CIS 240 Fall 2019: Midterm

Oct 23, 2019

Name :

_____

Please write your name on the exam **and** the exam booklet and turn in both. You can answer the questions on this exam sheet or in the exam booklet. Please number the questions you are addressing clearly in the exam booklet.

Question 1 {25 pts}

Your job is to design a circuit that will take as input a 3-bit value, I and produce a 2-bit output O, which indicates the number of 1 bits in the input. For example, if I = 101 then O should be 10, if I = 010 then O = 01. In your diagram I_2, I_1 and I_0 should indicate the 3 bits of the input and O_1 and O_0 the two bits of the output where O_1 is the MSB. Remember that we cannot grade what we cannot read so please make your diagrams as neat and clear as possible.

a) **{5 pts}** First provide a truth table indicating what the output O should be for every possible value of the input I.
b) **{5 pts}** Design 2 PLA circuits to produce the two output bits O_1 and O_0.
c) **{10 pts}** Design 2 CMOS circuits to produce the two output bits O_1 and O_0. You can assume that you have access to negated versions of all of the input bits. Your solution should consist of one CMOS network for each output bit. It should not involve cascading multiple CMOS circuits.
d) **{5 pts}** If the PLA circuit you provide for part b is ultimately implemented on the same kind of CMOS technology that you would use for your answer to part c which implementation do you think would have the lower overall delay? Explain your answer.

Question 2 {15 pts}

a) List every LC4 instruction that requires the Privilege.CTL signal to be set to 1
b) List every LC4 instruction that requires the ALU.CTL signal to be set to 5
c) List every LC4 instruction that requires the ALU.CTL signal to be set to 6
d) List every LC4 instruction that requires the DATA.WE signal to be set to 1
e) List every LC4 instruction that requires the regInputMux.CTL signal to be set to 2

Question 3 {20 pts}

The table below shows the contents of a region of User Code memory in PennSim. First convert the machine instructions you see here to an equivalent sequence of assembly instructions so you can read them. Note the assembly program on this answer sheet or in your test booklet. After you have done this, show what would happen when the program is executed by filling in the second table which shows the state of all of the registers at the **start** of each instruction cycle. Fill in the NZP entries with N for negative, Z for zero and P for positive. You must fill in the PC and NZP for every clock cycle, for the register values R0-R7 you only need to fill in the values of the registers that have changed from the previous cycle, if any. You should enter all values as decimal numbers, not hex.

Hint: start by looking at the four bit opcode – be careful how you break up the 16 bit fields, one bit can make a big difference.

Address	Machine Instruction	Assembly Instruction
0	`0001010000001001`	
1	`0000100000000010`	
2	`0000001000000010`	
3	`0000111111111111`	
4	`1001000000000100`	
5	`0001001001111111`	
6	`1100111111111001`	

Instruction Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
PC	0														
NZP	P														
R0	-3														
R1	2														
R2	0														
R3	0														
R4	0														
R5	0														
R6	0														
R7	0														

Question 4 {15 pts}

Your job in this question is to design part of the Decoder circuit for our Single Cycle LC4 implementation.

a) **{10 pts}** Produce a PLA circuit that takes as input the relevant bits from the current instruction and produces as output the regFile.WE signal.
b) **{5 pts}** Produce a second PLA circuit that generates the NZP.WE signal. Hint you can use the regFile.WE signal as an input to this second circuit.

Please use the convention I_{15}, I_{14}, … ,I_0 to refer to bits in the instruction word where I_{15} is the MSB and I_0 the LSB. Please note that we are asking for a PLA implementation specifically, alternative implementations will receive less points.

Question 5 {5 pts}

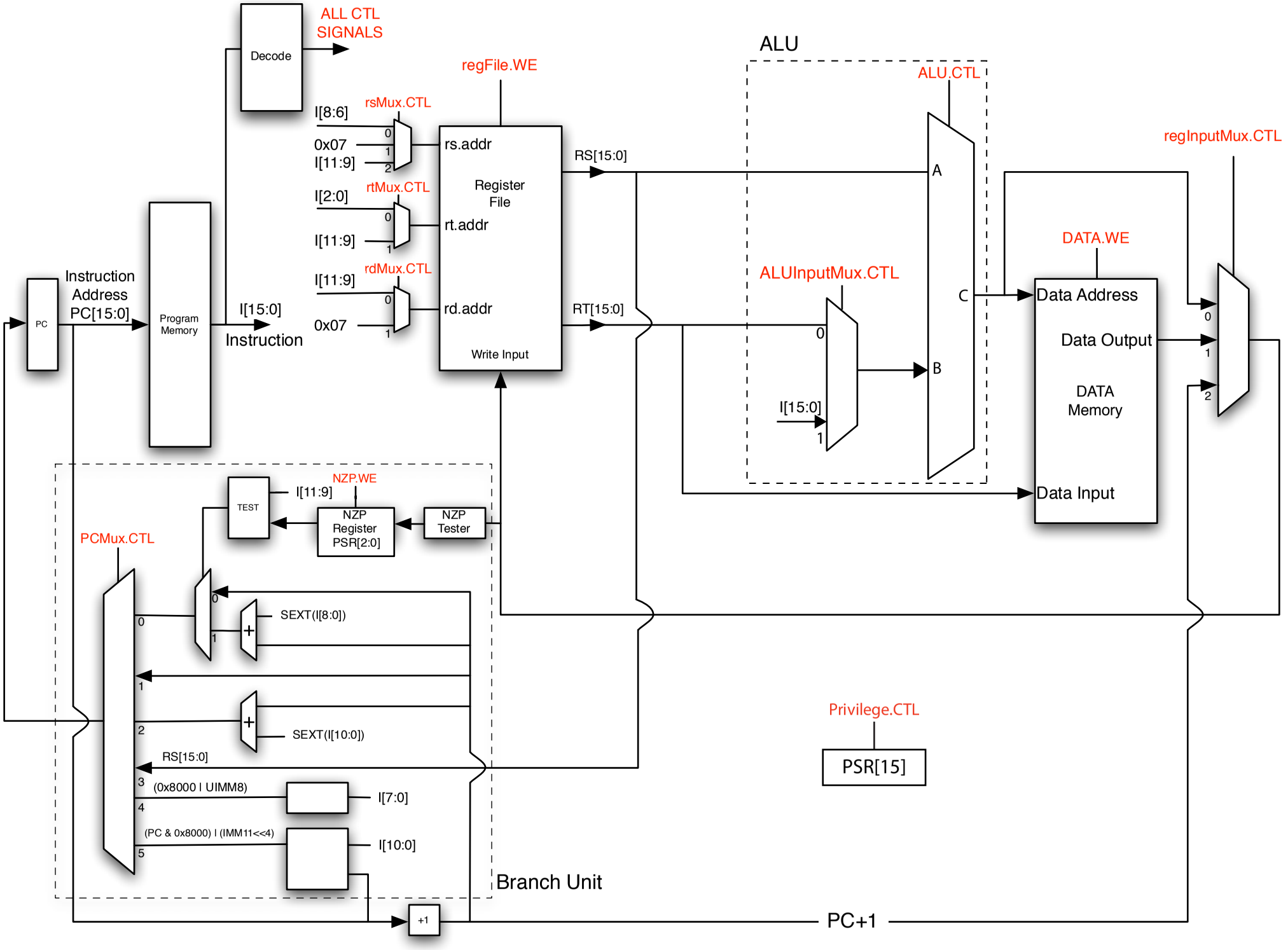
One odd feature of the LC4 instruction set is that the MOD operation does not have the same opcode as the other arithmetic operations that it is most closely related to namely: ADD, MUL, SUB, DIV and ADD Immediate. Explain briefly why we cannot give MOD the same opcode as these other operations given the way that these other instructions are currently encoded.

LC4 Instruction Set Reference v. 2017-01		
Mnemonic	**Semantics**	**Encoding**
NOP	PC = PC + 1	0000 000x xxxx xxxx
BRp <Label>	(P) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 001i iiii iiii
BRz <Label>	(Z) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 010i iiii iiii
BRzp <Label>	(Z|P) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 011i iiii iiii
BRn <Label>	(N) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 100i iiii iiii
BRnp <Label>	(N | P) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 101i iiii iiii
BRnz <Label>	(N|Z) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 110i iiii iiii
BRnzp <Label>	(N|Z|P) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 111i iiii iiii
ADD Rd Rs Rt	Rd = Rs + Rt	0001 ddds ss00 0ttt
MUL Rd Rs Rt	Rd = Rs * Rt	0001 ddds ss00 1ttt
SUB Rd Rs Rt	Rd = Rs - Rt	0001 ddds ss01 0ttt
DIV Rd Rs Rt	Rd = Rs / Rt	0001 ddds ss01 1ttt
ADD Rd Rs IMM5	Rd = Rs + sext(IMM5)	0001 ddds ss1i iiii
MOD Rd Rs Rt	Rd = Rs % Rt	1010 ddds ss11 xttt
AND Rd Rs Rt	Rd = Rs & Rt	0101 ddds ss00 0ttt
NOT Rd Rs	Rd = ~Rs	0101 ddds ss00 1xxx
OR Rd Rs Rt	Rd = Rs | Rt	0101 ddds ss01 0ttt
XOR Rd Rs Rt	Rd = Rs ∧ Rt	0101 ddds ss01 1ttt
AND Rd Rs IMM5	Rd = Rs & sext(IMM5)	0101 ddds ss1i iiii
LDR Rd Rs IMM6	Rd = dmem[Rs + sext(IMM6)]	0110 ddds ssii iiii
STR Rt Rs IMM6	dmem[Rs + sext(IMM6)] = Rt	0111 ttts ssii iiii
CONST Rd IMM9	Rd = sext(IMM9)	1001 ddd i iiii iiii
HICONST Rd UIMM8	Rd = (Rd & 0xFF) | (UIMM8 << 8) [1]	1101 dddx uuuu uuuu
CMP Rs Rt	NZP = sign(Rs - Rt) [2]	0010 sss0 0xxx xttt
CMPU Rs Rt	NZP = sign(uRs - uRt) [3]	0010 sss0 1xxx xttt
CMPI Rs IMM7	NZP = sign(Rs - sext(IMM7))	0010 sss1 0iii iiii
CMPIU Rs UIMM7	NZP = sign(uRs - UIMM7)	0010 sss1 1uuu uuuu
SLL Rd Rs UIMM4	Rd = Rs << UIMM4	1010 ddds ss00 uuuu
SRA Rd Rs UIMM4	Rd = Rs >>> UIMM4	1010 ddds ss01 uuuu
SRL Rd Rs UIMM4	Rd = Rs >> UIMM4	1010 ddds ss10 uuuu
JSRR Rs	R7 = PC + 1; PC = Rs	0100 0xxs ssxx xxxx
JSR <Label>	R7 = PC + 1; PC = (PC & 0x8000) | ((IMM11 offset to <Label>) << 4)	0100 1iii iiii iiii
JMPR Rs	PC = Rs	1100 0xxs ssxx xxxx
JMP <Label>	PC = PC + 1 + (sext(IMM11) offset to <Label>)	1100 1iii iiii iiii
TRAP UIMM8	R7 = PC + 1; PC = (0x8000 | UIMM8); PSR [15] = 1	1111 xxxx uuuu uuuu
RTI	PC = R7; PSR [15] = 0	1000 xxxx xxxx xxxx
Pseudo-Instructions		
RET	Return to R7	JMPR R7
LEA Rd <Label>	Store address of <Label> in Rd	CONST/HICONST
LC Rd <Label>	Store value of constant <Label> in Rd	CONST/HICONST
Assembler Directives		
.CODE	Current memory section contains instruction code	
.DATA	Current memory section contains data values	
.ADDR UIMM16	Set current memory address value to UIMM16	
.FALIGN	Pad current memory address to next multiple of 16	
.FILL IMM16	Current memory address's value = IMM16	
.STRINGZ "String"	Expands to a .FILL for each character in "String"	
.BLKW UIMM16	Reserve *UIMM16* words of memory from the current address	
<Label> .CONST IMM16	Associate *<Label>* with IMM16	
<Label> .UCONST UIMM16	Associate *<Label>* with UIMM16	

0101: opcode or sub-opcode ddd: destination register sss: source register 1 ttt: source register 2
iii: signed immediate value uuu: unsigned immediate value xxx: "don't care" value

[1] In this case the source and destination register are one and the same as HICONST reads and modifies the same register.
[2] sign(Rs- Rt) results in one of three values: +1, 0, or -1, which set the appropriate bit in the NZP register.
[3] sign(uRs- uRt) indicates that Rs and Rt are treated as unsigned values.
[4] The NZP register is updated on any instruction that writes to a register, and on CMPx instructions.

Single Cycle Implementation of the LC4 ISA



Description of Control Signals in Single Cycle Implementation of the LC4 ISA

Signal Name	# of bits	Value	Action
PCMux.CTL	3	0	Value of NZP register compared to bits I[11:9] of the current instruction if the test is satisfied then the output of TEST is 1 and NextPC = BRANCH Target, (PC+1) + SEXT(IMM9); otherwise the output of TEST is 0 and NextPC = PC + 1
		1	Next PC = PC+1
		2	Next PC = (PC+1) + SEXT(IMM11)
		3	Next PC = RS
		4	Next PC = (0x8000 | UIMM8)
		5	Next PC = (PC & 0x8000) | (IMM11 << 4)
rsMux.CTL	2	0	rs.addr = I[8:6]
		1	rs.addr = 0x07
		2	rs.addr = I[11:9]
rtMux.CTL	1	0	rt.addr = I[2:0]
		1	rt.addr = I[11:9]
rdMux.CTL	1	0	rd.addr = I[11:9]
		1	rd.addr = 0x07
regFile.WE	1	0	Register file not written
		1	Register file written: rd.addr indicates which register is updated with the value on the Write Input
regInput.Mux.CTL	2	0	Write Input = ALU output
		1	Write Input = Output of Data Memory
		2	Write Input = PC + 1
NZP.WE	1	0	NZP register not updated
		1	NZP register updated from Write Input to register file
DATA.WE	1	0	Data Memory not written
		1	Data Input written into location on Data Address lines
Privilege.CTL	2	0	PSR[15] = 0 – Clear privilege bit
		1	PSR[15] = 1 – Set privilege bit
		2	PSR[15] unchanged – no change to privilege bit
ALUInputMux.CTL	1	0	B[15:0] = RT[15:0] – B input to ALU = RT
		1	B[15:0] = I[15:0] – B input to ALU = Instruction Word

Signal Name	# of bits	Value	Action
ALU.CTL	6		
Arithmetic Ops		0	C = A + B : Addition
		1	C = A * B : Multiplication
		2	C = A - B : Subtraction
		3	C = A / B : Division
		4	C = A % B : Modulus
		5	C = A + SEXT(B[4:0])
		6	C = A + SEXT(B[5:0])
Logical Ops		8	C = A AND B : Bitwise Logical Product
		9	C = NOT A: Bitwise Negation
		10	C = A OR B: Bitwise Logical Sum
		11	C = A XOR B: Bitwise Exclusive OR
		12	C = A AND SEXT(B[4:0])
Comparator Ops		16	C = signed-CC(A-B) [-1, 0, +1]
		17	C = unsigned-CC(A-B) [-1, 0, +1]
		18	C = signed-CC(A-SEXT(B[6:0])) [-1, 0, +1]
		19	C = unsigned-CC(A-SEXT(B[6:0])) [-1, 0, +1]
Shifter Ops		24	C = A << B[3:0] : Shift Left Logical
		25	C = A >>> B[3:0] : Shift Right Arithmetic
		26	C = A >> B[3:0] : Shift Right Logical
Constant Ops		32	C = SEXT(B[8:0])
		33	C = (A & OxFF) | (B[7:0] << 8)