CSCI 463 Assignment 2 – Simple Microcode Machine

20 Points – Due Tuesday, October 15, 2019 at 23:59

Abstract

In this assignment, you will use the machine simulator demonstrated in lecture to design, write and debug a microcode program that will store the eight ASCII letters: A B C D E F G H into the memory starting at address `0x80`.

1 Input

The simulator reads two files:

1. Microdode expressed as one 16-bit binary value per line.
2. Up to 256 bytes expressed in hex that will be loaded into the system memory before the microcode starts executing.

Your microcode must generate and store the character data into the memory (in other words, you can not put the solution in your initial memory dump.)

Microcode for this machine matches that discussed in class and appearing on page four of the handout named "*cpu8c-schematic (microcode)*" whose microcode is described in "*microcode for cpu8c (version 2)*" reproduced, in part, below:

alu_func[1:0]	alu_comp_b	alu_ci	reg_we	mbr_alu	mar_we	mem_we	mbr_out_we	mbr_in_we	reg_addr_we[1:0]	alu_reg_a[1:0]	alu_reg_b[1:0]	operation	note
00	0	0	0	0	0	0	0	0	00	00	00	`alu_q` ← `r0 + r0`	fully specified addition
00	0	0	0	0	0	0	0	0	dd	aa	bb	`alu_q` ← `raa + rbb`	`aa` and `bb` are parameterized
00	0	0	0	0	0	0	0	0	xx	aa	bb	`alu_q` ← `raa + rbb`	`reg_addr_we[1:0]` doesn't matter
01	0	0	0	0	0	0	0	0	xx	aa	bb	`alu_q` ← `raa` ⊕ `rbb`	XOR
10	0	0	0	0	0	0	0	0	xx	aa	bb	`alu_q` ← `raa` ∧ `rbb`	AND

The headings in the above table match the columns and number of bits in each field of the microcode input file you must create.

As seen below, your microcode can have comments. Anything after the pound-sign '#' is ignored as are blank lines and lines that only have comments in them.

An Example Microcode Input File

```
00 0 1 0 0 0 0 0 0 00 00 00  # alu_q = 1, reg_in = 1
00 0 1 0 0 0 0 0 0 11 00 00  # alu_q = 1, reg_in = 1, route reg_we to R3
00 0 1 1 0 0 0 0 0 11 00 00  # alu_q = 1, reg_in = 1, clock R3
00 0 1 0 0 0 0 0 0 11 00 00  # alu_q = 1, reg_in = 1, falling edge on R3 reg_we
```

Your initial memory dump MUST NOT have the ASCII result character values in it. If you do not need an initial memory image dump, you may omit the `prog2.dump` file from your solution.

A memory image dump file is formatted as a set of whitespace-separated hex byte values as many or as few per line as you prefer as shown below. The byte values will be placed into memory starting at address zero in the order appearing in your input file.

An Example 32-byte Memory Dump Input File

```
0F 1F 2F 3F 4F 5F 6F 7F 8F 9F AF BF CF DF EF FF
f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 fa fb fc fd fe ff
01 02 03 04
05 06 07 08
09 0a 0b 0c
```

2 Output

The simulator will reset the CPU, load your input memory dump, print the initial memory contents then proceed to print your microcode instructions as they are executed and after the microcode instructions have all completed it will print the memory contents again.

Your program must complete with the proper values stored in memory. Any values in any memory location are acceptable **EXCEPT** that your program **MUST** leave the proper ASCII values in bytes 0x80-0x87 as shown here:

Example correct ASCII output

```
winans@hopper:~$ ~winans/csci463/a2/ucsim prog2.uc
0000: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0050: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0070: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0080: 41 42 43 44 45 46 47 48  00 00 00 00 00 00 00 00   <--- this row counts!
0090: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

3 File You Must Write

You will create and submit two files for this assignment.

Create a directory named `prog2` and place your source files for this assignment within. Implement your solution for this assignment in two files named `prog2.uc` and `prog2.dump`.

When we grade your assignment, we will compile and run it on `hopper.cs.niu.edu` using one of these commands (depending on the presence of a `prog2.dump` file):

- `~winans/csci463/a2/ucsim prog2.uc`
- `~winans/csci463/a2/ucsim prog2.uc prog2.dump`

4 How To Hand In Your Program

When you are ready to turn in your assignment, make sure that the only files in your `prog2` directory are the `prog2.uc` and, optionally, `prog2.dump` files discussed above. Then, in the parent of your `prog2` directory, use the `mailprog.463` command to send the contents of the files in your `prog2` project directory to your TA like this:

```
mailprog.463 prog2
```

If `mailprog.463` detects and problems, it will inform you that you have not followed the instructions given above and provide some hints how to proceed. If you followed these instructions you will see the following:

```
winans@hopper:~/$ mailprog.463 prog2

*******************************************************************
* WARNING : Do NOT use this program to mail notes to your Instructor *
*           Doing so may result in the loss of your program !!       *
*******************************************************************

Enter program number for your assignment : 2
shar: Saving /tmp/mailprog.11111 (text)
winans@hopper:~/$
```

5 Grading

The grade you receive on this programming assignment will scored according to the syllabus and its ability to execute on the CSCI Department's computer.

It is your responsibility to test your program thoroughly.

6 Hints

- Add a -v1 flag to the simulator to dump the state of the registers and signals withing the CPU after each microcode instruction. Note that the items printed with an asterisk '*' in first column are the fields provided by the microde instruction.
- Add a -v2 flag to dump the entire memory as well as the signal and register states.

For example:

Example simulator run as seen in class

```
winans@hopper:~$ ~winans/csci463/a2/ucsim prog2.uc
Memory before execution begins:
0000: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0050: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0070: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0080: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0090: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00

insn: 1000 '00 0 1 0 0 0 0 0 0 00 00 00  # alu_q = 1, reg_in = 1'
insn: 1030 '00 0 1 0 0 0 0 0 0 11 00 00  # alu_q = 1, reg_in = 1, route reg_we to R3'
insn: 1830 '00 0 1 1 0 0 0 0 0 11 00 00  # alu_q = 1, reg_in = 1, clock R3'
insn: 1030 '00 0 1 0 0 0 0 0 0 11 00 00  # alu_q = 1, reg_in = 1, falling edge on R3 reg_we'

Memory after execution ends:
0000: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0050: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0070: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0080: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0090: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

Example simulator run with v1 debugging enabled

```
winans@hopper:~$ ~winans/csci463/a2/ucsim -v1 prog2.uc

...

insn: 1000 '00 0 1 0 0 0 0 0 0 00 00 00  # alu_q = 1, reg_in = 1'
*    alu_func = 0x0
*  alu_comp_b = 0x0
*      alu_ci = 0x1
*      reg_we = 0x0
*     mbr_alu = 0x0
*      mar_we = 0x0
*      mem_we = 0x0
*  mbr_out_we = 0x0
*   mbr_in_we = 0x0
* reg_addr_we = 0x0
*   alu_reg_a = 0x0
*   alu_reg_b = 0x0
       alu_so = 0x0
       alu_uo = 0x0
        alu_a = 0x00
        alu_b = 0x00
        alu_q = 0x01
       reg_in = 0x01
       mbr_in = 0x00
        d_out = 0x00
         d_in = 0x00
     addr_out = 0x00
       reg(0) = 0x00
       reg(1) = 0x00
       reg(2) = 0x00
       reg(3) = 0x00

...
```

Example simulator run with v2 debugging enabled

```
winans@hopper:~$ ~winans/csci463/a2/ucsim -v2 prog2.uc

...

insn: 1000 '00 0 1 0 0 0 0 0 0 00 00 00  # alu_q = 1, reg_in = 1'
*     alu_func = 0x0
*   alu_comp_b = 0x0
*       alu_ci = 0x1
*       reg_we = 0x0
*      mbr_alu = 0x0
*       mar_we = 0x0
*       mem_we = 0x0
*   mbr_out_we = 0x0
*    mbr_in_we = 0x0
*  reg_addr_we = 0x0
*    alu_reg_a = 0x0
*    alu_reg_b = 0x0
       alu_so = 0x0
       alu_uo = 0x0
        alu_a = 0x00
        alu_b = 0x00
        alu_q = 0x01
       reg_in = 0x01
       mbr_in = 0x00
        d_out = 0x00
         d_in = 0x00
     addr_out = 0x00
       reg(0) = 0x00
       reg(1) = 0x00
       reg(2) = 0x00
       reg(3) = 0x00
0000: 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00
0010: 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00
0050: 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00
0070: 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00
0080: 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00
0090: 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00
00A0: 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00
00B0: 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00
00C0: 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00
00D0: 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00
00E0: 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00
00F0: 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 00

...
```