Digital Electronics & Computer Engineering (E85)

Harris Spring 2018

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. Please manage your time wisely and do not let the exam expand to take more time than is justified.

Return the exam to Sydney Torrey in the Engineering Department Office no later than Friday 5/11 at noon (5/4 at 5 pm for seniors).

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:	_____	/ 6
Pages 3-4:	_____	/ 12
Page 5:	_____	/ 6
Page 6:	_____	/ 3
Page 11:	_____	/ 6
Page 12:	_____	/ 3
Page 13-16:	_____	/ 8
Page 18:	_____	/ 6
Total:	_____	/ 50

[1] Give an expression for the maximum value of an `int` in C on a 32-bit microprocessor?

Maximum Value: _____

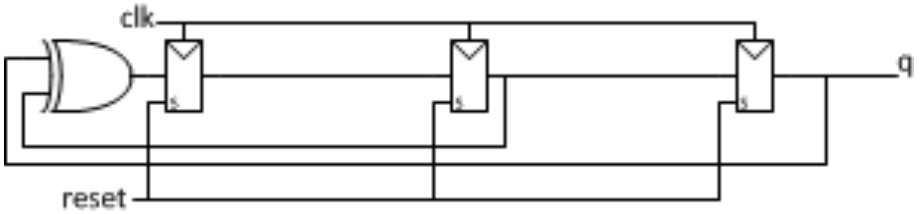
[2] Give the closest approximation to π that you can represent with an 8-bit two's complement fixed point number with four fractional bits. Express your result in hexadecimal.

Approximation of π: _____

[3] Determine the decimal value of the IEEE single precision floating point number C0C80000

Number: _____

Consider the following "LFSR" circuit. Each flip-flop has a set input s to initialize its output to 1 when reset is applied.



[1] Show the sequence of values q takes on for the next 10 cycles after the circuit is reset.

Cycle	q
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

[2] Give a succinct description of the circuit in Verilog.

```
module lfsr(input  logic clk, reset
            output logic q)




endmodule
```

The XOR has a propagation delay of 20 ps and a contamination delay of 10 ps. It's input capacitance is 1 fF on each input pin and the leakage current is 10 nA. The flip-flop has a clock-to-Q propagation delay of 25 ps and contamination delay of 16 ps. Its setup time is 9 ps and its hold time is 12 ps. It's input capacitance is 0.667 fF on the D pin and 2 fF on the CLK pin, and the leakage current is 30 nA. You also have buffers available with a propagation delay of 8 ps and contamination delay of 6 ps. The power supply is 0.8 V.

[2] What is the fastest clock rate at which the LFSR circuit could operate in the absence of skew?

Fastest Clock: _____ (ps)

[2] What is the static power consumption P_{static} of the circuit?

P_{static}: _____

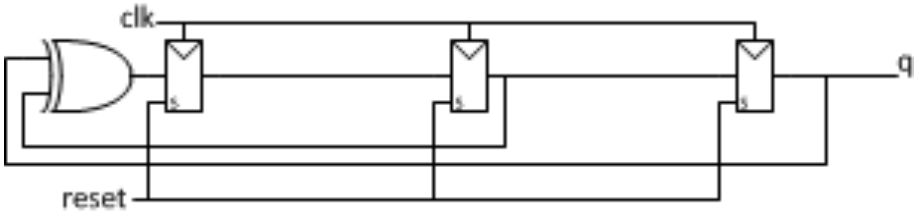
[2] What is the dynamic power consumption of the circuit operating at 1 MHz $P_{dynamic\text{-}1MHz}$?

$P_{dynamic\text{-}1MHz}$: _____

[1] Above what clock frequencies does the dynamic power consumption exceed the static consumption? Give your answer in terms of P_{static} and $P_{dynamic\text{-}1MHz}$.

Frequency: _____

[2] Mark up the circuit with minimum modifications necessary for the circuit to operate correctly if the clock skew between flip-flops may be as much as 11 ps.



[6] Translate the following function to ARM assembly language. `str` is passed in R0. Remember that R1-R3 and R12 do not need to be saved or restored across a function call.

```
void toupper(char str[])
     int i = 0;

     while (str[i]) {
          if (str[i] > 96) str[i] = str[i] - 32;
          i++;
     }
}
```

Assembly language

[3] Translate the following ARM assembly language into machine language. Refer to the instruction encodings on the next page. Express your instructions in hexadecimal.

do	**Machine Language**
LDR R2, [R5, R6]	_____
CMP R2, #42	_____
BLE do	_____

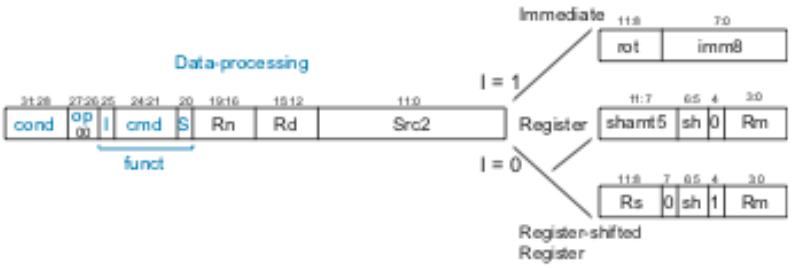


Figure B.1 Data-processing instruction encodings

Table B.1 Data-processing instructions

cmd	Name	Description	Operation
0000	`AND Rd, Rn, Src2`	Bitwise AND	`Rd ← Rn & Src2`
0001	`EOR Rd, Rn, Src2`	Bitwise XOR	`Rd ← Rn ^ Src2`
0010	`SUB Rd, Rn, Src2`	Subtract	`Rd ← Rn - Src2`
0011	`RSB Rd, Rn, Src2`	Reverse Subtract	`Rd ← Src2 - Rn`
0100	`ADD Rd, Rn, Src2`	Add	`Rd ← Rn+Src2`
0101	`ADC Rd, Rn, Src2`	Add with Carry	`Rd ← Rn+Src2+C`
0110	`SBC Rd, Rn, Src2`	Subtract with Carry	`Rd ← Rn - Src2 - C̄`
0111	`RSC Rd, Rn, Src2`	Reverse Sub w/ Carry	`Rd ← Src2 - Rn - C̄`
1000 ($S = 1$)	`TST Rd, Rn, Src2`	Test	`Set flags based on Rn & Src2`
1001 ($S = 1$)	`TEQ Rd, Rn, Src2`	Test Equivalence	`Set flags based on Rn ^ Src2`
1010 ($S = 1$)	`CMP Rn, Src2`	Compare	`Set flags based on Rn - Src2`
1011 ($S = 1$)	`CMN Rn, Src2`	Compare Negative	`Set flags based on Rn+Src2`
1100	`ORR Rd, Rn, Src2`	Bitwise OR	`Rd ← Rn | Src2`
1101	Shifts:		
$I = 1$ OR ($\text{instr}_{11:4} = 0$)	`MOV Rd, Src2`	Move	`Rd ← Src2`
$I = 0$ AND ($sh = 00$; $\text{instr}_{11:4} \neq 0$)	`LSL Rd, Rm, Rs/shamt5`	Logical Shift Left	`Rd ← Rm<< Src2`
$I = 0$ AND ($sh = 01$)	`LSR Rd, Rm, Rs/shamt5`	Logical Shift Right	`Rd ← Rm>> Src2`

P	W	Index Mode
0	0	Post-index
0	1	Not supported
1	0	Offset
1	1	Pre-index

L	B	Instruction
0	0	STR
0	1	STRB
1	0	LDR
1	1	LDRB

Branch

cond	Mnemonic	Name	CondEx
0000	EQ	Equal	Z
0001	NE	Not equal	$\overline{Z}$
0010	CS/HS	Carry set / unsigned higher or same	C
0011	CC/LO	Carry clear / unsigned lower	$\overline{C}$
0100	MI	Minus / negative	N
0101	PL	Plus / positive or zero	$\overline{N}$
0110	VS	Overflow / overflow set	V
0111	VC	No overflow / overflow clear	$\overline{V}$
1000	HI	Unsigned higher	$\overline{Z}C$
1001	LS	Unsigned lower or same	Z OR $\overline{C}$
1010	GE	Signed greater than or equal	$\overline{N \oplus V}$
1011	LT	Signed less than	$N \oplus V$
1100	GT	Signed greater than	$\overline{Z}(\overline{N \oplus V})$
1101	LE	Signed less than or equal	Z OR $(N \oplus V)$
1110	AL (or none)	Always / unconditional	Ignored

The STM32 has a 12-bit analog-to-digital converter (ADC). To initialize the ADC:

- Enable the ADC clock in the Reset and Clock Control register.
- Turn on the ADC by setting the ADEN bit.
- Wait until the ADRDY flag is set to indicate the ADC is ready for conversion
- Write 111 to the SMP bits to run the sampling clock sufficiently slow.

To convert the voltage from channel n:

- Write a 1 to the CHSELn bit and 0s to other CHSEL bits to select channel n for conversion.
- Start a conversion by setting the ADSTART bit
- Wait for the ADSTART bit to go low to indicate the conversion is complete
- Read the answer from the ADC_DR data register

Relevant register maps are given below:

Table 1. STM32F0xx peripheral register boundary addresses (continued)

Bus	Boundary address	Size	Peripheral	Peripheral register map
APB	0x4001 5C00 - 0x4001 7FFF	9 KB	Reserved	
	0x4001 5800 - 0x4001 5BFF	1 KB	DBGMCU	*Section 32.9.6 on page 924*
	0x4001 4C00 - 0x4001 57FF	3 KB	Reserved	
	0x4001 4800 - 0x4001 4BFF	1 KB	TIM17	*Section 20.6.16 on page 545*
	0x4001 4400 - 0x4001 47FF	1 KB	TIM16	*Section 20.6.16 on page 545*
	0x4001 4000 - 0x4001 43FF	1 KB	TIM15	*Section 20.5.18 on page 528*
	0x4001 3C00 - 0x4001 3FFF	1 KB	Reserved	
	0x4001 3800 - 0x4001 3BFF	1 KB	USART1	*Section 27.8.12 on page 753*
	0x4001 3400 - 0x4001 37FF	1 KB	Reserved	
	0x4001 3000 - 0x4001 33FF	1 KB	SPI1/I2S1	*Section 28.9.10 on page 813*
	0x4001 2C00 - 0x4001 2FFF	1 KB	TIM1	*Section 17.4.21 on page 391*
	0x4001 2800 - 0x4001 2BFF	1 KB	Reserved	
	0x4001 2400 - 0x4001 27FF	1 KB	ADC	*Section 13.12.11 on page 267*
	0x4001 2000 - 0x4001 23FF	1 KB	Reserved	
	0x4001 1C00 -0x4001 1FFF	1 KB	USART8	*Section 27.8.12 on page 753*
	0x4001 1800 - 0x4001 1BFF	1 KB	USART7	*Section 27.8.12 on page 753*
	0x4001 1400 - 0x4001 17FF	1 KB	USART6	*Section 27.8.12 on page 753*
	0x4001 0800 - 0x4001 13FF	3 KB	Reserved	
	0x4001 0400 - 0x4001 07FF	1 KB	EXTI	*Section 11.3.7 on page 219*
	0x4001 0000 - 0x4001 03FF	1 KB	SYSCFG COMP	*Section 9.1.38 on page 185* *Section 15.5.2 on page 300*
	0x4000 8000 - 0x4000 FFFF	32 KB	Reserved	

Table 50. ADC 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
0x00	**ADC_ISR**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	AWD	Res.	Res.	OVR	EOSEQ	EOC	EOSMP	ADRDY
	Reset value																									0			0	0	0	0	0
0x04	**ADC_IER**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	AWDIE	Res.	Res.	OVRIE	EOSEQIE	EOCIE	EOSMPIE	ADRDYIE
	Reset value																									0			0	0	0	0	0
0x08	**ADC_CR**	ADCAL	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ADSTP	Res.	ADSTART	ADDIS	ADEN
	Reset value	0																											0		0	0	0
0x0C	**ADC_CFGR1**	Res.	AWDCH[4:0]						Res.	Res.	AWDEN	AWDSGL	Res.	Res.	Res.	Res.	DISCEN	AUTOFF	WAIT	CONT	OVRMOD	EXTEN[1:0]		Res.	EXTSEL [2:0]			ALIGN	RES [1:0]		SCANDIR	DMACFG	DMAEN
	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
0x10	**ADC_CFGR2**	CKMODE[1:0]		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
	Reset value	0	0																														
0x14	**ADC_SMPR**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SMP [2:0]		
	Reset value																														0	0	0
0x18	Reserved	Reserved																															
0x1C	Reserved	Reserved																															
0x20	**ADC_TR**	Res.	Res.	Res.	Res.	HT[11:0]												Res.	Res.	Res.	Res.	LT[11:0]											
	Reset value					1	1	1	1	1	1	1	1	1	1	1	1					0	0	0	0	0	0	0	0	0	0	0	0
0x24	Reserved	Reserved																															
0x28	**ADC_CHSELR**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CHSEL18	CHSEL17	CHSEL16	CHSEL15	CHSEL14	CHSEL13	CHSEL12	CHSEL11	CHSEL10	CHSEL9	CHSEL8	CHSEL7	CHSEL6	CHSEL5	CHSEL4	CHSEL3	CHSEL2	CHSEL1	CHSEL0
	Reset value														0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x2C 0x30 0x34 0x38 0x3C	Reserved	Reserved																															
0x40	**ADC_DR**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DATA[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x44 … 0x304	Reserved	Reserved																															
0x308	**ADC_CCR**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	VBATEN	TSEN	VREFEN	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
	Reset value								0	0	0																						

[3] Complete the following function to initialize the ADC for a particular channel. The clock enabling code is provided for you.

```
void adcInit(int channel) {
      volatile unsigned long *RCC_APB2ENR = (unsigned long*)0x40021018;
      // Enable ADC clock by writing 1 to ADCEN bit of RCC_APB2ENR
      *RCC_APB2ENR |= 1<<9;









}
```

[3] Complete the following function to read a value from a specified channel:

```
int analogRead(int channel) {









}
```

Consider the 5-stage pipelined ARM processor with hazard unit from Chapter 7 running the following program. Assume that it has been enhanced to support the MOV instruction. The MOV instruction is issued on cycle 1.

```
MOV R3, #1
ADD R4, R3, #7
SUB R5, R3, R3
LDR R6, R3, R4
ADD R7, R6, R3
```

[1] What operation does the ALU perform on cycle 4? _____

[1] What is the output of the ALU on cycle 5? _____

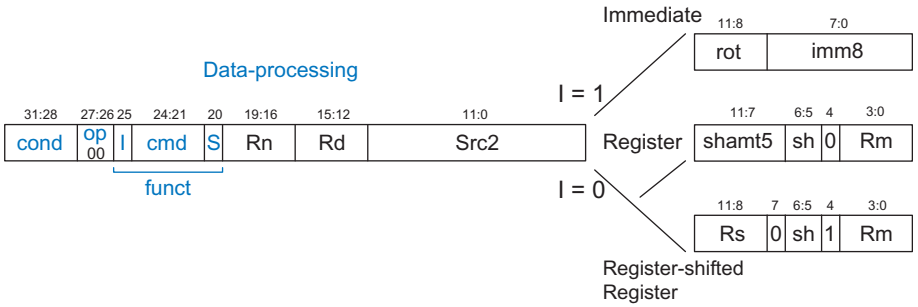
[1] In which cycle is R7 written? _____

The ARM `MOV Rd, Rn` instruction copies the value from Rn into Rd. It is a data processing instruction with a cmd field of 1101. MOVS does the same, and affects the N and Z flags. Modify the ARM multicycle processor to support the MOV and MOVS instructions, 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. Translate these three new lines of assembly to machine language. Express your code in hexadecimal. The format for a data processing instruction is given below. The cmd field for ORR is 1100 and the cond field for ALWAYS is 1110.



MOV R9, R2: _____

ORR R9, R9, #17: _____

MOV R2, R9: _____

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

Predicted Value: _____

Multicycle Processor

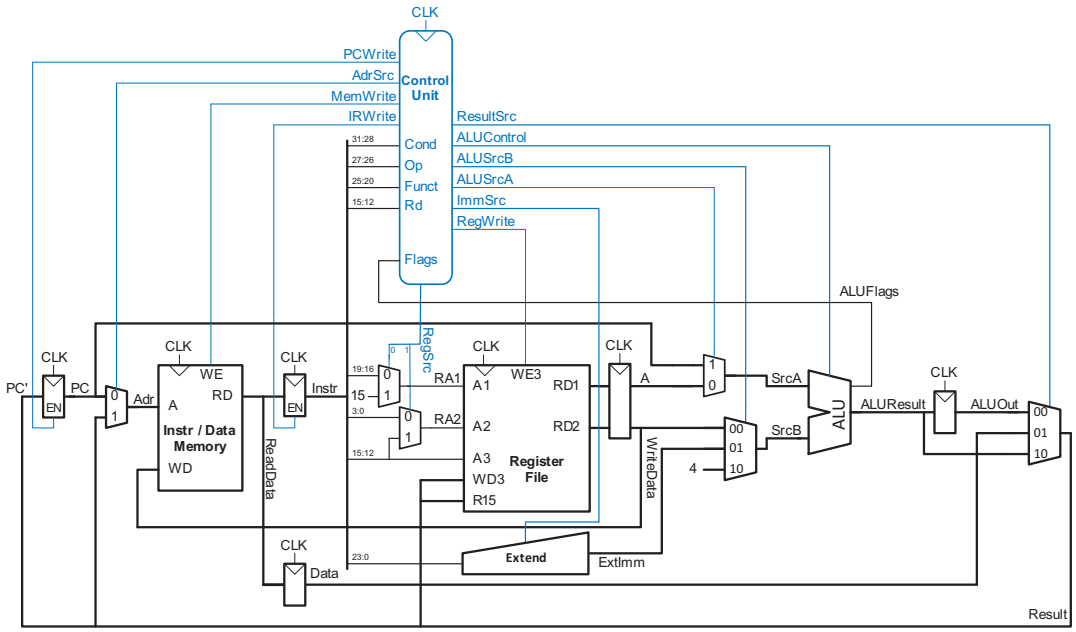
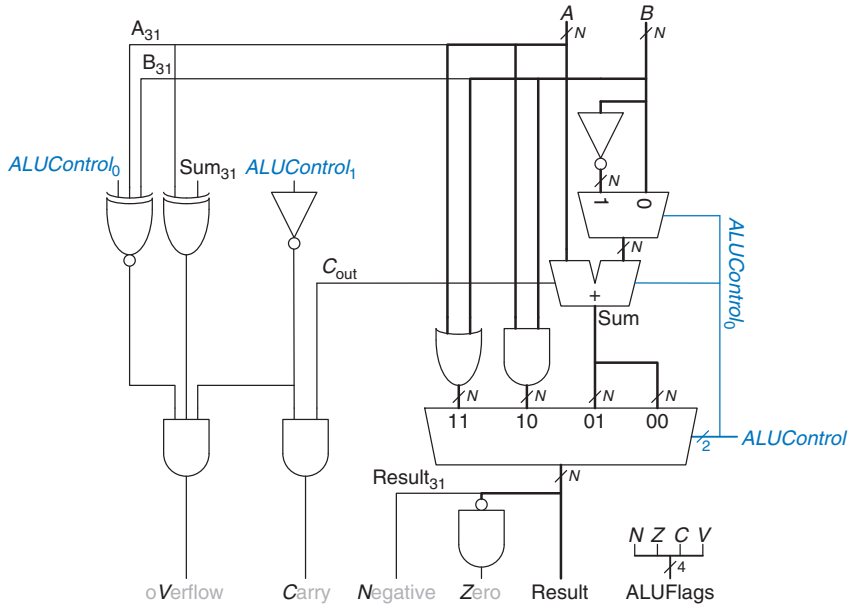


Figure 7.30 Complete multicycle processor

ALU



Multicycle Controller

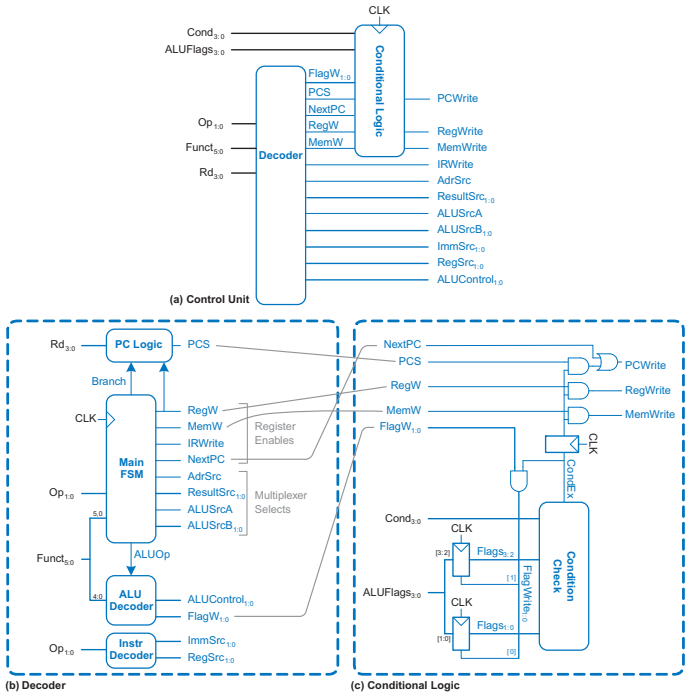


Figure 7.31 Multicycle control unit

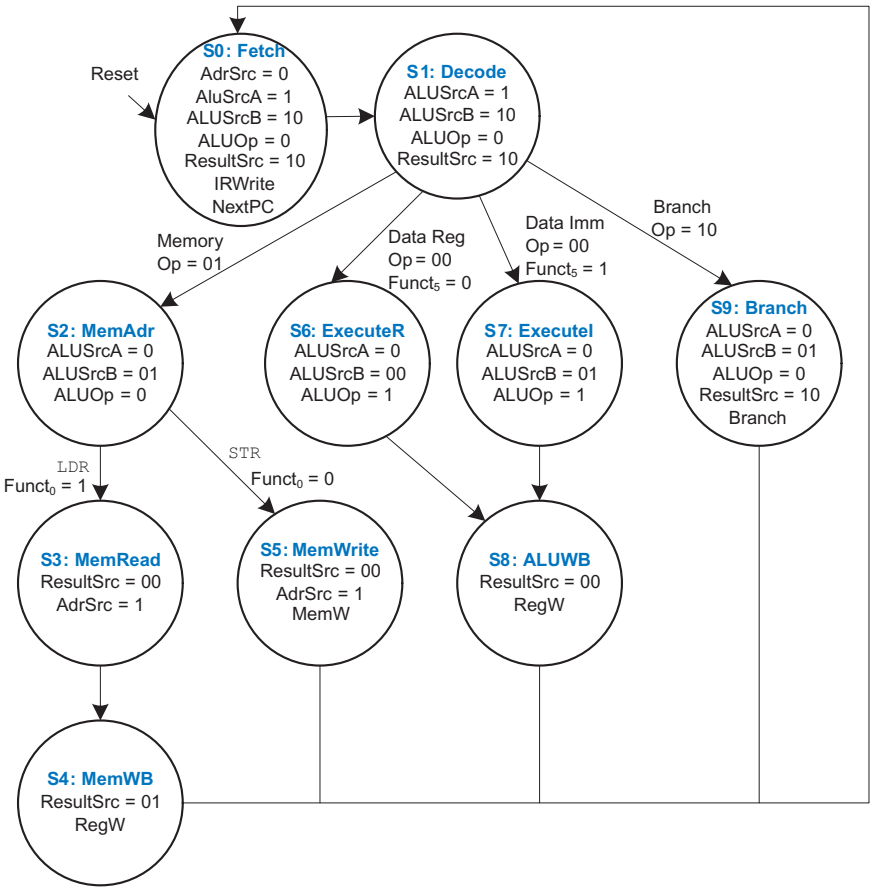


Table 7.6 Instr Decoder logic for *RegSrc* and *ImmSrc*

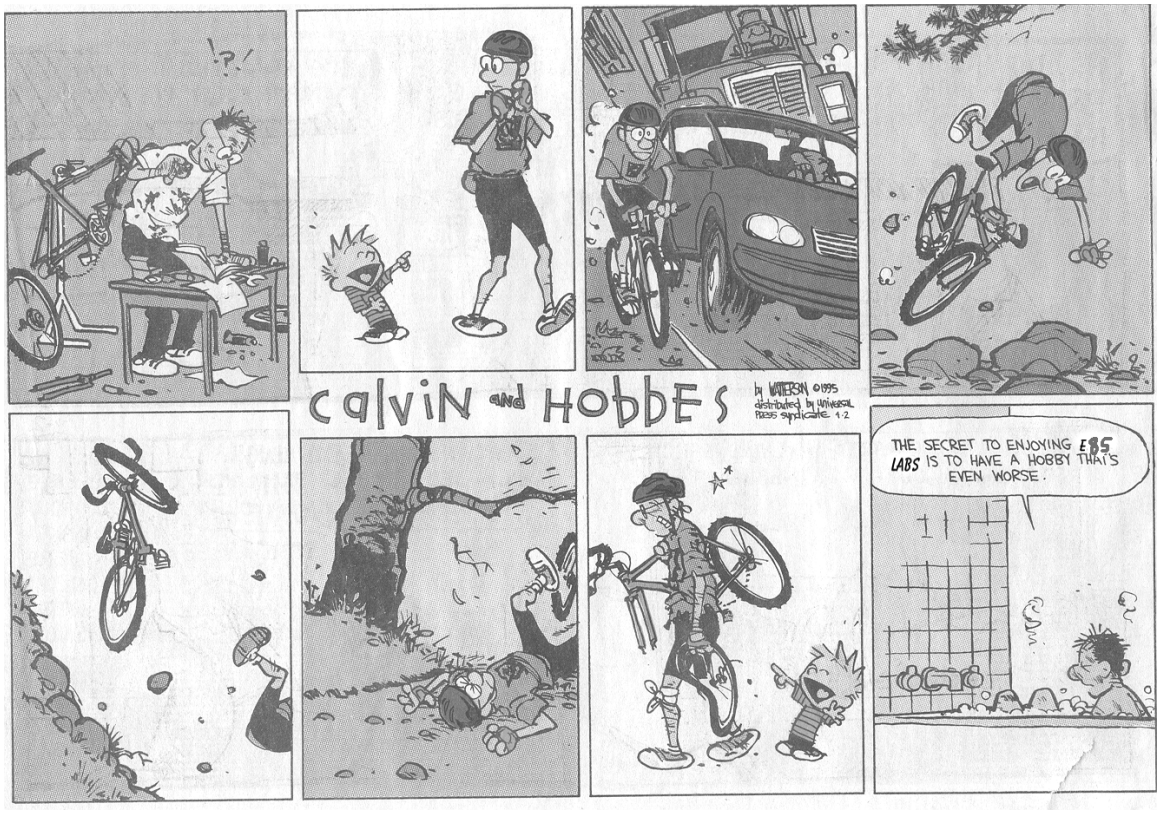
Instruction	Op	$Funct_5$	$Funct_0$	$RegSrc_1$	$RegSrc_0$	$ImmSrc_{1:0}$
LDR	01	X	1	X	0	01
STR	01	X	0	1	0	01
DP immediate	00	1	X	X	0	00
DP register	00	0	X	0	0	00
B	10	X	X	X	1	10

$ALUOp$	$Funct_{4:1}$ (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 R2, [R0, #236]     ; R2 = mem[236] = 7   1110 010 1100 1 0000 0010 0000 0110 0000 E59020EC 0x4c
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
MOV R9, R2
ORR R9, R9, #17
MOV R2, R9
B END                  ; always taken       1110 1010 0000  0000 0000 0000 0000 0001 EA000001 0x64
ADD R2, R0, #13        ; shouldn't happen   1110 001 0100 0 0000 0010 0000 0000 0001 E280200D 0x68
ADD R2, R0, #10        ; shouldn't happen   1110 001 0100 0 0000 0010 0000 0000 0001 E280200A 0x6C
END
STR R2, [R0, #248]     ; mem[248] = ?       1110 010 1100 0 0000 0010 0000 1111 1000 E58020F8 0x70
```

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 MOV instruction. Modify your memfile.dat to add the three new lines of machine language code from the previous question. Simulate your modified code.

[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.