1. Virtual Memory and Caches [50 points]

Assume that we have a byte-addressable processor that implements paging-based virtual memory using a **three-level** hierarchical page table organization. The virtual address is **46 bits**, and the physical memory is **4GB**. The page table base register (PTBR) stores the base address of the first-level table (*PT1*). All the page tables have the same size of **one physical page** for the first-level (*PT1*), second-level (*PT2*), and third level (*PT3*). Each *PT1*/*PT2* entry stores the base address of a second/third-level table (*PT2*/*PT3*). In contrast to *PT1* and *PT2*, each *PT3* entry stores a page table entry (PTE). The PTE is 4-bytes in size.

The processor has a 64KB **virtually-indexed physically-tagged (VIPT)** L1 cache that is direct mapped with a cache line size of 128 bytes, and a 64-entry TLB.

(a) What is the physical page size? Show all your work.

1. First, the physical address space is $\log_2 4GB = 32bits$. So each PT1 and PT2 entry stores 32bits (4bytes).
2. Since the virtual address space is 46 bits and assume the page size is P, then there are $\frac{2^{46}}{P}$ possible mappings that the page tables can store.
3. Each page table can store $\frac{P}{4}$ mappings because each entry is 4 bytes and the table is P bytes. The three-level hierarchical page table can in total store $(\frac{P}{4})(\frac{P}{4})(\frac{P}{4}) = \frac{2^{46}}{P}$ mappings.
4. $P = 2^{13}$.

(b) How many bits of the virtual page number are used to index the L1 cache?

The cache requires $\log_2 64KB = 16$ bits from the address for indexing. So 3 bits are overlapped with the VPN.

(c) What kind of aliasing problem does this processor's L1 cache have?

Synonym – multiple different virtual addresses map to the same physical address, causing multiple copies of the data belonging to the same physicall address residing in the cache.

(d) In lecture, we learned multiple techniques to resolve this particular aliasing problem (in part (c) above) in a VIPT cache. One of them is to increase the associativity of the cache. To address this aliasing problem for this processor's VIPT cache, what is the minimum associativity that is required for the cache? Show your work.

$2^3 = 8$

(e) We also learned another technique that searches all possible sets that can contain the same physical block. For this VIPT cache, how many sets need to be searched to fix the aliasing problem? Show your work.

$2^3 = 8$

2. Register Renaming [50 points]

In this problem, we will give you the state of the Register Alias Table (RAT), Reservation Stations (RS), and Physical Register File (PRF) for a Tomasulo-like out-of-order execution engine.

The out-of-order machine in this problem has the following characteristics:

- The processor is fully pipelined with four stages: Fetch, decode, execute, and writeback.
- For all instructions, fetch takes 1 cycle, decode takes 1 cycle, and writeback takes 1 cycle.
- The processor implements `ADD` and `MUL` instructions only. Both the adder and multiplier are fully pipelined. `ADD` instructions take 3 cycles and `MUL` instructions take 4 cycles in the execute stage. Note that the adder and multiplier have separate common data buses (CDBs), which allow both the adder and multiplier to broadcast results in the same cycle.
- An instruction always allocates the first reservation station that is available (in top-to-bottom order) at the required functional unit.

Suppose the pipeline is initially empty and the machine fetches exactly 5 instructions. The diagram below shows the snapshot of the machine at a particular point in time.

Register Alias Table

ID	V	Tag
R0	1	P1
R1	1	P8
R2	1	P12
R3	1	P4
R4	0	P7
R5	1	P5
R6	0	P11
R7	1	P14

Physical Register File

ID	V	Data	ID	V	Data
P0	0	2	P8	1	88
P1	1	11	P9	0	90
P2	0	2	P10	0	91
P3	0	30	P11	1	110
P4	1	3	P12	1	33
P5	1	50	P13	1	130
P6	0	5	P14	1	17
P7	1	70	P15	1	159

`ADD` **Reservation Station**

ID	V	Tag	V	Tag	Dest. Tag
A	1	P12	1	P7	P15
B	1	P5	0	P13	P11

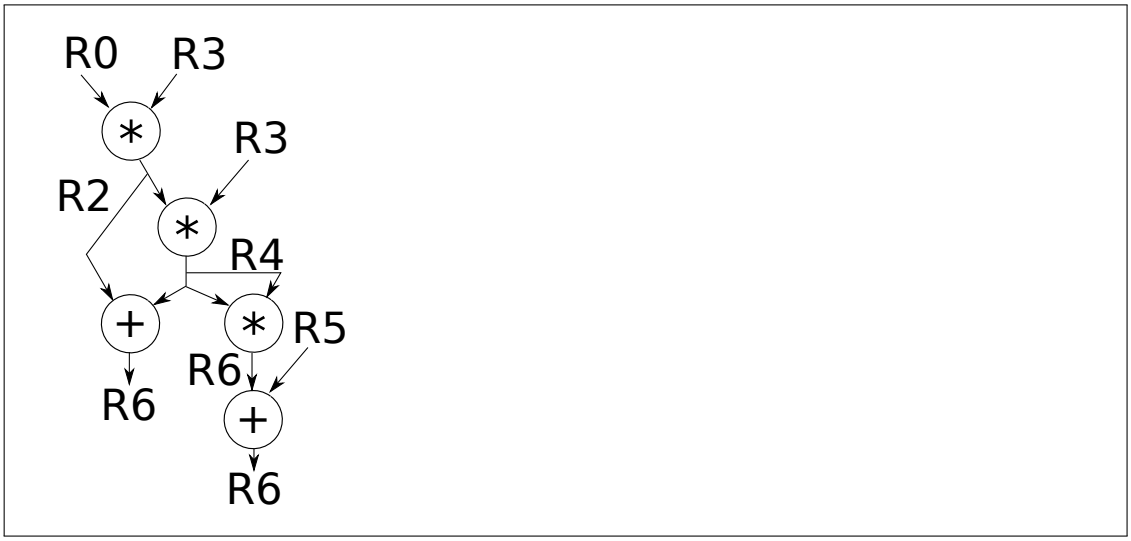
`ADD` **CDB**

`MUL` **Reservation Station**

ID	V	Tag	V	Tag	Dest. Tag
X	0	–	0	–	–
Y	1	P12	1	P4	P7
Z	1	P7	1	P7	P13

`MUL` **CDB**

(a) Your first task is to use only the supplied information to draw the data flow graph for the five instructions which have been fetched. Label nodes with the operation (+ or $\star$) being performed and edges with the architectural register alias numbers (e.g., R0).



(b) Now, use the data flow graph to fill in the table below with the five instructions being executed on the processor in program order. The source registers can be specified in either order. Give instructions in the following format: "opcode, source1, source2, *destination*."

OP	Src 1	Src 2	Dest
MUL	R0	R3	R2
MUL	R2	R3	R4
ADD	R2	R4	R6
MUL	R4	R4	R6
ADD	R5	R6	R6

(c) Now, show the full pipeline timing diagram below for the sequence of five instructions that you determined above, from the fetch of the first instruction to the writeback of the last instruction. Assume that the machine stops fetching instructions after the fifth instruction.

As we saw in class, use F for fetch, D for decode, En to signify the nth cycle of execution for an instruction, and W to signify writeback. You may or may not need all columns shown. Finally, identify the cycle after which the snapshot of the microarchitecture was taken. Shade the corresponding cycle in the last row of the table.

Cycle:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Instruction 1	F	D	E1	E2	E3	E4	W												
Instruction 2		F	D				E1	E2	E3	E4	W								
Instruction 3			F	D							E1	E2	E3	W					
Instruction 4				F	D						E1	E2	E3	E4	W				
Instruction 5					F	D									E1	E2	E3	W	
Snapshot cycle										X									

3. Branch Prediction [50 points]

Assume the following piece of code that iterates through two large arrays, j and k, each populated with completely (i.e., truly) random positive integers. The code has two branches (labeled `B1` and `B2`). When we say that a branch is *taken*, we mean that the code *inside* the curly brackets is executed. Assume the code is run to completion without any errors (there are no exceptions). For the following questions, assume that this is the only block of code that will ever be run, and the loop-condition branch (`B1`) is resolved first in the iteration before the if-condition branch (`B2`).

```
for (int i = 0; i < 1000000; i++) {     /* B1 */
                                        /* TAKEN PATH for B1 */
    if (i % 3 == 0) {                   /* B2 */
        j[i] = k[i] - i;                /* TAKEN PATH for B2 */
    }
}
```

You are running the above code on a machine with a two-bit global history register (GHR) shared by all branches, which starts with *Not Taken, Not Taken* (2'b00). Each pattern history table entry (PHTE) contains a 2-bit saturating counter.
The saturating counter values are as follows:
2'b00 - Strongly Not Taken
2'b01 - Weakly Not Taken
2'b10 - Weakly Taken
2'b11 - Strongly Taken

(a) You observe that the branch predictor mispredicts 100% of the time in the first 5 iterations of the loop. Is this possible? Fill in the table below with all possible initial values each entry can take. Leave the table blank if this is not possible.

PHT

PHT Entry	Value
TT	01
TN	00
NT	01
NN	00 or 01

Show your work here:

The pattern after 5 iterations: TTTNTNTTTN.

- For GHR=NN, the only observed branch is T, which is the first taken branch. Therefore, the PHTE for NN has to be either 00 or 01 so that the branch predictor mispredicts the taken branch.
- For GHR=TT, the observed branches are T N T N. The PHTE for TT has to be initialized to 01 in order to cause the predictor to always mispredict.
- For GHR=TN, the observed branches are T T. Thus, the initial PHTE value for TN has to be 00 to mispredict both taken branches.
- For GHR=NT, the observed branches are T (i.e., the second taken branch) N T. Similar to the TT entry, NT's PHTE has to be initialized 01.

(b) *Please read the entire question first before answering any part.*

Rachata believes that the misprediction rate can become 0% during the steady state.

Is this possible?

Circle one: YES [NO]

If it is possible (YES), fill in one possible set of initial PHTE values that can lead to a 0% misprediction rate.

PHT

PHT Entry	Value
TT	
TN	
NT	
NN	

If it is not possible (NO), what is the lowest misprediction rate that can be achieved during the steady state?

The lowest misprediction rate is 33.33%.

Show all your work here below:

No, it's not possible. The best correct prediction rate is 4/6, which is 1/3 misprediction rate.
At steady state, we will keep observing the following pattern which repeats over time: TTTNTN.
With GHR=TN, this entry will saturate to 11, taken all the time. Therefore, 2 Ts will be always predicted correctly out of the 6 branches in the pattern.
With GHR=NT or TT, the predictor will observe either T and N. No matter what the initial values are for these two entries, only one of the branches can be predicted correctly. Therefore 2 out of of remaining 4 branches in the pattern will be predicted correctly.
With GHR=NN, the predictor won't observe it during the steady state.

5. Memory Consistency [40 points]

There are 2 threads with 4 instructions. The two threads are executed concurrently on a dual-core processor. Assume that registers in both cores are initialized with the values shown in the table below. The instructions of each thread are also shown below.

R1	1
R2	2
R3	3
R4	4

Thread A	Thread B
ST R1, 0x1000	ST R3, 0x1000
LD R5, 0x1000	LD R5, 0x1000
ADD R5, R5, R2	ADD R5, R5, R4
ST R5, 0x1000	ST R5, 0x1000

(a) Assume the dual-core processor implements sequential consistency. List **all** the possible values that can be stored in address 0x1000, assuming both threads run to completion.

3, 5, 7, 9

(b) How many different memory instruction interleavings of the 2 threads **will guarantee** a value of 0x9 in address 0x1000, assuming both threads run to completion? Show your work.

4

(c) Assume now that the dual-core processor does **not** support sequential consistency. List **all the possible values** that can be stored in address 0x1000, assuming both threads run to completion. Explain your answer briefly.

3, 5, 7, 9
Thread B might never see the sequential updates of Thread A, but the result will be equivalent to that of a sequentially consistent model.

6. Memory Interference [50 points]

During the lectures, we introduced a variety of ways to tackle memory interference. In this problem, we will look at the Blacklisting Memory Scheduler (BLISS) to reduce unfairness. There are two key aspects of BLISS that you need to know.

- When the memory controller services η consecutive requests from a particular application, this application is blacklisted. We name this non-negative integer η the **Blacklisting Threshold**.
- The blacklist is cleared periodically every **10000** cycles starting at t=0.

To reduce unfairness, memory requests in BLISS are prioritized in the following order:

- Non-blacklisted applications' requests
- Row-buffer hit requests
- Older requests

The memory system for this problem consists of 2 channels with 2 banks each. Tables 1 and 2 show the memory request stream in the same bank for both applications at varying times. The memory requests are labeled with numbers that represent the row position of the data within the accessed bank. Assume the following for all questions:

- A row buffer hit takes **50 cycles**.
- A row buffer miss/conflict takes **200 cycles**.
- All the row buffers are closed at time t=0.

Application A (Channel 0, Bank 0)							
Application B (Channel 0, Bank 0)	Row 1	Row 1	Row 1	Row 1	Row 1	Row 1	Row 1

Table 1: Memory requests of the two applications at t=0

Application A (Channel 0, Bank 0)	Row 3	Row 7	Row 2	Row 0	Row 5		
Application B (Channel 0, Bank 0)	Row 1	Row 1	Row 1	Row 1	Row 1	Row 1	Row 1

Table 2: Memory requests of the two applications at t=10

(a) Compute the slowdown of each application using the FR-FCFS scheduling policy after both threads ran to completion. We define $slowdown = \frac{mem\_latency\_with\_others}{mem\_latency\_alone}$. Show your work.

Slowdown of A: $\frac{(200+6*50)+5*200}{5*200} = \frac{1500}{1000} = 1.50$

Slowdown of B: $\frac{200+6*50}{200+6*50} = \frac{500}{500} = 1.00$

(b) For what value(s) of η (the Blacklisting Threshold) will the slowdowns for both applications be equivalent to those obtained with FR-FCFS?

For $\eta \geq 7$ or $\eta = 0$.
We want both A and B to complete without blacklisting or to complete both blacklisted, thus $\eta \geq 7$ and $\eta = 0$ respectively.

(c) For what value(s) of η (the Blacklisting Threshold) will the slowdown for A be <1.4?

Impossible. Slowdown for A will always be $\geq$1.4.
If you trace the schedule carefully, you will observe that A will be the fastest when η=5, where slowdown of A=1.4. η=5 is the smallest η where A does not get blacklisted.

(d) For what value(s) of η (the Blacklisting Threshold) will B experience maximum slowdown it can experience with the Blacklisting Scheduler?

η=5 or η=6
We observe that as long as B gets blacklisted at least once, B will incur an additional miss and hence an extra of 200 cycles. Thus, we want 2 conditions to be satisfied: B to miss at least once AND an η such that B completes last. These conditions are satisfied only when η=5 or η=6. $MaximumSlowdownofB = \frac{1650}{500} = 3.3$

(e) What is a simple mechanism (that we discussed in lectures) that will make the slowdowns of both A and B 1.00?

Memory Channel Partitioning (MCP)

7. Memory Latency Tolerance [60 points]

Assume an in-order processor that employs runahead execution, with the following specifications:

- The processor enters Runahead mode when there is a cache miss.
- There is a 64KB cache. The cache block size is 64 Bytes.
- The cache is 2-way set associative and uses the LRU replacement policy.
- A cache hit is serviced instantaneously.
- A cache miss is serviced after X cycles.
- The cache replacement policy chooses to evict a cache block serviced by Runahead requests over non-runahead requests. The processor does not evict the request that triggers Runahead mode until after Runahead mode is over.
- The victim for cache eviction is picked at the same time a cache miss occurs.
- Whenever there is a cache miss, the processor always generates a new cache request and enters Runahead mode.
- There is no penalty for entering and leaving Runahead mode.
- ALU instructions and Branch instructions take one cycle each and never stall the pipeline.

Consider the following program. Each element of array A is one byte.

```
for(int i=0;i<100;i++) \\ 2 ALU instructions and 1 branch instruction
{
  int m = A[i*32*1024]+1; \\ 1 memory instruction followed by 1 ALU instruction
  26 ALU instructions
}
```

(a) After running this program, you find that there are 50 cache misses. What are all the possible values of X?

$30 < X < 61$.

(b) Is it possible that every cache access in the program misses in the cache? If so, what is the value of X that will make all cache accesses in the program miss in the cache? If not, why? Show your work.

A cache latency of less than 31 cycles will not generate any cache requests during the runahead mode. A cache latency of at least 61 cycles will generate two fetches to the cache. The second fetch will evict the first runahead request in the cache, causing every access into a cache miss.

(c) What is the minimum number of cache misses that this program can achieve? Show your work.

50 misses.

(d) Assume that each ALU instruction consumes 1uJ, a cache hit consumes 10uJ, and a cache miss consumes Y uJ. Does there exist a combination of X and Y such that the dynamic energy consumption of Runahead execution is better than a processor without Runahead execution? Show your work.

No. With Runahead execution, the processor will always have to fetch extra instructions. With runahead execution, the processor will cause at least 100 cache misses (some in Runahead mode) and 30*100 ALU instruction.

(e) Assume the energy parameters in part d. What is the dynamic energy consumption of the processor with Runahead execution in terms of X and Y when X generates the **minimum** number of cache misses? Show your work.

At most, this program will convert 50 cache misses to cache hits while leaving the other 50 be cache misses. This case will happen if and only if the cache latency (X) is between $31 < X < 60$
With runahead: $misses * Y + ALU * 1 + hits * 10$+3 instructions at the end of the loop+100∗ALU in Runahead mode
With runahead: $3 + 30 * 1 * 100 + 50 * 10 + 100 * Y + 100X = 3503 + 100Y + 100(X - 1)$

(f) Assume the energy parameters in part d. What is the dynamic energy consumption of the processor with Runahead execution in terms of X and Y when X generates the **maximum** number of cache misses? Show your work.

In the worst case, every cache accesses is a miss. This will happen when $X <= 30$ and $X > 60$ However, in this case, we have to break the calculation for the energy consumption into two cases, the first case is when $X < 31$ and the second case is when Runahead execution generates extra cache requests.
When $X < 31$, the energy consumption is $3003 + 100Y + 100X$.
When $X > 60$, there will be $\lfloor X/30 \rfloor$ additional memory access in Runahead mode. So, the energy consumption is $3003 + 100(X - \lfloor X/30 \rfloor) + 100Y * \lfloor X/30 \rfloor$.