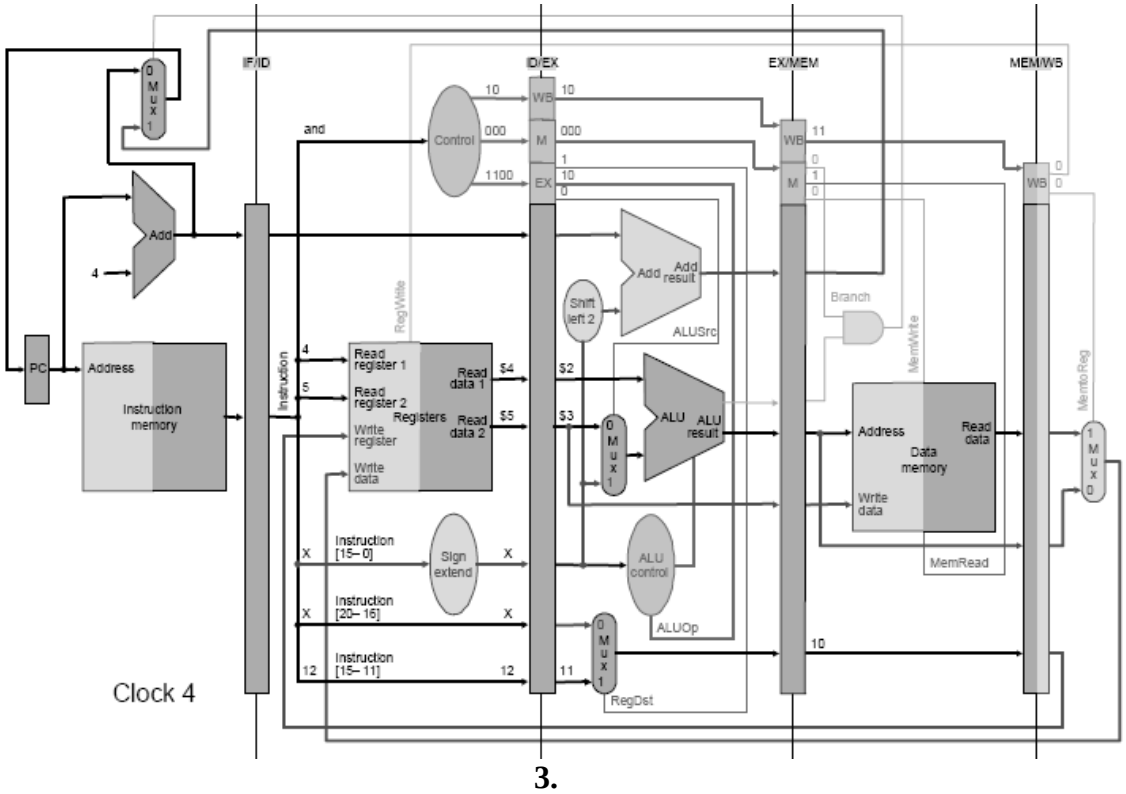
ECE 473 Computer Architecture and Organization, Fall 2010

Project 2: Design of a Five Stage Pipelined MIPS-like Processor

Instructor: Yifeng Zhu

Objectives

1. Implements the pipelined MIPS RISC processor core. The following figure is one example implementation.
2. Learn to work as a team to carry out a complex design task requiring task partitioning, effective communication, and cooperation.



3.

Figure 1. Example of Five Stage Pipelined Processor

1. Instruction Set and Test Programs

The pipeline processor should support the same set of instructions required in Project 1.

2. Project Team

This is a team project. All teams should comprise of exactly **TWO** members. The success of your project will depend greatly on how effectively and cooperatively your group works as a team.

3. Milestones & Timeline

- You have 6 weeks to complete Project 2 from Today (Oct 28, 2008).

#	Milestones	Time	Due
1	Integrate RAM and ROM into your single-cycle processor • Use three clocks, for register files, data memory, and instruction memory • Two clocks are generated by shifting the first one • Use *MegaWizard > I/O > ALTPLL* to shift	1 week	Nov. 4 (T)
2	Pipeline #1: pass all required R-type instructions • Test code 1: *add, nop, nop, nop, nop, nop* • Test code 2: *add, add, nop, nop, nop, nop* • Test code 3: *add, add, add, nop, nop, nop* • Test code 4: *add, add, add, add, add, add* • Forwarding from EX/MEM to ALU	2 weeks	Nov. 18(T)
3	Pipeline #2: pass all required I-Type and J-Type instructions • Hazard detection (Read after load) and stall implementation • Branch is solved at ID state • Hazard detection and stall implementation • Forwarding from MEM/WB to ALU	1 week	Nov. 25 (T)
4	Pipeline #3: pass the five test programs given in Project 1	1.5 weeks	Dec. 6 (M)
5	Write report and do project presentation	0.5 week	Dec. 9 (T)

4. Grading Criteria

Milestone	Basic	points
1	Integrate RAM and ROM	**10**
2	Pipeline for R-Type	**15**
3	Pipeline for I-Type & J-Type	**15**
4	Pass test programs	**50**
5	Report and presentation	**10**

Bonus	points
Cool stuff (You find it. Need approval from the instructor before the implementation.)	**TBD**

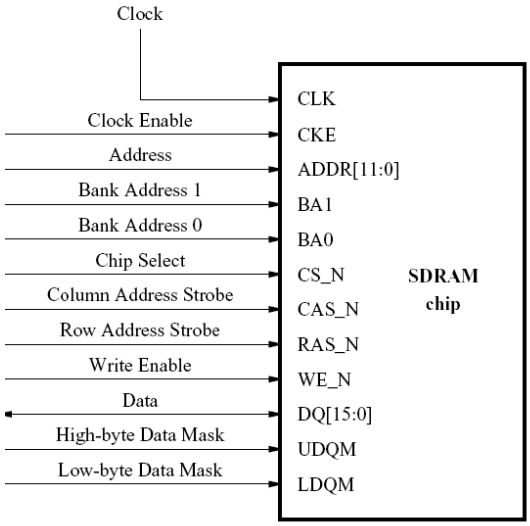
Milestone 1: RAM and ROM
10% of Project 2
Due: Thursday, November 4

In the pipelined processor design, you need to use RAM on the DE2 boards for instruction memory and data memory, respectively. The following documents on the project website are helpful to access RAM.

- RAM Megafunction User Guide
- Using the SDRAM Memory on Altera's DE2 Board with VHDL Design

1. The RAM Interface

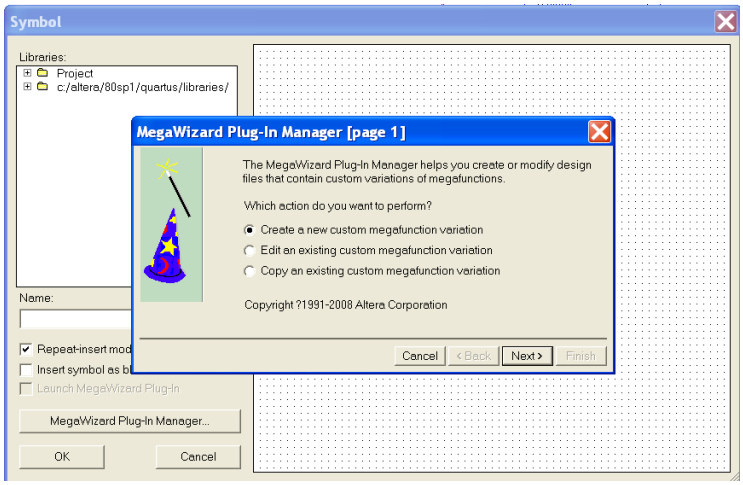
The DE2 board provides three types of memory, including 512-Kbyte SRAM, 8-Mbyte SDRAM and 4-Mbyte Flash memory. The SRAM is organized as 256K ×16bits, the SDRAM is organized as 1M ×16bits ×4 banks, and the Flash memory is 8-bit data bus. The signals needed to communicate with this chip are shown in the figure below. All of the signals, except the clock, can be provided by the SDRAM Controller. The clock signal is provided separately. It has to meet the clock-skew requirements. Note that some signals are active low, which is denoted by the suffix N.

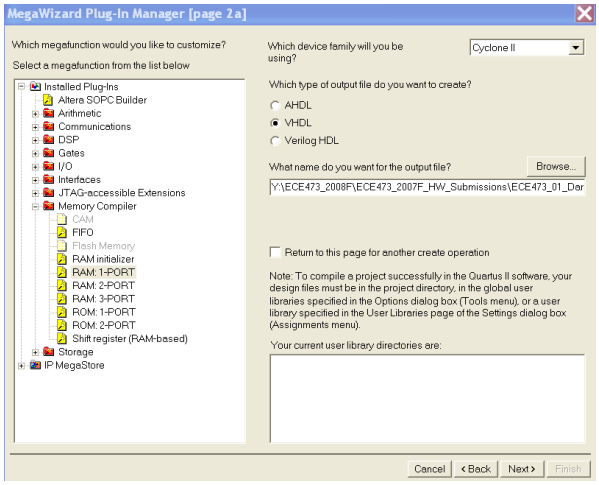


2. Using LPM to Access SDRAM and ROM

You can use *altsyncram* LPM provide in Quartus II to access the RAM on the DE2 board. (Quartus II also provides *lpm_ram_dq* and *lpm_rom* functions, but they are only for backward compatibility. The *altsyncram* megafunction is recommended by Altera.) The following briefly summarizes the procedure of using *altsyncram*.

- In this project, you need to create **a new** Quartus II projects to access. The target FPGA chip on the DE2 board is Cyclone II ***EP2C35F672C6***.
- The tutorial *Using Library Modules in VHDL Designs* describes the usage of MegaWizard to build a desired LPM module.
    i. In the Memory Compiler category, you can find *RAM: 1-PORT LPM*
    ii. Select *VHDL* HDL as the type of output file to create
    iii. Give the file the name *ramlpm.vhd*.
    iv. In the next windows, you can customize the RAM configurations, such as the width, clock. You can also initiate the values of RAM using a MIF file. The format of MIF is given in this handout.





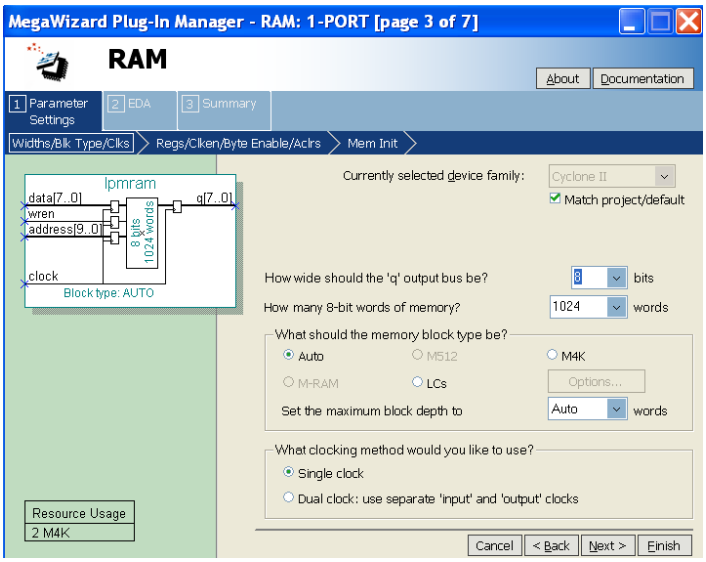


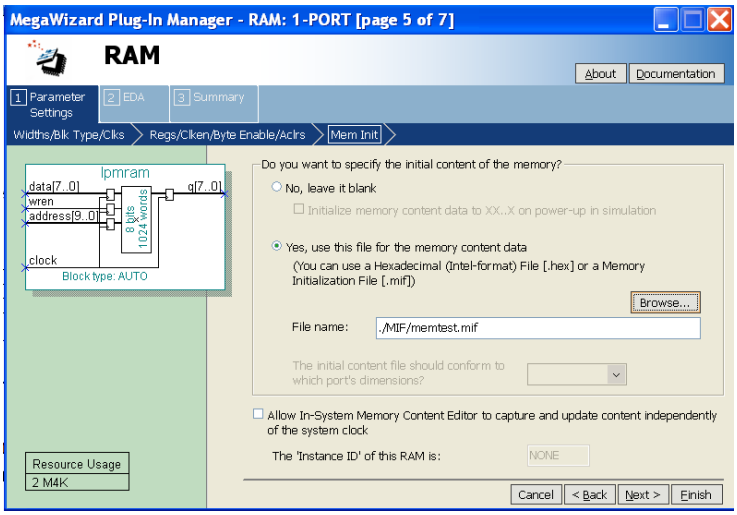
Table 2–1. RAM: 1-PORT MegaWizard Plug-In Manager Page 3 Options	
Function	**Description**
Currently selected device family:	Specify which Altera device family to use.
How wide should the 'q' output bus be?	Specify the width of the output data bus. Note that you can manually enter a number that is not in the drop-down list.
How many 8-bit words of memory?	Specify the number of 8-bit words in the memory. 8-bit represents the width of the 'q' output bus that you set. You can set different widths of output data bus. Note that you can manually enter a number that is not in the drop-down list.
What should the RAM block type be?	Specify the RAM block type. The options available vary depending on the device you select. *(1)*

The following notes are copied from Atlera's *RAM Megafunction User Guide*.

- The RAM: 1-PORT MegaWizard Plug-In Manager allows you to specify either of two clocking modes: a single clock mode or a dual clock (input/output) mode. In single clock mode, the read and write operations are synchronous with the same clock. In the Stratix and Cyclone series of devices, a single clock with a clock enable controls all registers of the memory block. Dual clock (input/output) mode operates with two independent clocks: inclock (input clock for write operation) and outclock (output clock for read operation). The input clock controls all registers related to the data input to the memory block, including data, address, byte enables, read enables, and write enables. The output clock controls the data output registers.

- Synchronous write operations into the memory block use the *address[]* and *data[]* ports, which are triggered by the rising edge of the *inclock* while the *we* (write enable) port is enabled. For asynchronous operation, the *address[]* and *data[]* signals must be valid at both edges of the write enable signal. Ideally, the values on the data and address lines should not be changed while the *we* port is active.

3. Memory Initialization Format (MIF)



The data of the RAM and Rom can be initialized according to an ASCII text file named Memory Initialization Format (with the extension .mif). The following gives two example MIF files.

```
DEPTH = 256;                % Memory depth and width are required %
WIDTH = 16;                 % Enter a decimal number      %
ADDRESS_RADIX = HEX;        % Address and value radixes are optional     %
DATA_RADIX = HEX;           % Enter BIN, DEC, HEX, or OCT; unless %
                            % otherwise specified, radixes = HEX  %
-- Specify initial data values for memory, format is address : data
CONTENT BEGIN
[00..FF]        :       0000;   % Range - Every address from 00 to FF = 0000 %
-- Computer Program for A = B + C
00              :       0210;   % LOAD A with MEM(10)  %
01              :       0011;   % ADD MEM(11) to A     %
02              :       0112;   % STORE A in MEM(12)   %
03              :       0212;   % LOAD A with MEM(12) check for new value of FFFF %
04              :       0304;   % JUMP to 04 (loop forever)    %
10              :       AAAA;   % Data Value of B              %
11              :       5555;   % Data Value of C              %
12              :       0000;   % Data Value of A - should be FFFF after running program $
END;
```

Note: If multiple values are specified for the same address, only the last value is used.

4. Project Instructions and Requirements

1. **Build a project that can read and write SDRAM by using RAM: 1-PORT MegaWizard Plug-In Manager. The output of SDRAM is 32 bits. The SDRAM will be used as the data memory in Project 2.**

   - The SDRAM is initialized by a MIF file. Display the value at the RAM location identified by a 5-bit address specified with toggle switches SW15−11. On the LCD screen, the first row shows the data address, and the second row show the data value. Use KEY0 as the Clock input.

- Write a value into the RAM. Use toggle switches SW7−0 to input a byte of data into the RAM location identified by a 5-bit address specified with toggle switches SW15−11. Use SW17 as the Write signal and use KEY0 as the Clock input. On the LCD screen, the first row shows the data address, and the second row show the data value in hex.

- Test your circuit and make sure that all 32 locations can be loaded and stored properly.

2. Build another project that can read ROM by using ROM: 1-PORT MegaWizard Plug-In Manager. The output of ROM is 32 bits. The ROM will be used as the instruction memory in Project 2.

- The ROM is initialized by a MIF file. Display the value at the RAM location identified by a 5-bit address specified with toggle switches SW15−11. On the LCD screen, the first row shows the data address, and the second row show the data value. Use KEY0 as the Clock input.

- Test your circuit and make sure that all 32 locations can be loaded properly.

3. Integrate the RAM and ROM into the single cycle processor designed in Project 1. You need pass the test programs. ROM is used to store instructions and RAM is used for data.

Before the integration, you need to perform high-level design first.

(1) What is width of the output port used of ROM? How to adjust the PC correspondingly?
(2) What is the width of the output bus for RAM? How to adjust the memory address correspondingly?
(3) You might need multiple clocks to drive the RAM and ROM. How would you design the clocks?

Write a short report to

(1) Describes the widths of each memory output.
(2) Draw an overall diagram to show the clocking of your processor. Mark clearly whether each component works on the rising edge or the falling edge.
(3) Summarize the results of each test program.

Submit your projects through the homework handin website.

Milestone 2: R-Type Instructions
10% of Project 2
Due: Thursday, November 18

In this milestone, your pipeline processor need to successfully run the required R-Type MIPS instructions required in the following table.

R-Type Instruction						
Instruction	OPCode	RS	RT	RD	Shamt	Funct
add $3, $2, $1	000000	00010	00001	00011	00000	100000
addu $3, $2, $1	000000	00010	00001	00011	00000	100001
sub $3, $2, $1	000000	00010	00001	00011	00000	100010
subu $3, $2, $1	000000	00010	00001	00011	00000	100011
and $3, $2, $1	000000	00010	00001	00011	00000	100100
or $3, $2, $1	000000	00010	00001	00011	00000	100101
nor $3, $2, $1	000000	00010	00001	00011	00000	100111
slt $3, $2, $1	000000	00010	00001	00011	00000	101010
sll $3, $2, 1	000000	00000	00010	00011	00001	000000
srl $3, $2, 1	000000	00000	00010	00011	00001	000010
~~**sra** $3, $2, 1~~	~~000000~~	~~00000~~	~~00010~~	~~00011~~	~~00001~~	~~000011~~
jr $2	000000	0010	00000	00000	00000	001000
nop	000000	00000	00000	00000	00000	000000

Test program (It does not have any logic meaning.)

```
#Assume $1 = -30 (0xFFFFFFE2), $2 = 56
(0x00000038). Set $1 and $2 before running the test.
1       add $3, $2, $1
2       addu $3, $2, $1
3       sub $3, $2, $1
4       subu $3, $2, $1
5       and $3, $2, $1
6       or $3, $2, $1
7       nor $3, $2, $1
8       slt $3, $2, $1
9       sll $3, $2, 1
10      srl $3, $2, 1
~~11      sra $3, $2, 1~~
12      jr $2
13      nop
```

	Instruction	Address	Code	Value of $3
1	**add** $3, $2, $1	0x00400000	**0x00411820**	
2	**addu** $3, $2, $1	0x00400004	**0x00411821**	
3	**sub** $3, $2, $1	0x00400008	**0x00411822**	
4	**subu** $3, $2, $1	0x0040000c	**0x00311823**	
5	**and** $3, $2, $1	0x00400010	**0x00411824**	
6	**or** $3, $2, $1	0x00400014	**0x00411825**	
7	**nor** $3, $2, $1	0x00400018	**0x00411827**	
8	**slt** $3, $2, $1	0x0040001c	**0x0041182a**	
9	**sll** $3, $2, 1	0x00400020	**0x00021840**	
10	**srl** $3, $2, 1	0x00400024	**0x00021842**	
~~**11**~~	~~**sra** $3, $2, 1~~	~~0x00400028~~	~~**0x00021843**~~	
12	**jr** $2	0x0040002c	**0x00400008**	PC =
13	**nop**	0x00400030	**0x00000000**	N/A

Simulate the test program and print the diagram. The diagram needs to show at least PC, 32-bit Instruction, and value of $3.

Milestone 3: I-Type and J-Type Instructions
10% of Project 2
Due: Thursday, November 25

In this milestone, your pipeline processor need to successfully run the required I-Type and J-Type MIPS instructions required in the following table.

I-Type Instruction				
Instruction	OPCode	RS	RT	Immediate
andi $2, $1, 20	001100	00001	00010	0000000000010100
ori $2, $1, 20	001101	00001	00010	0000000000010100
~~**slti** $2, $1, 20~~	~~001010~~	~~00001~~	~~00010~~	~~0000000000010100~~
addi $2, $1, 20	001000	00001	00010	0000000000010100
beq $1, $2, 20	000100	00001	00010	0000000000010100
bne $1, $2, 20	000101	00001	00010	0000000000010100
lw $1, 20($2)	100011	00010	00001	0000000000010100
sw $1, 20($2)	101011	00010	00001	0000000000010100
lui $1, 20	001111	00000	00001	0000000000010100

J-Type Instruction		
Instruction	OPCode	Address
j L5	000010	00000100000000000000001000
jal L5	000011	00000100000000000000001000

1. Test program for arithmetic

```
ori  $1,  $0, 0x1
ori  $2,  $0, 0x2
ori  $3,  $0, 0x3
ori  $4,  $0, 0x4
andi $5, $1, 0x10
~~slti  $5, $2,  0x12~~
addi $5, $3, 0x13
```

Simulate the test program and print the diagram. The diagram needs to show at least PC, 32-bit Instruction, and value of $5. Print out your simulation diagram.

	Instruction	Address	Code	Value of $5
1	**ori** $1, $0, 1	0x00400000	**0x34010001**	
2	**ori** $2, $0, 2	0x00400004	**0x34020002**	
3	**ori** $3, $0, 3	0x00400008	**0x34030003**	
4	**ori** $4, $0, 4	0x0040000c	**0x34040004**	

5	**andi** $5, $1, 16	0x00400010	**0x30250010**	
~~6~~	~~**slti** $5, $2, 18~~	~~0x00400014~~	~~**0x28450012**~~	
7	**addi** $5, $3, 19	0x00400018	**0x20650013**	

2. Test program for beq

```
        ori $1, $0, 1
        ori $2, $0, 1
        beq $1, $2, L
        nop
        ori $4,$0, -1
L:      ori $3, $0, 1
```

If the codes run correctly, then $4=0 and $3=1. If $4=-1, then your project does not jump correctly. Print out your simulation diagram.

	Instruction	Address	Code	Value of $4 and $3
1	**ori** $1, $0, 1	0x00400000	**0x34010001**	
2	**ori** $2, $0, 1	0x00400004	**0x34020001**	
3	**beq** $1, $2, L	0x00400008	**0x10220002**	
4	**nop**	0x0040000c	**0x00000000**	
5	**ori** $4, $0, -1	0x00400010	**0x3404ffff**	
6	**ori** $3, $0, 1	0x00400014	**0x34030001**	

3. Test program for bne

```
        ori $1, $0, 1
        ori $2, $0, 2
        bne $1, $2, L
        nop
        ori $4,$0, -1
L:      ori $3, $0, 1
```

If the codes run correctly, then $4=0 and $3=1. If $4=-1, then your project does not jump correctly. Print out your simulation diagram.

	Instruction	Address	Code	Value of $4 and $3
1	**ori** $1, $0, 1	0x00400000	**0x34010001**	
2	**ori** $2, $0, 2	0x00400004	**0x34020002**	
3	**bne** $1, $2, L	0x00400008	**0x14220002**	
4	**nop**	0x0040000c	**0x00000000**	
5	**ori** $4, $0, -1	0x00400010	**0x3404ffff**	
6	**L: ori** $3, $0, 1	0x00400014	**0x34030001**	

4. Test program for jump

```
      ori $1, $0, 1
      j, L
      nop
      ori $4,$0, -1
L:    ori $3, $0, 1
```

If the codes run correctly, then $4=0 and $3=1. If $4=-1, then your project does not jump correctly. Print out your simulation diagram.

	Instruction	Address	Code	Value of $4 and $3
1	**ori** $1, $0, 1	0x00400000	**0x34010001**	
2	**j** L	0x00400004	**0x08100004**	
3	**nop**	0x00400005	**0x00000000**	
4	**ori** $4, $0, -1	0x0040000c	**0x3404ffff**	
5	**L: ori** $3, $0, 1	0x00400010	**0x34030001**	

5. Test program for jump-and-link

```
      ori $1, $0, 1
      jal, L
      nop
      ori $4,$0, -1
L:    ori $3, $0, 1
```

The ***jal*** instruction saves the return address in register $31. This register is also called $ra (where "ra" means return address).

If the codes run correctly, then $4=0 and $3=1. If $4=-1, then your project does not jump correctly. In addition, *$31 should be correctly set*. Print out your simulation diagram.

	Instruction	Address	Code	Value of $4, $3, and $31
1	**ori** $1, $0, 1	0x00400000	**0x34010001**	
2	**jal** L	0x00400004	**0x0c100004**	
3	**nop**	0x00400005	**0x00000000**	
4	**ori** $4, $0, -1	0x0040000c	**0x3404ffff**	
5	**L: ori** $3, $0, 1	0x00400010	**0x34030001**	