4 [20 points] Cache

Assume in this question that you have a 4KiB 2-way set associative L1 data cache with 64-byte cache lines, and that you are using 32-bit addresses.

(1) How many bits are required for the tag, index and offset? [2 pts]

(2) Now consider the following program. Assume that each long element is 8 bytes, and that the matrix elements are stored in row-major order. Assume that the first element in the matrix is aligned to the start of a cache line.

```
int sum = 0;
int matrix[64][64];
for (int i = 0; i < 64; i++) {
    for (int j = 0; j < 64; j++) {
        sum += matrix[i][j];
    }
}
```

(a) [2 pts] What's the number of cache misses?

(b) [2 pts each, 8 pts total] What's the number of occurrences each of the following events?

i. Hits:
ii. Compulsory misses:
iii. Conflict misses:
iv. Capacity misses:

(3) Suppose we add software prefetching. Assume that each iteration of the inner loop takes 30 cycles (when there is a L1 miss). Of those 30 cycles, 25 cycles is the L1 miss penalty while 5 cycles is taken by the addition operation. Ignore any overheads from executing other instructions, including the software prefetch instruction. The prefetcher only prefetches when **prefetch_cond** becomes true (which is set by instructions that are not shown, and that you likewise do not need to model).

```
long sum = 0;
long matrix[64][64];
for (int i = 0; i < 64; i++) {
    for (int j = 0; j < 64; j++) {
        if (prefetch_cond) prefetch(&matrix[i][j]+OFFSET);
        sum += matrix[i][j];
    }
}
```

(a) [2 pts] What should **OFFSET** be set to in order to prefetch the correct value? The address calculation works on the granularity of bytes.

(b) [2 pts] What should **prefetch_cond** be to minimize total number of cycles of execution?

(c) [2 pts] How many prefetch requests would be issued?

(d) [2 pts] With prefetching, how many cache misses are there?

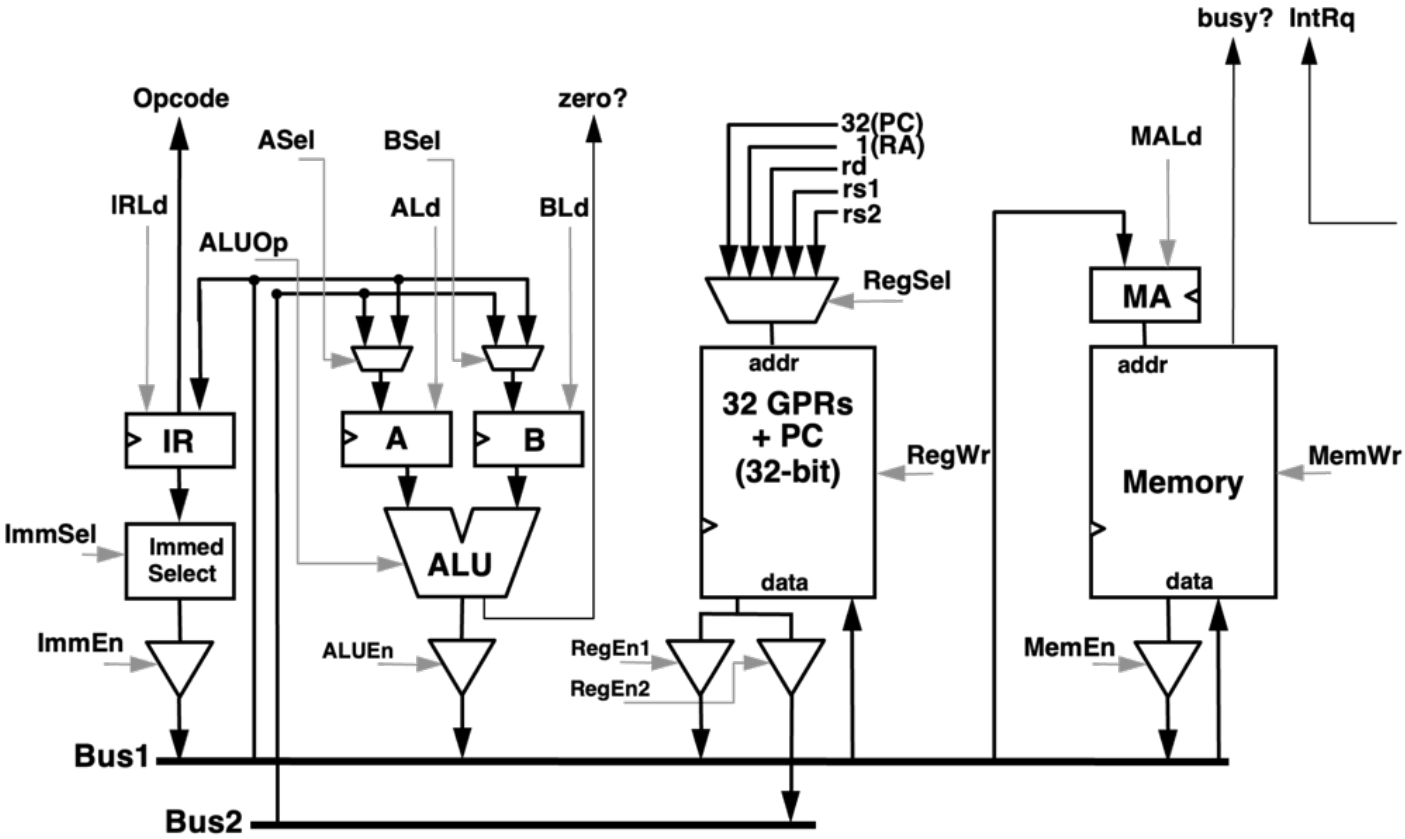
1 [20 points] Microcoding

One of the main issues with the single-bus datapath we've seen in class is that the bus frequently causes a structural hazard. To alleviate this issue, we will add a second bus to this design, called Bus2. However, to cut down on costs, we will only allow the RegFile write to this bus. Additionally, only the A and B registers will be able to read from this bus. Below is a description of the control signals we have to add:

RegEn1 and **RegEn2**: These signals replace RegEn from the original design. RegEn1 enables writing to Bus1, and RegEn2 enables writing to Bus2. These signals may be enabled simultaneously, allowing the RegFile to write the same data to both buses in the same cycle.

ASel and **BSel**: These signals determine which bus the A and B registers will load from during that cycle. They may be set to 1 or 2 to read from Bus1 or Bus2, respectively.

Below is a diagram of this double-bus approach, including the new control signals:



Your task in this question is to write microcode for this new design to implement the SetEqualMem instruction:

```
SetEqualMem rd, rs1, imm(rs2)
```

This instruction checks if the value in register rs1 is equal to the value stored in memory address rs2+imm. If they are equal, rd is set to 1. Otherwise, rd is set to 0. You may assume that reading from rs2+imm will not raise an exception. This instruction uses a new immediate type called X. Fill out the attached microcode table to implement this function. Your microcode should only use the optimal, minimal number of lines possible to implement SetEqualMem on the double-bus CPU. Additionally, your microcode is not allowed to modify either of the source registers (rs1 or rs2).

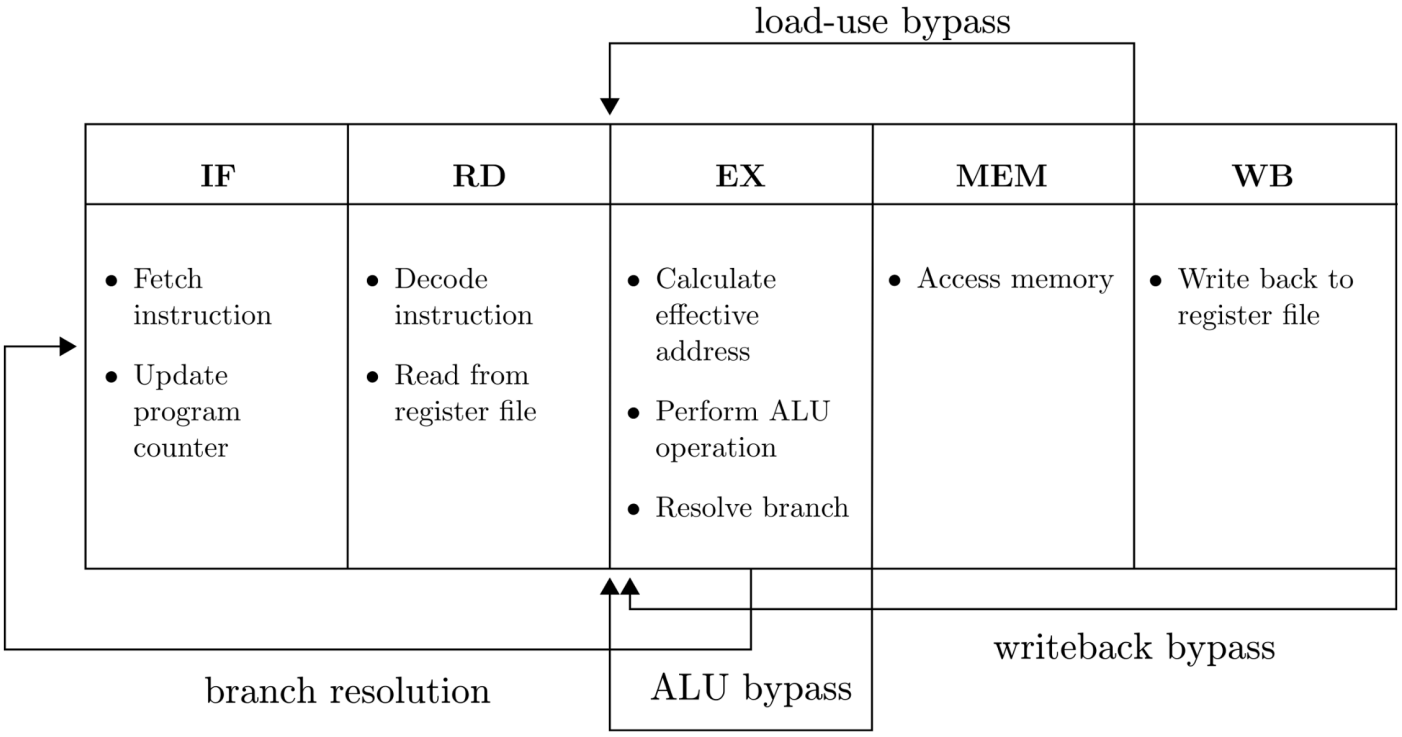
Some columns of the microcode table are gray. These columns will not be graded and you do not need to fill them out.

Focus first on making your implementation correct. Then, focus on making your correct implementation optimal. The majority of points will be awarded for your pseudocode.

State	Pseudocode	IR Ld	Reg Sel	Reg Wr	Reg En1	Reg En2	A Ld	B Ld	A Sel	B Sel	ALUOp	ALU En	MA Ld	Mem Wr	Mem En	Imm Sel	Imm En	μBr	Next State
FETCH0:	MA ← PC; A ← PC	*	PC	0	1	0	1	*	1	*	*	0	1	0	0	*	0	N	*
	IR ← Mem	1	*	0	0	0	0	*	*	*	*	0	0	0	1	*	0	S	*
	PC ← A+4	0	PC	1	0	0	0	*	*	*	INC_A_4	1	*	0	0	*	0	D	*
. . .																			
NOP0:	microbranch back to FETCH0	*	*	0	0	0	*	*	*	*	*	0	*	0	0	*	0	J	FETCH0
SetEqual Mem:																			

Problem 2: [14 points] Pipelines, Hazards, and CPI

Take the following 5-stage pipeline:



Assume memory accesses take one cycle and the branch predictor predicts not taken.

Take the following RISCV snippet, you may assume that accesses to memory addresses in x4, x5 will not cause an exception for the duration of the loop:

```
      1: mv x1, x0
      2: mv x2, x0
loop: 3: lw x6, 0(x4)
      4: lw x7, 0(x5)
      5: mul x6, x6, x7
      6: add x1, x1, x6
      7: addi x4, x4, 4
      8: addi x5, x5, 4
      9: addi x2, x2, 1
     10: bne x2, x8, loop
end:
```

(a) [1 pt] What does the code snippet compute? Your answer should be only a couple of words.

(b) [2 pts] Assume this code snippet is run on the given 5 stage in-order pipeline above. Identify all RAW data hazards that increase CPI. You may identify hazards either by writing their instruction number (e.g. 11 → 13) or the instructions themselves (e.g. addi x1 x2 x0 → lw x3 0(x1)).

(c) [3 pts] Again assuming this code is running on the given 5 stage in-order pipeline above. As the number of loop iterations approaches infinity, what is the average CPI? For your convenience, we have provided a pipeline diagram at the end of the question. **The pipeline diagram will not be graded.**

(d) [3 pts] To the right of the code sequence provided, provide a re-ordering of the instructions given (using instruction numbers or the instructions themselves) that would improve the CPI of this program. As the number of loop iterations approaches infinity, what is the new average CPI? Again, for your convenience, we have provided pipeline diagram scratch space, but **the pipeline diagram will not be graded.**

New Average CPI

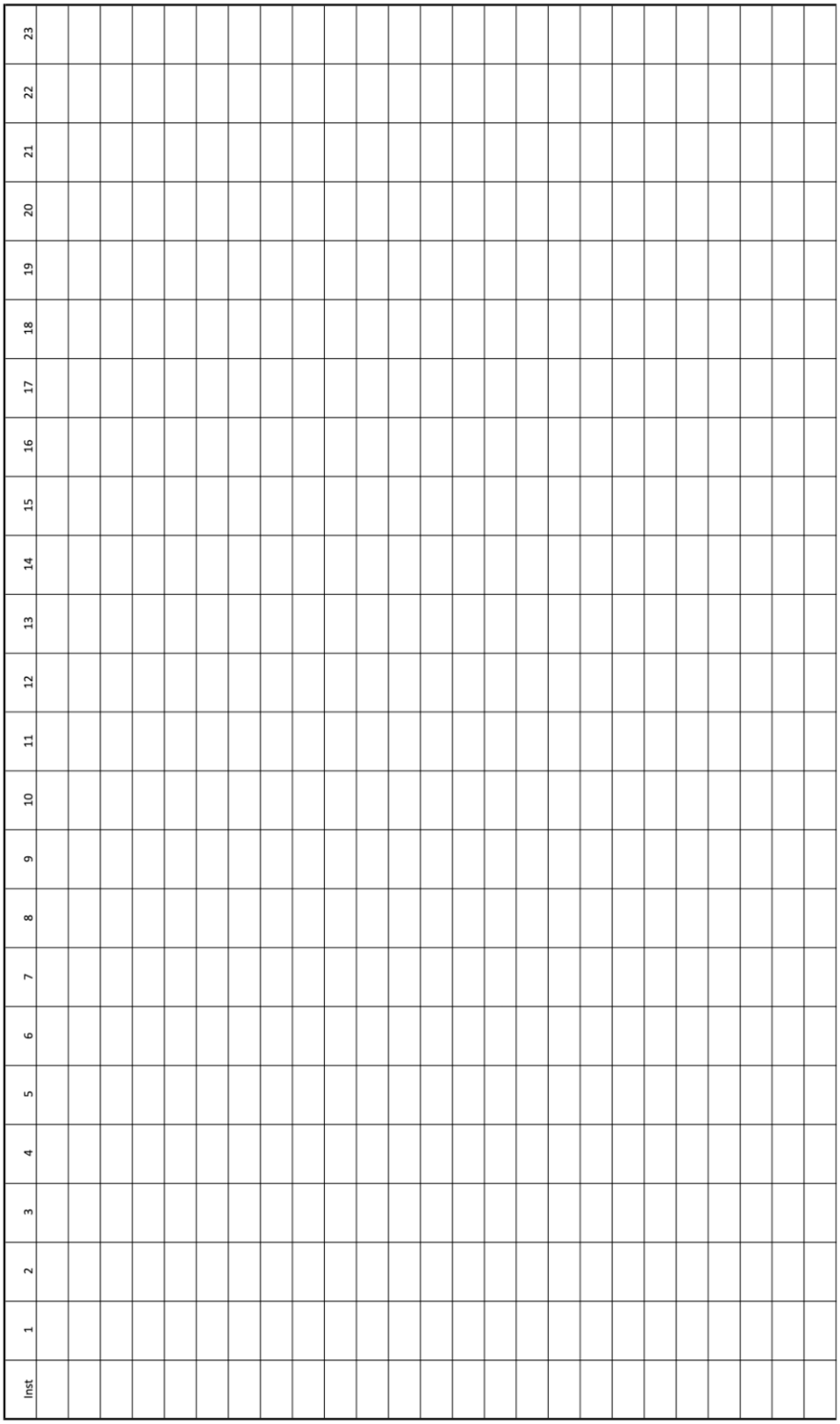
```
      1: mv x1, x0
      2: mv x2, x0
loop: 3: lw x6, 0(x4)
      4: lw x7, 0(x5)
      5: mul x6, x6, x7
      6: add x1, x1, x6
      7: addi x4, x4, 4
      8: addi x5, x5, 4
      9: addi x2, x2, 1
     10: bne x2, x8, loop
end:
```

(e) [1 pt] Say we add branch delay slots to the ISA. How many branch delay slots are required to hide all hazards for the above 5-stage pipeline?

(f) [2 pts] Label the instruction(s) that you would move into the branch delay slot(s) to improve CPI with their slot number. For example, the instruction in the first slot should be labeled with a (1). If you would like to place a `nop` in a delay slot, simply write nop at the end of the program and label it.

```
      1: mv x1, x0
      2: mv x2, x0
loop: 3: lw x6, 0(x4)
      4: lw x7, 0(x5)
      5: mul x6, x6, x7
      6: add x1, x1, x6
      7: addi x4, x4, 4
      8: addi x5, x5, 4
      9: addi x2, x2, 1
     10: bne x2, x8, loop
end:
```

(g) [2 pts] As the number of loop iterations for part (f) approaches infinity, what is the average CPI? Again, for your convenience, we have provided pipeline diagram scratch space, but **the pipeline diagram will not be graded.**


23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
Inst