Final Exam Solution

```
 _____ _____ ____ ____    _____ _____ ___  
| ____| ____/ ___/ ___|  |___ /|___  / _ \ 
|  _| |  _|| |   \___ \    |_ \   / / | | |
| |___| |__| |___ ___) |  ___) | / /| |_| |
|_____|_____\____|____/  |____/ /_/  \___/ 
```

EECS 370 Spring 2023: Introduction to Computer Organization

You are to abide by the University of Michigan College of Engineering Honor Code. Please sign below to signify that you have kept the honor code pledge:

I have neither given nor received aid on this exam,
nor have I concealed any violations of the Honor Code.

Signature: ______________________________

Name: ______________________________

Uniqname: ______________________________

Uniqname of person sitting to your ***Right***
(Write ⊥ if you are at the end of the row) ______________________________

Uniqname of person sitting to your ***Left***
(Write ⊥ if you are at the end of the row) ______________________________

Exam Directions:

- You have **120 minutes** to complete the exam. There are **8** questions in the exam on **13** pages (double-sided). **Please flip through your exam to ensure you have all pages.**
- You must show your work to be eligible for partial credit!
- Write legibly and dark enough for the scanners to read your answers.
- **Write your uniqname on the line provided at the top of each page.**

Exam Materials:

- You are allotted **one 8.5 x 11 double-sided** note sheet to bring into the exam room.
- You are allowed to use calculators that do not have an internet connection. All other electronic devices, such as cell phones or anything or calculators with an internet connection, are strictly forbidden.

Problem	1	2	3	4	5	6	7	8
Point Value	15	12	14	12	9	12	9	17

Problem 1: Multiple Choice 15 points

Completely shade in the boxes with the correct answers. Select only 1 answer, unless specified by "**(FILL IN ALL THAT APPLY).**"

1. Which of the following are valid **disadvantages** of a fully-associative cache over a direct-mapped cache? **(FILL IN ALL THAT APPLY)** *[1.5 points]*
   - ☑ Requires more storage needed for overhead bits
   - ☐ Slower to access since tags must be compared serially (one after the other)
   - ☑ Consume more power
   - ☐ Result in more conflict misses

2. Which of the following is generally true of pipelined architectures? **(FILL IN ALL THAT APPLY)** *[1.5 points]*
   - ☑ They enable multiple instructions to be executed simultaneously
   - ☐ They must contain at least 5 stages
   - ☐ They increase the clock frequency compared to a multi-cycle processor
   - ☑ They require more storage compared to a multi-cycle processor

3. For each performance optimization listed below (assume everything else is kept the same), indicate whether the performance is expected to improve by: decreasing CPI, decreasing the number of instructions fully executed, or decreasing the clock period by filling in the appropriate circle. **(FILL IN ALL THAT APPLY.)** (There may be multiple reasonable answers.) *[0.75 points per row]*:

Optimization	**Decrease CPI**	**Decrease # instructions executed**	**Decrease clock period**
Introducing cache into a previous cache-less system	●	○	●
Switching from a static "predict-not-taken" branch predictor to a more accurate, dynamic predictor	●	○	○
Converting a single-cycle machine into a 5-stage pipeline	○	○	●
Switching from avoidance to detect-and-forward to handle data hazards	○	●	○

For row 1: depends. Selecting both or either of Decrease CPI or clock period is acceptable.

4. Consider 4 segments of code below:

```
int sum_array(int *a, int M) {
  int sum=0;
  for(int i=0; i<M; i++)
    sum += a[i];
  return sum;
}
```
	```
	void add_matrix_row(int **a,
	        int **b, int M, int N) {
	  for(int i=0; i<M; i++)
	    for(int j=0; j<N; j++)
	      a[i][j] += b[i][j];
	}
	```

```
int sample_array(int *a, int M,
                 int s) {
  int sum=0;
  for(int i=0; i<s; i++) {
    int k = rand() % M;
    sum += a[k];
  }
  return sum / s;
}
```
	```
	void add_matrix_col(int **a,
	        int **b, int M, int N) {
	  for(int j=0; j<N; j++)
	    for(int i=0; i<M; i++)
	      a[i][j] += b[i][j];
	}
	```

Assume all variables except array elements are stored in registers. Which of the following do we expect to be true considering a 1 KB fully-associative cache with 64 byte blocks (assuming array sizes `M` and `N` and sample size `s` are much larger than the number of bytes in the cache)? **(FILL IN ALL THAT APPLY)** *[3 points]*

- [✓] sum_array will have a higher hit rate than add_matrix_col
- [ ] add_matrix_col will have a higher hit rate than add_matrix_row
- [✓] sum_array will have a higher hit rate than sample_array
- [✓] sample_array will have a higher hit rate than add_matrix_col

5. Select which type of page table best describes each situation. *[0.5 points per row]*

Statement	Single-level	Multi-level
_____ page tables are expected to provide translations with lower latency	[✓ ○]	○
_____ page tables are expected to require less memory in the worst case	[✓ ○]	○
_____ page tables are expected to require less memory in the typical case	○	[✓ ○]

6. What is the primary reason for including a translation lookaside buffer (TLB) in modern systems, as opposed to directly accessing a page table? *[1.5 points]*
   - ☐ To allow multiple processes to share the L1 cache
   - [✓ To decrease the average latency of instructions]
   - ☐ To ensure a process does not access memory outside of its address space
   - ☐ To enable more accurate translation from virtual addresses to physical addresses

7. Which of the following are defined as part of an ISA? **(FILL IN ALL THAT APPLY)** *[1.5 points]*
   - [✓ Number of registers]
   - ☐ Size of the cache
   - [✓ Virtual address size]
   - ☐ Physical address size
   - ☐ Whether a register is caller/callee saved
   - ☐ Whether page tables are single-level vs multi-level

8. Which of the following are advantages of the detect-and-stall scheme over avoidance regarding data hazards? **(FILL IN ALL THAT APPLY)** *[1.5 points]*
   - ☐ Detect-and-stall reduces the number of stalls due to data hazards when executing a program
   - [✓ Detect-and-stall preserves backwards compatibility of programs on new processors]
   - ☐ Detect-and-stall reduces hardware complexity
   - [✓ Detect-and-stall programs generally have fewer misses in the instruction cache]
   - ☐ None of the above

Problem 2: Shortest Sequences of them All 12 points

Consider three cache designs:

A. A 256 byte direct-mapped cache with 16 byte blocks (16 lines, 4 index bits)
B. A 256 byte direct-mapped cache with 32 byte blocks (8 lines, 3 index bits)
C. A 256 byte 2-way associative cache with 16 byte blocks (8 sets, 3 set bits)

1. Provide the shortest sequence of memory addresses which would result in a miss in cache A, but a hit in cache B (provide your answer in hex). *[4 points]*

Answer with lowest addresses: 0x0 0x10
As long as bit 4 differs between the two, and bits 5+ are the same, it works.

2. Provide the shortest sequence of memory addresses which would result in a miss in cache A, but a hit in cache C (provide your answer in hex). *[4 points]*

Answer with lowest addresses: 0x0 0x100 0x0
All addresses must map to the same set for both DM & 2-way (bits 4-7 are the same). The first and third addresses must be in the same block (only block offset = bits 0-3 differ) and the second address (which evicts the first in the DM cache) must have a different tag (bits 8+ differ).

3. Provide the shortest sequence of memory addresses which would result in a miss in cache C, but a hit in cache A (provide your answer in hex). *[4 points]*

Answer with lowest addresses: 0x0 0x80 0x180 0x0
All addresses must map to the same set for 2-way (bits 4-6 are the same). The first and last addresses must be in the same block (only block offset = bits 0-3 differ) and the second address & third addresses (which evict the first in the 2-way cache) must have a different tag (bits 8+ differ) from all other addresses. The second and third addresses must also share bit 7, but differ in bit 7 from the first/last address, so that they all map to the same set in 2-way but the 2nd/3rd are in a different line for DM from first/last.

Problem 3: The $ is Write 14 points

```
double data[16];    // Doubles are 8 Bytes
for(unsigned int i = 0; i < 8; i++) {
     data[i * 2] = data[i + 8];
}
```

Consider a 64 B fully-associative data cache with a block size of 16 B and a **write-back, no-allocate-on-write** policy. Evictions are based on true LRU policy. There are 256 B of memory. Assume that `data` begins at address 0, and that only `data` is cached. Answer the questions below and show your work for all parts.

1. How many overhead bits are there for one cache line? *[2 points]*

   64/16 = 4 blocks

   1 valid bit, 1 dirty bit, 2 LRU (since lg(4) = 2), 4 tag bits (lg(256) = 8, lg(16) = 4, 8 - 4 = 4)

   #Bits: ___8___

2. Fill in the table to indicate the number of occurrences of each type of access. *[4 points]*

	Reads	Writes
Hits	4	4
Misses	4	4

Pattern: ld 8, st 0, ld 9, st 2, ld 10, st 4, ld 11, st 6, ld 12, st 8, ld 13, st 10, ld 14, st 12, ld 15, st 14.

3. How many bytes are transferred from memory to the cache? *[2 points]*

Only read misses will transfer from mem to cache. 8 doubles are read, there is spatial locality, so 8 * 8 B or 16 * 4 = 64B.

#Bytes: __64____

4. How many bytes are written to memory? Assume all dirty blocks are written back into memory at program halt. *[3 points]*

8B (1 double) per store miss, 16B per dirty block. 4 store misses, 4 store hits to different blocks. 8*4 + 16*4 = 96B

#Bytes: __96____

5. Given a 1 ns access time for cache and a 100 ns access time for memory, what is the average access time latency over the execution of this code block assuming that the cache is always accessed before memory? Include the writeback of dirty blocks at program halt. *[3 points]*

16 total accesses. 4 read misses, 4 read hits. 4 write misses, 4 write hits. At halt, 4 writebacks.
[16 * 1ns (cache) + (4 + 4 + 4) * 100ns (mem)] / 16 = 1 + 1200/16 = 76

Avg Latency (ns): __76____

Problem 4: C’ing Clearly 12 points

Consider a byte-addressable architecture with the following data cache:

Cache size: 64 bytes

Block size: 16 bytes

Associativity: 2 way set-associative

Assume that the cache is initially empty, and uses true LRU replacement policy. Way-0 is chosen when the set is empty.

1. Say a program makes a series of memory accesses (in column “Reference” below). Identify whether each access is a: *Hit, Compulsory Miss, Capacity Miss, or Conflict Miss*. The other columns in the table are for your convenience. Block offsets are **bolded** for simplicity. *[9 points]*

Row	Reference	Binary reference	Set	Way	Hit or Miss Type
A	0x92**E**	0b1001 0010 **1110**	0	0	Compulsory Miss
B	0xA2**C**	0b1010 0010 **1100**	0	1	Compulsory Miss
C	0xA3**7**	0b1010 0011 **0111**	1	0	Compulsory Miss
D	0x34**6**	0b0011 0100 **0110**	0	0	Compulsory Miss
E	0x92**6**	0b1001 0010 **0110**	0	1	Conflict Miss
F	0x01**0**	0b0000 0001 **0000**	1	1	Compulsory Miss
G	0xA2**5**	0b1010 0010 **0101**	0	0	Capacity Miss
H	0x92**8**	0b1001 0010 **1000**	0	1	Hit

2. Say then that we made a new cache with the same cache size and associativity (2-way), but a block size of 32 bytes. Which one of the memory accesses above would become a hit?

HINT: With the size change, how many sets are there?

List only the row letter *[3 points]*: ___C____

Problem 5: 0xBEEF on the Tables 9 points

Consider a system that implements virtual memory with the following specifications:

- 16 bit virtual address
- 4KB of RAM
- 16 B pages
- Initially empty 3 level page table
- Each page table takes up 2 pages and is aligned to a page boundary
- 2 B page table entries
- Initially empty TLB with 2 entries and an LRU eviction scheme

To get some insight into this system's performance, a benchmark is executed. The following virtual memory addresses are accessed during this run, in the following order:

```
0xDEAD
0xBEEF
0xDEA1
0x1234
0xBEAD
0xBDA0
```

Answer the following questions about memory use during this run. Show your work for each part.

1. How many page tables are allocated across ALL levels if no page tables are in memory initially? *[3 points]*

   1 first level + 3 second level (0xD, 0xB, 0x1) + 4 third level (0xDE, 0xBE, 0x12, 0xBD)

   #Tables: ___8____

2. How many TLB accesses result in a hit? *[3 points]*

   Only 1 VPN is repeated: 0xDEA. And the TLB can hold 0xDEA and 0xBEE at once.

   #Hits: ___1___

3. How many page faults are generated from these accesses? *[3 points]*

   5: 0xDEA, 0xBEE, 0x123, 0xBEA, 0xBDA

   #Faults: ___5___

Problem 6: Addressing Performance 12 points

You are working with a system that implements virtual memory with the following specifications:

- Single level page table
- 16 bit virtual address
- 16B page size
- 64B physical memory

For performance, you are considering adding a cache with the following characteristics:

- Fully associative
- 4 lines
- 8B block size

The components available to you have the following delays. Assume components are accessed serially, and data is sent to the processor immediately upon retrieval with no added delay to update state of components.

Disk:	1000	ns
Memory:	100	ns
Cache:	10	ns
TLB:	1	ns

To decide whether you should make this cache virtually or physically addressable, you come up with a set of addresses that a single running process may access.

Suppose this is the initial state of the system, and **does not change** during process execution:

TLB:

VPN	PPN
0xDEA	0x1
0x123	0x3

Cache:

Valid	Start VA / PA of Block (shown instead of tag)
1	0x5668 / 0x08
0	0x1230 / 0x30
1	0xDEA8 / 0x18
0	0xDCB0 / 0x20

Memory:

PPN	VPN
0x0	Reserved for OS
0x1	0xDEA
0x2	0xBEE
0x3	0x123

Consider these virtual addresses accessed by the process:

0xDEAD 0xBEEF 0x1234 0xABCD

1. How long will the above accesses take with no cache installed? *[3 points, show work]*

   0xDEAD and 0x1234: TLB hit (1ns) + mem (100ns) = 101ns
   0xBEEF: TLB miss (1ns) + page table walk (100ns) + mem (100ns) = 201ns
   0xABCD: TLB miss (1ns) + page table walk (100ns) + fault (1000ns) = 1101ns

Total time (ns): ___1504___

2. How long will the above accesses take with a virtually addressed cache? *[4 points, show work]*

0xDEAD: Cache hit (10ns)
0xBEEF: Cache miss (10ns) + TLB miss (1ns) + page table walk (100ns) + mem (100ns) = 211ns
0x1234: Cache **miss** (10ns) + TLB hit (1ns) + mem (100ns) = 111ns
0xABCD: Cache miss (10ns) + TLB miss (1ns) + page table walk (100ns) + fault (1000ns) = 1111

Total time (ns): ___1443___

3. How long will the above accesses take with a physically addressed cache? *[5 points, show work]*

0xDEAD: TLB hit (1ns) + Cache hit (10ns) = 11ns
0xBEEF: TLB miss (1ns) + page table walk (100ns) + cache miss (10ns) + mem (100ns) = 211ns
0x1234: TLB hit (1ns) + cache miss (10ns) + mem (100ns) = 111ns
0xABCD: TLB miss (1ns) + page table walk (100ns) + fault (1000ns) = 1101ns

Total time (ns): __1434____

Problem 7: Predicting the Predictors 9 Points

A C program has three branches. Their outcomes during execution are listed below (note B2 and B3 have fewer executions than B1):

B1	NT	NT	NT	NT	NT	T
B2	NT	T	NT	T	NT	
B3	T	T	T	T	T	

1. Assume static prediction, where a compiler decides whether a branch is predicted **always** T or NT. Determine the optimal prediction (assuming perfect knowledge of the branch outcomes) and corresponding misprediction rate (reported as a fraction) for the above execution. *[1 point per column]*

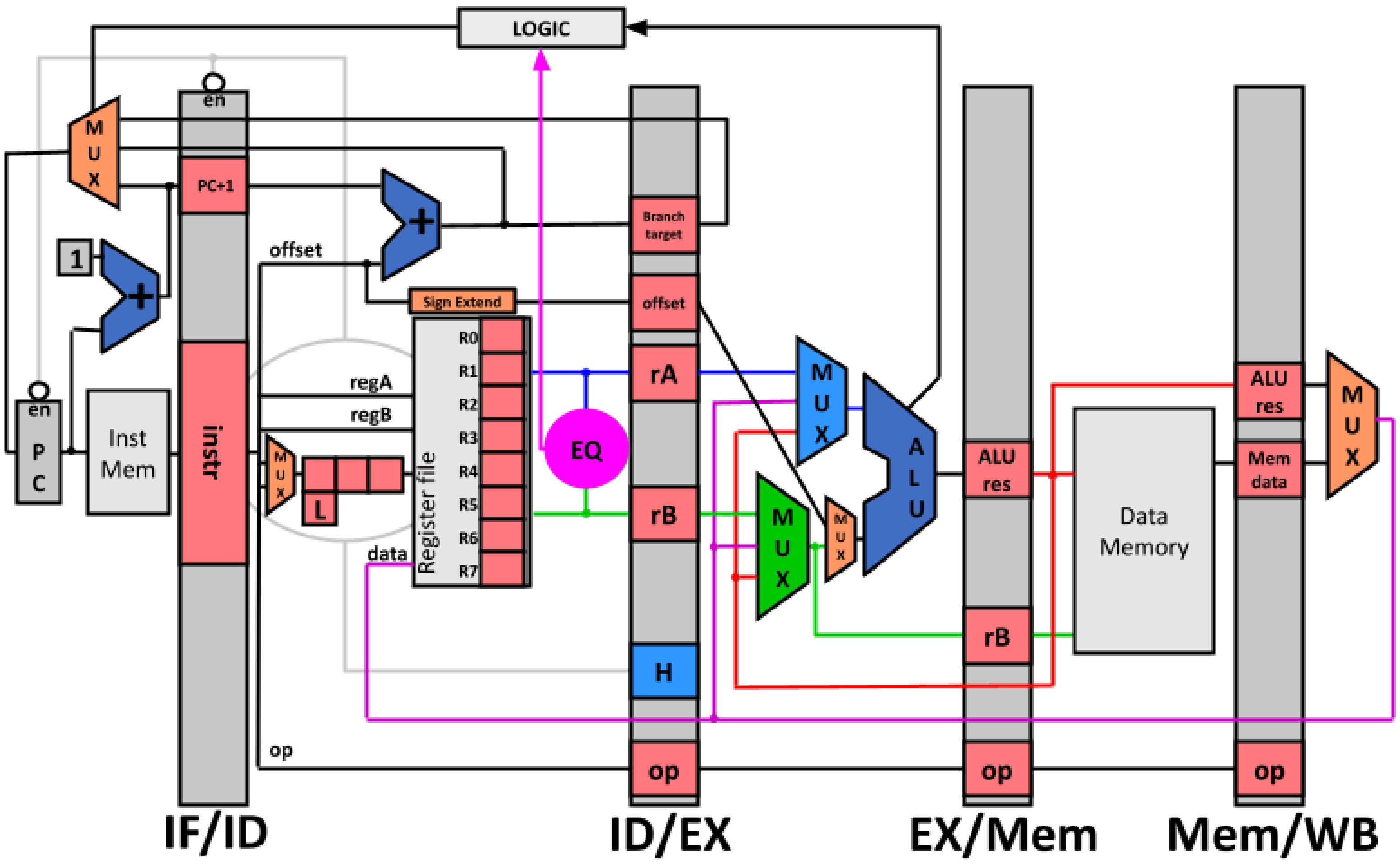
	B1	**B2**	**B3**
T or NT?	**NT**	**NT**	**T**
Misprediction rate	**1/6**	**2/5**	**0**

2. Assume each branch is dynamically predicted using a local **2-bit saturating counter** for each branch. Assume counters are initialized to **weakly not taken**. Determine mispredictions. *[2 points per box]*

	B1	**B2**	**B3**
Misprediction rate	**1/6**	**2/5**	**1/5**

Problem 8: Trimming the Branches 17 points

Consider the 5-stage pipeline discussed in lecture which supports internal forwarding in the register file, detect-and-forward for data hazards, and speculate-and-squash (predicting not taken) for control hazards.
A recent breakthrough in the logic required for 32-bit equality comparison has allowed an opportunity to reduce the penalty for mispredicted branches. A new version of the pipeline has been proposed that adds this "Fast Equality Test" to the decode stage, and also moves the branch target calculation to the decode stage.



This allows some branches to be resolved in the decode stage. The problem is that any beq with a data hazard (i.e. that compares a forwarded value) cannot be resolved in the decode stage. In such cases, the comparison is done in the execute stage (as in the base pipeline), but now the PC update is also done in the EX stage. The logic necessary for these changes is reflected in the above diagram. **See reference packet for larger, color version.**

1. For the following LC2K assembly programs, list how many total noops would be inserted by the hardware for stalls + squashes. For each program, assume the initial register state below, which results in all branches being taken: *[1.5 points each]*

Register	0	1	2	3
Initial Value	0	3	18	9

Benchmark 1:

```
      lw      0   1   data
      add     3   3   3
      beq     3   1   end
end   halt
data  .fill  18
```

# of noops: ___2____ (0 stall, 2 squash)

Benchmark 2:

```
      lw      0   1   data
      beq     2   1   end
end   halt
data  .fill  18
```

# of noops: ___3____ (1 stall, 2 squash)

Benchmark 3:

```
      lw      0   1   data
      lw      1   3   2
      beq     2   1   end
end   halt
data  .fill  18
```

# of noops: ___2____ (1 stall, 1 squash)

Benchmark 4:

```
      lw      0   1   data
      add     1   0   2
      beq     2   1   end
end   halt
data  .fill  18
```

# of noops: ___3____ (1 stall, 2 squash)

Benchmark 5:

```
      add     0   0   2
      add     1   1   1
      add     3   3   3
      beq     2   0   end
end   halt
```

# of noops: ___1____ (0 stall, 1 squash)

Benchmark 6:

```
      add     1   1   1
      add     0   0   2
      add     3   3   3
      beq     2   0   end
end   halt
```

# of noops: ___2____ (0 stall, 2 squash)

2. Consider a shortened version of the project 1 spec example test case below:

```
        lw      0       1       two     load reg1 with 2
        lw      0       2       neg1    load reg2 with -1
start   add     1       2       1       decrement reg1
        beq     0       1       done    goto end of program when r1==0
        beq     0       0       start   go back to loop beginning
        noop
done    halt                            end of program
two     .fill   2
neg1    .fill   -1
```

For certain cycles in the above code, the pipeline will have a branch in both the ID and EX stages. If the branches in both stages compute that the register values are equal and the branches should be taken, which stage should be prioritized? What would happen in the above code if the priority was reversed? *[3 points]*

EX stage should be prioritized, since it comes first in program order. The above code would infinitely loop otherwise because beq 0 1 done would never be taken and beq 0 0 start would always be taken backwards.

3. Consider the following benchmark of 800 instructions:
   - 35% of instructions are adds
   - 10% of instructions are nors
   - 5% of instructions are sws
   - 5% of instructions are noops or halt
   - 25% of instructions are beqs
     - 40% of beqs can be resolved in the ID stage on the new design
       - 50% of these are taken
     - The other 60% cannot
       - 20% of these are taken
   - 20% of instructions are lws
     - 40% of lws are immediately followed by a dependent instruction.

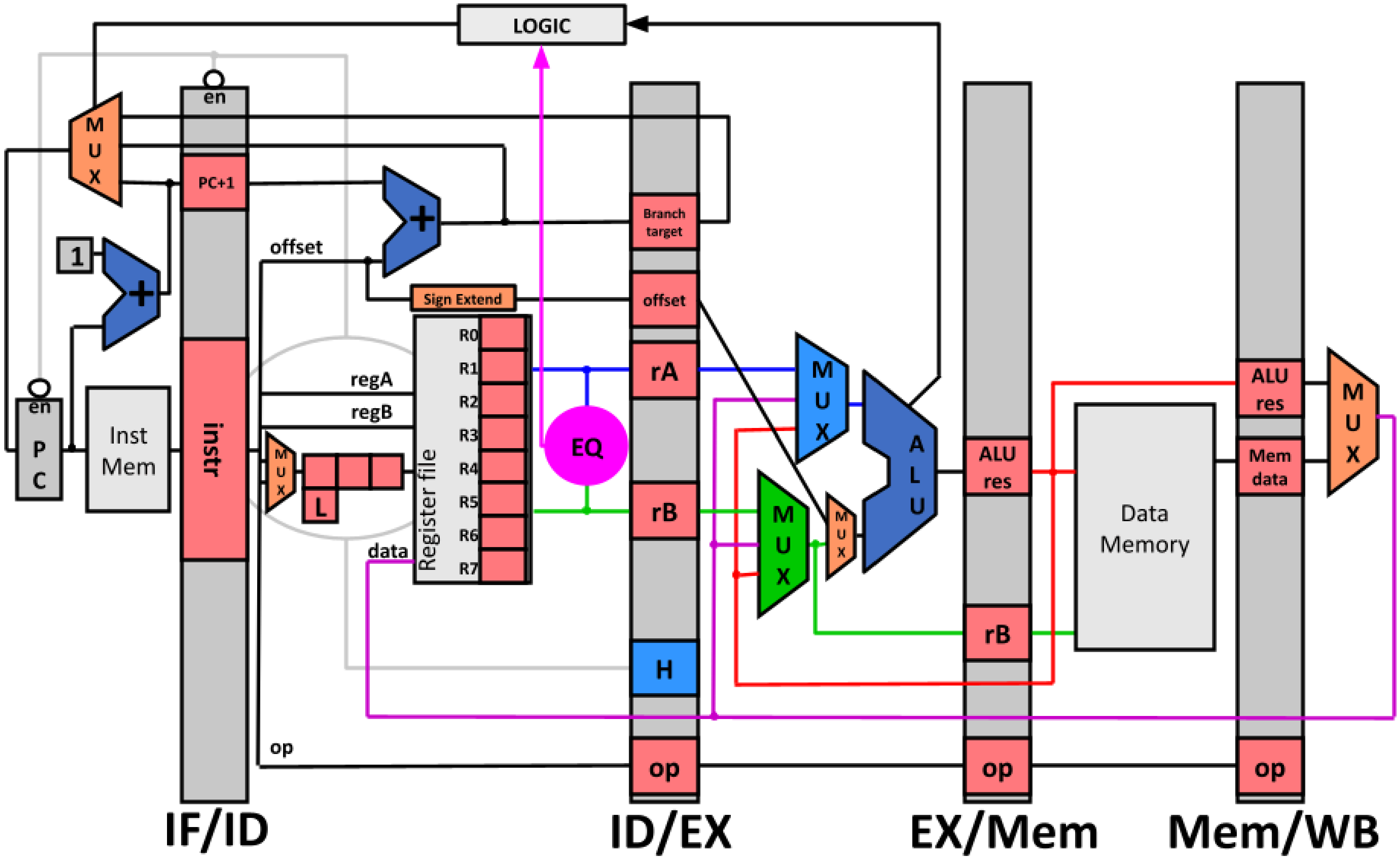
What is the CPI of the benchmark on the original and on the new pipelines? *[5 points]*

0.25 point for the 4 startup cycles: 4/800 = 0.005
0.75 point for stalls: 800*.2*.4 / 800 = 64 / 800 = 0.08
1 point for original beq squashes: 0.25 * (.4*.5 + .6*.2) * 3 = .32 * .75 = 0.24
2 pts for new beq squashes: 0.25 * (.4 * .5 * 1 + .6 * .2 * 2) = .25 * .44 = 0.11
0.5 pts for including base CPI
0.5 pts for not including extra terms

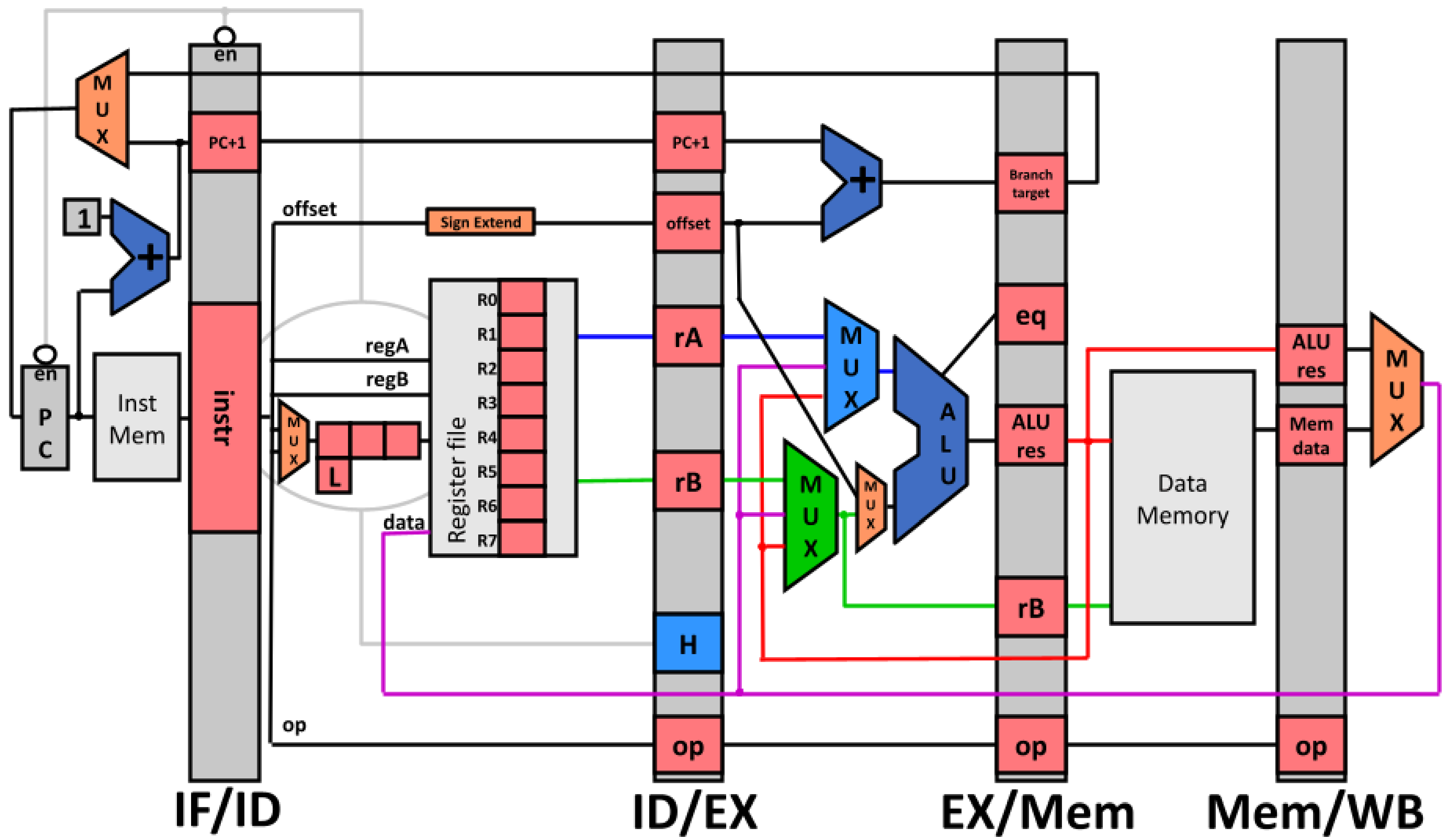
Original CPI: _________1.325__________ **New CPI:** _________1.195__________

This page intentionally left blank. You may use this for scratch work, but not graded work!

Modified Pipeline:



Lecture pipeline with Detect-and-forward (released ahead of exam):


en
MUX
PC+1
1
+
en
PC
Inst Mem
instr
offset
Sign Extend
regA
regB
MUX
L
data
Register file
R0
R1
R2
R3
R4
R5
R6
R7
op
PC+1
offset
rA
rB
H
op
+
MUX
MUX
MUX
ALU
Branch target
eq
ALU res
rB
op
Data Memory
ALU res
Mem data
op
MUX
IF/ID
ID/EX
EX/Mem
Mem/WB