[redacted]
Project 1
Due Date: 03/04/2014

Project 1: Create Your Own WIMP51 Version

Objectives:

Objectives were to create a version of WIMP51 processor in Quartus II which would include new instruction set, leaving the basic architecture of the processor intact. The instruction set assigned was CLRB A and SETB A, or clear bit in the accumulator and set bit in the accumulator, respectively.

Preliminary Analysis:

Both, CLRB A and SETB A were two byte instructions. The operational code (op-code) for CLRB A was C2 expressed in hexadecimal numbers (11000010 in binary) and SETB A was D2 in hexa [redacted] or the register states and values contained for different cycles of Wimp51: fetch, decode, and execute, during the MOV A,#dd instruction set.

	[redacted]	[redacted]	[redacted]	[redacted]	ECUTE		
IR_WE	ON	[redacted]	[redacted]	[redacted]	[redacted]	74	IR
REG_WE	OFF	[redacted]	[redacted]	[redacted]	[redacted]	xx	REG_TOP
AUX_WE	OFF	[redacted]	[redacted]	[redacted]	[redacted]	dd	AUX
PC_WE	OFF	[redacted]	[redacted]	[redacted]	[redacted]	↑	PC_ALU/PC
ACC_WE	OFF	xx	OFF	xx	ON	dd	ACC

Table 1. Register states and values for different cycles during MOV A,#dd instruction

From [redacted] was increased in the decode cycle. Thus, the [redacted] er. This memory register stored the value #dd to be move decode cycle the auxiliary register write enable logic (AU yte #dd was stored in the AUX register. In the execute cyc (ACC_WE) was in the ON state. This allowed for the , to be moved in to the accumulator. PC was again incren xt op-code.

From the on and the register states, it was obvious the new instructi ite enable logic states for the same cycles. Hence the logic B A should have been as shown in table 2.

		EXECUTE		
IR_WE	[redacted]	**OFF**	**C2**	**IR**
REG_WE	[redacted]	**OFF**	**xx**	**REG_TOP**
AUX_WE	[redacted]	**ON**	**0d**	**AUX**
PC_WE	[redacted]	**ON**	**↑**	**PC_ALU/PC**
ACC_WE	[redacted]	**ON**	**dd**	**ACC**

Table 2. cycles during CLRB A instruction

Instruction Set Modifications:

The instructions added were CLRB A, with an op-code C2 in hex or 11000010 in binary, and SETB A, with an op-code D2 in hex or 11010010 in binary. It was obvious these two were different only in bit-4. Hence, the modifications were same throughout most of the processor for both op-codes, except in the end where the chosen bit was either cleared (set to 0) or set to 1.

Instruction Register Modifications

The instruc [redacted] and in the OFF state for all c ade for IR or IR_WE logic. I This code was kept there al ing both, decode and execut

Register Top Mod

No modifi sters from R0 to R7 were not

Auxiliary Register

Auxiliary r structions. However, auxiliar op-codes. In its original ve ave been expressed as followed:

AUX_W [redacted]

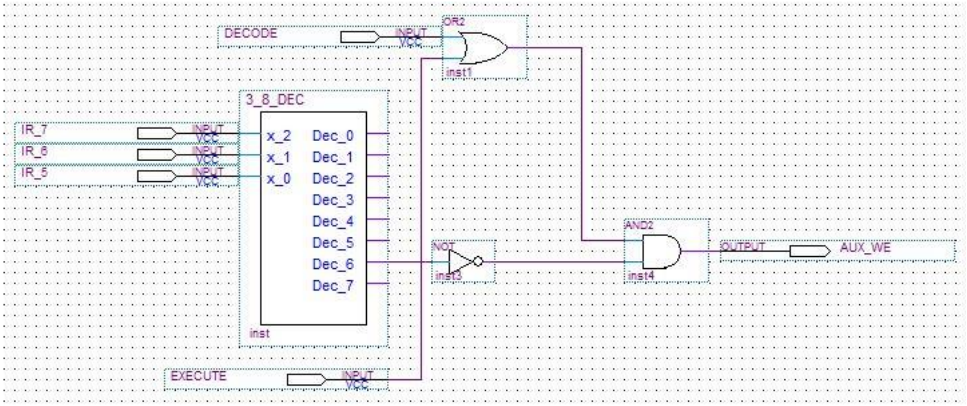


Figure 1. Original AUX_WE logic

Modified AUX_WE included C2 and D2 op-codes, where the AUX_WE was set to ON state for these tw [redacted] s changed to:

$AUX\_WE =$ [redacted] $\cdot (Decode$

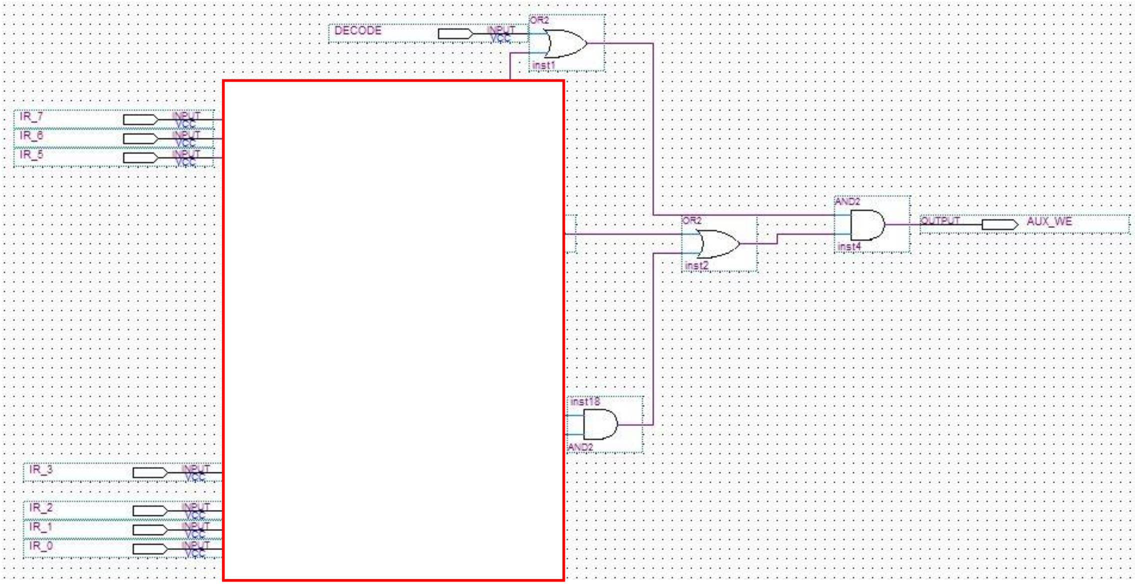

[redacted]

Figure 2. Modified AUX_WE logic

Program Counter Modifications

Since the new instructions were two byte long PC_ALU and the PC_WE had to be modified to accept new instructions. After the fetch cycle, where the op-code was stored in the IR, the PC had to be increased in the decode cycle to point at the second byte of the instruction. The second byte contained the information on the accumulator byte to be cleared or set. This

[redacted]

second byte was stored [redacted] The PC was increased again in the execute cyc .

As shown in fig e inputs to the priority encoder logic. The origi

$A = \overline{Q1}\cdot Q0 + Q1\cdot\overline{Q0}$ $\cdot\overline{IR3}\cdot\overline{IR2}\cdot\overline{IR1}\cdot\overline{IR0}$

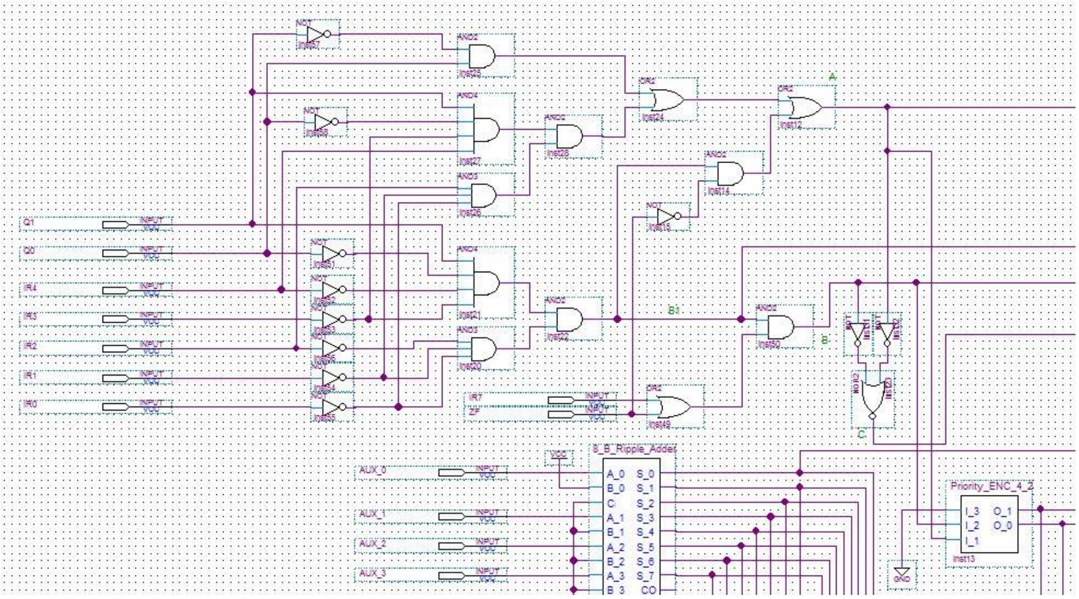
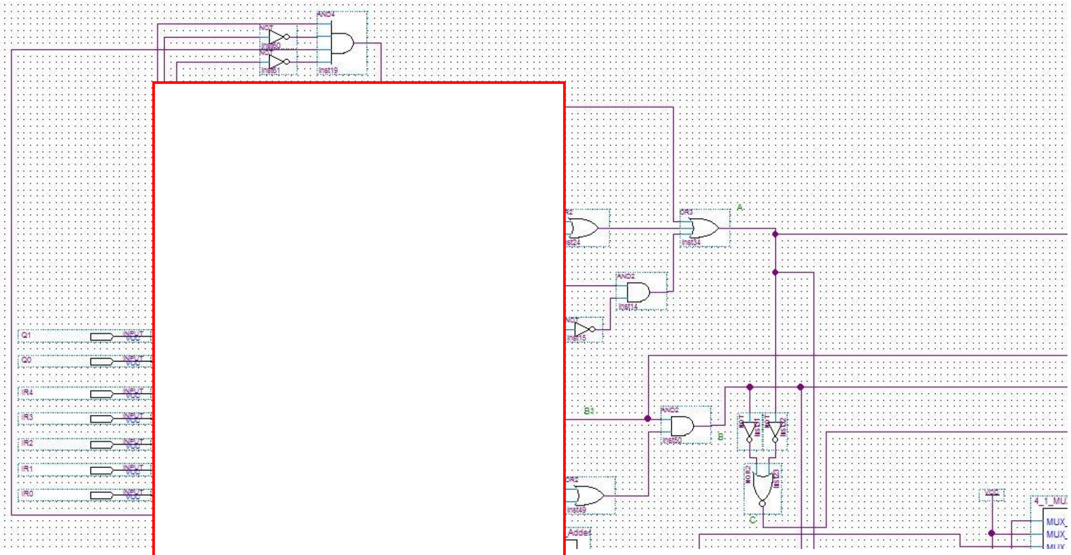


Figure 3. Section of the original PC_ALU



[redacted] the modified PC_ALU

[redacted]

The modified version of the PC_ALU was shown in figure 4. After the analysis of the equation for the original [redacted] utput was zero for the fetch cycle (since Q1=0 ut was one. During the execute cycle the output ne for the 74H (MOV) instruction. This modifica he A-section logic was changed to:

A_{modifi} $\cdot \overline{IR0}$

The modified PC_ e A-section was set to high for the execute cycle, just as for the MOV A,#dd instruction. No other modifications in the PC_ALU were necessary.

Original PC_WE logic was setup to be in the ON state for decode cycle and for only certain instructions in the execute cycle. This was shown in figure 5 (below). Modified PC_WE was setup with additional logic so it would include instructions C2 and D2 in the execute cycle. The logic added was shown below:

PC_WE_{m} [redacted] $\cdot \overline{IR0}$

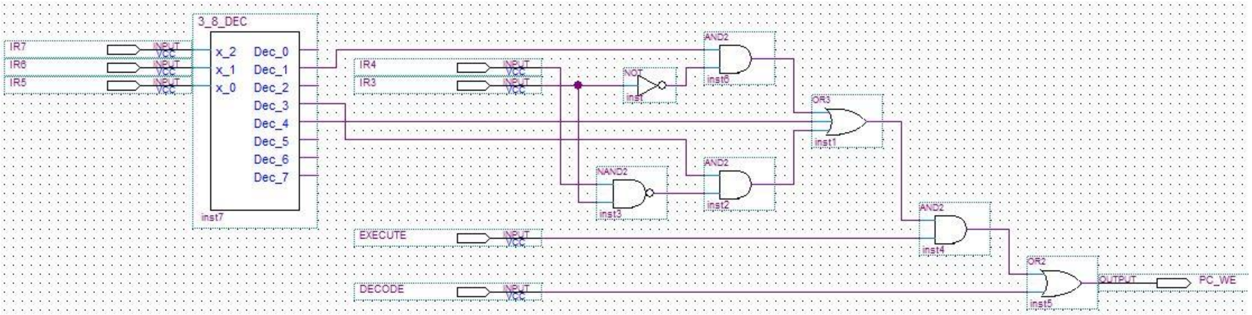


Figure 5. Original PC_WE logic

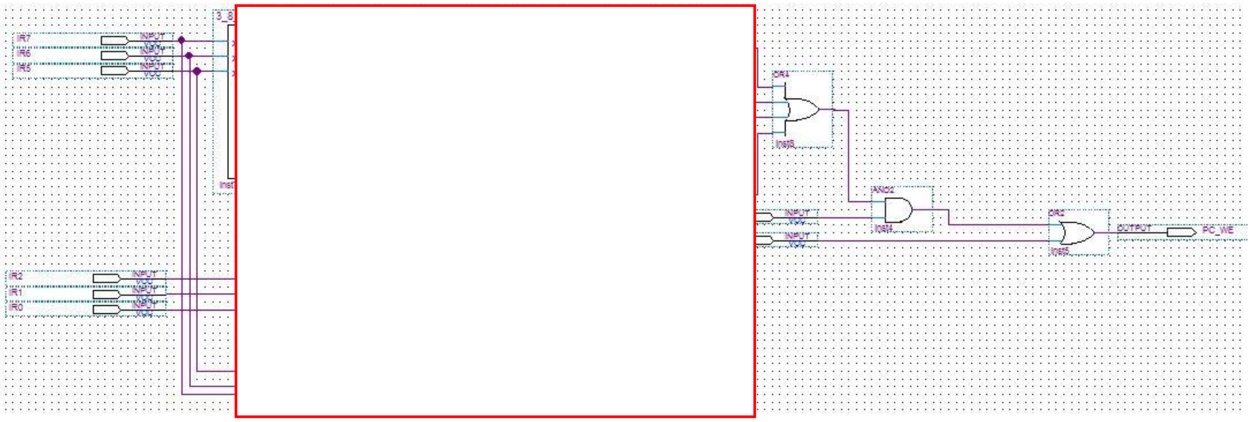

[redacted]

Figure 6. Modified PC_WE logic

Accumulator Register Modifications

There were no changes to the accumulator register. However, ACC_WE had to be modified to include the new instru [redacted] d ACC_WE logic, respectively. The l [redacted] e enable for the accumulator was in th [redacted] ot change this, but merely added two ne [redacted]

$$ACC\_WE_{modified} = ACC\_WE_{ori}$$ [redacted]

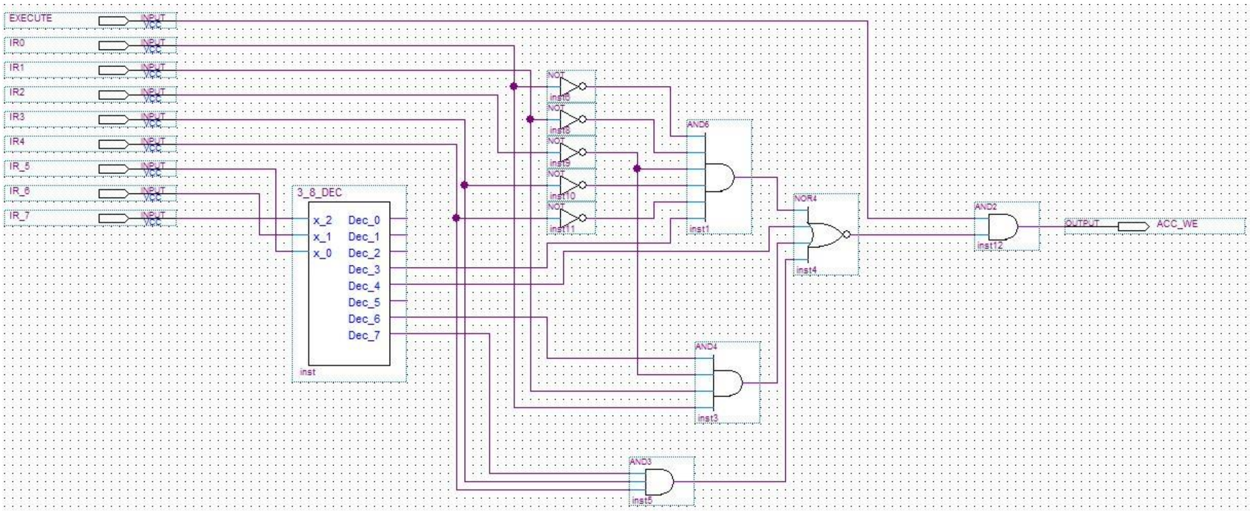


Figure 7. Original ACC_WE logic

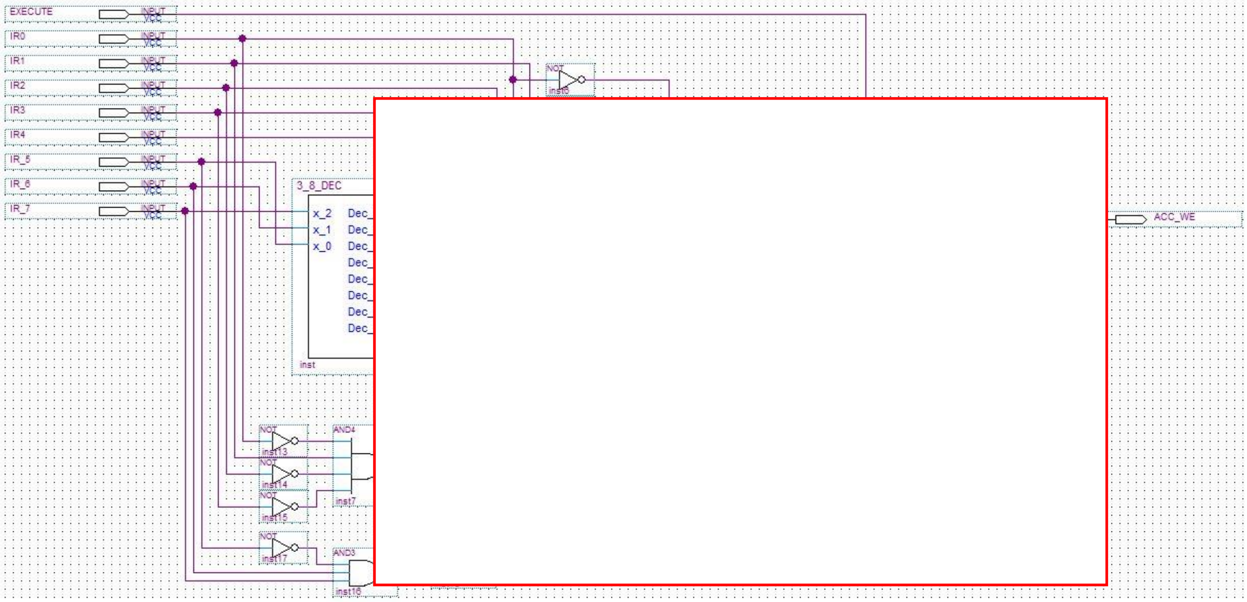

[redacted]

Figure 8. Modified ACC_WE logic

[redacted]

Arithmetic-Logic Unit Modifications

Lastly, but most importantly, ALU was modified. First, two new inputs were added to the
ALU, IR1 and IR[redacted]d in order for
instructions C2 and[redacted]LATOR logic
circuit was added,[redacted]ed to regulate
which bit was to be[redacted]s, inputs from
the accumulator we[redacted]ified. IR4 was
the only difference[redacted]scern between
the two instructions[redacted]set to zero). If
the IR4 bit was [redacted] register bits
AUX_REG0, AUX[redacted]ator was to be
modified.

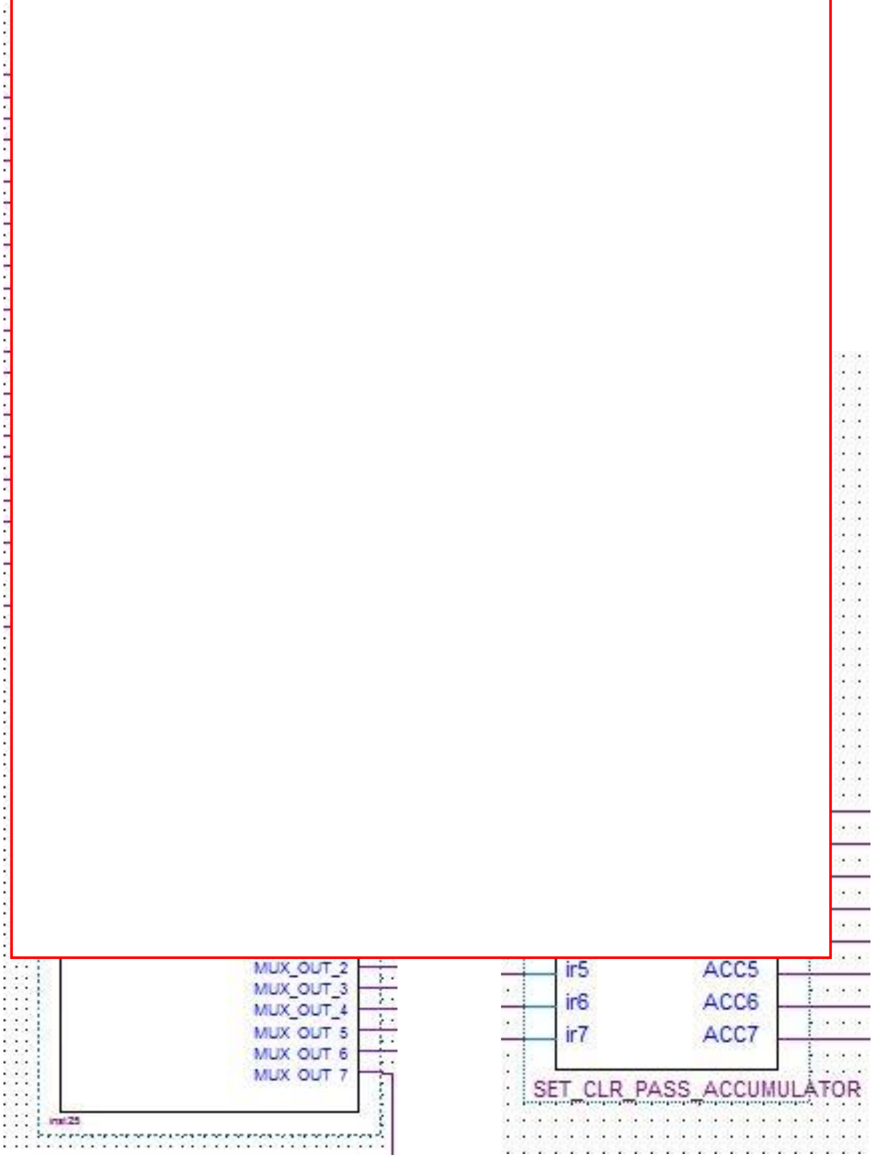
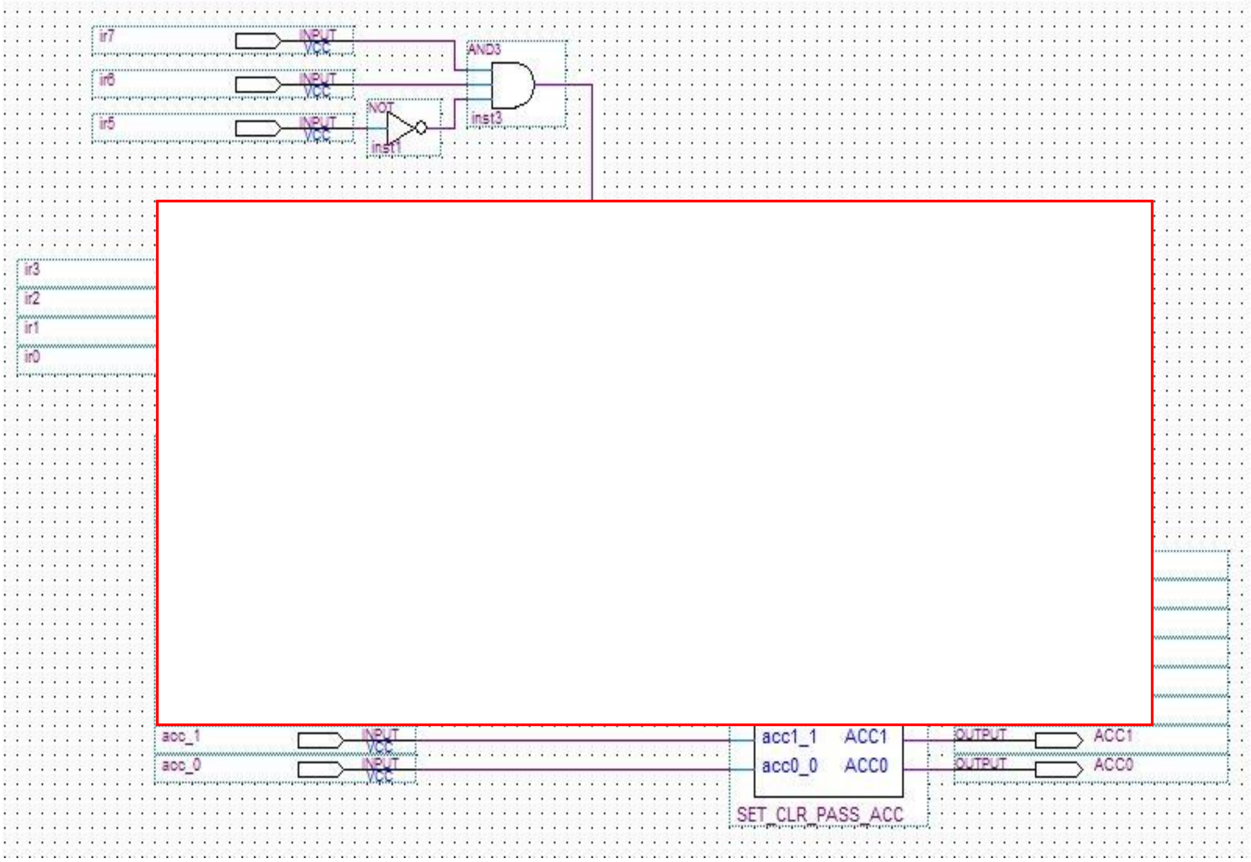


Figure 9. Modified ALU (left) and added SET_CLR_PASS_ACCUMULATOR logic (right)

Logic circuit of the new instructions was shown in figure 10. The logic was enabled if either C2 or D2 input was fed from th [redacted] ers two to zero determined which bit was changed, and IR4 dete to zero or one. In case the logic was not enabled, the input fror e just passed to the output without changes to any of the bits.

$$Enable = I \qquad \overline{IR0}$$



[redacted]

Figure 10. SET_CLR_PASS_ACCUMULATOR logic

The inside of the SET_CLR_PASS_ACC symbol file was shown in figure 11. It was noticeable the logic consisted of: accumulator inputs, 3:8-decoder, and eight 2:1 MUX logic units. The accumulator inputs were fed into the MUX units, where they were passed or changed. Decoder was used to determine the bit to be changed, by setting the input S0 of one of the MUX units to one. All the other would have input zero. Table 3 presented the logic behind the decoder. As explained before, if the enable input was set to one (using instruction register), the inputs of the auxiliary register decided which S0 input of the MUX units was set to high. Same unit changed the accumulator bit to one or zero, depending on the IR4 bit. Figure 13 showed the decoder logic. Lastly, 2:1 MUX units either passed or changed the accumulator bit. Logic was shown in figure 12 below.

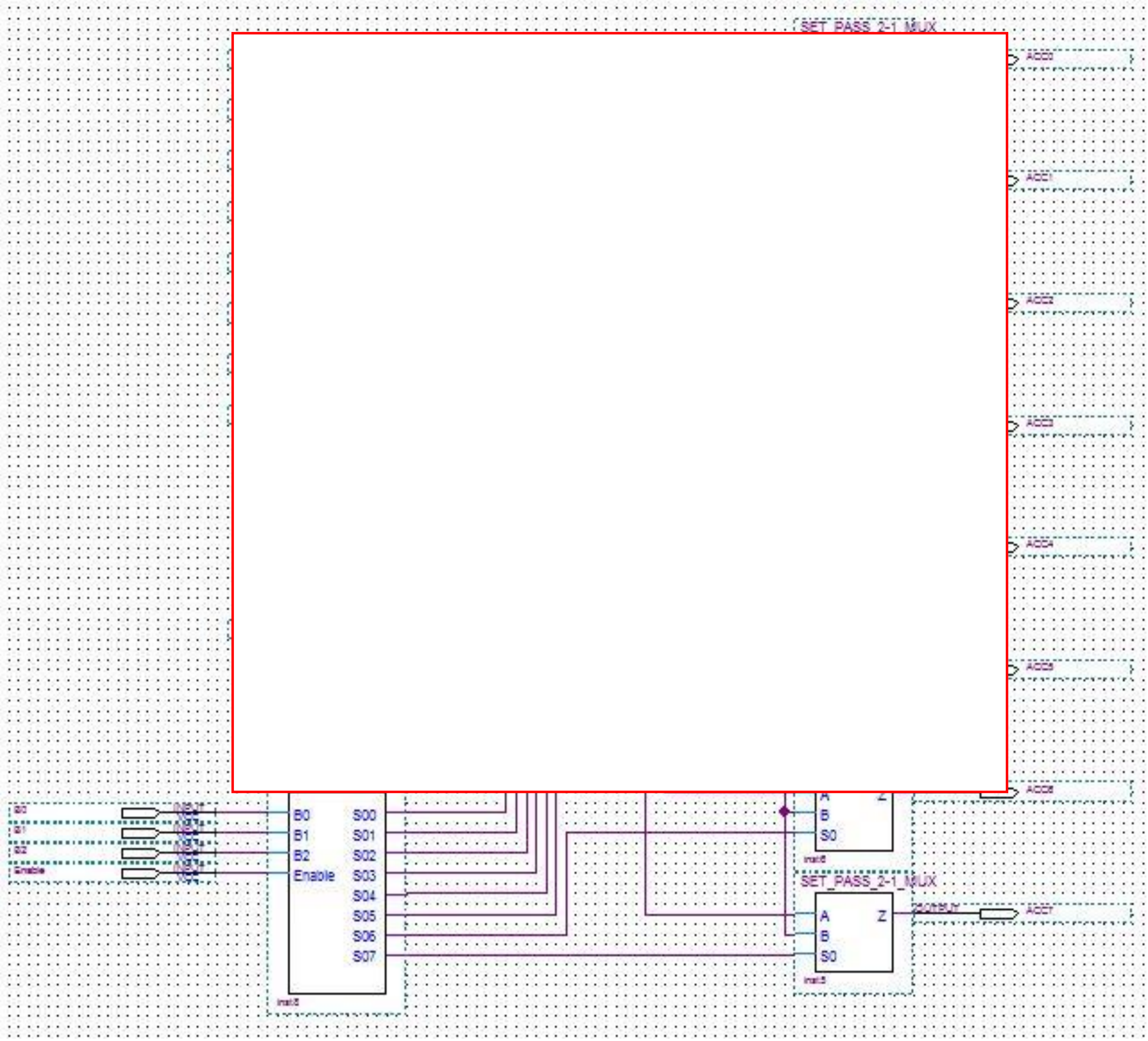

[redacted]

Figure 11. Inside the SET_CLR_PASS_ACC logic

$$Z = A \cdot \overline{S0} + B \cdot S0$$

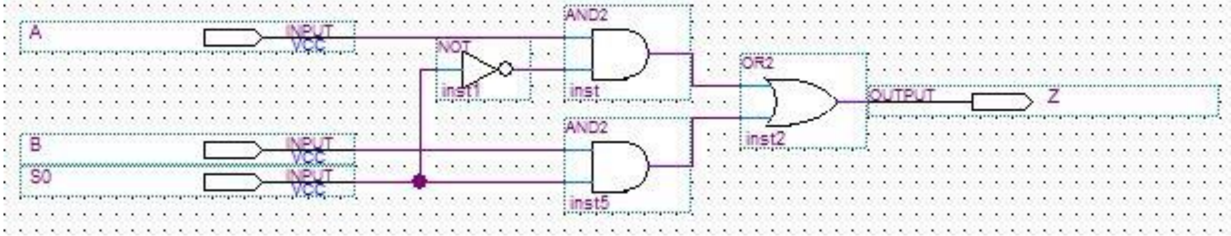


Figure 12. MUX 2:1 logic

[redacted]

CpE-213
Project 1
Due Date: 03/04/2014

Enable	A[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	S0				
							b4	b3	b2	b1	b0
0	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	0	0	0	0	0
1	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	0	0	0	0	1
1	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	0	0	0	1	0
1	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	0	0	1	0	0
1	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	0	1	0	0	0
1	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	1	0	0	0	0
1	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Table 3. Decoder 3:8 of the SET_CLR_PASS_ACC logic

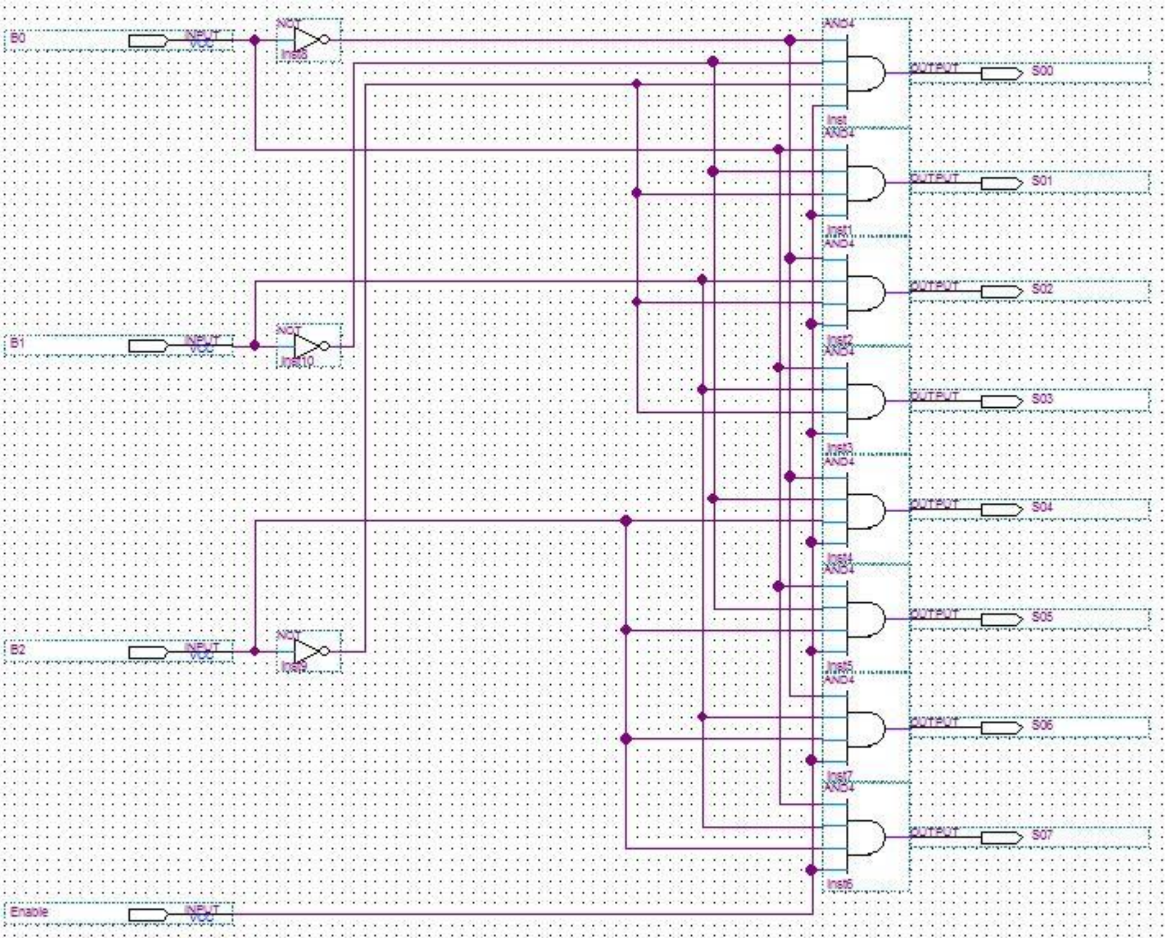


Figure 13. Decoder 3:8 logic

[redacted]

Besides already described SET_PASS_CLR_ACCUMULATOR logic circuit, two more were added to the ALU, as presented in figure 14. These were used to in order to keep the existing instructions unaffected and to prevent any overlapping of the new instructions with existing ones. Same 2:1 MUX units, shown in figure 12, were used to determine if the input from SET_PASS_CLR_ACCUMULATOR logic were used, or if the old accumulator output logic was used. This was decided using LOG_SETBA_CLRBA logic unit, shown in figure 15. Boolean logic of that circuit was shown below.

[redacted]
L_A_SEL
IR7
IR6
IR5
IR4
L1
L0
inst14
MUX
MUX
MUX
MUX

Figure 14. Inside the modified ALU

[redacted] $R1 \cdot \overline{IR0}$

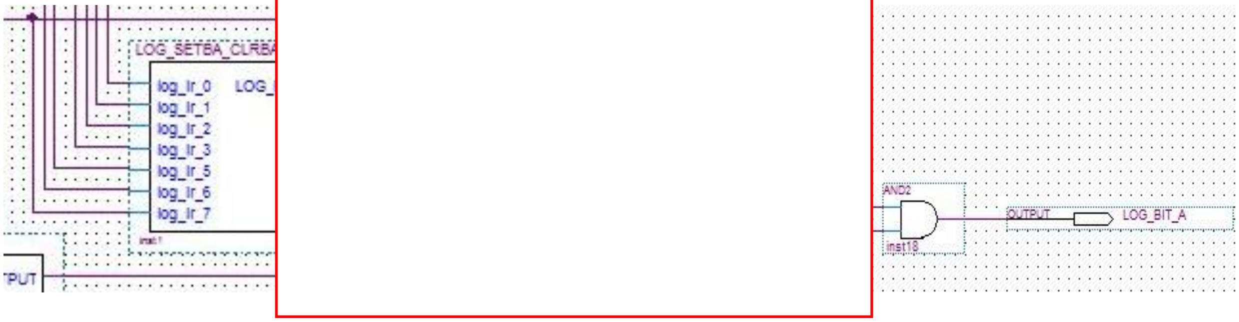

[redacted]

Figure 15. LOG_SETBA_CLRBA unit

Testing the Modified WIMP51:

Three different programs were used in testing the modified WIMP51 processor. All three tested programs have given satisfactory (expected) results. First one used was to test the CLRB A instruction. This program was used to load hex number FF into the accumulator and then clear its bits one by one. The program was given here:

CLRB A Test Program

PC	OP-CODE		
00	74	MOV A, #FF	;Store FF into accumulator, ACC=FF
01	FF		;ACC=FF
02	C2	CLRB A, #00	;Clear bit 0 of the accumulator
03	00		;ACC=FE
04	C2	CLRB A, #01	;Clear bit 1 of the accumulator
05	01		;ACC=FC
06	C2	CLRB A, #02	;Clear bit 2 of the accumulator
07	02		;ACC=F8
08	C2	CLRB A, #03	;Clear bit 3 of the accumulator
09	03		;ACC=F0
0A	C2	CLRB A, #04	;Clear bit 4 of the accumulator
0B	04		;ACC=E0
0C	C2	CLRB A, #05	;Clear bit 5 of the accumulator
0D	05		;ACC=C0
0E	C2	CLRB A, #06	;Clear bit 6 of the accumulator
0F	06		;ACC=80
10	C2	CLRB A, #07	;Clear bit 7 of the accumulator
11	07		;ACC=00
12	80	SJMP rel	;Jump back to here
13	FE		

Second testing program, shown below, was used to load hex number 00 into the accumulator and then set its bits one by one, thus ending with the accumulator value of FF. The program was given here:

SETB A Test Program

PC	OP-CODE		
00	74	MOV A, #FF	;Store 00 into accumulator, ACC=00
01	00		;ACC=00
02	D2	SETB A, #00	;Set bit 0 of the accumulator
03	00		;ACC=01
04	D2	SETB A, #01	;Set bit 1 of the accumulator
05	01		;ACC=03

06	**D2**	**SETB A, #02**	**;Set bit 2 of the accumulator**
07	**02**		**;ACC=07**
08	**D2**	**SETB A, #03**	**;Set bit 3 of the accumulator**
09	**03**		**;ACC=0F**
0A	**D2**	**SETB A, #04**	**;Set bit 4 of the accumulator**
0B	**04**		**;ACC=1F**
0C	**D2**	**SETB A, #05**	**;Set bit 5 of the accumulator**
0D	**05**		**;ACC=3F**
0E	**D2**	**SETB A, #06**	**;Set bit 6 of the accumulator**
0F	**06**		**;ACC=7F**
10	**D2**	**SETB A, #07**	**;Set bit 7 of the accumulator**
11	**07**		**;ACC=FF**
12	**80**	**SJMP rel**	**;Jump back to here**
13	**FE**		

The last test program was used to make sure no other instructions were affected by the modifications. This program was given here:

CLRBA and SETB A Test Program

PC	**OP-CODE**		
00	**74**	**MOV A, #FF**	**;Store 01 into accumulator, ACC=01**
01	**01**		**;ACC=01**
02	**F8**	**MOV R0,A**	**;R0=ACC=01**
03	**38**	**ADDC A,R0**	**;ACC=ACC+R0=02**
04	**D2**	**SETB A, #03**	**;Set bit 3 of the accumulator**
05	**03**		**;ACC=0A**
06	**F9**	**MOV R1,A**	**;R1=ACC=0A**
07	**C4**	**SWAP A**	**;Swap upper and lower and upper nibble, ACC=A0**
08	**C2**	**CLRB A, #05**	**;Clear bit 5 of the accumulator**
09	**05**		**;ACC=80**
0A	**FA**	**MOV R2,A**	**;R2=ACC=80**
0B	**80**	**SJMP rel**	**;Jump back to here**
0C	**FE**		

Conclusion:

In this project WIMP51 processor was modified and tested on the Altera board. The task was to add two more instructions, CLRB A and SETB A, which would access accumulator and clear or set one bit at the time. This was done by modifying: auxiliary register and its write enable logic, program counter ALU and the program counter write enable, accumulator write enable logic, and the ALU. These modifications were tested using three different programs, which have given expected results, thus proving the functionality of the modified WIMP51.