CS 61C Fall 2018 — RISC-V Addressing and Caches

Discussion 5: September 24, 2018

1 RISC-V Addressing

We have several *addressing modes* to access memory (immediate not listed):

1. Base displacement addressing adds an immediate to a register value to create a memory address (used for lw, lb, sw, sb).
2. PC-relative addressing uses the PC and adds the immediate value of the instruction (multiplied by 2) to create an address (used by branch and jump instructions).
3. Register Addressing uses the value in a register as a memory address (jr)

1.1 What is range of 32-bit instructions that can be reached from the current PC using a branch instruction?

The immediate field of the branch instruction is 12 bits. This field only references addresses that are divisible by 2, so the immediate is multiplied by 2 before being added to the PC. Thus, the branch immediate can move the reference 2-byte instructions that are within $[-2^{11}, 2^{11}-1]$ instructions of the current PC. The instructions we use, however, are 4 bytes so they reside at addresses that are divisible by 4 not 2. Therefore, we can only reference half as many 4-byte instructions as before, and the range of 4-byte instructions is $[-2^{10}, 2^{10}-1]$

1.2 What is the range of 32-bit instructions that can be reached from the current PC using a jump instruction?

The immediate field of the jump instruction is 20 bits. Similar to above, this immediate is multiplied by 2 before added to the PC to get the final address. Since the immediate is signed, the range of 2-byte instructions that can be referenced is $[-2^{19}, 2^{19}-1]$. As we actually want the number of 4-byte instructions, we actually can reference those within $[-2^{18}, 2^{18}-1]$ instructions of the current PC.

1.3 Given the following RISC-V code (and instruction addresses), fill in the blank fields for the following instructions (you'll need your RISC-V green card!).

```
1  0x002cff00: loop: add t1, t2, t0
2  0x002cff04:       jal ra, foo
3  0x002cff08:       bne t1, zero, loop
4  ...
5  0x002cff2c: foo:  jr ra
```

ra= 0x002cff02

1	0x002cff00: loop: add t1, t2, t0	0	5	7	0	6	0x33
2	0x002cff04: jal ra, foo	0	0x14	0	0	1	0x6F
3	0x002cff08: bne t1, zero, loop	1	0x3F	0	6	1	0x63

② Jal imm = 40 = 0000000000101000 / 2 = 0000000000010100 = 0x14

```
4  ...
5  0x002cff2c: foo:  jr ra                ra=__0x002cff08___
```

2 Understanding T/I/O

When working with caches, we have to be able to break down the memory addresses we work with to understand where they fit into our caches. There are three fields:

Tag - Used to distinguish different blocks that use the same index - Number of bits: leftovers
Index - The set that this piece of memory will be placed in - Number of bits: $\log_2$(# of indices)
Offset - The location of the byte in the block - Number of bits: $\log_2$(size of block)

2.1 Assume we have a direct-mapped byte-addressed cache with capacity 32B and block size of 8B. Of the 32 bits in each address, which bits do we use to find the index of the cache to use?

$Index = \log_2\left(\frac{32}{8}\right) = 2$
$offset = \log_2(8) = 3$

We use the 4^{th} and 5^{th} least significant bit

2.2 Which bits are our tag bits? What about our offset?

$Tag = 32 - 3 - 2 = 27$

The offset is 3 bits, and our tag is the remaining high-order bits.

2.3 Classify each of the following byte memory accesses as a cache hit (H), cache miss (M), or cache miss with replacement(R). It is probably best to try drawing out the cache before going through so that you can have an easier time seeing the replacements in the cache. The following white space is to do this:

Address (Tag | Index | Offset)	T/I/O	Hit, Miss, Replace
0x00000004 (000 | 0 | 100)		M compulsory
0x00000005 (000 | 0 | 101)		H
0x00000068 (011 | 0 | 000)		M compulsory
0x000000C8 (110 | 0 | 000)		R compulsory
0x00000068 (011 | 0 | 000)		R conflict
0x000000DD (110 | 1 | 101)		M compulsory
0x00000045 (010 | 0 | 101)		R compulsory
0x00000004 (000 | 0 | 100)		R capacity
0x000000C8 (110 | 0 | 000)		R capacity

- 0x00000004 Index 0, Tag 0: M, Compulsory
- 0x00000005 Index 0, Tag 0: H
- 0x00000068 Index 1, Tag 3: M, Compulsory
- 0x000000C8 Index 1, Tag 6: R, Compulsory
- 0x00000068 Index 1, Tag 3: R, Conflict
- 0x000000DD Index 3, Tag 6: M, Compulsory
- 0x00000045 Index 0, Tag 2: R, Compulsory

cache

Index	Tag	
0	000	→ 010 → 000
1	011	→ 110 → 011 → 110
2	...	
3	110	

- 0x00000004 Index 0, Tag 0: R, Capacity
- 0x000000C8 Index 1, Tag 6: R, Capacity

3 The 3 C's of Misses

if not seen block before:
=> Compulsory
else if from start, fully associative, if miss would also conflict Tags
else:
capacity.

3.1 Classify each M and R above as one of the 3 types of misses described below:

I. Compulsory: First time you ask the cache for a certain block. A miss that must occur when you first bring in a block. Reduce compulsory misses by having a longer cache lines (bigger blocks), which bring in the surrounding addresses along with our requested data. Can also pre-fetch blocks beforehand using a hardware prefetcher (a special circuit that tries to guess the next few blocks that you will want).

II. Conflict: Occurs if you hypothetically went through the ENTIRE string of accesses with a fully associative cache and wouldn't have missed for that specific access. Increasing the associativity or improving the replacement policy would remove the miss.

III. Capacity: The only way to remove the miss is to increase the cache capacity, as even with a fully associative cache, we had to kick a block out at some point.

Note: There are many different ways of fixing misses. The name of the miss doesn't necessarily tell us the best way to reduce the number of misses.

4 Extra Practise

In the following diagrams, each blank box represents 1 byte (8 bits) of data. All of memory is byte addressed.Let's say we have a 8192KiB cache with an 128B block size, how many bits are in tag, index, and offset? What parts of the address of 0xFEEDF00D fit into which sections?

$\log_2\left(\frac{2^{23}}{2^7}\right) = 16$

$\log_2(2^7) = 7$

	Tag	Index	Offset
Number of bits			
Bits of address			

32 − 16 − 7 = 9

4.1

	Tag	Index	Offset
Number of bits	9	16	7
Bits of address	111111101 (0x1FD)	1101101111100000 (0xDBE0)	0001101 (0x0D)

4.2 Now fill in the table below. Assume that we have a write-through cache, so the number of bits per row includes only the cache data, the tag, and the valid bit.

Address size (bits)	Cache Size	Block Size	Tag Bits	Index Bits	Offset Bits	Bits per row
16	4KiB	4B				
32	32KiB	16B				
32			16	12		
64	2048KiB			14		1068

Address size (bits)	Cache Size	Block Size	Tag Bits	Index Bits	Offset Bits	Bits per row
16	4KiB	4B	4	10	2	32 + 4 + 1 = 37
32	32KiB	16B	17	11	4	128 + 17 + 1 = 146
32	64KiB	16B	16	12	4	128 + 16 + 1 = 145
64	2048KiB	128B	43	14	7	1068