ECE 120: Introduction to Computing

Spring 2019 ZJUI

NOT DUE, AND NOT GRADED

Homework 14: LC-3 Control Signals and Error Control Coding

In this homework, you must implement control signals for an LC-3 processor. For problems 1-4, you should consider only the abbreviated list given in Section 4.1.1 (p. 133) of the notes. The full set in Appendix C of Patt and Patel includes several aspects of LC-3 functionality that we do not discuss in our class. For examples of mapping RTL to these control signals, see Sections 4.1.2 and 4.1.3.

1. **Control Signals for ST**
   Write the full set of control signals necessary to implement an ST instruction from fetch through execution. Use the state numbers in Patt and Patel's FSM diagram to identify each state (for example, the first fetch state is #18, and the decode state is #32).

#	LD. MAR	LD. MDR	LD. IR	LD. BEN	LD. REG	LD. CC	LD. PC	Gate PC	Gate MDR	Gate ALU	Gate MARMUX	PCMUX	DRMUX	SR1MUX	ADDR1MUX	ADDR2MUX	MARMUX	ALUK	MIO.EN	R.W
18	1	0	0	0	0	0	1	1	0	0	0	00	xx	Xx	x	xx	x	xx	0	x
33	0	1	0	0	0	0	0	0	0	0	0	xx	xx	Xx	x	xx	x	xx	1	0
35	0	0	1	0	0	0	0	0	1	0	0	xx	xx	Xx	x	xx	x	xx	0	x
32	0	0	0	1	0	0	0	0	0	0	0	xx	xx	Xx	x	xx	x	xx	0	x
3	1	0	0	0	0	0	0	0	0	0	1	xx	xx	Xx	0	10	1	xx	0	x
23	0	1	0	0	0	0	0	0	0	1	0	xx	xx	00	x	xx	x	11	0	x
16	0	0	0	0	0	0	0	0	0	0	0	xx	xx	Xx	x	xx	x	xx	1	1

Note that there are two options for state 23. The signals shown use the ALU, but one can also use the address generation adder to add 0 to SR1 and pass it onto the bus through the MARMUX. Either version is acceptable.

2. **Control Signals for BR**
   Write the full set of control signals necessary to implement the execution (not fetch nor decode) of a BR instruction. Again, use the state numbers to identify the states.

#	LD. MAR	LD. MDR	LD. IR	LD. BEN	LD. REG	LD. CC	LD. PC	Gate PC	Gate MDR	Gate ALU	Gate MARMUX	PCMUX	DRMUX	SR1MUX	ADDR1MUX	ADDR2MUX	MARMUX	ALUK	MIO.EN	R.W
0	0	0	0	0	0	0	0	0	0	0	0	xx	xx	Xx	x	xx	x	xx	0	x
22	0	0	0	0	0	0	1	0	0	0	0	10	xx	Xx	0	10	x	xx	0	x

State 22 again offers two choices: route the sum from the address generation adder directly into the PC or over the bus into the PC. Again, either approach is acceptable (the first is shown).

3. Adding a New Instruction

Prof. Lumetta is annoyed with having to load values into registers whenever he needs to use an immediate operand of more than 5 bits. Help Prof. Lumetta by adding a new instruction that allows a 16-bit immediate field to the LC-3. In particular, your new instruction must require two memory locations, and the second memory location will contain the 16-bit immediate value. Your implementation should load the immediate value into R7, advance the PC again, and use the existing ADD and AND execution states to perform the operation as though the instruction is using a second register operand. Do not worry about overwriting R7.

a. Draw your instruction format in the style used in the class handouts (and the textbook). The new instruction should use opcode 1101 to reach your sequence of states. The DR and SR1 fields from AND and ADD are, of course, still needed. You also need one bit—IR[3]—to select between ADD (0) and AND (1). You may assume that the LC-3 ignores IR[3] (as well as IR[15:12]!) when executing ADD and AND. However, you must configure the remaining 5 bits of your new instruction format to make these states operate correctly with the 16-bit immediate value.

15 … 12	11 … 9	8 … 6	5 … 4	3	2 … 0
1101	DR	SR1	00	op	111

op = 0 for ADD, 1 for AND

b. Draw the sequence of FSM states needed to implement your new instruction. The first state should be state #13 (the DECODE state sends your any instruction with opcode 1101 to that state). Number the other states in your diagram starting at 90 (these are not valid state numbers, but are needed for **part (c)** below), but reuse the ADD and AND FSM execution states—numbers 1 and 5, respectively—to make use of the ALU with your new instruction (do not reimplement these two states). Be sure to label which bits are used to make transition decisions if your states have more than one next state.

state 13: MAR ← PC, PC ← PC + 1 — next state: 90
state 90: MDR ← M[MAR] — next state: R': 90, R: 91
state 91: R7 ← MDR — next state: IR[3]': 1, IR[3]: 5

c. For each state in your new sequence, write the 25 control signals necessary to implement that state. Use the state numbers to identify the states.

#	LD. MAR	LD. MDR	LD. IR	LD. BEN	LD. REG	LD. CC	LD. PC	Gate PC	Gate MDR	Gate ALU	Gate MAR MUX	PC MUX	DR MUX	SR1 MUX	ADDR1 MUX	ADDR2 MUX	MAR MUX	ALUK	MIO. EN	R.W
13	1	0	0	0	0	0	1	1	0	0	0	00	xx	Xx	x	xx	x	xx	0	x
90	0	1	0	0	0	0	0	0	0	0	0	xx	xx	Xx	x	xx	x	xx	1	0
91	0	0	0	0	1	0	0	0	1	0	0	xx	01	Xx	x	xx	X	xx	0	x

4. Microsequencer Bits

For each of the FSM states in problems 1 and 2 (complete processing of an ST instruction, and execution of a BR instruction), write the sequencing control bits J, COND, and IRD for the Patt and Patel microsequencer, as described in Appendix C.4.

#	J	COND	IRD
18	100001	101	0
33	100001	001	0
35	100000	000	0
32	xxxxxxx	xxx	1
3	011101	000	0
23	010000	000	0
16	010000	001	0
0	010010	010	0
22	010010	000	0

5. Pair Codes?

A friend of yours is quite excited about having developed a new class of codes: pair codes. Pair codes are a generalization of the 2-out-of-5 representation discussed in Section 4.2.1 of the notes. In a pair code, each code word has exactly two 1 bits. However, one can define a pair code on any number of bits N. For example, if N=100, one has 100 bits in the code words, and exactly two 1 bits.

Your friend points out that as N grows, the fraction of valid code words drops dramatically. For N=100, for example, there are only 4950 valid code words (100 times 99 divided by 2), but there are 2^{100} bit patterns. Your friend argues that error correction capabilities for pair codes must be quite powerful, since the codes are so sparse.

a. What is the Hamming distance of the pair code with 6-bit code words? Use an example to prove that your answer is correct.
   The Hamming distance of any pair code is 2. For 6-bit code words, for example, the code words 000011 and 000101 are distance 2 apart.
b. What is the Hamming distance of the pair code on 100-bit code words? Explain how you can again prove that your answer is correct (please avoid writing 100-bit numbers).
   As stated for **part (a)**, the Hamming distance for any pair code is 2. To show that it cannot be greater, consider the example given in **part (a)** and add 0s to the front. The results are still code words (of any size, including 100-bit), but are still distance 2 apart. Any code words that are not identical are separated by at least Hamming distance 2, however, since one word has a 1 in place of a 0 in the other, and vice-versa. So the code overall has Hamming distance 2.
c. How many bits can be corrected using a pair code with N-bit code words?
   None.

6. Hamming Codes

The notes provided details on 7-bit Hamming codes. Consider the use of Hamming codes on more bits.

a. If a Hamming code is used with 10 bits, how many of the bits are parity check bits?
Four. Bits 1, 2, 4, and 8.

b. If a Hamming code is used with 100 bits, how many of the bits are parity check bits?
Seven. Bits 1, 2, 4, 8, 16, 32, and 64.

c. Why does it make little sense to use a Hamming code with 128 bits? Explain your answer.
Bit 128 is a parity bit and covers only itself, so the number of data bits as well as the error detection/correction properties are the same as with the 127-bit Hamming code.

7. SEC-DED Codes

A communication system uses a SEC-DED code to protect transmissions. The code is based on a 7-bit Hamming code extended with an odd parity bit (the first bit). Using the notation from the notes, each received word consists of the parity bit followed by the 7-bit Hamming code word $x_7x_6x_5x_4x_3x_2x_1$. For each of the following received words, extract the 4-bit data word when possible, or mention that the received word had uncorrectable errors.

a. 00101101
Parity is wrong, but error syndrome is 000, so parity bit was flipped.
Data is 0101.

b. 10011110
Parity is correct, and error syndrome is 000, so no error.
Data is 0011.

c. 10101001
Parity is wrong, and error syndrome is 011, so bit 3 was flipped.
After correction, data is 0101.

d. 10010001
Parity is correct, but error syndrome is 100, so more than one bit flip—uncorrectable!

e. 00111001
Parity is wrong, and error syndrome is 110, so bit 6 was flipped.
After correction, data is 0010.