A06: รวมเข้าด้วยกัน.

"*Whether you think you can, or you think you can't--you're right.*" — Henry Ford

เราทำมากันเยอะแล้ว--- สร้าง ALU ไปแล้ว. ตอนนี้, มาสร้างโปรเซสเซอร์ (เวอร์ชั่นนักเรียน) กันเถอะ.
งานมอบหมายนี้ มีพื้นฐานจากเนื้อหาบท Datapath and Control ของ ตำรา P&H 2nd edition.
จากสถาปัตยกรรม (ชุดคำสั่ง) ที่ให้ตามตารางที่ 1, ส่วนประกอบต่างๆและเส้นทางข้อมูล (datapath) ที่เชื่อมต่อมัน แสดงดังรูปที่ 1. สรุปแต่ละขั้นตอนที่ใช้ในการทำคำสั่งแสดงในตารางที่ 2.

MIPS core instruction	Format		m/c code					
		Bits =	6	5	5	5	5	6
add	add rd, rs, rt		0	rs	rt	rd	0	0x20
subtract	sub rd, rs, rt		0	rs	rt	rd	0	0x22
and	and rd, rs, rt		0	rs	rt	rd	0	0x24
or	or rd, rs, rt		0	rs	rt	rd	0	0x25
set on less than	slt rd, rs, rt		0	rs	rt	rd	0	0x2A
load word	lw rt, addr		0x23	rs	rt	offset		
store word	sw rt, addr		0x2B	rs	rt	offset		
branch on equal	beq rs, rt, label		4	rs	rt	offset		
jump	j target		2	target				
Table 1: an instruction set that we are going to implement a hardware for. Woohoo! sounds fun.								

หน้าที่:

1. ศึกษาเนื้อหา Section 5.4 A Multiple-cycle Implementation (P&H 2e).

2. ศึกษาพฤติกรรมการทำงานของแต่ละชิ้นส่วน.
ศึกษาพฤติกรรมการทำงานของแต่ละชิ้นส่วน. สังเกตุความสัมพันธ์ระหว่าง เอาต์พุต และ อินพุตของมัน. ศึกษาการทำงานของ state component, เช่น memory.

a) The word ALU.
ลองรัน `WordALU_tb` และตรวจดูว่าแต่ละชิ้นส่วนทำงานได้ถูกต้อง.
นักเรียนควรจะได้ waveform ดังแสดงในรูปที่ 2.
ควรฝึกให้เป็นนิสัยในการตรวจสอบผลด้วยมือ เช่น ตรวจดูว่าการคำนวณทำงานถูกต้องหรือไม่ **ผลลัพธ์**, สัญญาณ **overflow**, และ สัญญาณ **Zero** เป็นไปตามที่คาดไว้หรือไม่.

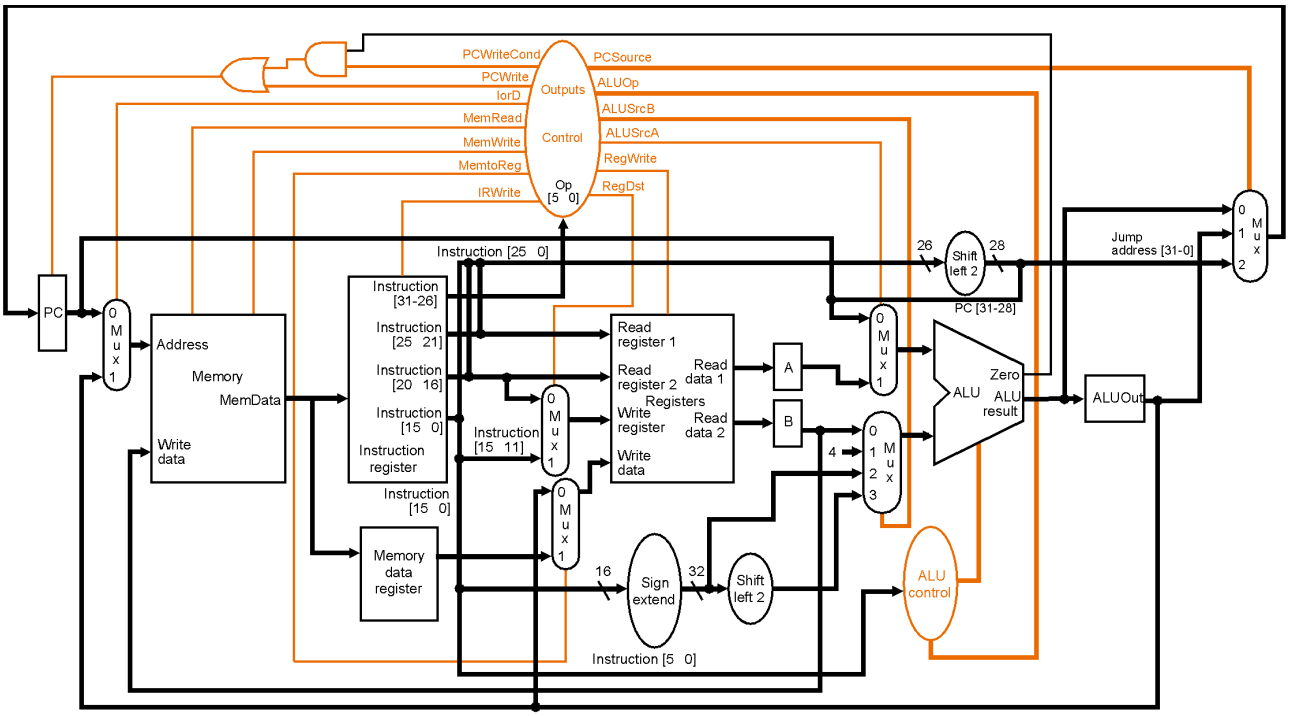


Figure 1: Fig. 5.33 (P&H 2e) The complete datapath for the multicycle implementation of the instruction set shown in Table 1.

Step	Action for R-type instructions	Action for memory-reference instructions		Action for branches	Action for jumps

1	**IR <= Memory[PC]**
	PC <= PC + 4
	`IorD = 0, MemRead, IRWrite`
	`ALUSrcA = 0, ALUSrcB = 01, ALUOp =00, PCSource = 00, PCWrite`				

2	**A <= Reg[IR[25:21]]** **B <= Reg[IR[20:16]]** **ALUOut <= PC + (sign-extend (IR[15:0]) << 2)** `ALUSrcA = 0,  ALUSrcB = 11, ALUOp =00`				
3	**ALUOut <= A op B** `ALUSrcA = 1, ALUSrcB = 00, ALUOp = 10`	**ALUOut <= A + sign-extend (IR[15:0])** `ALUSrcA = 1, ALUSrcB =10, ALUOp = 00`		**if (A == B)** **PC <= ALUOut** `ALUSrcA = 1, ALUSrcB = 00, ALUOp = 01, PCSource = 01, PCWriteCond`	**PC <= {PC[31:28],(IR[25:0]],2'b00)}** `PCSource = 10, PCWrite`
4	**Reg[IR[15:11]] <= ALUOut** `RegDst = 1, MemtoReg = 0, RegWrite`	**Load:** **MDR <= Memory[ALUOut]** `IorD = 1 MemRead`	**Store:** **Memory[ALUOut] <= B** `IorD = 1 MemWrite`		
5		**Load:**			

		Reg[IR[20:16]] <= MDR `RegDst = 0, MemtoReg = 1, RegWrite`			
Table 2: Summary of the steps taken to execute any instruction class with corresponding control signals.					

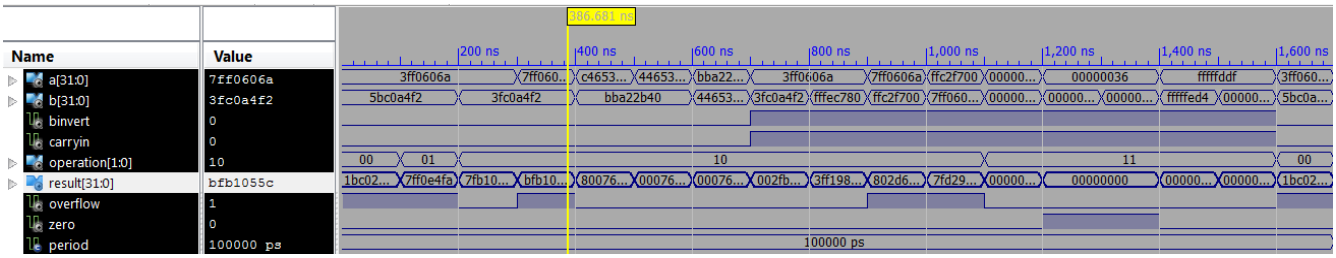


Figure 2: Waveform obtained from WordALU_tb.

(b) The register file.

ลองรัน `regfile_tb` และตรวจสอบการทำงานว่าถูกต้องหรือไม่.

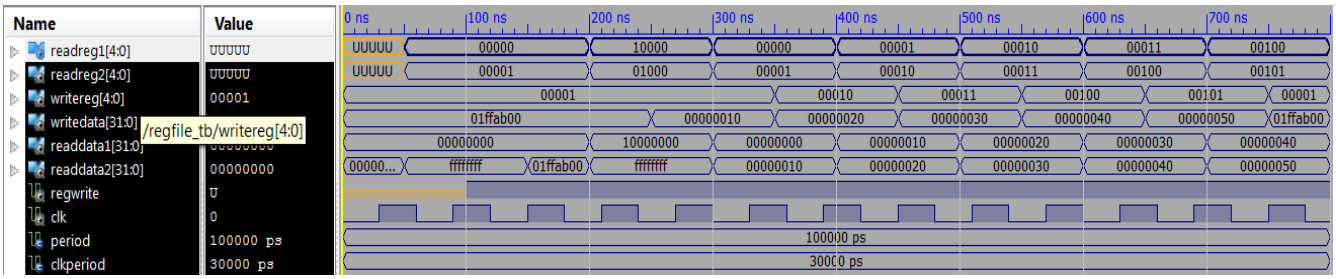


Figure 3: Waveform obtained from regfile_tb.

สังเกตุว่า สัญญาณเอาต์พุต **readdata1** และ **readdata2** ตอบสนองทันทีที่สัญญาณ **readreg1** หรือ **readreg 2** มีการเปลี่ยนแปลง. รูปที่ 4 ขยายรูปที่ 3 ออกมาพร้อมทั้งแสดงข้อความอธิบาย

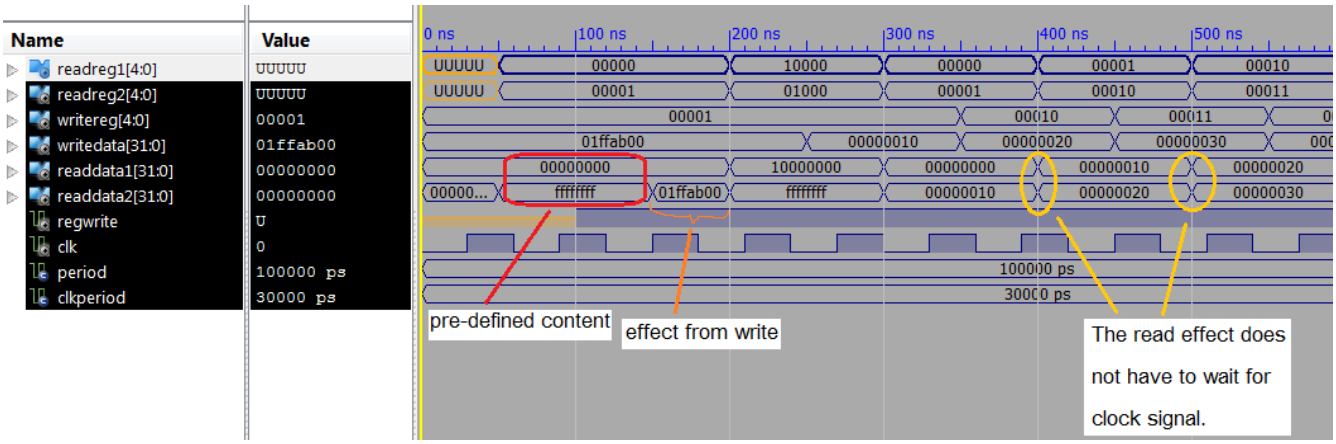


Figure 4: Enlarged vesion of Figure 3

(c) The one-word registers.

ลองรัน `nbitregister_tb` และตรวจสอบดูว่ามันทำงานถูกต้อง.

การที่ใส่สัญญาณ **regload** เข้าไป (กรณีนี้ **regload** เป็น '1') จะทำให้ register รับค่าอินพุตเข้าไปและเก็บไว้ จนกระทั่งเกิดการเปลี่ยนแปลงใหม่ (จาก สัญญาณ **regload** หรือ **regclear**).

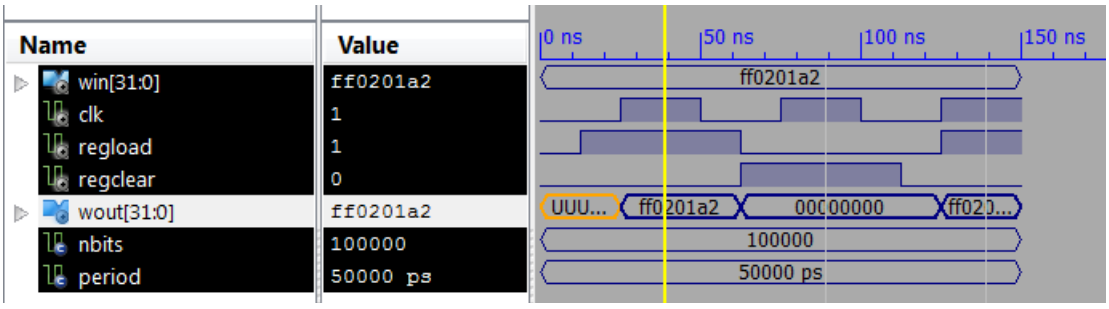


Figure 5: Waveform obtained from nbitregister_tb.

(d) The multiplexors.

(d1) The 2-to-1 multiplexor.

ลองรัน `mux2_tb` และตรวจดูการทำงานของ multiplexor (แบบสองเลือกหนึ่ง) ว่าทำงานได้อย่างถูกต้อง. (ถ้าทำงานถูกต้อง) เอาต์พุตของ multiplexor จะเหมือนกับอินพุต ที่เลือก

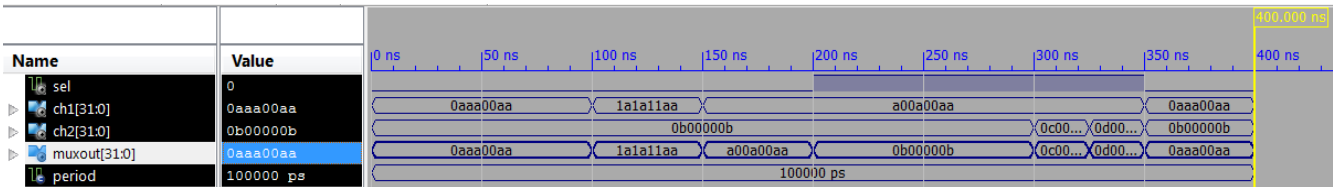


Figure 6: Waveform obtained from mux2_tb.

(d2) The 4-to-1 multiplexor.

ลองรัน `mux4_tb` และตรวจสอบการทำงานของ multiplexor (แบบสี่เลือกหนึ่ง) ว่าทำงานได้ถูกต้อง.

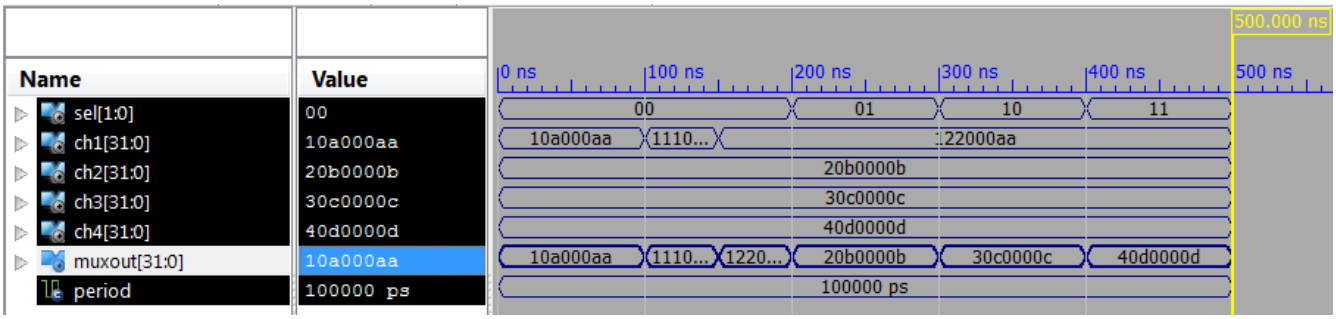


Figure 7: Waveform obtained from mux4_tb.

(e) The sign extension unit.

ลองรัน `signextend_tb` และตรวจสอบดูว่าทำงานถูกต้องหรือไม่.

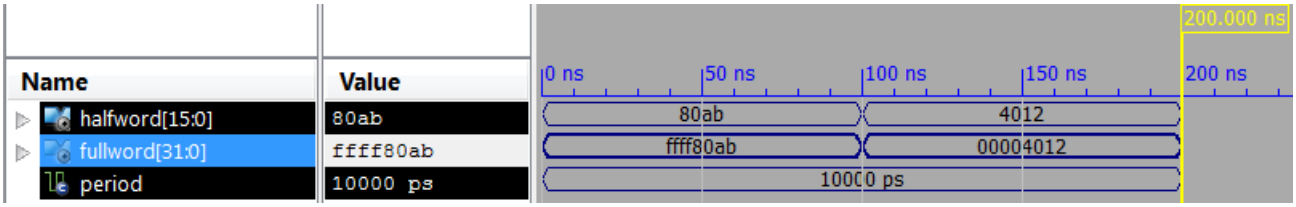


Figure 8: Waveform obtained from signextend_tb.

(f) The memory unit.

ลองรัน `ch5mem_tb` และตรวจสอบดูว่ามันทำงานถูกต้อง.

ตัว program segment จะมีการโหลดคำสั่งเข้าไปเมื่อ สัญญาณ startup เป็น '1'. ที่แอดเดรส 0x10000000 และ 0x10000004, หน่วยความจำ (memory) จะมีข้อมูลอยู่ (ถูกโหลดเข้าไปตอนสัญญาณ startup เป็น '1'). ข้อมูลในหน่วยความจำสามารถจะถูกเขียนใหม่หรือแก้ไขได้โดยใส่ ค่าใหม่ไปที่พอร์ต `data`, ดังแสดงในรูปที่ 9 ระหว่างช่วงเวลา 120 ถึง 160 ns. ผลของการเขียนข้อมูลเข้าไปใหม่จะเห็นตอนที่อ่านข้อมูลออกมา, ดังแสดง ที่ค่า `q` ในรูป ระหว่างเวลา 170 to 210 ns. สังเกตุว่า เอาต์พุตของหน่วยความจำที่แอดเดรส 00400000, 00400004, 00400008

และ 0040000C จะเก็บไบนารีที่แปลเป็น machine codes ของคำสั่ง `lw $t0, 0($s0)`, `lw $t1, 4($s0)`, `slt $t2, $t0, $t1`, และ `beq $t2, 3` ตามลำดับ.

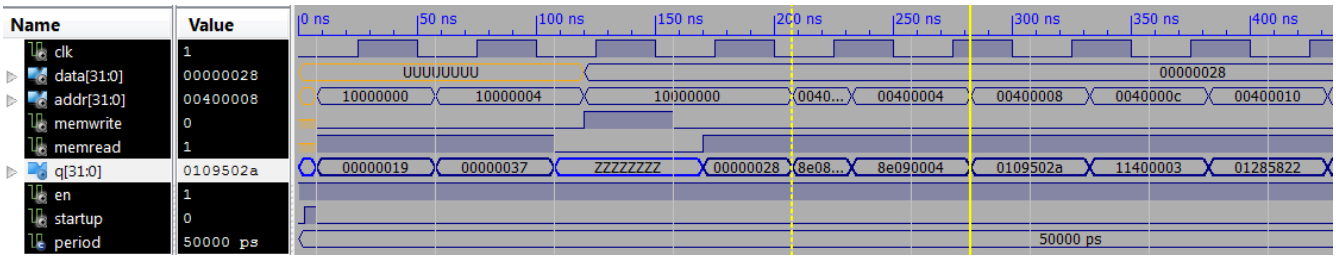


Figure 9: Waveform obtained from ch5mem_tb.

คำถาม:

* f1. ใน write mode, หน่วยความจำนี้ทำงานที่ rising หรือ falling clock edge?

* f2. คุณจะออกแบบการทดลองอย่างไรเพื่อ ยืนยันว่า คำตอบของคุณในข้อ f1 ถูกต้อง?

* f3. เขียนโค้ด VHDL สร้าง test bench เพื่อทำซิมมูเลชั่น และสร้าง waveform สนับสนุนคำตอบของคุณ.

(g) ALU control unit.

ตัว ALU control unit ทำหน้าที่ สร้างสัญญาณ operations, Binvert และ CarryIn สำหรับ ALU ตามสัญญาณ ALUop และ instruction bits จาก funct field (ดูตารางที่ 3). แนวคิดคือ แทนที่จะให้ ตัว main control ทำทุกอย่าง, เราจะมอบ หน้าที่การสร้างสัญญาณควบคุม ALU ไปให้กับ ALU control unit. วิธีนี้จะทำให้วงจร main control ซับซ้อนน้อยลง และ ทำให้การทำงานของ main control มีประสิทธิภาพมากขึ้น (สุดท้ายจะทำให้ CPU มีประสิทธิภาพมากขึ้น).

Instruction	m/c code						binary				
	6	5	5	5	5	6	ALUop	funct field	Desired ALU action	ALU control input (Binvert, ALU operation)	CarryIn of LSB
lw rt, addr	0x23	rs	rt	offset			00	xx xxxx	add	010	0
sw rt, addr	0x2B	rs	rt	offset			00	xx xxxx	add	010	0
beq rs, rt, label	4	rs	rt	offset			01	xx xxxx	subtract	110	1
add rd, rs, rt	0	rs	rt	rd	0	0x20	10	10 0000	add	010	0
sub rd, rs, rt	0	rs	rt	rd	0	0x22	10	10 0010	subtract	110	1
and rd, rs, rt	0	rs	rt	rd	0	0x24	10	10 0100	and	000	0
or rd, rs, rt	0	rs	rt	rd	0	0x25	10	10 0101	or	001	0

slt rd, rs, rt	0	rs	rt	rd	0	0x2A	10	10 1010	set on less than	111	1

Table 3 (Adapted from Fig 5.14 of P&H 2e): How the ALU control bits are set depends on the ALUop control bits and the differenct function codes for the R-type instruction. A value of "xxxxxx" means that we "don't care" about the value of the function code.

คำถาม:

g1. สัญญาณอะไรบ้างเป็นอินพุตของ ALU control unit?

g2. สัญญาณอะไรบ้างเป็นเอาต์พุตของ ALU control unit?

g3. สมมติว่า คำสั่ง เป็น r-type (ALUop = "10"), เติมตารางข้างล่างนี้ให้สมบูรณ์.

ALUop		Funct field						specific instr.	Binvert	ALU operation (2 bits)	CarryIn of LSB
ALUop1	ALUop2	F5	F4	F3	F2	F1	F0				
1	0	1	0	0	0	0	0	add	0	10	0
1	0	1	0	0	0	1	0	subtract	1	10	1
1	0	1	0	0	1	0	0				
1	0	1	0	0	1	0	1				
1	0	1	0	1	0	1	0				

Since the ALUop does not use the encoding 11, either value x1 or 1x would not be ambiguous.

g4. เขียน boolean expressions สำหรับตัวแปรเหล่านี้:

Binvert = _____
ALU operation(1) = _____
ALU operation(0) = _____
CarryIn_LSB = _____

g5. ลองรัน `ALUControl_tb` และตรวจสอบดูว่า ALU Control Unit ทำงานถูกต้อง.

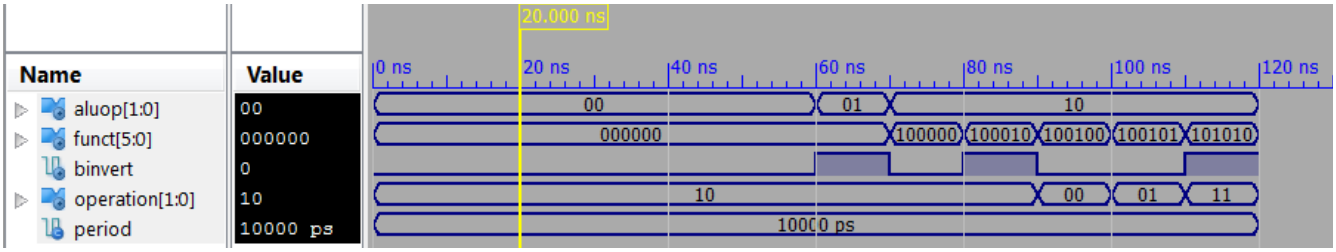


Figure 10: Waveform obtained from ALUControl_tb.

Bonus g6. Propose the alternative architecture of the ALU control to the given architecture (ALUControl process in `datapath.vhd`).
[Yes, there are alternatives to the given code. For example, you can use the boolean expressions in g4 with a multiplexor to define structural architecture of the ALU.]

Bonus g7. Write VHDL code to implement your proposal and make a test bench to verify its functionality.

Bonus g8. Discuss pros and cons of your proposal comparing to the given architecture.

(h) The main control unit.

Questions:
h1. จาก datapath ในรูปที่ 1, สัญญาณอะไรบ้างที่เป็นอินพุตของ main control unit?

h2. จาก datapath ในรูปที่ 1, สัญญาณอะไรบ้างที่เป็นเอาต์พุตของ main control unit?

h3. จากสรุปของขั้นตอนการทำคำสั่ง ในตารางที่ 2, เติมตารางตรรกะของ main control ข้างล่างให้สมบูรณ์:

step	instr.	opcode	PCWrite Cond	PCWrite	IorD	Mem Read	Mem Write	Mem toReg	IRWrite	PCSource	ALU Op	ALU SrcB	ALU SrcA	Reg Write	RegDst	
1	all	'xxxxxx	0	1	0	1	0	x	1	'00	'00	'01	0	0	x	
2	all	'xxxxxx	0	0	x	0	0	x	0	'xx	'00	'11	0	0	x	
3	r-type	'000000	0	0	x	0	0	x	0	'xx	'10	'00	1	0	x	
3	lw/sw	'10x011														
3	beq	'000100														
3	j	'000010														
4	r-type	'000000														
4	lw	'100011														
4	sw	'101011														
5	lw	'100011														
'x' (don't care) represents the logic whose value does not affect the processor functionality. All asserting signals are assumed to be active on '1'.																

bonus h4. How do we implement keep track of the step shown in the table of question h3? (What component do we need? And, how many bits? Draw a diagram illustrating your idea.)

h5. Run `maincontrol_tb` and verify that the component works correctly.
Check all control signals.
What are signals stagein and stageout for?

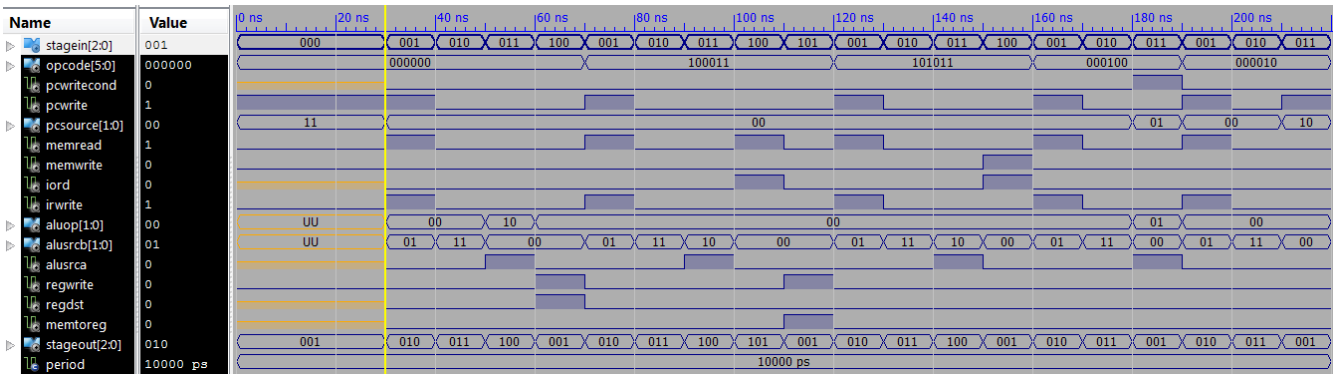


Figure 11: Waveform obtained from maincontrol_tb. The waveform here should be conformed to summary of control signals shown in Table 2. Stage 000 is added to add starting-up capability such that the PC will be loaded with 0x00400000 for starting address. The PCSource multiplexor (in Fig. 5.33 P&H 2e) is implemented with a 4-to-1 multiplexor and the starting address 0x00400000 is mounted at "11" position.

bonus h6. Propose the alternative architecture of the main control to the given architecture (MainControl process in `maincontrol.vhd`).

3. Examine how these components work together in each instruction class.

3.1) Examine simulation results of `Ch5path_tb.vhd`.
The signals shown in the waveform viewer reveal data flow and timing of control signals.
A memory preloaded program to calculate an absolute value of difference between two numbers is shown below.
The program is:

```
                lw $t0, 0($s0)
                lw $t1, 4($s0)
                slt $t2, $t0, $t1
                beq $t2, $zero, LAB_t1ELTt0
                sub $t3, $t1, $t0
                j LAB_SAVE
LAB_t1ELTt0:    sub $t3, $t0, $t1
LAB_SAVE:       sw $t3, 8($s0)
                lw $t0, 8($s0) # to check if it is well-saved in memory
```

and it is hard-coded in ch5mem.vhd from address 0x00400000, which is the MIPS starting address for instructions.

You are encourage to familiarize this program with SPIM. (I, myself, use QtSPIM in development of the program.) It should be noted that the different starting address of the program may lead to different machine code, especially the machine code for an instruction with absolute addressing mode, e.g. jump (`j`).

The data flow of execution of instructions (1) `lw $t0, 0($s0)`, (2) `slt $t2, $t0, $t1`, (3) `beq $t2, $zero, LAB_t1ELTt0`, (4) `j LAB_SAVE`, and (5) `sw $t3, 8($s0)` are shown in Fig. 10 to 14, respectively. These five instructions are picked from instructions coded for the absolute difference calculation program such that each represents different instruction class.

Questions based on Fig. 5.33 (P&H 2e) and Table 2.
3.1.1) Given the PC content is 0x00400000 at stage 1 of any execution, what is the value at the address port of the memory at stage 1?

3.1.2) According to 3.1.1, what content of the PC will be at the end of stage 2?

3.1.3) Suppose the memory contains data as shown in the table below, after fetch and decode an instruction of the address 0x00400000, what value does the Instruction Register (IR) contain?

Address	Data
0x00400000	0x00
0x00400001	0x00
0x00400002	0x08
0x00400003	0x8e
:	:

3.1.4) According to 3.1.3, if the content of the IR is a valid MIPS instruction, what instruction is it?

! recommended bonus ! 3.1.5) Suppose we are to change the functionality of lw/sw instruction so that the offset field refers to a word, not a byte. This will free us from a tedious task of multiplying an index by 4, every time we want to go by word. It sounds good, right? How will you do it?

To have it more fun, you are allowed to change only the datapath, but leave all control signals untouched. No new component is allowed.
(You are pretty much left with options of re-routing signal or signals.)
What would you do?
[Hint: the answer may be much simpler than you think. Now you are forced to think about datapath.]

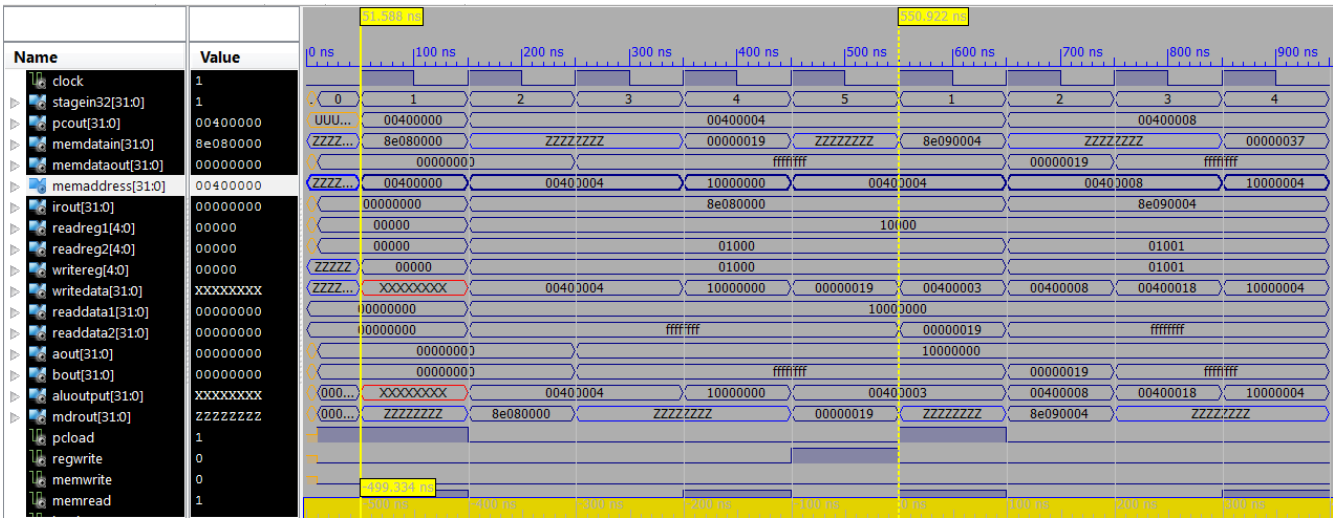


*Figure 12: Waveform obtained from Ch5path_tb.vhd showing signal dynamic of the lw instruction. The second signal from top is **stagein32** indicating which stage the execution is going on. The **pcout** indicates the current value of the PC. The **memdatain**, **memdataout**, and **memaddress** indicate data obtained from the memory, data candidate to put in the memory, and the memory address, respectively. The **irout** indicates the current value of the instruction register (IR). The **readreg1**, **readreg2**, **writereg**, **writedata**, **readdata1**, and **readdata2** are index and data ports of the register file. The **aout**, **bout**, and **aluoutput** indicate current values of registers A, B, and ALUOut respectively. The **mdrout** indicates the current content of register MDR. The **pcload** is a signal enabling overwrite of PC content. [VHDL:* `pcload <= PCWrite or (PCWriteCond and Zero)`*.] The **regwrite**, **memwrite**, and **memread** are signals controlling writing a content of the register file, writting a content of the memory, and reading a content of the memory, respectively. It should be noted that there are more control signals that are not shown here.*

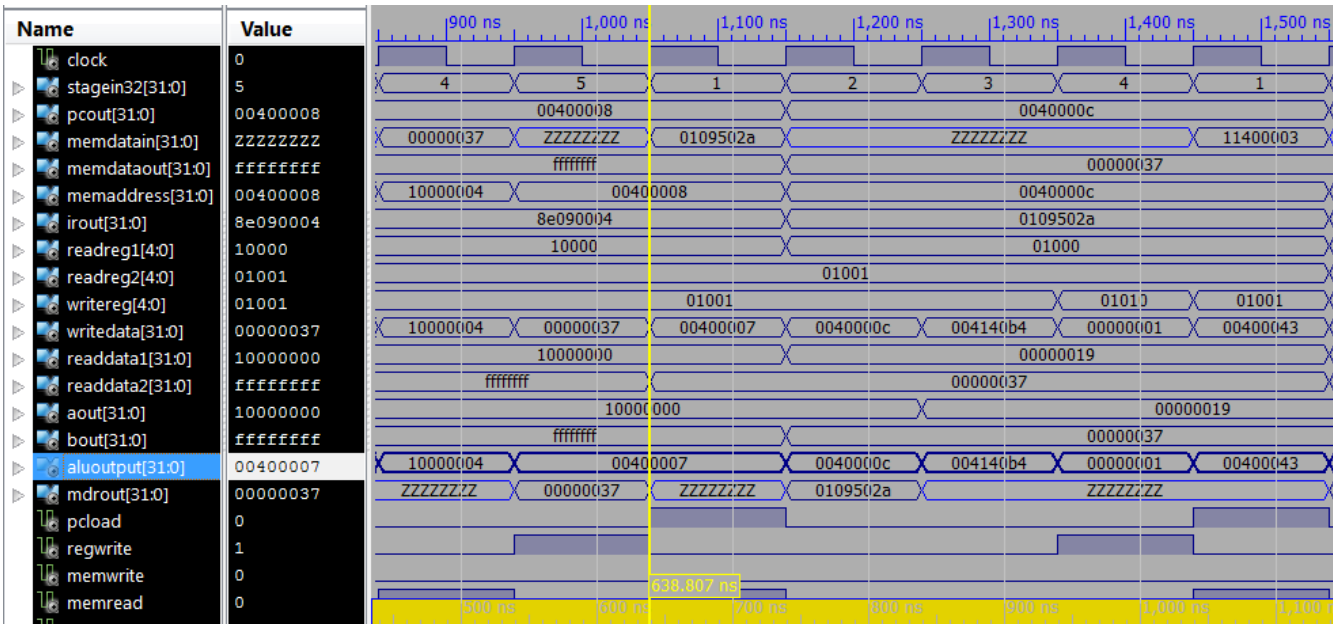


*Figure 13: Data flow of the execution of slt instruction. The yellow line marks the beginning of the execution. In stage 1, instruction is fetched from the momory. The value is at **memdatain**. After stage 1, PC and IR values are updated. In stage 2, values of registers referred by decoded instruction are ready. See **readdata1** and **readdata2**. The contents of registers A and B (**aout** and **bout**) are ready for use in stage 3. The content of ALUOut (**aluputput**) is ready for stage 4 and the **writedata** port of the register file is set to 0x00000001 in the stage 4 of the execution. An R-type instruction such as slt requires 4 stages to complete the execution.*

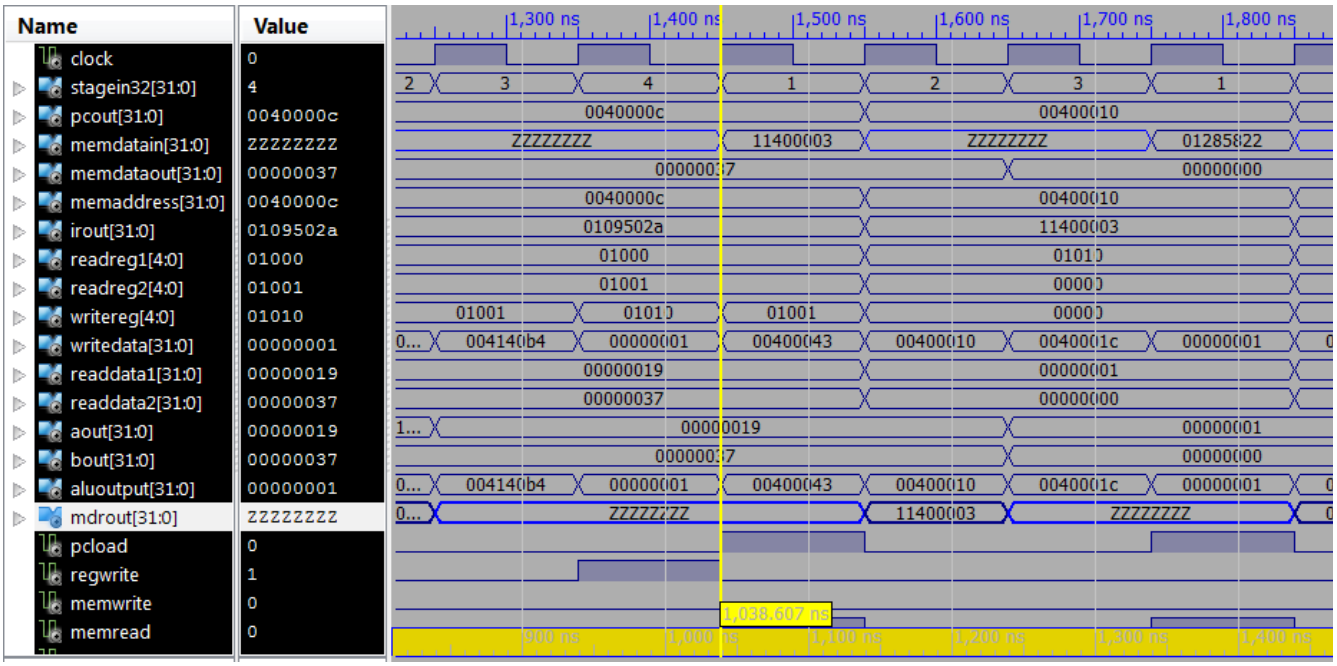
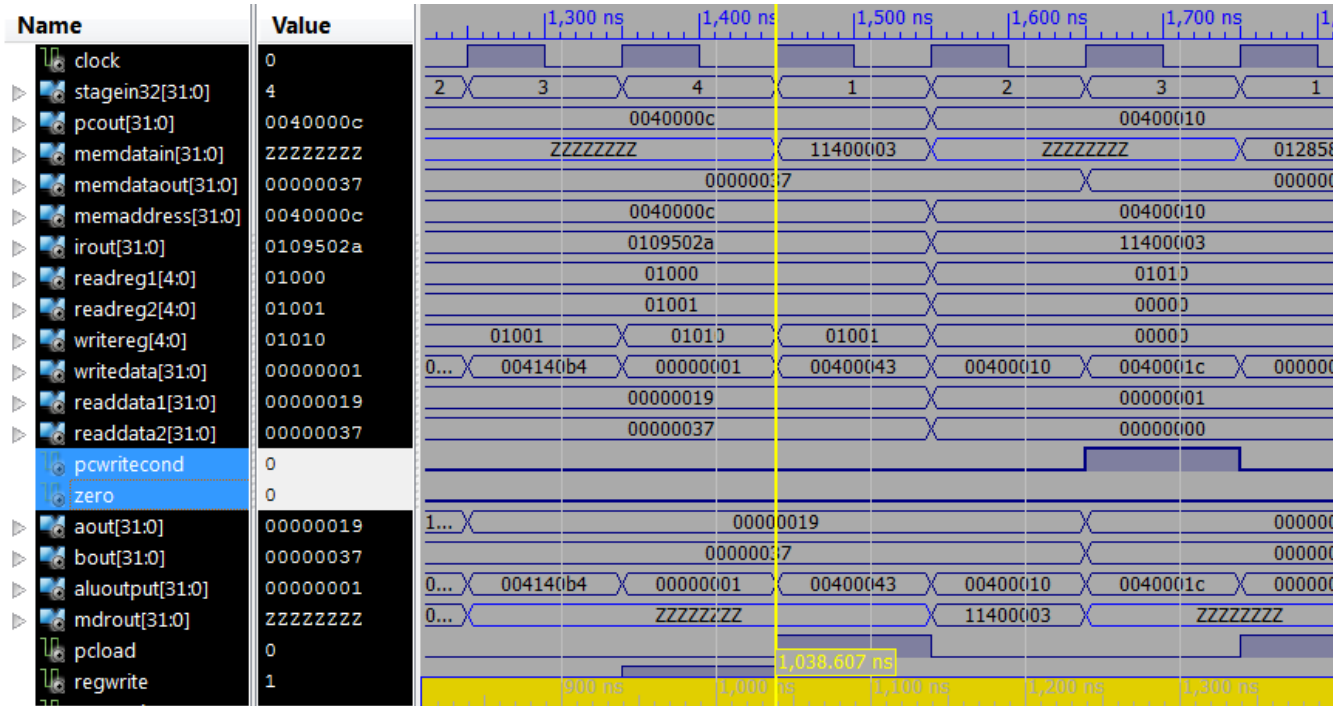


Figure 14: Data flow of the execution of beq instruction. The branch address calculated in stage 2 (0x0040001C) is ready in ALUOut for stage 3. The two values being compared, 0x00000001 and 0x00000000, are loaded into registers A and B whose contents are ready after stage 2. The compared values are not equal, so the branch is not taken and PC keeps its address.



*Supplement 1: Moving **pcwritecond** and **zero** signals upto the viewable area showing **pcwritecond** is asserted at stage 3 allowing change address of the PC if **zero** is '1'.*

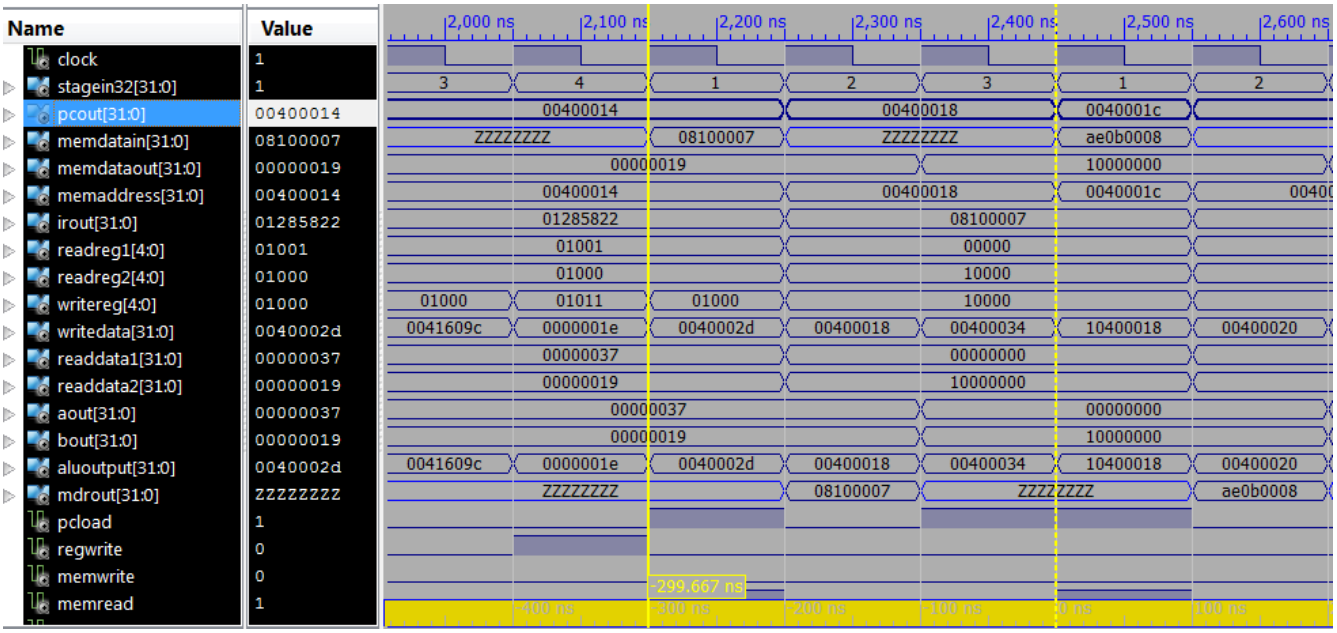


Figure 15: Data flow of the execution of j instruction. In stage 3, pcload is '1', because PCWrite, not shown here, is '1'. The PC is updated after the end of stage 3 and having its new address ready for the next execution, as seen by that 0x0040001C is a content of PC at stage 1 of the following instruction.

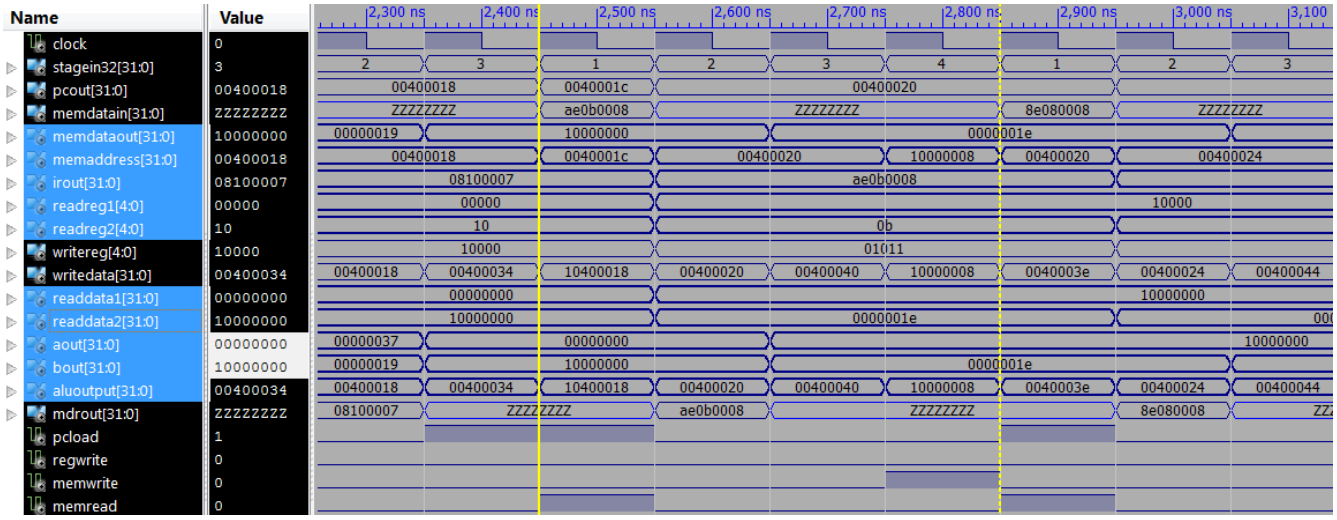


*Figure 16: Data flow of the execution of sw instruction. After stage 2, values of registers A and B, indicated by **aout** and **bout** are 0x10000000 (the base address) and 0x0000001E (the value to be stored). The value of register B is connected to write data port of the memory (**memdataout**). After stage 3, ALUOut (**aluoutput**) is updated and its value is the target address. In stage 4, the address port of the memory is rerouted to the ALUOut and the **memwrite** signal is asserted to write the data into the memory.*

3.2) Your task is to change contents of the memory at addresses 0x10000000 and 0x10000004 to be

0xFFFFFF19 and 0xFFFFFF37.

Questions
(3.2.1) What are the decimal values of 0xFFFFFF19 and 0xFFFFFF37, if they are in 2's complement format?

(3.2.2) With this change, will the branch be taken when we run the same program? Why? (or, if you think the branch will not be taken, why not?)

(3.2.3) With this data, what will be the value to be stored at the address 0x10000008?

(3.2.4) Run the simulation, report the result, and explain what is going on in each stage of the execution of the branch instruction.

(d1) How can you find the branch instruction in a long simulated waveform?
(d2) By looking at the waveform, how can you tell whether the branch is taken or not?

***Bonus* Extend what was given. It's your turn, Little engineer.**

B.1 What aspect of the given implementation do you want to improve? Why and how?
Propose your idea. You don't need to do the VHDL code.

B.2 Now, choose your contribution you like to work on this processor project. Code it in VHDL and make a test bench and verify that it functions as expected.
Hint: there are many things you can do, such as adding capacity to handle other instructions. This does not have to be implementation of your answer to B.1.

B.3 The function of the memory data register.
(a) What is the function of the memory data register (MDR)?
(b) Instead of sticking to the datapath in Fig. 5.33 (P&H 2e) and the control signals as shown in Table 2, what would happen if we bypass the MDR?
This is to connect the MemData directly to the multiplexor controlled with MemtoReg signal.
What are the pros and cons of doing this?
(c) Design experiments to support your answer in 2.2b.
(d) Run your designed experiments, present results, and provide conclusions.

"*It's not about what happened in the past, or what you think might happen in the future. It's about the ride, for Christ's sake. There is no point in going through all this crap, if your are not going to enjoy the ride. And you know what... when you least expect something great might come along. Something better then you even planned for.*" - Irving Feffer, from Along Came Polly (2004).

=== Want more? Try these out. ===
* http://www.xilinx.com/itp/xilinx4/data/docs/xst/hdlcode.html

Closing note

It is a good practice to test every newly created component, so that any flaw in design will be spotted earlier in the development cycle. It's also easier to pin point an error in a single component than trouble shooting unexpected results of an entire system.

Appendix: VHDL codes

ch5path_tb

```
----------------------------------------------------------------------------------
-- Company:       comp. engr. KKU.
-- Engineer:      TK
--
-- Create Date:    17:22:42 09/05/2011
--
-- Module Name:    ch5path_tb - Behavioral
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ch5path_tb is
end ch5path_tb;

architecture Behavioral of ch5path_tb is

-- output of datapath --
signal          clock                          : std_logic;
signal          memdatain               : std_logic_vector(31 downto 0);
signal          memdataout              : std_logic_vector(31 downto 0);
signal          memaddress              : std_logic_vector(31 downto 0);
signal          memwrite                : std_logic;
signal          memread                 : std_logic;
signal          startup                 : std_logic;                                                    -- to
load a start-up setting

------------------------

-- test bench component --

        component ch5mem is
        port
        (
                clock                          : in  std_logic;
                datain                  : in  std_logic_vector(31 downto 0);
                memaddress              : in  std_logic_vector(31 downto 0);
                memwrite                : in  std_logic;
                memread                 : in  std_logic;
                dataout                 : out std_logic_vector(31 downto 0);
                enable                  : in std_logic;
                startup                 : in  std_logic -- to load pre-written data
        );
        end component;

-- original components --


component WordALU is
    Port ( a, b : in  std_logic_vector (31 downto 0);
           Binvert, CarryIn : in  std_logic;
           Operation : in  std_logic_vector (1 downto 0);
           Result : out  std_logic_vector (31 downto 0);
           Overflow, Zero : out  std_logic);
```

```
end component;

component mux2 is
port(
        sel:            in std_logic;
        ch1:            in std_logic_vector(31 downto 0);
        ch2:            in std_logic_vector(31 downto 0);
        muxout: out std_logic_vector (31 downto 0)
);
end component;

component mux4 is
port(
        sel:            in std_logic_vector(1 downto 0);
        ch1:            in std_logic_vector(31 downto 0);
        ch2:            in std_logic_vector(31 downto 0);
        ch3:            in std_logic_vector(31 downto 0);
        ch4:            in std_logic_vector(31 downto 0);
        muxout: out std_logic_vector (31 downto 0)
);
end component;

component nbitregister is
         generic (n : integer := 32);
    Port (
                win:                    in std_logic_vector (31 downto 0);
                clock:          in std_logic;
                regload:        in std_logic;
                regclear:       in std_logic;
                wout:                   out std_logic_vector (31 downto 0)
);
end component;

component regfile is
port(
        readreg1:               in std_logic_vector (4 downto 0);       -- index referring to a specific
register
        readreg2:               in std_logic_vector (4 downto 0);       -- index referring to a specific
register
        writereg:               in std_logic_vector (4 downto 0);       -- index referring to a specific
register
        writedata:              in std_logic_vector (31 downto 0);      -- data to put in the specified
register's content
        readdata1:              out std_logic_vector (31 downto 0);     -- data contained in register indexed by
readreg1
        readdata2:              out std_logic_vector (31 downto 0);     -- data contained in register indexed by
readreg2
        RegWrite:               in std_logic;
        clock:                  in std_logic
);
end component;

component signextend
    port(
         halfword : in  std_logic_vector(15 downto 0);
         fullword : out  std_logic_vector(31 downto 0)
        );
end component;

COMPONENT maincontrol
PORT(
      PCWriteCond : OUT  std_logic;
      PCWrite           : OUT  std_logic;
      PCSource  : OUT  std_logic_vector(1 downto 0);
      MemRead           : OUT  std_logic;
      MemWrite  : OUT  std_logic;
```

```
      IorD                      : OUT  std_logic;
      IRWrite           : OUT  std_logic;
      ALUop             : OUT  std_logic_vector(1 downto 0);
      ALUSrcB           : OUT  std_logic_vector(1 downto 0);
      ALUSrcA           : OUT  std_logic;
      RegWrite  : OUT  std_logic;
      RegDst            : OUT  std_logic;
      MemtoReg  : OUT  std_logic;
      stagein           : IN  std_logic_vector(2 downto 0);
      opcode            : IN  std_logic_vector(5 downto 0);
                stageout        : OUT  std_logic_vector(2 downto 0)
        );
END COMPONENT;


component ALUControl
    port(
         ALUOp          : in  std_logic_vector(1 downto 0);
         funct          : in  std_logic_vector(5 downto 0);
         Binvert                : out  std_logic;
         Operation      : out  std_logic_vector(1 downto 0)
        );
end component;

-------------------------------------------------------------------------
-- signals --
-------------------------------------------------------------------------

signal clk      : std_logic;

-- register file --
signal readreg1 : std_logic_vector (4 downto 0);
signal readreg2 : std_logic_vector (4 downto 0);
signal writereg : std_logic_vector (4 downto 0);        -- we use only 4 bits, but this would allow it for being
used with a 32-bit mux.
signal writedata: std_logic_vector (31 downto 0);
signal readdata1: std_logic_vector (31 downto 0);
signal readdata2: std_logic_vector (31 downto 0);

-- alu --
signal a, b                             : std_logic_vector (31 downto 0);
signal Binvert, CarryIn : std_logic;
signal Operation                        : std_logic_vector (1 downto 0);
signal Result                           : std_logic_vector (31 downto 0);
signal Overflow, Zero   : std_logic;

-- registers --
signal PCin                             : std_logic_vector (31 downto 0);
signal PCload                   : std_logic;
signal PCout, IRout, MDRout, Aout, Bout, ALUoutput              : std_logic_vector (31 downto 0);

-- main control signals --
signal RegWrite : std_logic;
signal IRWrite          : std_logic;
signal ALUSrcB          : std_logic_vector (1 downto 0);
signal PCSource : std_logic_vector (1 downto 0);
signal IorD, RegDst, MemtoReg, ALUSrcA : std_logic;
signal ALUOp            : std_logic_vector (1 downto 0);
signal PCWrite, PCWriteCond     : std_logic;

-- pathway signals --

signal SignEx           : std_logic_vector (31 downto 0);

signal adapterMX0               : std_logic_vector (31 downto 0);
signal adapterMX1               : std_logic_vector (31 downto 0);
```

```
signal adapterMXout     : std_logic_vector (31 downto 0);

signal jaddr            : std_logic_vector (31 downto 0);
signal beqoffset: std_logic_vector (31 downto 0);

signal PCSourceOut      : std_logic_vector (31 downto 0);

signal stagein32, stageout32 : std_logic_vector (31 downto 0);

-- simulation variables --

   -- Clock period definitions
   constant clock_period : time := 100 ns;


begin

        clk <= clock;

        regs: regfile
        port map (
                readreg1        => readreg1,
                readreg2        => readreg2,
                writereg        => writereg,
                writedata       => writedata,
                readdata1       => readdata1,
                readdata2       => readdata2,
                RegWrite        => RegWrite,
                clock                   => clk);

        alu: WordALU
        port map (
                a => a,
                b => b,
                Binvert => Binvert,
                CarryIn => CarryIn,
                Operation => Operation,
                Result => Result,
                Overflow => Overflow,
                Zero => Zero);

        PC: nbitregister
        port map (
                win => PCin,
                clock => clk,
                regload => PCload,
                regclear => '0',
                wout => PCout);

        IR: nbitregister
        port map (
                win => memdatain,
                clock => clk,
                regload => IRWrite,
                regclear => startup,
                wout => IRout);

        MDR: nbitregister
        port map (
                win => memdatain,
                clock => clk,
                regload => '1',
                regclear => startup,
                wout => MDRout);

        Areg: nbitregister
```

```
port map (
        win => readdata1,
        clock => clk,
        regload => '1',
        regclear => startup,
        wout => Aout);

Breg: nbitregister
port map (
        win => readdata2,
        clock => clk,
        regload => '1',
        regclear => startup,
        wout => Bout);

ALUout: nbitregister
port map (
        win => Result,
        clock => clk,
        regload => '1',
        regclear => startup,
        wout => ALUoutput);

mxsrcB: mux4
port map (
        sel             => ALUSrcB,
        ch1             => Bout,                                                -- R-type
        ch2             => x"00000004",                                 -- PC increment
        ch3             => SignEx,                                              -- lw/sw
        ch4             => beqoffset,                                   -- branch address
        muxout  => b);

beqoffset <= SignEx(29 downto 0) & "00";

mxsrcPC: mux4
port map (
        sel             => PCSource,
        ch1             => Result,
                        -- PC <= PC + 4
        ch2             => ALUoutput,
                        -- beq
        ch3             => jaddr,
                        -- j
        ch4             => x"00400000",
                -- boot up address
        muxout  => PCSourceOut);

jaddr <= PCOut(31 downto 28) & IRout(25 downto 0) & "00";

mxIorD: mux2
port map (
        sel             => IorD,
        ch1             => PCout,                       -- instruction access
        ch2             => ALUOutput,           -- memory access (lw/sw)
        muxout  => memaddress);

mxRegDst: mux2
port map (
        sel             => RegDst,
        ch1             => adapterMX0,  -- lw
        ch2             => adapterMX1, -- R-type
        muxout  => adapterMXout);

        adapterMX0 <= "00000000000000000000000000000000" & IRout(20 downto 16);
        adapterMX1 <= "00000000000000000000000000000000" & IRout(15 downto 11);
        writereg <= adapterMXout(4 downto 0);
```

```
        mxMemtoReg: mux2
        port map (
                sel             => MemtoReg,
                ch1             => ALUOutput,   -- R-type
                ch2             => MDRout,              -- lw
                muxout  => writedata);

        mxsrcA: mux2
        port map (
                sel             => ALUSrcA,
                ch1             => PCout,       -- PC increment / branch address
                ch2             => Aout,        -- R-type, lw, sw
                muxout  => a);

        PCin <= PCSourceOut;

        SE: signextend
    port map (
                halfword => IRout(15 downto 0),
       fullword => SignEx);

    alucon: ALUControl
        port map (
          ALUOp => ALUOp,
          funct => IRout(5 downto 0),
          Binvert => Binvert,
          Operation => Operation);

        CarryIn <= Binvert;     -- Both Binvert and CarryIn indicate subtraction (of sub, slt, and comparing for
beq) when set.
                                                                                -- Both are reset for other operations.

    maincon: maincontrol
        port map (
          PCWriteCond => PCWriteCond,
          PCWrite => PCWrite,
          PCSource => PCSource,
          MemRead => MemRead,
          MemWrite => MemWrite,
          IorD => IorD,
          IRWrite => IRWrite,
          ALUop => ALUop,
          ALUSrcB => ALUSrcB,
          ALUSrcA => ALUSrcA,
          RegWrite => RegWrite,
          RegDst => RegDst,
          MemtoReg => MemtoReg,
          stagein => stagein32(2 downto 0),
          opcode => IRout(31 downto 26),
          stageout => stageout32(2 downto 0));

        stageout32(31 downto 3) <= "00000000000000000000000000000";

        statereg: nbitregister
        port map (
                win => stageout32,
                clock => clk,
                regload => '1',
                regclear => startup,
                wout => stagein32);


        -- nodes --
        memdataout <= Bout;
```

```
        readreg1 <= IRout(25 downto 21);
        readreg2 <= IRout(20 downto 16);

        PCload <= PCWrite or (PCWriteCond and Zero);

-- Simulation part --

        mem: ch5mem
        port map (
                clock           => clock,
                datain          => memdataout,
                memaddress      => memaddress,
                memwrite        => memwrite,
                memread         => memread,
                dataout         => memdatain,
                enable          => '1',
                startup         => startup);

   -- Clock process definitions
   clock_process :process
   begin
                clock <= '0';
                wait for clock_period/2;
                clock <= '1';
                wait for clock_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin

                startup <= '0';
                wait for clock_period/10;

                startup <= '1';
      wait for clock_period/10;

                startup <= '0';
                wait;
   end process;

        -- simulation time = 4000 ns
end Behavioral;
```

ALUControl_tb

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   08:59:04 09/06/2011
-- Design Name:
-- Module Name:   C:/xilinx_tutorial/Ch05/ALUControl_tb.vhd
-- Project Name:  Ch05
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: ALUControl
--
-- Dependencies:
--
-- Revision:
```

```
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY ALUControl_tb IS
END ALUControl_tb;

ARCHITECTURE behavior OF ALUControl_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT ALUControl
    PORT(
         ALUOp : IN  std_logic_vector(1 downto 0);
         funct : IN  std_logic_vector(5 downto 0);
         Binvert : OUT  std_logic;
         Operation : OUT  std_logic_vector(1 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal ALUOp : std_logic_vector(1 downto 0) := (others => '0');
   signal funct : std_logic_vector(5 downto 0) := (others => '0');

        --Outputs
   signal Binvert : std_logic;
   signal Operation : std_logic_vector(1 downto 0);
   -- No clocks detected in port list. Replace <clock> below with
   -- appropriate port name

   constant period : time := 10 ns;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
   uut: ALUControl PORT MAP (
          ALUOp => ALUOp,
          funct => funct,
          Binvert => Binvert,
          Operation => Operation
        );

   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 50 ns.
      wait for 50 ns;

               -- lw/sw: add --
               ALUOp <= "00";
               wait for period;
```

```
                -- beq: sub --
                ALUOp <= "01";
                wait for period;

                -- R-type --
                ALUOp <= "10";          funct <= "100000";
                wait for period;

                ALUOp <= "10";          funct <= "100010";
                wait for period;

                ALUOp <= "10";          funct <= "100100";
                wait for period;

                ALUOp <= "10";          funct <= "100101";
                wait for period;

                ALUOp <= "10";          funct <= "101010";
                wait for period;

      wait;
   end process;

END;
```

maincontrol_tb

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   08:08:24 09/06/2011
-- Design Name:
-- Module Name:   C:/xilinx_tutorial/Ch05/maincontrol_tb.vhd
-- Project Name:  Ch05
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: maincontrol
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY maincontrol_tb IS
END maincontrol_tb;
```

```
ARCHITECTURE behavior OF maincontrol_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT maincontrol
    PORT(
         PCWriteCond : OUT  std_logic;
         PCWrite : OUT  std_logic;
         PCSource : OUT  std_logic_vector(1 downto 0);
         MemRead : OUT  std_logic;
         MemWrite : OUT  std_logic;
         IorD : OUT  std_logic;
         IRWrite : OUT  std_logic;
         ALUop : OUT  std_logic_vector(1 downto 0);
         ALUSrcB : OUT  std_logic_vector(1 downto 0);
         ALUSrcA : OUT  std_logic;
         RegWrite : OUT  std_logic;
         RegDst : OUT  std_logic;
         MemtoReg : OUT  std_logic;
         stagein : IN  std_logic_vector(2 downto 0);
         opcode : IN  std_logic_vector(5 downto 0);
         stageout : OUT  std_logic_vector(2 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal stagein : std_logic_vector(2 downto 0) := (others => '0');
   signal opcode : std_logic_vector(5 downto 0) := (others => '0');

        --Outputs
   signal PCWriteCond : std_logic;
   signal PCWrite : std_logic;
   signal PCSource : std_logic_vector(1 downto 0);
   signal MemRead : std_logic;
   signal MemWrite : std_logic;
   signal IorD : std_logic;
   signal IRWrite : std_logic;
   signal ALUop : std_logic_vector(1 downto 0);
   signal ALUSrcB : std_logic_vector(1 downto 0);
   signal ALUSrcA : std_logic;
   signal RegWrite : std_logic;
   signal RegDst : std_logic;
   signal MemtoReg : std_logic;
   signal stageout : std_logic_vector(2 downto 0);
   -- No clocks detected in port list. Replace <clock> below with
   -- appropriate port name

   constant period : time := 10 ns;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
   uut: maincontrol PORT MAP (
          PCWriteCond => PCWriteCond,
          PCWrite => PCWrite,
          PCSource => PCSource,
          MemRead => MemRead,
          MemWrite => MemWrite,
          IorD => IorD,
          IRWrite => IRWrite,
          ALUop => ALUop,
          ALUSrcB => ALUSrcB,
          ALUSrcA => ALUSrcA,
          RegWrite => RegWrite,
          RegDst => RegDst,
```

```
        MemtoReg => MemtoReg,
        stagein => stagein,
        opcode => opcode,
        stageout => stageout
      );

-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   wait for 20 ns;

             stagein <= "000";
   wait for period;

             -- r-type --
   opcode  <= "000000";

             stagein <= "001";
   wait for period;

             stagein <= "010";
   wait for period;

             stagein <= "011";
   wait for period;

             stagein <= "100";
   wait for period;

             -- lw --
   opcode  <= "100011";

             stagein <= "001";
   wait for period;

             stagein <= "010";
   wait for period;

             stagein <= "011";
   wait for period;

             stagein <= "100";
   wait for period;

             stagein <= "101";
   wait for period;

             -- sw --
   opcode  <= "101011";

             stagein <= "001";
   wait for period;

             stagein <= "010";
   wait for period;

             stagein <= "011";
   wait for period;

             stagein <= "100";
   wait for period;

             -- beq --
   opcode  <= "000100";
```

```
                stagein <= "001";
      wait for period;

                stagein <= "010";
      wait for period;

                stagein <= "011";
      wait for period;

                -- j --
      opcode  <= "000010";

                stagein <= "001";
      wait for period;

                stagein <= "010";
      wait for period;

                stagein <= "011";
      wait for period;

                -- simulation time = 300 ns --
      wait;
   end process;

END;
```

WordALU_tb

```
--------------------------------------------------------------------------------
-- Company:             comp. engr. KKU.
-- Engineer:            TK
--
-- Create Date:    09:55:02 08/22/2011
-- Design Name:
-- Module Name:    WordALU_tb - Behavioral
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--   "If you think you can do a thing or think you can't do a thing, you're right."
--   - Henry Ford

--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity WordALU_tb is
end WordALU_tb;

architecture Behavioral of WordALU_tb is

component WordALU is
    Port ( a, b : in  STD_LOGIC_VECTOR (31 downto 0);
```

```
          Binvert, CarryIn : in  STD_LOGIC;
          Operation : in  STD_LOGIC_VECTOR (1 downto 0);
          Result : out  STD_LOGIC_VECTOR (31 downto 0);
          Overflow, Zero : out  STD_LOGIC);
end component;

        signal a, b : STD_LOGIC_VECTOR (31 downto 0);
        signal Binvert, CarryIn : STD_LOGIC;
        signal Operation : STD_LOGIC_VECTOR (1 downto 0);
        signal Result : STD_LOGIC_VECTOR (31 downto 0);
        signal Overflow, Zero : STD_LOGIC;

   constant PERIOD : time := 100 ns;

begin

        walu: WordALU
        port map (
                a => a,
                b => b,
                Binvert => Binvert,
                CarryIn => CarryIn,
                Operation => Operation,
                Result => Result,
                Overflow => Overflow,
                Zero => Zero);

        PROCESS
   BEGIN

                Binvert <= '0';
                CarryIn <= '0';


        -- AND --
                Operation <= "00";      -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)
                a                       <= "00111111111110000011000000011010101";
                b                       <= "01011011110000001010010011110010";
                -- expect:  "00011011110000000010000001100010" = '0x1bc02062'
                WAIT FOR PERIOD;

        -- OR --
                Operation <= "01";      -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)
                a                       <= "00111111111110000011000000011010101";
                b                       <= "01011011110000001010010011110010";
                -- expect:  "01111111111110000111001001111101010" = '0x7ff0e4fa'
                WAIT FOR PERIOD;

        -- ADDITION --
                -- A + B; A > 0, B > 0, no overflow

                Binvert <= '0';         -- A + B
                CarryIn <= '0';
                Operation <= "10";      -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

                a                       <= "00111111111110000011000000011010101";
                b                       <= "00111111111000000101001001111001010";
                -- expect:      "01111111101100010000010101011100" = 0x7fb1055c

                WAIT FOR PERIOD;

                -- A + B; A > 0, B > 0, overflow

                Binvert <= '0';         -- A + B
                CarryIn <= '0';
                Operation <= "10";      -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)
```

```
a                       <= "01111111111100000110000001101010";
b                       <= "00111111110000001010010011110010";
-- expect:      "10111111101100010000010101011100" = 0xbfb1055c

WAIT FOR PERIOD;

-- A + B; A < 0, B < 0, no overflow

Binvert <= '0';         -- A + B
CarryIn <= '0';
Operation <= "10";      -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

a                       <= "11000100011001010011011000000000"; -- = 0xc4653600 = -1000000000
b                       <= "10111011101000100010101101000000"; -- = 0xbba22b40 = -1147000000
-- expect: "d10000000000000111011000010100000"; -- = 0x80076140 = -2147000000

WAIT FOR PERIOD;

-- A + B; A > 0, B < 0

Binvert <= '0';         -- A + B
CarryIn <= '0';
Operation <= "10";      -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

a                       <= "01000100011001010011011000000000"; -- = 0x44653600 =  1147483648
b                       <= "10111011101000100010101101000000"; -- = 0xbba22b40 = -1147000000
-- expect: "?00000000000000111011000010100000"; -- = 0x00076140 = 483648

WAIT FOR PERIOD;

-- A + B; A < 0, B > 0

Binvert <= '0';         -- A + B
CarryIn <= '0';
Operation <= "10";      -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

a                       <= "10111011101000100010101101000000"; -- = 0xbba22b40 = -1147000000
b                       <= "01000100011001010011011000000000"; -- = 0x44653600 =  1147483648
-- expect: "?00000000000000111011000010100000"; -- = 0x00076140 = 483648

WAIT FOR PERIOD;

-- A - B; A > 0, B > 0

Binvert <= '1';         -- A - B
CarryIn <= '1';
Operation <= "10";      -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

a                       <= "00111111111100000110000001101010"; -- = 0x3ff0606a   = 1072717930
b                       <= "00111111110000001010010011110010"; -- = 0x3fc0a4f2   = 1069589746
-- expect:      "00000000001011111011101101111000"              = 0x002fbb78    =    3128184

WAIT FOR PERIOD;

-- A - B; A > 0, B < 0, no overflow

Binvert <= '1';         -- A - B
CarryIn <= '1';
Operation <= "10";      -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

a                       <= "00111111111100000110000001101010"; -- = 0x3ff0606a   = 1072717930
b                       <= "11111111111111101100011110000000"; -- = 0xfffec780   =     -80000
-- expect:      "00111111111100011001100011101010"              = 0x3ff198ea    = 1072797930

WAIT FOR PERIOD;
```

```
		-- A - B; A > 0, B < 0, overflow

		Binvert <= '1';		-- A - B
		CarryIn <= '1';
		Operation <= "10";	-- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

		a			<= "01111111111100000110000001101010"; -- = 0x7ff0606a   = 2146459754
		b			<= "11111111110000101111011100000000"; -- = 0xffc2f700   =   -4000000
		-- expect:  "10000000001011010110100101101010";		= 0x802d696a    ; 2150459754

		WAIT FOR PERIOD;

		-- A - B; A < 0, B < 0

			-- <input code for this test case>

		-- A - B; A < 0, B > 0, no overflow

			-- <input code for this test case>

		-- A - B; A < 0, B > 0, overflow

		Binvert <= '1';		-- A - B
		CarryIn <= '1';
		Operation <= "10";	-- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

		a			<= "11111111110000101111011100000000"; -- = 0xffc2f700   =    -4000000
		b			<= "01111111111100000110000001101010"; -- = 0x7ff0606a   =
	2146459754
		-- expect: "?01111111110100101001011010010110";		= 0x7fd29696    ; -2150459754

		WAIT FOR PERIOD;

	-- LESS (slt) --

		Operation <= "11";	-- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

		-- a < b
		a	<= "00000000000000000000000000000101"; -- 5
		b	<= "00000000000000000000000000011100"; -- 28
		WAIT FOR PERIOD;

		-- a = b
		a	<= "00000000000000000000000000110110"; -- 54
		b	<= "00000000000000000000000000110110"; -- 54
		WAIT FOR PERIOD;

		-- a > b
		a	<= "00000000000000000000000000110110"; -- 54
		b	<= "00000000000000000000000000011100"; -- 28
		WAIT FOR PERIOD;

		-- a < b, a < 0, b < 0
		a	<= "11111111111111111111110111011111"; -- -545
		b	<= "11111111111111111111111011010100"; -- -300
		WAIT FOR PERIOD;

		-- a < b, a < 0, b > 0
		a	<= "11111111111111111111110111011111"; -- -545
		b  <= "00000000000000000000000100101100"; -- 300
		WAIT FOR PERIOD;

		-- simulation time: 1600 + 100 ns

END PROCESS;
```

```
end Behavioral;
```

regfile_tb

```
----------------------------------------------------------------------------------
-- Company:                     comp. engr. KKU.
-- Engineer:                    TK
--
-- Create Date:         08:27:46 09/01/2011
-- Design Name:
-- Module Name:         regfile_tb - Behavioral
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- "Memory can change the shape of a room; it can change the color of a car.
-- And memories can be distorted. They're just an interpretation, they're not a record,
-- and they're irrelevant if you have the facts." - Leonard Shelby in Memento (2000)
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.ALL;

entity regfile_tb is
end regfile_tb;

architecture Behavioral of regfile_tb is

component regfile is
port(
        readreg1:               in std_logic_vector (4 downto 0);       -- index referring to a specific
register
        readreg2:               in std_logic_vector (4 downto 0);       -- index referring to a specific
register
        writereg:               in std_logic_vector (4 downto 0);       -- index referring to a specific
register
        writedata:              in std_logic_vector (31 downto 0);      -- data to put in the specified
register's content
        readdata1:              out std_logic_vector (31 downto 0);     -- data contained in register indexed by
readreg1
        readdata2:              out std_logic_vector (31 downto 0);     -- data contained in register indexed by
readreg2
        RegWrite:               in std_logic;
        clock:                  in std_logic
);
end component;

signal readreg1 : std_logic_vector (4 downto 0);        -- index referring to a specific register
signal readreg2 : std_logic_vector (4 downto 0);        -- index referring to a specific register
signal writereg : std_logic_vector (4 downto 0);        -- index referring to a specific register
signal writedata: std_logic_vector (31 downto 0);       -- data to put in the specified register's content
signal readdata1: std_logic_vector (31 downto 0);       -- data contained in register indexed by readreg1
signal readdata2: std_logic_vector (31 downto 0);       -- data contained in register indexed by readreg2
signal RegWrite : std_logic;
signal clk                      : std_logic;

constant PERIOD         : time := 100 ns;
constant clkperiod      : time := 30 ns;

begin
```

```
	regs: regfile
	port map (
		readreg1	=> readreg1,
		readreg2	=> readreg2,
		writereg	=> writereg,
		writedata	=> writedata,
		readdata1	=> readdata1,
		readdata2	=> readdata2,
		RegWrite	=> RegWrite,
		clock			=> clk);

	process
	begin
		clk <= '0';
		wait for clkperiod;
		clk <= '1';
		wait for clkperiod;
	end process;

	process
	variable regidx: integer :=0;

	begin

		writedata	<= X"01FFAB00";
		writereg	<= "00001";
		wait for PERIOD/2;

		readreg1 <= "00000";
		readreg2 <= "00001";
		wait for PERIOD/2;

		RegWrite	<= '1';
		writedata	<= X"01FFAB00";
		writereg	<= "00001";
		wait for PERIOD/2;

		readreg1 <= "00000";
		readreg2 <= "00001";
		wait for PERIOD/2;

		readreg1 <= "10000";
		readreg2 <= "01000";
		wait for PERIOD/2;

		checkreg0: for regidx in 1 to 5 loop

			writereg <= std_logic_vector(to_unsigned(regidx, writereg'length));
			writedata <= std_logic_vector(to_unsigned(regidx*16, writedata'length));
			wait for PERIOD/2;

			readreg1 <= std_logic_vector(to_unsigned(regidx - 1, writereg'length));
			readreg2 <= std_logic_vector(to_unsigned(regidx, writereg'length));
			wait for PERIOD/2;

		end loop checkreg0;

		-- simulation time = 800 ns

	end process;


end Behavioral;
```

nbitregister_tb

```
----------------------------------------------------------------------------------
-- Company:             comp. engr. KKU.
-- Engineer:            TK
--
-- Create Date:    12:15:13 08/29/2011
-- Design Name:
-- Module Name:    nbitregister_tb - Behavioral
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity nbitregister_tb is
end nbitregister_tb;

architecture Behavioral of nbitregister_tb is

constant Nbits : integer := 32;

component nbitregister is
         generic (n : integer := Nbits);
    Port (
                win:                    in std_logic_vector (Nbits-1 downto 0);
                clock:          in std_logic;
                regload:        in std_logic;
                regclear:       in std_logic;
                wout:                   out std_logic_vector (Nbits-1 downto 0));
end component;

signal win              : std_logic_vector (Nbits-1 downto 0);
signal clk      : std_logic;
signal regload  : std_logic;
signal regclear: std_logic;
signal wout             : std_logic_vector (Nbits-1 downto 0);

constant PERIOD : time := 50 ns;

begin

        reg: nbitregister
        port map (
                win => win,
                clock => clk,
                regload => regload,
                regclear => regclear,
                wout => wout);

        PROCESS
        BEGIN
                clk <= '0';
                wait for PERIOD/2;
                clk <= '1';
                wait for PERIOD/2;
        END PROCESS;

        PROCESS
   BEGIN
                win             <= "11111111000000100000000110100010";
                regload <= '0';
                regclear<= '0';
```

```
                wait for PERIOD/4;

                regload <= '1';
                wait for PERIOD;

                regload <= '0';
                regclear<= '1';
                wait for PERIOD;

                -- simulation time = 150 ns

   END PROCESS;


end Behavioral;
--
```

mux2_tb

```
--
--------------------------------------------------------------------------------
-- Company:                     comp. engr. KKU.
-- Engineer:                    TK
--
-- Create Date:         16:27:46 09/01/2011
-- Design Name:
-- Module Name:         mux2_tb - Behavioral
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- "Memory can change the shape of a room; it can change the color of a car.
-- And memories can be distorted. They're just an interpretation, they're not a record,
-- and they're irrelevant if you have the facts." - Leonard Shelby in Memento (2000)
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux2_tb is
end mux2_tb;

architecture Behavioral of mux2_tb is

component mux2 is
port(
        sel:            in std_logic;
        ch1:            in std_logic_vector(31 downto 0);
        ch2:            in std_logic_vector(31 downto 0);
        muxout: out std_logic_vector (31 downto 0)
);
end component;

signal  sel:            std_logic;
signal  ch1:            std_logic_vector(31 downto 0);
signal  ch2:            std_logic_vector(31 downto 0);
signal  muxout: std_logic_vector (31 downto 0);

constant PERIOD         : time := 100 ns;

begin

        mux: mux2
        port map (
                sel             => sel,
```

```
                ch1             => ch1,
                ch2             => ch2,
                muxout  => muxout);

        process
        begin
                ch1 <= X"0AAA00AA";
                ch2 <= X"0B00000B";

                sel <= '0';
                wait for PERIOD;
                ch1 <= X"1A1A11AA";
                wait for PERIOD/2;
                ch1 <= X"A00A00AA";
                wait for PERIOD/2;

                sel <= '1';
                wait for PERIOD;
                ch2 <= X"0C00000C";
                wait for PERIOD/4;
                ch2 <= X"0D00000D";
                wait for PERIOD/4;

                -- simulation time = 400 ns
        end process;


end Behavioral;
--
```

mux4_tb

```
--
----------------------------------------------------------------------------------
-- Company:                     comp. engr. KKU.
-- Engineer:                    TK
--
-- Create Date:         12:34:46 09/02/2011
-- Design Name:
-- Module Name:         mux4_tb - Behavioral
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- "You cannot dream yourself into a character; you must hammer and forge yourself one."
-- - By the English historian, James Anthony Froude (1818-1894) from his book, The Nemesis of Faith.

----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux4_tb is
end mux4_tb;

architecture Behavioral of mux4_tb is

component mux4 is
port(
        sel:            in std_logic_vector(1 downto 0);
        ch1:            in std_logic_vector(31 downto 0);
        ch2:            in std_logic_vector(31 downto 0);
        ch3:            in std_logic_vector(31 downto 0);
        ch4:            in std_logic_vector(31 downto 0);
        muxout: out std_logic_vector (31 downto 0)
```

```
);
end component;

signal  sel:            std_logic_vector(1 downto 0);
signal  ch1:            std_logic_vector(31 downto 0);
signal  ch2:            std_logic_vector(31 downto 0);
signal  ch3:            std_logic_vector(31 downto 0);
signal  ch4:            std_logic_vector(31 downto 0);
signal  muxout: std_logic_vector (31 downto 0);

constant PERIOD         : time := 100 ns;

begin

        mux: mux4
        port map (
                sel             => sel,
                ch1             => ch1,
                ch2             => ch2,
                ch3             => ch3,
                ch4             => ch4,
                muxout  => muxout);

        process
        begin
                ch1 <= X"10A000AA";
                ch2 <= X"20B0000B";
                ch3 <= X"30C0000C";
                ch4 <= X"40D0000D";

                sel <= "00";
                wait for PERIOD;

                ch1 <= X"111000AA";
                wait for PERIOD/2;

                ch1 <= X"122000AA";
                wait for PERIOD/2;

                sel <= "01";
                wait for PERIOD;

                sel <= "10";
                wait for PERIOD;

                sel <= "11";
                wait for PERIOD;

                -- simulation time = 500 ns
        end process;


end Behavioral;
--
```

signextend_tb

```
--
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   00:25:00 09/05/2011
-- Design Name:
-- Module Name:   C:/xilinx_tutorial/Ch05/signextend_tb.vhd
```

```
-- Project Name:  Ch05
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: signextend
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY signextend_tb IS
END signextend_tb;

ARCHITECTURE behavior OF signextend_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT signextend
    PORT(
         halfword : IN  std_logic_vector(15 downto 0);
         fullword : OUT  std_logic_vector(31 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal halfword : std_logic_vector(15 downto 0) := (others => '0');

        --Outputs
   signal fullword : std_logic_vector(31 downto 0);
   -- No clocks detected in port list. Replace <clock> below with
   -- appropriate port name

   constant PERIOD : time := 10 ns;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
   uut: signextend PORT MAP (
          halfword => halfword,
          fullword => fullword
        );

   -- Stimulus process
   stim_proc: process
   begin

                halfword <= x"80AB";
      wait for 100 ns;
```

```
               halfword <= x"4012";
      wait for 100 ns;

   end process;

END;
--
```

ch5mem_tb

```
--
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    21:05:13 09/04/2011
-- Design Name:
-- Module Name:    ch5mem_tb - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ch5mem_tb is
end ch5mem_tb;

architecture Behavioral of ch5mem_tb is

component ch5mem is
        port
        (
                clock                           : in  std_logic;
                datain                  : in  std_logic_vector(31 downto 0);
                memaddress              : in  std_logic_vector(31 downto 0);
                memwrite                : in  std_logic;
                memread                 : in  std_logic;
                dataout                 : out std_logic_vector(31 downto 0);
                enable                  : in std_logic;
                startup                 : in  std_logic -- to load pre-written data
        );
end component;

signal  clk                     : std_logic;
signal  data                    : std_logic_vector(31 DOWNTO 0);
signal  addr                    : std_logic_vector(31 DOWNTO 0);
signal  memwrite        : std_logic;
signal  memread         : std_logic;
signal  q                               : std_logic_vector(31 DOWNTO 0);
signal  en                              : std_logic;
signal  startup         : std_logic;


constant PERIOD : time := 50 ns;
```

```
begin

        mem: ch5mem
        port map (
                clock           => clk,
                datain          => data,
                memaddress      => addr,
                memwrite        => memwrite,
                memread         => memread,
                dataout         => q,
                enable          => en,
                startup         => startup);

        process
        begin
                clk <= '0';
                wait for PERIOD/2;
                clk <= '1';
                wait for PERIOD/2;
        end process;

PROCESS
   BEGIN
                en      <= '1';

--              startup <= '1';
--              wait for PERIOD/10;
                startup <= '0';
                wait for PERIOD/20;
                startup <= '1';
                wait for PERIOD/10;
                startup <= '0';


                -- test read pre-written data --
                memread <= '1';
                memwrite <= '0';

                addr <= x"10000000";
                wait for PERIOD;

                addr <= x"10000004";
                wait for PERIOD;

                -- test write memory --
                memread <= '0';
                wait for PERIOD/4;

                memwrite <= '1';
                addr <= x"10000000";
                data <= x"00000028";
                wait for PERIOD * 3/4;

                -- test read memory after re-written --
                memwrite <= '0';
                wait for PERIOD/4;

                memread <= '1';
                addr <= x"10000000";
                wait for PERIOD * 3/4;

                -- test read pre-written program --
                memread <= '1';
                addr <= x"00400000";
                wait for PERIOD/2;
```

```
                addr <= x"00400004";
                wait for PERIOD;

                addr <= x"00400008";
                wait for PERIOD;

                addr <= x"0040000C";
                wait for PERIOD;

                addr <= x"00400010";
                wait for PERIOD;

                addr <= x"00400014";
                wait for PERIOD;

                addr <= x"00400018";
                wait for PERIOD;

                addr <= x"0040001C";
                wait for PERIOD;

                addr <= x"00400020";
                wait for PERIOD;

                -- test overwrite program segment --
                memread <= '0';

                addr <= x"00400000";
                data <= x"00000000";
                wait for PERIOD/4;

                memwrite <= '1';
                wait for PERIOD * 3/4;

                memwrite <= '0';
                wait for PERIOD/4;

                memread <= '1';
                wait for PERIOD * 3/4;

                addr <= x"00400004";
                wait for PERIOD/2;

                en <= '0';
                wait for PERIOD/2;

                -- set simulation time to 950ns


   END PROCESS;

end Behavioral;
--
```

Components

regfile.vhd

```
--
----------------------------------------------------------------------------------
-- Company:             comp. eng. KKU.
-- Engineer:            TK
--
-- Create Date:    14:46:23 08/31/2011
```

```
-- Module Name:    regfile - Behavioral
-- Description:  register file
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

entity regfile is

port(
        readreg1:               in std_logic_vector (4 downto 0);       -- index referring to a specific
register
        readreg2:               in std_logic_vector (4 downto 0);       -- index referring to a specific
register
        writereg:               in std_logic_vector (4 downto 0);       -- index referring to a specific
register
        writedata:              in std_logic_vector (31 downto 0);      -- data to put in the specified
register's content
        readdata1:              out std_logic_vector (31 downto 0);     -- data contained in register indexed by
readreg1
        readdata2:              out std_logic_vector (31 downto 0);     -- data contained in register indexed by
readreg2
        RegWrite:               in std_logic;
        clock:                  in std_logic
);
end regfile;

architecture Behavioral of regfile is
        TYPE registers IS ARRAY(0 TO 2 ** 5 - 1) OF std_logic_vector(31 DOWNTO 0);

        SIGNAL reg_block : registers := (
    0  => x"00000000",
    1  => x"FFFFFFFF",
    2  => x"FFFFFFFF",
    3  => x"FFFFFFFF",
    4  => x"FFFFFFFF",
    5  => x"FFFFFFFF",
    6  => x"FFFFFFFF",
    7  => x"FFFFFFFF",
    8  => x"FFFFFFFF",
    9  => x"FFFFFFFF",
    10 => x"FFFFFFFF",
    11 => x"FFFFFFFF",
    12 => x"FFFFFFFF",
    13 => x"FFFFFFFF",
    14 => x"FFFFFFFF",
    15 => x"FFFFFFFF",
    16 => x"10000000",  -- base address for data segment
    17 => x"FFFFFFFF",
    18 => x"FFFFFFFF",
    19 => x"FFFFFFFF",
    20 => x"FFFFFFFF",
    21 => x"FFFFFFFF",
    22 => x"FFFFFFFF",
    23 => x"FFFFFFFF",
    24 => x"FFFFFFFF",
    25 => x"FFFFFFFF",
    26 => x"FFFFFFFF",
    27 => x"FFFFFFFF",
```

```
    28 => x"FFFFFFFF",
    29 => x"FFFFFFFF",
    30 => x"FFFFFFFF",
    31 => x"FFFFFFFF");

BEGIN
        PROCESS (clock, readreg1, readreg2)
        BEGIN

                        readdata1 <= reg_block(to_integer(unsigned(readreg1)));
                        readdata2 <= reg_block(to_integer(unsigned(readreg2)));

                        IF (RegWrite = '1') THEN
                                reg_block(to_integer(unsigned(writereg))) <= writedata;
                        END IF;

        END PROCESS;
END Behavioral;

--
```

WordALU.vhd

```
--
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    08:54:36 08/22/2011
-- Design Name:
-- Module Name:    WordALU - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity WordALU is
    Port ( a : in  STD_LOGIC_VECTOR (31 downto 0);
           b : in  STD_LOGIC_VECTOR (31 downto 0);
           Binvert : in  STD_LOGIC;
           CarryIn : in  STD_LOGIC;
           Operation : in  STD_LOGIC_VECTOR (1 downto 0);
           Result : out  STD_LOGIC_VECTOR (31 downto 0);
           Overflow : out  STD_LOGIC;
                        Zero  : out STD_LOGIC);
end WordALU;
```

```
architecture Behavioral of WordALU is

    COMPONENT BitALU
    Port ( a, b, cin, Binvert, Lessin : in  STD_LOGIC;
           Qo, cout : out  STD_LOGIC;
           operation : in STD_LOGIC_VECTOR (1 downto 0)
                     );
    END COMPONENT;

        component MSBALU
                port (
                                a, b, cin, Binvert, Lessin : in  STD_LOGIC;
                                Qo, set, overflow : out  STD_LOGIC;
                                operation : in STD_LOGIC_VECTOR (1 downto 0)
                );
        end component;

        signal c: STD_LOGIC_VECTOR (31 downto 0);
        signal set : STD_LOGIC;
        signal qr: STD_LOGIC_VECTOR (31 downto 0);

begin

        c(0) <= CarryIn;

        bit0: BitALU port map(
                        a => a(0),
                        b => b(0),
                        cin => c(0),
                        Binvert => Binvert,
                        Lessin => set,
                        Qo => qr(0),
                        cout => c(1),
                        operation => Operation
        );

        bits1to30: for i in 1 to 30 generate
                ibit: BitALU port map(
                        a => a(i),
                        b => b(i),
                        cin => c(i),
                        Binvert => Binvert,
                        Lessin => '0',
                        Qo => qr(i),
                        cout => c(i+1),
                        operation => Operation
                );
        end generate;

        bit31: MSBALU port map (
                        a => a(31),
                        b => b(31),
                        cin => c(31),
                        Binvert => Binvert,
                        Lessin => '0',
                        Qo => qr(31),
                        set => set,
                        overflow => Overflow,
                        operation => Operation
        );

        Result <= qr;

        Zero <= not (
                                qr(0)   or qr(1)        or qr(2)        or qr(3)        or
```

```
                                qr(4)   or qr(5)        or qr(6)        or qr(7)        or
                                qr(8)   or qr(9)        or qr(10)       or qr(11)       or
                                qr(12)  or qr(13)       or qr(14)       or qr(15)       or
                                qr(16)  or qr(17)       or qr(18)       or qr(19)       or
                                qr(20)  or qr(21)       or qr(22)       or qr(23)       or
                                qr(24)  or qr(25)       or qr(26)       or qr(27)       or
                                qr(28)  or qr(29)       or qr(30)       or qr(31) );

end Behavioral;

--
```

nbitregister.vhd

```
--
----------------------------------------------------------------------------------
-- Company:             comp. engr. KKU.
-- Engineer:            TK
--
-- Create Date:    11:53:57 08/29/2011
-- Design Name:
-- Module Name:    nbitregister - Behavioral
-- Description:
--      adapted from http://esd.cs.ucr.edu/labs/tutorial/register.vhd, retrieved on Aug, 29th, 2011.
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments: "Attitude is a little thing that makes a big difference." - Winston Churchill
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity nbitregister is
generic(n: integer := 32);
port(
        win:                    in std_logic_vector (n-1 downto 0);
        clock:          in std_logic;
        regload:        in std_logic;
        regclear:       in std_logic;
        wout:                   out std_logic_vector (n-1 downto 0)
);
end nbitregister;

architecture Behavioral of nbitregister is

        signal wtmp : std_logic_vector (n-1 downto 0);

begin
        process(win, clock, regload, regclear)
        begin
                if regclear = '1' then
                        wtmp <= (wtmp'range => '0');
                elsif (clock = '1' and clock'event) then
                        if regload = '1' then
                                wtmp <= win;
                        end if;
                end if;

        end process;

        wout <= wtmp;

end Behavioral;
--
```

mux4.vhd

```
--
----------------------------------------------------------------------------------
-- Company:             comp. engr. KKU.
-- Engineer:            TK
--
-- Create Date:    11:58:12 09/02/2011
-- Design Name:
-- Module Name:    mux - Behavioral
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- "The idea of wilderness needs no defense. It only needs more defenders."
-- - Edward Abbey
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.NUMERIC_STD.ALL;

entity mux4 is
port(
        sel:            in std_logic_vector(1 downto 0);
        ch1:            in std_logic_vector(31 downto 0);
        ch2:            in std_logic_vector(31 downto 0);
        ch3:            in std_logic_vector(31 downto 0);
        ch4:            in std_logic_vector(31 downto 0);
        muxout: out std_logic_vector (31 downto 0)
);
end mux4;

architecture Behavioral of mux4 is

begin
    SEL_PROCESS: process (sel,ch1,ch2,ch3,ch4)
    begin
      case sel is
        when "00"  => muxout <= ch1;
        when "01"  => muxout <= ch2;
                  when "10"  => muxout <= ch3;
                  when "11"  => muxout <= ch4;
        when others => muxout <= (muxout'range => 'Z');
      end case;
    end process SEL_PROCESS;
end Behavioral;
--
```

mux2.vhd

```
--
----------------------------------------------------------------------------------
-- Company:             comp. engr. KKU.
-- Engineer:            TK
--
-- Create Date:    15:50:12 09/01/2011
-- Design Name:
-- Module Name:    mux - Behavioral
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
```

```
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.NUMERIC_STD.ALL;

entity mux2 is
port(
        sel:            in std_logic;
        ch1:            in std_logic_vector(31 downto 0);
        ch2:            in std_logic_vector(31 downto 0);
        muxout: out std_logic_vector (31 downto 0)
);
end mux2;

architecture Behavioral of mux2 is

begin
    SEL_PROCESS: process (sel,ch1,ch2)
    begin
      case sel is
        when '0'  => muxout <= ch1;
        when '1'  => muxout <= ch2;
        when others => muxout <= (muxout'range => 'Z');
      end case;
    end process SEL_PROCESS;
end Behavioral;
--
```

signextend.vhd

```
--
----------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    00:19:34 09/05/2011
-- Design Name:
-- Module Name:    signextend - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity signextend is
    Port ( halfword : in  STD_LOGIC_VECTOR (15 downto 0);
```

```
          fullword : out  STD_LOGIC_VECTOR (31 downto 0));
end signextend;

architecture Behavioral of signextend is

signal upperhalf : STD_LOGIC_VECTOR (15 downto 0);

begin

        fullword <= upperhalf & halfword;
        upperhalf <= (upperhalf'range => halfword(15));

end Behavioral;

--
```

ALUControl.vhd

```
--
----------------------------------------------------------------------------------
-- Company:             comp. eng. KKU.
-- Engineer:            TK
--
-- Create Date:    08:48:11 09/06/2011
-- Design Name:
-- Module Name:    ALUControl - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ALUControl is

        port(
                ALUOp                   : in std_logic_vector(1 downto 0);
                funct                   : in std_logic_vector(5 downto 0); -- IRout(5 downto 0)
                Binvert         : out std_logic;
                Operation       : out std_logic_vector(1 downto 0) );

end ALUControl;

architecture Behavioral of ALUControl is

begin

        ALUControl: process(ALUOp, funct)
```

```
	begin

		case ALUop is
		when "00" =>
			-- lw/sw: add --
			Binvert <= '0';
			Operation <= "10";
		when "01" =>
			-- beq: sub --
			Binvert <= '1';
			Operation <= "10";
		when "10" =>
			-- R-type --
			case funct is
			when "100000" =>
			-- add --
			Binvert <= '0';
			Operation <= "10";

			when "100010" =>
			-- sub --
			Binvert <= '1';
			Operation <= "10";

			when "100100" =>
			-- and --
			Binvert <= '0';
			Operation <= "00";

			when "100101" =>
			-- or --
			Binvert <= '0';
			Operation <= "01";

			when "101010" =>
			-- slt --
			Binvert <= '1';
			Operation <= "11";

			when others => Binvert <= '0'; Operation <= "00";
			end case;
		when others =>
			-- default case --
			Binvert <= '0';
			Operation <= "00";
		end case;

	end process;

end Behavioral;

--
```

maincontrol.vhd

```
--
----------------------------------------------------------------------------------
-- Company:             comp. engr. KKU.
-- Engineer:            TK
--
-- Create Date:    07:41:53 09/06/2011
-- Design Name:
-- Module Name:    maincontrol - Behavioral
-- Project Name:
```

```
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--      "Truth will set you free." - James A. Garfield
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity maincontrol is

        port(
                PCWriteCond : out  std_logic;
                PCWrite         : out  std_logic;
                PCSource        : out std_logic_vector(1 downto 0);
                MemRead         : out  std_logic;
                MemWrite        : out  std_logic;
                IorD                    : out  std_logic;
                IRWrite         : out  std_logic;
                ALUop           : out std_logic_vector(1 downto 0);
                ALUSrcB         : out std_logic_vector(1 downto 0);
                ALUSrcA         : out  std_logic;
                RegWrite        : out  std_logic;
                RegDst          : out  std_logic;
                MemtoReg        : out  std_logic;
                stagein         : in  std_logic_vector(2 downto 0);
                opcode          : in    std_logic_vector(5 downto 0); -- IRout(31 downto 26)
                stageout        : out std_logic_vector(2 downto 0)      );

end maincontrol;

architecture Behavioral of maincontrol is

begin

        MainControl: process(stagein)

        begin

                case stagein is
                when "000" =>
                        -- boot up --
                        PCWrite <= '1'; PCSource <= "11";
                        MemRead <= '0'; MemWrite <= '0';IRWrite <= '0'; RegWrite <= '0';

                        stageout <= "001";

                when "001" =>
                        -- Instruction Fetch --
                        PCWriteCond <= '0';     PCWrite <= '1'; IorD <= '0';
                        MemRead <= '1';         MemWrite <= '0';MemtoReg <= '0';
                        IRWrite <= '1';         PCSource <= "00";
```

```
        ALUop <= "00";                  ALUSrcB <= "01";        ALUSrcA <= '0';
        RegWrite <= '0';        RegDst <= '0';

        stageout <= "010";
when "010" =>
        -- Instruction Decode & Brach Address Calculation --
        PCWriteCond <= '0';     PCWrite <= '0'; IorD <= '0';
        MemRead <= '0';         MemWrite <= '0';MemtoReg <= '0';
        IRWrite <= '0';         PCSource <= "00";
        ALUop <= "00";                  ALUSrcB <= "11";        ALUSrcA <= '0';
        RegWrite <= '0';        RegDst <= '0';

        stageout <= "011";
when "011" =>

        case opcode is
        when "000000" =>
                -- r-type --
                PCWriteCond <= '0';     PCWrite <= '0'; IorD <= '0';
                MemRead <= '0';         MemWrite <= '0';MemtoReg <= '0';
                IRWrite <= '0';         PCSource <= "00";
                ALUop <= "10";                  ALUSrcB <= "00";        ALUSrcA <= '1';
                RegWrite <= '0';        RegDst <= '0';

                stageout <= "100";

        when "100011" =>
                -- lw --
                PCWriteCond <= '0';     PCWrite <= '0'; IorD <= '0';
                MemRead <= '0';         MemWrite <= '0';MemtoReg <= '0';
                IRWrite <= '0';         PCSource <= "00";
                ALUop <= "00";                  ALUSrcB <= "10";        ALUSrcA <= '1';
                RegWrite <= '0';        RegDst <= '0';

                stageout <= "100";

        when "101011" =>
                -- sw --
                PCWriteCond <= '0';     PCWrite <= '0'; IorD <= '0';
                MemRead <= '0';         MemWrite <= '0';MemtoReg <= '0';
                IRWrite <= '0';         PCSource <= "00";
                ALUop <= "00";                  ALUSrcB <= "10";        ALUSrcA <= '1';
                RegWrite <= '0';        RegDst <= '0';

                stageout <= "100";

        when "000100" =>
                -- beq --
                PCWriteCond <= '1';     PCWrite <= '0'; IorD <= '0';
                MemRead <= '0';         MemWrite <= '0';MemtoReg <= '0';
                IRWrite <= '0';         PCSource <= "01";
                ALUop <= "01";                  ALUSrcB <= "00";        ALUSrcA <= '1';
                RegWrite <= '0';        RegDst <= '0';

                stageout <= "001";

        when "000010" =>
                -- j --
                PCWriteCond <= '0';     PCWrite <= '1'; IorD <= '0';
                MemRead <= '0';         MemWrite <= '0';MemtoReg <= '0';
                IRWrite <= '0';         PCSource <= "10";
                ALUop <= "00";                  ALUSrcB <= "00";        ALUSrcA <= '0';
                RegWrite <= '0';        RegDst <= '0';

                stageout <= "001";
```

```
                when others =>
                        -- unknown operation --
                        stageout <= "000";
                end case;
        when "100" =>
                case opcode is
                when "000000" =>
                        -- r-type --
                        PCWriteCond <= '0';     PCWrite <= '0'; IorD <= '0';
                        MemRead <= '0';         MemWrite <= '0';MemtoReg <= '0';
                        IRWrite <= '0';         PCSource <= "00";
                        ALUop <= "00";                  ALUSrcB <= "00";        ALUSrcA <= '0';
                        RegWrite <= '1';        RegDst <= '1';

                        stageout <= "001";

                when "100011" =>
                        -- lw --
                        PCWriteCond <= '0';     PCWrite <= '0'; IorD <= '1';
                        MemRead <= '1';         MemWrite <= '0';MemtoReg <= '0';
                        IRWrite <= '0';         PCSource <= "00";
                        ALUop <= "00";                  ALUSrcB <= "00";        ALUSrcA <= '0';
                        RegWrite <= '0';        RegDst <= '0';

                        stageout <= "101";

                when "101011" =>
                        -- sw --
                        PCWriteCond <= '0';     PCWrite <= '0'; IorD <= '1';
                        MemRead <= '0';         MemWrite <= '1';MemtoReg <= '0';
                        IRWrite <= '0';         PCSource <= "00";
                        ALUop <= "00";                  ALUSrcB <= "00";        ALUSrcA <= '0';
                        RegWrite <= '0';        RegDst <= '0';

                        stageout <= "001";

                when others =>
                        -- unknown operation --
                        stageout <= "000";
                end case;
        when "101" =>

                if opcode = "100011" then
                        -- lw --
                        PCWriteCond <= '0';     PCWrite <= '0'; IorD <= '0';
                        MemRead <= '0';         MemWrite <= '0';MemtoReg <= '1';
                        IRWrite <= '0';         PCSource <= "00";
                        ALUop <= "00";                  ALUSrcB <= "00";        ALUSrcA <= '0';
                        RegWrite <= '1';        RegDst <= '0';

                end if;

                stageout <= "001";
        when others =>
                -- default --
                stageout <= "000";
        end case;

    end process;

end Behavioral;
--
```

ch5mem.vhd

```
--
----------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    19:18:21 09/04/2011
-- Design Name:
-- Module Name:    ch5mem - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:  to provide interface of 32-bit address/data
--                                              It allows only addresses:
--                                              1000 0000 to 1000 00FF for the data segment
--                                              and 0040 0000 to 0040 00FF for the program segment (full range:
0040 0000-0FFF FFFF)
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;

entity ch5mem is
        port
        (
                clock                           : in  std_logic;
                datain                  : in  std_logic_vector(31 downto 0);
                memaddress              : in  std_logic_vector(31 downto 0);
                memwrite                : in  std_logic;
                memread                 : in  std_logic;
                dataout                 : out std_logic_vector(31 downto 0);
                enable                  : in std_logic;
                startup                 : in  std_logic -- to load pre-written data
        );

end ch5mem;

architecture Behavioral of ch5mem is
        type RAMROM is array(0 TO 2 ** 8 - 1) of std_logic_vector(31 downto 0);
        signal data_block : RAMROM;
        signal program_block: RAMROM;

begin

        process(enable, clock, memread, memwrite, memaddress, startup)

        variable index: integer range 0 to 7;
        begin

                if startup = '1' then
                        -- initialize the ROM contents
                        program_block(0) <= x"8e080000"; -- [00400000]                                  lw $t0,
0($s0)
                        program_block(1) <= x"8e090004"; -- [00400004]                                  lw $t1,
4($s0)
                        program_block(2) <= x"0109502a"; -- [00400008]                                  slt $t2,
$t0, $t1
                        program_block(3) <= x"11400003"; -- [0040000C]                                  beq $t2,
$zero, LAB_t1ELTt0
```

```
                        program_block(4) <= x"01285822"; -- [00400010]                                  sub $t3,
$t1, $t0
                        program_block(5) <= x"08100007"; -- [00400014]                                  j
LAB_SAVE
                        program_block(6) <= x"01095822"; -- [00400018] LAB_t1ELTt0:      sub $t3, $t0, $t1
                        program_block(7) <= x"ae0b0008"; -- [0040001C] LAB_SAVE:        sw $t3, 8($s0)
                        program_block(8) <= x"8e080008"; -- [00400020]                                  lw $t0,
8($s0) # to check if it is well-saved in memory

                        -- initialize the data content
                        -- This may not sound practical, but since we don't have any instruction dealing with a
constant yet.
                        -- So, it is for simplicity sake of homework.
                        data_block(0) <= x"00000019";           -- [10000000]
                        data_block(1) <= x"00000037";           -- [10000004]

-- [10000008]

-- after run, it should has a value, abs( data_block(0) - data_block(1) ) = x"0000001E"

                end if;

                -- 0x10000xXX [0001 0000 0000 0000 0000 00xx xxxx xx--]=> data segment;
                -- 0x00400xXX [0000 0000 0100 0000 0000 00xx xxxx xx--]=> program segment

                index := to_integer(unsigned( memaddress(9 downto 2) ));

                if (enable = '1' and memread = '1') then

                        if memaddress(31 downto 10)             = "0001000000000000000000" then
                                dataout <= data_block(index);
                        elsif memaddress(31 downto 10)  = "0000000001000000000000" then
                                dataout <= program_block(index);
                        else
                                dataout <= (dataout'range => 'Z');
                        end if;

                elsif (enable = '1' and memwrite = '1' and clock'event and clock = '0') then

                        if memaddress(31 downto 10)     = "0001000000000000000000" then
                                data_block(index) <= datain;
                        elsif memaddress(31 downto 10) = "0000000001000000000000" then
                                program_block(index) <= datain;
                        end if;

                        dataout <= (dataout'range => 'Z');

                else
                        dataout <= (dataout'range => 'Z');
                end if;

        end process;

end Behavioral;
--
```