Homework 3

CDA 4102 / CDA 5155: Fall 2025

Due: October 28, 2025 at **11:30 pm** **Total Points:** 20 points

You are not allowed to take or give help in completing this assignment. Submit the PDF version of the submission in e-learning before the deadline. Late submission (by email attachment to parvath.harikris@ufl.edu) is allowed (up to 24 hours) with a 20% penalty (irrespective of whether it is late for 10 minutes or 10 hours). No grades for late submissions after 24 hours from the deadline. Handwritten (scanned PDF) submissions will NOT be accepted. Please write it in LaTeX or Microsoft Word and convert it to PDF. Please do not take any help from LLM (e.g., ChatGPT) or any other sources. *Please do not include the questions in your solution (PDF) since it affects the plagiarism checker*. Please include the following sentence on top of your submission (PDF):

I have neither given nor received any unauthorized aid on this assignment.

---

1. Consider the following RISC-V instructions. This code sequence does not have delay slot instruction.

```
Loop: fld    f0, 32(x1)
      fld    f2, 64(x1)
      fmul.d f4, f0, f2
      fadd.d f4, f2, f0
      fsd    f4, 64(x1)
      addi   x1, x1, #-8
      bne    x1, x0, Loop
```

The third column in the following table shows the number of cycles of latency between a source instruction (first column) and any subsequent instruction (of any type) consuming the result of the source instructions. The fourth column indicates the number of functional units available for executing the respective type of source instruction.

Function Unit	Related Instruction	Latency Cycles	Number of Units
ALU1	addi, bne	1	1
ALU2	fld, fsd (Address Calculation)	1	1
Memory Unit	fld, fsd	2	2
FP Adder	fadd.d	2	1
FP Multiplier	fmul.d	3	1

Assume that the reservation station and the reorder buffer both have infinite size. ALU1 is used for execution of addi and bne instructions, and ALU2 is used for effective address calculation for load/store instructions. Assume that you can make at most two writes to CDB in one clock cycle. Create two tables showing when each instruction issues, begins execution, accesses memory and writes its result to the CDB for the first two iterations for the following two scenarios using **Tomasulo's algorithm.** Assume that WAW and WAR dependencies are eliminated due to implicit register renaming. Indicate other dependencies (Structural, RAW, etc.) in the entry (if applicable). For fld and fsd operations, the "Execute" stage is used by ALU2 (address calculation). "Memory" column implies memory access and therefore it will be blank for all instructions except for fld and fsd.

a) [5] Use a RISC-V pipeline with **two-issue** and **without speculation**. Assume branch prediction is perfect. Some of the entries are filled already to show how to indicate the dependencies/hazards. You need to fill the remaining entries. [*Slide 91 of paralleism.ppt solves a similar problem with different set of assumptions.*]

Iter.	Instruction		Issue	Execute	Memory	Write-CDB
1	fld	f0, 32(x1)	1	2	3-4	5
1	fld	f2, 64(x1)	1	3 (Structural)	4-5	6
1	fmul.d	f4, f0, f2	2	7-9 (RAW)		10
1	fadd.d	f4, f2, f0				
1	fsd	f4, 64(x1)				
1	addi	x1, x1, #-8				
1	bne	x1, x0, Loop				
2	fld	f0, 32(x1)				
2	fld	f2, 64(x1)				
2	fmul.d	f4, f0, f2				
2	fadd.d	f4, f2, f0				
2	fsd	f4, 64(x1)				
2	addi	x1, x1, #-8				
2	bne	x1, x0, Loop				

b) [5] Use a RISC-V pipeline with **two-issue** and **with speculation**. You also need to specify when each instruction commits. Please follow in-order commit for instructions. Assume that up to two instructions of any type can commit per cycle. Branch prediction is perfect. Note that stores will spend 2 cycles in the commit stage, because its memory access occurs during commit. Some of the entries are filled already to show how to indicate the dependencies/hazards. You need to fill the remaining entries. [*Slide 92 of parallelsim.ppt solves a similar problem with different set of assumptions.*]

Iter	Instruction		Issue	Execute	Memory Read	Write-CDB	Commit
1	fld	f0, 32(x1)	1	2	3-4	5	6
1	fld	f2, 64(x1)	1	3 (Structural)	4-5	6	7
1	fmul.d	f4, f0, f2	2	7-9 (RAW)		10	11
1	fadd.d	f4, f2, f0					
1	fsd	f4, 64(x1)					
1	addi	x1, x1, #-8					
1	bne	x1, x0, Loop					
2	fld	f0, 32(x1)					
2	fld	f2, 64(x1)					
2	fmul.d	f4, f0, f2					
2	fadd.d	f4, f2, f0					
2	fsd	f4, 64(x1)					
2	addi	x1, x1, #-8					
2	bne	x1, x0, Loop					

Problem 2 [2+2] Assume a sequence of memory requests (word addresses) are: A, B, C, A, B, C, A, B, C. Assume that the cache is fully-associative and has only two blocks (lines). Each block can store only one word. **Simulate** (complete the entries in the table) both least-recently used (LRU) and most-recently-used (MRU) replacement policy. **Compute** the hit rate to **identify** which policy is better for this sequence?

Use **LRU** Policy

	A	B	C	A	B	C	A	B	C
Block 1	A	A							
Block 2		B							
Hit/Miss	Miss	Miss							

Hit Rate = # of hits / # of accesses =

Use **MRU** Policy

	A	B	C	A	B	C	A	B	C
Block 1	A	A							
Block 2		B							
Hit/Miss	Miss	Miss							

Hit Rate = # of hits / # of accesses =

Problem 3 [2+2+2] Consider a cache design with cache size 32,768 bytes, block size 8 words, and word size 4 bytes. Determine the **tag size** (number of tag bits per block) for three cache designs (direct-mapped, 4-way set-associative, and fully associative) on a computer with 32-bit address. Show major steps. *Please refer to page B-7 (Appendix-B) of the textbook or slides 10-13 (cache.pptx) to learn about different cache implementations and their effect on tag size.*

Direct:

Block size = 8 x 4 = 32 bytes ➔ Block-offset = 5 bits

Number of blocks =

Tag =

4-way Set Associative:

Fully-Associative: