## **CS/ECE 552: Introduction to Computer Architecture**

## **Spring 2020**

## **Professor Matthew D. Sinclair**

## **Final Exam**

## **Thursday, May 7th, 2020**

## **Weight: 20%**

## **TAKE-HOME EXAM**

## **NAME:**

Read over the entire exam before beginning. You should verify that your exam includes all of the problems listed in the table below.

This exam is intended to take 2 hours, although since it is a take-home exam you may take longer if you so choose. The exam will be **due at 2:25 PM Central on Thursday, May 7th**, **no exceptions**. You should submit your solution to Canvas, and we will be uploading these solutions to Gradescope to grade (similar to the midterm). You are expected to turn in either a) a scanned, handwritten PDF of the exam with your solutions or b) a typed PDF of the exam with your solutions. If possible, please do not change where the questions are and answer in the provided space, as this will make it easier for Gradescope to identify where the answers are.

Regardless of which option you choose, we will be checking for academic misconduct (including checking online tutoring services like Chegg), and you are expected to take the exam yourself and write your own answers. Any issues will be handled according to the UW Academic Misconduct policy.

Since the exam is being released on May 5th, also known as *Revenge of the Fifth*, to honor this the problems have been appropriately named.

|  |  |  |
| --- | --- | --- |
| Problem | Possible Points | Points |
| **Problem 1** | 10 |  |
| **Problem 2** | 12 |  |
| **Problem 3** | 12 |  |
| **Problem 4** | 10 |  |
| **Problem 5** | 10 |  |
| **Problem 6** | 22 |  |
| **Total** | 78 |  |

**Problem 1 [10 points] *The Phantom Parallelism***

**Part A [2 points]**

You are running a program *A*, which is capable of running 4 threads in parallel, on a processor *Q* that has 8 superscalar, out-of-order cores. If your program can be evenly parallelized across all 4 threads, what is the speedup of *A* on *Q*? Assume that *Q* does not implement fine-grained multithreading (FGMT), coarse-grained multithreading (CGMT), or simultaneous multithreading (SMT). To receive full credit, you must show your work or justify your answer.

**Part B [2 points]**

Now you are running a program *B* (where *B* is a different program than *A*), which is capable of running 8 threads in parallel, on a processor *U* that has 4 superscalar, out-of-order cores. If your program can be evenly parallelized across all 8 threads, what is the speedup of *B* on *U*? Assume that *U* does not implement FGMT, CGMT, or SMT. To receive full credit, you must show your work or justify your answer.

**Part C [6 points]**

Other than adding more cores to *U*, an option to improve the performance of *B* on *U* is to add support for multiple threads running on a single processor. In this situation, would it be best to add FGMT, CGMT, or SMT support to *U* if your goal is to improve performance? Why? To receive credit, you must justify your answer.

**Problem 2 [12 points] *Translation Wars***

Consider a memory system with the following parameters. The virtual address is 64 bits (v63-v0 where v0 is least significant). The physical address is 46 bits (p45-p0). The page size is 32 kilobytes (KB). The translation lookaside buffer (TLB) has 32 entries total, is two-way set-associative, and is accessed *in parallel* with cache. The cache is unified (i.e., instructions and data), 32 KB, four-way set-associative, 128-byte blocks (lines), and has LRU replacement. All addresses are byte addresses.

**Part A [4 points]**

Consider a naïve page table that stores an entry for every page in the virtual address space. How many entries would the page table store? In each page table entry, which bits of the physical address would be stored? To receive credit, you must justify your answer.

**Part B [4 points]**

Describe at least one technique for improving on the naïve page table of part A? To receive credit, you must justify your answer.

**Part C [4 points]**

For the TLB in this problem, which bits will be stored in tag and which will select the set (the index)? To receive full credit, you must show your work or justify your answer.

**Problem 3 [12 points] *Revenge of the Cache***

Data cache performance is extremely important for modern processors, which is why architects often spend significant amounts of time optimizing their behavior. Assume that the following code snippet is from an application that is important to your company:

int main(int argc, char \* argv[])

{

int arrSize = atoi(argv[1]);

int offset = atoi(argv[2]);

…

int \* arrA = (int \*)malloc((arrSize + offset) \* sizeof(int));

int \* arrB = (int \*)malloc(arrSize \* sizeof(int));

int \* arrC = (int \*)malloc(arrSize \* sizeof(int));

…

for (int i = 0; i < arrSize; ++i)

{

arrB[i] = arrC[i] + arrA[i + offset];

}

…

}

Your job is to determine what the performance of the cache will be under the following assumptions:

* The cache block size is 16 bytes
* Array entries are 4 bytes
* The cache is initially empty
* Local variables are stored in registers, not memory
* The base address of arrA is 0x0010, arrB is 0x0050, and arrC is 0x0070
* arrB and arrC each have 8 elements, while arrA has 16 elements
* The inputted offset is 8
* Your cache is direct-mapped and 128 Bytes large
* All addresses are 32 bits, and byte addressable
* You can ignore translation, and assume only physical addresses are used
* In each loop iteration arrA is accessed first, then arrC, and finally arrB.

**Part A [8 points]**

Given these assumptions, what is the hit rate for the given program? To receive credit, you must justify your answer and/or show your work.

**Part B [4 points]**

Which type of cache misses are the dominant source of cache misses in part A? Name one optimization you can apply to improve performance for this type of cache miss. To receive credit, you must justify your answer.

**Problem 4 [10 points] *A New Design***

**Part A [6 points]**

You are tasked with optimizing the memory subsystem for your company’s latest processor, a 5-stage, in-order MIPS processor, like the one discussed in class and in the project. When you are testing your design, you notice that you have accidentally connected your instruction cache in the Memory stage, and your data cache in your Fetch stage! *Assuming you are unable to fix this (e.g., your cache design has been frozen by another team at your company)*, what are the implications of this mistake? Assume that your instruction cache has been optimized such that it can only read memory, while your data cache can read or write memory, that both of your caches are write back caches, and that your instruction and data caches are backed by global memory. Suggest a fix at either the hardware (except for changing or switching the cache design) or software level, or an provide explanation why no fix is possible. To receive credit, you must justify your answer.

**Part B [4 points]**

In your cache design for Phase 2.3 of the project, one of the most expensive operations was when the cache line being evicted was dirty. Why is this operation so expensive? To receive credit, you must justify your answer.

**Problem 5 [10 points] *Reliability Strikes Back***

You are in charge of choosing the RAID system your company will use. Your CTO has informed you that the final choices are a 6-disk system using either RAID-1 or RAID-5. What are some of the criteria you should use to determine which setup is superior? To receive credit, you must justify your answer.

**Problem 6 [22 points] *Return of the MIPS R10K***

Consider the following MIPS assembly program running on the single-issue R10K-like processor discussed in class (F, De, Di, S, X, C, R):

1. ADD $1, $2, $3
2. SUB $4, $5, $1
3. OR $6, $1, $4
4. AND $7, $4, $6
5. LW $9, 4($3)
6. ADD $1, $9, $2
7. SW $1, 4($7)

Assume that all instructions take one cycle in the Execute (X) stage except for loads and stores, which take 2 cycles in X. Also assume that there are infinitely many reservation stations, functional units, reorder buffer entries, dispatch buffer entries, and instruction buffer entries, but are *limited to the physical registers in the free list at the start of the program* (see below).

Complete the pipeline diagram below. On the next page, we have also included the map table, free list, reservation stations/functional units, and reorder buffer, for your convenience.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instr / Cycle** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** |
| ADD $1, $2, $3 | F | De | Di | S | X | C | R |  |  |  |  |  |  |  |  |  |  |  |  |
| SUB $4, $5, $1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| OR $6, $1, $4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| AND $7, $4, $6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| LW $9, 4($3) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ADD $1, $9, $2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| SW $1, 4($7) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Map Table** | | |
| **Architectural Reg** | **Physical Reg** | **Ready (Y/N)** |
| $1 | P1 | Y |
| $2 | P2 | Y |
| $3 | P3 | Y |
| $4 | P4 | Y |
| $5 | P5 | Y |
| $6 | P6 | Y |
| $7 | P7 | Y |
| $8 | P8 | Y |
| $9 | P9 | Y |

|  |  |
| --- | --- |
| **Free List** | P10, P11, P12 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Reservation Stations and Functional Units** | | | | | |
| **Functional Unit** | **Instruction** | **Operand (rs)** | | **Operand (rt)** | |
| **Physical Reg** | **Ready (Y/N)** | **Physical Reg** | **Ready (Y/N)** |
| add/sub |  |  |  |  |  |
| and |  |  |  |  |  |
| or |  |  |  |  |  |
| lw |  |  |  |  |  |
| sw |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Reorder Buffer** | | | |
| **Instruction** | | **Dest Physical Reg**  **(rd)** | **Old Physical Reg**  **(rd)** |
| **Head (Oldest)**  **Tail (Youngest)** |  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Part B [2 points]**

If the reorder buffer from Part A were not infinitely sized, how many entries would it need to avoid stalls due to a lack of reorder buffer entries? To receive credit, you must justify your answer.