Digital Electronics & Computer Engineering (E85)

Harris Fall 2019

Final Exam

This is a closed-book take-home exam. Electronic devices including calculators are not allowed, except on the computer question on the last page. You are permitted two 8.5x11" sheets of paper with notes.

You are bound by the HMC Honor Code while taking this exam.

The first part of the exam is written, while the final page is done on the computer based on your E85 Lab 11. The entire Lab 11 that you use (including datapath and controller) must be your own work; it cannot, for example, include somebody else's controller. The exam is intended to be doable in 3 hours if you have prepared adequately. However, there will be no limit on the time you are allowed except that the written portion must be completed in one contiguous block of time and the computer part must be completed in another contiguous block of time. A contiguous block of time is a period of time working at a desk without breaking for meals, naps, socializing, etc. You cannot study for E85 or consult E85 resources between the two blocks of time. Please manage your time wisely and do not let the exam expand to take more time than is justified.

Return the exam to the E85 box in the Engineering Department Office no later than Wednesday 12/18 at noon.

Alongside each question, the number of points is written in brackets. All work and answers should be written directly on this examination booklet, except for printouts. Use the backs of pages if necessary. Write neatly; illegible answers will be marked wrong. Show your work for partial credit.

Name: _____

Do Not Write Below This Point

Page 2:	_____	/ 4
Page 3:	_____	/ 4
Page 4:	_____	/ 7
Page 5:	_____	/ 3
Page 6:	_____	/ 5
Page 7:	_____	/ 5
Page 9:	_____	/ 4
Page 10:	_____	/ 4
Page 11:	_____	/ 8
Page 16:	_____	/ 6
Total:	_____	/ 50

[4] Interpret A = 0x40D40000 and B = 0xC0C80000 as IEEE single-precision floating point numbers. Compute their sum C = A + B, and write the result as a floating-point number in hexadecimal.

Decimal Value of A: _____

Decimal Value of B: _____

Decimal Value of C: _____

Hexadecimal Representation of C: _____

An AND-OR-INVERT-22 (AOI22) gate computes Y = ~(AB + CD). The symbol is shown below. An AOI22 is considered one compound gate, not three individual gates.

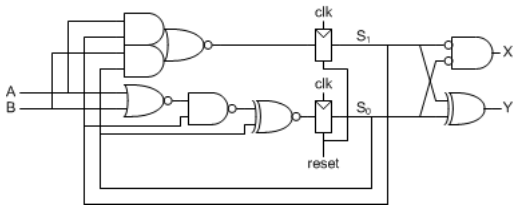
[2] Using only an AOI22 gate and inverters, sketch a schematic for a 2:1 multiplexer. Let your inputs be S, D0, and D1, and your output be Y.

[2] Is it possible to perform any arbitrary function of 2 variables using a single AOI22 gate and as many inverters as you want? Explain why, or give a counterexample.

YES / NO

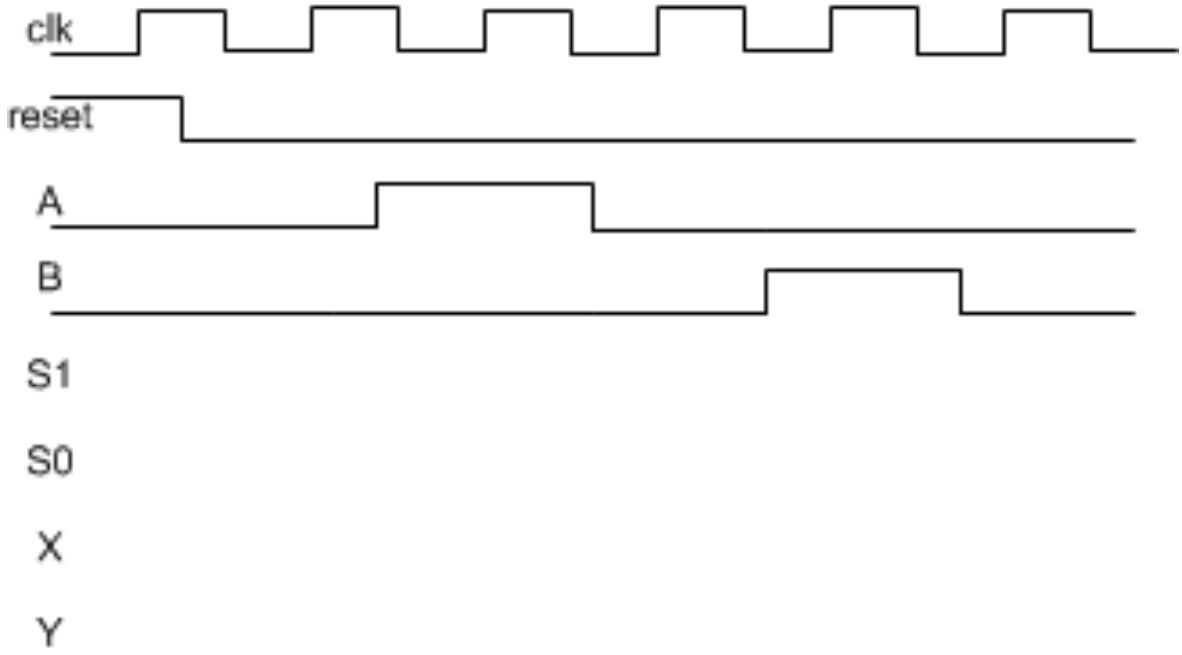
(explanation / counterexample)

Consider the circuit below. Notice that the circuit uses an AOI22 gate defined in the previous problem.



[4] Draw a state transition diagram for the circuit.

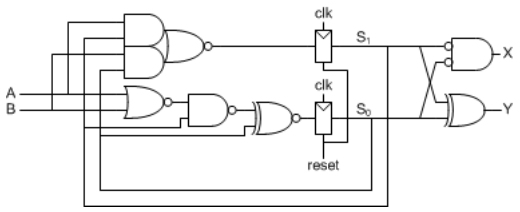
[3] Suppose the inputs below are applied. Sketch the behavior of the states and outputs.



[3] Write Behavioral Verilog code for the circuit based on your state transition diagram.

```
module final(input  logic clk, reset,
             input  logic a, b,
             output logic x, y);









endmodule
```

Consider the same circuit, repeated here for your convenience.



[2] How many logic elements are required to build this circuit on our Spartan FPGA?

Logic Elements _____

[3] Suppose that each input or clock pin of each gate or register has 3 fF of capacitance. (Note that the AOI22 has 4 input pins.) The X and Y outputs each drive 10 fF of capacitance. Suppose that the data signals have an average activity factor α of 0.1, and reset has an activity factor of approximately 0. The power supply voltage is $\sqrt{2}$ V, and the circuit operates at 1 GHz. Compute the power consumption of the circuit.

Power (Watts) _____

[5] Write an assembly language function to compare two strings. The answer should be 1 if the first string comes earlier in alphabetical (ASCII) order than the second, -1 if the first string comes later than the second, and 0 if the strings are identical. When the function is called, R0 and R1 contain the base address of two null-termianted strings. Return your result in R0. Hint: `MVN R0, #0` puts -1 in R0.

Use ASCII order for comparison of nonalphabetic characters. For example,

`strcmp("Alicia", "Ben") = 1`

`strcmp("MIT", "HMC") = -1`

`strcmp("Finals", "Finals") = 0`

`strcmp("E85A", "E85") = -1`

`strcmp("cash", "ca$h") = -1` because the ASCII value for s is 115 and for $ is 36.

[4] Refer to the memory maps below using the I2C1 Inter-integrated circuit port. Write a C function to set the 10-bit Own Address field (OA1) of the I2C_OAR1 to 0x047 and then wait until the Receive Not Empty (RXNE) field of the I2_ISR is true. Don't rely on any libraries; define the register addresses yourself.

Table 101. I2C register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0	**I2C_CR1**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PECEN	ALERTEN	SMBDEN	SMBHEN	GCEN	WUPEN	NOSTRETCH	SBC	RXDMAEN	TXDMAEN	Res.	ANFOFF	DNF[3:0]				ERRIE	TCIE	STOPIE	NACKIE	ADDRIE	RXIE	TXIE	PE
	Reset value									0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0
0x4	**I2C_CR2**	Res.	Res.	Res.	Res.	Res.	PECBYTE	AUTOEND	RELOAD	NBYTES[7:0]								NACK	STOP	START	HEAD10R	ADD10	RD_WRN	SADD[9:0]									
	Reset value						0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x8	**I2C_OAR1**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OA1EN	Res.	Res.	Res.	Res.	OA1MODE	OA1[9:0]									
	Reset value																	0					0	0	0	0	0	0	0	0	0	0	0
0xC	**I2C_OAR2**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OA2EN	Res.	Res.	Res.	Res.	OA2MSK [2:0]	OA2[7:1]									Res.
	Reset value																	0					0	0	0	0	0	0	0	0	0	0	
0x10	**I2C_TIMINGR**	PRESC[3:0]				Res.	Res.	Res.	Res.	SCLDEL[3:0]				SDADEL[3:0]				SCLH[7:0]								SCLL[7:0]							
	Reset value	0	0	0	0					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	**I2C_TIMEOUTR**	TEXTEN	Res.	Res.	Res.	TIMEOUTB[11:0]												TIMOUTEN	Res.		TIDLE	TIMEOUTA[11:0]												
	Reset value	0				0	0	0	0	0	0	0	0	0	0	0	0	0			0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	**I2C_ISR**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ADDCODE[6:0]							DIR	BUSY	Res.	ALERT	TIMEOUT	PECERR	OVR	ARLO	BERR	TCR	TC	STOPF	NACKF	ADDR	RXNE	TXIS	TXE
	Reset value									0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 1. STM32F0xx peripheral register boundary addresses (continued)

Bus	Boundary address	Size	Peripheral	Peripheral register map
	0x4000 7C00 - 0x4000 7FFF	1 KB	Reserved	
	0x4000 7800 - 0x4000 7BFF	1 KB	CEC	*Section 31.7.7 on page 910*
	0x4000 7400 - 0x4000 77FF	1 KB	DAC	*Section 14.10.15 on page 291*
	0x4000 7000 - 0x4000 73FF	1 KB	PWR	*Section 5.4.3 on page 92*
	0x4000 6C00 - 0x4000 6FFF	1 KB	CRS	*Section 7.6.5 on page 147*
	0x4000 6800 - 0x4000 6BFF	1 KB	Reserved	
	0x4000 6400 - 0x4000 67FF	1 KB	CAN	*Section 29.9.5 on page 854*
	0x4000 6000 - 0x4000 63FF	1 KB	USB/CAN SRAM	*Section 30.6.3 on page 890*
	0x4000 5C00 - 0x4000 5FFF	1 KB	USB	*Section 30.6.3 on page 890*
	0x4000 5800 - 0x4000 5BFF	1 KB	I2C2	*Section 26.7.12 on page 685*
	0x4000 5400 - 0x4000 57FF	1 KB	I2C1	*Section 26.7.12 on page 685*

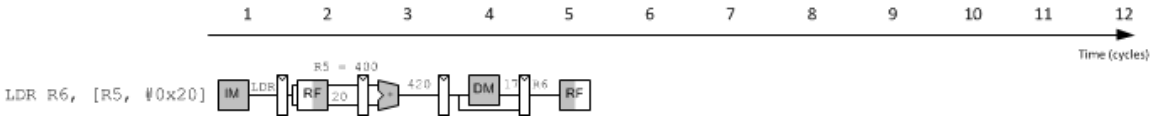
(put solution on next page)

```
void i2c_twiddling(void) {

}
```

Consider the following program. Initially, suppose R5 contains the value 0x0400 and memory location 0x420 contains the value 17.

```
LDR R6, [R5, #0x20]
ADD R7, R6, #1
SUBS R1, R5, #42
BNE AROUND
SUB R8, R5, #30
ADD R8, R8, R7
STR R8, [R5, #0x24]
AROUND
STR R7, [R5, #0x28]
```

The program is executed on our pipelined processor with the same hazard handling you considered in class and Problem Set 10. The behavior of the pipeline for the first instruction is illustrated below. For example, in cycle 2, the value 0x0400 is read from the first port of the register file.



[1] What value is written to the register file on cycle 5? _____

[1] What does the ALU do on cycle 6? _____

[1] In which cycle is the memory written? _____

[1] Which memory address is written? _____

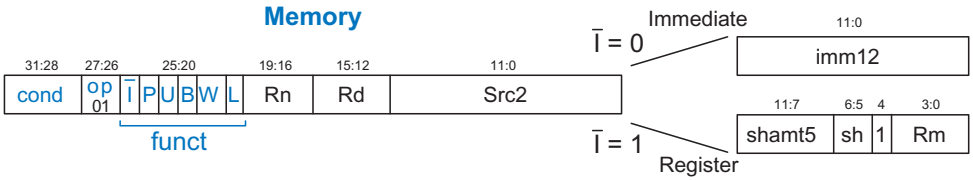
The ARM `LDR Rd, [Rn, imm]!` is like an ordinary LDR but also updates the base address Rn = Rn + imm. This is called a pre-indexed load. For example, if R1 is initially 0x1000, and memory location 0x1008 contains 42, then `LDR R2, [R1, #8]` puts 42 in R2 and 0x1008 in R1. In ARM machine language, the W bit of the function field is 1 for pre-indexed loads and 0 for ordinary loads. Note that Ibar = 0, P = 1, U = 1, B = 0, and L = 1 for both kinds of loads. Modify the ARM multicycle processor to support the preindexed load instruction, using as little additional hardware as feasible.

[3] Mark up the attached multicycle processor diagram and ALU to handle the new instructions.

[2] Mark up the attached multicycle controller (including state transition diagram and truth tables) to handle the new instructions.

[2] The attached multicycle memfile.s test code has highlighted modifications to test the new instruction. As compared to the memfile.s from Lab 11, it replaces the LDR instruction at 0x4C with an a pre-indexed load.

Translate the pre-indexed command to machine language. Express your code in hexadecimal. The format for a memory instruction is given below. The cond field for ALWAYS is 1110. Hint: $231_{10} = E7_{16}$.



LDR R3, [R2, #231]! _____

[1] Predict what value should be written to mem[248] at the last line of the program.

Predicted Value: _____

Multicycle Processor

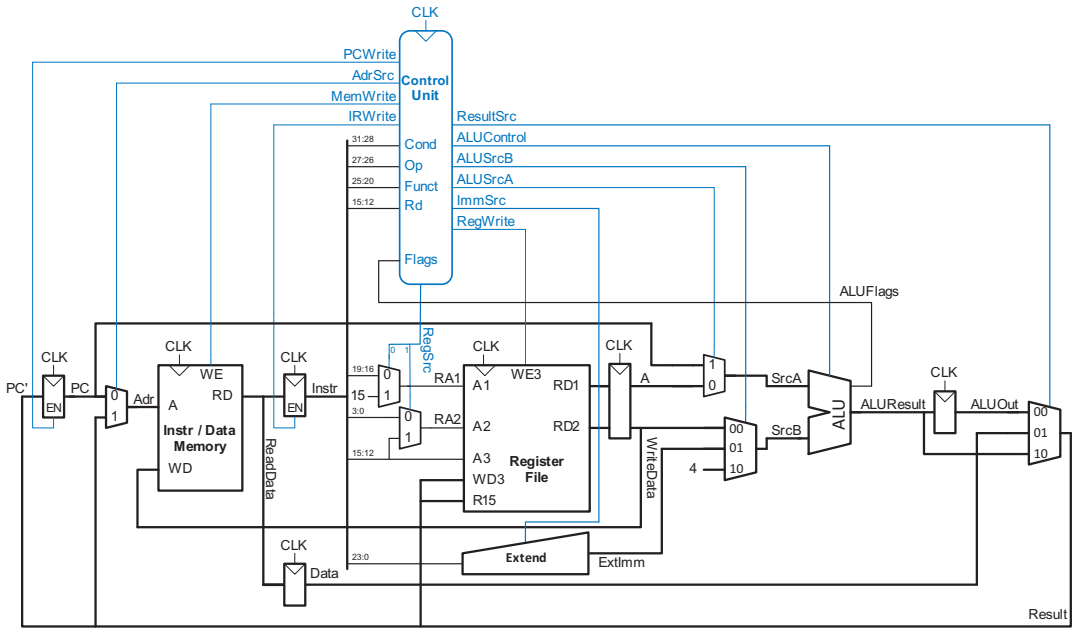
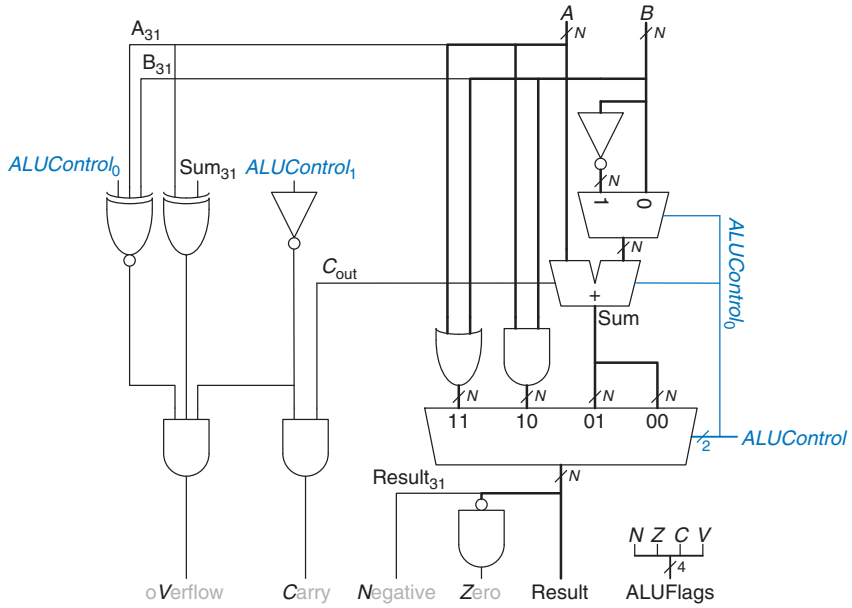


Figure 7.30 Complete multicycle processor

ALU



Multicycle Controller

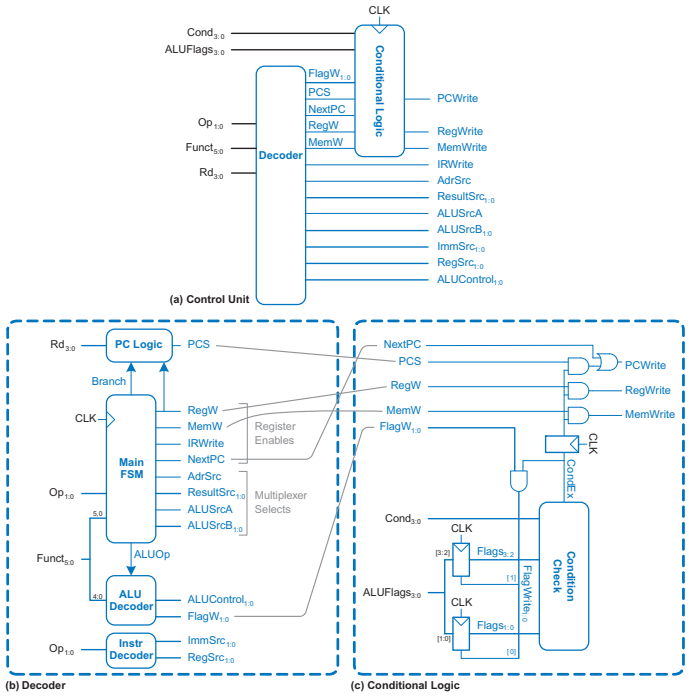


Figure 7.31 Multicycle control unit

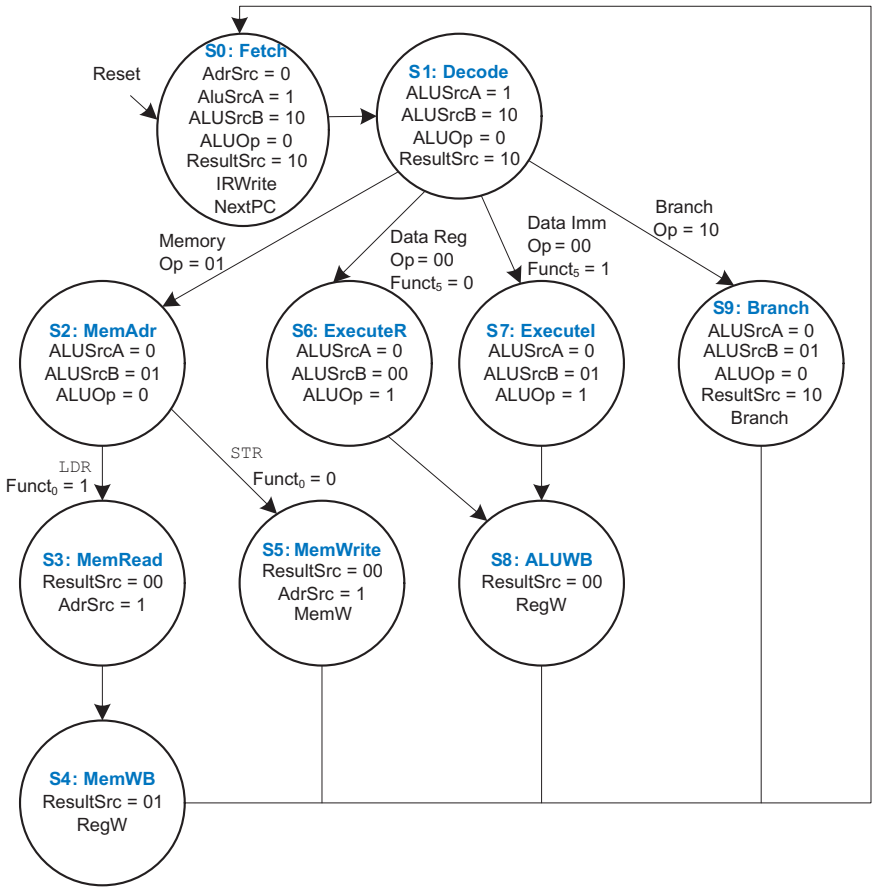


Table 7.6 Instr Decoder logic for *RegSrc* and *ImmSrc*

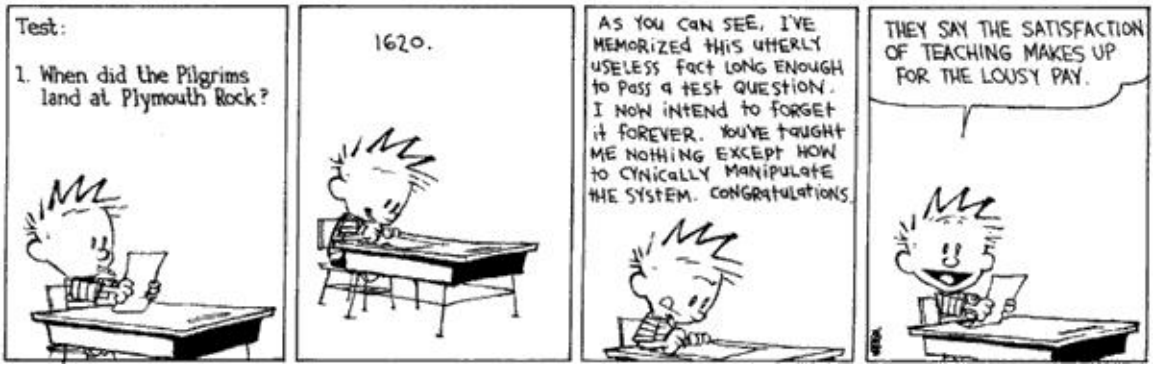
Instruction	Op	$Funct_5$	$Funct_0$	$RegSrc_1$	$RegSrc_0$	$ImmSrc_{1:0}$
LDR	01	X	1	X	0	01
STR	01	X	0	1	0	01
DP immediate	00	1	X	X	0	00
DP register	00	0	X	0	0	00
B	10	X	X	X	1	10

$ALUOp$	$Funct_{4:1}$ (cmd)	$Funct_0$ (S)	Type	$ALUControl_{1:0}$	$FlagW_{1:0}$
0	X	X	Not DP	00 (Add)	00
1	0100	0	ADD	00 (Add)	00
		1			11
	0010	0	SUB	01 (Sub)	00
		1			11
	0000	0	AND	10 (And)	00
		1			10
	1100	0	ORR	11 (Or)	00
		1			10

```
        ; memfile.dat
MAIN
 SUB R0, R15, R15      ; R0 = 0            1110 000 0010 0 1111 0000 0000 0000 1111 E04F000F 0x00
 ADD R2, R0, #5        ; R2 = 5            1110 001 0100 0 0000 0010 0000 0000 0101 E2802005 0x04
 ADD R3, R0, #12       ; R3 = 12           1110 001 0100 0 0000 0011 0000 0000 1100 E280300C 0x08
 SUB R7, R3, #9        ; R7 = 3            1110 001 0010 0 0011 0111 0000 0000 1001 E2437009 0x0c
 ORR R4, R7, R2        ; R4 = 3 OR 5 = 7   1110 000 1100 0 0111 0100 0000 0000 0010 E1874002 0x10
 AND R5, R3, R4        ; R5 = 12 AND 7 = 4 1110 000 0000 0 0011 0101 0000 0000 0100 E0035004 0x14
 ADD R5, R5, R4        ; R5 = 4 + 7 = 11   1110 000 0100 0 0101 0101 0000 0000 0100 E0855004 0x18
 SUBS R5, R5, #10      ; R5 = 11 - 10 = 1  1110 001 0010 1 0101 0101 0000 0000 1010 E255500A 0x1c
 SUBSGT R5, R5, #2     ; R5 = 1 - 2 = -1   1100 001 0010 1 0101 0101 0000 0000 0010 C2555002 0x20
 ADD R5, R5, #12       ; R5 = -1 + 12 = 11 1110 001 0100 0 0101 0101 0000 0000 1100 E285500C 0x24
 SUBS R8, R5, R7       ; R8 = 11 - 3 = 8   1110 000 0010 1 0101 1000 0000 0000 0111 E0558007 0x28
 BEQ END               ; not taken         0000 1010 0000  0000 0000 0000 0000 1100 0A00000F 0x2c
 SUBS R8, R3, R4       ; R8 = 12 - 7  = 5  1110 000 0010 1 0011 1000 0000 0000 0100 E0538004 0x30
 BGE AROUND            ; should be taken   1010 1010 0000  0000 0000 0000 0000 0000 AA000000 0x34
 ADD R5, R0, #0        ; should be skipped 1110 001 0100 0 0000 0101 0000 0000 0000 E2805000 0x38
AROUND
 SUBS R8, R7, R2       ; R8 = 3 - 5 = -2   1110 000 0010 1 0111 1000 0000 0000 0010 E0578002 0x3c
 ADDLT R7, R5, #1      ; R7 = 11+1 = 12    1011 001 0100 0 0101 0111 0000 0000 0001 B2857001 0x40
 SUB R7, R7, R2        ; R7 = 12-5 = 7     1110 000 0010 0 0111 0111 0000 0000 0010 E0477002 0x44
 STR R7, [R3, #224]    ; mem[12+224] = 7   1110 010 1100 0 0011 0111 0000 0101 0100 E58370E0 0x48
 LDR R3, [R2, #231]!   ;
 ADD R15, R15, R0      ; PC <- PC + 8      1110 000 0100 0 1111 1111 0000 0000 0000 E08FF000 0x50
 ADD R2, R0, #14       ; shouldn't happen  1110 001 0100 0 0000 0010 0000 0000 0001 E280200E 0x54
 B END                 ; always taken      1110 1010 0000  0000 0000 0000 0000 0001 EA000001 0x58
 ADD R2, R0, #13       ; shouldn't happen  1110 001 0100 0 0000 0010 0000 0000 0001 E280200D 0x5C
 ADD R2, R0, #10       ; shouldn't happen  1110 001 0100 0 0000 0010 0000 0000 0001 E280200A 0x60
END
 STR R2, [R0, #248]    ; mem[248] = ?      1110 010 1100 0 0000 0010 0000 1111 1000 E58020F8 0x64
```

END OF WRITTEN PORTION OF EXAM

DO NOT PROCEED PAST THIS POINT UNTIL YOU ARE PREPARED TO CEASE ALL WORK ON THE WRITTEN PORTION AND MOVE ON TO THE COMPUTER PORTION.



COMPUTER PORTION OF EXAM

Once you start this question, you may refer to the written portion of the exam, but may not spend any more time on the written portion or change any of your answers on that portion.

Modify your ARM multicycle processor from Lab 11 to support the pre-indexed load instruction. Modify your memfile.dat to replace the existing LDR instruction with the preindexed load.

[2] Print out your Verilog code and circle or highlight the lines you modified.

[4] Print out a simulation waveform showing at least the value being written to memory location 248 on the last cycle. Circle this value in the waveform.