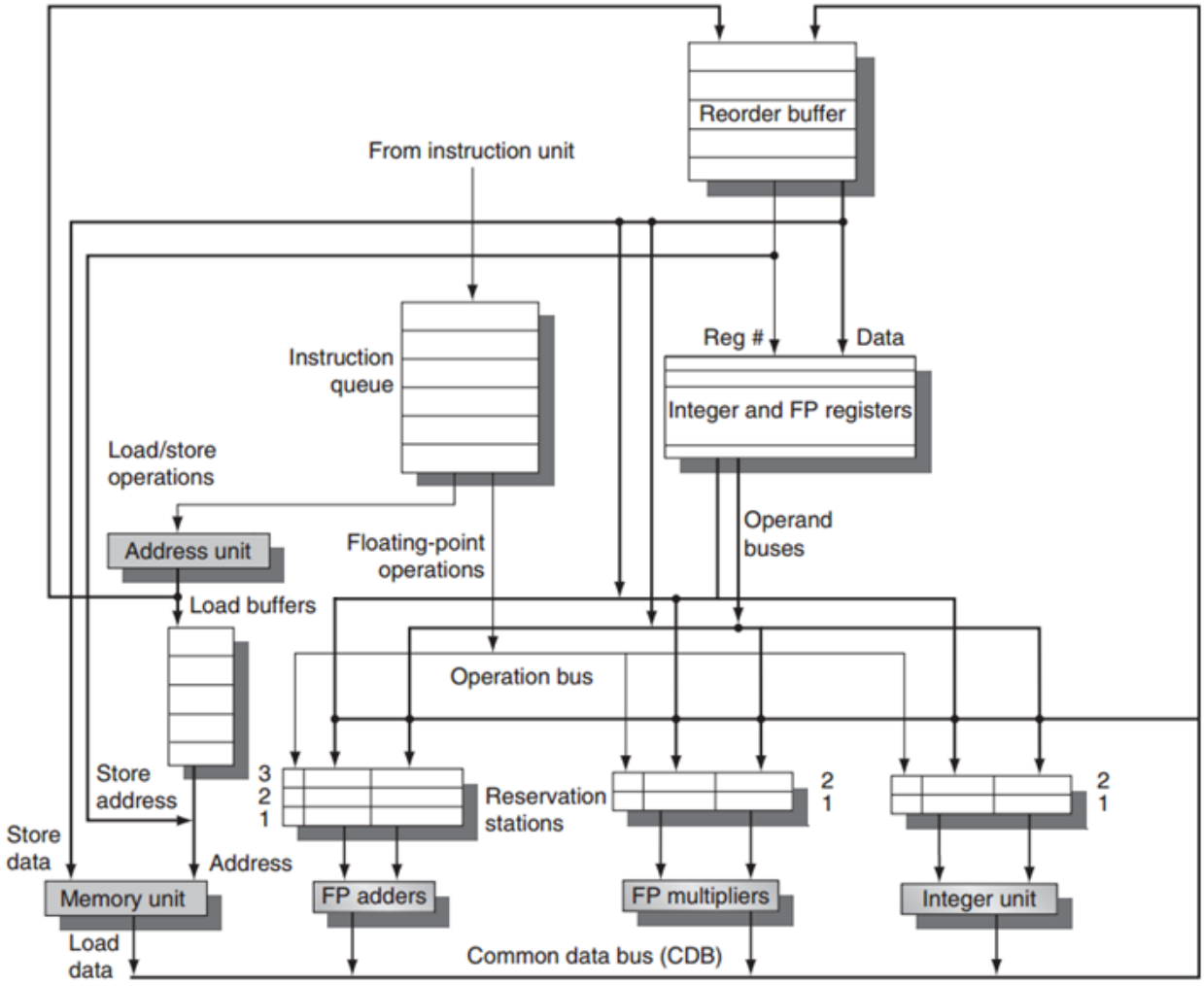
CpE 5110 - Principles of Computer Architecture
Semester Project

Spring 2015
Dr. Mihail Cutitaru

Due: April 28, 2015

This project requires you to simulate a modern superscalar architecture. The goal is to develop a simulation that allows you to study the impact of different branch prediction algorithms on a piece of code. It will be your task to develop your simulation so that you can easily implement different branch prediction algorithms, then choose three algorithms to compare.

This is your starting architecture and you may modify as you deem appropriate. It is suggested that you use Tomasulo's algorithm, though you may choose to use the scoreboard if you wish (you will need to modify the architecture slightly). The architecture below is a single-issue architecture.



Instruction Set:

Instruction	Mnemonics	Operands	Operation	Cycles
FP Addition	FPADD	dest, src1, src2	dest ← src1 + src2	3
FP Subtraction	FPSUB	dest, src1, src2	dest ← src1 – src2	3
FP Multiplication	FPMULT	dest, src1, src2	dest ← src1*src2	5
FP Division	FPDIV	dest, src1, src2	dest ← src1/src2	8
Int Addition	ADD	dest, src1, src2	dest ← src1 + src2	1
Int Subtraction	SUB	dest, src1, src2	dest ← src1 – src2	1
Load	LOAD	dest, src2	dest ← M[src2]	1
Move	MOV	dest, src2	dest ← src2	1
Store	STR	src1, src2	M[src2] ← src1	3
Branch	BR	src2	PC ← PC + src2	1
Branch (> 0)	BGT	src1, src2	if src1 > 0 PC ← PC + src2	1
Branch (< 0)	BLT	src1, src2	if src1 < 0 PC ← PC + src2	1
Branch (≥ 0)	BGE	src1, src2	if src1 ≥ 0 PC ← PC + src2	1
Branch (≤ 0)	BLE	src1, src2	if src1 ≤ 0 PC ← PC + src2	1
Branch (= 0)	BZ	src1, src2	if src1 = 0 PC ← PC + src2	1
Branch (≠ 0)	BNEZ	src1, src2	if src1 ≠ 0 PC ← PC + src2	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)
FP operations are single-precision (32-bits)

Operand Formats:

Operand	Formats	Example
Dest	R0-R15	R3
Src1	R0-R15	R12
Src2	R0-R15, #literal, or address	R5, #23, #4.3, or 43

Memory:
The system will include a data memory addressed 0-1023 and 16 integer/FP registers (R[0]...R[15]). 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
FPMULT R0, R0,#0
FPADD  R0,  R0,#3.75
FPMULT R1, R1,#0
-- Another random comment
FPADD  R1,R1,#26.7
FPMULT 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.

Your tasks:

- Implement the architecture above using your preferred programming language. Make sure each of the units in the architecture is accounted for in your design (no need to worry about the size of the load registers, instruction queue, or ROB). Extensively test your implementation to be able to handle any kind of code combination, with or without memory accesses, with or without comments in any location, and check that your code produces correct results. A few sample input files will be provided to you for testing purposes (in addition to the one above.)
- Implement the architecture above to be able to handle both in-order instruction issue and out-of-order instruction issue. Instruction execution and completion will be out-of-order for both cases. Similar to examples and problems presented in class and homeworks, you need to count the number of cycles each test program takes for both types of issue. Optimize your code as much as possible to be able to finish any program in the minimum amount of cycles (e.g. use register renaming).
- Implement any 3 branch prediction schemes from the ones discussed in class. The more challenging the algorithm, the more points you will receive. Measure performance of each branch prediction scheme on the test cases (provided later).

Deliverables:

Source code and executable(s)
Report on project implementation, documentation/diagrams, explanations, and other details
Branch prediction analysis report (part of the overall report)
Complete peer evaluation forms
Class presentation and demonstration

Extra credit:

Implement interrupts (the details are up to you)