Memory and its Performance

June 17, 2023

Contents

1 Beginning

- Note taken on *[2023-06-07 Wed 22:37]*
  This chapter has huge amount of content, but arranged in a shitty way. Becareful.

1.

2. Main Memory. Main

3. Cache. Main CPU (Processor)

4. Memory. Main .

5. . External . .

2

- Note taken on *[2023-06-07 Wed 22:24]*
  And you don't have to remember the classification here. You can just learn it, with no pressure.
- Note taken on *[2023-06-07 Wed 22:22]*
  This section is absolutely shit.

?

- CPU , , , .
- , IO , IO , CPU . CPU .

, , , . , , Memory Heirarchy. :

1. ;

2. ;

3. .

. .

2.1

- Note taken on *[2023-06-16 Fri 12:53]*
  You should the word "random accessing" which means that you can access to the memory specified by the address you have given.

:

1. SemiConductor . volatile , " " .
2. . Magnetic Disk. .
3. . ?.
4. . .

1. RAM. . SRAM (Static RAM) DRAM (Dynamic RAM).
2. ROM. . , EPROM, EEPROM, Flash Memory. , Flash Memory , ROM , , .
3. . , Random accessing . RA , . , , , . .

• , . .

2.2 Memory heirarchy

. CPU, , ; CPU, , , . ,,

```
CPU <->  <-> Main  <->
```

huancun refers to the cache. Cache is usually not visible to the programmers. Programmers mostly manipulate with the memory, which is here, the main memory. fucun refers to the external memory, for example disk.

Main . CPU Main . hit rate , hit rate 1 , , CPU "" Main , , Main . , Main Cache Internal Memory. , , External Memory. , Chip . External Memory , Bus. .

---

main CPU . , . Main , Main "" Main . .
: 1. (Logical) ; 2. (Physical) . , , , Chip , .

2.3 Main

Word: word . 4 bytes, . , . n, word 2^m bytes , "" 2^{n-m} 2^n.
word, bytes
: , "", , , , "". "". .
: , : 1. Capacity; 2. Speed.

- Capacity: Main bits , bytes. . , bytes . 1M , 20. 2^{20} 1M, , 1M bytes.
- Speed: . Latency
- : , . , bandwidth Speed .

3 SemiConductors Chips

3.1

. , ; : RAM , , RAM , . , , word. , .

3.2 RAM

3.2.1 SRAM DRAM

- Note taken on *[2023-06-16 Fri 15:02]*
  The unit of DRAM can be consist of 4 transistors. But we can simplify it such that it consist of 1 transistor.

SRAM (6,), DRAM 1 (). , , , , , DRAM fade away, , . , , .
DRAM : . . , , (Capacitor) , , , recharge.
SRAM : . , SRAM . SRAM Latch. latch. : A_1, A_2. bit , B, B A_1, bar B A_2. bit .
OK, , . Tang 76 . , Tang . 81 DRAM , Tang . Stallings .

3.2.2 SRAM DRAM

. wikipedia for more information. , . , SRAM DRAM ? . .
DRAM SRAM , :

- ,
- ,

, . , DRAM . , . .

One should learn the structure of DRAM and then the structure of SRAM. Because you need to know how the transistor works, which is very important.

OK, , . Tang 76 . , Tang . 81 DRAM , Tang . Stallings .

1. The structure of DRAM

   DRAM : . . , , (Capacitor) , , , recharge.

   DRAM consists of a capatitor (idk how it spell) and three transistor (and we also have a version with only one transistor).

   The transistor works like a gate. The middle input can be viewed as a handle. If it is on the signal can go through the transistor, (the gate is open).

   The capacitor store some ; if the gate is open, the current just let go; if the gate is closed, the current remains. The read process is the exact process of let go the electron stored in capacitor.

   Conversely, the process of write is to store some electron into the capacitor.

2. The structure of SRAM

   SRAM : . , SRAM . SRAM Latch. latch. : A_1, A_2. bit , B, B A_1, bar B A_2. bit .

   Simple practise: go and check the book. Because I can't draw a picture here. If you think the book is shit, then you can check *Stallings*' book.

3.2.3 SRAM

- Note taken on *[2023-06-16 Fri 17:53]*
  we analyse a cycle of read or write operation. The start of the cycle and the end of the cycle are marked by the change of the address line. That is to say, if the address line change, then we are heading to next cycle. What we want to know is what elements are here to compose the minimun of the cycle time.

Although the analysis seems so useless, you should read it as well.

Read: Anyway, we need to figure out the signals first. As you can see, the addresss line is the input, and the there is a signal called pianxuan signal (OK, the input method is down for now, I can just type in English). And the output is the data line. Namely, we have:

- **A**: address line: usually it is 32-bit long or 64-bit long. It depends on the system, you know. Our device is mainly x64, that is to say the address line is 64-bit long.
- $\overline{\mathbf{CS}}$: the signal. CS is . It is triggerred when the signal become 0
- $\mathbf{D_{out}}$: the data we get from the chip.

All this is what we need to analyse the read operation. The process is like 1. The address is ready; 2. CS is ready; 3. Consequently, the Data is ready. The rest of the text is all blah.

Write: Write is relatively complex. We have the write enable signal, namly, $\overline{\mathbf{WE}}$. And we should note that when the data is not valid, it is at .

- **A**: the address line
- $\overline{\mathbf{WE}}$: Write enable
- $\mathbf{D_{in}}$: the input Data, that is the data that we want to write.

When the address line is ready, the write enable and cs signal should wait for a moment for the data line ready. After the dataline is (almost) ready, we enable the write function. And then we write the data to the memory. And some how idk why we need to hold on for a sec, and then we shall continue.

It takes many procedure: 1. A ready; 2. wait; 3. WE and CS is valid; 4. wait for writing done; 5.(we and cs is not valid now) and another wait (this wait is called write restoration time); 6. OK for next cycle.

what we have here is a lot of phase: 2. wait Data line be ready; 4. wait for writing operation (latency); 5. another wait (write restoration).

Here is some other thing that you should notice; you can check out the book.

3.2.4 TODO The Examples of SRAM and DRAM

This section is like shit.

What we are going to do is to analyse the pictures in the book which show some examples of SRAM and DRAM chips.

None of the example is useful. But it is a chance for use to practise some skills of shit-eating.

3.2.5 TODO DRAM

3.2.6 DRAM

, . 128 Œ 128 DRAM . 128 64 s, 2ms .
: 2ms 64 s . .
: .
: $\frac{64\,\mu s}{128}$. , . .

3.2.7 The comparison between DRAM and SRAM

This one is relatively simple. You should be already very clear about it after you have learned all these shit.

3.3 ROM

3.3.1 ROM

- **ROM** (read-only memory rom) (long ago). , (); , . The read operation is a little bit tricky here. Anyway if thre is transistor (MOS), then the corresponding bit is **1**.
- **PROM** (Programmable ROM) , program. program . ROM.
- **EPROM** (Erasable) , optically erasable. . .
- **EEPROM** (electrically erasable) , .
- **Flash Memory** , RAM .

3.3.2 EPROM

, , , . . . D , , . , 0. , 1.
. .

3.4 Chips CPU (important)

3.4.1 How to Deal lianxian

- Note taken on *[2023-06-16 Fri 18:42]*
  The problem here is a completely garbage. As you can see the previous chapter just finished saying things like "the direct link between the CPU (processor) and the memory will slow down the performance". Shit. Now, what are we doing? Garbage actually.

CPU RAM ROM chip :
chips (you can't use other chips) (in general case), CPU chip . , . . , ROM RAM CS .
:

1. . 74138 .
2. , chip
3. CPU . .
4. . . , CPU MREQ . (MREQ signal is the signal that saying the cpu is going to read / write memory or not. If MREQ is no valid, then the whole memory should not be working).

Tang 99. . . . , .

3.4.2 TODO Example of the lianxian of chips and cpu

- Note taken on *[2023-06-16 Fri 18:48]*
  The problems here are boring. But it may vary, in a shitty way though.

SHIT

3.5 Parity

Stallings .

3.5.1

Stallings . . , f, K bits . .
, , . XOR , XOR Syndrome Word.
N . K bits , 2^K . :

$$2^K - 1 \geq N + K$$

K bits . K. , Syndrome Word 0 , . -1,

3.5.2 Hamming Code

Hamming Code single error correction code. . n XOR . Stallings .
, Hamming Code. Hamming Code bits . 2 , Hamming Code . C1 C2 C4 Code , Dn n . .

```
C1C2D1C4D2D3D4
```

4 (D stands for data), (C for idk) . Cn. , , , 6 110. C1 , 1 XOR. C2 " 1 XOR".

001	010	011	100	101	110	111
1	2	3	4	5	6	7
C1	C2	D1	C4	D2	D3	D4

```
C1 = D1 ^ D2 ^ D4
```

3.6 DONE

- State "DONE" from "TODO" *[2023-06-17 Sat 02:02]*

3.6.1

: 1. ; 2. . Stallings . .
: ""? . . , RAM . , , , , , .
: Stallings. : 1. Synchronous DRAM; 2. Rambus DRAM; 3. Cache DRAM. . DRAM, , , Tang , . , Cache .

- **Synchronous DRAM**, . RAM . . , , CPU , . SDRAM Burst Mode. , (word), SDRAM , . burst mode. wikipedia.
- **Rambus DRAM**. Use bus and modules (and of course a control unit)
- **Cache DRAM**. Patterson . . (introduce more concepts) , Cache , SRAM buffer.

3.6.2

bandwidth , n. , n , . T/n .

3.6.3

- Note taken on *[2023-06-16 Fri 19:13]*
  ; . Tang .

: 1. ; 2. . , .
. : , , . , .
. , . () , . . , , , . , . , .
Please read the note under the current item.

3.6.4

. Patterson ; .

3.6.5 DONE Synchronous DRAM

- State "DONE" from "TODO" *[2023-06-16 Fri 20:56]*
- Note taken on *[2023-06-16 Fri 20:54]*
  Things are weird. There is bus. Does the cpu really have to wait for the output of the data without SDRAM?

The idea is the key. SDRAM use a clock. The clock enables an important feature, namly the **burst mode**.

When we want to retrive data from memory, we inform memory with address; then the data is retrived from memory, we capture it; we inform memory with another address (usually this address is adjacent to the previous one); the same thing happens. The cpu have to **wait** for output data. And specify every address of every blocks.

Why? Maybe it is because that it doesn't have a clock. Anyway SDRAM fixes the problem here: the cpu call the memory to do some read / write operation, and then cpu just goes back to its own business; after the data is available, the cpu capture the data (**without waiting for the output**).

In **burst mode**, we want to get a series of blocks; we inform the memory with the address (the start address); then the memory just keeps outputing data without the need to informing the address again. that is why it is called **burst mode**.

DDR is short for **Double Data Rate SDRAM**. Anyway, it is fast. I mean, fast.

For more information, please check wikipedia.

3.6.6 TODO Rambus DRAM

- Note taken on *[2023-06-16 Fri 21:35]*
  So Stallings wrote shit too.
- Note taken on *[2023-06-16 Fri 21:02]*
  Gonna drown in all that Tang shit. How can a man just keeping shit around?

Source: Stallings

DRAM use **block-oriented** protocol to deliver address and control information (that is actually to say it use something like bus, but the transfer of the information is asychronous).

Much of the details are omitted here. Just get it and that will be fine.

Anyway, you can check out the Stallings for more information.

3.6.7 Cache DRAM

One shall read Patterson for this section.

This one is rather simple.

4 Cache

4.1 An introduction

- *miss*, *miss penalty*, *hit*, *hit rate*

This section tells the principle of cache and then tells some key concepts like **hit**, **miss** and so on.

Anyway. A cache is memory that lies between processor and main memory. It is used to speed up the accessing time. Some blocks of the data are loaded into cache. (Mind the word "block") And if processor want to access to the data, it will check cache first. And if the data is indeed in the cache, then the processor can just get the data via accessing to cache, none of the main memory's business.

So it will speed up the accessing time, since the cache is faster (and is more expensive than main memory).

If the data is on the cache, then it is called a **hit**; if not, it is called a **miss**. If a miss occurs, we will have to access to the main memory, and send the data to cache and to processor. The extra time that it takes is called **miss penalty**.

4.2 DONE The Elements of Cache

- State "DONE" from "TODO" *[2023-06-16 Fri 21:59]*

Anyway, we are going to talk about the structure of the cache here. And moreover we are going to talk something about the attributes of a cache.

4.2.1 Hit and Miss

- Note taken on *[2023-06-16 Fri 19:23]*
  This is called improve stability by **redundancy**.

Anyway. A cache is memory that lies between processor and main memory. It is used to speed up the accessing time. Some blocks of the data are loaded into cache. (Mind the word "block") And if processor want to access to the data, it will check cache first. And if the data is indeed in the cache, then the processor can just get the data via accessing to cache, none of the main memory's business.

So it will speed up the accessing time, since the cache is faster (and is more expensive than main memory).

If the data is on the cache, then it is called a **hit**; if not, it is called a **miss**. If a miss occurs, we will have to access to the main memory, and send the data to cache and to processor. The extra time that it takes is called **miss penalty**.

4.3 The Structure of the Cache

- Note taken on *[2023-06-16 Fri 19:27]*
  For the name of **line**, Stallings has discussed about it: it is used to distinguish between that in cache and that in memory.

- Note taken on *[2023-06-16 Fri 19:24]*
  For principle of locality, check either Stallings or Patterson.

This part is rather simple, for we have already been familiar with the structure of cache.

The data transferred between cache and main memory is by **blocks**. A block consists of words.

It uses the principle of locality, to improve the performance. So we know that a block has usually more than one word (it won't use spatial locality if it does). The main memory is divided into blocks. Cache can load blocks of data from memory. The **mapping** between the blocks number in main memory and the block number in cache (that is called **line** number, which is the position that in cache) is a topic in next some section.

A line in a cache is the **basic unit** of a cache. It consists of a block and some **extra** information field including **tag field** and **valid-tag field**. The name of line is used, to note the difference of the blocks that in main memory and that in cache, and to note that there is some other information in the cache line.

4.3.1 The REAL Structure of Cache

Cache lies between processor and memory.

There are two more important parts: 1. Mapping; 2. Replacing.

Mapp is the map from **read address** to the address in cache. Replacing is about how we deal with the situation where the cache is full. These two part require at least two modules.

So the structure of cache consists of 1. processor; 2. cache; 3. memory; 4. map module; 5. replace module; 6. bus.

4.3.2 Block Size and its Effect to Hit Rate

Source: Patterson

According to the principle of locality, the bigger block size can improve the hit rate, subsequently improving the performance.

But if we consider the latency (that is the miss penalty), things get interesting, because if the improvement brought by the increase of hit rate is no greater than the **degeneration** brought by the increase of miss penalty, then the performance is being worse as the block size grows. For more information, you can check *Patterson* for more information.

Moreover, the miss rate will go up eventually if the block size keeps increasing. Because as the blocksize goes up, total number of the blocks is low.

4.4 DONE Mapping Strategy

- State "DONE" from "TODO" *[2023-06-17 Sat 00:21]*

The mapping is from the blocks in the memory to the line in the cache, that is to say when given the position of a block, how do we find the corresponding position in the cache? There are some strategies of mapping.

The simplest one is called **Direct Mapping**. It is simple. There are also other ways called **associative mapping** and **set-associative mapping**.

4.4.1 Direct Mapping

Modula!

The details are omitted here. You can check Patterson and Stallings for more information.

4.4.2 (full) Associative Mapping

First, cut the word address into two field: 1. tag, 2. word. Word field is used to locate the data inside of the block. Tag is used to identify.

Anyway, initially, we give an address. We load the block into cache (into the first line). Store the tag into the line.

Next, afterwards, if we want to retrieve from the memory, we check the cache, to see if there is a line, whose tag is the same as the tag of the given address. If there is, then it is a hit; if not, it is a miss. If there is a miss, then load the block into the second line.

And we have done here.

The procedure is done here.

The feature is that the line number is not affected by the address of the memory. Randomly given an address, the line number could be anyone.

4.4.3 Set-Associative Mapping

Set-Associative mapping is to combine the two kinds of methods.

Let us look at direct mapping. The address is divided into two groups. Line number field is the line number. Yes, indeed.

1. line number field
2. word field

Let us look at associative mapping. The address is divided into two groups.

1. tag field
2. word field

In the combination of the two method, the address is divided into three groups.

1. tag field
2. set field
3. word field

where the tag field works just like that in associative mapping, and set field works just like that in the direct mapping: set field is the set number.

Anyway, the cache is divided into many sets. The set field determined the set number. JUST LIKE THAT IN DIRECT MAPPING. Commonly, the arrangement is like:

field	tag	set	word
address	xxx	xxxxxxx	xx
length	3	7	2

here I specify the length of the field. OK, then we have a 12-bits address. and the block size is four bytes; and there are 2^7 sets; there are 2^3 lines in one set; there are 2^{10} lines in the cache in total.

4.5 DONE Write Policy

- State "DONE" from "TODO" *[2023-06-17 Sat 00:30]*

For write policy, one can check the page 137 of Stallings.

4.5.1 Why We have to maintain the consistency of the memory heirarchy

In computer science, a consistency model specifies a contract between ther programmer and a system. The system guarantees that if the programmer follows the rules for operations on memory, memory will be consistent and the results of reading, writing, or updating memory will be predictable. This is important because it allows for reliable and predicable behavior of programs that rely on shared memory.

idk.

I just don't know why.

4.5.2 Two strategies

There are ways to maintain the consistency. In short, there are two ways: **write-through** and **write-back.**

Let us look at write-through, to check how it maintain the consistency. Write-through is to say, when you want to change some data, if it is on cache, you need to change the content of cache and that of the main memory.

4.5.3 Write Through

Write through is to say, when we **change** the data in **cache**, we also change the data that in **memory**. It is a good idea? It slows down the performance, right? The speed of writing data remains **unchanged**. The time it takes to write to cache is the same as that to memory.

4.5.4 Write Back

Write Back is to say, when we **change** the data in **cache**, we write the data **back** when that **block** is about to be subtituted. The speed is guaranteed, but its realization is more sophisticated.

4.5.5 TODO The More into Write Policy

4.6 DONE Ways to Improve the Performance of Cache

- State "DONE" from "TODO" *[2023-06-17 Sat 01:54]*

One can use multiple level of cache.

4.6.1 Multi-level Cache

Source: Stallings page 138

1. Before Muti-level Cache

   We add a level of cache. Let us say L1 and L2.

   It can improve the performance.

   But how?

   First we should know that what have constrained the speed of a memory chip. Ok, let me put it straight, the **latency** constrain the speed.

   Next we should know the principle of the cache. Of course, you know the principle of locality, but what we need here is something more essential:

   The speed of cache memory should be faster than the speed of the main memory.

   That is right. So, the speed of L1 should be above of that of L2. How can we do that?

   At the first place, when we consider one-level cache, the cache use SRAM. SRAM is more expensive and takes more space than DRAM and it is faster than DRAM. So SRAM can be the cache if the main memory is consist of DRAM.

   OK, that is very convincing, but does it have to do with multi-level cache?

   Of course, what we need here, is to find a way to improve the speed of SRAM, or specifically, to find a way to distinguish the speed of one kind of SRAM and the other, such that we can build multi-level cache.

2. The Bus

   WHAT HAD HAPPENED?

   The main point is the position of the SRAM chips. We need the knowledge of the bus here. If we put the SRAM chips on the **processor** 's chip, then the speed should be faster. Let us call those SRAM chips L1.

   And we put the L2 chips outside of the processor chip.

   The physical distance between the L1 and the processor is smaller. Consequently, the speed of the L1 should be faster. It is because the communication between L1 and processor does not depend on bus. But the communication between processor chip and L2 depend on bus.

   The dependency leads to the performance difference. So, as a result, we can build a multi-level cache. And indeed just like SRAM takes more space, the space on the processor chip is limited. So the size of L1 is limited too.

3. **WAITING** The performance analysis

   - State "WAITING" from "SOMEDAY" *[2023-06-17 Sat 03:03]*

4.6.2 WAITING Split Cache

- State "WAITING" from "SOMEDAY" *[2023-06-17 Sat 02:45]*
  when I review about the pipeline one shall complete the task.

Source: Stallings page 140

It is a way to split the cache into two parts:

1. One is for instructions
2. the other is for data

The performance is enhanced, for these reasons:

1. No write is need in instruction L1. Consequently, the implementations are different.
2. The key advantage is that it eliminates contention for the cache between the instruction fetch / decode unit and the execution unit.

The second advantage concerns with the pipeline design, which I have completely forget about. Shit.

4.7 DONE Replacement Algorithm

- State "DONE" from "TODO" *[2023-06-17 Sat 02:52]*

Source: Stallings page 136. It ain't hard.

When the cache is full, and we want to fill some blocks into the cache, we need the replacement algorithm to determine which blcok is about to be replaced.

We have three kinds of algorithms available.

1. **LRU**: least recently used
2. **LFU**: least frequently used
3. **FIFO**: first in first out

of course, each of them has some advantages.

4.7.1 WAITING More into Replacement Algorithm

- State "WAITING" from "TODO" *[2023-06-17 Sat 03:11]*
  till i wanna do it.

5 TODO Virtual Memory (from *Patterson*)

6 External Memory

6.1 DONE RAID from Stallings

- State "DONE" from "TODO" *[2023-06-17 Sat 17:09]*

Source: Stallings page 194

The key word is **redundancy**

`redundancy`

redundancy

REDUNDANCY
redundancy

6.1.1 RAID and some related Concepts

RAID is an external memory system coined by some researchers in UCB.

redundancy: It takes advantages of the **redundancy** to improve the performance.

Strip and Stripe: strip is the concept used to describe how the memory is divided. It tells you the basic the unit of the some strategies here. For example, in RAID 2, the strip is very small. and the very strip has a simple parity bit. For example, the strip is RAID 0, is like the block that we use (but the name is different, and the concepts are different after all). The memory of RAID 0 is striped across the available disks (leads to a high access speed). As for stripe, the word stripe is the verb form of strip.

From RAID 0 to RAID 6: The raid system integrate many types of the disk type. For different type of the memory operation (for example, an array of read operations or an array of write operations, and for example, access request from multiple I/O devices, and for example, large, continuous blocks access request). Different RAID can best-coped with certain situation.

The key of RAID (Idk): In the construction of different RAID, two things are important: **1. How Data is Distributed**; **2. How Redundancy is implemented**. The exact two keys determine the performance of the RAID and its application.

6.1.2 RAID Level 0

RAID 0 uses no redundancy. Technically, RAID 0 does not belong to RAID system. Anyway there it is.

RAID 0 is used for the **user and system** data. It is distributed over an array of disks.

> If there are two I/O requests, and the requested blocks are likely on different disks, then the two requests can be issued in parallel.

The quote turns out to be quite inaccurate. The implementation of RAID 0 can serve two purpose: **1. High Data Transfer Capacity**; **2. High I/O Request Rate**. The quote is telling about the second purpose. And the implementation is already been discussed one can check 3.6.3 for more information.

6.1.3 RAID Level 1

RAID 1 is a mirrored RAID 0. It implements redundancy by using a copy of itself, that is, there are two disks, and they carry the same data.

There is something we should notice:

1. Advantage: Higher Read Speed. The read operation can be runned uncondtionally parrallelly. So the read speed is twice as the RAID 0's.

2. Neither Advantage or Disadvantage: The write speed remains unchanged. Although we have to write data into two disks, the write operations is parrallel. So the speed is unchanged.

3. Advantage: Safety. Recovery from a failure is simple. "When a drive fails, the data may still be accessed from the second drive".

4. Disadvantage: Expensive.

Suit for:

1. for the data transfer intensive application with a high percentage of reads.

2. for system software and data and other highly critical files.

6.1.4 RAID Level 2

RAID 2 and RAID 3 use parity code. But the striping is very different.

The parity code is store in other strips. The strips are very small in RAID 2. Usually one byte or word.

Suit for: where many disk errors occur.

6.1.5 TODO RAID Level 3

RAID 3 use a simple parity bit for a byte. That is it usually XOR all the bits in a byte and store it right next to the byte.

And the parity bit and the byte compose a strip.

6.1.6 RAID Level 4

RAID 4 is like RAID 3 but the strip is different.

The strips are large.

6.1.7 RAID Level 5

RAID 5 is like RAID 4 but the striping is different. The parity block is interleaved so that parity block and data block can be accessed simultaneously.

6.1.8 RAID Level 6

RAID 6 is a revised version of RAID 5. It uses two kinds of parity code. And as the RAID 5, the parity block is interleaved.

6.2 Magnetic Disk & Optical Memory & Magnetic Tap