ELEC3441: Computer Architecture
Second Semester, 2023–24

Homework 2 (r1.0)

Part A: Problem Set

A.1 Resolving Branch Hazards

A.1.1

Sequence	Instruction Address	Instruction	Branch Taken? (Y/N)
1	I10	beqz	Y
2	I14	bnez	Y
3	I10	beqz	Y
4	I14	bnez	Y
5	I10	beqz	Y
6	I14	bnez	Y
7	I10	beqz	Y
8	I14	bnez	Y
9	I10	beqz	Y
10	I14	bnez	Y
11	I10	beqz	N
12	I14	bnez	Y
13	I10	beqz	N
14	I14	bnez	Y
15	I10	beqz	Y
16	I14	bnez	Y
17	I10	beqz	Y
18	I14	bnez	Y
19	I10	beqz	N
20	I14	bnez	N

A.1.2 For `I10`: `I5`, and `I13` can be inserted to the delay slot, so 2 NOP is needed.
For `I14`: `I12` can be inserted to the delay slot, so 3 NOPs is needed.
Therefore,

$$\begin{aligned}\text{execution cycles} &= \text{executed instructions} + \text{NOP stall}\\ &= (2 + (12 \times 10 - 7) + 3) + (2 \times 10 + 3 \times 10)\\ &= 168\end{aligned}$$

A.1.3 16 branches are mispredicted. Each misprediction causes 4 cycles penalty.
Therefore,

$$\begin{aligned}\text{execution cycles} &= \text{executed instructions} + \text{misprediction penalty}\\ &= (2 + (12 \times 10 - 7) + 3) + 4 \times 16\\ &= 182\end{aligned}$$

A.1.4 5% of total 20 branch instructions will be mispredicted. Therefore 1 branches are mispredicted. Each misprediction causes 4 cycles penalty.
Therefore,

$$\begin{aligned}\text{execution cycles} &= \text{executed instructions} + \text{misprediction penalty}\\ &= (2 + (12 \times 10 - 7) + 3) + 1 \times 4\\ &= 122\end{aligned}$$

A.1.5

	s1	s2	s3
Number of cycle	168	182	122
cycle time	1 ns	1.5 ns	2 ns
Total time	168 ns	273 ns	244 ns

Therefore, s1 (using a delay slot) is the fastest for this particular code segment.

A.2 Cache Access

Consider the following sequence of memory accesses to the main memory in a 32-bit processor:

Address (hex)	Type
BA443E98	R
BA443FA8	W
54E99A88	R
54E99BB8	R
30013C80	R
54E99A84	R
54E99BA0	R
BA443FA4	W
54E99A90	W
BA443E88	R

A.2.1 Assume the following data cache organization:

- Capacity: 512 B
- Line size: 8 words
- Organization: direct map
- Policy: write back, write allocate

Below is the current status of your cache:

Cache contents:

index	tag	valid	dirty	data							
				7	6	5	4	3	2	1	0
0	-	0	0	-	-	-	-	-	-	-	-
1	-	0	0	-	-	-	-	-	-	-	-
2	-	0	0	-	-	-	-	-	-	-	-
3	-	0	0	-	-	-	-	-	-	-	-
4	5D221F	1	1	BEEF0007	BEEF0006	0EEE3441	0EEE3441	BEEF0003	BEEF0002	BEEF0001	0EEE3441
5	-	0	0	-	-	-	-	-	-	-	-
6	-	0	0	-	-	-	-	-	-	-	-
7	-	0	0	-	-	-	-	-	-	-	-
8	-	0	0	-	-	-	-	-	-	-	-
9	-	0	0	-	-	-	-	-	-	-	-
10	-	0	0	-	-	-	-	-	-	-	-
11	-	0	0	-	-	-	-	-	-	-	-
12	-	0	0	-	-	-	-	-	-	-	-
13	-	0	0	-	-	-	-	-	-	-	-
14	-	0	0	-	-	-	-	-	-	-	-
15	-	0	0	-	-	-	-	-	-	-	-

You can assume all data in your main memory contain the value `0x0EEE3441`. Trace through the above memory access and answer the following:

(i) For each access, is it a hit or a miss?
(ii) Reason for miss (e.g. Conflict miss due to access X; Compulsory miss, etc)
(iii) Show the final content in the cache, including the tag.

Offset size: $\log_2(\text{line\_size} \times 4) = 5$ bit
Index size: $\log_2(\text{cache\_capacity}/\text{line\_size}) = 4$ bit
Tag size: $32 - \text{offset\_size} - \text{index\_size} = 23$ bit
Cache access:

Address (hex)	Type	Index	Hit/Miss	Reason
BA443E98	R	0100	H	
BA443FA8	W	1101	M	Compulsory miss
54E99A88	R	0100	M	Conflict miss
54E99BB8	R	1101	M	Conflict miss
30013C80	R	0100	M	Conflict miss
54E99A84	R	0100	M	Conflict miss
54E99BA0	R	1101	H	
BA443FA4	W	1101	M	Conflict miss
54E99A90	W	0100	H	
BA443E88	R	0100	M	Conflict miss

Cache contents:

index (Hex)	tag	valid	dirty	data range
0	-	0	0	-
1	-	0	0	-
2	-	0	0	-
3	-	0	0	-
4	5D221F	1	0	BA443E9C-BA443E80
5	-	0	0	-
6	-	0	0	-
7	-	0	0	-
8	-	0	0	-
9	-	0	0	-
A	-	0	0	-
B	-	0	0	-
C	-	0	0	-
D	5D221F	1	1	BA443FBC-BA443FA0
E	-	0	0	-
F	-	0	0	-

A.2.2 Repeat A.2.1 but with a different cache organization:

- Organization: 2-way set associative
- Policy: true LRU, write back, write allocate

Cache access:

Address (hex)	Type	Index	Hit/Miss	Reason
BA443E98	R	100	M	Compulsory miss
BA443FA8	W	101	M	Compulsory miss
54E99A88	R	100	M	Conflict miss
54E99BB8	R	101	M	Compulsory miss
30013C80	R	100	M	Conflict miss
54E99A84	R	100	H	
54E99BA0	R	101	H	
BA443FA4	W	101	H	
54E99A90	W	100	H	
BA443E88	R	100	M	Conflict miss

Cache contents:

index (Hex)	tag	valid	dirty	data range	tag	valid	dirty	data range
0	-	0	0	-	-	0	0	-
1	-	0	0	-	-	0	0	-
2	-	0	0	-	-	0	0	-
3	-	0	0	-	-	0	0	-
4	BA443E	1	0	BA443E9C-BA443E80	54E99A	1	1	54E99A9C-54E99A80
5	BA443F	1	1	BA443FBC-BA443FA0	54E99B	1	0	54E99BBC-54E99BA0
6	-	0	0	-	-	0	0	-
7	-	0	0	-	-	0	0	-

A.2.3 Repeat A.2.1 but with a different cache:

- Organization: Fully Associative
- Policy: write through, no write allocate

Cache access:

Address (hex)	Type	Index	Hit/Miss	Reason
BA443E98	R	0	M	Compulsory miss
BA443FA8	W	0	M	Compulsory miss
54E99A88	R	0	M	Compulsory miss
54E99BB8	R	0	M	Compulsory miss
30013C80	R	0	M	Compulsory miss
54E99A84	R	0	H	
54E99BA0	R	0	H	
BA443FA4	W	0	H	
54E99A90	W	0	H	
BA443E88	R	0	H	

Cache contents:

location (Hex)	tag	valid	dirty	data range
0	5D221F4	1	0	BA443E9C-BA443E80
1	5D221FD	1	1	BA443FBC-BA443FA0
2	2A74CD4	1	1	54E99A9C-54E99A80
3	2A74CDD	1	0	54E99BBC-54E99BA0
4	18009E4	1	0	30013C9C-30013C80
⋮	-	0	0	-

A.2.4 Assume you are allowed to make the following modification to the cache:

Cache Parameter	Range
capacity	4 B to 512 B
line size	1 to 256 words
set associativity	1 (direct map) to 4
write miss policy	any valid policy

With the above sequence of memory access, design a cache that would result in the *minimum* number of misses (N_{miss}) using *minimum* capacity (C). That is, you want to minimize the quantity:

$$N_{\text{miss}} \times C$$

Explain your answer.

A.3 Simple Pipeline

A.3.1 Denote the array pointed by `a0`, `a1`, `a2` and `a3` as `A[]`, `B[]`, `C[]` and `D[]` respectively. Then the code performed is

```
sum_c = 0x00A00000;
sum_d = 0x00A10000;
for (i = 100; i > 0; i--) {
  C[i] = A[i] + B[i];
  D[i] = A[i] - B[i];
  sum_c += C[i];
  sum_d += D[i];
}
sum_c = 16 * sum_c;
sum_d = 16 * sum_d;
```

Since `A[i]` $= 100$, and `B[i]` $= 17$ for all `i`, then

$$\begin{aligned}\texttt{sum\_c} &= 16 \times (\texttt{0x00A00000} + 100 \times (100 + 17)) \\ &= 167959360 \\ \texttt{sum\_d} &= 16 \times (\texttt{0x00A10000} + 100 \times (100 - 17)) \\ &= 168953536\end{aligned}$$

A.3.2 $5 + 14 \times 100 + 8 = 1413$ instructions are executed.

A.3.3

```
lw    t2, 0(s1)    # I1: ID EX MA WB
add   t3, t1, t2   # I2: IF ID ID ID ID EX
```

I2 can only start ID stage after I1 finish WB stage.

A.3.4 The data hazard occurs 100 times and each stalls 3 cycles.
The branch misprediction occurs 99 times and each stalls 2 cycles.
Execution cycle in total:

$$1413 + 3 \times 100 + 2 \times 99 = 1911$$

Therefore,

$$\text{CPI} = \frac{1911}{1413} \approx 1.352$$

A.4 Streaming Cache Performance

You are investigating the cache performance of your processor regarding the following code segment:

```
// int i, N, a, b;
// int y[], x[];
for (i = 0; i < N; i++) {
  y[i] = a*x[i] + b*y[i];
}
```

Now the above C code is compiled into the following RISC-V instructions:

```
# a0 initially contains the constant N
# a1 is base address of x[]
# a2 is base address of y[]
# constant a is stored in register a3
# constant b is stored in register a4
00| loop: lw    t1, 0(a1)
04|       lw    t2, 0(a2)
08|       mult  t0, t1, a3
0C|       mult  t1, t2, a4
10|       add   t0, t0, t1
14|       sw    t0, 0(a2)
18|       addi  a1, a1, 4
1C|       addi  a2, a2, 4
20|       addi  a0 a0 -1
24|       bne   a0, zero, loop
```

A.4.1 Cache Hit or Miss Assume array `x` is stored at memory address starting at `0xEA000000` while the array `y` is stored at memory address `0xEB000000`.

The processor data cache is initially empty and has the following properties:

- Capacity: 1 MiB
- Organization: 2-way set associative
- Line size: 8 words
- Replacement policy: true LRU
- Write policy: write through, no write allocate

A large write buffer is available such that the processor can resume running immediately after writing the data to the buffer.

Let $N = 2^{10}$, trace through the above code, then show and explain the sequence of cache hit or miss that will occur. Use the notation **RH** for read hit, **WH** for write hit, **RM** for read miss, and **WM** for write miss.

Each iteration has three access: read `x[i]`, read `y[i]`, write `y[i]`. In first iteration (i=0), read `x[0]` generates a compulsory miss (RM), read `y[0]` generates a compulsory miss (RM), write `y[0]` is a hit (WH). In the given cache organization, the line of `x[0]` and `y[0]` have same index but different tag. Since the cache is 2-way s.a., the line of `x[i]` and `y[i]` are not going to conflict with each other. Therefor, subsequent 7 iterations access the same line and they are all hit. Later iterations repeat the same pattern.

The overall sequence is:

RM RM WH, then repeat **RH RH WH** 7 times, then repeat all of them 2^7 times.

A.4.2 Based on your result above, what is the overall data cache miss rate of the above code when executed in the main CPU? Consider BOTH read and write access. Assume the write and read miss penalties are both 300 cycles. Hit time is 1 cycle. What is the average memory access time (AMAT) for this code?

Overall miss rate is $2/24 = 1/12$. Therefore, AMAT is

$$1 + \frac{1}{12} \times 300 = 26\text{cycles}$$

A.4.3 For simplicity, assume there is no pipeline in the processor. Furthermore, assume `add`, `addi` and `bne` takes 1 cycle; `mult` takes 4 cycles; and performance of `lw` and `sw` depends on the cache performance. What is the total run time of the above code in cycles? You may leave the variable N in your answer.

$N \times (1 \times 5 + 4 \times 2 + 26 \times 3) = 91N$

A.4.4 When $N = 2^{23}$, array `x` need 32 MiB memory which is larger than the cache capacity. Previous cached line might be evicted in later iterations. This is not happened when $N = 2^{10}$. However, evicted line will not be accessed later in this program. The miss rate will not change.

A.4.5 Write Back Cache If the write policy of the cache is changed to *write back, with write allocate*, how would the following be affected? Consider 2 cases: (i) with write buffer, (ii) without write buffer.

(i) Read miss penalty
(ii) Write miss penalty
(iii) Overall performance of the above code

Explain your answer by tracing through the execution of the above code, highlighting any different in the required cache content handling with the use of write back cache.

Since the entire array stays within the capacity of the cache, no conflict miss occurs. Therefore, the write policy does not change the overall performance of the given code.

In particular, changing write policy does not affect read miss penalty.

With a write buffer, assuming it is not full, then the write miss penalty would be zero because nothing need to affect the cache. Without write buffer, however, means that the processor must stall until the write completes. Therefore, write miss penalty increases.