Computer System Architecture
6.823 Quiz #1
March 19th, 2021

Name: _____SOLUTIONS_____

90 Minutes
17 Pages

Notes:

- Not all questions are equally hard. Look over the whole quiz and budget your time carefully.
- Please state any assumptions you make, and show your work.
- Please write your answers by hand, on paper or a tablet.
- Please email all 17 pages of questions with your answers, including this cover page. Alternatively, you may email scans (or photographs) of separate sheets of paper. Emails should be sent to 6823-staff@csail.mit.edu
- Please ensure your name is written on every page you turn in.
- Do not discuss a quiz's contents with students who have not yet taken the quiz.
- Please sign the following statement before starting the quiz. If you are emailing separate sheets of paper, copy the statement onto the first page and sign it.

I certify that I will start and finish the quiz on time, and that
I will not give or receive unauthorized help on this quiz.

Sign here: _____

Part A	_____	20 Points
Part B	_____	40 Points
Part C	_____	40 Points
TOTAL	**_____**	**100 Points**

Name _____

Part A: Caches (20 Points)

Ben Bitdiddle wants to run the following code on his machine:

```
uint32_t A[32], B[32];
uint32_t d;
...

for (uint32_t rounds = 0; rounds < 10; rounds++) {
  for (uint32_t i = 0; i < 32; i++) {
    B[i] = d*A[i];
  }
}
```

His machine has a direct-mapped data cache that has 16 lines with 16 bytes per line. Note that arrays `A` and `B` have 4-byte elements. Array `A` starts at address 0x0000 and `B` starts at address 0x0100. The values of `i`, `d`, and `sum` are stored in registers.

Question 1 (3 points)

How many compulsory, capacity, and conflict misses will occur when running the above code? (Focus on data accesses only, not on instruction accesses.)

Direct-mapped with 16 lines and 16B per line = 4 bit block offset, 4 bit index
Entries A[i] and B[i] conflict at index = floor(i/4)
Since our access pattern is A[i]->B[i]->A[i+1]->B[i+1]->... we always conflict.
Compulsory misses = 8 (entries of A) + 8 (entries of B) = 16
Capacity misses = 0 (we would not solve this problem by increasing capacity
Conflict misses = Total misses - Compulsory misses = 640 - 16 = 624

Question 2 (5 points)

Ben changes the configuration of the cache to be 2-way set-associative with 8 sets and 16 bytes per line. The cache uses a Least Recently Used (LRU) replacement policy. With the new cache, how many total misses will occur when running the above code?

Now, all previous conflict misses are hits due to sufficient associativity. Thus, total misses = 16

Name _____

Question 3 (7 points)

Alyssa P. Hacker comes by and changes the code such that elements of array `C` are added to the final result:

```
uint32_t A[32], B[32], C[32];
uint32_t d;
...

for (uint32_t rounds = 0; rounds < 10; rounds++) {
  for (uint32_t i = 0; i < 32; i++) {
    B[i] = d*A[i] + C[i];
  }
}
```

Assume that array `C` starts at address 0x0200.

(a) (3 points) With Ben's 2-way cache configuration from Question 2, how many total misses will occur when running this code?

Now our access pattern is A[i]->C[i]->B[i]->A[i+1]->C[i+1]->B[i+1]->...
Notice that due to LRU policy, we will kick out the line containing A[i+1] (brought in just before by access to A[i]) when we access B[i]. Same goes for entries of C and B. Thus, we get all misses again = 32*3*10 = 960.

(b) (4 points) Can a different replacement policy further reduce the number of misses? If so, briefly describe such a policy (do not assume knowledge of the future, i.e., Belady's MIN is not an option). If not, briefly state your reasoning.

There are multiple solutions. One simple policy is Most Recently Used (MRU), which kicks out the most recently accessed line among the different ways. Thus, when B[i] is accessed we would evict C[i], turning a previous miss into a hit.

Name _____

Question 4 (5 points)

We notice that a program we want to run is mostly accessing consecutive memory locations in sequence. A way to reduce the effect of misses for such an access pattern is to *prefetch* the data before it is actually needed by the processor. For example, given an access stream of `A`, `A+1`, `A+2`, ... where `A` is the location in memory initially accessed by the program, we can prefetch the data at addresses `A+3`, `A+4`, ... ahead of the processor, so that they are already in the cache by the time the processor explicitly requires them.

Assume that our processor is issuing a memory request every 4 cycles on average, and the average memory access latency is 100 cycles. Also assume that the prefetcher predicts future accesses perfectly. How many prefetch requests must be in flight on average to prefetch the data required by the processor on time?

Memory issue throughput = T = 1/4
Memory latency = L = 100
Number of requests needed in flight = N
Relationship is given by Little's Law: N = T * L = 25 requests in flight

Name _____

Part B: Caches and Virtual Memory (40 Points)

Question 1 (10 points)

Answer whether the following statements are **True** or **False** (2 points each).

(a) Translation Lookaside Buffers (TLBs) usually have low associativity to enable efficient virtual to physical address translations.

False. We want high associativity to minimize misses since page table walks are expensive.

(b) In a multilevel cache hierarchy, using exclusive caches achieves a higher effective capacity than using inclusive caches of same size.

True. Lines are never duplicated across multiple levels.

(c) You can only swap out pages containing page table entries that reference physical pages residing in secondary storage.

True. Swapping out pages containing PTEs that reference pages residing in primary memory may result in deadlock.

(d) Doubling the number of cache lines of a direct-mapped cache (while leaving all other parameters unchanged) always decreases conflict misses.

False. There are corner cases where this does not work, such as only the high order bits of the tag being different between conflicting lines.

(e) A larger cache, given fixed associativity and block size, will always reduce the average memory access time (AMAT).

False. AMAT = hit latency + miss rate * miss penalty. While large cache may reduce miss rate, it may also increase hit latency.

Name _____

Question 2 (8 points)

Consider a machine with byte-addressable memory, 24-bit virtual addresses, and 24-bit physical addresses. We want to compare two different page table designs for this machine.

The machine executes the following piece of code:

```
int A[262144]; // A 1MB array
int sum;

while (1) {
  for (int i = 0; i < 256; i++) {
    sum += A[i*1024];
  }
}
```

Array `A` starts at virtual address `0x800000`. Ignore accesses to the stack and code regions and only focus on accesses to the 1MB array.

Alyssa P. Hacker first wants to use a linear (i.e., single-level) page table with 4KB pages. Each page table entry is 32 bits. Assume there is no TLB in this machine.

(a) (2 points) Derive the number of bits needed for the physical page number and page offset.

4KB pages = 2^12B -> 12 bits of page offset, 12 bits of physical page number.

(b) (2 points) What is the maximum size of the page table?

Maximum size is Entry size * #Page Table Entries = 4B * 2^12 = 16KB

(c) (2 points) What is the size of the page table when executing the above code?

This is the **same** as the maximum size since we must allocate the entire page table for each running process.

If you assume that there is smart bounds checking going on (e.g., Kernel notices you only access 1MB of heap and assigns a limited page table to your process), then:
0x800000 starting address for array = 2^11 pages needed before the array
1MB array = 256 pages needed
So, we need 2^11+ 256 PTEs in total = (2^11 + 256)*4B = 9KB

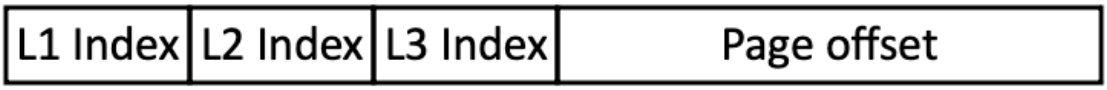
(d) (2 points) How many memory references to the page table are required to execute one iteration of the while loop?

256, since this page table has a single level.

Name _____

Question 3 (7 points)

Alyssa now switches to a 3-level hierarchical page table. The virtual address is divided in the following way:



Here, the L1, L2, and L3 indices evenly divide the bits that exclude the page offset (i.e., the three fields are the same size). Each entry of the L1, L2, and L3 page tables is 32 bits.

(a) (2 points) What is the maximum size of the page table?

Maximum size is 1 L1 table, 16 L2 tables, and 256 L3 tables. Each page table is 16 entries = 2^6 Bytes.
Thus, 2^6(1 + 16 + 256) = 17472 Bytes

(b) (3 points) What is the size of the page table when executing the above code?

12/3 = 4 bits of index per level.
We access 1MB of array sequentially, visiting each page contained in the array.
1MB = 2^20B => require 1 L1 entry, 16 L2 entries, and 16*16 L3 entries.
This translates to 1 L1 table, 1 L2 table, and 16 L3 tables.
Thus, 2^6(1 + 1 + 16) = 1152 Bytes

(c) (2 points) How many memory references to the page table are required to execute one iteration of the while loop?

256*3 references, 256 for each level.

Name _____

Question 4 (5 points)

Alyssa wants to add a TLB to speed up address translation. What is the minimum number of entries necessary to have no TLB misses in steady state for the above code?

We access 256 distinct pages, so we need at least 256 TLB entries.

Question 5 (5 points)

Alyssa also decides to implement a virtually-indexed, physically-tagged cache. Her cache is 4-way set-associative with 512 lines and 32 bytes per line. Can her cache have aliasing issues? Briefly explain.

512 lines with 4-ways => 512/4 = 128 sets => 7 bits of index
32 bytes per line => 5 bits of block offset
Since index + block offset equals the page offset (12 bits), we do not have aliasing issues since the virtual index does not overlap with the physical tag.

Name _____

Question 6 (5 points)

Alyssa now wants to add support for large pages, i.e., pages that are larger than 4KB.

(a) (2 points) Given her 3-level hierarchical page table, what are the possible large page sizes while using the same page table design?

We can merge entries of lower level tables into one big table
=> 64KB pages using L1 and L2 page tables, and 1MB pages using only L1 page table.

(b) (3 points) Using large pages, what is the **minimum** number of TLB entries needed to eliminate all misses from the TLB?

Using 1MB pages, we only need a single TLB entry to store the entire array in one big page.

Name _____

Part C. Pipelining (40 Points)

Ben Bitdiddle wants to run the following C code on his 5-stage pipelined MIPS processor:

```
int A[100];
int B[100];
int difference = 0;
int matches = 0;
...

for (int i = 100; i != 0; i--) {
  int a = A[i];
  int b = B[i];
  if (a == b)
    matches++;
  else
    difference += (a - b);
}
...
```

The equivalent MIPS assembly code segment is shown below. We assume the following at the start of the code segment:

- Registers `r1`, `r2`, `r3`, and `r4` hold the values of `a`, `b`, `matches`, and `difference`, respectively.
- Register `r5` holds the value of loop iteration index.
- Registers `r6` and `r7` hold the base address of arrays `A` and `B`, respectively.
- Registers `r8` is used for intermediate results.

```
          ...
          ADDI r5, r0, 100
  _loop:  LW   r1, 0(r6)      ; a = A[i]
          LW   r2, 0(r7)      ; b = B[i]
          ADDI r6, r6, 4
          ADDI r7, r7, 4
          SUB  r8, r1, r2
          BNEZ r8, _else      ; Jump to _else label if a != b
          ADDI r3, r3, 1      ; matches++;
          J    _then
  _else:  ADD  r4, r4, r8     ; difference += (a - b);
  _then:  ADDI r5, r5, -1     ; i--;
          BNEZ r5, _loop      ; Check loop bound
          ...
```

Name _____

Ben notices that his simple 5-stage pipelined processor suffers from frequent control hazards on branches. To eliminate branches, Ben decides to implement predicated execution in his MIPS processor. Refer to the accompanying handout for more details on predicated instructions.

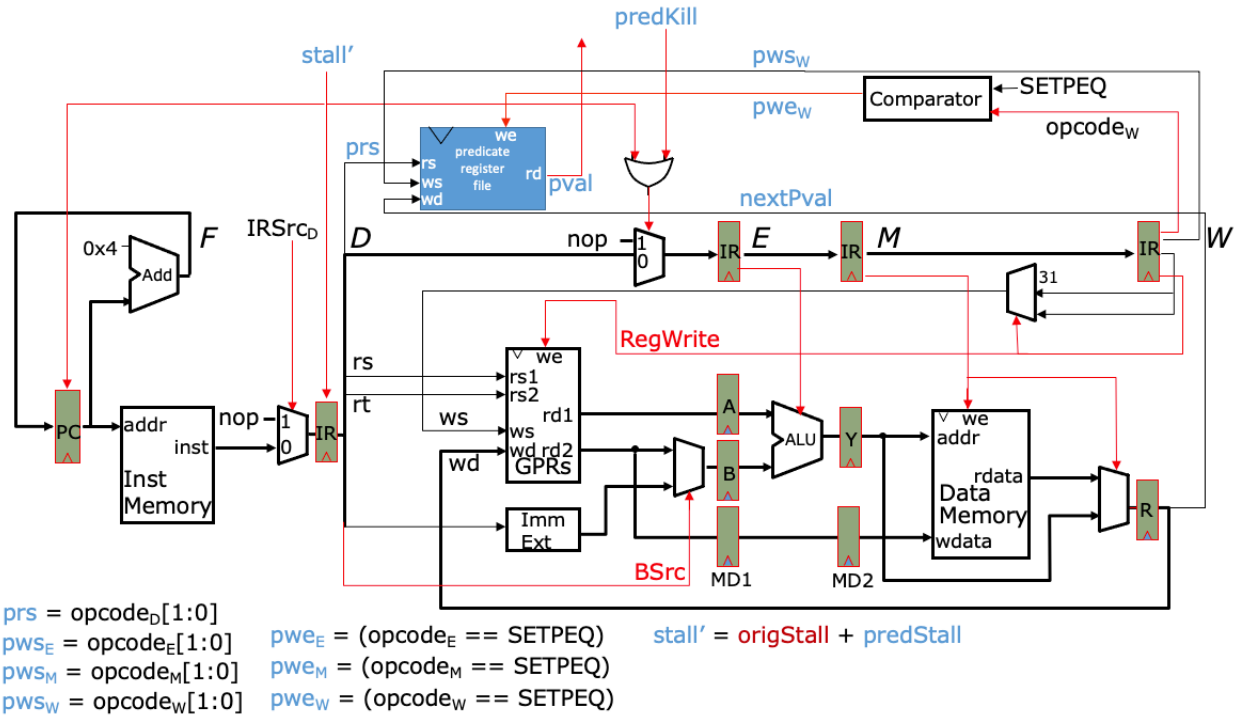
Question 1 (5 points)

Rewrite the code denoted by the red box using predicated instructions instead of conditional branches and jumps. Your code should not have any branches or jumps remaining.

```
       SETPEQ p0, r1, r2
 (p0)  ADDI r3, r3, 1
(!p0)  SUB r8, r1, r2
(!p0)  ADD r4, r4, r8
```

Name _____

To implement predicated instructions, Ben adds a *predicate register file* to the Decode stage of his pipeline, as shown below (for simplicity, this pipeline diagram does not handle control instructions). New control signals and datapaths are highlighted in blue.



Every predicated instruction and SETPEQ has its source/destination predicate register address encoded in the bottom two bits of its opcode. When a predicated instruction enters the decode stage, it uses *prs* to select the predicate register to read. The register file outputs *pval*, which is the corresponding predicate register value. Predicated instructions must be treated as no-ops if the predicate register value is 0. To do this, Ben implements a new kill condition *predKill*.

The SETPEQ instruction sets the destination predicate register pws_W and the predicate value *nextPval* in the writeback stage. Thus, predicated instructions must stall in the decode stage if the pipeline has a SETPEQ instruction in flight that will write to the same predicate register. This requires Ben to implement a new stall signal *stall'*, which adds an additional stall condition *predStall* on top of the original stall signal *origStall*.

Finally, Ben implements a new control signal pwe_W that enables writes to the predicate register when SETPEQ reaches the writeback stage.

Name _____

Question 2 (6 points)

Derive the Boolean expression for new control signals *predStall* and *predKill*. You are also provided the following two control signals.

PredIns = True if instruction in decode is predicated and reads the predicate register normally (i.e., the instruction is executed normally if predicate register is True / 1).

InvPredIns = True if instruction in decode is predicated on the ***inverse*** of the predicate register. (i.e., the instruction is executed normally if predicate register is False / 0).

predStall = *(PredIns || InvPredIns) &&*
((pwe$_E$ && (pws$_E$ == prs) ||
(pwe$_M$ && (pws$_M$ == prs) ||
(pwe$_W$ && (pws$_W$ == prs))

predKill = *(PredIns && !pval) || (InvPredIns && pval)*

Name _____

Question 3 (4 points)

Ben now adds **full bypassing** for register values. Ben runs the original code with branches and jumps on his new processor. How many cycles does it take to execute the code in the red box? Assume branches are resolved in the Execute stage and jumps are resolved in the Decode stage, and there are no branch delay slots. You only need to consider one iteration of the loop.

Recall the original code:

```
          SUB  r8, r1, r2
          BNEZ r8, _else     ; Jump to _else label if a != b
          ADDI r3, r3, 1     ; matches++;
          J    _then
  _else:  ADD  r4, r4, r8    ; difference += (a - b);
  _then:
```

Let's consider both cases of BNEZ being taken and not taken.

1. BNEZ taken:
2 cycles for SUB and BNEZ
2 cycles of bubbles to resolve the branch at Execute stage
1 cycle for the ADD
total of 5 cycles

2. BNEZ not taken:
4 cycles for SUB, BNEZ, ADDI, J
1 cycle of bubble to resolve the jump at Decode stage
total of 5 cycles

Question 4 (4 points)

Now Ben runs the predicated code on his new processor with full bypassing for register values. How many cycles does it take to execute the code in the red box? You only need to consider one iteration of the loop.

4 cycles + 3 cycles of stalling to resolve data hazard between SETPEQ and following ADDI = 7 cycles

Name _____

Question 5 (4 points)

Ben adds full bypassing for predicate register values as well. How many cycles does it take to execute the code in the red box? You only need to consider one iteration of the loop.

We've removed all stall cycles, so it's 4 cycles

Question 6 (4 points)

Ben wants to write to the predicate register file in the Execute stage instead of waiting until Writeback. This would allow removing most of the bypass paths for predicate registers. Can this result in incorrect behavior? Justify your reasoning.

If we write the predicate register file in the E stage, we can have a case where an invalid instruction writes to the predicate register file. Consider the case when an instruction preceding SETPEQ (i.e., an instruction that is in the later pipeline stage) triggers an exception in the M stage when SETPEQ modifies the predicate register file in the E stage. This would cause the result of SETPEQ to be reflected in the predicate register file when it shouldn't.

Name _____

Question 7 (4 points)

Alyssa notes that predicated instructions need not read the predicate register in the decode stage. As long as the instruction is not changing any architectural state, the predicated instruction can flow normally through the pipeline until it is necessary to read the predicate register. Given this information, where is the latest stage Ben can put the predicate register file? Take exceptions into account.

We need to read the predicate register file before the Memory stage so that a predicated store that should be a no-op does not reflect its change in the architectural state. Thus, we can delay reading until the Execute stage.

If you assume that you can perform a predicate register read and write to memory in the same cycle, we can put the predicate register file in the Memory stage.

Question 8 (4 points)

In Question 2, we defined the new stall signal as *stall' = origStall + predStall*. If you want to minimize the number of stalls, is this the best we can do, or could you change the stall signal to avoid some stalls? Briefly explain why or why not. (You do not need to write the precise stall signal.)

Based on your answer to the question above, if we wanted to maximize performance and, for equal performance, minimize complexity, which would you choose: (1) keeping the predicate register file in Decode, with bypasses; or (2) placing the predicate register file at a later stage as in Question 7, to reduce/avoid bypasses?

Say the predicated instruction in the Decode stage should be killed due to the predicate being False, but is stalling due to a data dependence on a prior instruction (e.g., a lw). Then, we can kill the instruction early instead of stalling until the data hazard is resolved.

With this optimization, we will need the predicate register in the Decode stage (1) to kill the predicated instruction as early as possible.

Name _____

Question 9 (5 points)

Ben recompiles Linux to target his new ISA and runs it on his new MIPS processor. To his dismay, it crashes immediately. He notices that the OS kernel does not save/restore predicate registers on context switches. Help Ben fix the kernel by writing code that saves the predicate registers upon a context switch. You only need to write code to store one predicate register, say `p1`, into one general-purpose register, say `r1`.

```
     XOR  r1, r1, r1  ; Not needed if r1 is already set to 0
(p1) ADDI r1, r0, 1
```