CMU 18-447 Introduction to Computer Architecture, Spring 2015

Midterm Exam 2

Date: Fri., 4/24

Instructor: Onur Mutlu
TAs: Rachata Ausavarungnirun, Kevin Chang, Albert Cho, Jeremie Kim, Clement Loh

Name:

Problem 1 (30 Points):	
Problem 2 (40 Points):	
Problem 3 (40 Points):	
Problem 4 (40 Points):	
Problem 5 (40 Points):	
Problem 6 (50 Points):	
Problem 7 (80 Points):	
Total (320 Points):	

Instructions:

1. This is a closed book exam. You are allowed to have two letter-sized cheat sheets.
2. No electronic devices may be used.
3. This exam lasts 1 hour and 50 minutes.
4. Clearly indicate your final answer for each problem.
5. Please show your work when needed.
6. Please write your initials at the top of every page.
7. Please make sure that your answers to all questions (and all supporting work that is required) are contained in the space required.

Tips:

- **Be cognizant of time.** Do not spend too much time on one question.
- **Be concise.** You will be penalized for verbosity and unnecessarily long answers.
- **Show work when needed.** You will receive partial credit at the instructors' discretion.
- **Write legibly.** Show your final answer.

1. Potpourri [30 points]

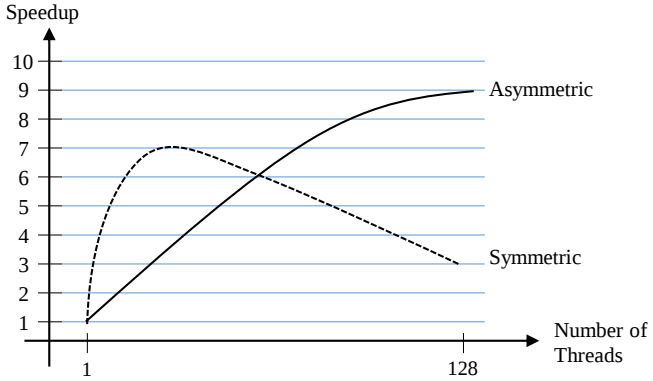
(a) What is the advantage of using a **virtually-indexed physically-tagged (VIPT)** L1 cache? Be precise and concise.

To mitigate the TLB lookup latency by overlapping it with the accesses to the cache.

(b) Typically, for a realistic workload, the parallel fraction of a program is *not* perfectly parallelizable. What are the three fundamental reasons why?

1. Synchronization
2. Load imbalance
3. Resource contention

(c) The figure below shows the speedup curve for a workload on two systems: symmetric multicore and asymmetric multicore. Assume an area budget of 32 small cores.



What is the performance improvement of the asymmetric multicore over the symmetric one? Show your work.

9/7 = 1.29 You need to compare the two systems at their best number of threads to make a fair comparison.

(d) In a MESI cache coherence protocol that transitions from **Shared** to **Exclusive** state on a *processor write*, what does the cache controller need to perform to correctly complete the transition? Be precise.

1. Invaldiate other cache lines. 2. Write back dirty data to memory.

2. Memory Scheduling [40 points]

We have a byte-addressable processor that is connected to a single memory channel that has a single rank of DRAM. The physical address space is 32 bits, and the processor uses the following mapping shown in Table 1 to index the DRAM. Each DRAM row has a certain number of columns, where a column has the same size of a cache line. The processor uses 64-byte cache lines.

MSB			LSB
Rows	Banks	Columns	Cache Line Offset

Table 1: Mapping from the physical address to DRAM.

(a) Table 2 shows the snapshot of the memory request queue that has 6 pending memory requests. Assume all the rows are closed when the snapshot was taken. With the FR-FCFS scheduling policy, issuing these 6 requests results in a row buffer hit rate of 1/3 (i.e., 1/3 of the requests retrieve data from the row buffers directly).

Request	Physical Address
A (oldest)	0x0000_0000
B	0x0000_1000
C	0x0000_4000
D	0x0000_0040
E	0x0000_0800
F (youngest)	0x0010_0040

Table 2: The state of the memory request queue.

What is the row size in KB? Show your work.

Since the hit rate is 1/3, there are two row hit requests. With the amount of information we have now, we know that there are 6 bits for the cache line offset. So requests A and D must go to the same row. A accesses row0, bank0, and col0. D accesses the same row and bank, but it goes to col1. Since A results in a miss and D is a hit, we need to find another hit request. The best candidate is request E.
By observing the bit-pattern difference between B and E, one can find out that the column bits end at the 12th bit, thus there are 6 bits for the columns as there are 6 bits for the cache line offset. Therefore, the row size is $2^{12} = 4KB$.
Summary: B goes to bank 1, C goes to bank 0 (row 1), and F goes to bank 0 (row 64). Therefore, B, C, and F are row misses.

(b) Table 3 shows the memory request queue that has 4 pending memory requests at time 0. Assume the following:

- A row buffer hit takes **50 cycles**.
- A row buffer conflict takes **250 cycles**.
- Requests going to different banks can be processed by the banks in parallel.
- All the row buffers are closed at time 0.
- The controller cannot issue two requests at the same time. Each request takes **10** cycles to process, so it takes 10 cycles between issuing two separate requests to the memory.
- The controller employs FR-FCFS scheduling policy.

Request	Physical Address
A (oldest)	0x0000_4000
B	0x0000_1040
C	0x0000_3040
D (youngest)	0x0000_4a00

Table 3: The state of the memory request queue at time 0.

If it takes **320 cycles** to finish processing all four requests in the memory, **at least** how many banks does this rank of DRAM have? Show your work.

320 = 10 + 250 (miss) + 10 + 50 (hit) So there must be one bank serving one miss and one hit, which is on the critical path. In parallel, there are two other banks serving requests.
With the information from part a, we know that A and D go to the same bank and same row. Therefore, B and C must go to two different banks. If they were to the same bank, that would incur 510 cycles of latency. By comparing the bit patterns between A and C, one can find that there are two bits for the banks. Thus, there are at least four banks in the rank.

3. Cache Coherence [40 points]

We have a system with four byte-addressable processors (P0, P1, P2, and P3). Each processor has a private 256-byte, direct-mapped, write-back L1 cache with a block size of 64 bytes. Coherence is maintained using the MESI protocol we discussed in class. The protocol transitions from **Shared (S)** to **Modified (M)** on a processor write. Accessible memory addresses range from 0x10000000 to 0x1FFFFFFF.

```
Executed Memory instructions:
ld R1, 0x110000c0 \\ A memory instruction from P1
st R2, 0x11000080 \\ A memory instruction from P1
st R3, 0x1FFFFF40 \\ A memory instruction from P0
ld R4, 0x1FFFFF00 \\ A memory instruction from P1
ld R5, 0x1FFFFF40 \\ A memory instruction from P2
```

After executing the above sequence of 5 memory instructions, we find the *final tag* store state of the four caches to be as follows:

Final State

Cache 0		
	Tag	*MESI State*
Set 0	0x1FFFFF	S
Set 1	0x1FFFFF	S
Set 2	0x110000	I
Set 3	0x110000	I

Cache 1		
	Tag	*MESI State*
Set 0	0x1FFFFF	S
Set 1	0x1FFFFF	I
Set 2	0x110000	M
Set 3	0x110000	E

Cache 2		
	Tag	*MESI State*
Set 0	0x10FFFF	I
Set 1	0x1FFFFF	S
Set 2	0x10FFFF	M
Set 3	0x10FFFF	M

Cache 3		
	Tag	*MESI State*
Set 0	0x133333	E
Set 1	0x000000	I
Set 2	0x000000	I
Set 3	0x10FFFF	I

Fill in the following table with the *initial* tag store states (i.e., *Tag* and *MESI state*) before executing the five memory instructions shown above. Answer X if the tag value of the address is unknown. For the MESI states, put in all the possible values (i.e., M, E, S, and I).

Initial State

Cache 0		
	Tag	*MESI State*
Set 0	0x1FFFFF	M,E,S
Set 1	X	M,E,S,I
Set 2	0x110000	M,E,S,I
Set 3	0x110000	I

Cache 1		
	Tag	*MESI State*
Set 0	X	M,E,S,I
Set 1	0x1FFFFF	M,E,S,I
Set 2	X	M,E,S,I
Set 3	X	M,E,S,I

Cache 2		
	Tag	*MESI State*
Set 0	0x10FFFF	I
Set 1	X	M,E,S,I
Set 2	0x10FFFF	M
Set 3	0x10FFFF	M

Cache 3		
	Tag	*MESI State*
Set 0	0x133333	E
Set 1	0x000000	I
Set 2	0x000000	I
Set 3	0x10FFFF	I

Scratchpad for the cache coherence problem

4. Amdahl's Law [40 points]

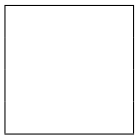
Consider Processor X with area A. You will analyze the performance of different processors with respect to Processor X. All processors discussed in this problem will be built on a die of area **16A**. Assume that the single-thread performance of a core increases with the **square root of its area**. You are given a workload where S fraction of its work is serial and $P = 1 - S$ fraction of its work is **perfectly** parallelizable.

In this problem, we define, for a given program:

Speedup of $Processor A = \frac{Runtime\ on\ Processor X}{Runtime\ on\ Processor A}$

(a) Given Processor Z is made of a single core with area $16A$, what is the speedup of this processor when running a program with $S = 1$? Show your work.

Processor Z
1 *large* core of area $16A$



4
One core with 4 times the speedup of Processor X runs the whole program.

(b) Given the same Processor Z from part (a), what is the speed up of this processor when running a program with $S = 0$? Show your work.

4
One core with 4 times the speedup of Processor X runs the whole program.

(c) What is the speedup equation for a heterogeneous processor that has one large core with area $N \times A$ and the remaining die area filled with small cores, each with area A? Write in terms of S and N.
Assume that (i) only the large core will be running the serial portion, and (ii) all cores, including the large core, will be running the parallel portion of the program.

$$\frac{1}{\frac{S}{\sqrt{N}} + \frac{1-S}{(16-N)+\sqrt{N}}}$$

(d) Now, you are given a chance to design a heterogeneous processor of area $16A$ to run some workload. The large core on the processor has an area size of $M^2 \times A$, where M is a natural number. What is M such that the processor can maximize the speedup when running a program with $P = 0.8$? Show your work.

M = 3.
We have 1,2,3,4 as options for M. When plugged into the equation from the previous part, M=3 gives the best speedup.

5. Prefetching [40 points]

A processor is observed to have the following access pattern to cache blocks. Note that the addresses are **cache block addresses**, not byte addresses. This pattern is repeated for a large number of iterations.

Access Pattern P: A, A + 3, A + 6, A, A + 5

Each cache block is 8KB. The hardware has a fully associative cache with LRU replacement policy and a total size of 24KB.

None of the prefetchers mentioned in this problem employ confidence bits, but they all start out with empty tables at the beginning of the access stream shown above. Unless otherwise stated, assume that 1) each access is separated long enough in time such that all prefetches issued can complete before the next access happens, and 2) the prefetchers have large enough resources to detect and store access patterns.

(a) You have a stream prefetcher (i.e., a next-N-block prefetcher), but you don't know the prefetch degree (N) of it. However, you have a magical tool that displays the coverage and accuracy of the prefetcher. When you run a large number of repetitions of access pattern P, you get 40% coverage and 10% accuracy. What is the degree of this prefetcher (how many next blocks does it prefetch)?

Next 4 blocks.
40% coverage with a stream prefetcher for this pattern means blocks A+3 and A+6 are prefetched. Possible N at this point are 3 and 4. Accuracy 10% = 2/(N*5), so N is 4.

(b) You didn't like the performance of the stream prefetcher, so you switched to a PC-based stride prefetcher that issues prefetch requests based on the stride detected for each memory instruction. Assume all memory accesses are incurred by the *same* load instruction (i.e., the same PC value) and the initial stride value for the prefetcher is set to 0.

Circle which of the cache block addresses are prefetched by this prefetcher:

A, A + 3, $\boxed{A+6}$, A, A + 5

A, A + 3, $\boxed{A+6}$, A, A + 5

A, A + 3, $\boxed{A+6}$, A, A + 5

A, A + 3, $\boxed{A+6}$, A, A + 5

Explain:

This prefetcher remembers the last stride and applies that to prefetch the next block from the current access.

(c) Stride prefetcher couldn't satisfy you either. You changed to a Markov prefetcher with a correlation table of 12 entries (assume each entry can store a single address to prefetch, and remembers the most recent correlation). When all the entries are filled, the prefetcher replaces the entry that is least-recently accessed.

Circle which of the cache block addresses are prefetched by this prefetcher:

A, A + 3, A + 6, A, A + 5

A, A + 3, $\boxed{A+6}$, $\boxed{A}$, A + 5

$\boxed{A}$, A + 3, $\boxed{A+6}$, $\boxed{A}$, A + 5

$\boxed{A}$, A + 3, $\boxed{A+6}$, $\boxed{A}$, A + 5

Explain:

All entries are filled after the first repetition, except the entry for A+5. Accesses to A+3 and A+5 thrash each other for block A's next-block entry, so they cannot be correctly prefetched.

(d) Just in terms of coverage, after how many repetitions of access pattern P does the Markov prefetcher from part (c) start to outperform the stream prefetcher from part (a), if it can at all? Show your work.

5 repetitions.
Coverage for Markov prefetcher from part (c) is (0+2+3*(N-2))/(5N). This is equal to 40% when N=4, so it outperforms 40% at the 5th repetition. We gave full credit if you wrote either 4 or 5, depending on the work shown.

(e) You think having a correlation table of 12 entries makes the hardware too costly, and want to reduce the number of correlation table entries for the Markov prefetcher. What is the minimum number of entries that gives the same prefetcher performance as 12 entries? Similar to the last part, assume each entry can store a single next address to prefetch, and remembers the most recent correlation. Show your work.

4 entries.
With 4 different accesses, we need at least 4 entries.

(f) Your friend is running the same program on a different machine that has a Markov prefetcher with 2 entries. The same assumptions from part (e) apply.

Circle which of the cache block addresses are prefetched by the prefetcher:

A, A + 3, A + 6, A, A + 5

A, A + 3, A + 6, A, A + 5

A, A + 3, A + 6, A, A + 5

A, A + 3, A + 6, A, A + 5

Explain:

NONE. Not enough entries. Entries will be evicted before they are used again.

(g) As an avid computer architect, you decide to update the processor by increasing the cache size to 32KB with the same cache block size. Assume you will be only running a program with the same access pattern P for a large number of iterations (i.e., one trillion), describe a prefetcher that provides smaller memory bandwidth consumption than the baseline without a prefetcher. Explain:

No prefetcher. Since all lines will sit in the cache after the first iteration, the prefetcher needs to achieve 100% accuracy on the first iteration in order to consume the same amount of memory bandwidth as the baseline, which is not possible without any training on the prefetcher beforehand.
Note: We leave it up to you to think whether or not you can ever design a prefetcher that leads to less memory consumption than the baseline without a prefetcher.

6. DRAM Refresh [50 points]

You are given a memory system that has four channels, and each channel has two ranks of DRAM chips. A separate memory controller controls each memory channel. Each rank of DRAM contains eight banks. A bank contains **R** rows. Each row in one bank is 8KB. The minimum retention time among all DRAM rows in the system is 64 ms. In order to ensure that no data is lost, every DRAM row is refreshed once per 64 ms. Refresh of each row is initiated by a command from the memory controller. The command refreshes only the specified row. The command occupies the command bus on the associated memory channel for 5 ns and the associated bank for 40 ns.

We define refresh utilization of a resource (such as a bus or a memory bank) as the fraction of total time for which a resource is occupied by a refresh command.

For each calculation in this section, you may leave your answer in simplified form in terms of powers of 2 and powers of 10. Show all your work for each section for full credit.

(a) Given a per-bank utilization of 2.048% caused by DRAM refreshes, what is the number of rows (**R**) in each bank? Show your work.

$\frac{R*40ns}{64ms} = \frac{2.048}{100}$
R = 32,768

(b) Using the **R** found in part (a), determine the command bus utilization. Show your work.

$\frac{2*8*32K*5ns}{64ms} = \frac{x}{100}$
x = 4.096%

(c) Determine the size of this memory system using the value of **R** found in part (a).

size = 32K * 4 * 8 * 2 * 8KB
size = 16 GB

(d) Only changing the number of rows per bank, find the **maximum number of rows per bank** for which either the bank utilization or the command bus utilization reaches 100%.

Because the command bus utilization will reach 100% before the bank utilization, we will look at how changing the number of rows will affect the command bus utilization
$\frac{R*5ns*2ranks*8banks}{64ms} = 1$
R = 800,000
We find that with 800k rows per bank, the command bus utilization reaches 100%.

(e) How can you reduce the command bus utilization due to refreshes? You cannot change the refresh rate when answering this question.

Have each command be responsible for multiple row refreshes.

7. Virtual Memory [80 points]

We have added a 4-set, 2-way write-through physically-indexed physically-tagged cache and an x86 style 2-level virtual-to-physical translation scheme to the LC-3b. Each virtual address is 16 bits. VA[15:11] is used to index the page directory, VA[10:6] is used to access the level 2 page table, and VA[5:0] is used to access the byte in the page. The PDE/PTE (page directory entry/page table entry) only contains a valid bit as its most significant bit and the PFN (page frame number) as its least significant bits. All remaining bits in the PDE/PTE are reserved and always set to zeros.

Assume the following:

- Physical address is 10 bits
- The cache implements a perfect LRU replacement policy
- The cache block size is 4 bytes
- Instructions and data share the same cache
- The cache is initially empty
- The page directory or a page table is one page in size
- Memory accesses to the page directory, page tables, and pages are **all** cached
- No page faults happen during the execution of an instruction
- Assume that the LC-3b does **not** permit unaligned accesses
- If the LRU bit is 0, the data in Way 0 will be evicted next
- For decoding instructions and page table/directory entries, the byte at the lower address is at bits [15:8] and the byte at the higher address is at bits [7:0]. (i.e., we assume **big endian**)

The PC is loaded with address x3184 and one instruction is run to completion. The figure below shows the contents of the cache **after** the execution of this instruction is complete. For each cache block, we index the data from left to right (e.g., in $Data_0$ of Set_1, if byte a references $0x60$, byte $a+1$ references $0x43$).

Use the figure below to answer the questions on the following three pages. There is an LC-3b ISA cheatsheet for your benefit in the appendix.

	LRU	V_0	Tag_0	V_1	Tag_1	$Data_0$	$Data_1$
Set_0	0	0	101100	0	010010	x80 x70 xAE xFD	x80 x06 x80 x77
Set_1	1	1	011100	0	011100	x60 x43 x59 xAE	xFD xE9 x46 x57
Set_2	0	0	011000	0	110110	x53 x74 x65 x70	x68 x65 x6E x21
Set_3	0	1	111100	1	110000	x80 x0C x00 x0F	x80 x07 x80 x06

Initials:

(a) How big is each page (in bytes)? Show your work.

2^6

(b) What is the value in the Page Directory Base Register? Show your work.

x3C0

(c) Fill in the table in the order of physical memory accesses caused by the instruction. Note that some rows in this table may be left blank.

Physical Address	**Data**	**Explanation of Data**
x3CC	x800C	PDE of instruction
x30C	x8007	PTE of instruction
x1C4	x6043	Instruction
x3CC	x800C	PDE of Data
x30C	x8007	PTE of Data
x1C4 or x1C6	x6043 or x59AE	Data

Show your work in the box below.

	LRU	V_0	Tag_0	V_1	Tag_1	$Data_0$	$Data_1$
Set_0	0	~~0~~	~~101100~~	~~0~~	~~010010~~	~~x80 x76 xAE xFD~~	~~x80 x06 x80 x77~~
Set_1	1	1	011100	~~0~~	~~011100~~	x60 x43 x59 xAE	~~xFD xE9 x40 x57~~
Set_2	0	~~0~~	~~011000~~	~~0~~	~~110110~~	~~x53 x74 x65 x70~~	~~x68 x65 x6E x21~~
Set_3	0	1	111100	1	110000	x80 x0C x00 x0F	x80 x07 x80 x06

- We ignore the entries crossed out in red because they are not valid entries in the cache.
- There are 3 valid entries in the cache. One must hold the instruction, one must hold the page table entry, and one must hold the page directory entry.
- The values underlined in green look like PDE/PTEs. (there is a valid bit at the MSB of each)
- The cache blocks in Set3 must contain the PTE and PDE and they cannot be in the same cache block.
- Looking at the LRU bit (due to the nature of page table walking), we can determine that x800C is the PDE and the PTE is either x8007 or x8006.
- We notice that all interesting data is in sizes of two bytes and aligned.
- From the cache we can find the addresses associated with each piece of data. Mem[x1C4] = x6043 Mem[x1C6] = x59AE Mem[x3CC] = x800C Mem[x30C] = x8007 Mem[x30E] = x8006

0x3184 is the address of the instruction being executed. Below is the binary representation of the address.

16h'3184 = 16b'00110_00110_000100

PD Index: 6 PT Index: 6 Page Index: 4

We see that the instruction must have a physical address with the following constraint: xxxx000100, because of the page index. Mem[x1C4] = x6043 is the only entry in the cache that follows this rule.
Now we know that x6043 is the instruction (i.e., a load instruction which is decoded into R0 = Mem[R1 + 6]). Since pages are aligned, we know that the page containing the two possible PTEs starts at address x300: Mem[x30C] = x8007 and Mem[x30E] = x8006. Because the page table index is 6, we know that x8007 is the PTE. We must multiply the index by two because the size of each entry is 2 bytes. We have determined which entries in the cache are the PTE, PDE and the instruction.

(d) What are the values of R0 and R1? There are two possible answers to this question and either one will get full credit. Show your work.

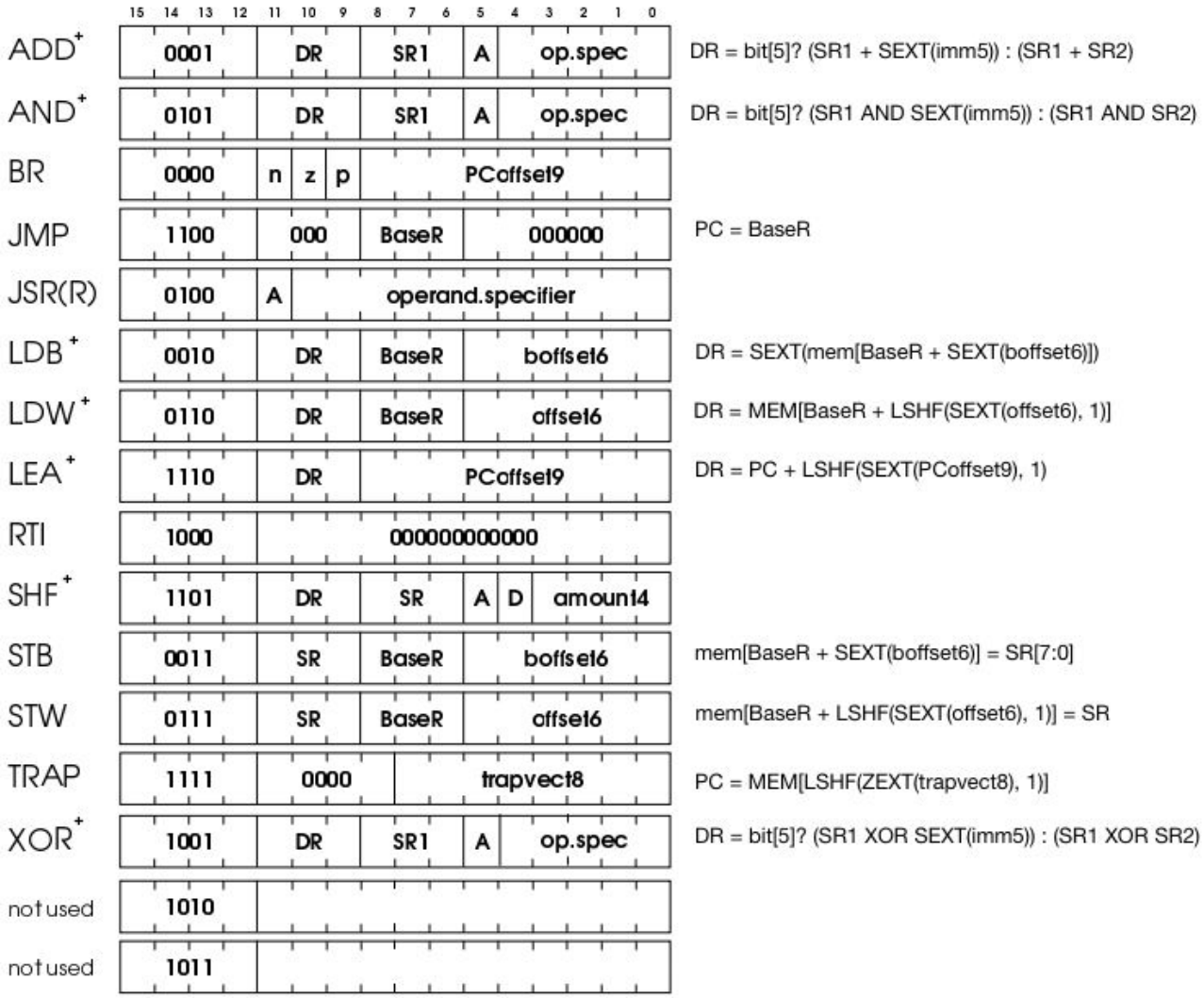
R0: x6043 or x59AE R1: x317E or x3180
Since the instruction that was executed was a load, we must walk through the page tables once again with the unknown address (R1 + 6). Given that there are no other entries valid in the cache, we can infer that the this address hits the same page. By this logic, either 0x6043 or x59AE is the data that was loaded into R0. Depending on which data value you chose, the address in R1 will be either x317E or x3180 (remember we added 6 to this address before probing memory).

Scratchpad

Scratchpad

Scratchpad

Initials:


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
ADD+ 0001 DR SR1 A op.spec DR = bit[5]? (SR1 + SEXT(imm5)) : (SR1 + SR2)
AND+ 0101 DR SR1 A op.spec DR = bit[5]? (SR1 AND SEXT(imm5)) : (SR1 AND SR2)
BR 0000 n z p PCoffset9
JMP 1100 000 BaseR 000000 PC = BaseR
JSR(R) 0100 A operand.specifier
LDB+ 0010 DR BaseR boffset6 DR = SEXT(mem[BaseR + SEXT(boffset6)])
LDW+ 0110 DR BaseR offset6 DR = MEM[BaseR + LSHF(SEXT(offset6), 1)]
LEA+ 1110 DR PCoffset9 DR = PC + LSHF(SEXT(PCoffset9), 1)
RTI 1000 000000000000
SHF+ 1101 DR SR A D amount4
STB 0011 SR BaseR boffset6 mem[BaseR + SEXT(boffset6)] = SR[7:0]
STW 0111 SR BaseR offset6 mem[BaseR + LSHF(SEXT(offset6), 1)] = SR
TRAP 1111 0000 trapvect8 PC = MEM[LSHF(ZEXT(trapvect8), 1)]
XOR+ 1001 DR SR1 A op.spec DR = bit[5]? (SR1 XOR SEXT(imm5)) : (SR1 XOR SR2)
not used 1010
not used 1011