CS6133 Midterm

Xiaotian Bai

4/15/2017

Submission contents

- XiaotianBai_Midterm.pdf
- XiaotianBai_mips_ss_v2_part3.qar
- Other files.zip

Other files

(Also archived in qar file)

- In XiaotianBai_Part2 sub-folder
    - XiaotianBai_Part2.stp
    - XiaotianBai_Part2.jpg
    - XiaotianBai_IRAM_Part2.mif
    - XiaotianBai_DRAM_Part2.mif

- In XiaotianBai_Part3 sub-folder
    - XiaotianBai_mips_ss_v2_part3.qar
    - XiaotianBai_IRAM_Displacement.mif
    - XiaotianBai_DRAM_Displacement.mif
    - XiaotianBai_IRAM_RegIndirect.mif
    - XiaotianBai_DRAM_RegIndirect.mif
    - main_ctrl.jpg
    - mips_ss_v2.jpg

Part 1

The question is answered in 3 parts.

Part 3.1

Q:Why is the instruction and data memory separated in MIPS SS v2 CPU?
A:Separated IRAM and DRAM make it possible to fetch instruction and data simultaneously, so the components who process instructions and the components who process data can work at the same time. The performance of the CPU will be improved. What's more, the instruction won't be affected by data write operations, they won't share the memory even when data memory is fully used.

Part 3.2

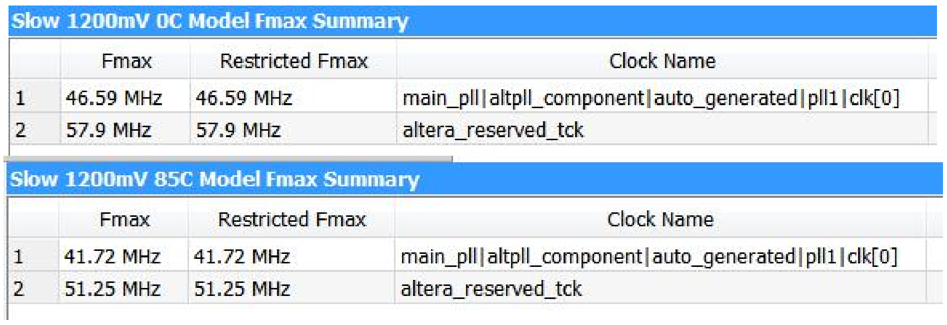
Q:Explain the advantage and disadvantage of separate instruction and data memory
A:
Advantages: 1.The CPU now has 2 memory buses instruction and data can be fetched simultaneously, so the CPU has a higher performance, less time is needed to execute a program. 2.There will be no address collision of instructions and data. We don't need to assign the address very carefully.

Disadvantages: 1.Less flexibility. When one of the two memories fill up, they can't share space with each other and external memory is needed. 2.It increases hardware complexity to split the memory.

Part 3.3

Q:What is the PLL's clock rate or frequency?
A:PLL output clock frequency=input frequency*multiplication factor/division factor. In this case, input frequency=50MHz, output frequency=50/50=1MHz.

Q:What is the highest clock rate that could be achieved without any modifications?
A:



Slow 1200mV 0C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name
1	46.59 MHz	46.59 MHz	main_pll|altpll_component|auto_generated|pll1|clk[0]
2	57.9 MHz	57.9 MHz	altera_reserved_tck

Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name
1	41.72 MHz	41.72 MHz	main_pll|altpll_component|auto_generated|pll1|clk[0]
2	51.25 MHz	51.25 MHz	altera_reserved_tck

At 0C its 46.59MHz, while at 85C its 41.72MHz.

Q:What limits the clock rate in the design?
A:1.Delay. Signals should be received and processed by next gate in a clock cycle. If the maximum delay between two gate is higher, the maximum clock rate has to be lower.
2.Temperature. Higher clock rate produces more heat. At a higher temperature the maximum clock rate has to be lower.

Part 2

The question can be solved by adding 1, 2, 3,…, 20 to a register, then subtract 20, 19, 18,…, 1from the register. In the program I wrote I used r4 to be the counter, r1 to save the addend/subtrahend. It's basically a simple loop program .

Below are IRAM and DRAM mif files of my program.

XiaotianBai_IRAM_Part2.mif

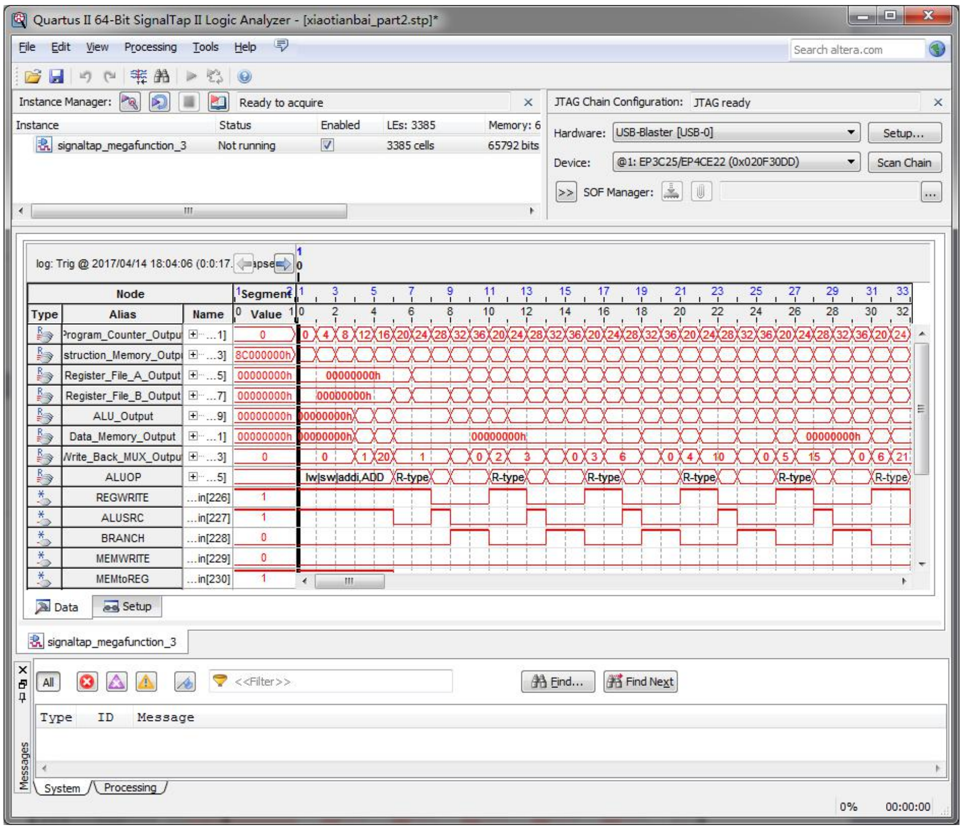
```
-- NYU:Poly CS6133 Computer Architecture
-- Instructor: Vikram Padman

-- Modified by Xiaotian Bai for the Spring 2017 Midterm
--
--
-- File format:
-- Hex Address : bit31 ...... bit0;
WIDTH=32;
DEPTH=1024;
ADDRESS_RADIX=HEX;
DATA_RADIX=BIN;
CONTENT BEGIN
----------------------------------------------------------------------
-----
-- Overall program structure:
-- r4 is our counter
-- r1 is our increment/decrement
-- r2 is our upper upwards
-- r3 is used to change r1 value, its value is 1
-- r0 is our lower limit, its value is 0
--
-- 000 lw r0,0(r0) -- Load data memory 0 -> r0
-- 001 lw r1,0(r0) -- Load data memory 0 -> r1
-- 002 lw r4,0(r0) -- Load data memory 0 -> r4
-- 003 lw r3,4(r0) -- Load data memory 1 -> r3
-- 004 lw r2,8(r0) -- Load data memory 2 -> r2
-- 005 add r1,r1,r3 -- Add 1 to r1
-- 006 add r4,r1,r4 -- Add r1 to r4
-- 007 out r4 -- Display the 8 least signifant bits through the LEDs
-- 008 beq r1,r2,00A -- Branch if r1 reaches 20
-- 009 beq r0,r0,005 -- Loop if r1 does not reach 20
-- 00A out r0 -- Set LEDs to blink
-- 00B lw r2,4(r0) -- Load data memory 1 -> r2
-- 00C sub r4,r4,r1 -- Count backwards
-- 00D out r4 -- Display the 8 least signifant bits through the LEDs
-- 00E sub r1,r1,r2 -- Subtract 1 from r1
-- 00F beq r1,r0,011 -- Branch if r1 reaches 0
-- 010 beq r0,r0,00C -- Loop if r1 does not reach 0
-- 011 out r0 -- Set LEDs to blink
----------------------------------------------------------------------
--Hex Address : bit31...........bit15..........bit0;
--    |            |               |              |
    000     :    10001100000000000000000000000000;
--               |____||___||___||______________|
--                 |    |    |          |
--               op=lw, rs=0,rt=0,   offset=0
-- lw r0,0(r0) -- Load data memory 0 -> r0
--
    001     :    10001100000000010000000000000000;
--               |____||___||___||______________|
--                 |     |    |           |
--               op=lw, rs=0,rt=1,   offset=0
-- lw r1,0(r0) -- Load data memory 0 -> r1
--
    002     :    10001100000001000000000000000000;
--               |____||___||___||______________|
--                 |     |    |           |
--               op=lw, rs=0,rt=4,   offset=0
-- lw r4,0(r0) -- Load data memory 0 -> r4
--
    003     :    10001100000000110000000000000100;
--               |____||___||___||______________|
--                 |     |    |           |
--               op=lw, rs=0,rt=3,   offset=4
-- lw r3,4(r0) -- Load data memory 1 -> r3
--
    004     :    10001100000000100000000000001000;
--               |____||___||___||______________|
--                 |     |    |           |
--               op=lw, rs=0,rt=2,   offset=8
-- lw r2,8(r0) -- Load data memory 2 -> r2
--
    005     :    00000000001000110000100000100000;
--               |____||___||___||___||___||____|
--                 |     |    |    |    |    |
--               R-type,rs=1,rt=2,rd=3,---,f=add
-- add r1,r1,r3 -- Add 1 to r1
--
    006     :    00000000001001000010000000100000;
--               |____||___||___||___||___||____|
--                 |     |    |    |    |    |
--               R-type,rs=1,rt=4,rd=4,---,f=add
-- add r4,r1,r4 -- Add r1 to r4
--
    007     :    10110000100000000000000000000000;
--               |____||___||___||______________|
--                 |     |    |           |
--               op=out, rs=4,rt=0,   offset=0
-- out r4 -- Display the 8 least signifant bits through the LEDs
--
    008     :    00010000001000100000000000000001;
--               |____||___||___||______________|
--                 |     |    |           |
--             op=beq, rs=1,rt=2,   offset=1
-- beq r1,r2,00A -- Branch if r1 reaches 20
--
    009     :    00010000000000001111111111111011;
--               |____||___||___||______________|
--                 |     |    |           |
--             op=beq, rs=0,rt=0,   offset=-5
-- beq r0,r0,005 -- Loop if r1 does not reach 20
--
    00A     :    10110000000000000000000000000000;
--               |____||___||___||______________|
--                 |     |    |           |
--              op=out, rs=0,rt=0,   offset=0
-- out r0 -- Set LEDs to blink
```

```
--
    00B     :    10001100000000100000000000000100;
--               |____||___||___||______________|
--                  |    |    |           |
--               op=lw, rs=0,rt=2,   offset=4
-- lw r2,4(r0) -- Load data memory 1 -> r2
--
    00C     :    00000000100000010010000000100010;
--               |____||___||___||___||___||____|
--                  |    |    |    |     |    |
--               R-type,rs=4,rt=1,rd=4,---,f=sub
-- sub r4,r4,r1 -- Count backwards
--
    00D     :    10110000100000000000000000000000;
--               |____||___||___||______________|
--                  |    |    |           |
--              op=out, rs=4,rt=0,   offset=0
-- out r4 -- Display the 8 least signifant bits through the LEDs
--
    00E     :    00000000001000100000100000100010;
--               |____||___||___||___||___||____|
--                  |    |    |    |     |    |
--               R-type,rs=1,rt=2,rd=1,---,f=sub
-- sub r1,r1,r2 -- Subtract 1 from r1
--
    00F     :    00010000001000000000000000000001;
--               |____||___||___||______________|
--                  |    |    |           |
--             op=beq, rs=1,rt=0,   offset=1
--  beq r1,r0,011 -- Branch if r1 reaches 0
--
    010     :    00010000000000001111111111111011;
--               |____||___||___||______________|
--                  |    |    |           |
--               op=beq, rs=0,rt=0,   offset=-5
--  beq r0,r0,00C -- Loop if r1 does not reach 0
--
    011     :    10110000000000000000000000000000;
--               |____||___||___||______________|
--                  |    |    |           |
--              op=out, rs=0,rt=0,   offset=0
-- out r0 -- Set LEDs to blink
END;
```

XiaotianBai_DRAM_Part2.mif

```
-- NYU:Poly CS6133 Computer Architecture
-- Instructor: Vikram Padman
--  Xiaotian Bai CS6133 Midterm

WIDTH=32;
DEPTH=1024;

ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;

CONTENT BEGIN

---------------------------------------------------
--  Variables used in the program
--  000 The value 0
--  001 The value 1
--  002 The value to count up to (20)
--------------------------------------------------
000  : 00000000;
001  : 00000001;
002  : 00000014;
END;
--------------------------------------------------
```

After running the program, I got the signal capture in SignalTap II Logic Analyzer:



Quartus II 64-Bit SignalTap II Logic Analyzer - [xiaotianbai_part2.stp]*

Instance Manager: Ready to acquire

Instance	Status	Enabled	LEs: 3385	Memory: 6
signaltap_megafunction_3	Not running	✓	3385 cells	65792 bits

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0]

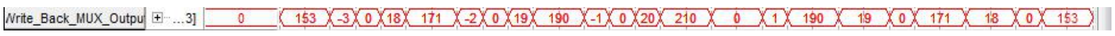
Device: @1: EP3C25/EP4CE22 (0x020F30DD)

log: Trig @ 2017/04/14 18:04:06 (0:0:17.

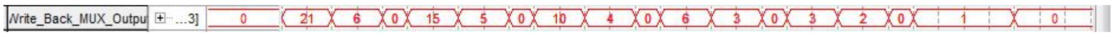
Alias	Name	Value	Waveform
Program_Counter_Output	...1]	0	0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 20, 24, 28, 32, 36, 20, 24, 28, 32, 36, 20, 24, 28, 32, 36, 20, 24, 28, 32, 36, 20, 24
Instruction_Memory_Output	...3]	8C000000h	
Register_File_A_Output	...5]	00000000h	00000000h
Register_File_B_Output	...7]	00000000h	00000000h
ALU_Output	...9]	00000000h	00000000h
Data_Memory_Output	...1]	00000000h	00000000h, 00000000h, 00000000h
Write_Back_MUX_Output	...3]	0	0, 1, 20, 1, 0, 2, 3, 0, 3, 6, 0, 4, 10, 0, 5, 15, 0, 6, 21
ALUOP	...5]		lw, sw, addi, ADD, R-type, R-type, R-type, R-type, R-type, R-type
REGWRITE	...in[226]	1	
ALUSRC	...in[227]	1	
BRANCH	...in[228]	0	
MEMWRITE	...in[229]	0	
MEMtoREG	...in[230]	1	

Type ID Message

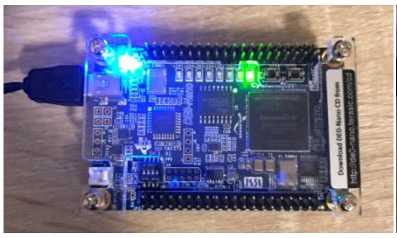
Midway of count:

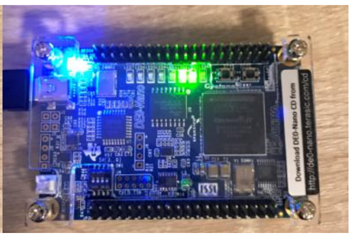

Write_Back_MUX_Output ...3] 0 | 153, -3, 0, 18, 171, -2, 0, 19, 190, -1, 0, 20, 210, 0, 1, 190, 19, 0, 171, 18, 0, 153

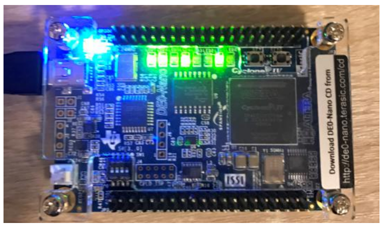
End of count:


Write_Back_MUX_Output ...3] 0 | 21, 6, 0, 15, 5, 0, 10, 4, 0, 6, 3, 0, 3, 2, 0, 1, 0

As shown in the screenshots of SignalTapII Logic Analyzer above, my program counted to the maximum of Triangle number 210(n=20), sent out a 0 signal to make the LEDs blink and then counted backwards to 0.







The photos above show thatthe LEDs display decimal Triangle numbers 1, 3, 6…210(b’11010010’) in binary format.

The associated .mif files “XiaotianBai_IRAM_Part2.mif” and “XiaotianBai_DRAM_Part2.mif” as well as the SignalTapII capture file “XiaotianBai_Part2.stp” and its diagram“XiaotianBai_Part2.jpg” can be found in the sub-folder “XiaotianBai_Part2”.

Part 3

The aim of this part is to add displacement and register indirect addressing modes to the mips_ss_v2 CPU. First I will show my theoretical design and implementation of design for each of the two addressing modes, then I will put them together to form the final design.

Part 3.1 Displacement Addressing

The purpose of displacement addressing is to enable the CPU to process the instruction “add rt, offset(rs)”, that is, rt+=M[offset+rs], here M[offset+rs] refers to contents of DRAM whose address is the offset plus the contents of rs.
This instruction should consist of 4 parts: Opcode, rs index, rt index and offset and it is an I-type instruction with 32-bit fixed format. The format of the instruction is

Opcode (31:26)	rs (25:21)	rt (20:16)	offset (15:0)

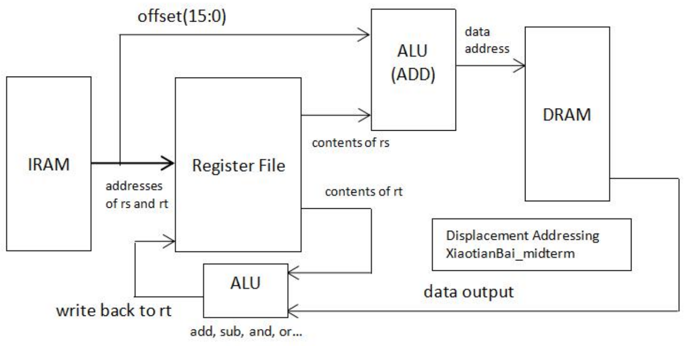
I designed 5 basic instructions for displacement addressing, listed below.

Instruction	Function
add_dis	rt=rt+M[rs+offset]
sub_dis	rt=M[rs+offset]-rt
and_dis	rt=rt AND M[rs+offset]
or_dis	rt=rt OR M[rs+offset]
out_dis	OUT M[rs+offset]

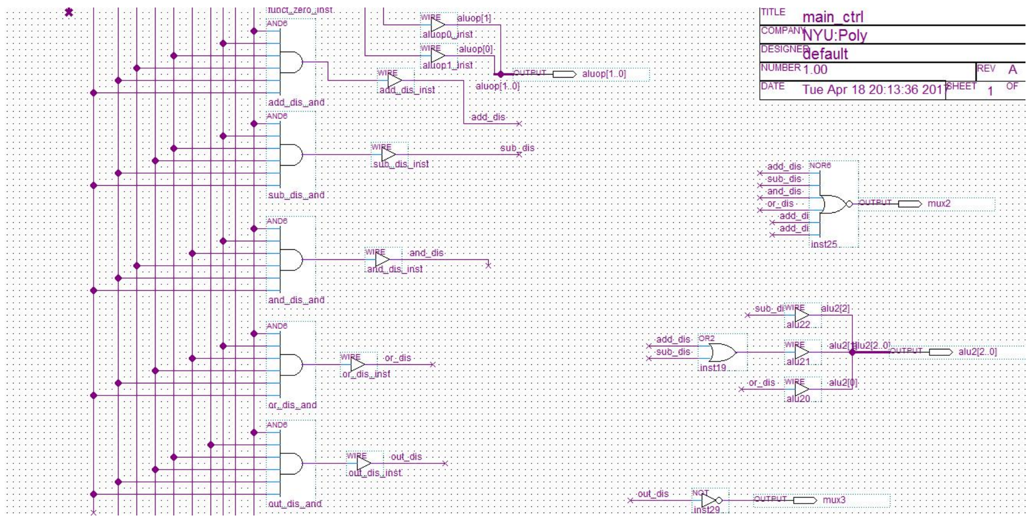
The Opcode should be different from those of any other existing instructions. In the form below all displacement instructions and their corresponding signal values are listed.

Instruction	Opcode	Regdst	Memtoreg	Memread	Memwright	Aluop	Alusrc	Regwrite	ALU2op
add_dis	110000	0	1	1	0	00	1	1	010
sub_dis	110100	0	1	1	0	00	1	1	110
and_dis	111000	0	1	1	0	00	1	1	000
or_dis	111100	0	1	1	0	00	1	1	001
out_dis	100100	0	0	1	0	00	1	0	outen=1

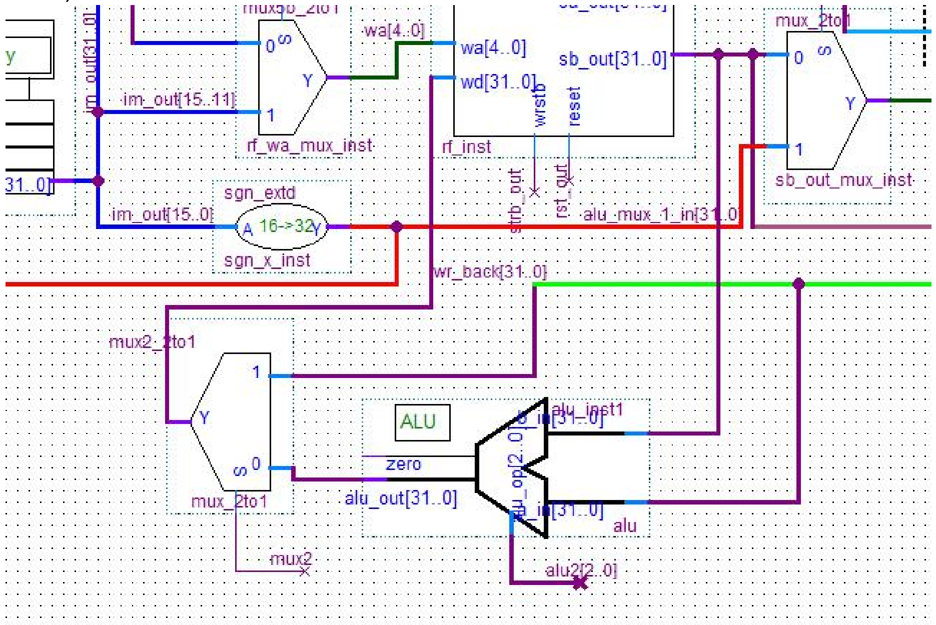
Below is the flow chart of the CPU processing the displacement addressing instruction.



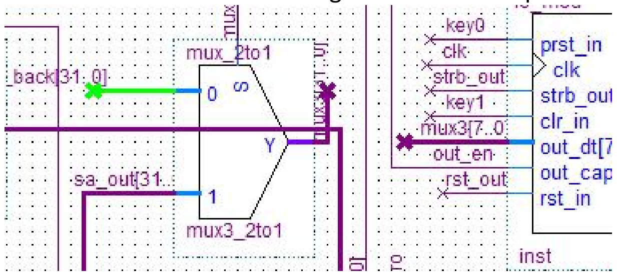
To implement this design, a new ALU is needed. Also, we need some more muxes to reuse the pins exclusively for displacement addressing without collision with other addressing modes. The Opcode I assign will control the ALUs and muxes and let them function in the expected way.
The hardware modification is shown as following screenshots. In the design of displacement addressing, I added an ALU and 2 2-to-1 muxes.



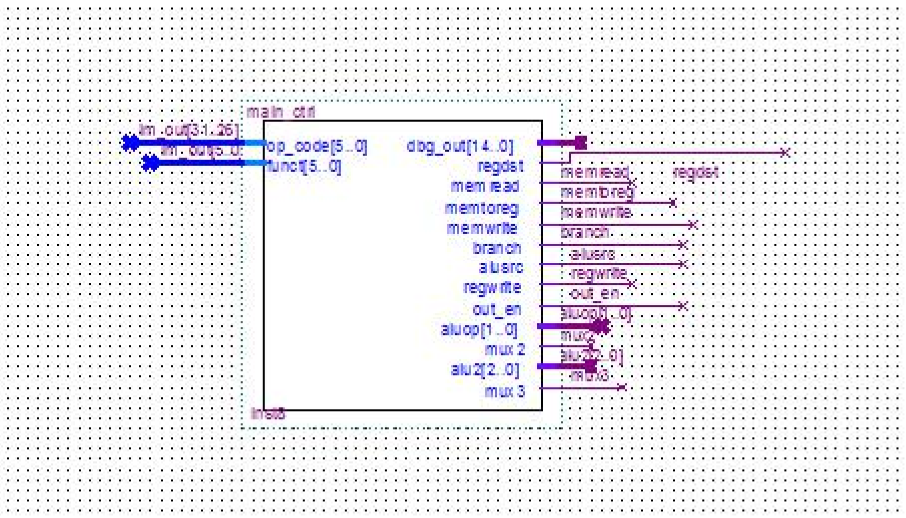
As shown the above screenshot, the new Opcodes generate their instruction signals and these signals controls ALU2, mux2 and mux3.



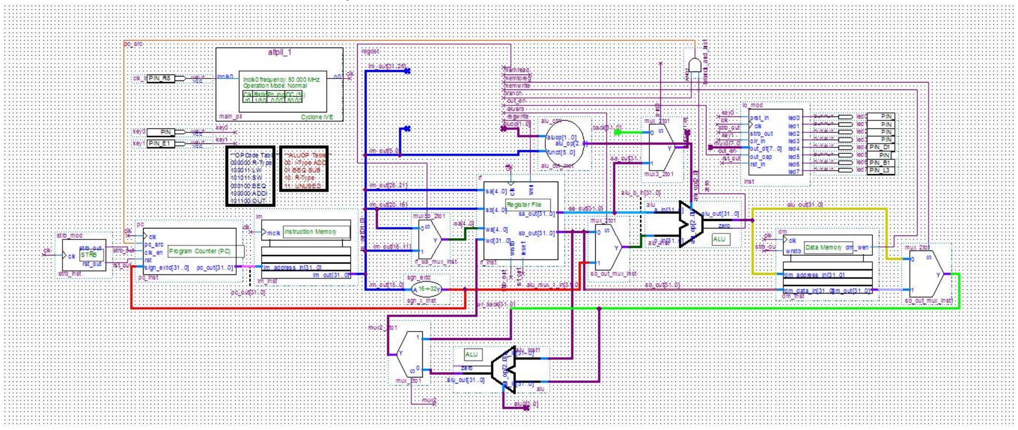
Here we can see ALU2 processes the data from DRAM and Register File and mux2 decides whether the ALU output should be wright back to registers or not. Mux2 makes function of original instructions intact, it's like an isolator who lets the new addressing mode work independently.



Mux3 is solely used for OUT instructions, with its help we can output the data directly from DRAM, makes contents in registers unchanged e.g. out_dis 10(r1), the output is M[r1+10], and the contents of r1 does not need to be changed.



Finally the main control was updated. It became larger and cannot fit the position so I dragged it away and used net names to connect the pins.



After testing for multiple times, all functions of the 5 instructions were testified.
Below is the mif files I used to test all the 5 instructions. The numbers are arbitrary but the numbers used as pointers must be addresses of other numbers.

XiaotianBai_IRAM_Displacement.mif

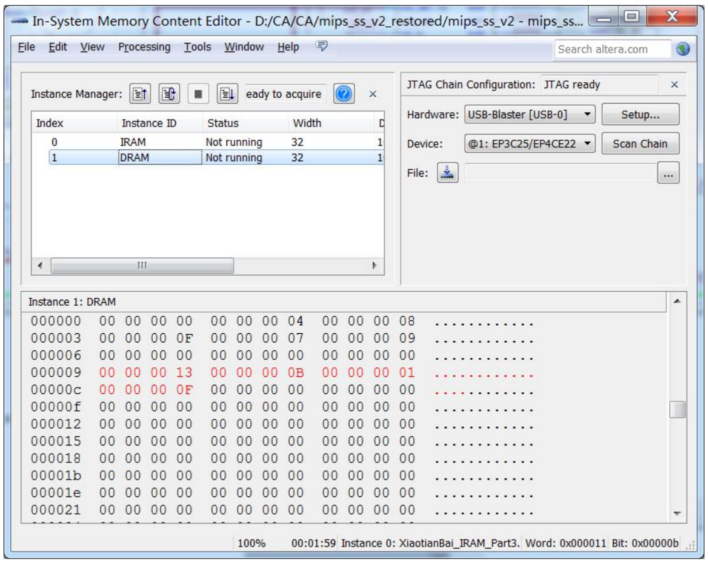
```
-- NYU:Poly CS6133 Computer Architecture
-- Instructor: Vikram Padman

-- Modified by Xiaotian Bai for the Spring 2017 Midterm
-- The codes include 5 instructions of displacement addressing
-- Namely, add_dis, sub_dis, and_dis, or_dis and out_dis
--
-- File format:
-- Hex Address : bit31 ...... bit0;
WIDTH=32;
DEPTH=1024;
ADDRESS_RADIX=HEX;
DATA_RADIX=BIN;
CONTENT BEGIN
---------------------------------------------------------------------
-----
-- Overall program structure:
-- Row 000 to 004 to test add_dis in which row 002 is the instruction
-- Row 005 to 009 to test sub_dis in which row 007 is the instruction
-- Row 00A to 00E to test and_dis in which row 00C is the instruction
-- Row 00F to 013 to test or_dis in which row 011 is the instruction
-- Row 014 to 015 to test out_dis in which row 015 is the instruction
--------------------------------------------------------------------

-- Row 000 to 004 to test add_dis
--
--Hex Address : bit31...........bit15..........bit0;
--   |           |               |              |
    000     :    10001100000000100000000000001000;
--               |____||___||___||______________|
--                 |     |    |          |
--               op=lw, rs=0,rt=2,   offset=8
-- lw r2,8(r0) -- Load data memory 2 -> r2
--
    001     :    10001100000000010000000000000100;
--               |____||___||___||______________|
--                 |     |    |          |
--               op=lw, rs=0,rt=1,   offset=4
-- lw r1,4(r0) -- Load data memory 1 -> r1
--
    002     :    11000000010000010000000000000100;
--               |____||___||___||______________|
--                 |     |    |          |
--         op=add_dis, rs=2,rt=1,   offset=4
-- add_dis(r1,r2) -- r1=r1+M[offset(r2)]
--
    003     :    10110000001000000000000000000000;
--               |____||___||___||______________|
--                 |     |    |          |
--               op=out, rs=1,rt=0,   offset=0
-- out r1 -- Display the result on the LEDs
--
    004     :    10101100000000010000000000100100;
--               |____||___||___||______________|
--                 |     |    |          |
--               op=sw, rs=0,rt=1,   offset=36
-- sw r1,36(r0) -- Save r1 ->  data memory 9
--
-- Row 005 to 009 to test sub_dis
--
    005     :    10001100000001000000000000001000;
--               |____||___||___||______________|
--                 |     |    |          |
--               op=lw, rs=0,rt=4,   offset=8
-- lw r4,8(r0) -- Load data memory 2 -> r4
--
    006     :    10001100000000110000000000000100;
--               |____||___||___||______________|
--                 |     |    |          |
--               op=lw, rs=0,rt=3,   offset=4
-- lw r3,4(r0) -- Load data memory 1 -> r3
--
    007     :    11010000100000110000000000000100;
--               |____||___||___||______________|
--                 |     |    |          |
--         op=sub_dis, rs=4,rt=3,   offset=4
-- sub_dis(r4,r3) -- r3=M[offset(r4)]-r3
--
    008     :    10110000011000000000000000000000;
--               |____||___||___||______________|
--                 |     |    |          |
--              op=out, rs=3,rt=0,   offset=0
-- out r3 -- Display the result on the LEDs
--
    009     :    10101100000000110000000000101000;
--               |____||___||___||______________|
--                  |    |    |          |
--               op=sw, rs=0,rt=3,   offset=40
-- sw r3,40(r0) -- Save r3 ->  data memory 10
--
-- Row 00A to 00E to test and_dis
--
    00A     :    10001100000001100000000000001000;
--               |____||___||___||______________|
--                  |    |    |          |
--               op=lw, rs=0,rt=5,   offset=8
-- lw r5,8(r0) -- Load data memory 2 -> r5
--
    00B     :    10001100000001010000000000010100;
--               |____||___||___||______________|
--                  |    |    |          |
--               op=lw, rs=0,rt=6,   offset=20
-- lw r6,20(r0) -- Load data memory 4 -> r6
--
    00C     :    11100000110001010000000000001000;
--               |____||___||___||______________|
--                  |    |    |          |
--         op=and_dis, rs=6,rt=5,   offset=8
-- and_dis(r6,r5) -- r5=r5 and M[offset(r6)]
--
    00D     :    10110000101000000000000000000000;
--               |____||___||___||______________|
--                  |    |    |          |
--              op=out, rs=5,rt=0,   offset=0
-- out r5 -- Display the result on the LEDs
--
    00E     :    10101100000001010000000000101100;
--               |____||___||___||______________|
--                  |    |    |          |
--               op=sw, rs=0,rt=5,   offset=44
-- sw r5,44(r0) -- Save r5 ->  data memory 11
--
-- Row 00F to 013 to test or_dis
--
    00F     :    10001100000010000000000000001000;
--               |____||___||___||______________|
--                  |    |    |          |
--               op=lw, rs=0,rt=8,   offset=8
-- lw r8,8(r0) -- Load data memory 2 -> r8
--
    010     :    10001100000001110000000000010100;
--               |____||___||___||______________|
--                  |    |    |          |
--               op=lw, rs=0,rt=7,   offset=20
-- lw r7,20(r0) -- Load data memory 5 -> r7
--
    011     :    11110001000001110000000000001000;
--               |____||___||___||______________|
--                  |    |    |          |
--           op=or_dis, rs=8,rt=7,   offset=8
-- or_dis(r8,r7) -- r7=r7 or M[offset(r8)]
--
    012     :    10110000111000000000000000000000;
```

```
--               |____||___||___||______________|
--                 |     |     |           |
--              op=out, rs=7,rt=0,   offset=0
-- out r7 -- Display the result on the LEDs
--
    013     :    10101100000001110000000000110000;
--               |____||___||___||______________|
--                 |     |     |           |
--               op=sw, rs=0,rt=7,   offset=48
-- sw r7,48(r0) -- Save r7 ->  data memory 12
--
-- Row 014 to 015 to test out_dis
--
    014     :    10001100000010010000000000001000;
--               |____||___||___||______________|
--                 |     |     |           |
--               op=lw, rs=0,rt=9,   offset=8
-- lw r9,8(r0) -- Load data memory 2 -> r9
--
    015     :    10010001001000000000000000001000;
--               |____||___||___||______________|
--                 |     |     |           |
--          op=out_dis, rs=9,rt=0,   offset=8
-- out_dis 8(r9) -- Display M[8(r9)] on the LEDs
End;
```

XiaotianBai_DRAM_Displacement.mif

```
-- NYU:Poly CS6133 Computer Architecture
-- Instructor: Vikram Padman
--  Xiaotian Bai CS6133 Midterm

WIDTH=32;
DEPTH=1024;

ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;

CONTENT BEGIN

-------------------------------------------------
--  Variables used in the program
--  000 The value 0
--  001 and 002 are divisible by 4 for addressing
--  003 The value F
--  004 and 005 are used for and_dis, or_dis and out_dis
-------------------------------------------------
000 : 00000000;
001 : 00000004;
002 : 00000008;
003 : 0000000F;
004 : 00000007;
005 : 00000009;
END;
-------------------------------------------------
```

The results are saved in DRAM:


In-System Memory Content Editor - D:/CA/CA/mips_ss_v2_restored/mips_ss_v2 - mips_ss...
File Edit View Processing Tools Window Help
Search altera.com
Setup...
Scan Chain

Instance Manager: eady to acquire

Index	Instance ID	Status	Width
0	IRAM	Not running	32
1	DRAM	Not running	32

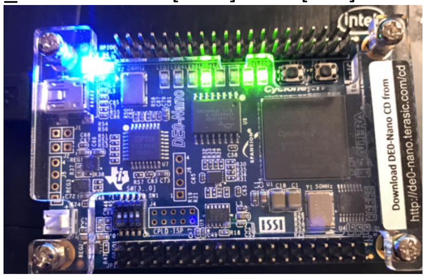
JTAG Chain Configuration: JTAG ready
Hardware: USB-Blaster [USB-0]
Device: @1: EP3C25/EP4CE22
File:

Instance 1: DRAM

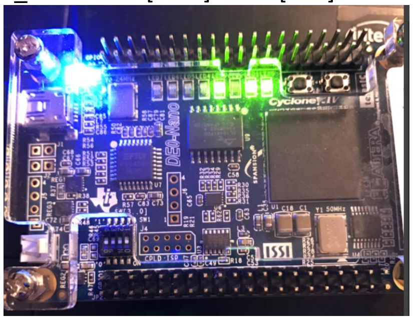
```
000000  00 00 00 00  00 00 00 04  00 00 00 08  ............
000003  00 00 00 0F  00 00 00 07  00 00 00 09  ............
000006  00 00 00 00  00 00 00 00  00 00 00 00  ............
000009  00 00 00 13  00 00 00 0B  00 00 00 01  ............
00000c  00 00 00 0F  00 00 00 00  00 00 00 00  ............
00000f  00 00 00 00  00 00 00 00  00 00 00 00  ............
000012  00 00 00 00  00 00 00 00  00 00 00 00  ............
000015  00 00 00 00  00 00 00 00  00 00 00 00  ............
000018  00 00 00 00  00 00 00 00  00 00 00 00  ............
00001b  00 00 00 00  00 00 00 00  00 00 00 00  ............
00001e  00 00 00 00  00 00 00 00  00 00 00 00  ............
000021  00 00 00 00  00 00 00 00  00 00 00 00  ............
```

100% 00:01:59 Instance 0: XiaotianBai_IRAM_Part3. Word: 0x000011 Bit: 0x00000b

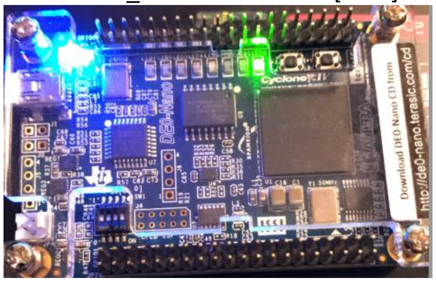
The outcome of add_dis is r1=r1+M[r2+4]=4+M[8+4]=4+F=13(Hex)=10011(Bin).



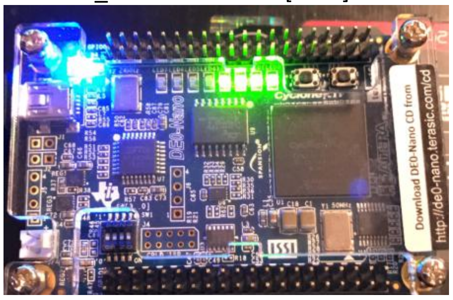
The outcome of sub_dis is r3=M[r4+4]-r3=M[8+4]-4=F-4=B (Hex)=1011(Bin).



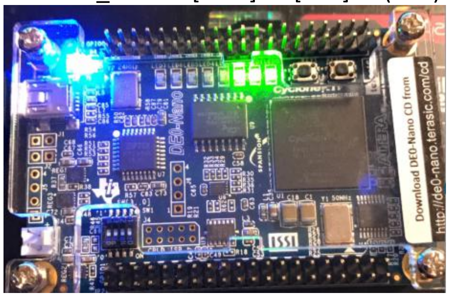
The outcome of and_dis is r5=r5 and M[r6+8]=9 AND 7=1.



The outcome of or_dis is r7=r7 or M[r8+8]=9 OR 7=1111 (Bin).



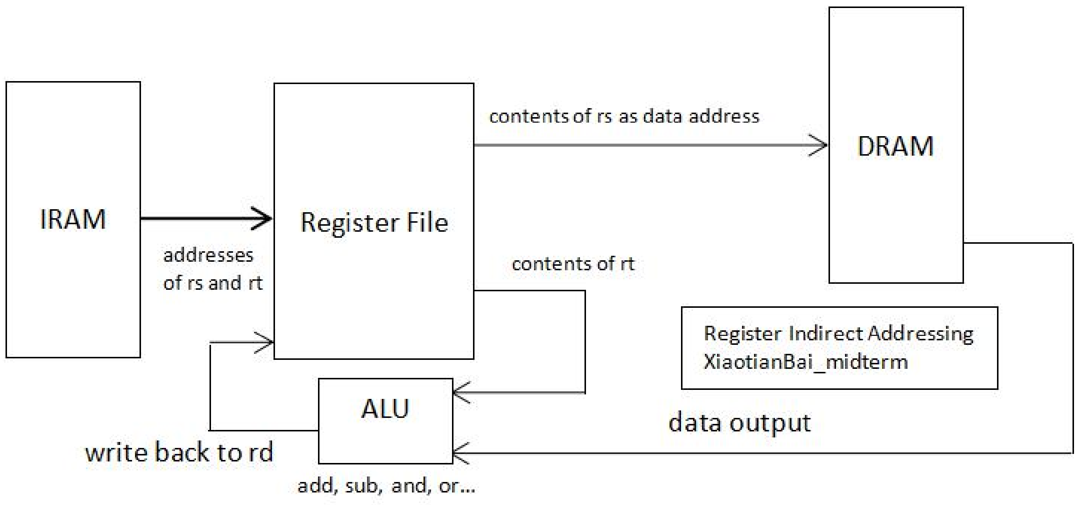
The outcome of out_dis is M[8+r9]=M[8+8]=7 (Hex)=0111(Bin) .



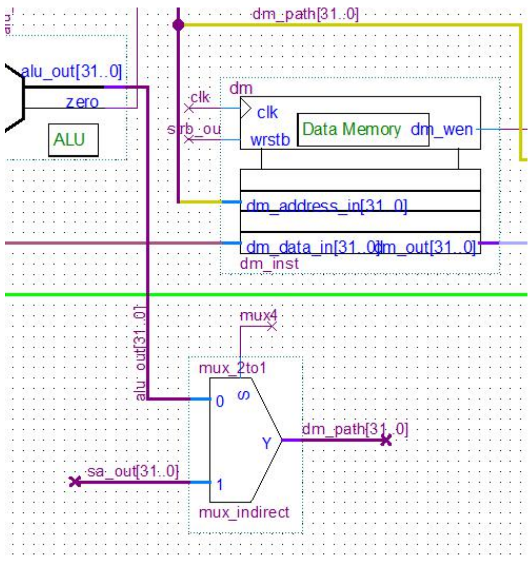
The associated .mif files "XiaotianBai_IRAM_Displacement.mif" and "XiaotianBai_DRAM_Displacement.mif" can be found in the sub-folder "XiaotianBai_Part3".

Part 3.2 Register Indirect Addressing

For register indirect addressing we have a similar flow chart for register indirect addressing. The data address is no longer processed by ALU, instead, it's sent from Register File directly to DRAM.



It seems that we need to build a path from Register File to DRAM.



Here I added another mux to make it possible to send sa_out(contents of rs) directly to DRAM as data address(dm_path) when we are using register indirect addressing.

The instruction format is R-type.

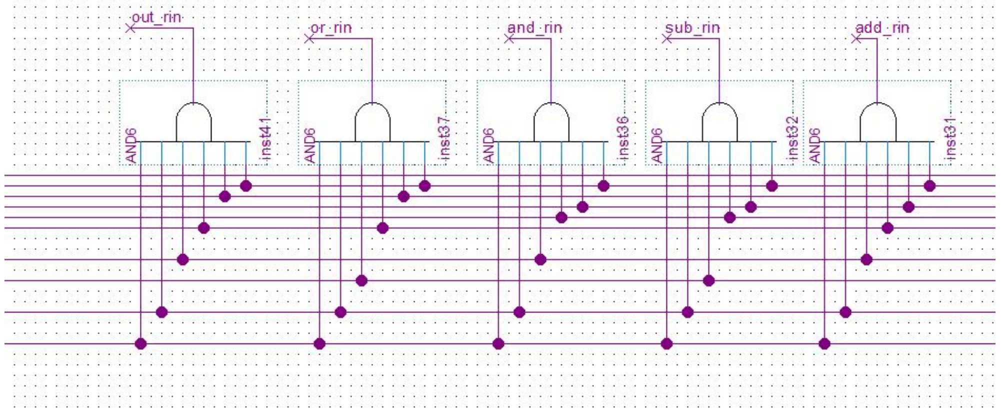
Opcode (31:26)	rs (25:21)	rt (20:16)	rd (15:11)	Don't care (10:0)

Here I didn't use funct code, instead I used the instruction to control the ALU directly as I did for displacement addressing. It is illustrated in the following figures.
I also designed the5 basic instructions for register indirect addressing:

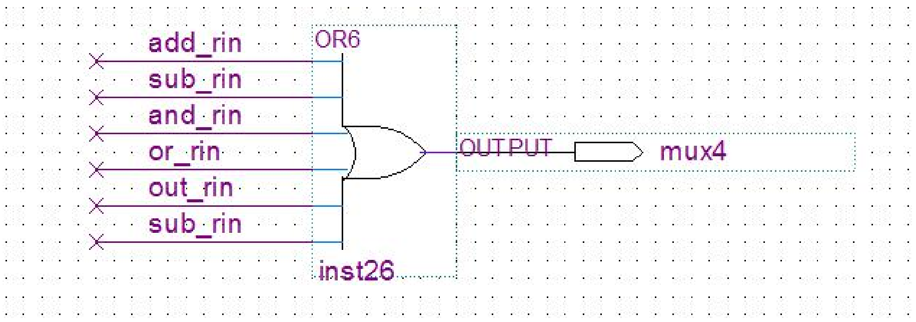
Instruction	Function
add_rin	rd=rt+M[rs]
sub_rin	rd=M[rs]-rt
and_rin	rd=rt AND M[rs]
or_rin	rd=rt OR M[rs]
out_rin	OUT M[rs]

instruction	Opcode	Regdst	Memtoreg	Memread	Memwright	Aluop	Alusrc	Regwrite	ALU2op
add_rin	000111	1	1	1	0	XX	X	1	010
sub_rin	001011	1	1	1	0	XX	X	1	110
and_rin	001111	1	1	1	0	XX	X	1	000
or_rin	010011	1	1	1	0	XX	X	1	001
out_rin	010111	1	0	1	0	XX	X	0	outen=1

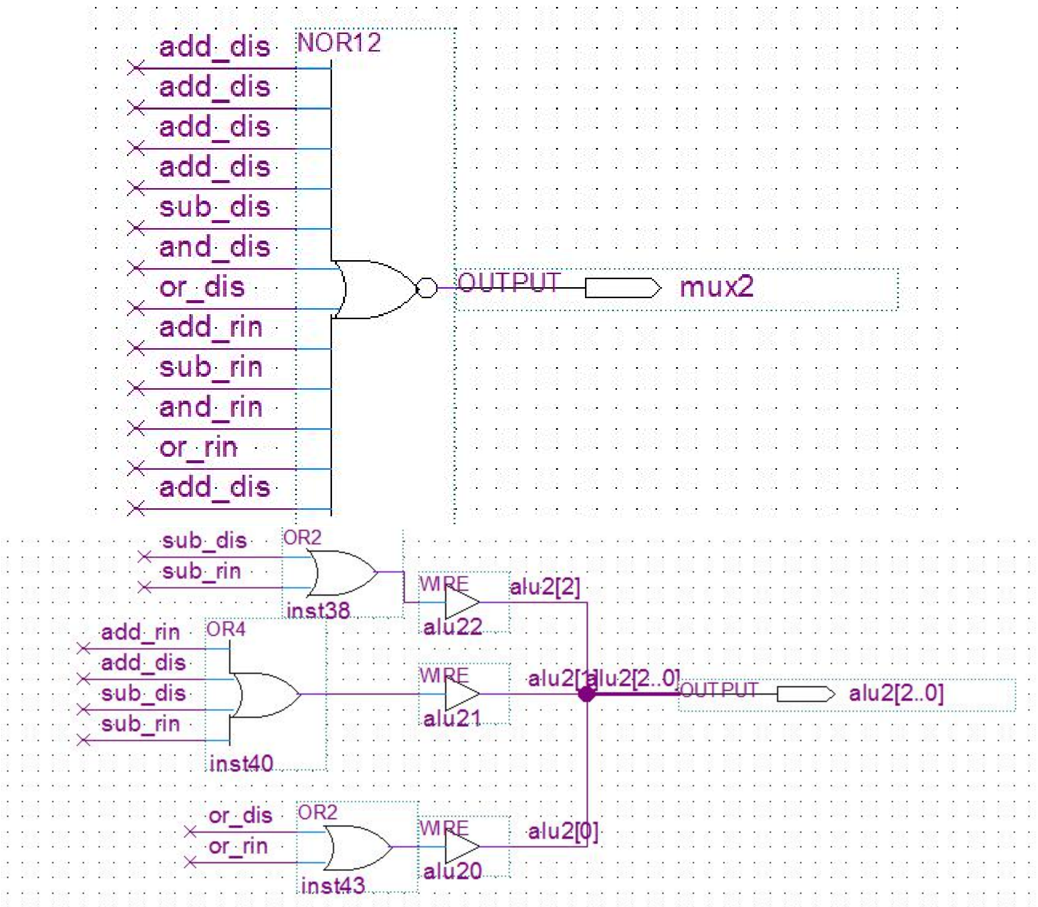
Hardware modification:
New Opcodes generate their corresponding instruction signals.



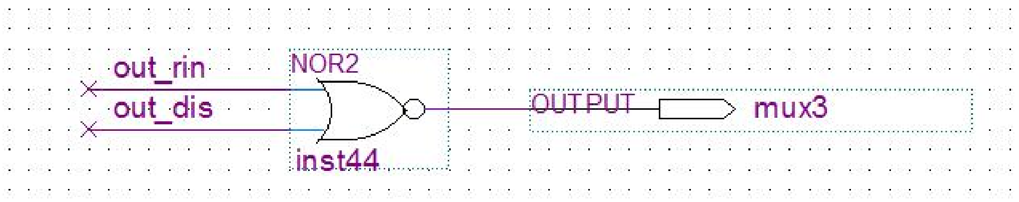
Signal mux4 in the below printscreen is the control signal of the new mux.



The components added for displacement addressing were also modified for new instructions.



ALU2 generates 010 for add, 110 for sub, 001 for or and 000 for and.



All functions of the 5 instructions were testified with the following mif flies I wrote.The numbers are arbitrary but the numbers used as pointers must be addresses of other numbers.

XiaotianBai_IRAM_RegIndirect.mif

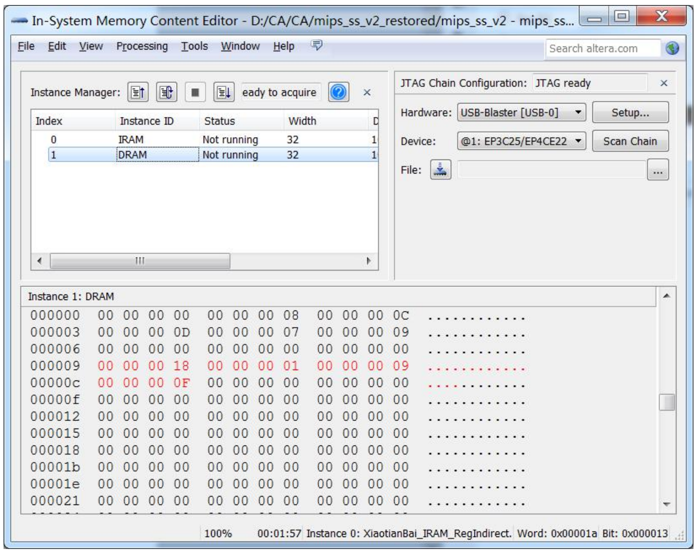
```
-- NYU:Poly CS6133 Computer Architecture
-- Instructor: Vikram Padman

-- Modified by Xiaotian Bai for the Spring 2017 Midterm
-- The codes include 5 instructions of register indirect addressing
-- Namely, add_rin, sub_rin, and_rin, or_rin and out_rin
--
-- File format:
-- Hex Address : bit31 ...... bit0;
WIDTH=32;
DEPTH=1024;
ADDRESS_RADIX=HEX;
DATA_RADIX=BIN;
CONTENT BEGIN
---------------------------------------------------------------------
-----
-- Overall program structure:
-- Row 000 to 004 to test add_rin in which row 002 is the instruction
-- Row 005 to 009 to test sub_rin in which row 007 is the instruction
-- Row 00A to 00E to test and_rin in which row 00C is the instruction
-- Row 00F to 013 to test or_rin in which row 011 is the instruction
-- Row 014 to 015 to test out_rin in which row 015 is the instruction
-- For convenience I set rs=r1, rt=r2, rd=r3 for all instructions
---------------------------------------------------------------------

-- Row 000 to 004 to test add_rin
--
--Hex Address : bit31..........bit15.........bit0;
--   |           |              |              |
    000     :    10001100000000010000000000000100;
--               |____||___||___||______________|
--                 |    |    |          |
--               op=lw, rs=0,rt=1,   offset=8
-- lw r1,4(r0) -- Load data memory 1 -> r1
--
    001     :    10001100000000100000000000001000;
--               |____||___||___||______________|
--                 |    |    |          |
--               op=lw, rs=0,rt=2,   offset=8
-- lw r2,8(r0) -- Load data memory 2 -> r2
--
    002     :    00011100001000100001100000000000;
--               |____||___||___||___|
--                 |    |    |    |
--         op=add_rin, rs=1,rt=2,rd=3
-- add_rin(r1,r2,r3) -- r3=M[r1]+r2
--
    003     :    10110000011000000000000000000000;
--               |____||___||___||______________|
--                 |    |    |          |
--               op=out,rs=3,rt=0,   offset=0
-- out r3 -- Display the result on the LEDs
--
```

```
    004     :     10101100000000110000000000100100;
--                |____||___||___||______________|
--                  |    |    |          |
--               op=sw, rs=0,rt=3,   offset=36
-- sw r1,36(r0) -- Save r3 ->  data memory 9
--
-- Row 005 to 009 to test sub_rin
--
    005     :     10001100000000010000000000001000;
--                |____||___||___||______________|
--                  |    |    |          |
--               op=lw, rs=0,rt=1,   offset=8
-- lw r1,8(r0) -- Load data memory 2 -> r1
--
    006     :     10001100000000100000000000001000;
--                |____||___||___||______________|
--                  |    |    |          |
--               op=lw, rs=0,rt=2,   offset=8
-- lw r2,8(r0) -- Load data memory 2 -> r2
--
    007     :     00101100001000100001100000000000;
--                |____||___||___||___|
--                  |    |    |    |
--         op=sub_rin, rs=1,rt=2,rd=3
-- sub_rin(r1,r2,r3) -- r3=M[r1]-r2
--
    008     :     10110000011000000000000000000000;
--                |____||___||___||______________|
--                  |    |    |          |
--              op=out, rs=3,rt=0,   offset=0
-- out r3 -- Display the result on the LEDs
--
    009     :     10101100000000110000000000101000;
--                |____||___||___||______________|
--                  |    |    |          |
--               op=sw, rs=0,rt=3,   offset=40
-- sw r3,40(r0) -- Save r3 ->  data memory 10
--
-- Row 00A to 00E to test and_rin
--
    00A     :     10001100000000010000000000001000;
--                |____||___||___||______________|
--                  |    |    |          |
--               op=lw, rs=0,rt=1,   offset=8
-- lw r1,8(r0) -- Load data memory 2 -> r1
--
    00B     :     10001100000000100000000000010100;
--                |____||___||___||______________|
--                  |    |    |          |
--               op=lw, rs=0,rt=2,   offset=20
-- lw r2,20(r0) -- Load data memory 4 -> r2
--
    00C     :     00111100001000100001100000000000;
--                |____||___||___||___|
--                  |    |    |    |
--         op=and_rin, rs=1,rt=2,rd=3
-- and_rin(r1,r2,r3) -- r3=M[r1] AND r2
--
    00D     :     10110000011000000000000000000000;
--                |____||___||___||______________|
--                  |    |    |          |
--              op=out, rs=3,rt=0,   offset=0
-- out r3 -- Display the result on the LEDs
--
    00E     :    10101100000000110000000000101100;
--               |____||___||___||______________|
--                 |     |     |           |
--               op=sw, rs=0,rt=3,   offset=44
-- sw r3,44(r0) -- Save r3 ->  data memory 11
--
-- Row 00F to 013 to test or_rin
--
    00F     :    10001100000000010000000000000100;
--               |____||___||___||______________|
--                 |     |     |           |
--               op=lw, rs=0,rt=1,   offset=4
-- lw r1,8(r0) -- Load data memory 1 -> r1
--
    010     :    10001100000000010000000000010000;
--               |____||___||___||______________|
--                 |     |     |           |
--               op=lw, rs=0,rt=7,   offset=16
-- lw r2,16(r0) -- Load data memory 2 -> r7
--
    011     :    01001100001000100001100000000000;
--               |____||___||___||___|
--                 |     |     |     |
--         op=or_rin, rs=1,rt=2,rd=3
-- or_rin(r1,r2,r3) -- r3=M[r1] OR r2
--
    012     :    10110000011000000000000000000000;
--               |____||___||___||______________|
--                 |     |     |           |
--              op=out, rs=3,rt=0,   offset=0
-- out r3 -- Display the result on the LEDs
--
    013     :    10101100000000110000000000110000;
--               |____||___||___||______________|
--                 |     |     |           |
--               op=sw, rs=0,rt=3,   offset=48
-- sw r3,48(r0) -- Save r3 ->  data memory 12
--
-- Row 014 to 015 to test out_rin
--
    014     :    10001100000000010000000000001000;
--               |____||___||___||______________|
--                 |     |     |           |
--               op=lw, rs=0,rt=1,   offset=8
-- lw r1,8(r0) -- Load data memory 2 -> r1
--
    015     :    01011100001000000000000000000000;
--               |____||___|
--                 |     |
--         op=out_dis, rs=1
-- out_rin r1 -- Display M[r1] on the LEDs
End;
```

XiaotianBai_DRAM_RegIndirect.mif

```
-- NYU:Poly CS6133 Computer Architecture
-- Instructor: Vikram Padman
--  Xiaotian Bai CS6133 Midterm

WIDTH=32;
DEPTH=1024;

ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;

CONTENT BEGIN
-----------------------------------------------
--  Variables used in the program
--  000 The value 0
--  001 to 003 are divisible by 4 for addressing
--  003 to 005 are used to test and_rin and or_rin
-----------------------------------------------
000 : 00000000;
001 : 00000008;
002 : 0000000C;
003 : 0000000D;
004 : 00000007;
005 : 00000009;
END;
-----------------------------------------------
```

Again the results are saved in DRAM:


In-System Memory Content Editor - D:/CA/CA/mips_ss_v2_restored/mips_ss_v2 - mips_ss...
File Edit View Processing Tools Window Help
Search altera.com
Instance Manager:
eady to acquire
JTAG Chain Configuration: JTAG ready
Hardware: USB-Blaster [USB-0]
Setup...
Device: @1: EP3C25/EP4CE22
Scan Chain
File:

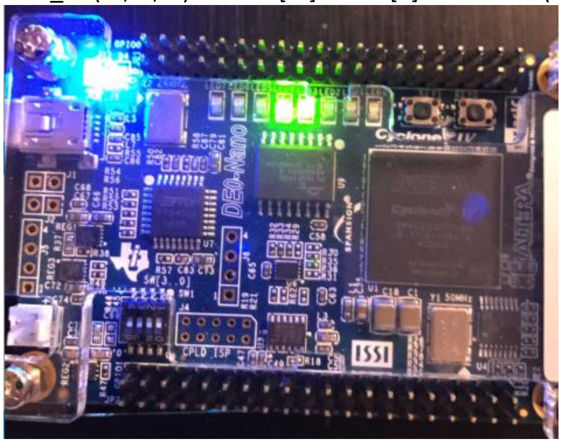
Index	Instance ID	Status	Width
0	IRAM	Not running	32
1	DRAM	Not running	32

Instance 1: DRAM

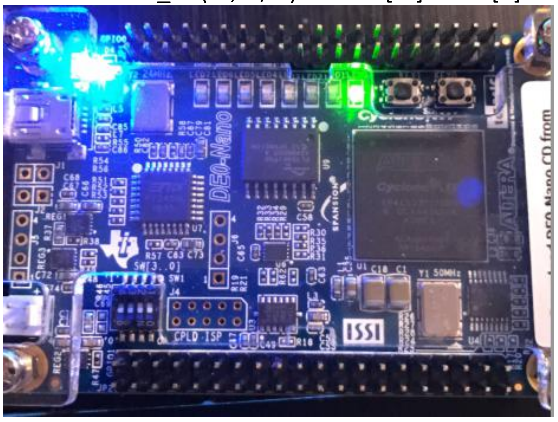
```
000000  00 00 00 00  00 00 00 08  00 00 00 0C  ............
000003  00 00 00 0D  00 00 00 07  00 00 00 09  ............
000006  00 00 00 00  00 00 00 00  00 00 00 00  ............
000009  00 00 00 18  00 00 00 01  00 00 00 09  ............
00000c  00 00 00 0F  00 00 00 00  00 00 00 00  ............
00000f  00 00 00 00  00 00 00 00  00 00 00 00  ............
000012  00 00 00 00  00 00 00 00  00 00 00 00  ............
000015  00 00 00 00  00 00 00 00  00 00 00 00  ............
000018  00 00 00 00  00 00 00 00  00 00 00 00  ............
00001b  00 00 00 00  00 00 00 00  00 00 00 00  ............
00001e  00 00 00 00  00 00 00 00  00 00 00 00  ............
000021  00 00 00 00  00 00 00 00  00 00 00 00  ............
```

100% 00:01:57 Instance 0: XiaotianBai_IRAM_RegIndirect. Word: 0x00001a Bit: 0x000013

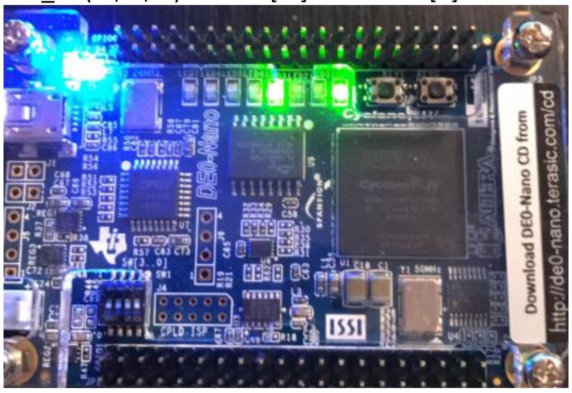
The outcome of add_rin(r1,r2,r3) is r3=M[r1]+r2=M[8]+C=C+C=16(Hex)=11000(Bin).



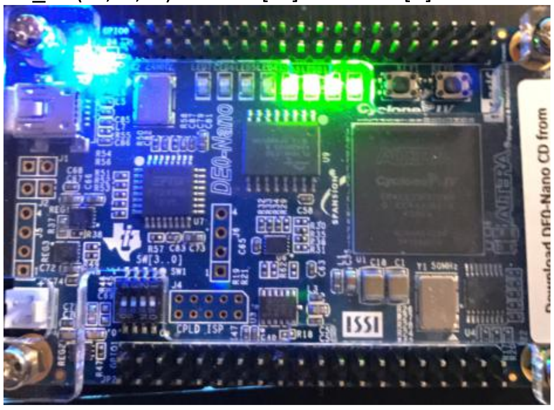
The outcome of add_rin(r1,r2,r3) is r3=M[r1]-r2=M[C]-C=D-C=1



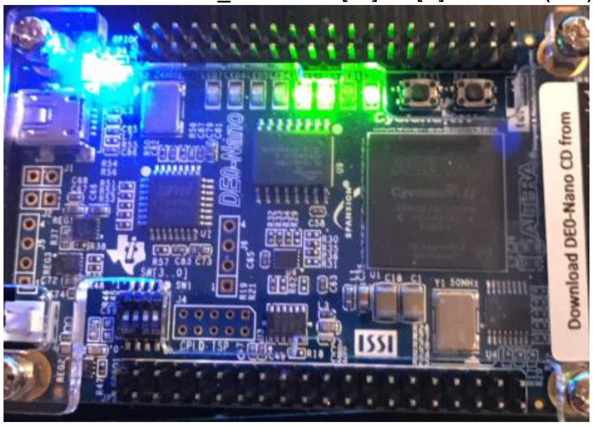
The outcome of and_rin(r1,r2,r3) is r3=M[r1] AND r2=M[C] AND 9=D AND 9=1001(Bin)



The outcome of or_rin(r1,r2,r3) is r3=M[r1] OR r2=M[8] AND 7=C AND 7=1111(Bin)

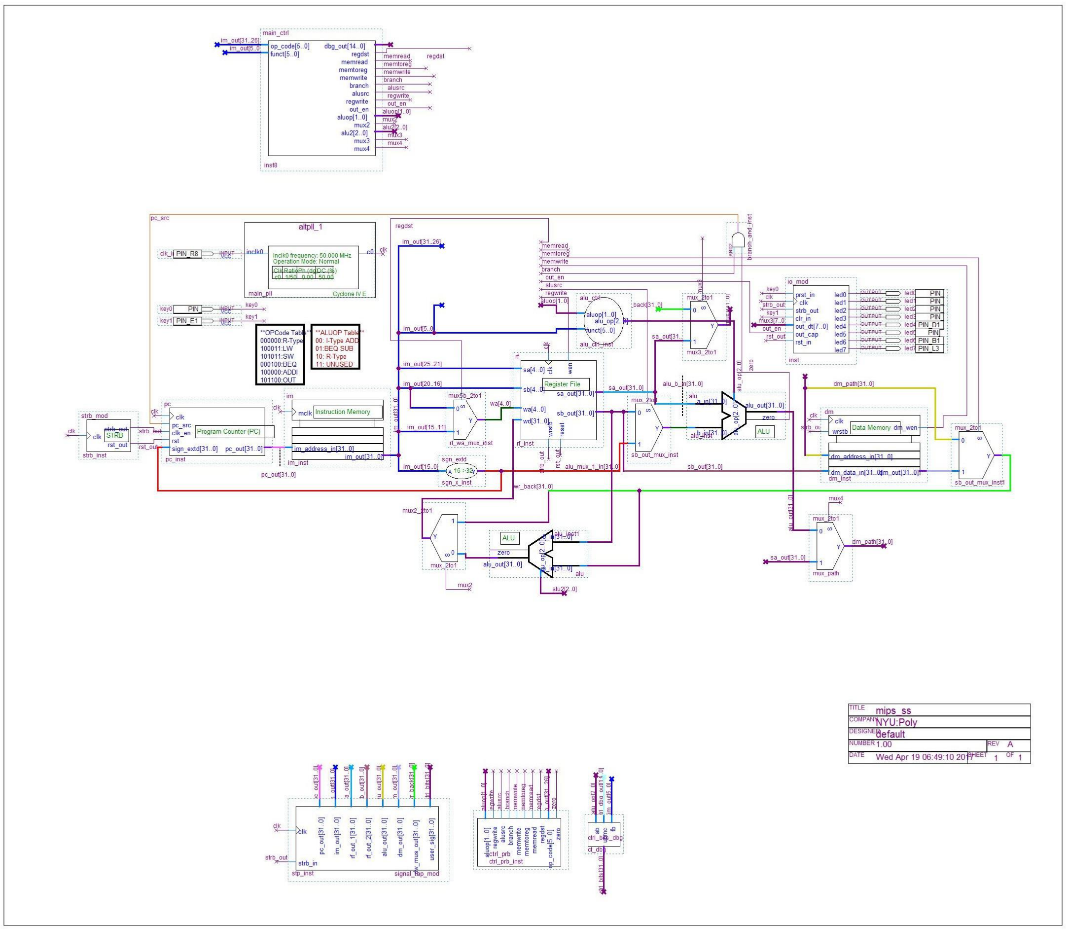


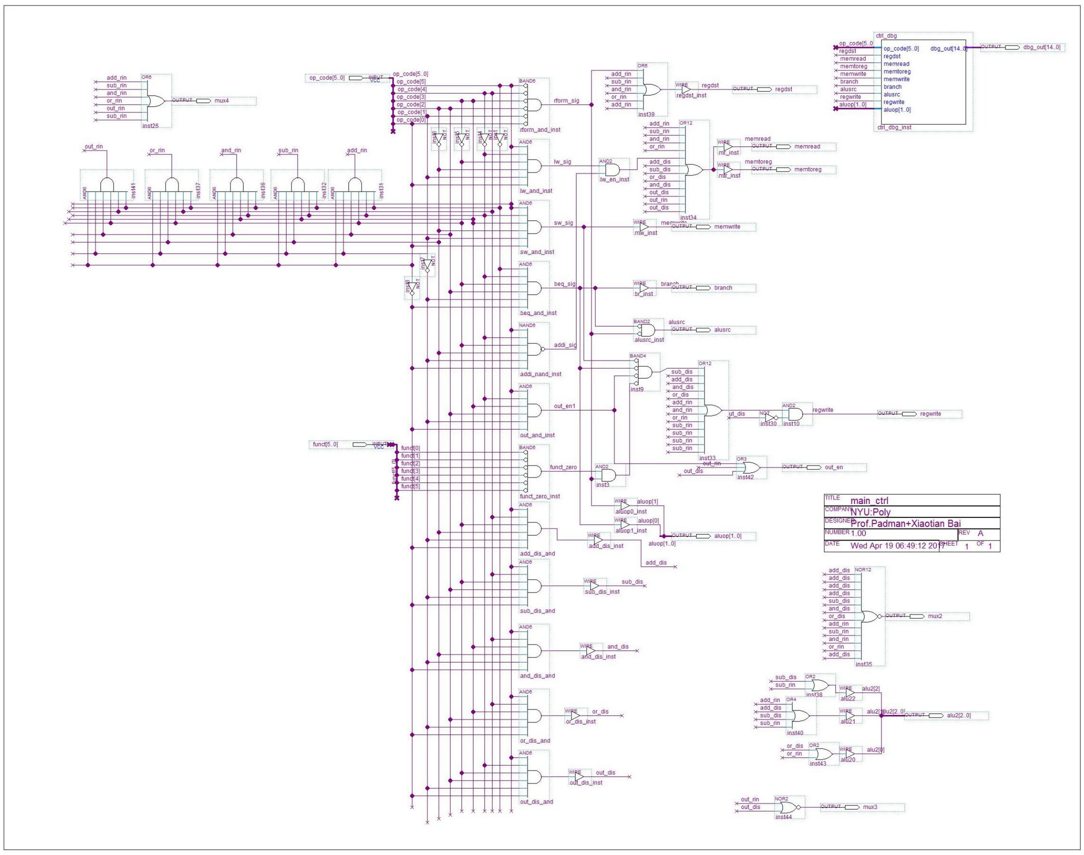
The outcome of out_rin r1 is M[r1]=M[C]=D=1101(Bin)



The associated .mif files “XiaotianBai_IRAM_RegIndirect.mif” and “XiaotianBai_DRAM_RegIndirect.mif” can be found in the sub-folder “XiaotianBai_Part3”.

After implementing the two addressing modes, the schematics of the CPU and main control became:





The two schematics are also submitted in “XiaotianBai_Part3” sub-folder.