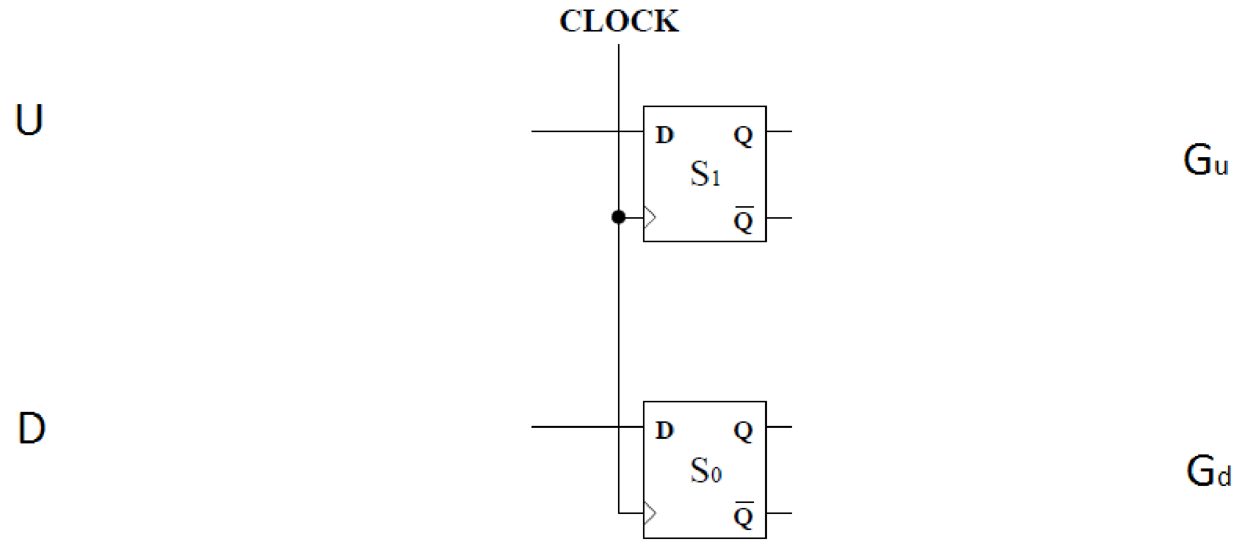
		S_1S_0			
		00	01	11	10
UD	00				
	01				
	11				
	10				

S_1^+ = ____________________________________

		S_1S_0			
		00	01	11	10
UD	00				
	01				
	11				
	10				

S_0^+ = ____________________________________

4. Add gates as necessary to complete the implementation of your FSM using the two positive edge-triggered D flip-flops shown below.



Problem 7 (9 pts): Von Neumann architecture

1. Name all components of the von Neumann architecture and briefly describe their function.

2. List all phases of the von Neumann instruction cycle.

3. Describe the purpose of the following registers in the von Neumann architecture.

PC:

IR:

MDR:

MAR:

General-purpose register file:

Problem 8 (15 pts): LC-3 ISA

The following LC-3 program fragment, represented as three hexadecimal numbers, is stored in memory at the indicated locations and the following values are stored in registers:

address	instruction
…	
x4001	x6081
x4002	x1204
x4003	x0110
…	

register	value
R0	x4000
R1	x4001
R2	x4002
R3	x4003
R4	x4004

1. Re-write first two instructions in binary representation and provide their corresponding RTL. (*Note: formats of the entire LC-3 instruction set are provided at the end of the exam booklet.*)

address	instruction	binary instruction	RTL (be specific to this instruction)
…			
x4001	x6081		
x4002	x1204		
…			

2. Assuming PC is initially set to x4001, trace the execution of **two** steps of the given program segment and fill in the table below. Write down values stored in PC, IR, MAR, MDR, N, Z, and P registers **at the end of the instruction cycle**. Values for PC, IR, MAR, and MDR should be written in hexadecimal. Values for N, Z, and P, if known, should be written in binary, or marked with X if they cannot be determined based on the provided information.

PC	**IR**	**MAR**	**MDR**	**N**	**Z**	**P**

3. What hexadecimal value will be stored in R1 once the two instructions are executed?

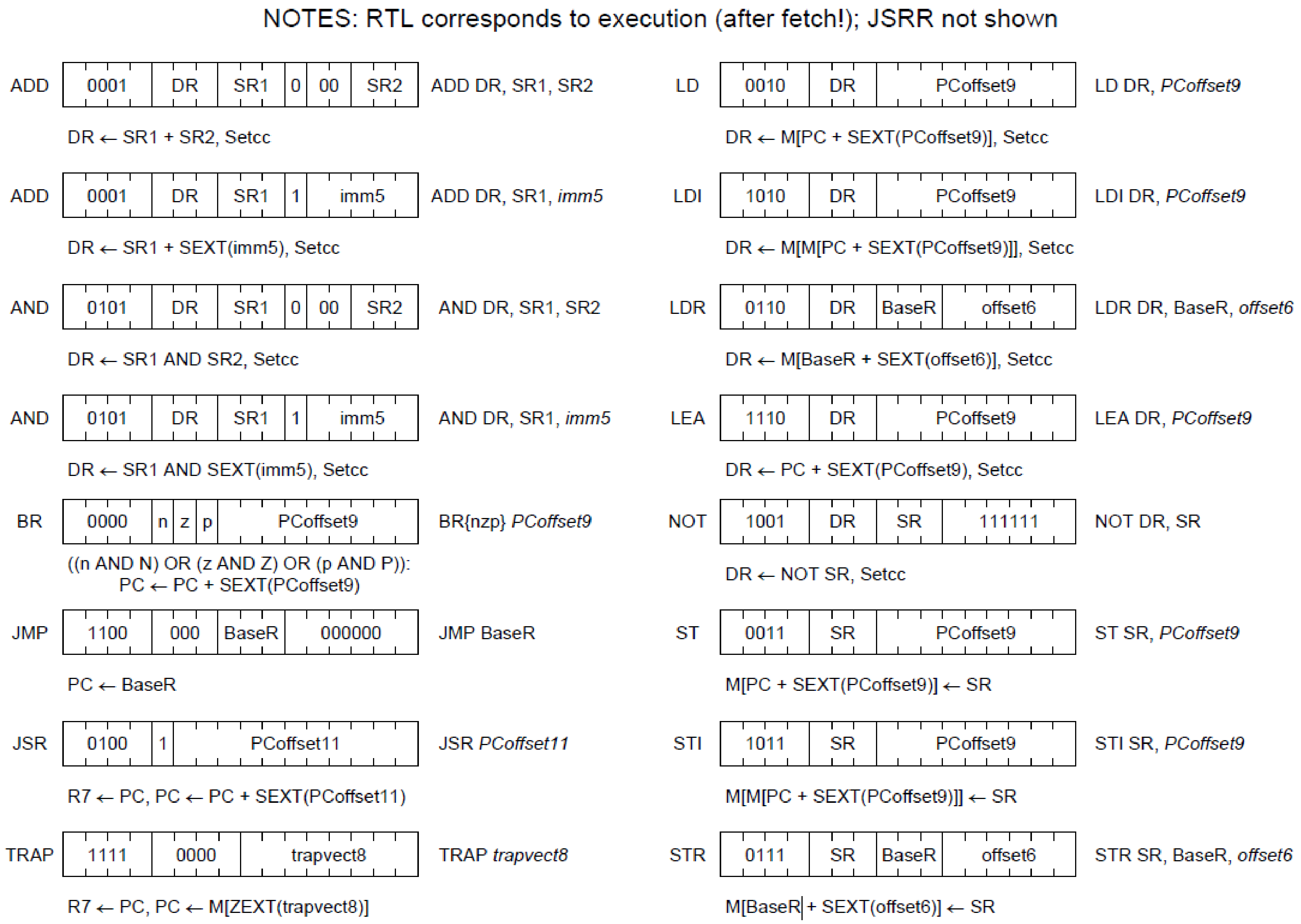
Answer: ________________________

4. What is the address of the next instruction to be executed after executing these two instructions?

Answer: ________________________

5. How many memory reads will take place in order to execute these two instructions, including instruction FETCH?

6. Answer: ________________________



NOTES: RTL corresponds to execution (after fetch!); JSRR not shown

ADD	0001	DR	SR1	0	00	SR2	ADD DR, SR1, SR2	DR ← SR1 + SR2, Setcc
ADD	0001	DR	SR1	1	imm5		ADD DR, SR1, *imm5*	DR ← SR1 + SEXT(imm5), Setcc
AND	0101	DR	SR1	0	00	SR2	AND DR, SR1, SR2	DR ← SR1 AND SR2, Setcc
AND	0101	DR	SR1	1	imm5		AND DR, SR1, *imm5*	DR ← SR1 AND SEXT(imm5), Setcc
BR	0000	n	z	p	PCoffset9		BR{nzp} *PCoffset9*	((n AND N) OR (z AND Z) OR (p AND P)): PC ← PC + SEXT(PCoffset9)
JMP	1100	000	BaseR	000000			JMP BaseR	PC ← BaseR
JSR	0100	1	PCoffset11				JSR *PCoffset11*	R7 ← PC, PC ← PC + SEXT(PCoffset11)
TRAP	1111	0000	trapvect8				TRAP *trapvect8*	R7 ← PC, PC ← M[ZEXT(trapvect8)]
LD	0010	DR	PCoffset9				LD DR, *PCoffset9*	DR ← M[PC + SEXT(PCoffset9)], Setcc
LDI	1010	DR	PCoffset9				LDI DR, *PCoffset9*	DR ← M[M[PC + SEXT(PCoffset9)]], Setcc
LDR	0110	DR	BaseR	offset6			LDR DR, BaseR, *offset6*	DR ← M[BaseR + SEXT(offset6)], Setcc
LEA	1110	DR	PCoffset9				LEA DR, *PCoffset9*	DR ← PC + SEXT(PCoffset9), Setcc
NOT	1001	DR	SR	111111			NOT DR, SR	DR ← NOT SR, Setcc
ST	0011	SR	PCoffset9				ST SR, *PCoffset9*	M[PC + SEXT(PCoffset9)] ← SR
STI	1011	SR	PCoffset9				STI SR, *PCoffset9*	M[M[PC + SEXT(PCoffset9)]] ← SR
STR	0111	SR	BaseR	offset6			STR SR, BaseR, *offset6*	M[BaseR + SEXT(offset6)] ← SR