UNIVERSITY of HOUSTON

Computer Organization & Architecture
College of Natural Sciences and Mathematics
Fall 2020

Exam 3

This exam is given under the University of Houston Honor Code System. You must observe and sign the Honor Pledge: "I have neither given nor received aid on this exam." Your print name and signature below signifies your compliance with this honor code. Note that we will not grade your exam unless it is acknowledged.

Student ID (last four digits)

Name:

By clicking the checkbox below, I acknowledge my responsibility and commitment to the Academic Honor Code.

☐ Acknowledgment

1. ______________ (16 pts)
2. ______________ (36 pts)
3. ______________ (12 pts)
4. ______________ (36 pts)

Total (100 pts) _____________

Make sure you use a software that allows filling PDF forms.

- *Foxit (Web, Android, iOS, Windows, Mac) to edit PDFs everywhere*
- *PDF Expert (iOS, Mac) to quickly edit PDF text and images*
- *PDFelement (Android, iOS,Windows, Mac) to edit PDFs*
- *Adobe Acrobat (Windows, Mac) to create detailed PDFs and forms*

Make sure you save your answers (doublecheck before submit). In addition, doublecheck that you have submitted your filled pdf.

If you see dots, it means that you are using a pdf software or a setting that has missing font. You may resolve this by switching to a different pdf software (such what mentioned above) or write down your solution in a separate file.

```
cd lab0-your-github-account-name/exams/
git add "*"
git commit -m "Exam3"
git push origin master
```

1. Multi-core Performance. **(16 pts)** Designers are comparing performance between a many-core processor equipped with 64 **in-order cores** versus a **quad-core** superscalar processor. The quad-core superscalar processor has four identical Out-of-Order cores. Each core is a 4-issue superscalar. The many-core processor has 64 in-order processor cores. Each in-order core of the many-core processor can achieve about **10%** the performance of the OoO superscalar core. A benchmark program K includes both sequential codes that can only be executed on one CPU core and parallel codes that can be distributed to multiple CPU cores. For program K, execution of the sequential codes takes **25%** of the total execution time. Please answer which processor would deliver faster performance for benchmark program K and show your calculations (use an in-order core as performance baseline).

Multi-core OOO machine speedup over baseline

In-order multi-core machine speedup over baseline

Which is faster?

2. Set Associative Cache (36 pts) Given the following address access stream, please answer 2.1, 2.2 and 2.3. All the addresses are 32-bit. The sequence is shown below.

Load	**0x12650144**
Store	**0x22160484**
Load	**0x1265014C**
Store	**0x09000D40**
Load	**0x22160488**

2.1 (11 pts) A **64 bytes**, **2-way writeback cache**. The cache line size is **8 bytes**. Please calculate the **number of bits used for tag, set index, and offset**.

Number of tag bits =

Number of index bits =

Number of offset bits =

2.2 (15 pts) Fill the correct tag, index, and offset values for each memory access.

	Access	Address	Tag (hex) – skip 0x	Index (binary)	Offset (binary)
1	**Load**	**0x12650144**			
2	**Store**	**0x22160484**			
3	**Load**	**0x1265014C**			
4	**Store**	**0x09000D40**			
5	**Load**	**0x22160488**			

2.3 (10 pts) Fill the correct values for V, D and Tag based on the access stream. Use **Hex** values for tags.
After completing (Assume you always fill way0 first. If case replacement is needed, use LRU)

1	**Load**	**0x12650144**

	Way 0			Way 1		
	V	D	TAG	V	D	TAG
0	☐	☐		☐	☐	
1	☐	☐		☐	☐	
2	☐	☐		☐	☐	
3	☐	☐		☐	☐	

After completing,

2	**Store**	**0x22160484**

	Way 0			Way 1		
	V	D	TAG	V	D	TAG
0	☐	☐		☐	☐	
1	☐	☐		☐	☐	
2	☐	☐		☐	☐	
3	☐	☐		☐	☐	

After completing,

3	**Load**	**0x1265014C**

	Way 0			Way 1		
	V	D	TAG	V	D	TAG
0	☐	☐		☐	☐	
1	☐	☐		☐	☐	
2	☐	☐		☐	☐	
3	☐	☐		☐	☐	

After completing,

4	**Store**	**0x09000D40**

	Way 0			Way 1		
	V	D	TAG	V	D	TAG
0	☐	☐		☐	☐	
1	☐	☐		☐	☐	
2	☐	☐		☐	☐	
3	☐	☐		☐	☐	

After completing,

5	**Load**	**0x22160488**

	Way 0			Way 1		
	V	D	TAG	V	D	TAG
0	☐	☐		☐	☐	
1	☐	☐		☐	☐	
2	☐	☐		☐	☐	
3	☐	☐		☐	☐	

3. Memory System (12 points). All of the questions pertaining to memory systems, unless stated otherwise, will use the following memory hierarchy. We have a processor with 32-bit words, 1 GByte of main memory.

We have split, instruction and data L1 caches, each a direct mapped 32 **KByte cache with 32 byte lines**. The L2 cache is a 4 **MByte, 8-way set associative cache with 256 byte lines**.

When there is a L2 miss, the cache controller will fetch 256 bytes from the main memory, which will take 150 clock cycles.

Table 3-1: L1 and L2 details

	Cache Structure	**Hit Latency**	**Miss Rate**
L1 Data Cache	**32KB, direct mapped, 32B cache line**	**1**	**25%**
L1 Instruction Cache	**32KB, direct mapped, 32B cache line**	**1**	**15%**
L2 Cache	**4MB, 4 way set associative, 256B line per set**	**8**	**8% for data** **5% for instruction**

Based on Table 3-1, what is the **average data cache latency**? (Show how you calculate. You don't have to compute the final number.)

Average Data Cache Latency =

4. Gshare Branch Prediction. Designers of a new computer decided to implement a Gshare branch prediction scheme shown in the drawing below.

Figure 4-1: Gshare Branch Predictor

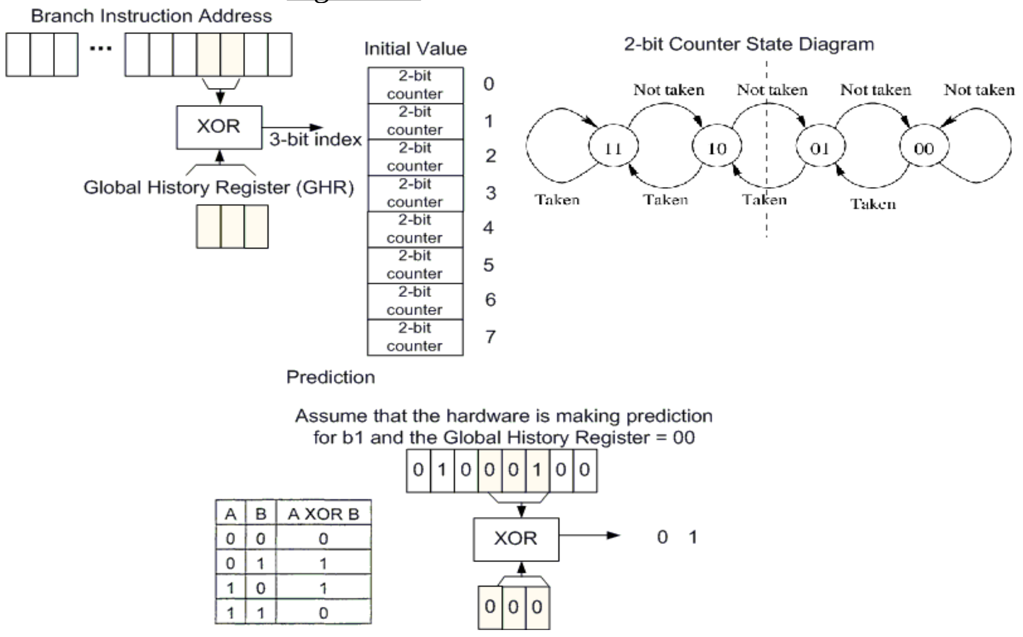
Branch Instruction Address
XOR
3-bit index
Global History Register (GHR)
Initial Value
2-bit counter
0
1
2
3
4
5
6
7
Prediction
2-bit Counter State Diagram
Not taken
Taken
11
10
01
00
Assume that the hardware is making prediction for b1 and the Global History Register = 00
0 1 0 0 0 1 0 0
XOR
0 1
0 0 0
A
B
A XOR B
0 0 0
0 1 1
1 0 1
1 1 0

Figure 4-1 shows a table of eight 2-bit counters. The table uses 3-bit index, generated by combining 3 bits from the branch instruction address and 3 bits from a global history register using XOR operation. Please refer to the state diagram of 2-bit counter branch prediction. The table below records branch instructions and outcomes when a program is executed, It also tracks the value of GHR, status of the eight 2-bit counters, and the branch prediction result. GHR is computed based on the previous three branch outcomes.

Please complete the table below using Gshare branch predictor as shown in Figure 4-1. Pay attention to the 2-bit counter state diagram. **Note that all addresses given are word addresses (not byte address).**

Table 4-1: Gshare Predictors

	Branch Instruction Address (hex) – **word addr**	Branch Actual Outcome	GHR	3-bit Index	8 2-bit Counters (select based on the 3-bit index) **000 (0)**	**001 (1)**	**010 (2)**	**011 (3)**	**100 (4)**	**101 (5)**	**110 (6)**	**111 (7)**	Branch Prediction
1	c06bc	**0 (NT)**	**001**	**101**	**2**	**1**	**2**	**0**	**2**	**1**	**0**	**3**	**NT**
2	c06be	**1 (T)**											☐ T ☐ NT
3	c149a	**1 (T)**											☐ T ☐ NT
4	c149f	**0 (NT)**											☐ T ☐ NT
5	c14a7	**1 (T)**											☐ T ☐ NT
6	c14ab	**0 (NT)**											☐ T ☐ NT