ECGR 3183 Spring 2019
Dr. Mihail Cutitaru
Semester Project (25%) – Draft

Assigned: Thursday, February 14, 2018
Due: by Thursday, April 25, 5PM

Project Overview

You are tasked with designing and simulating a floating point co-processor with a single-cycle architecture and a pipelined architecture. For the pipelined architecture, you are to implement 2 branch prediction algorithms (1 static and 1 dynamic) as well as no branch prediction. You are provided with:

- an instruction set
- architecture parameters
- test program(s)

You will provide:

- documented ISA
- architecture and controller design (units, diagrams, etc)
- VHDL/C++/other simulation
- performance results and discussion for pipelined vs. unpipelined approaches

Project Milestones:

Milestones are provided for the project to give you guidelines to completing the work. Milestones will not be graded. However, if you complete a milestone by the designated date, feedback will be provided as quickly as possible, allowing you to catch problems as early as possible. You will also be given an opportunity to schedule time to sit down with me or the TAs and discuss your milestone(s). If you miss the deadline, feedback will be provided if and when the opportunity arises, but will receive lower priority. All work should be submitted electronically on Canvas unless otherwise specified.

Milestone 1	ISA Specification Instruction formats Opcodes Condition codes Architectural features	Feb 28
Milestone 2	Architecture Design Architecture for your designs	Mar 21
Milestone 3	Controller Design Architecture	Apr 4
Milestone 4	Simulation: complete with testing Controller Complete Testing	Apr 18
Final Project	Demo, Results, Final Report	Apr 25

Results submitted for a milestone are not cast in stone. You may rework milestones when working on future milestones as the need arises.

System Parameters:

Data: IEEE single-precision floating point numbers
Register file: 16 registers
Timings:

- Clock cycle (pipelined): 100ns
- Register Read/Write: 100ns
- Memory Read/Write: 300ns
- Single ALU Op: 200ns

Memory:

The system will include a data memory addressed 0-1023 and 16 FP registers (R0...R15). Each memory location and register uses a 32-bit value.

Your simulation will read an input file containing the operational parameters, code, and memory contents.

The file format will be:

<# of instructions in program>

<1st instruction>

…

<last instruction>

<# of memory address contents specified>

<1st memory address>

…

<last memory address>

Example:

```
-- This line is a comment
6          --6 lines of code
Fmul R0, R0, #0
Fadd R0, R0, #3.75
Fmul R1, R1, #0
-- Another random comment
Fadd R1, R1, #26.7
Fmul R3, R0, R1
Halt
0          -- no memory accesses
```

Memory addresses will have the format: <memory address (decimal)> <value (decimal)>, e.g. <10><5.45>.

Should a line in the file contain "--", the rest of the line is a comment until a "newline" character is encountered. Also, lines that start with a "--", are not counted as a line of code. Any comment that extends multiple lines will have "--" at the beginning of each line. This allows for extra comments to be inserted to assist you in understanding the file and the code. For running the program, you may also split this file into multiple files if that assists you with I/O.

Instruction Set:

Instruction	Mnemonic and Operands	Operation	Num. Cycles in ALU
Set	**`Set Ri, #FPvalue`**	Ri ← FPvalue	1
Load	**`Load Ri, Rj`**	Ri ← M[Rj]	1
Store	**`Store Ri, Rj`**	M[Ri] ← Rj	1
Move	**`Move Ri, Rj`**	Ri ← Rj	1
Add	**`Fadd Ri, Rj, Rk`**	Ri ← Rj + Rk	3
Subtract	**`Fsub Ri, Rj, Rk`**	Ri ← Rj – Rk	3
Negate	**`Fneg Ri, Rj`**	Ri ← -Rj	1
Multiply	**`Fmul Ri, Rj, Rk`**	Ri ← Rj * Rk	5
Divide	**`Fdiv Ri, Rj, Rk`**	Ri ← Rj ÷ Rk	8
Floor	**`Floor Ri, Rj`**	Ri ← $\lfloor Rj \rfloor$	1
Ceiling	**`Ceil Ri, Rj`**	Ri ← $\lceil Rj \rceil$	1
Round	**`Round Ri, Rj`**	Ri ← round(Rj)	1
Absolute Value	**`Fabs Ri, Rj`**	Ri ← $\lvert Rj \rvert$	1
Minimum	**`Min Ri, Rj, Rk`**	Ri ← min(Rj, Rk)	1
Maximum	**`Max Ri, Rj, Rk`**	Ri ← max(Rj, Rk)	1
Power	**`Pow Ri, Rj, #integer_value`**	Ri ← $Rj^{integer\_value}$	6
Exponent	**`Exp Ri, Rj`**	Ri ← e^{Rj}	8
Square Root	**`Sqrt Ri, Rj`**	Ri ← $\sqrt{Rj}$	8
Branch (Uncond.)	**`B Ri`**	PC ← M[Ri]	1
Branch Zero	**`BZ Ri, LABEL`**	If (Ri == 0) PC ← LABEL (line)	3
Branch Negative	**`BN Ri, LABEL`**	If (Ri < 0) PC ← LABEL (line)	3
No-op	**`Nop`**	No operation	1
Halt	**`Halt`**	Stop Program	-

Additional Features:

FP Multiply by -1, 1, or 0 takes 1 cycle
FP Multiply by power of 2 takes 2 cycles
Condition Codes: **ZNV** (zero, negative, overflow)

- All condition codes are set as needed on arithmetic operations (pay special attention to the FP ALU operations and results)

The round, ceiling, and floor functions would round up to the nearest integer (not always a power of 2), expressing the result in FP format.

Grading:

ISA: 10%
Architecture: 30%
Controller: 20%
Simulation: 20%
Report/Results: 20%
(includes self- and peer-evaluations)

The intermediate milestones are optional and require a high-level simulation of this system, but the simulation will have to be present in the final submission. Work will be done in teams of 3.

The following is provided as a general outline for your final report. This is not to say that you need to follow this, but it is intended to give you a starting point.

1. Introduction
2. ISA (instruction set, instruction formats)
3. Architecture
   3.1. Datapath
   3.2. Controller
4. VHDL/HLL Description (do not include code, upload your entire project zipped up via Canvas, this should describe the model, any diagrams, etc)
5. Testing
   5.1. Results and discussion on pipelined vs. unpipelined and branch prediction
6. Conclusion

Make sure you focus on your design. Your design is not captured in your VHDL/HLL code, it should be laid out in your report. Figures are excellent for showing your structure. This includes architecture diagrams, state machine diagrams, etc. Your VHDL/HLL code then should reflect these diagrams. Do not just show the waveform when showing waveforms demonstrating the functionality of your architecture. The waveform should be annotated, i.e. put comment blocks on the image with pointers to explain the waveform, or explain it in your write-up.

Your testing section should include a test plan, some sample tests, and the test results. Your tests should be small sections of code that focus on testing one aspect of your architecture, maybe one instruction at first with incremental capabilities.