Final Exam

EECS 370 Winter 2024: 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: *ANSWER KEY*

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 7 questions in the exam on 16 pages (double-sided). **Please flip through your exam to ensure you have all pages.**
- 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
Point Value	15	17	8	8	12	20	20

THIS PAGE INTENTIONALLY LEFT BLANK

Problem 1: Multiple Choice 15 points

Completely shade in the boxes with the best answer. **PROBLEMS 1-5 CONTAIN POTENTIALLY MULTIPLE CORRECT ANSWERS. SELECT ALL THAT APPLY** *[2 points each]*

1. Which of the following are advantages of detect-and-forward over detect-and-stall scheme regarding data hazards? **(FILL IN ALL THAT APPLY)**
   - ☑ ~~Detect-and-forward reduces the number of stalls due to data hazards when executing a program~~
   - ☐ Detect-and-forward preserves backwards compatibility of programs on new processors
   - ☐ Detect-and-forward reduces hardware complexity
   - ☐ Detect-and-forward architectures typically take up less space in the instruction cache due to fewer noops
   - ☐ None of the above

2. Which of the following is true regarding the 32-bit floating-point (FP) representation discussed in class (**FILL IN ALL THAT APPLY)**
   - ☑ ~~There are some 32-bit FP numbers that can't be represented as 32-bit 2's complement integers~~
   - ☑ ~~There are some 32-bit 2's complement integers that can't be represented as 32-bit FP numbers~~
   - ☑ ~~There are an equal number of positive FP values and negative FP values that can be represented exactly~~
   - ☐ There is a "gap" between any two adjacent FP numbers that can be represented exactly, and the size of this gap is constant across all numbers
   - ☐ None of the above

3. Which of the following are defined as part of an ISA? **(FILL IN ALL THAT APPLY)**
   - ☑ ~~How many bits the PC is~~
   - ☐ Whether a register is caller/callee saved
   - ☐ Number of pipeline registers
   - ☐ Number of levels in the page table
   - ☑ ~~Number of bits in a virtual address~~
   - ☐ Number of bits in a physical address

4. Write-back caches have what advantage over write-through caches on average? **(FILL IN ALL THAT APPLY)**
   - ☑ ~~Less total amount of memory transferred~~
   - ☐ Fewer conflict misses
   - ☐ Fewer compulsory misses
   - ☐ Smaller storage overhead

5. Moving from a three-level hierarchical page table to a two-level page table is expected to (**FILL IN ALL THAT APPLY)**
   - ☐ Increase the latency of translation
   - ☐ Decrease the amount of memory used for the page table in the common case
   - ☐ Lower the risk of two virtual addresses from different programs colliding to the same physical address
   - ☑ ~~None of the above~~

PROBLEMS 6-10 ONLY HAVE 1 CORRECT ANSWER *[1 point each]*

6. The CPI of the pipelined processor discussed in class is expected to be near what if no hazards or cache misses occur?
   - ☐ 0
   - ☐ 1/5
   - ☐ 1/2
   - ☑ ~~1~~
   - ☐ 2
   - ☐ 5

7. A BTB is effective because control flow instructions (i.e. branches and jumps):
   - ☐ usually (but not always) follow a predictable pattern of outcomes of taken/not-taken
   - ☐ always follow a predictable pattern of outcomes of taken/not-taken
   - ☑ ~~usually (but not always) have the same targets~~
   - ☐ always have the same targets
   - ☐ always resolve before write-back

8. Increasing the block size of the cache while keeping the total size constant takes better advantage of _____ locality
   - ☑ ~~Spatial~~
   - ☐ Homomorphic
   - ☐ Hysteresis
   - ☐ Temporal
   - ☐ None of the above

9. Which of the following cache designs (assuming equal total size and number of entries) is expected to consume the least amount of power?
   - ☐ Fully-associative
   - ☑ ~~Direct-mapped~~
   - ☐ Set-associative
   - ☐ They are all expected to use the same amount

10. Which caches are generally expected to have the lowest overall latency to retrieve their data on a hit?
    - ☑ ~~Virtually addressed~~
    - ☐ Physically addressed

Problem 2: Short Answers 17 points

Consider two cache designs:

A. A 128 byte direct-mapped cache with 32 byte blocks
B. A 128 byte 2-way associative cache with 32 byte blocks

1. Assuming the cache starts out empty, what is the smallest number of memory accesses that could result in a **hit in cache A**, but the exact same accesses would result in a **miss in cache B**. Write "N/A" if this situation is impossible *[2 points]*

(E.g. if the 3 accesses "100, 200, 100" resulted in a hit in A but a miss in B, and no shorter set of accesses did so, you would write "3"). You don't need to specify what addresses cause this behavior)

Number of accesses:

4

2. Assuming the cache starts out empty, what is the smallest number of memory accesses that could result in a **hit in cache B**, but the exact same accesses would result in a **miss in cache A**. Write "N/A" if this situation is impossible *[2 points]*

Number of accesses:

3

3. Consider each of the optimizations below. Assume that when each is individually applied (meaning everything else in the system stays the same), a program's performance improves. For each optimization indicate why the performance is expected to increase. **(FILL IN ALL THAT APPLY.)** (There may be multiple reasonable answers.) *[4 points]*:

Optimization	Decrease CPI	Decrease # instructions executed	Decrease clock period
Switching from a multi-cycle processor to a 5-stage pipeline	◯	◯	◯
Expanding the ISA from just using "nor" to "nor", "and", "or", "not", and "xor"	◯	◯	◯
Doubling the size of a cache	◯	◯	◯
Switching from avoidance to detect-and-forward to handle data hazards	◯	◯	◯

4. Consider moving **from** a 4-way set associative cache **to** a 2-way set associative cache (both using LRU replacement and having the same number of lines and total data storage). Select whether each metric below is, **in the average case**, expected to **increase (↑)**, **decrease (↓)**, or **stay constant (0)** *[5 points]*

Metric	↑ / ↓ / 0
Tag overhead	○↑ ○↓ ○0
LRU overhead	○↑ ○↓ ○0
Number of bits used to select a byte within one block	○↑ ○↓ ○0
Hit rate	○↑ ○↓ ○0
Access time	○↑ ○↓ ○0

Consider a byte-addressable system with a 256 B fully-associative cache that has a block size of 64 B. The following lines of code are executed:

```
uint16_t data[30];
 for(unsigned int i = 0; i < 4; i++) {
   for(unsigned int j = 0; j < 30; j++) {
      data[i] = data[i] + i;
   }
 }
```

How many bytes would be written into main memory for `data` array accesses, in each case of write-through and write-back policy for the cache. You can assume `data` starts from address 0 and all dirty blocks are written *in their entirety* back to cache at the end of the program. You can assume nothing besides `data` is placed in the cache. *[4 points]*

Write-through: __240_____ B Write-back: __64_____ B

(If you assumed uint16_t was 16 bytes instead of bits, you'd get 1920 & 64 - no penalty)

Problem 3: Cache Basics 8 points

Indicate the number of bits used for the index (either set or line, depending on the circumstance) and block offset of each of the following caches. Assume the address size (physical and virtual) is 32-bits and that memory is byte addressable.

1. A 128KB, 4-way associative cache with 32-byte blocks. Index: __10_ Offset: _5___
2. A 48KB, 3-way associative cache with 8-byte blocks. Index: __11_ Offset: _3___
3. A 1MB, direct-mapped cache with 32-byte blocks. Index: _15__ Offset: _5___
4. A 1MB fully-associative cache with 512-byte blocks. Index: __0__ Offset: _9___

Problem 4: Classifying Cache Misses 8 points

Consider a 2-way set-associative cache with 2 sets and 16 byte blocks. The cache is initially empty and uses LRU replacement.

For each access select whether the access is a "hit", or if a miss, indicate either "compulsory", "capacity", or "conflict".

Tag	Set Index	Block Offset	"Hit" or Miss Type
A	0	0	◯ Hit ◯ Compulsory ◯ Capacity ◯ Conflict
A	1	3	◯ Hit ◯ Compulsory ◯ Capacity ◯ Conflict
B	0	7	◯ Hit ◯ Compulsory ◯ Capacity ◯ Conflict
B	1	7	◯ Hit ◯ Compulsory ◯ Capacity ◯ Conflict
C	0	F	◯ Hit ◯ Compulsory ◯ Capacity ◯ Conflict
A	0	6	◯ Hit ◯ Compulsory ◯ Capacity ◯ Conflict
B	0	B	◯ Hit ◯ Compulsory ◯ Capacity ◯ Conflict
A	1	D	◯ Hit ◯ Compulsory ◯ Capacity ◯ Conflict

Problem 5: Trees and Branches 12 points

Consider the following assembly program which counts how many numbers in a series are odd. Assume all registers are initialized to zero.

```
        lw 0 1 one      //r1 = 0x1
        lw 0 4 four     //r4 = 4
loop    beq 3 4 end     //beqA
        nor 3 3 5
        nor 1 1 2
        nor 2 5 5       //r5 = r3 & r1
        beq 0 5 endif   //beqB
        add 6 1 6       //r6 counts the odd numbers
endif   add 3 1 3
        beq 0 0 loop    //beqC
end     halt
one     .fill 1
four    .fill 4
```

For each branch, write the sequence of branch decisions for each beq instruction. Let “taken” be denoted as “T” and “not taken” as “N”. Additionally, indicate the fraction of each branch that we would predict **correctly** for each of these 3 prediction schemes (list your answer as a fraction)

- Scheme X: Predict always not taken
- Scheme Y: Predict backwards taken, forwards not taken
- Scheme Z: Local[1] 2-bit Branch Predictor: Initialized to weakly taken

Branch	**Outcome Sequence**	**X prediction rate**	**Y prediction rate**	**Z prediction rate**
beqA	NNNNT	4/5	4/5	3/5
beqB	TNTN	2/4	2/4	2/4
beqC	TTTT	0/4	4/4	4/4

[1] "Local" describes the branch predictor discussed in class and homework, where predictions are made considering only one branch's history

Problem 6: Virtual Memory 20 points

Thomas Anderson is developing the NEO architecture and is designing its virtual memory implementation. Here are some parameters of the NEO architecture's virtual memory implementation:

- Page size: 256 bytes
- Virtual address size: 24 bits
- Physical address size: 15 bits
- Byte-addressable architecture
- Physical page 0 is reserved for the OS and is never allocated to any process.
- Physical page #0 (reserved for OS) and pages allocated to page tables cannot be evicted to disk.

Some other parameters of NEO are changed in the following sub-questions. For each sub-question, use the appropriate given values of the parameters.

A page is allocated to hold the page table for a given process immediately as that process is launched using the criteria listed above. No launched processes are terminated.

1. Thomas initially decides to use single-level page tables for his virtual memory implementation. Assume that each single-level page table entry has space for exactly a page number and a valid bit, and uses as few bytes as possible. How many pages are taken up by a given process's single-level page table? Show your work. *[4 points]*

Page offset = log2(256) = 8
VPN bits = 24 - 8 = 16
PPN bits = 15 - 8 = 7
Page table entry size = 7 bits PPN + 1 valid bit = 8 bits = 1 byte
Single level page table size = 1 byte * 2^16 VPNs / 2^8 bytes per page = 2^8 pages
Number of pages:

256 pages

2. Thomas now decides to use 2-level page tables to avoid waste. Assume that each 1st level page table entry has space for an address and a valid bit, and uses as few bytes as possible. Assume that each 2nd level page table entry has space for a page number and a valid bit, and uses as few bytes as possible. If the 1st level page table of a process must be as big as possible while fitting into a single page, how many bits of the virtual address should be used to index into each level of the page table? Your answer should be a range of indices. Show your work. *[6 points]*

# bits for first level	# bits for second level
7 (23-17)	9 (16-8)

1st level entry size = 15 bits (Phys Addr is 15 bits) + valid = 2 bytes
1st level entries = 2^8 (Page Size) / 2 (Size of a 1st Level PTE) = 128 entries
1st level index bits = 7 (log2(#1st level entries))
2nd level entry = 1 byte (PPN bits = 7 + 1 valid bit) = 8 bits
2nd level index bits = 16 - 7 = 9

Using the 2-level page table described earlier, what is the **minimum** size of a process's page table (across all levels, in bytes) when using a multi-level page table on NEO? *[3 points]*

1 first level, 0 second levels
2^7 * 2 = 256 bytes

Size in bytes:

256

3. Using the 2-level page table described earlier, what is the **maximum possible** size of a process's page table (across all levels, in bytes) when using a multi-level page table on NEO? Assume the following:
    a. No physical pages are shared between different processes
    b. If a second-level page table has all its entries invalidated, it will be immediately deallocated.
    c. The OS will assign a process a maximum of 40 pages (not including the page tables).

(**Hint:** How many mappings can a given process have in its page table in this system?) *[4 points]*

Maximum size occurs when 1 page allocated per second level page table
Implies 40 second level page tables

2^7*2 + 40*2^9 = 256 + 20480 = 20736

Size in bytes:

20736

4. Thomas notices that the latency of his memory accesses has increased significantly due to his multi-level page table implementation, making his `noSpoon` benchmark application run quite slowly. To fix this issue, Thomas decides to add a 4-entry TLB to the system. The TLB is direct-mapped, and the last 2 bits of the virtual page number are used to index into the TLB.

   After adding the data-specific TLB (i.e. it is not used to translate instruction addresses) to the NEO system, Thomas notices that the runtime of the `noSpoon` program is only slightly reduced, which surprises him. The `noSpoon` program spends most of its time running the `computeCount` function below. `arr` is a 64-entry array of `dataVals` structs. Each `dataVals` struct takes up 1KB (including all its required padding).

```
1  int computeCount(dataVals *arr) {
2    int count = arr[0].numVals;
3    for(int i = 1; i < 16; i++) {
4      count +=  arr[i+4].numVals;
5      arr[0].numVals = count;
6    }
   }
```

Which of the following best describes why the TLB is not significantly improving the runtime? *[3 points]* EDIT: there may be multiple reasonable answers. You only need to select one

- ☐ The program does not re-perform the same translations
- ☑ ~~Translations keep knocking each other out of the TLB~~
- ☑ ~~The TLB is not large enough to hold all the translations~~
- ☐ A TLB is not expected to improve latency, just prevent programs accessing each other's memory.

Problem 7: Pipeline Performance 20 points

Consider adding "MULT" as an R-type instruction (same encoding format as ADD and NOR) to LC2K to perform multiplications. We build an implementation of the pipeline using 8 stages:

IF-> ID1 -> ID2 -> EX1 -> EX2 -> EX3 -> MEM -> WB

Assume the following:

- The system has a clock frequency of 2 GHz (2 billion Hz)
- Result of MULT is available at the start of MEM
- Result of ADD/NOR is available at start of EX3
- Forwarding is used to resolve data hazards when possible, and all values are forwarded to EX1.
- A static predict-not-taken branch predictor is used to resolve ~~data~~ control hazards via speculate-and-squash. Branches are resolved in the MEM stage as in the lecture's pipeline.
- Separate instruction and data caches are used, both return values on "hits" within a single cycle. Both have hit rates of 80%.
- Cache misses are passed to main memory, which has a latency of 100 ns (ns = 10^{-9} seconds)

1. Define the "window size" of instruction X to be the minimum number of noops (or any other independent instruction) that would need to be inserted between X and a dependent use of X for no stalls to occur. For example, in our 5-stage pipeline discussed in class, the window size of "lw" is 1, since 1 noop would need to be in between "lw" and a dependent use for no stalls to occur.

   Fill in the table below with the appropriate window size of each instruction. *[6 points]*

Instruction	**Window Size**
ADD/NOR	1
LW	3
MULT	2

2. For the following LC2K assembly snippets, list how many noops will be inserted by the hardware for stalls + squashes (assume all cache accesses hit). *[6 points]*

Benchmark 1:

```
lw      0   1   data
add     1   2   3
nor     3   1   4
```

# of noops: _4_____
LW + ADD

Benchmark 2:

```
add     0   1   2
add     1   1   3
lw      2   3   1
```

# of noops: _0_____

Benchmark 3:

```
nor     0   1   2
lw      1   2   3
nor     3   1   4
```

# of noops: __0_____

Benchmark 4:

```
mult 0 1 2
add  1 2 3
lw   0 3 2
```

# of noops: __2_____
MULT

Benchmark 5:

```
beq 0 0 1
lw  0 2 1
add 2 2 3
```

# of noops: __6_____

Benchmark 6:

```
lw  0 1 0
beq 1 1 1
nor 7 7 7
add 1 1 2
```

# of noops: _9_____
branch + lw

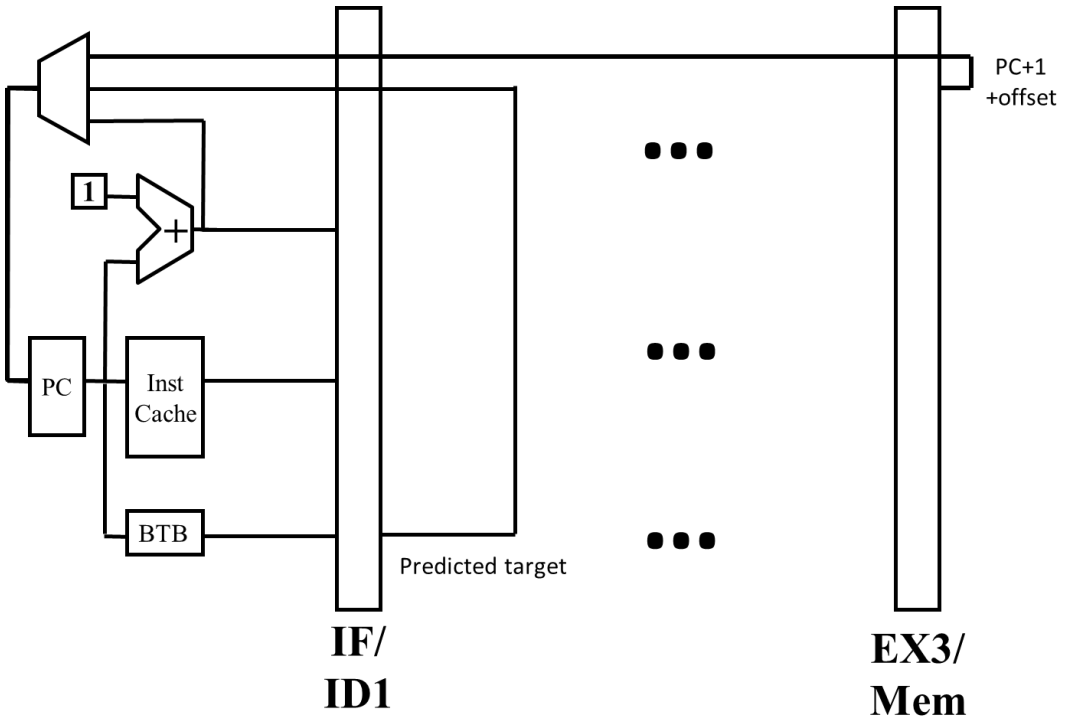
3. Over the course of 1 million instructions, how many cycles of stalls do we expect due to instruction cache misses? *[3 points]*

Clock period = 0.5 ns
Memory latency = 100 ns / 0.5 ns per cycle = 200 cycles
1 million * 200 * .2 = 40 million

Number of cycles:

40 million

4. Say we replace our static predictor with a dynamic one. The BTB and direction predictor are accessed in IF, but the result is not available until the start ID1 (at which point we've already started fetching the next PC value). Thus, branches are always initially predicted "not taken" during IF, but if the predictor indicates "taken", we send the predicted target back during the ID1 stage and squash any mis-fetched instructions on the next clock edge. Mispredicted branches are then resolved in the MEM stage as in our normal pipeline, squashing when appropriate. A snippet of the pipeline (with several stages omitted) is shown below.



If 20% of instructions are branches, 80% of branches are predicted taken, and 10% of both predicted-taken and predicted-not-taken branches are mispredicted, what is the increase in CPI ovezr the base value of 1 due to branches? *[5 points]*

For each predicted taken branch, add 1 cycle

For every branch we mispredicted as taken, add 5 cycles (one stage is already a noop)

For every branch we mispredicted as not taken, add 6 cycles

.2 * .8 * 1 = .16 .2*.8*.1*5 = .08 .2*.2*.1*6 = .024 Total: .264	Alternate approach: .2 * .8 * .9 * 1 = .144 (correctly predict T) .2 * .2 * .9 * 0 = 0 (correctly predict NT) .2 *.1 * 6 = .12 (mispredicted) Total = .264

THIS PAGE INTENTIONALLY LEFT BLANK