Cache (T9-T10)

VE370SU22 TA Runxi Wang

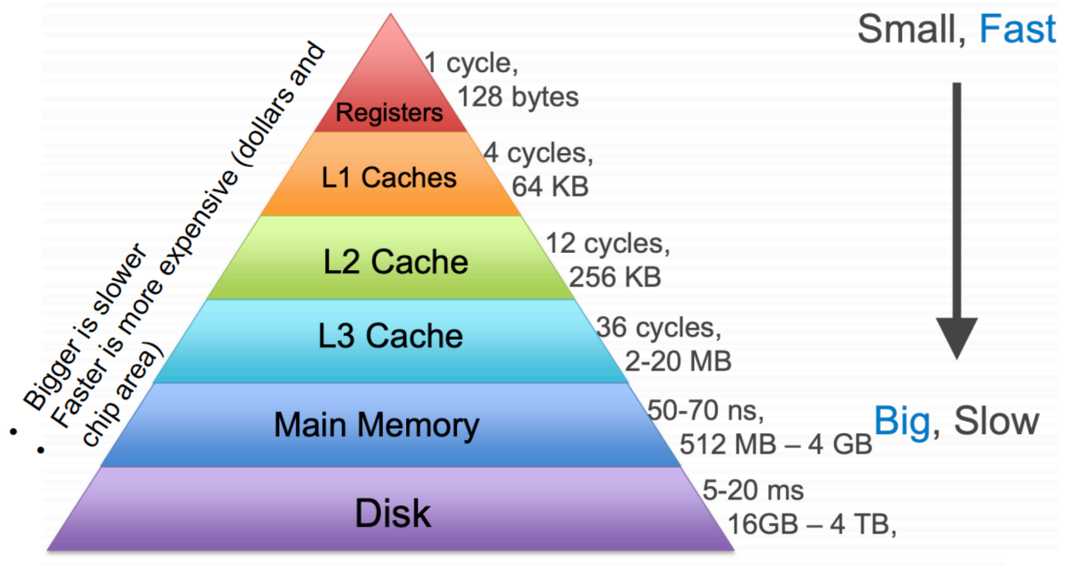
Memory Basics

Memory Cell

- SRAM (static random access memory): quicker access, higher cost, 6 transistors in a bit cell
- DRAM (dynamic random access memory): slower access, lower cost, 1 transistor and 1 capacitor in a bit cell

Memory Hierarchy

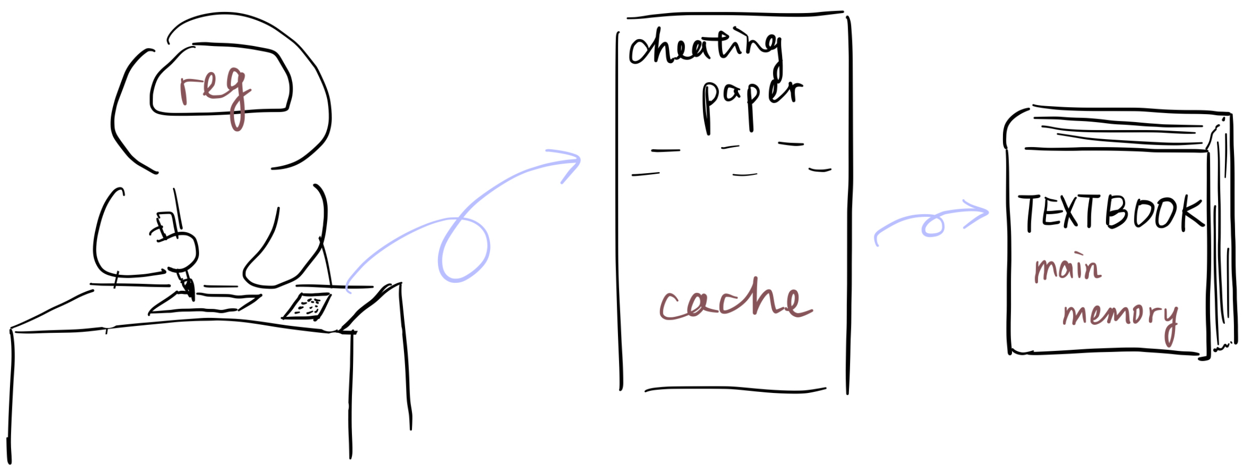
- Why? Low main memory access speed, but fast memory (e.g. SRAM) is costly,
- What is new here? We introduce cache, which composes of SRAM, that directly communicates with CPU and stores a small fraction of data in main memory.
- Complete memory hierarchy: (We may have caches of several levels)



- Influences for instruction memory and data memory?

  Let's assume that our instruction memory and data memory is stored in separate memory part(check Von Neumann Architercture & Havard Architecture if you are interested:). Instruction memory and data memory have their own cache and main memory.

Locality



Two Localities

- Temporal locality: Items that are accessed recently are likely to be accessed again soon
- Spatial locality: Items near those that are accessed recently are like to be accessed soon

Access Memory Hierarchy

Block, Word, Byte

1 block = 1-n words
1 word = 4 bytes

Address
Example.

Suppose a 10-bit address and each block has 4 words.

Byte Address (Word Number[9:2]+Byte Offset[1:0])	Contents
11111100 00	Byte1
11111100 01	Byte2
11111100 10	Byte3
11111100 11	Byte4

Word Number (Block Number[7:2]+Word Offset[1:0])	Contents
111111 00	Word1
111111 01	Word2
111111 10	Word3
111111 11	Word4

Hit & Miss

- Hit: Accessed data present in upper level (e.g. hit in cache)
- Miss: Accessed data is absent. And then we need to spend more time to fetch data from a lower level
- Hit time: time to determine whether a hit or miss + time to pass the requested block to requestor. For miss cases, we need addtional miss panelty
- Hit rate: hits/accesses; Miss rate: 1 - hit rate

Direct Mapped Cache

Definition: Each memory location corresponds to one choice in cache.

Cache Location (Cache Block Index)
cache location = lower log2(number of blocks in the cache) bits of block address
e.g. Assume 8-bit block address 1111_1001 and there are 4 blocks in cache. Its cache location is 01 (the second slot in the cache)

Tag

tag = high-order bits of memory block address, excluding lower log2(number of blocks in the cache) bits of block address

Valid Bit

For each block (line) in cache, we assign a "valid" bit to indicate whether the data in this line is ready to be used. So when searching data, we first check whether "valid" = 1 and then check whether "tag" = the tag we want.

Example

We assume that we have a 4-block cache, and each of the block has 2 words. And we have 8-bit address, which means

block number[7:3]+word offset[2]+byte offset[1:0]

We have the following memory access command to be processed,

```
lw R5 <- mem[9] # word addr without byte offset: 001001
lw R6 <- mem[21] # 010101
lw R7 <- mem[20] # 010100
lw R28 <- mem[4] # 000100
```

Suppose the cache is unfilled initially.

Index	V (valid)	Tag	Word 1	Word 2
00	N			
01	N			
10	N			
11	N			

1. `lw R5 <- mem[9]`
   read miss, fetch data from main memory

Index	V (valid)	Tag	Word 1	Word 2
00	Y	001	mem[8]	mem[9]
01	N			
10	N			
11	N			

2. `lw R6 <- mem[21]`
   read miss, fetch data from main memory

Index	V (valid)	Tag	Word 1	Word 2
00	Y	001	mem[8]	mem[9]
01	N			
10	Y	010	mem[20]	mem[21]
11	N			

3. `lw R7 <- mem[20]`
   read hit

Index	V (valid)	Tag	Word 1	Word 2
00	Y	001	mem[8]	mem[9]
01	N			
10	Y	010	mem[20]	mem[21]
11	N			

4. `lw R28 <- mem[4]`
   read miss, fetch data from main memory, replace the conflict block in cache

Index	V (valid)	Tag	Word 1	Word 2
00	Y	001	mem[8]	mem[9]
01	N			
10	Y	000	mem[4]	mem[5]
11	N			

Handle Data Write

When processors write data to cache, when should we update the corresponding data in main memory? We need to maintain the data consistency cross the memory hierarchy.

Write Through

Write the data to main memory once it is updated in cache

For hit, processor writes cache, and then cache writes main memory right afterwards.
For miss, cache fetch corresponding block from main memory, processor writes cache, and then cache writes main memory.

Disadvantage
This is time consuming!

Solution
Write buffer: Buffer holds the data to be written from cache to main memory so that processors do not need to wait that long if the buffer is not full.

Write Back

Write the data back to main memory only when the block in cache is needed to be eliminated and is written by processors previously ("dirty")

Add a "dirty" bit in the block (line) in cache to indicate whether the data in this block is modified.

For hit, processor writes cache and it is done here.
For miss, check the "dirty" bit of the conflicted block and modifiy main memory according for this block. Next, fetch requested block from main memory to cache and then write it.

Write allocation

For write through on miss:

Write allocate: Allocate cache block on miss by fetching corresponding memory block. Then modify the cache block and main memory block

No write allocate: Write directly to main memory and then fetch the block to cache.

For write back on miss:

Usually use write allocate.

Cache Performance

Components of CPU time = program execution cycles (hit time included) + memory stall cycles

Memory stall cycles = memory-access instruction count * miss rate * miss panelty

Example:

Given instruction cache miss rate 1%, data cache miss rate 4%, miss panelty for both 100 cycles, base CPI is 2, and we have 50% memory access instructions. Then what is the total CPI?

total CPI = base CPI + instruction cache total miss cycles + data cache total miss cycles
= 2 + 100% * 1% * 100 + 50% * 4% * 100
= 5

Reference

[1] VE370SU22 slides T9
[2] VE370SU22 slides T10