Homework 7 Solutions

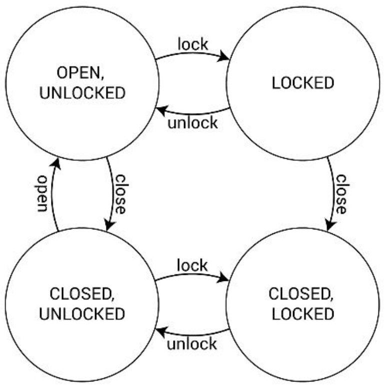
1. List the major components of a state machine and give descriptions for each.

   (3)

   The states: What possible states are there? These should be exclusive (it cannot be in two states at once) and exhaustive (it must always be in one of these states).
   The inputs: What possible actions can be performed in the system?
   The transitions: How does each action affect the state the system is in?

2. For the state machine in the figure below, mention the states transitioned to when the following sequence of inputs is applied every clock. If no option for the state with the input is listed, then the machine stays in that state. The initial state is (CLOSED, UNLOCKED).
   lock, open, unlock, open, lock, lock. The first couple states are given.

   (4)



Input	Current State	Next State
lock	Closed, Unlocked	Closed, Locked
open	Closed, Locked	**Closed, Locked**
unlock	**Closed, Locked**	**Closed, Unlocked**
open	**Closed, Unlocked**	**Open, Unlocked**
lock	**Open, Unlocked**	**Locked**
lock	**Locked**	**Locked**

3. Why are operations that have data dependencies not allowed in the same state? **(2)**

   Dependent operations have data dependencies. Since each state operates on those values, either by preparing them or doing some operation, and the operations are dependent, then the preceding operation needs to finish its values so that the dependent operation can do the next state.

4. What are Auxiliary registers? Can a programmer access them? **(2)**

   They store intermediate values in between stages. No, a programmer cannot access them, they are just used to hold values while calculation happens.

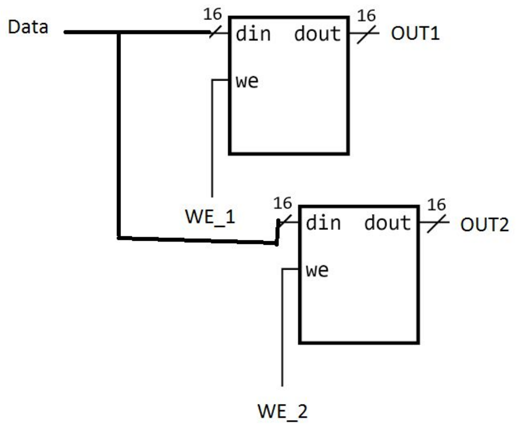
   Alternative definition from book: auxiliary registers are named memory slots where any given stage can store values and a later stage can read those values out.

5. What are the five stages of a state machine for a computer's microarchitecture? **(5)**

The "fetch" stage: These are the states that retrieve the next instruction we are supposed to execute.
The "decode" stage: These states will figure out what the instruction does--e.g. for ldi, which value it wants to store and which register it wants to store it in.
The "execute" stage: These states will perform any arithmetic requested by the operation. For example, ldi will not have any states in this stage.
The "memory" stage: These states will perform any RAM-related operations the the instruction requires. ldi will not have any states in this stage either.
The "writeback" stage: These states will perform any updates to registers. ldi will go through a state here that writes the value to the desired register.

6. In the following circuit, we have two auxiliary registers wired up as shown. We have three inputs, WE_1, WE_2, and Data. We have two outputs, OUT1 and OUT2. The table below has the inputs filled in over a certain period of time. Fill in the outputs. Notice the inputs happen sequentially from left to right on the table. Assume that OUT1 and OUT2 are initially 0.

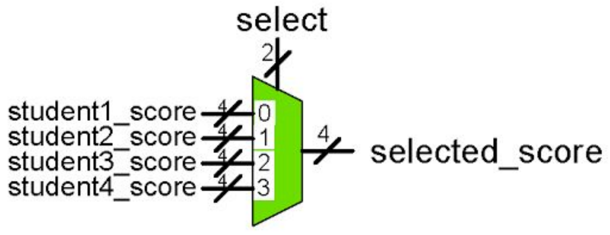
(5)



WE_1	**0**	**1**	**0**	**0**	**1**
WE_2	**0**	**1**	**1**	**0**	**0**
Data	**0**	**1**	**0**	**1**	**0**
OUT1	0	1	1	1	0
OUT2	0	1	0	0	0

7. You are given scores (out of 10) for 4 students. Draw a circuit which can select the score of one among these 4 students, as an output. Clearly mention the bit-width of each bus. **(2)**

The answer is a MUX. Inputs are 4 bits each to represent 10, and then the select line is 2 bits, to select between 4 values.



8. You want to create a circuit using MUXes, ALU, and wires which has the following definition:
Inputs: **(6)**
$A(A_4A_3A_2A_1)$: 4-bit input
$B(B_4B_3B_2B_1)$: 4-bit input
SAL: Select ALU operation: 2 bit input

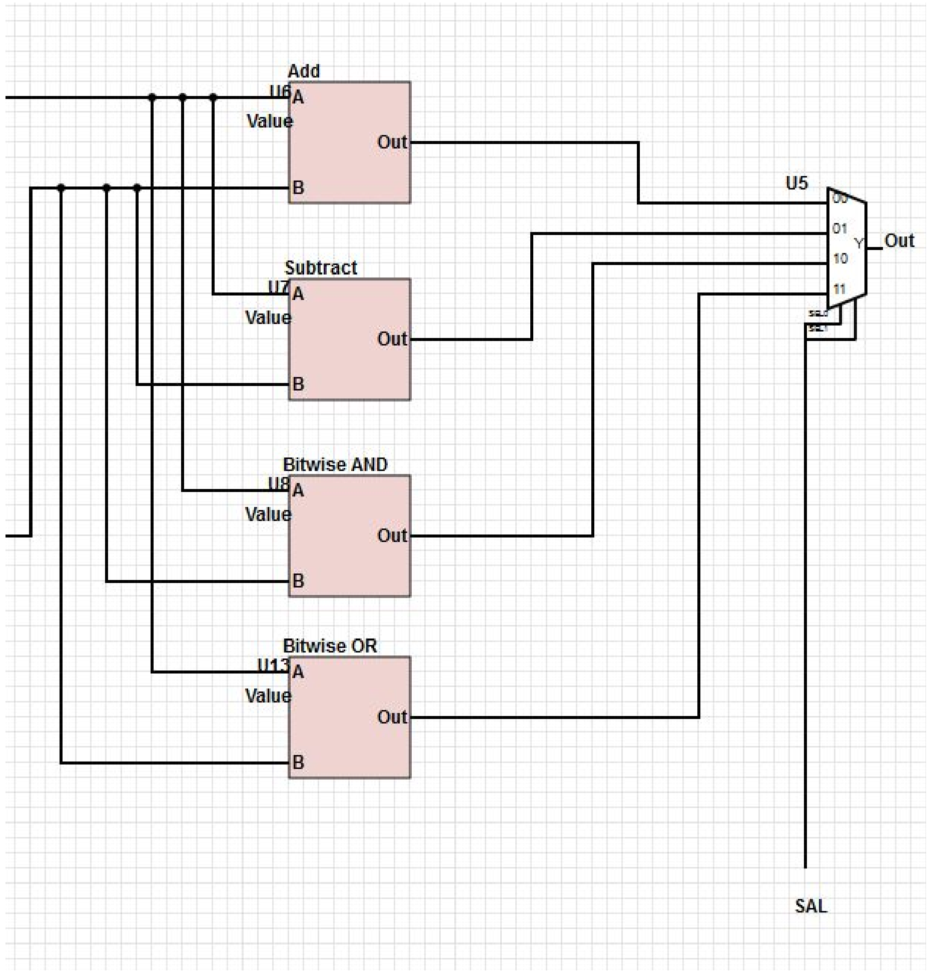
Output: $R(A_4A_3A_2A_1)$: 4-bit output

The operation to be performed by the ALU is selected according to the following table:

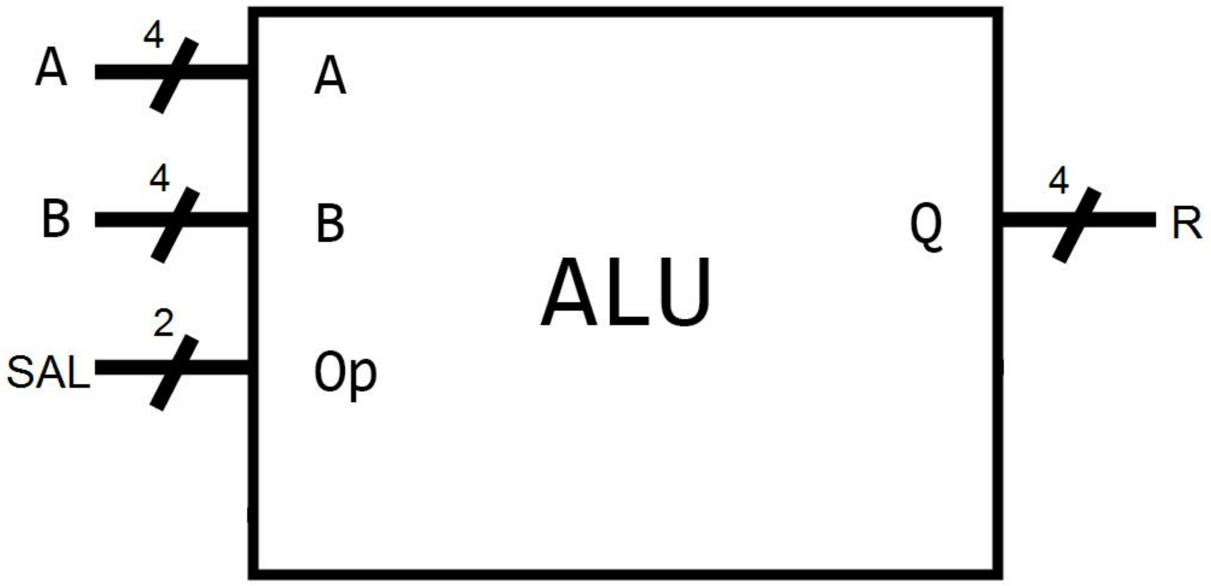
SAL	**Operation**
00	Add
01	Subtract
10	bitwise AND
11	bitwise OR

Draw the circuit diagram for this.

A few ways to do this.
Solution 1:

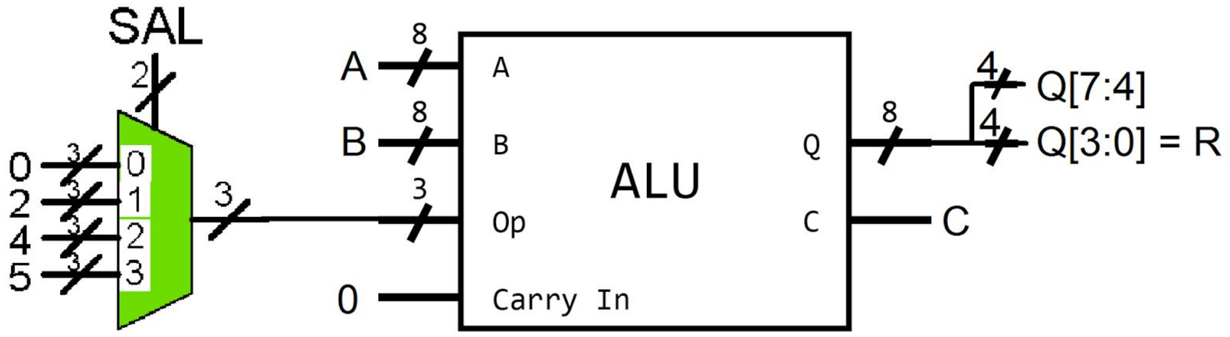


Alternative solution 2: Start with ALU from Figure 7.2.2.6 (from book). Change the ALU so that it does the operations described above. The MUX is inside the ALU, and not shown in the figure below.

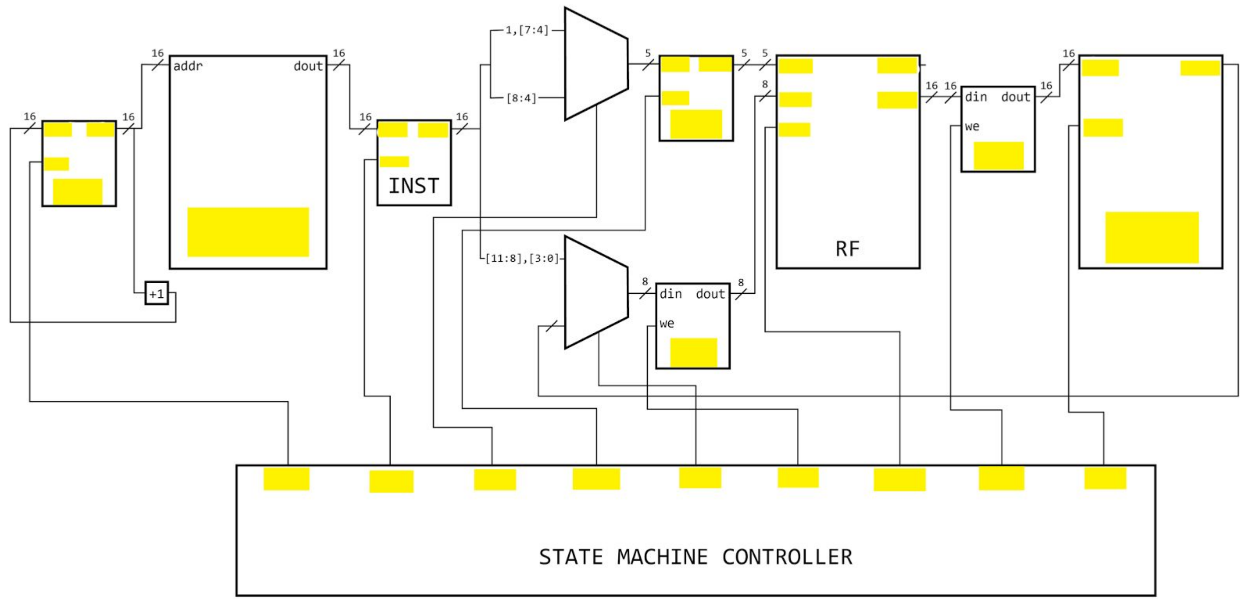


Alternative solution 3: Use the table and ALU from Figure 7.2.2.6. Use a mux to feed the Op to match the operations described above.

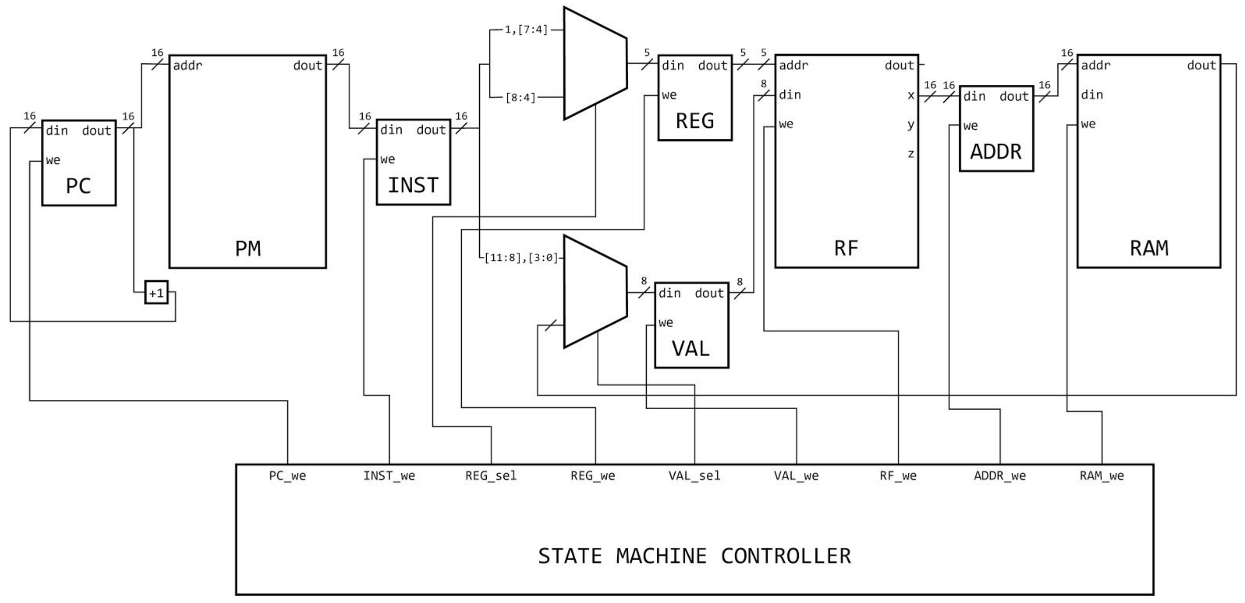
Number	Operation
0	Addition
1	Addition with carry
2	Subtraction
3	Subtraction with carry
4	Binary and
5	Binary or
6	Binary xor
7	Binary nor



9. Below is a diagram for a circuit that can implement the LD and LDI instruction. Write in all the missing labels for ports and components. The missing labels are highlighted. **(2)**



Answer from book:



10. In the table at the end of the chapter, several blocks are vacant (without a 0 or a 1). Why is that? **(2)**

    These are select lines where we are not writing the corresponding register they are inputting to. In these situations, we don't care about what the select line is set to, as it isn't going to affect the auxiliary register.

11. Trace through the instructions below and give the list of states that the program goes through until completion by filling out the following table. The first few states for the first instruction are given to you.To determine the state numbers, reference the state diagram given in the lecture 25 slides (http://pages.cs.wisc.edu/~karu/courses/cs252/fall2015//handouts/lecture/lec-notes-25.pdf, 2nd slide) **(5)**

```
ldi r16, 15
ldi r17, -15
add r17, r16
subi r17, 1
```

Cycle	**Current State**	**Instruction/Opcode**
0	**00000**	**ldi**
1	**00001**	**ldi**
2	**00101**	**ldi**
3	**10010**	**ldi**
4	10011	ldi
5	00000	ldi
6	00001	ldi
7	00101	ldi
8	10010	ldi
9	10011	ldi
10	00000	add

11	00010	add
12	00111	add
13	01010	add
14	01000	add
15	01011	add
16	01101	add
17	10010	add
18	10011	add
19	00000	subi
20	00001	subi
21	00111	subi
22	01111	subi
23	01100	subi
24	01101	subi
25	10010	subi
26	10011	subi

Since simulator changed, it is okay to start from either 0 or 1. It is also okay with either reset it after each instruction or continue counting up for the entire program.

12. Trace through the instructions below. For each instruction, walk through the states and give the name of the auxiliary register modified (if any) and its value, and give the control signals that change for that state. You do not need to write if a control signal goes low in the next cycle after it is used (for example, *we* signals). The program starts at PC=0 **(10)**

```
ldi r16, 15
ldi r17, -15
add r17, r16
breq 1
ld r26, X
subi r17, 1
```

Cycle	**State**	**INST**	**Changed aux registers/values and control signals (if any):**
0	00000	ldi	INST = 0xe00f INST_we = 1
1	00001	ldi	REG = 0x10 REG_we = 1 REG_sel = 0
2	00101	ldi	VAL = 0x0f VAL_we = 1 VAL_sel=3
3	10010	ldi	RF_sel = 0 RF_we = 1
4	10011	ldi	PC = 0x0001 PC_sel=1 PC_we=1
5	00000	ldi	INST = 0xef11 = 61201 INST_we = 1
6	00001	ldi	REG = 0x11 REG_we = 1 REG_se = 0
7	00101	ldi	VAL = 0xf1 VAL_we = 1 VAL_sel=3 ; also okay to write VAL=-15
8	10010	ldi	RF_sel = 0 RF_we = 1
9	10011	ldi	PC = 0x0002 PC_sel=1 PC_we=1
10	00000	add	INST=0x0f10 = 3856 INST_we=1
11	00010	add	REG=0x11 REG_sel=1 REG_we=1
12	00111	add	VAL1=0xf1 VAL1_we=1
13	01010	add	REG2=0x10 REG2_we=1

14	01000	add	VAL2=0x0f RF_sel=1 VAL2_sel=0 VAL2_we=1
15	01011	add	VAL=0x00 VAL_sel=1 VAL_we=1 ALU_op=0 A_sel=1 B_sel=0
16	01101	add	SREG Z=1 SREG_we=1
17	10010	add	RF_sel=0 RF_we=1
18	10011	add	PC = 0x03 PC_sel=1 PC_we=1
19	00000	breq	INST=0xf009 = 61449 INST_we=1
20	00011	breq	OFF=0x01 OFF_sel=1 OFF_we=1
21	01110	breq	VAL=0x04 VAL_sel=1 VAL_we=1 A_sel=0 B_sel=1 ALU_op=0
22	10100	breq	PC=0x04 PC_sel=0 PC_we=1
23	10011	breq	PC=0x05 PC_sel=1 PC_we=1
24	00000	subi	INST=0x5011 = 20497 INST_we=1
25	00001	subi	REG_sel=0 REG_we=1
26	00111	subi	VAL1_we=1
27	01111	subi	VAL2_we=1 VAL2_sel=1
28	01100	subi	VAL=0x01 VAL_sel=1 VAL_we=1 ALU_op=1 A_sel=1 B_sel=0
29	01101	subi	SREG=? SREG_we=1
30	10010	subi	RF_sel=0 RF_we=1
31	10011	subi	PC=0x06 PC_sel=1 PC_we=1

Since simulator changed, it is okay to start from either 0 or 1. It is also okay with either reset it after each instruction or continue counting up for the entire program.