CS 382 | Computer Architecture
Fall 2023 | and Organization | Homework 4

Michael Buzzetta
I pledge my honor that I have abided by then stevens honor system
Requirement (READ FIRST!)

You have to **type** the solutions. **Handwritten homework will not be graded and will receive zero credit.** You can annotate on this document directly (you might need to know how to insert a picture into a PDF file); or you can submit a separate PDF, but with solutions clearly marked with question numbers.

1 (15 points)

Consider three different processors P1, P2, and P3 executing the same instruction set. P1 has a 3.2 GHz clock rate and a CPI of 1.5. P2 has a 2.0 GHz clock rate and a CPI of 1.0. P3 has a 4.0 GHz clock rate and has a CPI of 2.3.

1.1 (6 points)

Calculate the performance of each processor expressed in instructions per second.

- **P1**: 3.2 x 10^9 / 1.5 = 2.13 x 10^9 instructions per second
- **P2**: 2.0 x 10^9 / 1.0 = 2.0 x 10^9 instructions per second
- **P3**: 4.0 x 10^9 / 2.3 = 1.7 x 10^9 instructions per second

1.2 (6 points)

If each of these processors executes a program in 10 seconds, calculate the number of cycles and the number of instructions.

- **Instructions**: 2.13 x 10^9 * 10 = 2.13 x 10^10 instructions per 10 seconds
- **Cycles**: 3.2 x 10^9 * 10 = 3.2 x 10^10 cycles

- **Instructions**: 2.0 x 10^9 * 10 = 2.0 x 10^10 instructions per 10 seconds
- **Cycles**: 2.0 x 10^9 * 10 = 2.0 x 10^10 cycles

- **Instructions**: 1.7 x 10^9 * 10 = 1.7 x 10^10 instructions per 10 seconds
- **Cycles**: 4.0 x 10^9 * 10 = 4.0 x 10^10 cycles

1.3 (3 points)

We are trying to reduce the execution time of P2 by 30%, but this leads to an increase of 20% in the CPI. What should the clock rate be to obtain this reduction in execution time?

10 * 0.7 = 7 seconds

CPI = 1 * 1.2 = 1.2

New clock rate = $\frac{\boldsymbol{Old\ clock\ rate * (CPI)}}{\boldsymbol{Execution\ Time}}$

2.0 x 10^10 * 1.2 / 7 = **3.4E9 Hertz**

2 (10 points)

Consider two different implementations of the same instruction set architecture. The instructions can be divided into four classes according to their CPI (classes A, B, C, and D). P1 with a clock rate of 2.5 GHz and CPIs of 1, 2, 3, and 3 for the corresponding classes, and P2 with a clock rate of 3 GHz and CPIs of 2, 2, 2, and 2. Given a program with a dynamic instruction count of 1.0e6 instructions divided into classes as follows: 30% class A, 20% class B, 30% class C, and 20% class D:

2.1 (4 points)

What is the total CPI for each implementation?
Class A: 3 x 10^5
(1 x 10^6) * 0.3

Class B: 2 x 10^5
(1 x 10^6) * 0.2

Class C: 3 x 10^5
(1 x 10^6) * 0.3

Class D: 2 x 10^5
(1 x 10^6) * 0.2

P1: 1*(3 x 10^5) + 2*(2 x 10^5) + 3*(3 x 10^5) + 3*(2 x 10^5) = (3 x 10^5) + (4 x 10^5) + (9 x 10^5) + (6 x 10^5) = **2.2$_{E}$6 Hertz**

P2: 2*(3 x 10^5) + 2*(2 x 10^5) + 2*(3 x 10^5) + 2*(2 x 10^5) = (6 x 10^5) + (4 x 10^5) + (6 x 10^5) + (4 x 10^5) = **2.0$_{E}$6 Hertz**

P1: (2.2 x 10^6) / (1 x 10^6) = **2.2 cycles per instruction**
P2: (2.0 x 10^6) / (1 x 10^6) = **2.0 cycles per instruction**

2.2 (4 points)

Calculate the clock cycles required in both cases.

$\frac{CPI}{Clock\ rate}$ **= clock cycle time**

- P1: 2.2 x 10^6 / 2.5 x 10^9 = 8.8 x 10^-4 seconds
- P2: 2.0 x 10^6 / 3.0 x 10^9 = 6.7 x 10^-4 seconds

2.3 (2 points)

Which is faster: P1 or P2?

Using the values that we calculated in 2.2, we can see that P1 takes 8.8$_{E}$-4 seconds and P2 takes 6.7$_{E}$-4. These values show that P2 is faster than P1 taking about 2$_{E}$-4 seconds less

3 (15 points)

In this exercise we look at memory locality properties of matrix computation. The following code is written in C, where *elements within the same row of a matrix are stored contiguously*. You can assume that undefined symbols are primitive types, *i.e.*, int, double*etc*.

```c
for( int j = 1; j < n-1; j++) {
    for(int i = 1; i < m-1; i++) {
        Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] + A[j-1][i] + A[j+1][i]);
        err = max(err, fabs(Anew[j][i] - A[j][i]);
    }
}
```

3.1 (5 points)

Which variables exhibit temporal locality?

- i
- j
- m
- n
- Anew[j][i]
- err

3.2 (5 points)

Which variables exhibit spatial locality? Locality is affected by both the reference order and data layout.

- A[j][i-1]
- A[j-1][i]
- A[j][i+1]
- A[j+1][i]

3.3 (5 points)

Would the program be slower or faster if the outer loop iterated over i and the inner loop iterated over j? Why?

It would be slower due to the elements in each row being stored contiguously and the code populating the matrix by rows, then columns. If we were to flip the loops, the values would no longer be stored contiguously, therefore taking longer

4 (30 points)

Below is a list of 8-bit memory address references, given as **word** addresses. Answer the following questions based on **Section 4.2.3.1** of the textbook.

0x43, 0xc4, 0x2b, 0x42, 0xc5, 0x28, 0xbe,
0x05, 0x92, 0x2a, 0xba, 0xbd

4.1 (15 points)

For each of these references, identify the binary word address, the tag, and the index given a direct-mapped cache with 16 one-word blocks. Also list whether each reference is a hit or a miss, assuming the cache is initially empty.

- No offset because it's a one-word and word-addressed
    - log2(16) = 4 bits
        - Tag Index Hit/Miss: Check for prior similar tag and index
    - 0x43: 0100 0011 Miss
    - 0xc4: 1100 0100 Miss
    - 0x2b: 0010 1011 Miss
    - 0x42: 0100 0010 Miss
    - 0xc5: 1100 0101 Miss
    - 0x28: 0010 1000 Miss
    - 0xbe: 1011 1110 Miss
    - 0x05: 0000 0101 Miss
    - 0x92: 1001 0010 Miss
    - 0x2a: 0010 1010 Miss
    - 0xba: 1011 1010 Miss
    - 0xbd: 1011 1101 Miss

4 (30 points)

4.2 (15 points)

For each of these references, identify the binary word address, the tag, the index, and the offset given a direct-mapped cache with two-word blocks and a total size of eight blocks. Also list if each reference is a hit or a miss, assuming the cache is initially empty.

- log2(2) = 1. Offset size: 1 bit
- log2(8) = 3. Set Index size: 3 bit
- 8-3-1 = 4. Tag size: 4 bits

Tag Set Index Offset Hit/Miss

- 0x43: 0100 001 1 Miss
- 0xc4: 1100 010 0 Miss
- 0x2b: 0010 101 1 Miss
- 0x42: 0100 001 0 Hit
- 0xc5: 1100 010 1 Hit
- 0x28: 0010 100 0 Miss
- 0xbe: 1011 111 0 Miss
- 0x05: 0000 010 1 Miss
- 0x92: 1001 001 0 Miss
- 0x2a: 0010 101 0 Hit
- 0xba: 1011 101 0 Miss
- 0xbd: 1011 110 1 Miss

5 (20 points)

Consider a **byte addressing** architecture with 64-bit memory addresses.

5.1 (5 points)

Which bits of the address would be used in the tag, index and offset in a direct-mapped cache with 512 1-word blocks?

1 word: 4 bytes

Cache size: 2^n = 4

log2(4) = 2 bits -- Offset: 2 bits

log2(512) = 9 -- Set Index: 9 bits

64-9-2 = 53

Tag: 53 bits

Tag: 53 bits -- Set Index: 9 bits -- Offset: 2 bits

5.2 (5 points)

Which bits of the address would be used in the tag, index and offset in a direct-mapped cache with 64 8-word blocks?

8 word block: 32 bytes

log2(32) = 5 -- Offset: 5 bits

log2(64) = 6 -- Set Index: 6 bits

64-5-6 = 53 bits

Tag: 53 bits

Tag: 53 bits -- Set Index: 6 bits -- Offset: 5 bits

5.3 (5 points)

What is the ratio of bits used for storing data to total bits stored in the cache in each of the above cases?

32 bytes per (53+1+32)

256 bytes per (53+1+256)

Case1: 32/86

Case2: 256/310

5 (20 points)

5.4 (5 points)

Which bits of the address would be used in the tag, index and offset in a two-way set associative cache with 1-word blocks and a total capacity of 512 words.

1 word block = 4 bytes
log2(4) = 2 -- Offset: 2 bits
512/2 = 256
log2(256) = 8 -- Set index: 8 bits
64-2-8 = 54 bits
Tag: 54 bits
Tag: 54 bits -- Set Index: 8 bits -- Offset: 2 bits

6 (10 points)

Given a byte-addressed memory with 12-bit addresses, and an attached four-way set associate cache with a total of 16 one-word blocks, make a table showing the final state of the cache after accessing the following memory addresses. The table should include the tags and data that will be stored in cache at the end of all insertions. Data should be shown using the corresponding addresses in main memory. Use the least-recently-used rule to evict from cache as needed.

The sequence of addresses accessed is:

0x143, 0xc4a, 0x22b, 0x42f, 0x492, 0x2a2, 0x3ba, 0xb2d

16 blocks 4 way
16/4 = 4 blocks
log2(4) = 2 bits -- Offset: 2 bits
log2(4) = 2 bits -- Set Index: 2 bits
12-2-2 = 6 bits
Tag: 6 bits
Tag: 6 bits -- Set Index: 2 bits -- Offset: 2 bits

	Address	Tag	Set Index	Offset Hit/Miss
0x143:	000101000011	00010100	00 11	Miss
0xc4a:	110001001010	11000100	10 10	Miss
0x22b:	001000101011	00100010	10 11	Miss
0x42f:	010000101111	01000010	11 11	Miss
0x492:	010010010010	01001001	00 10	Miss
0x2a2:	001010100010	00101010	00 10	Miss
0x3ba:	001110111010	00111011	10 10	Miss
0xb2d:	101100101101	10110010	11 01	Miss

	00			01			10			11		
Set Index	V	Tag	Data	V	Tag	Data	V	Tag	Data	V	Tag	Data
00	0			0			0	0010 1010	M[0x2a2]	0	0001 0100	M[0x143]
01	0			0			0			0		
10	0			0			0	0011 1011	M[0x3ba]	0	0010 0010	M[0x22b]
11	1			1	1011 0010	M[0xb2d]	1			1	0100 0010	M[0x42f]

I evicted 0xc4a to 0x492 because they were in locations that is allocated to the new data. Following the the least-recently-used rule, these were evicted