ECE 443

MIPS-16, part 2: Designing a 16-bit RISC style microprocessor

Lab #5: 16-bit ALU and datapath for R-format instructions

Assigned **2/28/2022**
Report due **3/21/2022**

NOTE: The lab assignment is due March 21, but try to finish early, before Spring Break.

Introduction

We are designing a 16-bit microprocessor based on the 32-bit Reduced Instruction Set Computer (RISC) machine that is presented by Patterson and Hennessey in their book "Computer organization and design: the hardware/software interface".

Equipment:
Intel Quartus Prime (aka Altera Quartus)

Procedure:

1. Refresh your knowledge of flip-flops, latches, registers, multiplexers, decoders, etc.
2. Get familiar with the *Quartus* environment. Try to learn how to work faster – use keyboard shortcuts, copy and paste, automatic naming, etc.
3. Carry out attached worksheet and exercises.

MIPS-16:

The processor is a 16-bit machine with the following instruction formats;

```
R-Type   opcode = I[15:12], Rs = I[11:9], Rt = I[8:6], Rd = I[5:3], funct = I[2:0]
I-Type   opcode = I[15:12], Rs = I[11:9], Rt = I[8:6], imm = I[5:0]
J-Type   opcode = I[15:12], adr = I[11:0]
```

The instruction set is given in Table 1.

Worksheet:

1. **ALU**: *Draw a diagram, similar to the one given in Figure C.5.12, showing all the components and connections involved in constructing a 16-bit ALU out of 4-bit ALUs. Use the names given below.*

   (a) Record the timing delays for the **Cyclone IV EP4CE115F29C7** (or Cyclone II EP2C35F672C6) for each of the components created in steps (b) through (g), while testing their functionality. With Intel Prime you will have to use Timing Analyzer for this.

   (b) Create a structural VHDL model of a 4-bit adder **based (for example) on** a commercial *Binary Adder with Fast Carry* – 74LS283 (see the Blackboard for the datasheet). You must use logical gates to implement the adder; using VHDL "add" instruction is not acceptable. Name this entity **ADD4**. (*Hint: use For loops for Ps, Gs, and Ts*)

   (c) Create a 4-bit bitwise AND (**BWAND4**) and a 4-bit bitwise OR (**BWOR4)** components in VHDL.

   (d) Reuse a 4-input 4-bit multiplexer (**MUX4X4**) designed for Lab4.

   (e) Create a 4-bit programmable inverter with a one-bit *control input*, **PINV4**, in VHDL. This inverter will be used for SUB and SLT operations. It should be capable of inverting (or not, based on the *control input*) a 4-bit number, based on the value of one pin (coming from the operation code – see Table 2).

   (f) Use these components to create a 4-bit ALU (**ALU4)**, with inputs A[3:0], B[3:0], LESS, CIN, and SEL[2:0], and outputs F[3:0], COUT, OVERFLOW, SET, and ZERO. The meaning of control signals is given in Table 2. [*Hint: Use MUX4X4 to select ALU output based on SEL signals; use a process to actually send out the result (F) and other output signals (ZERO, OVERFLOW, …) of the ALU4*].

   (g) Use four ALU4s to build a 16-bit ALU (**ALU16)**. The 16-bit ALU has inputs A[15:0], B[15;0], and SEL[2:0], and outputs F[15:0], COUT, OVERFLOW, and ZERO. [*Hint: you can define a signal (for example like this -* `signal NC : STD_LOGIC`;*) and use it to "sink" signals in port map statements that are supposed to be not connected*]

2. Simulations

   Conduct simulations of the ALU and verify its operation for all operations required for MIPS R-type instructions in our Instruction Set. Annotate waveforms to explain results.

Requirements:

1. **Follow the template provided on Blackboard**
2. The report should include printouts of all design files and all simulation files. **Annotate the output waveforms** to explain what is happening during simulation. Provide enough inputs to test all operations, including those involving negative numbers. You can use a single or multiple *.vwf files.
3. *Estimate the maximum clock rate your ALU will be able to sustain if it were clocked.* **Justify your answer.**
4. **Upload all relevant *.vhd and *.vwf files on Blackboard.**

TABLE 1. MIPS-16 INSTRUCTION SET

Instruction	*HDL*	*Type*
add Rd, Rs, Rt	Rd := Rs + Rt	R-type
sub Rd, Rs, Rt	Rd := Rs - Rt	R-type
and Rd, Rs, Rt	Rd := Rt and Rd	R-type
or Rd, Rs, Rt	Rd := Rt or Rd	R-type

slt Rd, Rs, Rt	if (Rs < Rt) then
	Rd = 0001_{16}
	else
	Rd := 0000_{16}
		R-type

lw Rd, imm(Rs)	Rd := M(Rs + imm)	I-type
sw Rd, imm(Rs)	M(Rs + imm) := Rd	I-type
beq Rs, Rt, imm	if(Rs = Rd) then PC := PC + (imm×2)	I-type
j adr	PC := PC[15:13]:adr:0	J-type

TABLE 2. MIPS-16 ALU OPERATIONS

Operation	*SEL2*	*SEL[1:0]*
AND	0	00
OR	0	01
ADD	0	10
SUB	1	10
SLT	1	11

TABLE 3. MIPS-16 Instruction Set and Formats			
Format	**Fields**	**HDL**	**Example**
R-Format	opcode:Rs:Rt:Rd:funct	Rd ← f(Rs, Rt)	`add $1, $2, $3`
I-format	opcode:Rs:Rt:imm	Rt ← M(Rs + imm)	`lw $1,addr($2)`
		if (Rt == Rs) then PC ← PC + (imm×2)	`beq $1,$2, addr`
J-format	opcode:addr	PC ← PC(15..p):addr:'0'	`j addr`

Note: The current value of the PC is the address of the next sequential instruction to be executed.

TABLE 4. MIPS-16 Instruction Opcodes and Function Fields			
Instruction	***Format***	***Hex Opcode***	***Function Field***
`add`	R	0_{16}	010
`sub`	R	0_{16}	110
`and`	R	0_{16}	000
`or`	R	0_{16}	001
`slt`	R	0_{16}	111
`lw`	I	B_{16}	n/a
`sw`	I	F_{16}	n/a
`beq`	I	4_{16}	n/a
`j`	J	2_{16}	n/a

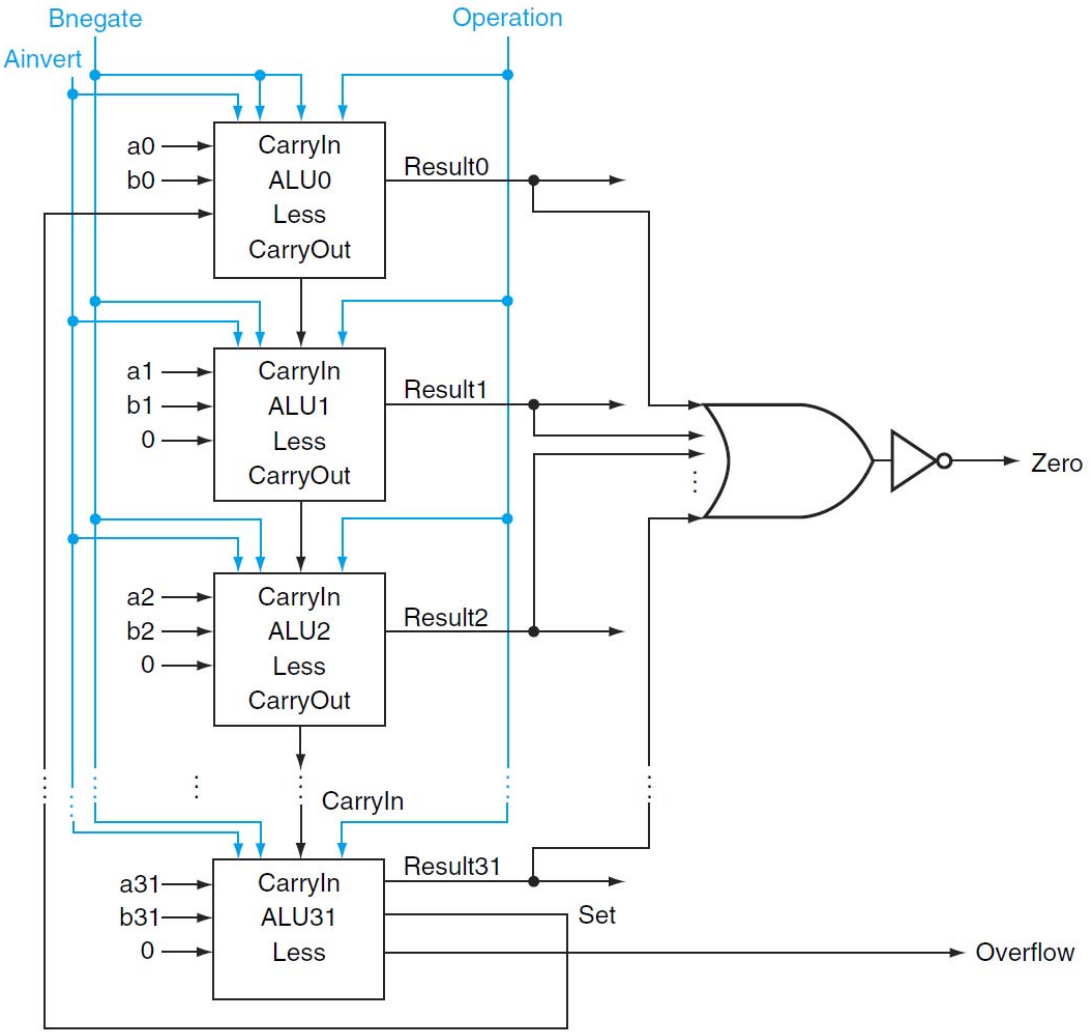

FIGURE C.5.12 The final 32-bit ALU.