An 8-Bit CPU Control Circuit Design
CSE 207 Final Project

Section 2

Haibei Zhang

Dec 10, 2002

I. Objectives

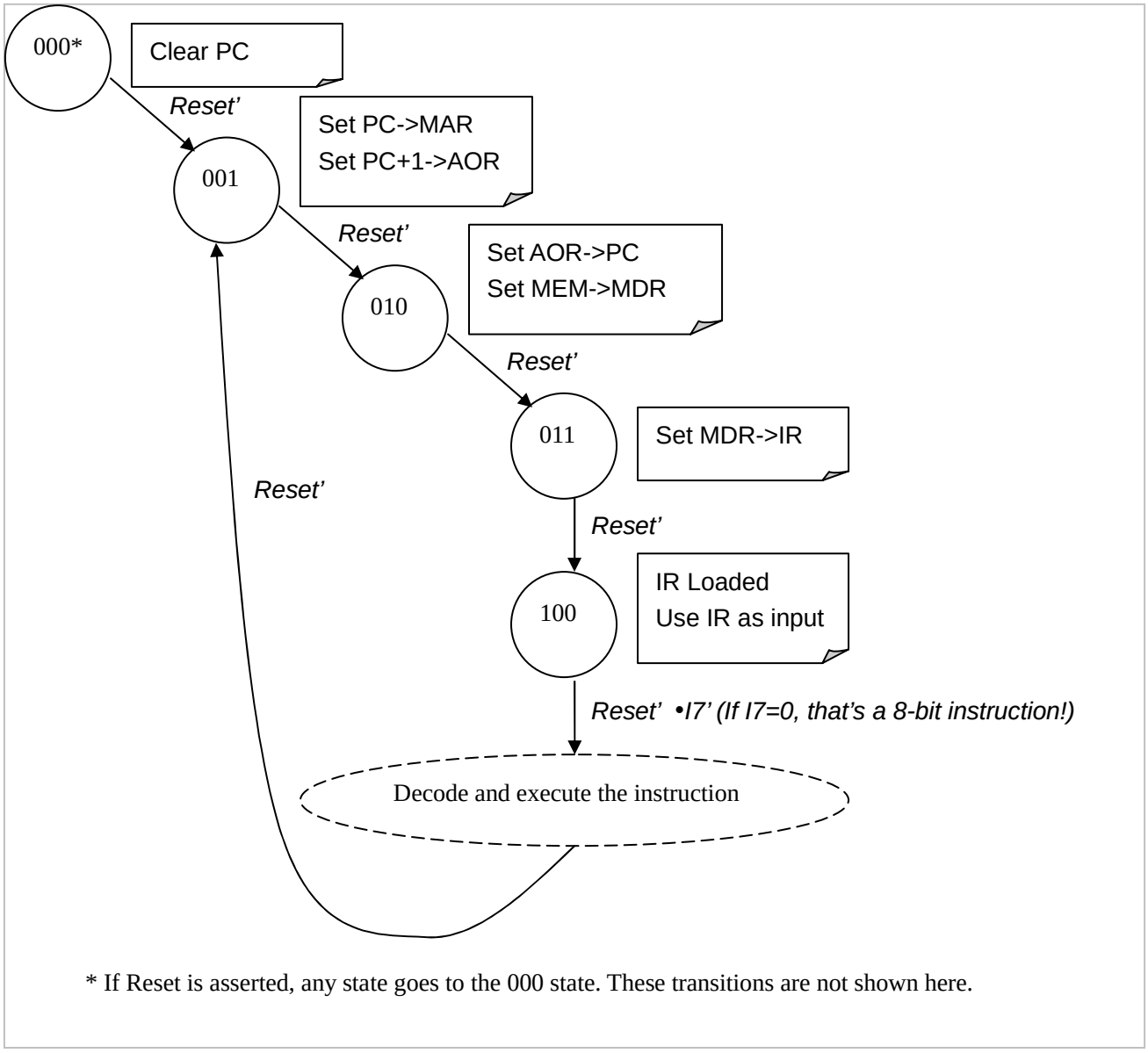
The objective of this project is to design a control circuit for the Robertics WFBCSC II CPU, a 8-bit Von Neumann machine.

The inputs and outputs are:

Input	**Output *(Abbreviation)***
Reset (Active High)	PC_TO_BUS (Active Low) (*PB)*
ZCC (Zero Condition Code)	BUS_TO_PC (Active Low) (*BP)*
Clock	BUS_TO_MAR (Active Low) (*BMAR)*
I7 (Instruction 8th bit)	BUS_TO_MDR (Active Low) (*BMDR)*
I6 (Instruction 7th bit)	MDR_TO_BUS (Active Low) (*MDRB)*
I5 (Instruction 6th bit)	R (Active Low)
I4 (Instruction 5th bit)	W (Active Low)
I3 (Instruction 4th bit)	BUS_TO_AIR (Active Low) (*BAIR)*
I2 (Instruction 3rd bit)	AOR_TO_BUS (Active Low) (*AORB)*
I1 (Instruction 2nd bit)	BUS_TO_IR (Active Low) (*BI)*
I0 (Instruction 1st bit)	R3 (Register Address 4th bit)
	R2 (Register Address 3rd bit)
	R1 (Register Address 2nd bit)
	R0 (Register Address 1st bit)
	REG_TO_BUS (Active Low) (*RB)*
	BUS_TO_REG (Active Low) (*BR)*
	S3
	S2
	S1
	S0
	CIN (Active Low) *(CN)*

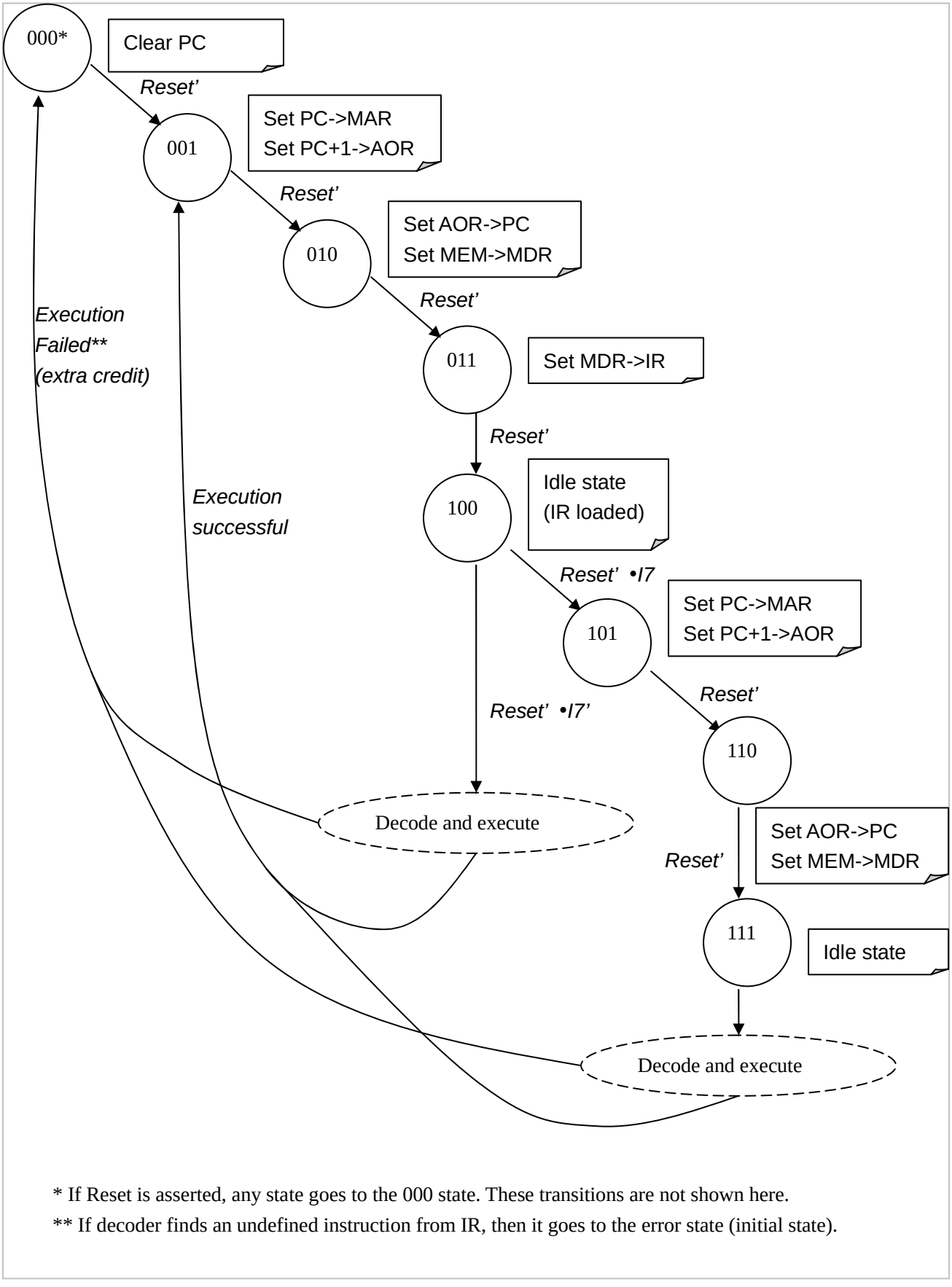
II. Design Procedures

1. Design the state diagram for the 8-bit instruction fetching. (Each state is assigned a 3-bit binary number)



* If Reset is asserted, any state goes to the 000 state. These transitions are not shown here.

2. Design the complete state diagram for both 8-bit and 16-bit instruction fetching. (Each state is assigned a 3-bit binary number)



* If Reset is asserted, any state goes to the 000 state. These transitions are not shown here.

** If decoder finds an undefined instruction from IR, then it goes to the error state (initial state).

3. Design the decoder

At state 100 and 111 shown above, IR is ready to input instruction into the circuit. I7 is used to determine whether the instruction is 8-bit or 16-bit (at state 100). I6, I5, I4 are used to choose a circuit which handles the execution of the instruction. (The 3 inputs of the decoders.) Therefore two 74138 decoders are used for 8-bit and 16-bit instructions. For 8 bit instructions, the decoder is enabled when state is 100, Reset=0, I7=0; for 16 bit instructions, the decoder is enabled when state is 111 and Reset=0. Once the decoder is enabled, it decodes I6, I5, I4, then sets one of its output pin active (the **Start** signal for the execution circuit), which triggers that execution to run.

4. Design the execution circuit (instruction processor)

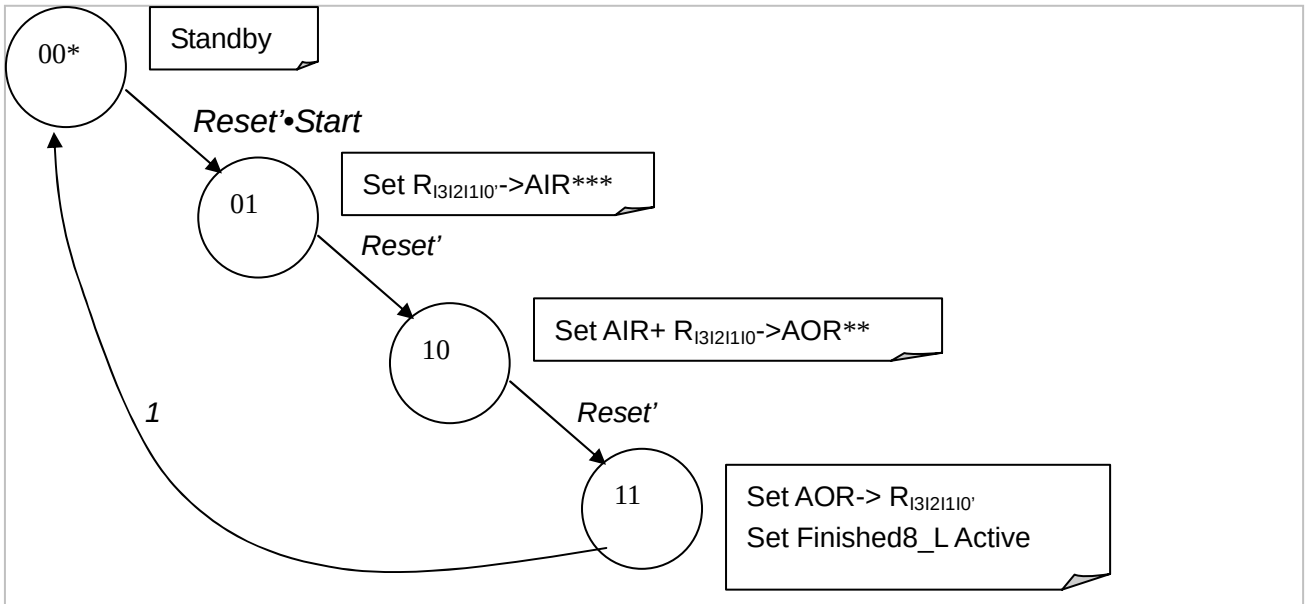
Each instruction has its own circuit, which are also state machines. These machines are normally staying in a standby state. Once it receives a "**Start**" signal from the decoder, it starts running.

When the instruction-fetching machine reaches state 100 and 111, the next state will take instruction processor's output into consideration. When the instruction is still in processing, it stays at 100 or 111; when the instruction is finished, it goes to 001; when the instruction encounters error, it goes to 000.

Therefore, the execution circuit should have these 2 outputs: **Finished_L, Error_L**. If both of them are inactive, the machine is processing an instruction.

Following are transition diagrams for each instruction:

ADDR (Opcode: 0010)

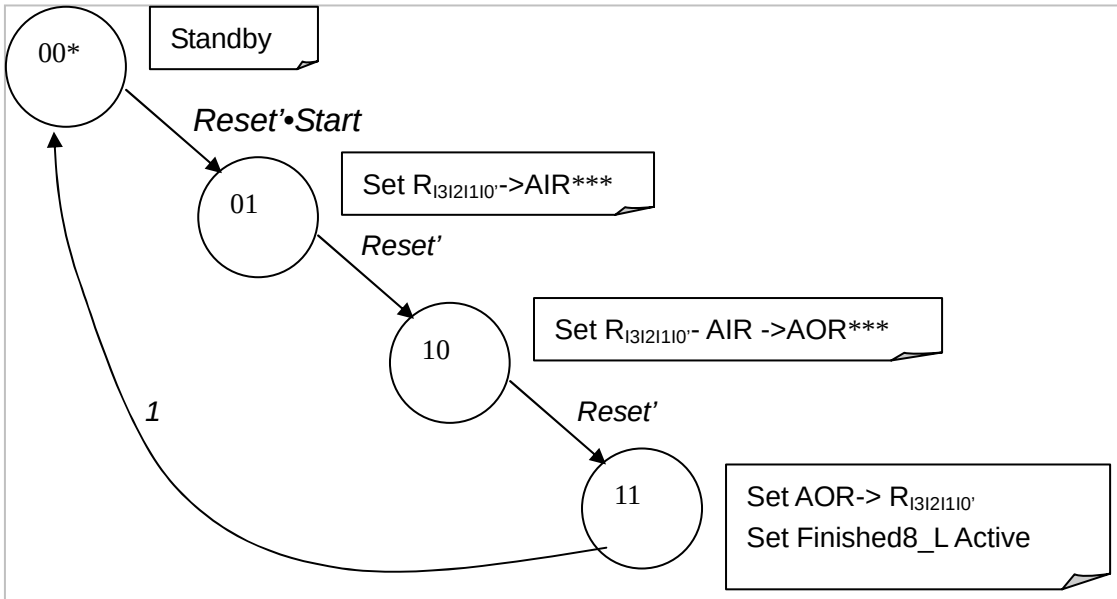


* If Reset is asserted, any state goes to the 00 state. These transitions are not shown here.

** "R$_{I3I2I1I0}$" means "data at register address I3 I2 I1 I0"

*** "R$_{I3I2I1I0'}$" means "data at register address I3 I2 I1 I0' " where I0' is I0 inverted

SUBR (Opcode: 0011)

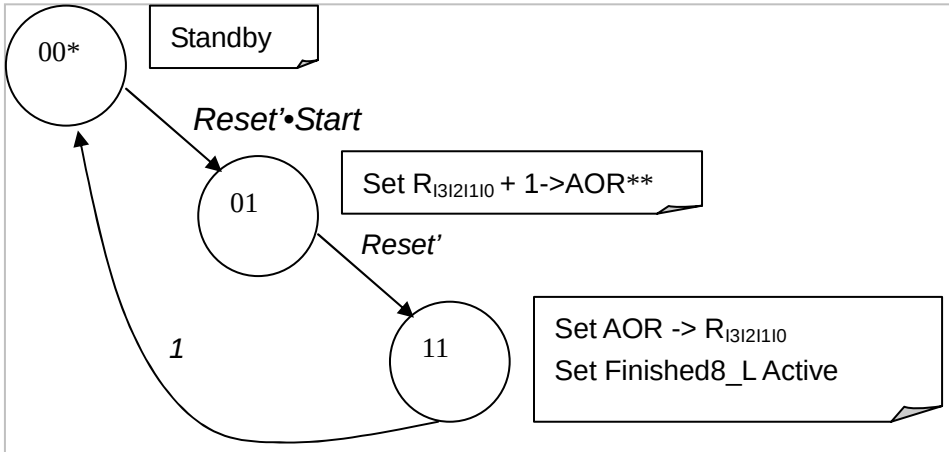


* If Reset is asserted, any state goes to the 00 state. These transitions are not shown here.

** "R$_{I3I2I1I0}$" means "data at register address I3 I2 I1 I0"

*** "R$_{I3I2I1I0'}$" means "data at register address I3 I2 I1 I0' " where I0' is I0 inverted

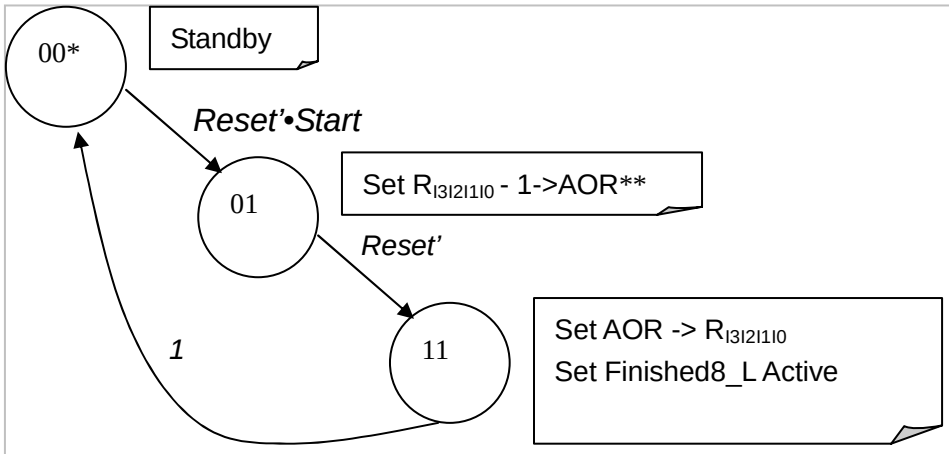
INCR (Opcode: 0100)



* If Reset is asserted, any state goes to the 00 state. These transitions are not shown here.

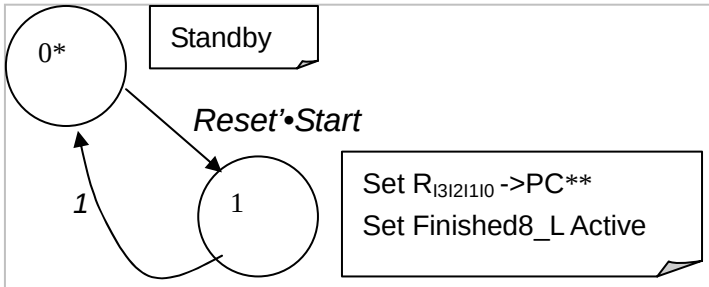
** "R$_{I3I2I1I0}$" means "data at register address I3 I2 I1 I0"

DECR (Opcode: 0101)



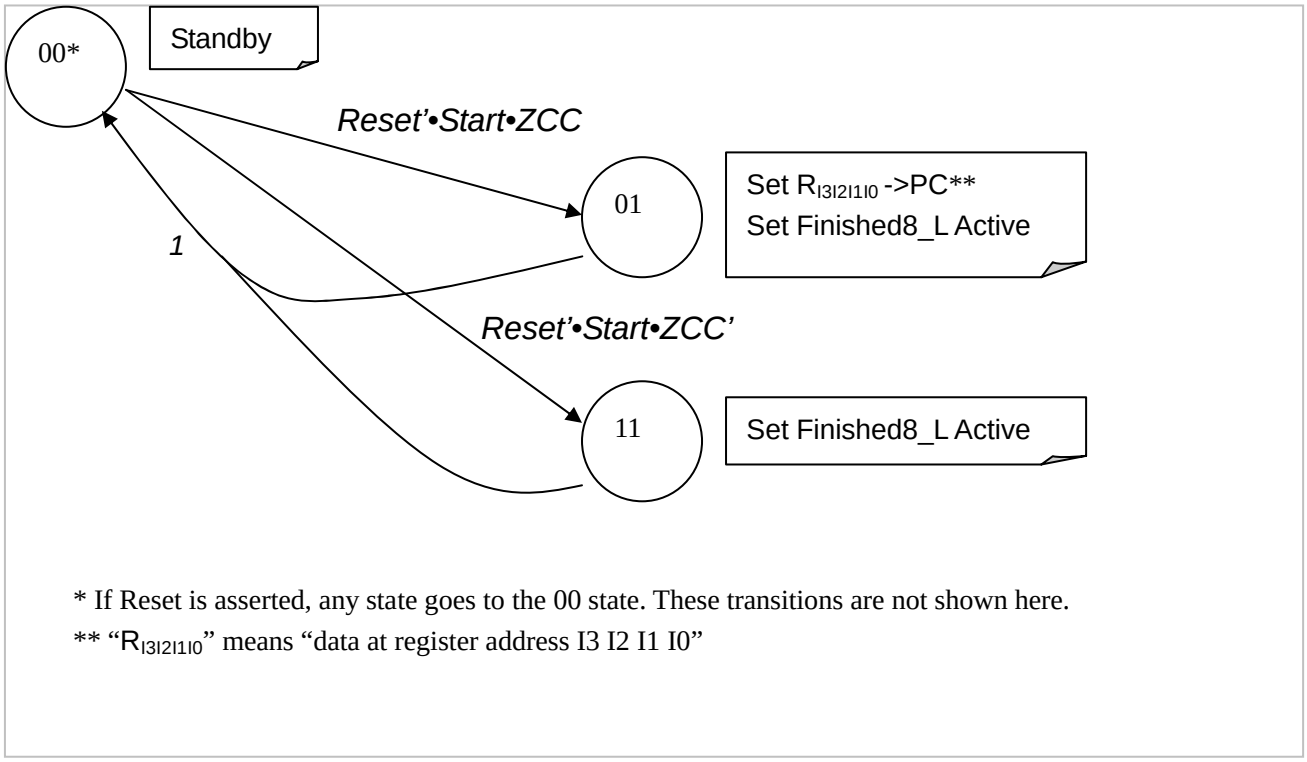
* If Reset is asserted, any state goes to the 00 state. These transitions are not shown here.
** "R$_{I3I2I1I0}$" means "data at register address I3 I2 I1 I0"

BR (Opcode: 0110)



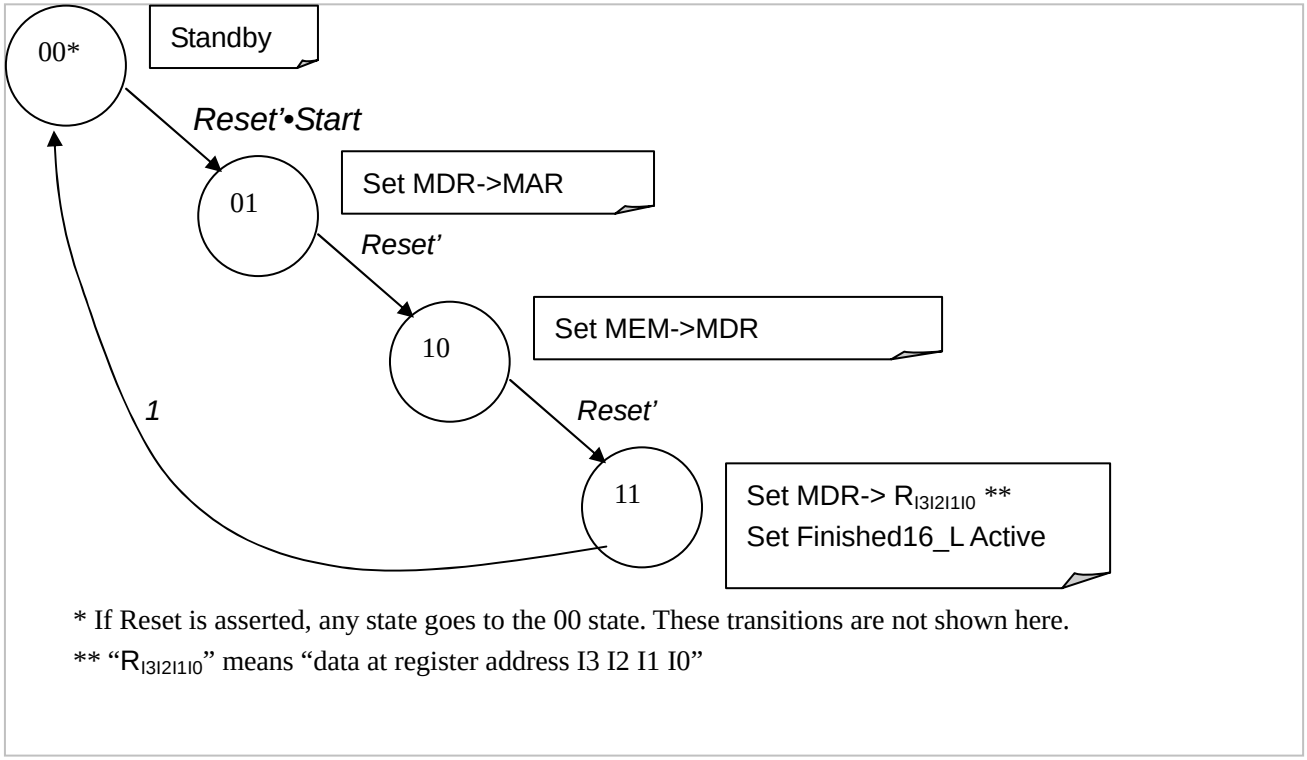
* If Reset is asserted, any state goes to the 0 state. These transitions are not shown here.
** "R$_{I3I2I1I0}$" means "data at register address I3 I2 I1 I0"

BRZ (Opcode: 0111)



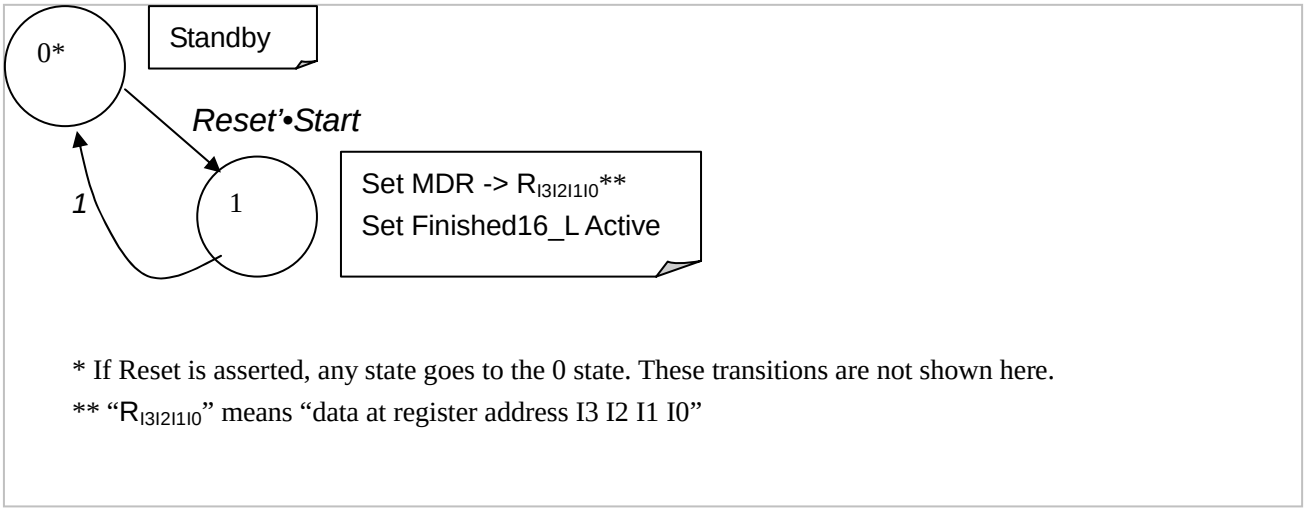
* If Reset is asserted, any state goes to the 00 state. These transitions are not shown here.
** "$R_{I3I2I1I0}$" means "data at register address I3 I2 I1 I0"

LOAD (Opcode: 1000)



* If Reset is asserted, any state goes to the 00 state. These transitions are not shown here.
** "$R_{I3I2I1I0}$" means "data at register address I3 I2 I1 I0"

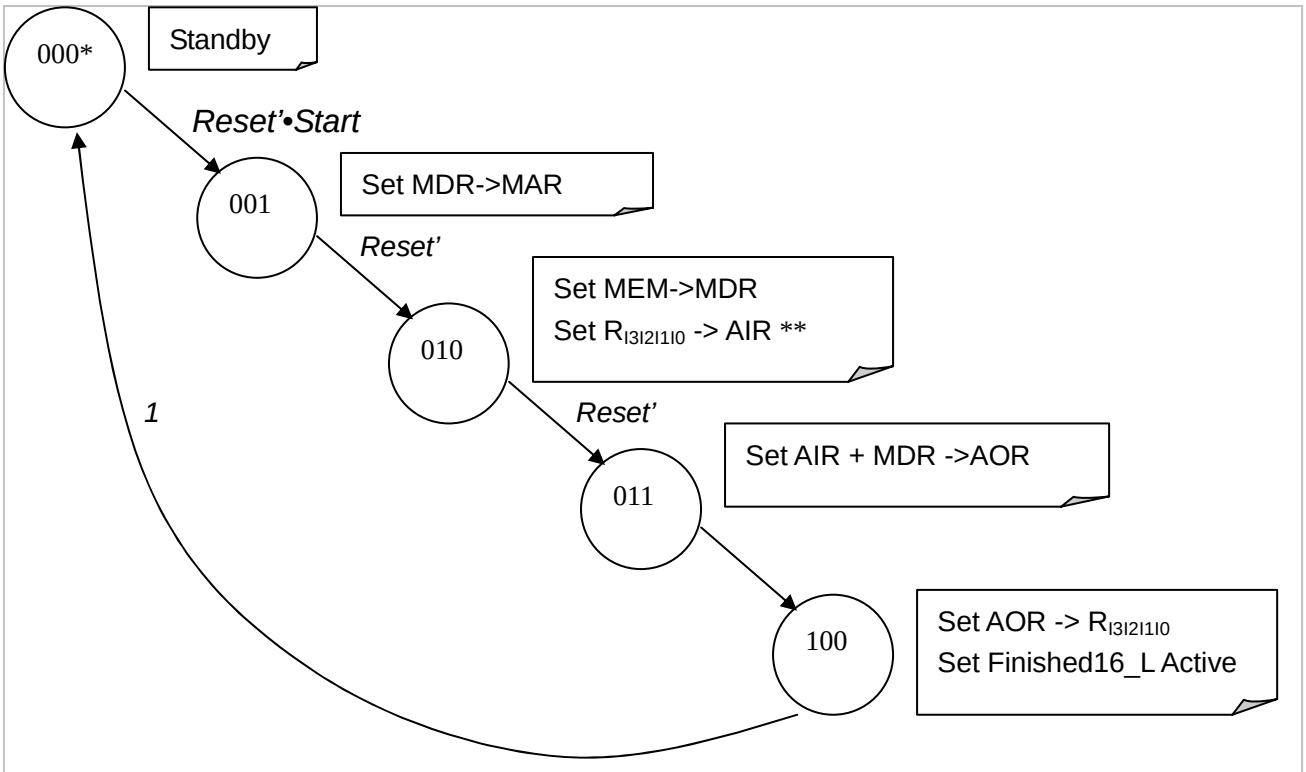
LDI (Opcode: 1001)



* If Reset is asserted, any state goes to the 0 state. These transitions are not shown here.

** "$R_{I3I2I1I0}$" means "data at register address I3 I2 I1 I0"

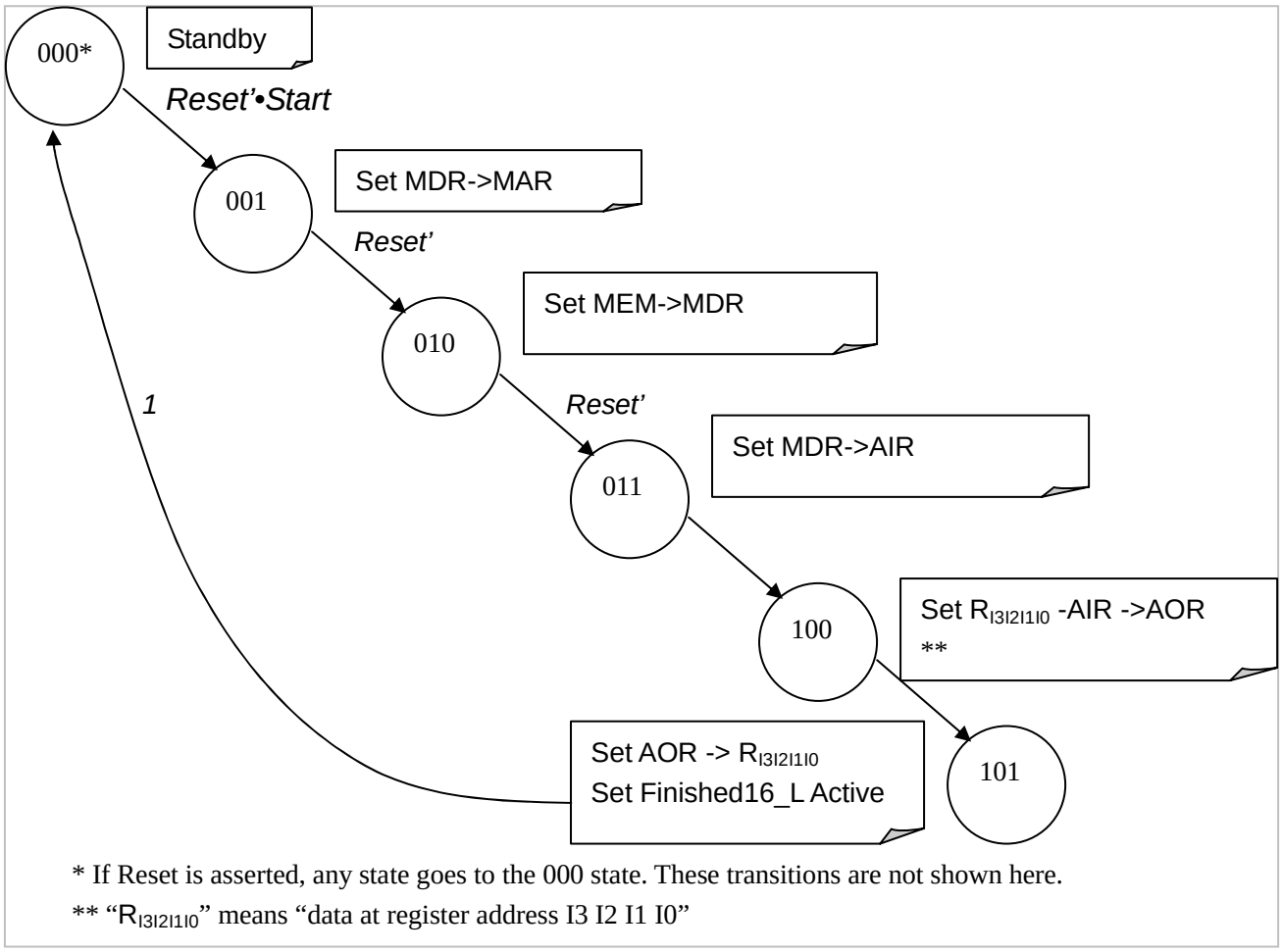
ADD (Opcode: 1010)



* If Reset is asserted, any state goes to the 000 state. These transitions are not shown here.

** "$R_{I3I2I1I0}$" means "data at register address I3 I2 I1 I0"

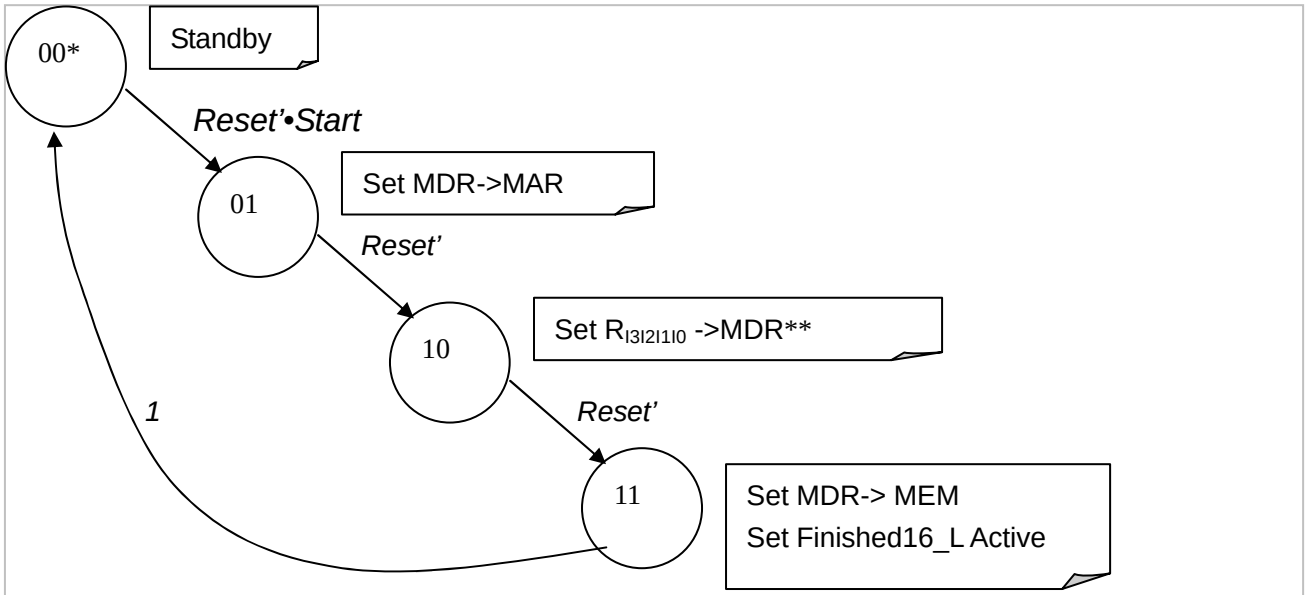
SUB (Opcode: 1011)



* If Reset is asserted, any state goes to the 000 state. These transitions are not shown here.

** "$R_{I3I2I1I0}$" means "data at register address I3 I2 I1 I0"

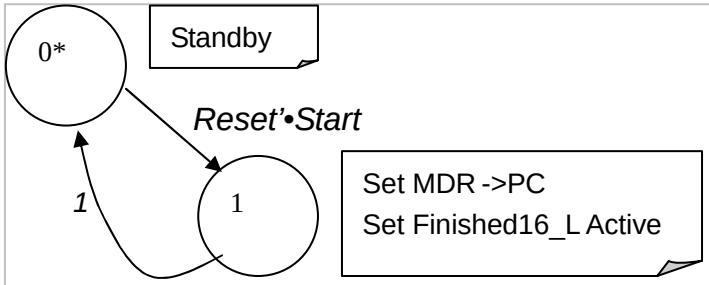
STORE (Opcode: 1100)



* If Reset is asserted, any state goes to the 00 state. These transitions are not shown here.

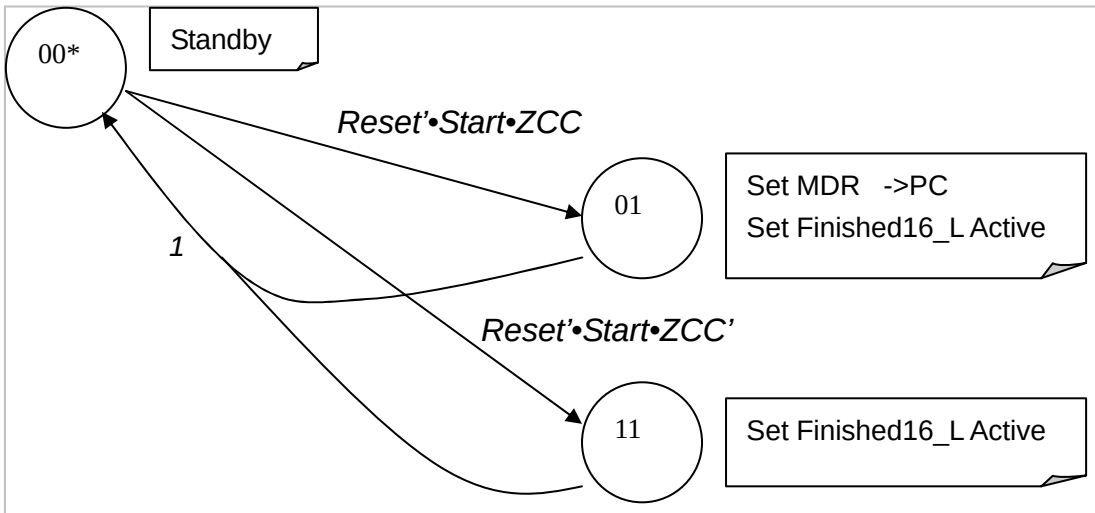
** "$R_{I3I2I1I0}$" means "data at register address I3 I2 I1 I0"

B (Opcode: 1110)



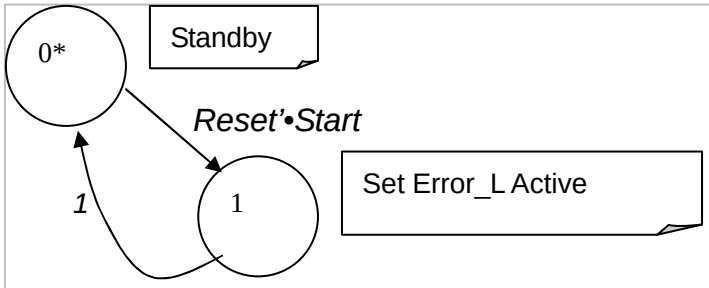
* If Reset is asserted, any state goes to the 0 state. These transitions are not shown here.

BZ (Opcode: 1111)



* If Reset is asserted, any state goes to the 00 state. These transitions are not shown here.

UNDEFINED INSTRUCTIONS (Opcode: 0000, 0001, 1101)



* If Reset is asserted, any state goes to the 0 state. These transitions are not shown here.

(For extra credit!)

III. <u>Transition Lists, Excitation Equations (D Flip-Flop), Output Equations (This part was finished before Robert introduced MSI Counter. In practice, I will use MSI counter rather than D Flip-Flop.)</u>

The Main Circuit (Instruction Fetching Cycle)

a) State – Next State transition list:

$Q_2Q_1Q_0$	Reset (Active High)	
	1	0
000	000	001
001	000	010
010	000	011
011	000	100
100	000	001 if I7=0, Finished8_L=0 (go to next cycle)
		000 if I7=0, Error_L=0 (go to error state)
		100 if I7=0, Error_L=1, Finished8_L=1 (waiting)
		101 if I7=1 (go on fetching a 16-bit inst.)
101	000	110
110	000	111
111	000	001 if Finished16_L=0 (go to next cycle)
		000 if Error_L=0 (go to error state)
		111 if Finished16_L=1, Error_L=1 (waiting)
	$D_2D_1D_0$ (Q_2*Q_1*Q_0*)	

b) Excitation Equation:

D_2=Reset'•(Q_2'•Q_1•Q_0 + Q_2•Q_1'•Q_0'•(I7 + I7'•Finished8_L•Error_L) + Q_2•Q_1'•Q_0 + Q_2•Q_1•Q_0' + Q_2•Q_1•Q_0•Finished16_L•Error_L)

D_1=Reset'•(Q_1•Q_0' + Q_1'•Q_0 + Q_2•Q_1•Q_0•Finished16_L•Error_L)

D_0=Reset'•(Q_2'•Q_0' + Q_1•Q_0' + Q_2•Q_1'•Q_0'•(I7'•Finished8_L' + I7) + Q_2•Q_1•Q_0•(Finished16_L' + Finished16_L•Error_L))

c) State – Output table

Part 1:

$Q_2Q_1Q_0$	PB	BP	BMAR	BMDR	MDRB	R	W	BAIR	AORB	BI
000	1	1	1	1	1	1	1	1	1	1
001	0	1	0	1	1	1	1	1	1	1
010	1	0	1	1	1	0	1	1	0	1
011	1	1	1	1	0	1	1	1	1	0
100*	1	1	1	1	1	1	1	1	1	1
101	0	1	0	1	1	1	1	1	1	1
110	1	0	1	1	1	0	1	1	0	1
111*	1	1	1	1	1	1	1	1	1	1

Part 2:

$Q_2Q_1Q_0$	R3	R2	R1	R0	RB	BR	S3	S2	S1	S0	CN
000	1	1	1	1	1	1	1	1	1	1	1
001	1	1	1	1	1	1	0	0	0	0	0
010	1	1	1	1	1	1	1	1	1	1	1
011	1	1	1	1	1	1	1	1	1	1	1
100*	1	1	1	1	1	1	1	1	1	1	1
101	1	1	1	1	1	1	0	0	0	0	0
110	1	1	1	1	1	1	1	1	1	1	1
111*	1	1	1	1	1	1	1	1	1	1	1

*In the state 100 and 111, outputs are controlled by execution circuits (the main circuit is disconnected from outputs). Therefore, use Hi-Z signal instead of 1 in practice.

d) Output Equations:
PB=BMAR=S3=S2=S1=S0=CN=$Q_1 + Q_0$'
BP=R=AORB=Q_1' + Q_0
MDRB=BI=$Q_2 + Q_1$' + Q_0'
All other outputs are not affected by this circuit.

IV. <u>Transition Lists and Equations (Using MSI Counter 74x163)</u>
The Main Circuit (Instruction Fetching Cycle)

Step	Reset (Active High)	
	1	0
St0	St0	St1
St1	St0	St2
St2	St0	St3
St3	St0	St4
St4	St0	St1 if Finished8_L=0 (go to next cycle)
		St0 if Error_L=0 (go to error state)
		St4 if I7=0, Error_L=1, Finished8_L=1 (waiting)
		St5 if I7=1 (go on fetching a 16-bit inst.)
St5	St0	St6
St6	St0	St7
St7	St0	St1 if Finished16_L=0 (go to next cycle)
		St0 if Error_L=0 (go to error state)
		St7 if Finished16_L=1, Error_L=1 (waiting)
	Next Step	

The CLR input of counter is set:
CLR=Reset'

The LOAD of counter is set:
LOAD=Step4•Step7

Input values C, B, A of the counter need to be set at Step4 and Step7 (by converting from D Flip-Flop's excitation equations):
C=Error_L• (I7+Finished8_L) • (I7'+Finished16_L)
B=Finished16_L•Error•Step7'
A=Error_L• (Step4+I7+Error_L'+Finished8_L')

State(Step) – Output table
Part 1:

Step	PB	BP	BMAR	BMDR	MDRB	R	W	BAIR	AORB	BI
St0	1	1	1	1	1	1	1	1	1	1
St1	0	1	0	1	1	1	1	1	1	1
St2	1	0	1	1	1	0	1	1	0	1
St3	1	1	1	1	0	1	1	1	1	0
St4*	1	1	1	1	1	1	1	1	1	1
St5	0	1	0	1	1	1	1	1	1	1
St6	1	0	1	1	1	0	1	1	0	1
St7*	1	1	1	1	1	1	1	1	1	1

Part 2:

Step	R3	R2	R1	R0	RB	BR	S3	S2	S1	S0	CN
St0	1	1	1	1	1	1	1	1	1	1	1
St1	1	1	1	1	1	1	0	0	0	0	0
St2	1	1	1	1	1	1	1	1	1	1	1
St3	1	1	1	1	1	1	1	1	1	1	1
St4*	1	1	1	1	1	1	1	1	1	1	1
St5	1	1	1	1	1	1	0	0	0	0	0
St6	1	1	1	1	1	1	1	1	1	1	1
St7*	1	1	1	1	1	1	1	1	1	1	1

Instruction execution circuits
A similar MSI counter and decoder is used for each instruction processor. I don't list all state-output tables here because it's easy to look at the state diagram and decide which outputs need to me set.

The three undefined instructions are also handled by resetting the state of the instruction-fetching circuit. (For extra credit.)

V. <u>Testing</u>
Testing of this CPU involves a circuit level testing and a system level testing.
1. Circuit level testing (See file ProcessorControlCircuit.cct.)
The control circuit alone can be tested by manually setting inputs and observe

the values of each output.

First, find the “testing area” and attach all binary switches to the input. (I have to detach them so that the circuit can work in the whole system.) Now, reset the circuit by setting Reset=1. Then input different I7I6I5I4 and see the state of the circuit and all output values at each state. The testing is easy because every state and output should be same as described in the diagrams in this document.

I have tested all 16 types of instructions (including undefined). All observed values are expected. The circuit is working fine at this level.

2. System level testing (See file processor.cct)

Now this circuit is transferred into a symbol and installed in the main CPU architecture. I have finished the testing of following list of instructions:

Memory Addr.	Instruction /Hex value	Comments
00000000	1001 1001 /99	LDI (load a value to a register)
00000001	0010 1100 /2C	The value of last LDI inst. (2c)
00000002	0100 1001 /49	INCR (increase that value)
00000003	1001 1000 /98	LDI (load another value)
00000004	1000 0000 /80	The value of last LDI inst. (80)
00000005	0011 1000 /38	SUBR between 2 register values

The final result (Register 1000’s value) is expected to be 53 (Hexadecimal).

Procedures:

i) Set control circuit’s Reset=1 (Resent is active high)

ii) Enable the 2 bus drivers at the bottom (the 8-bit 74x541 to set data, another 4-bit one to set control signals) so that you can manually set values.

iii) Set pc to 00000000

iv) Load memory with above instructions one by one, from memory address 00000000 first. This is a very tedious job. For each instruction, I need to (1) load memory address to MAR (2) load instruction into MDR (3) assert W.

v) Once all instructions have been input into the memory, disable the 2 bus drivers (disconnect the manual input), and set Reset=0. Show time begins:

Here are the values read from Hex probes, each row indicates a clock cycle.

BUS	PC	MAR	MDR	AIR	AOR	IR
xx	00	Last	Last	xx	xx	Last
00	00	Last	Last	Xx	Xx	Last
01	00	00	Last	xx	01	Last

99	01	00	99	xx	xx	Last
xx	01	00	99	xx	xx	99
01	01	00	99	xx	xx	99
02	01	01	99	Xx	02	99
Xx	02	01	2c	xx	xx	99
2c	02	01	2c	xx	xx	99
02	02	01	2c	xx	xx	99
03	02	02	2c	Xx	03	99
49	03	02	49	xx	xx	99
xx	03	02	49	Xx	Xx	49
2c	03	02	49	xx	xx	49
2d	03	02	49	Xx	2d	49
03	03	02	49	Xx	Xx	49
04	03	03	49	Xx	04	49
98	04	03	98	xx	xx	49
xx	04	03	98	Xx	Xx	98
04	04	03	98	xx	xx	98
05	04	04	98	xx	05	98
xx	05	04	80	xx	xx	98
80	05	04	80	xx	xx	98
05	05	04	80	xx	xx	98
06	05	05	80	Xx	06	98
38	06	05	38	xx	xx	98
xx	06	05	38	Xx	Xx	38
2d	06	05	38	xx	xx	38
80	06	05	38	2d	Xx	38
53	06	05	38	2d	53	38

By now, the expected result (53) has appeared in the AOR and Bus!

VI. Discussions and Conclusions

The circuit level testing can only tell me whether the circuit is a good implementation of my logic. The system level testing is more important as it shows the machine's ability to solve real problems. Due to the difficulty of I/O and other things that have to be done manually, it is hard to conduct a comprehensive testing. However, by running this randomly chosen set of instructions, the CPU has shown correctness in instruction fetching and execution. Miracle happens.