Texas Tech University
Department of Computer Science

Course Name: Computer Architecture
Number: CS3375 **Semester:** Fall 2024
Instructor: Dr. Juan Carlos Rojas **Email:** Juan-Carlos.Rojas@ttu.edu

Final Project

This an individual project. You only need to submit your deliverables on Blackboard. There will be no project presentations.

Submit your code and your report document in Word or PDF formats.

Part 1

Create a program that simulates the scheduling of instructions into a multi-issue processor under certain conditions.

It should take as inputs:

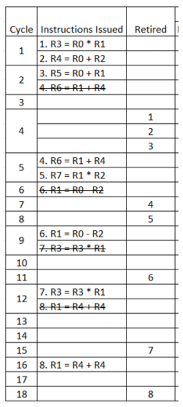
- The input sequence of instructions in assembly
- The number of parallel functional units
- The processor capabilities (to be defined further below)

It should produce a schedule of instructions for each cycle.

For example, the program seen in class:

```
R3 = R0 * R1
R4 = R0 + R2
R5 = R0 + R1
R6 = R1 + R4
R7 = R1 * R2
R1 = R0 - R2
R3 = R3 * R1
R1 = R4 + R4
```

Got scheduled into a 2-issue slot machine with no renaming and in-order execution as:



Cycle	Instructions Issued	Retired
1	1. R3 = R0 * R1	
	2. R4 = R0 + R2	
2	3. R5 = R0 + R1	
	~~4. R6 = R1 + R4~~	
3		
4		1
		2
		3
5	4. R6 = R1 + R4	
	5. R7 = R1 * R2	
6	~~6. R1 = R0 - R2~~	
7		4
8		5
9	6. R1 = R0 - R2	
	~~7. R3 = R3 * R1~~	
10		
11		6
12	7. R3 = R3 * R1	
	~~8. R1 = R4 + R4~~	
13		
14		
15		7
16	8. R1 = R4 + R4	
17		
18		8

Your simulation can display the results of each instruction issued and retired in every cycle.

Program requirements:

- You can write your program in any language. I suggest Python.

Assembly format:

- The assembly input format you can decide.
  - It could be like what is shown in the picture above, or another syntax that you find convenient
- You can assume 8 fixed registers (R0 to R7) are used in the assembly code
- You can assume just a few fixed instructions exist: +, -, *, Load, Store.
  - Don't worry about the memory addresses. Just say R1 = Load, Store = R2, or something like that.

Instruction latencies:

- The instruction latencies can be assumed as follows:
  - +, -: 1 cycle
  - *: 2 cycles
  - Load, Store: 3 cycles
- These refer to the number of cycles that are needed for the instruction to finish. An addition started in cycle 1 will finish in cycle 2. A multiplication started in cycle 1 will complete in cycle 3.

Processor capabilities:

- You will test processors with the following capabilities:
  a. Single instruction, in-order execution
  b. Superscalar, in-order execution
  c. Superscalar, out-of-order issue and retirement
- Assume all functional units can execute all instructions. The number of functional units is configurable from 1 to 3
- For this part we will not test any register renaming capabilities

Dependency checking:

- The scheduler should check for RAW, WAR and WAW dependencies, and delay instructions as appropriate

Testing

Test with the class example code:

```
R3 = R0 * R1
R4 = R0 + R2
R5 = R0 + R1
R6 = R1 + R4
R7 = R1 * R2
R1 = R0 - R2
R3 = R3 * R1
R1 = R4 + R4
```

Also test it with another example code of your creation. Make it about 30 instructions long, and please include all kinds of operations and instructions. Make sure that there are data dependencies of all kinds. The code doesn't need to make sense as a program.

Test in the following scenarios:

1. 1 issue slot, in-order
2. 1 issue slot, out-of-order issue and retirement
3. 2 issue slots, in-order
4. 2 issue slots, out-of-order issue and retirement
5. 3 issue slots, in-order
6. 3 issue slots, out-of-order issue and retirement

Analyze your results and comment on the effectiveness of out-of-order architectures without the use of register renaming.

Part 2

For this part we want to simulate register renaming. You can add up to 8 temporary registers used for renaming purposes. You can assume an unlimited memory window for storing renaming rules.

Re-run the examples in the same configurations as in part 1:

1. 1 issue slot, in-order
2. 1 issue slot, out-of-order issue and retirement
3. 2 issue slots, in-order
4. 2 issue slots, out-of-order issue and retirement
5. 3 issue slots, in-order
6. 3 issue slots, out-of-order issue and retirement

Analyze your results and comment on the effectiveness of register renaming for multi-issue out-of-order architectures.

Code expectations

Your program should have comments, meaningful variable and function names, and use whitespace to facilitate reading.

Your program should work as-is, without any modification. It should be thoroughly tested against a variety of test cases.

Report Document expectations

You should describe your design. You should use diagrams to help explain the design, and some simple examples to illustrate the expected behavior.

You should include some small code samples alongside your explanations of key pieces of the code.

You should describe your test cases and include the results of a few of them.