EE 469 Lab 4

Designing a Single Cycle CPU

Bring it Together Phase 1….

University of Washington - Department of Electrical Engineering

Matt Staniszewski, Hoon Kwon, Andrew Lawrence, and James K. Peckol

---

Introduction:

In this fourth project, we will bring the hardware modules that we have designed and tested in the previous projects together with some additional capabilities into a simple single-cycle Harvard architecture CPU. We will add the instructions, LW, SW, J, JR, BGE to the set that we worked with in the earlier projects. We will develop the corresponding machine code for each instruction and the necessary signals to control the datapath elements to ensure proper execution.

We will continue to grow our knowledge of the C language by starting with a simple program, hand compiling it then loading and executing it on our CPU. Execution of the program will follow the *fetch, decode, execute, write back, next* instruction cycle.

Prerequisites:

Familiarity with the Quartus development environment and the signal tap logic analyzer. A continued willingness to learn and to explore. No beer until the project is completed and you can turn on a several LEDs. Hey, these are all still good.

Musings:

Are cross purposes really angry or just a little upset? If C and C++ support casting integers and floats, why don't they also support casting aspersions? Can fishermen cast aspersions? Why do the same kinds of animals make different sounds in different countries? Do they really learn to speak the local language? If a Chinese rooster comes to the U.S., does it crow in English or Chinese? Which dialect? Does it have an accent? If you told a French dog to lay down, would it understand? What if you told it to sleep? Why is Donald Duck a duck not a drake? Is it possible to defenestrate an elephant? What would happen if an elephant swallowed a mangel wurtzel then sneezed?

Is a baby bear running through the forest without the bottom part called a bare minimum?

Is it true that in Portugal they call Portuguese babies, Portugoslings?

Is it easier to get ketchup out of a bottle if you are south of the equator where gravity makes things fall down?

In the autumn, do they call geese feathers Fall down?

If we are in Seattle on a really really tall building and can see all the way to Australia, we see that the birds are all flying upside down.

If a bird is flying south for the winter, when do they decide to turn over and fly upside down? How about flying north in the spring?

Life is a challenge…so many questions…

Cautions and Warnings:

Now that we are working with a CPU, always make certain that the cables connecting the PC to the DE1-SoC board are not twisted and don't have any knots. If they are twisted or tangled, the compiled C instructions might get reversed as they are downloaded into the target and your program may run backwards, the source and destination registers may be reversed, and writes will become reads and reads writes. Also, try to keep your DE1-SoC board lower than your PC thus making it easier for the electrons to get to your board and making the data transfer much faster. However, this will not affect the speed at which your program runs because by that point, the program is already at its destination. Once implemented on the target platform, you can increase its speed performance by selectively applying an FPGA grease to lubricate the appropriate module inputs and outputs. Avoid using too much, however otherwise, you may have to add several NOPs to the end of the program to give it enough time to stop before it runs off the end.

Always practice environmentally friendly and safe bit management…don't leave unused bits laying around the lab or as Scott Adams writes in the Dilbert strip.

©Dilbert by Scott Adams – 26 January 2013

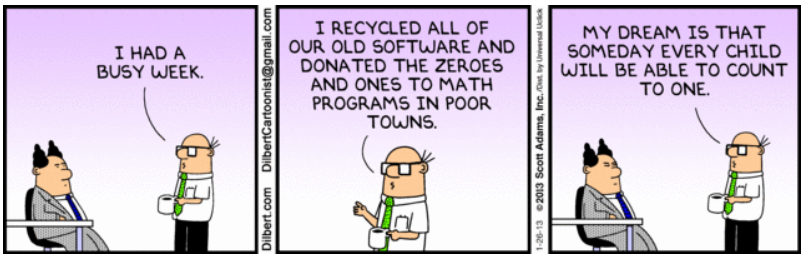

Figure 1 Counting to 1

Background:

For this project, you should have the subsystems from your previous projects fully working. Have a working understanding of memory, registers, and busses and how to move data amongst them. Understand how to compile a C code fragment into an assembly level program utilizing the instructions supported by your design. Understand how to decode an ISA level instruction into the necessary steps and actions necessary to effect the execution of that instruction. Understand finite state machines and how to use them as a tool for controlling the behavior of a digital system. Have a working understanding of basic methods of system I/O such as switches and LEDs.

As we noted in our previous projects, most successful designers begin a design with a high-level architecture for the system they are intending to develop. We now refine and elaborate the architecture that we have discussed in class as given in figure 1.0.

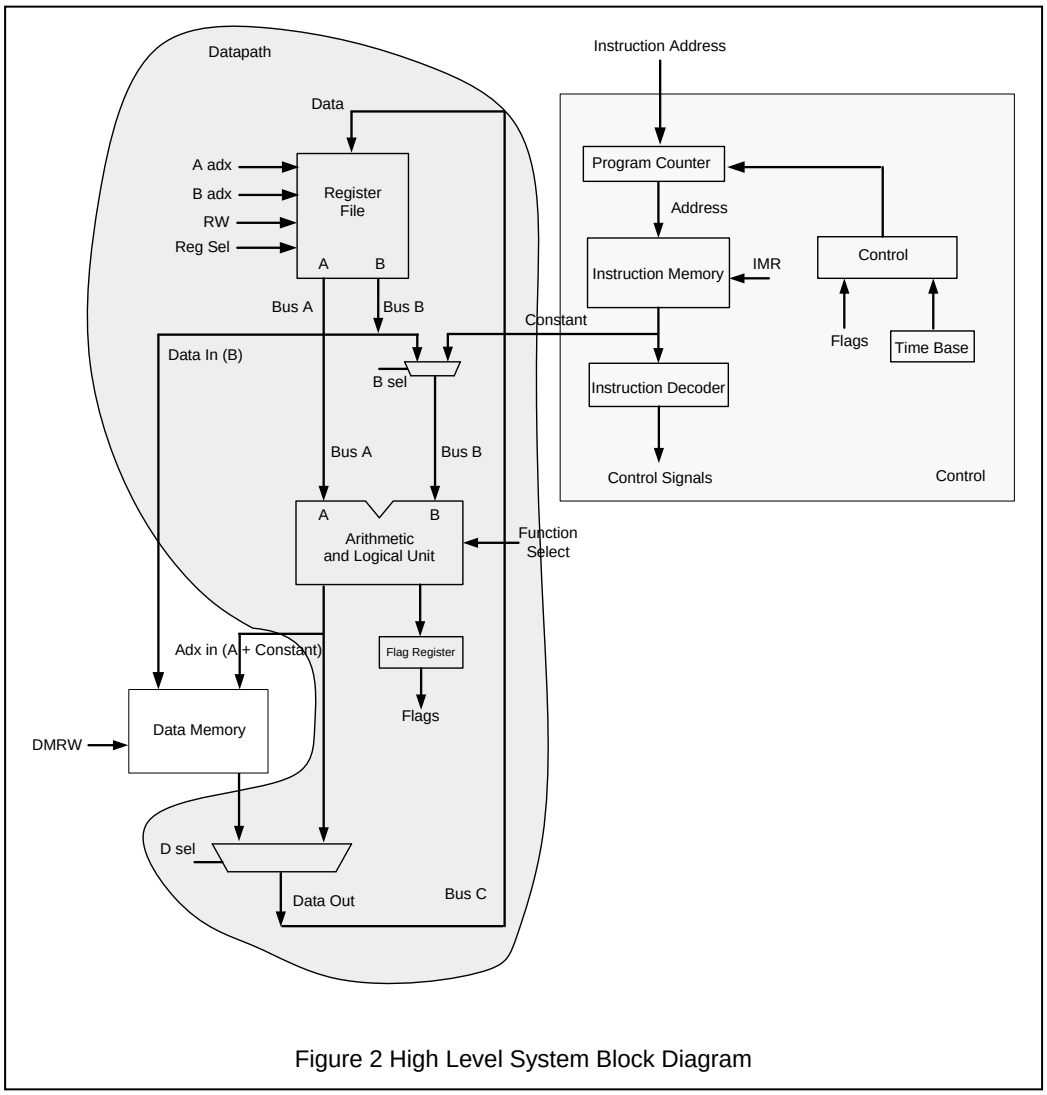

Figure 2 High Level System Block Diagram

Take note, however, this diagram is incomplete and may not necessarily reflect the capabilities of the available hardware components. Before you begin your design, it is highly recommended that you carefully read sections 4.1-4.4 in the Patterson and Hennessy 4th ed. text posted on the class webpage under documentation and go over the data sheets for your components.

Suggestions of things to think about as you're reading….

- ✓ Does the SRAM we are using support the capabilities identified in the diagram?
- ✓ Where does the first instruction address that is loaded into the PC come from?
- ✓ Where do subsequent addresses come from?
- ✓ How do we support branch and jump addresses?
- ✓ How can we support both next address computation and arithmetic or logical operations in a single clock cycle?
- ✓ How does the system support input and output from or to the external world?

Your text may have some answers…

In the previous projects, we completed the design and integration of memory, a register file, and ALU subsystems. In this project, we will now complete the design of a single cycle CPU.

Objectives:

The major objectives of this project include:

- Identify the control steps and actions necessary to support the following instructions: NOP, ADD, SUB, AND, OR, XOR, SLT, SLL, LW, SW, J, JR, BGT.
- Develop the machine code to control the datapath elements to affect each of the control signals and actions to ensure proper execution of each instruction.
- Hand compile a C code fragment into an assembly level program comprising instructions from the supported set.
- Hand compile the assembly level program into machine code.
- Load the machine code into memory then execute the program.

Designing and Implementing the System

1. Based upon the system block diagram given in figure 1.0 above and using the modules that you have designed in the previous projects, complete the design and integration of the single cycle computer.
2. Instruction memory is to be designed and implemented as a 128x32 RAM on the Cyclone V FPGA. You build this on the FPGA just like you did in the first project.
3. Data memory will reside in the SRAM. Words can be stored and accessed as 16 bit quantities for this design, however, they must be sign extended to 32 bits for any operations within the machine.

Designing the Control

1. For each of the instructions in the instruction set given in Table 1.0 below, identify the necessary actions and signals needed to control the elements of the data path to affect the execution of the instruction.
2. Develop the necessary machine code instructions to execute the instruction.
3. Design and develop the instruction decoder and the control block to manage the execution.

Compiling and Executing a C Program:

1. For this part of this project, hand compile the accompanying C code fragment into MIPS assembly language.
2. Next, using the MIPS op codes for each of the instructions given in the adjoining figure, hand assemble the code fragment into machine code for our CPU then load the machine code into the instruction memory on the

```
int A = 7;
int B = 5;
int C = 2;
int D = 4;
int* dPtr = &D;

if (A – B)  > 3
{
      C = 6;
      D = D << 2
}
else
{
      C = C << 5;
      *dPtr  = 7;
}
```

Cyclone V FPGA and the data into the SRAM. These will be our instruction and data memories.

3. Load the starting address of the program into the program counter. Following the basic instruction cycle, *fetch* each instruction from instruction memory, *decode* the op code, *execute* the instruction, *store* any results, determine the address of the *next* instruction, place that address into the program counter and repeat the process until the program terminates.
4. Assign A an initial value of 8 and B an initial value of 4 and repeat steps 1 to 3.

Mnemonic	***Function***
nop	No Operation
add	Add
sub	Subtract
and	Logical AND
or	Logical OR
xor	Exclusive OR
slt	Set Less Than
sll	Shift Left Logical
lw	Load Word
sw	Store Word
j	Jump
jr	Jump Register
bgt	Branch Greater Than

Table 1.0 Instruction Set

Extra Credit:

Implement a 16-word instruction cache on the Cyclone V FPGA. Architect the cache as four 4-word blocks. Your tag table must support the tag, a dirty bit, and a valid bit and utilize a direct mapped design.

You can only receive the extra credit if your core design is fully functional and fully integrated.

Deliverables:

A lab demo showing…

1. A working implementation of a single cycle CPU design that meets the specified requirements.
2. The specified C program working on the CPU.

A lab report containing

1. The annotated Verilog and C source code for all applications both on the DE1-SoC board and on the PC.
2. The annotated assembler and machine code for the code fragment.

3. Representative screen shots showing the results of testing the various designs – from iVerilog or your favourite tool.
4. Representative screen shots showing the logic analyzer outputs for the various designs.
5. Answers to any questions above.
6. Other things that you deem to be important.
7. Anything that we haven’t thought of.