CS-UY 2214 — Homework 7

Jeff Epstein

Introduction

Unless otherwise specified, put your answers in a plain text file named `hw7.txt`. Number each answer. Submit your work on Gradescope.

You may consult the E20 manual, which is available on Brightspace.

Problems

1. Consider the following data structure, found in the file `linked.s`, expressed in E20 assembly language:

```
main:
    # TODO: your code here

chain4:
    .fill 100
    .fill chain5
    .fill 300
chain7:
    .fill 16384
    .fill 909
    .fill 0
chain1:
    .fill 23
    .fill chain2
head:           # beginning of linked list
    .fill 34
    .fill chain1
    .fill 82
    .fill 10
chain3:
    .fill 4
    .fill chain4
    .fill 229
    .fill 449
chain2:
    .fill 0
    .fill chain3
chain5:
    .fill 12
    .fill 0     # end of linked list
    .fill 9999
chain6:
```

```
    .fill 99
    .fill 49
```

The above code describes a linked list, starting at memory address `head`. Each node in the linked list occupies two memory cells: the cell at address n stores a 16-bit number, an element of the list. The cell at address $n+1$ stores either the pointer to the next node, or zero, indicating that there is no next node.

In other words, the memory layout of the nodes of this linked list correspond to this C++ data structure:

```
struct Node {
    int value;      // at offset +0
    Node *next;     // at offset +1
};
```

In addition to the linked list itself, the above code includes certain "garbage values" that, although present in memory, are not part of the linked list.

Your task is to write a program in E20 assembly language that will calculate the sum of all elements in the above linked list. Your program must iterate through the list until it reaches its end. The only data label that your program may refer to by name is `head`; however, you may define and use your own labels within your code. Your program should leave the sum in register `$1` before halting.

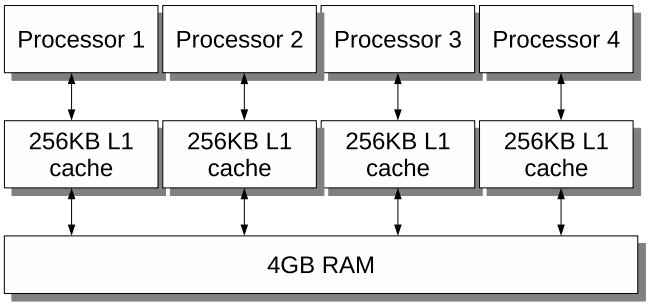
Your solution must be thoroughly commented, or else it will not be graded. Make sure that the comments help the grader understand the intent of your code.

Include the provided data structure in your submission. Submit your complete file named `linked.s`.

2. We have a processor with a single cache. Every cache access (whether it results in a hit or a miss) takes 4ns. Accessing main memory takes 40ns. A particular program executes a series of `lw` instructions, during which it experiences a 98% hit rate. What is the processor's average time per `lw`?

3. In this class, we've discussed computer architecture primarily from the perspective of a single-processor system: that is, we assume that each computer has exactly one processor. In today's world, most computers have multiple processors, each of which is capable of operating independently of the others. Each processor can concurrently run a different program. Often, these processors will share RAM, but will each have their own cache.

   Consider the following architecture, where the computer has four processors, each with their own L1 cache:



   Your task is to discuss the choice between *write-back* and *write-through* cache in such a multiprocessor architecture. Specifically address any potential problems or complications that may arise by using one or the other of these two write policies. Also discuss a possible strategy to mitigate said complications.

4. Consider a 3GHz processor that has 64-bit addresses. It has 512KB of L1 cache (excluding tags and valid bits) and 4MB of L2 cache (also excluding tags and valid bits). All blocks are 512 bytes. Both caches are direct-mapped. Assume that each memory cell is one byte.

   (a) What is the size of the the tag, in bits, for each of the caches?

   (b) What is the total actual size (in bits) of each of the caches, including block storage, valid bits, and tags?

   (c) We define a "cache reference" as an attempt to access a cache, which may result in either a hit or a miss.

   In this question, a "non-memory instruction" refers to an assembly language instruction that does not access memory, such as `addi` or `jal`; as opposed to a "memory instruction," such as `lw` or `sw`.

   Assume that all non-memory instructions and instructions that cause an L1 cache references execute in one cycle. L2 cache references carry a 10ns penalty, and main memory references carry an 100ns penalty. Assume that 10% of instructions result in an L1 reference; 5% of all instructions result in an L2 reference; and 3% of all instructions go to main memory.

   Calculate the average cycles per instruction (CPI).

5. A particular program executes a sequence of reads at the following memory byte addresses (in hex):

```
40 47 50 42 81 83 85 57 46
```

   We have a cache with a capacity of 32 bytes (not including metadata, such as tags and valid bits). For each of the following cache types, assume that the above sequence of memory accesses is made. For each memory access in each cache, show whether it is (a) a cache hit (H), (b) a cache miss resulting in an eviction (M), or (c) a cache miss not resulting in an eviction (MC). Then give the hit ratio (i.e. the number of hits per total number of memory accesses) for each cache.

   Each memory cell is one byte.

   All associative caches use an LRU replacement policy.

   (a) direct mapped cache, block size = 4 bytes

Addr	40h	47h	50h	42h	81h	83h	85h	57h	46h
H, M, or MC?									

   Hit ratio:

   (b) fully associative cache, block size = 4 bytes

Addr	40h	47h	50h	42h	81h	83h	85h	57h	46h
H, M, or MC?									

   Hit ratio:

   (c) direct mapped cache, block size = 8 bytes

Addr	40h	47h	50h	42h	81h	83h	85h	57h	46h
H, M, or MC?									

   Hit ratio:

   (d) two-way set associative cache, block size = 8 bytes

Addr	40h	47h	50h	42h	81h	83h	85h	57h	46h
H, M, or MC?									

   Hit ratio:

6. Consider a computer with two caches:

- An L1 cache. It is fully associative, stores 2 blocks in total, and has a block size of 4 cells. It uses an LRU replacement strategy.
- An L2 cache. It is direct-mapped, has two rows, and has a block size of 8.

Consider the following sequence of memory accesses. Complete the table showing the row and tag of each access in both caches, as well as whether it's a hit or a miss. If an access causes a hit in L1, leave the L2 columns empty. Note that there is no "L1 row" column because L1 is fully associative. The first row has been completed for you.

Addr	L1 tag	L1 h/m	L2 row	L2 tag	L2 h/m
53	13	m	0	3	m
50					
52					
40					
60					
52					