Zachary Frederick
EEL 3701
Section 7287
Alpha: KN
Lab: 6

Pre-Lab Questions

Part 1

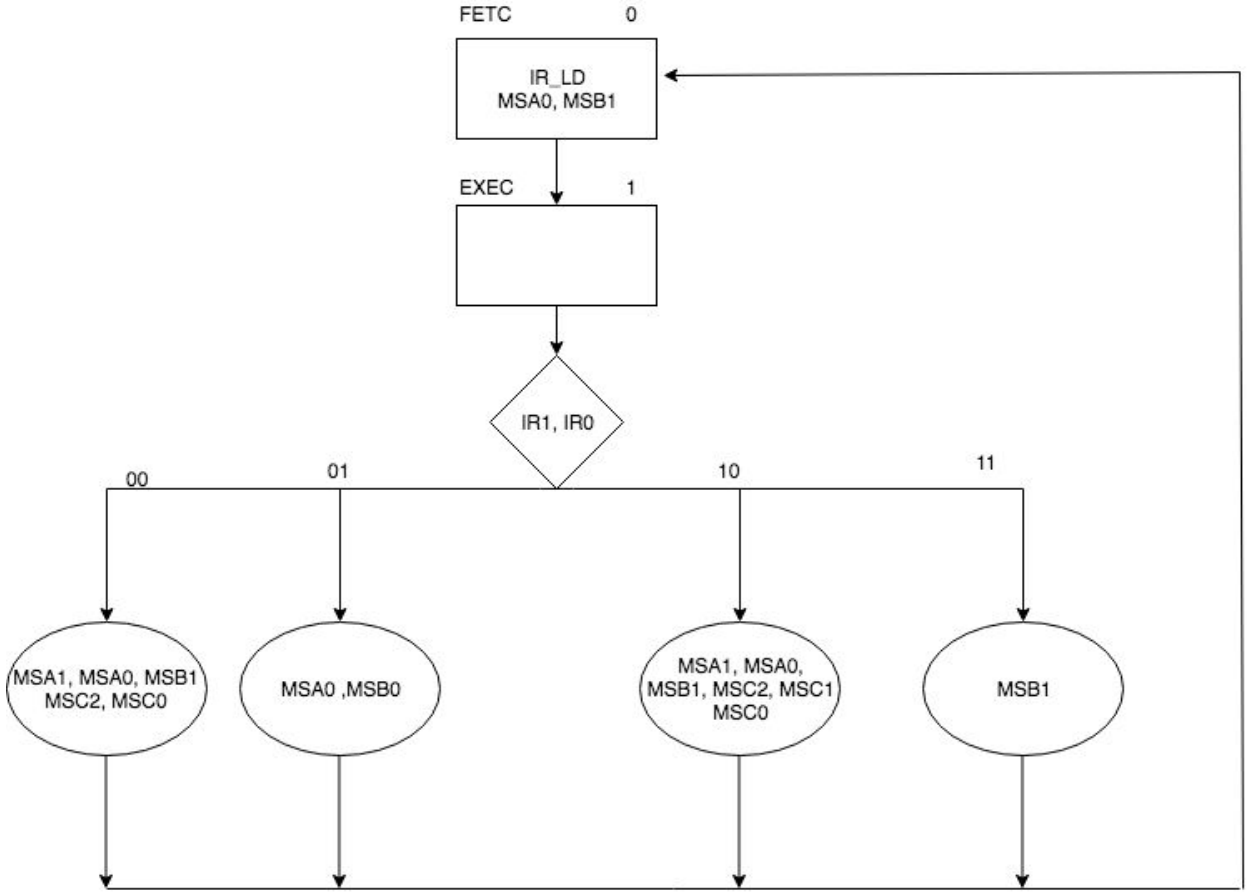
1. This design required an instruction register because we have two seperate states. We have a fetch state that loads the opcode and then a separate state that executes that particular opcode. Because both of these states happen in separate clock cycle we need a register to remember these values. Additionally, the IR allows us to have opcodes that have their own states. Such as with LDAA #data and JMP Addr in part2.
2. You could add a program counter that stores the current location in memory and then increments with each clock cycle. So each cycle the correct address from memory would be loaded.
3. To have more instructions we could simply us a bigger Instruction Register. More bits allows us to have more instructions. A 3 bit register would allow for 8 instructions.

Part 2

1. LDAA and JMP require separate states in part 2 because the our design allows us to only load 4 bits at a time from memory. If we had a bigger memory bus and PC we could load the data for LDAA and JMP in the same clock cycle.
2. In order to change the address lines to start at $5370 instead of $3AB0 we would need to change the configuration of the the first 11 bits. Currently the address bus has the first 11 bits set to 01110101011 through the use of Ground and VCC. However in order to change this to point to location $5370 we need to change the first 11 bits to be 010100110111.

Part 1 Pre-Lab Materials

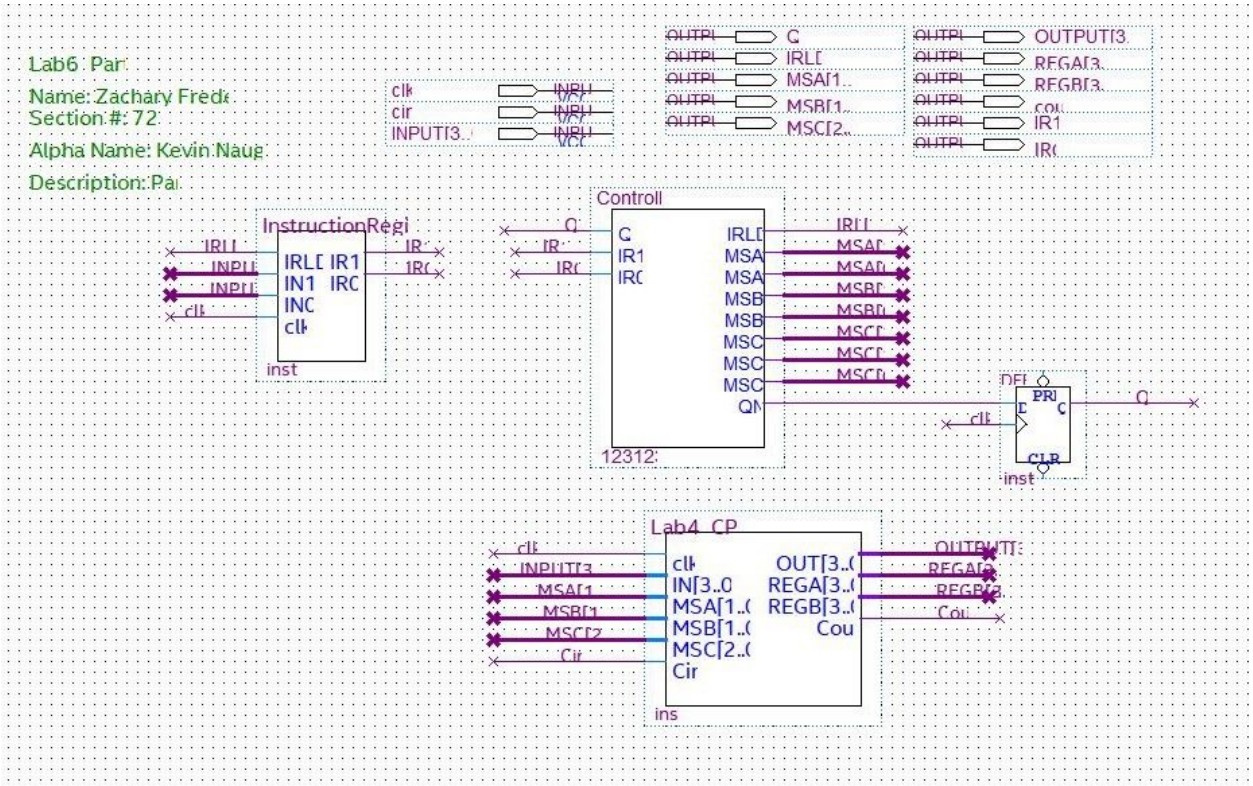
Part 1 asm



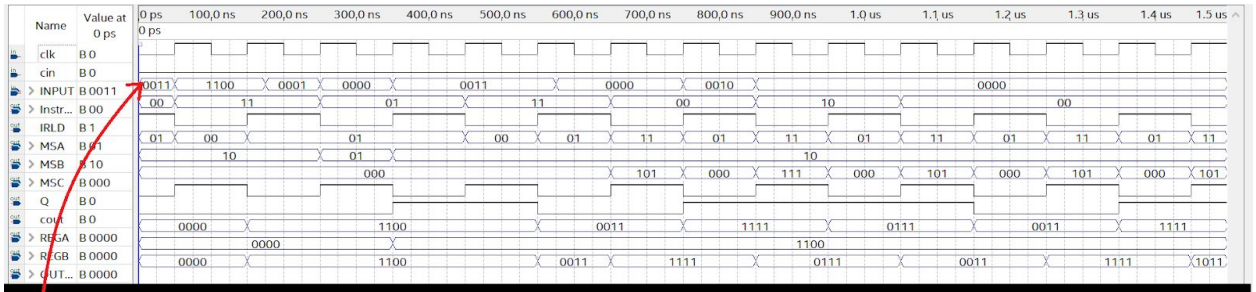
Part 1 Next State Truth Table

STATE	Q	IR1:0	IR.LD	MSA1:0	MSB1:0	MSC2:0	STATE+	Q+
FETCH	0	XX	1	01	10	000	EXEC	1
EXEC	1	00	0	11	10	101	FETCH	0
EXEC	1	01	0	01	01	000	FETCH	0
EXEC	1	10	0	11	10	111	FETCH	0
EXEC	1	11	0	00	10	000	FETCH	0

Part 1 Design



Part 1 Simulation



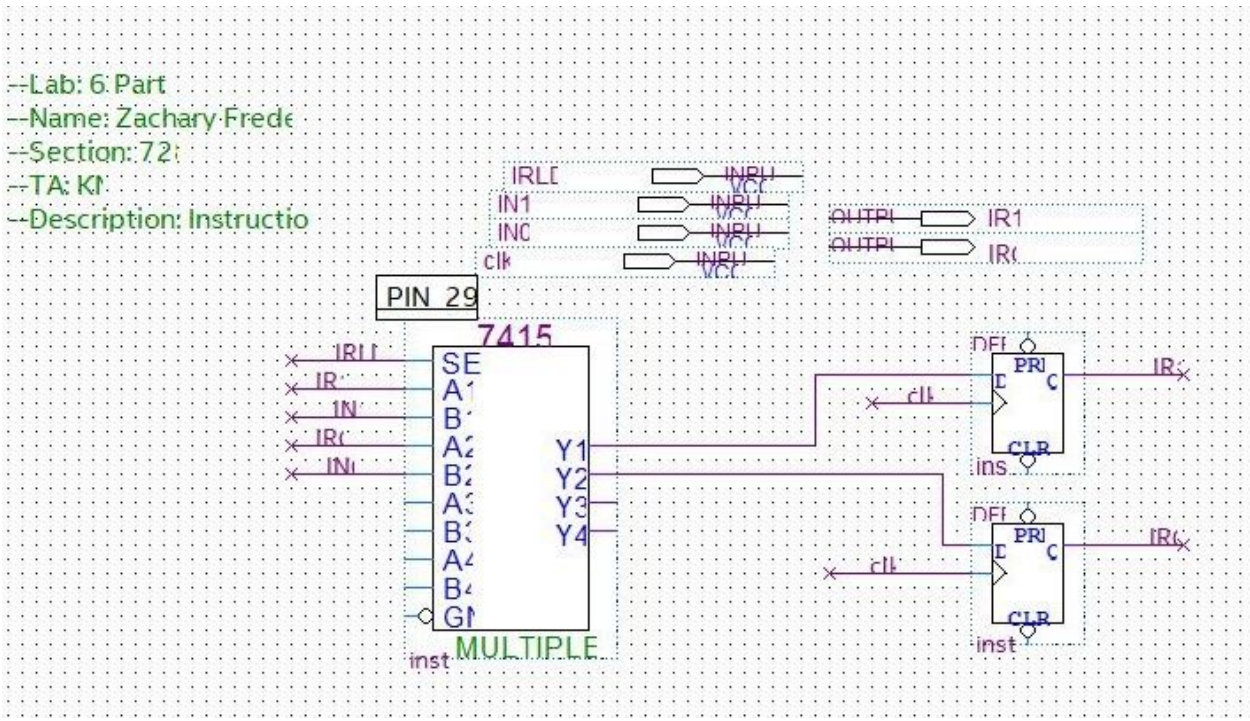
= state before clock cycle

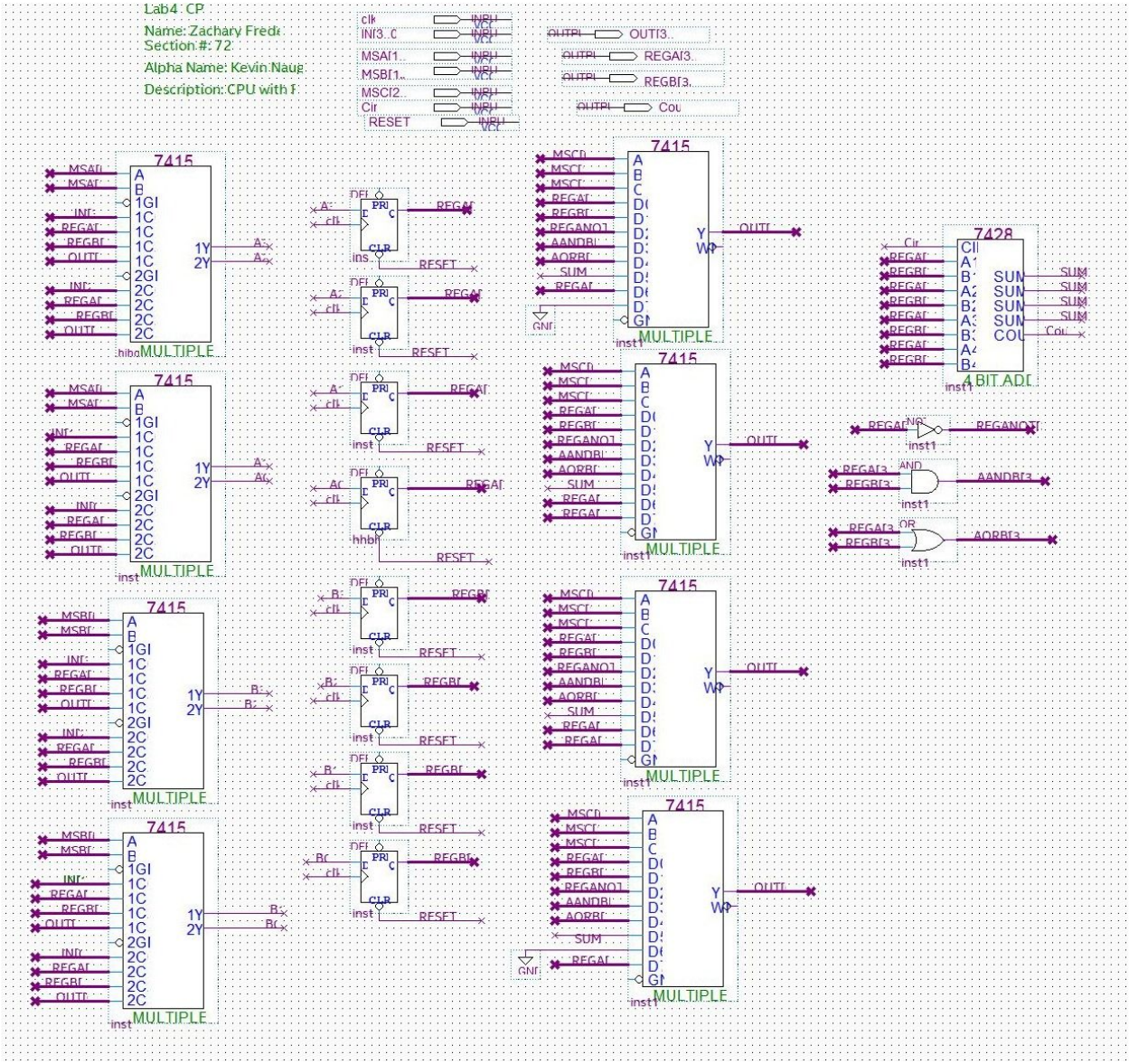
1. Load InstrReg with 11
2. MSA is set to 00 and input is set to 1100. After clock RegA is loaded with 1100
3. Load InstrReg with 01
4. MSB is set to 01 so the value of RegA gets stored in RegB after clock
5. Load InstrReg with 11
6. MSA is set to 00 and input is set to 0011. After clock RegA holds 0011
7. Load InstrReg with 00
8. Output is 1111 (RegA + RegB) and RegA holds this value after the clock
9. Load InstrReg with 10
10. MSA is set to 111 and MSC is set to 111, after clokc RegA holds 0111 which is the value of RegA Right Shifted

When Q = Low that is the state to load an instruction. The last two bits of the input bus will be stored in the instruction register.
When Q = High that is the state to execute the instruction stored in the InstrReg.

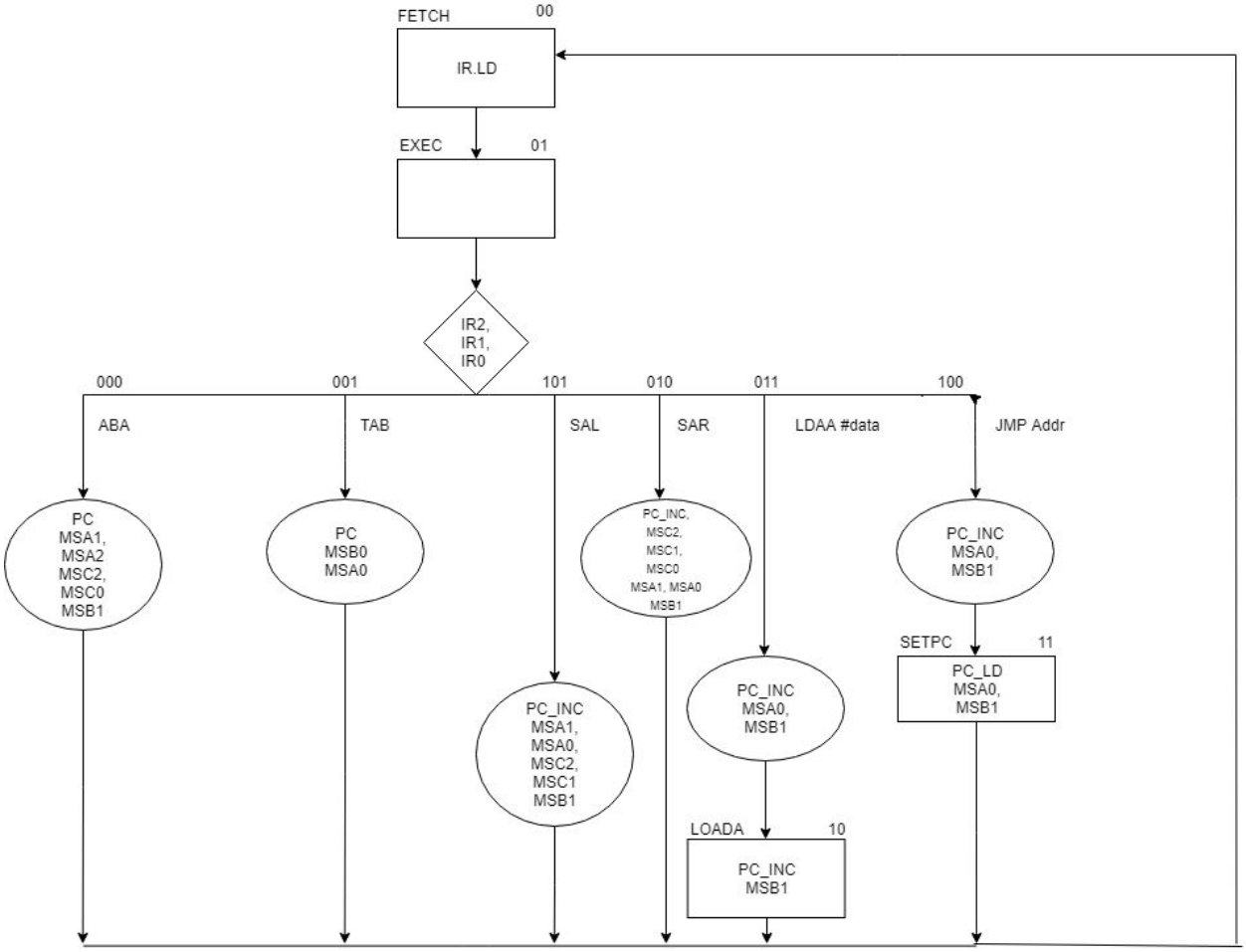
Part 1 Code

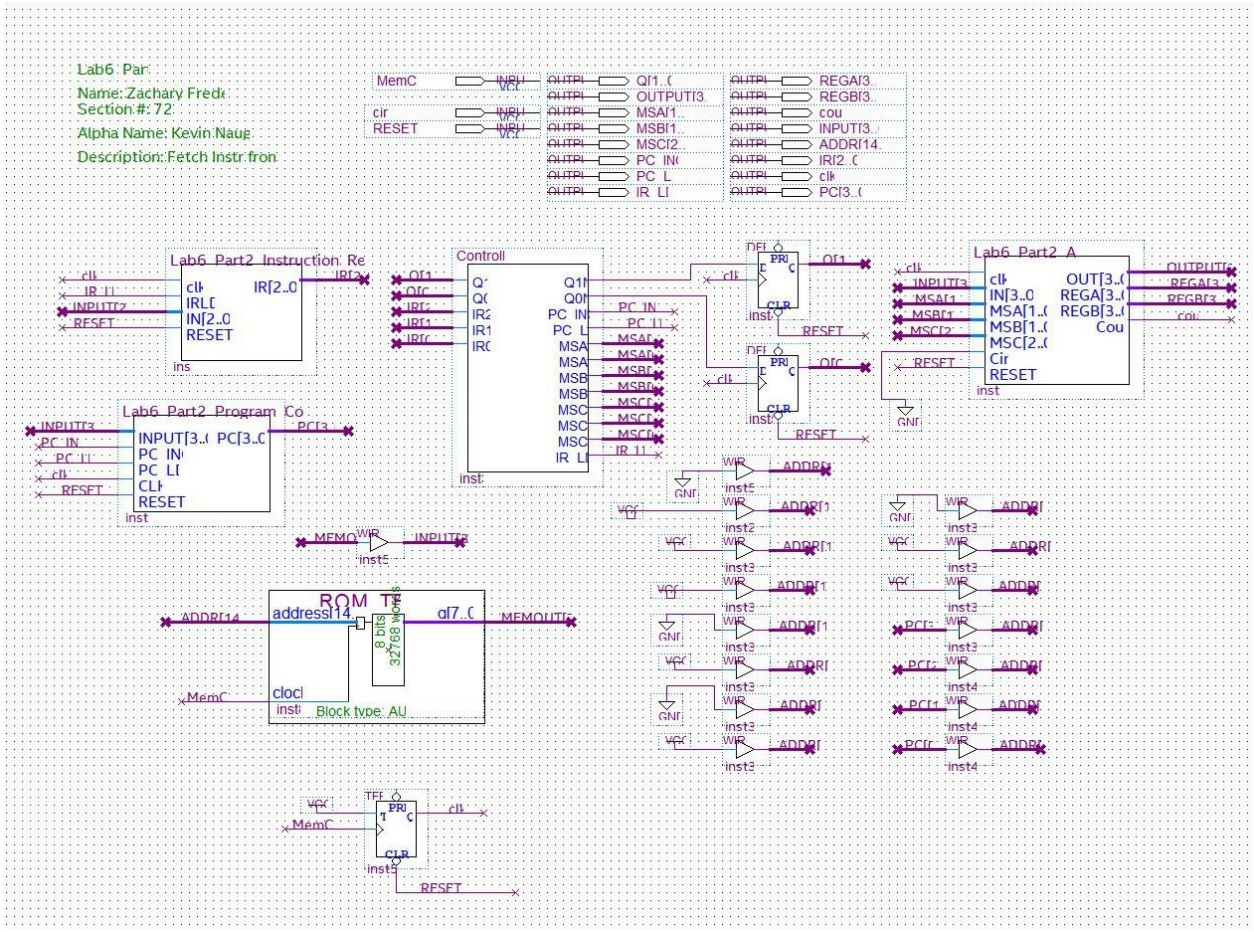
```
--Lab: 6 Part 1
--Name: Zachary Frederick
--Section: 7287
--TA: KN
--Description: Controller vhdl code


entity Controller is port(
Q, IR1, IR0: IN BIT;
IRLD, MSA1, MSA0, MSB1, MSB0: OUT BIT;
MSC2, MSC1, MSC0: OUT BIT;
QN: OUT BIT);
END Controller;

Architecture Behavior of Controller is

begin

QN    <= NOT Q;
IRLD  <= NOT Q;
MSA1  <= (Q AND NOT IR1 AND NOT IR0) OR (Q AND IR1 AND NOT IR0);
MSA0  <= NOT Q OR NOT IR1 OR NOT IR0;
MSB1  <= NOT Q OR IR1 OR NOT IR0;
MSB0  <= Q AND NOT IR1 AND IR0;
MSC2  <= (Q AND NOT IR1 AND NOT IR0) OR (Q AND IR1 AND NOT IR0);
MSC1  <= Q AND IR1 AND NOT IR0;
MSC0  <= (Q AND NOT IR1 AND NOT IR0) OR (Q AND IR1 AND NOT IR0);

end Behavior;
```



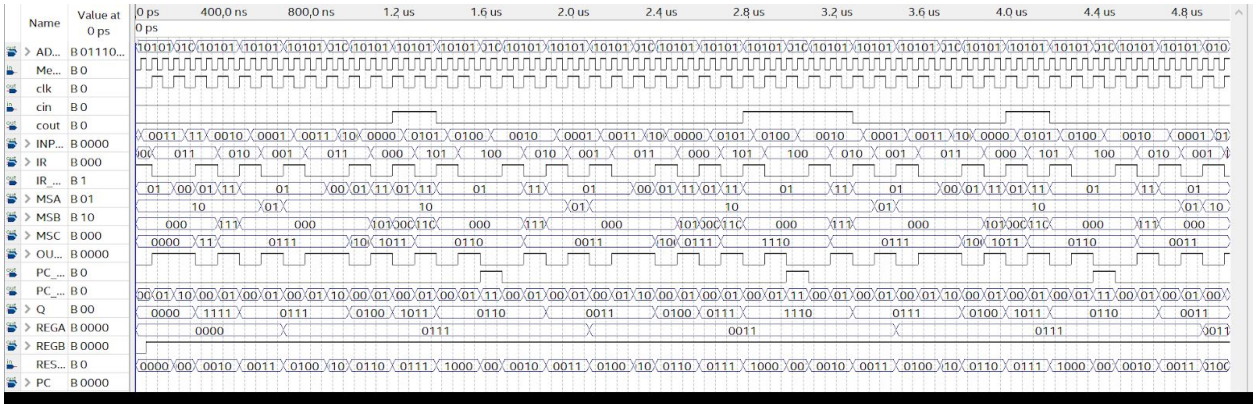

Lab4: CP
Name: Zachary Freda
Section #: 72
Alpha Name: Kevin Naug
Description: CPU with I
7415
MULTIPLE
7428
4 BIT ADD
AND
OR
INV
RESET

Part 2


FETCH
00
IR.LD
EXEC
01
IR2,
IR1,
IR0
000
001
101
010
011
100
ABA
TAB
SAL
SAR
LDAA #data
JMP Addr
PC
MSA1,
MSA2
MSC2,
MSC0
MSB1
PC
MSB0
MSA0
PC_INC,
MSC2,
MSC1,
MSC0
MSA1, MSA0
MSB1
PC_INC
MSA0,
MSB1
SETPC
11
PC_LD
MSA0,
MSB1
PC_INC
MSA1,
MSA0,
MSC2,
MSC1
MSB1
PC_INC
MSA0,
MSB1
LOADA
10
PC_INC
MSB1



Simulation



Pay attention to Input, RegA, RegB and the instruction register. Prior to first clock set Reset_L low to reset CPU.

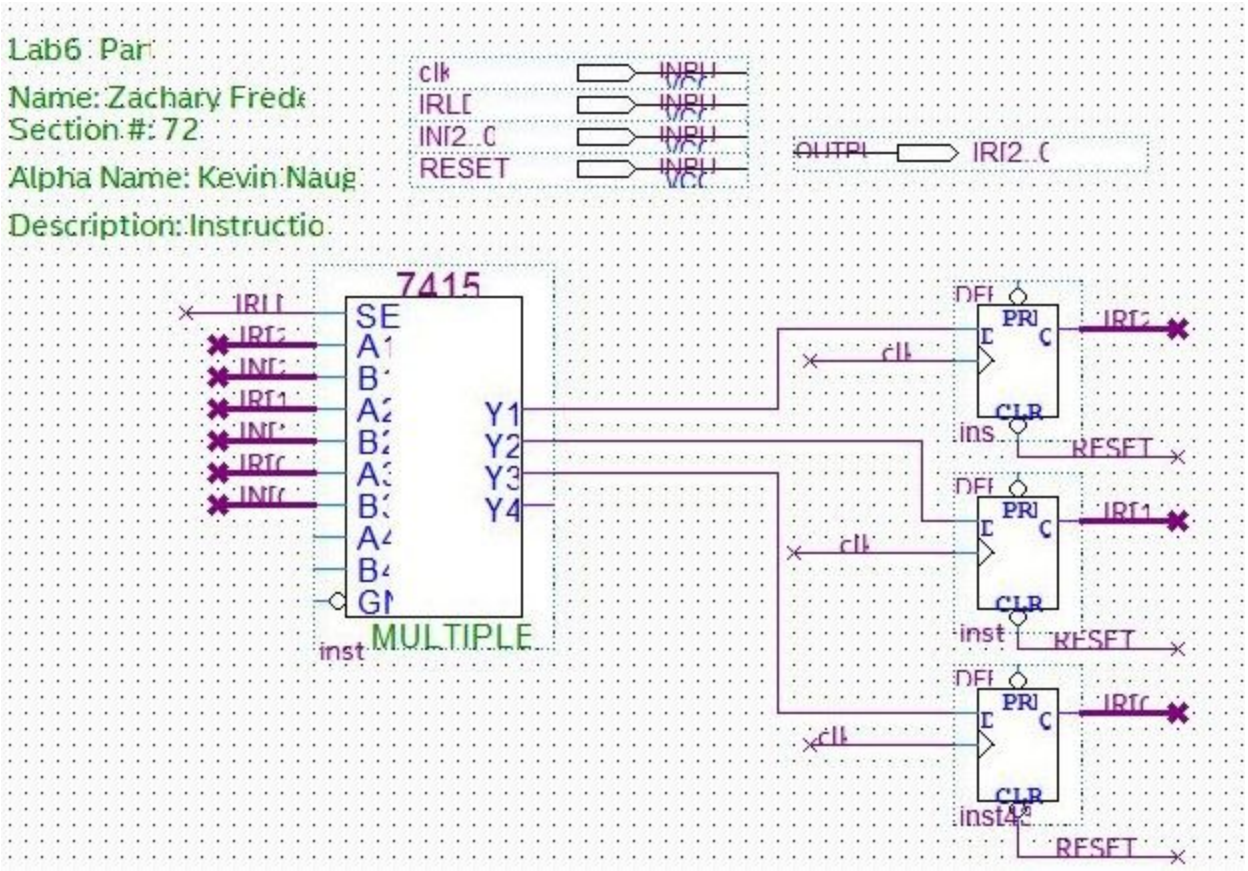
1. During the first fetch state we load 0011 from input to 011 in the IR. On the next cycle we set input to 1111 and load that value into RegA.
2. After this we set the input to 010 which is the SAR opcode. This causes the value of RegA to be shifted from 1111 to 0111.
3. Input is set to 001 which causes the value of RegA to Be transfered to RegB
4. After this we input 011 and also set input to 100. This loads 4 into RegA
5. The next instruction 000 which sums RegA and RegB and stores it in RegA. This sets RegA to 1011 or 11.
6. The next instruction is 101 which is SAL. This shifts the value in RegA left. RegA updates from 1011 to 0110.
7. The next instruction is 010 which is jump to address $3AB2 which sets the PC to 0010 After this the same sequence repeats. The code continues to exectute instructions from $3AB0 to $3AB9 endlessly.

If comparing the values of Input, IR, RegA, and RegB to the hand assembled table you can see that the computer is operating correctly

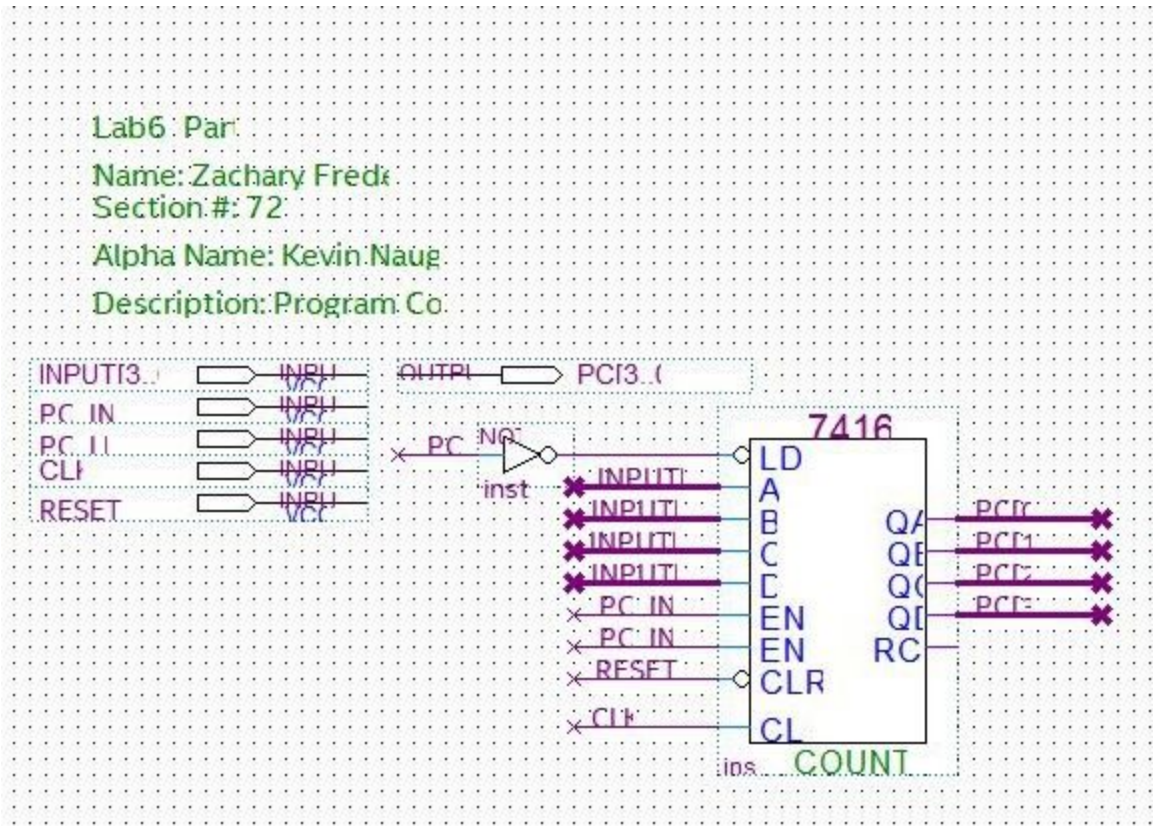
Next State Truth Table

STATE	Q1	Q0	IR2	IR1	IR0	IR.LD	PC_INC	PC_LD	MSA1	MSA0	MSB1	MSB0	MSC2	MSC1	MSC0	STATE+	Q1+	Q0+
FETCH	0	0	X	X	X	1	0	0	0	1	1	0	0	0	0	EXEC	0	1
EXEC	0	1	0	0	0	0	1	0	1	1	1	0	1	0	1	FETCH	0	0
EXEC	0	1	0	0	1	0	1	0	0	1	0	1	0	0	0	FETCH	0	0
EXEC	0	1	0	1	0	0	1	0	1	1	1	0	1	1	1	FETCH	0	0
EXEC	0	1	0	1	1	0	1	0	0	1	1	0	0	0	0	LOADA	1	0
EXEC	0	1	1	0	0	0	1	0	0	1	1	0	0	0	0	SETPC	1	1
EXEC	0	1	1	0	1	0	1	0	1	1	1	0	1	1	0	FETCH	0	0
LOADA	1	0	X	X	X	0	1	0	0	0	1	0	0	0	0	FETCH	0	0
SETPC	1	1	X	X	X	0	0	1	0	1	1	0	0	0	0	FETCH	0	0

Instruction Register



Program Counter



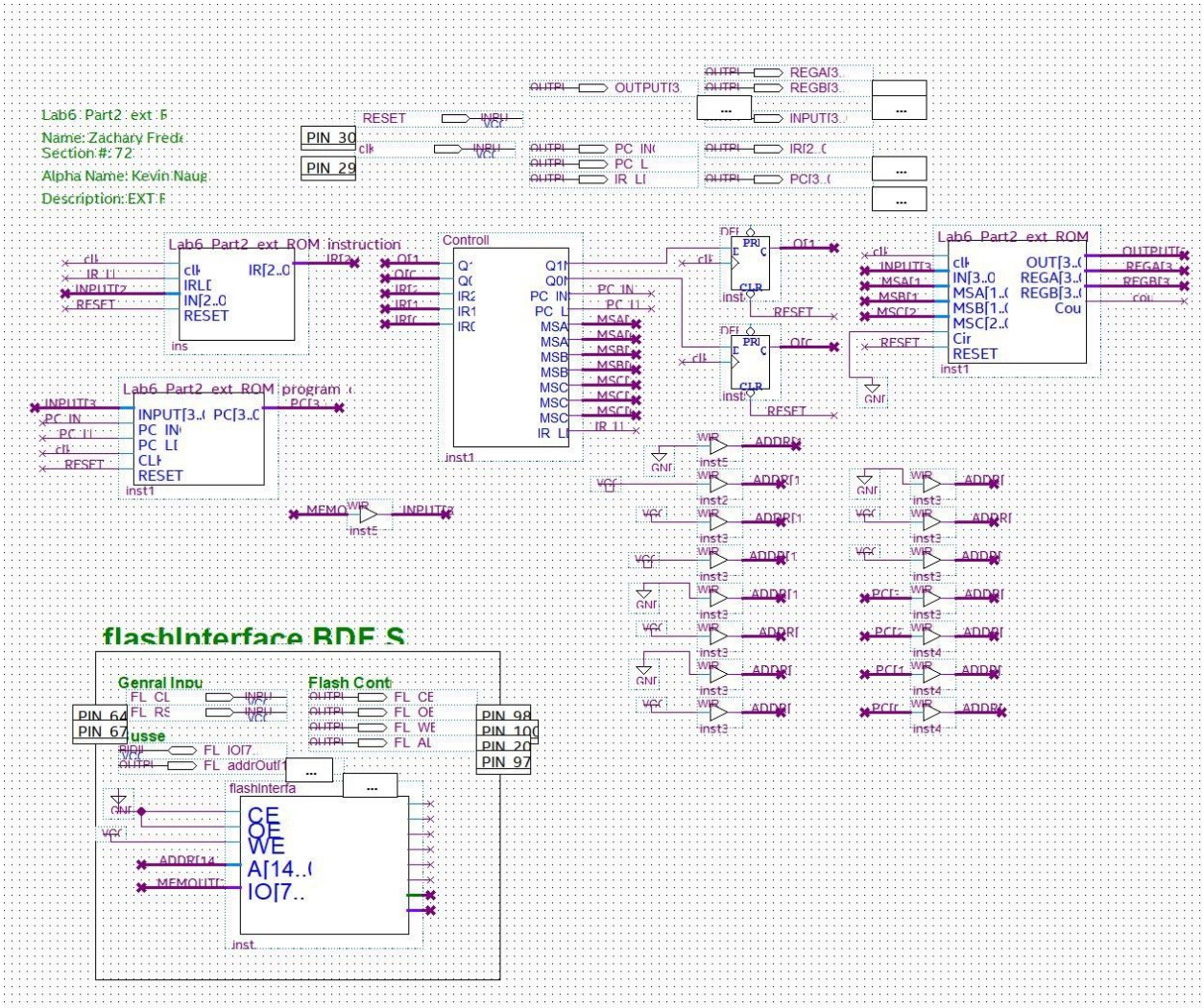
Controller Code

```
--Lab6_Part2
--Name:         Zachary Frederick
--Section #:    7287
--Alpha Name:   Kevin Naughton
--Description: Controller for CPU
entity Controller is port(
Q1, Q0:           IN    BIT;
IR2, IR1, IR0:    IN    BIT;
Q1N, Q0N:         OUT   BIT;
PC_INC, PC_LD:    OUT   BIT;
MSA1, MSA0:       OUT   BIT;
MSB1, MSB0:       OUT   BIT;
MSC2, MSC1, MSC0: OUT   BIT;
IR_LD:            OUT   BIT);
end Controller;
Architecture Behavior of Controller is
begin
Q1N <= (NOT Q1 AND Q0 AND NOT IR2 AND IR1 AND IR0) OR (NOT Q1 AND Q0 AND IR2 AND NOT IR1 AND NOT IR0);
Q0N <= (NOT Q1 AND NOT Q0) OR (NOT Q1 AND Q0 AND IR2 AND NOT IR1 AND NOT IR0);
IR_LD <= (NOT Q1 AND NOT Q0);
PC_INC <= (Q1 OR Q0) AND (NOT Q1 OR NOT Q0);
PC_LD <= Q1 AND Q0;
MSA1 <= (NOT Q1 AND Q0 AND NOT IR2 AND NOT IR1 AND NOT IR0)
OR (NOT Q1 AND Q0 AND NOT IR2 AND IR1 AND NOT IR0)
OR (NOT Q1 AND Q0 AND IR2 AND NOT IR1 AND IR0);
MSA0 <= NOT Q1 OR Q0;
MSB1 <= Q1 OR NOT Q0 OR IR2 OR IR1 OR NOT IR0;
MSB0 <= NOT Q1 AND Q0 AND NOT IR2 AND NOT IR1 AND IR0;
MSC2 <= (NOT Q1 AND Q0 AND NOT IR2 AND NOT IR1 AND NOT IR0)
OR (NOT Q1 AND Q0 AND NOT IR2 AND IR1 AND NOT IR0)
OR (NOT Q1 AND Q0 AND IR2 AND NOT IR1 AND IR0);
MSC1 <= (NOT Q1 AND Q0 AND NOT IR2 AND IR1 AND NOT IR0)
OR (NOT Q1 AND Q0 AND IR2 AND NOT IR1 AND IR0);
MSC0 <= (NOT Q1 AND Q0 AND NOT IR2 AND NOT IR1 AND NOT IR0)
OR (NOT Q1 AND Q0 AND NOT IR2 AND IR1 AND NOT IR0);
end Behavior;
```

Assembly table

Address	Instruction	Machine Code	A	B	A	B	A	B	A	B
$3AB0	LDAA $15	$3 $F	$15	-	-	-	-	-	-	-
$3AB2	SAR	$2	$7	-	$3	$7	$7	$3	$3	$7
$3AB3	TAB	$1	$7	$7	$3	$3	$7	$7	$3	$3
$3AB4	LDAA #4	$3 $4	$4	$7	$4	$3	$4	$7	$4	$3
$3AB6	ABA	$0	$B	$7	$7	$3	$B	$7	$7	$3
$3AB7	SAL	$5	$6	$7	$E	$3	$6	$7	$E	$3
$3AB8	JMP 2	$4 $2	$6	$7	$E	$3	$6	$7	$E	$3
$3ABA	TAB	$1	-	-	-	-	-	-	-	-

Lab6_Part2_Ext_Rom Design



Problems Encountered

No major problems were encountered. Everything went smoothly with this lab.

Future Work

I would be interested in expanding the instruction register for more opcodes and a bigger memory bus to be able to load more data at once.