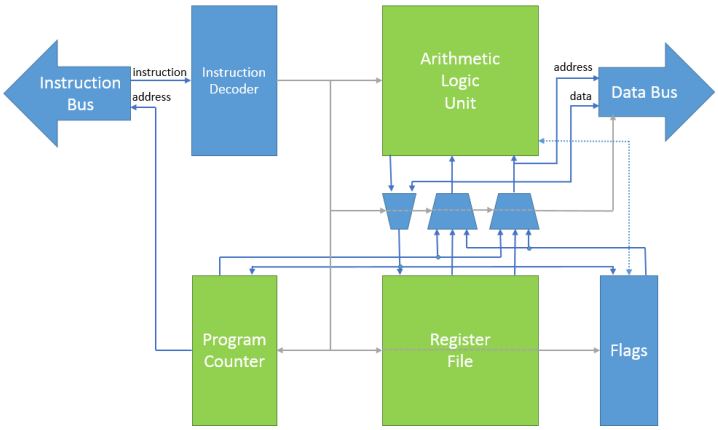
EE20021 Digital Systems Design: week 9 tutorial

C T Clarke

20/11/15

Introduction

In this tutorial session you will put your Arithmetic Logic Unit, Program Counter and Register File together to form a processor like this:



Setup

Download the file set from moodle and create a quartus project in the same folder as the downloaded files using HighRiscSystem as the top level module. You will need to add the following files to the project:

- HighRiscProcessor.sv
- HighRiscSystem.sv
- InstructionSet.sv
- VgaController.sv
- RomBlock.qip
- RAM.qip
- Data.mif
- Program.mif

The .qip and .mif files are used to describe memory blocks in a way that makes it easy to download new files to them. Import system_soc.qsf to set pin locations including KEY 0 which is the reset. This processor system has the following memory and I/O devices:

Module	**Address**	**Size**	**Bus**
Program Memory	0	4000	Instruction
Data Memory	0	4000	Data
VGA display buffer	4000	4000	Data
Switches SW0 - 9	C000	1	Data
LEDs LED0 - 9	C200	1	Data

Activity 1 - Test your modules in a processor

Add your Program Counter, ALU and Register File modules to the project and build the project. If there are any issues, fix them by changing your files. Do not modify the downloaded files. Run compilation and download your system to the De1-Soc. The test program provided in Program.mif copies the switch values to the LEDs. In order for this to work, your modules must have the following functionality:

- Arithmetic Logic Unit: The instructions provided in the first tutorial must still work.
- Program Counter: The output value must count upwards. Reset, LoadEnable, and OffsetEnable do not need to do anything but OffsetEnable should either work or do nothing at all.
- Register file: This module must be fully working.

Activity 2- Change the program

In addition to the files above, you have also been provided with an Assembler and example input file.

The example input assembler file used to create the standard Program.mif file is:

```
// loop.txt
//
// Assembler test - load switch inputs to LEDS
// C. T. Clarke
// 23/11/15
//
LIL R0, 0      // Set R0 to the address of the switches (0xC000)
LIU R0, 0x38
LIL R1, 0      // Set R1 to the Address of the LEDs (0xC200)
LIU R1, 0x08
LIU R1, 0x38
LOAD R2,(R0)   // Get the switch values to R2 - Load instructions
LOAD R2,(R0)   // must be done twice.
STORE (R1),R2  // Copy R2 to the LEDs
```

This is assembled using the 0x0056 mask (more on this later) so that the LIL and LIU instructions are generated from NAND and ROL.

The format of the instructions is:

```
JR Condition, Offset // Optional Comments
```

or:

```
Opcode Destination,Source // Optional Comments
```

The JR opcode causes the offset input of the program counter to be enabled. For this opcode, the condition is one of:

- C – The carry flag is set
- Z – The zero flag is set
- N – The negative flag is set
- P – The parity flag is set
- V – The overflow flag is set
- NC – The carry flag is not set
- NZ – The zero flag is not set
- A – The jump is always taken

The offset is in the range +255 to -256. Note that the offset is relative to the NEXT instruction address. Also note that this processor has a "delay slot" this means that the instruction that follows a jump (or other operation that changes PC) will execute and then the jump will occur.

For the LOAD opcode, the source should be bracketed to indicate that the register holds the address of the memory location to load into the destination. Note also that all LOAD instructions must be issued twice. To account for delays through to the data memory.

For the STORE opcode, the destination should be bracketed to indicate that the register holds the address of the memory location in which the source should be stored.

For other opcodes the opcode is as described in the Instruction set architecture document handed out earlier in the semester. The source and destination are register names: R0 to R62, PC (program counter) or FL (flags).

To run the assembler, open a command window and use the cd command to move to your project directory. Then use the command:

Hra -i *InputFileName* -o *OutputFileName* –m 0xFFFF

If you set the output file name to Progam.mif it will be loaded when you compile the quartus project. It is also possible to use the In-System Memory Content Editor from the tools menu of Quartus to load the program into the PROG memory.

The –m (instruction mask) option makes it possible to use some opcodes even if they are not implemented by constructing them from opcodes that are implemented. Each instruction is indicated by a single bit in the mask based upon its opcode number. For example, LIL and LIU (opcodes 8 and 9 respectively) can be bypassed by using the mask 0xFCFF. The most reduced set possible is 0x0056. Note however, that this process modifies registers in the range 32 to 62 and can break relative jumps. The following instructions cannot currently be bypassed: JR, ADC, SUB, DIV, MOD, MUL, and MUH.

Your task is to try to change the program to make a different LED output. Work in very small steps. For example you could rotate the switch value left by one bit and then output it to the LEDs. If you have implemented LIL and LIU, try re-assembling with those in-place to test them.