Jon Dis 5 Notes

Sunday, September 23, 2018 5:36 PM

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? 12 bits $\Rightarrow$ $[-2^{11}, 2^{11}-1]$ half words

$\Rightarrow$ $[-2^{10}, 2^{10}-1]$ words

☆ 1.2 What is the range of 32-bit instructions that can be reached from the current PC using a jump instruction? 20 bits $\Rightarrow$ $[-2^{19}, 2^{19}-1]$ half words

$\Rightarrow$ $[-2^{18}, 2^{18}-1]$ words

☆ 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
```

neg offset

+4B

op codes

0	5	7	0	6		0x33
0	0x14	0	0		1	0x6F
1	0x3F	0	6	1	0xC | 1	0x63

ra= 0x002cff08

Green sheet

func7	rs2	rs1	func3	rd	opcode

t1 = x6
t2 = x7
t0 = x5

add : R type
jal : UJ type
bne : SB type

① find type

② find format

③ fill in w/ instruction

$3\frac{1}{2}$ find offsets

jal: $0x002cff2c - 0x002cff04 = 0x28 = 0b\ 0010\ 1000$

20 bits, halfwords

$= \boxed{0\ 0 \ldots 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0 /\ 0}$

20 |—10:1—| 0 ← bit #

Offset: find data in block
index: find block in cache
tag: ensure we're looking @ correct block

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 leftover bits

find first

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)

log(# indices) log(Block Size)

$\frac{\text{cache size}}{\text{block size}}$ for direct mapped cache

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?

Offset $= \log 2^3 = 3$ $\frac{32B}{8B} = 4$ blocks $\Rightarrow$ index $= \log 2^2 = 2$

31 ... 5	4 3	2 1 0
T	I	O

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

Tag: 31:5 Index: 4:3 Offset: 2:0

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:

use TIO from before

Address	T/I/O			Hit, Miss, Replace
0x00000004 ... 0100	0	0	4	M, Comp
0x00000005 ... 0101	0	0	5	H
0x00000068 0110 1000	3	1	0	M, Comp
0x000000C8 1100 1000	6	1	0	R, comp
0x00000068 0110 1000	3	1	0	R, conflict
0x000000DD 1101 1101	6	3	5	M, Comp
0x00000045 0100 0101	2	0	5	R, comp
0x00000004 ... 0100	0	0	4	R, capacity
0x000000C8 1100 1000	6	1	0	R, Capacity

I O

① ② ③ ④ 4 blocks of data brought in ⇓ Theoretical Fully Assoc. Cache is now full!

Cache

8B of data here

index	Tag	contents	valid
00	~~0~~ ~~2~~ 0	—	1
01	~~3~~ ~~6~~ ~~3~~ 6	—	1
10			0
11	6	—	1

We KNOW it is a capacity miss b/c if we had a FA cache, we still would have missed

3 The 3 C's of Misses

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

TL;DR

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).

first time seen

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.

can organize better

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.

cache full, need to enlarge

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 ~~Exam~~ 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?

32 − 16 − 7 = 9 $2^{13}\cdot 2^{10}/2^7$ $\log 2^7 = 7$

	Tag	Index	Offset
Number of bits	9	16	7
Bits of address	1111 1110 1	110 1101 111100000	0001101

$0x\ \underbrace{1111\ 1110\ 1}_{T}\underbrace{110\ 1101\ 1111\ 0000\ 0}_{I}\underbrace{000\ 1101}_{O}$

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.

Data + tag + valid

	Address size (bits)	Cache Size	Block Size	Tag Bits	Index Bits	Offset Bits	Bits per row
1.	16	4KiB	4B	4	10	2	37
2.	32	32KiB	16B	17	11	4	146
3.	32	64KiB	16B	16	12	4	145
4.	64	2048KiB	128B	43	14	7	1068

only 1 less tag bit

1. O: log 4 = 2 I: $\log \frac{2^{12}}{2^2} = 10$ T: 16 − 10 − 2 = 4 32 + 4 + 1 = 37

2. O: log 16 = 4 I: $\log \frac{2^{15}}{2^4} = 11$ T: 32 − 11 − 4 = 17 128 + 17 + 1 = 146

3. O: 32 − 16 − 12 = 4 Block: 2^4 = 16 B size: $2^{12}\cdot 16B = 2^{16}$ = 64KiB

4. Block: $\frac{2^{11}\cdot 2^{10}}{2^{14}} = 2^7$ = 128B O: $\log 2^7 = 7$ T: 64 − 14 − 7 = 43