CSCI 463-1

Computer Systems Organization

Spring 2023

Homework #6

# I. Hardwired control

The essence of hardwired control is the signal sequence for each instruction. The slides give you an example of developing the signal sequence for MARIE’s ADD instruction on slides 4f‑56 through 57b.

Using ADD as an example and the simplified RTL chart from last week’s assignment to get the data, write signal sequences for the following instructions. Use the signals in the slides; the signals in the book differ from edition to edition.

Don’t forget the reset counter step at the end of each instruction. There is no need to use signal A3, i.e., if no accumulator operation is needed, just don’t use any A signal.

(Note: write the signals as P0, etc., i.e., don’t use subscripts. Subscripts would be a pain for you to enter and for your T.A. to read.)

1. SUBT

2. LOAD

3. STORE

# II. Microprogrammed control

As shown on slide 4g-64b, the table on slide 4g‑64 is not completely accurate. Since X is identical to IR[11:0], it contains a pair of duplicates. To eliminate this problem, always use 01010 and do not use 00111. Similarly, these entries have been added:

11000 MBR <- PC

11001 MBR <- X

11010 AC <- 1

11011 PC <- AC

11100 AC <- MBR

4. Study the MARIE microprogram on slide 4g-65. First, compare it to the hardwired execution cycle on slide 4c‑39. Then answer the following questions:

a) Which line(s) of the microprogram correspond to the fetch step of the fetch-decode-execute cycle? Use their addresses to answer this question.

Note: For binary numbers larger than 4 digits, such as these addresses, leave a space between groups of 4 counting from the right, as in the example below. This is similar to adding commas to decimal numbers in groups of 3, again counting from the right.)

b) Which line(s) of the microprogram correspond to the decode step?

5. Since we have now completed the common part of the microcode execution cycle, the fetch operand and execute steps must be encoded separately for each instruction in the jump table. This makes sense since every instruction does something different (i.e., has different execution instructions) and could also have a different number of operands to decode (although on Marie, the range is only from 0 to 1).

We’ll see how the jump table works in the next problem. In the meantime, for this problem, convert the microprogram to binary, using the format on slide 63 and the table on slide 64, with the corrections noted on slide 64b. (Don’t use the table from the book; it should be the same but it isn’t.) The first line is filled in for you as an example.

6. Now let’s look at the jump table. Study the second line of the the jump table, i.e., the line at address 000 0101. Which instruction does it reference? It is comparing the opcode to MicroOp2[4-1], which has the value 0001. From your RTL chart, you can see that this is the jump table entry for LOAD.

a) Look at the first line of the jump table, i.e., the line at address 000 0100. What opcode (in binary) is the incoming opcode being compared to in this line of microcode?

b) Which RTL instruction does that opcode correspond to?

c) If the incoming opcode matches that RTL instruction, what address will the microprogram jump to?

7. Let’s go back to the microcode for LOAD, i.e., the microcode that the jump table entry for LOAD points to. From the jump table, you can see that the microcode for LOAD will start at address 010 0111. Let’s fill in the microcode for LOAD.

a) The addresses increase sequentially from the starting address.

b) We can get the RTL from the RTL chart.

c) Since our simplified RTL for MARIE does not contain any simultaneous activity (i.e., it does not put two RTL instructions on one line), the second microoperation will always be NOP.

d) The instructions are executed sequentially except that when the last one is complete, the system must branch back to the fetch logic to fetch the next instruction. Therefore every line has a JUMP entry of 0 except the last line, which has a JUMP entry of 1 to indicate that it contains a branch.

e) If the JUMP entry is 0, the destination is irrelevant. When the JUMP entry is 1, the DEST entry must contain the destination of the JUMP. In this case we are branching back to the fetch logic, whose address is 000 0000.

So here is our almost-complete microcode for LOAD.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 010 0111 | MAR <- X | NOP | 0 | 000 0000 |
| 010 1000 | MBR <- M[MAR] | NOP | 0 | 000 0000 |
| 010 1001 | AC <- MBR | NOP | 1 | 000 0000 |

The last step is to look up the binary equivalents for the RTL instructions:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 010 0111 | 01010 | 00000 | 0 | 000 0000 |
| 010 1000 | 01100 | 00000 | 0 | 000 0000 |
| 010 1001 | 11100 | 00000 | 1 | 000 0000 |

Your job is to write the microcode for the first line of the jump table, i.e., the line at address 000 0100.

Hint: As in the example we just did, your first step should be to look up the RTL for the instruction you identified in the previous problem.