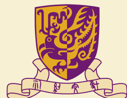


香港中文大學
The Chinese University of Hong Kong

CENG 3420: HW3 Review

Ziyi Wang
Department of Computer Science & Engineering
Chinese University of Hong Kong
ziyiwang21@cse.cuhk.edu.hk

April 5, 2022

Q1 Description

Assume a program requires the execution of 50×10^6 FP instructions, 110×10^6 INT instructions, 80×10^6 L/S instructions, and 16×10^6 branch instructions. The CPI for each type of instruction is 1, 1, 4, and 2, respectively. Assume that the processor has a 2 GHz clock rate.

1. By how much must we reduce the CPI of L/S instructions if we want the program to run two times faster?
2. By how much is the execution time of the program improved if the CPI of INT and FP instructions is reduced by 40% and the CPI of L/S and Branch is reduced by 30%?

$$CPU\ time = \frac{Instruction\ count \times CPI_{avg}}{clock\ rate}$$

Since Instruction count and clock rate **remain the same**, it's all about the averaged CPI:

$$CPI_{avg} = \sum_{i=1}^{n} Instruction_count_i \times CPI_i$$

Solution to Q1.1

$$Instruction\ count = 50 \times 10^6 + 110 \times 10^6 + 80 \times 10^6 + 16 \times 10^6 = 256 \times 10^6$$

'run two times faster' equals to 'reduce the CPU time by 50% ', which then equals to **'reduce the average CPI by 50% '**. The original average CPI can be calculated as :

$$CPI'_{avg} = 1 \times \frac{50 \times 10^6}{256 \times 10^6} + 1 \times \frac{110 \times 10^6}{256 \times 10^6} + 4 \times \frac{80 \times 10^6}{256 \times 10^6} + 2 \times \frac{16 \times 10^6}{256 \times 10^6}$$
$$= 2$$

By assuming the target CPI of L/S instruction as C, we can get the following equation:

$$CPI_{avg} = 1 \times \frac{50 \times 10^6}{256 \times 10^6} + 1 \times \frac{110 \times 10^6}{256 \times 10^6} + C \times \frac{80 \times 10^6}{256 \times 10^6} + 2 \times \frac{16 \times 10^6}{256 \times 10^6}$$
$$= 0.5 \times CPI^*_{avg} = 1$$

According to the above equation, we get $C = 0.8$, which means the CPI of L/S instructions should be reduced by $(1 - 0.8/4) \times 100\% = 80\%$.

Solution to Q1.2

$$CPI_{avg} = 0.6 \times \frac{50 \times 10^6}{256 \times 10^6} + 0.6 \times \frac{110 \times 10^6}{256 \times 10^6} + 2.8 \times \frac{80 \times 10^6}{256 \times 10^6} + 1.4 \times \frac{16 \times 10^6}{256 \times 10^6}$$
$$= 1.3375$$

Since the CPU time is proportional to the average CPI, then the execution time is improved by $1 - \frac{CPUtime}{CPUtime^*} = 1 - \frac{CPI_{avg}}{CPI^*_{avg}} = 1 - 1.3375/2 = 0.331 = 33.1\%$

Q2 Description

In this exercise, we examine how pipelining affects the clock cycle time of the processor. For simplicity, we limit our attention to 4 instruction classes: Load (lw), Store (sw), R-type, and Branch(beq). Problems in this exercise assume that individual stages of the datapath have the following latencies:

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

And the stages that each instruction class need to execute is listed as follows:

Instruction Class	IF	ID	EX	MEM	WB
Load (lw)	✓	✓	✓	✓	✓
Store (sw)	✓	✓	✓	✓	
R-type	✓	✓	✓		✓
Branch	✓	✓	✓		

1. What is the clock cycle time in a pipelined and non-pipelined processor?
2. What is the total latency of an lw instruction in a pipelined and non-pipelined processor? Assuming there is no data harzard.

Q2 Analysis

Thekey to solving this question is:

- In a non-pipelined processor, the clock cycle time must allow for **the slowest instruction**.

  clock cycle time equals to the excution time of the slowest instruction.

- In a pipelined processor, all the pipeline stages take a single clock cycle, so clock cycle must be long enough to accommodate **the slowest stage**.

  clock cycle time equals to the execution time of the slowest stage.

Solution to Q2

1. For the non-pipelined processor, we first list the total execution time for each instruction as follows:

Instruction Class	IF	ID	EX	MEM	WB	Total time
Load (lw)	250ps	350ps	150ps	300ps	200ps	1250ps
Store (sw)	250ps	350ps	150ps	300ps		1050ps
R-type	250ps	350ps	150ps		200ps	950ps
Branch	250ps	350ps	150ps			750ps

   For the **non-pipelined processor**, clock cycle time equals to the excution time of the slowest instruction –that is 'lw' –so the clock cycle time required is $1250ps$.

   For the **pipelined processor**, the clock cycle time equals to the excution time of the slowest stage– –that is 'ID' –so the clock cycle time required is $350ps$.

2. For the non-pipelined processor, the total latency of an lw instruction is $1250ps$ as shown in the above table. While for the pipelined processor, the total latency of an lw instruction is $350 \times 5 = 1750ps$.

Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a five-stage pipelined datapath:

```
add x15, x12, x11
lw x13, 4(x15)
or x13, x15, x13
```

1. If there is no forwarding, draw a pipeline diagram to show where to insert NOPs to ensure correct execution.
2. If there is forwarding, draw a pipeline diagram with forwarding to ensure correct execution. You can insert NOPs if necessary.

Solution to Q3.1

add x15, x12, x11

NOP

NOP

lw x13, 4(x15)

NOP

NOP

or x13, x15, x13

Solution to Q3.2

add x15, x12, x11 | IM | Reg | ALU | DM | Reg

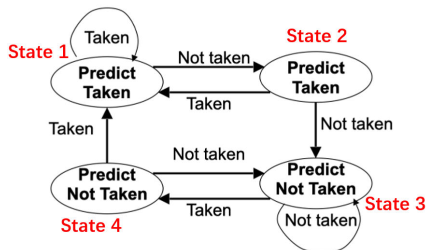
lw x13, 4(x15) | IM | Reg | ALU | DM | Reg

NOP

or x13, x15, x13 | IM | Reg | ALU | DM | Reg

Ex forwarding   Mem forwarding

Note that we still **need a NOP** here.



This exercise examines the accuracy of various **branch predictors** for the following repeating pattern (e.g., in a loop) of branch outcomes: T, NT, T, T, NT. (T means 'Taken' and NT means 'Not taken')

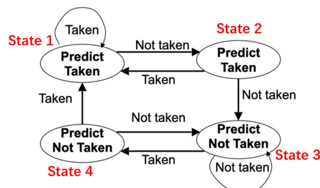
1. What is the accuracy of **always-taken** and **always-not-taken predictors** for this sequence of branch outcomes?
2. What is the accuracy of the 2-bit predictor if this pattern is repeated forever, assuming that the predictor starts off in the **top right state (State 2)** of the FSM on slide page 36, lec 12?

Q4 Analysis

- For the **always-taken predictor**, its accuracy equals to the probability that the branch outcome is Taken.
- For the **always-not-taken predictor**, its accuracy equals to the probability that the branch outcome is Not Taken.
- For the **2-bit predictor**, we should find out the repeating pattern of the prediction.

Solution to Q4

1. the accuracy of always-taken predictor is $3/5 * 100\% = 60\%$, and the accuracy of always-not-taken predictor is $2/5 * 100\% = 40\%$.



2. Let's first list the conditions for the first two iterations of repeating pattern as follows:

	Iteration1					Iteration2				
Branch output	T	NT	T	T	NT	T	NT	T	T	NT
State	2	1	2	1	1	2	1	2	1	1
Predict	T	T	T	T	T	T	T	T	T	T
Change state or not	yes	yes	yes	no	yes	yes	yes	yes	no	yes
Accurate or not	✓	×	✓	✓	×	✓	×	✓	✓	×

We can see that the condition of the first branch of iteration 2 is just the same as that of iteration 1, which means the following iterations keep the same as iteration 1. So we can use the result of iteration 1 to evaluate the accuracy of the 2-bit predictor, and that is $3/5 * 100\% = 60\%$.

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 are stored contiguously.** Assume each word is a 64-bit integer.

```
for (I=0; I<8; I++)
    for (J=0; J<8000; J++)
        A[I][J]=B[I][0]+A[J][I];
```

1. Which variable references exhibit temporal locality?
2. Which variable references exhibit spatial locality?

Q5 description

3. Locality is affected by both the reference order and data layout. The same computation can also be written below in Matlab, which differs from C in that **it stores matrix elements within the same column contiguously in memory.**

```
for I=1:8
    for J=1:8000
        A(I,J)=B(I,0)+A(J,I);
    end
end
```

   Which variable references exhibit temporal locality in the Matlab version?

4. Which variable references exhibit spatial locality in the Matlab version?

Q5 Analysis

- **Spatial Locality** means that all the data which is stored nearby to the recently visited data have high chances of usage.
  - It is relevant to how the matrix is stored in memory! (row-first / column-first)

- **Temporal Locality** means that data which is recently used have high chances of usage again.

Solution to Q5

1. B[I][0].

   For each I, B[I][0] is needed for the next 8000 iterations of J.

2. A[I][J].

   Since the elements within the same row are stored contiguously, then A[I][J+1] is next to A[I][J].

3. B[I][0].

   For each I, B[I][0] is needed for the next 8000 iterations of J.

4. A[J][I].

   Since the elements within the same column are stored contiguously, then A[J+1][I] is next to A[J][I].

How many total bits are required for a direct-mapped cache with 8 KiB of data and four-word blocks, assuming a 32-bit address?

The number of bits includes both the storage for data and for the tags

- For a direct mapped cache with 2^n blocks, n bits are used for the index
- For a block size of 2^m words (2^{m+2} bytes), m bits are used to address the word within the block, and 2 bits are used to address the byte within the word.

Size of the tag field: 32 - (n + m + 2)
Total number of bits in a direct-mapped cache:

$$2^n \times (\textit{block size} + \textit{tag field size} + \textit{valid field size}) \quad (1)$$

Solution to Q6

We know that 8 KiB is 2048 (2^{11}) words. With a block size of four words (2^2), there are 512 (2^9) blocks.
Each block has 4×32 or 128 bits of data plus a tag, which is 32-9-2-2 bits, plus a valid bit. Thus, the complete cache size is

$$2^9 \times (4 \times 32 + (32 - 9 - 2 - 2) + 1) = 74 \times 2^{10} = 74Kibits$$

THANK YOU!