
计算机系统基础

许向阳

xuxy@hust.edu.cn

QQ: 3303945868

学习资源

QQ:



群名称: 计算机系统基础_HU...

群 号: 202914111

学习资源

微助教:



课堂名称: 计算机系统基础_HUST_CS_XU
课堂编号: IY884

课程基本信息



《计算机系统基础》，袁春风，机械工业出版社

课程基本信息

- **主要参考书:**

- 《x86汇编语言程序设计》，许向阳，华中科技大学出版社，2020年
- 《深入理解计算机系统》（第3版），Randal E. Bryant, david R. O' Hallaron著，龚奕利，贺莲 译，机械工业出版社，2016 年
- Brian W. Kernighan, Dennis M. Ritchie, The C Programming Language (second Edition), 北京：机械工业出版社，2006
- 《计算机系统概论》(原书第2版)，Yale N. Patt, Sanjay J. Patel著，梁阿磊,蒋兴昌,林凌译，机械工业出版社，2007年

课程基本信息

- 实验平台:

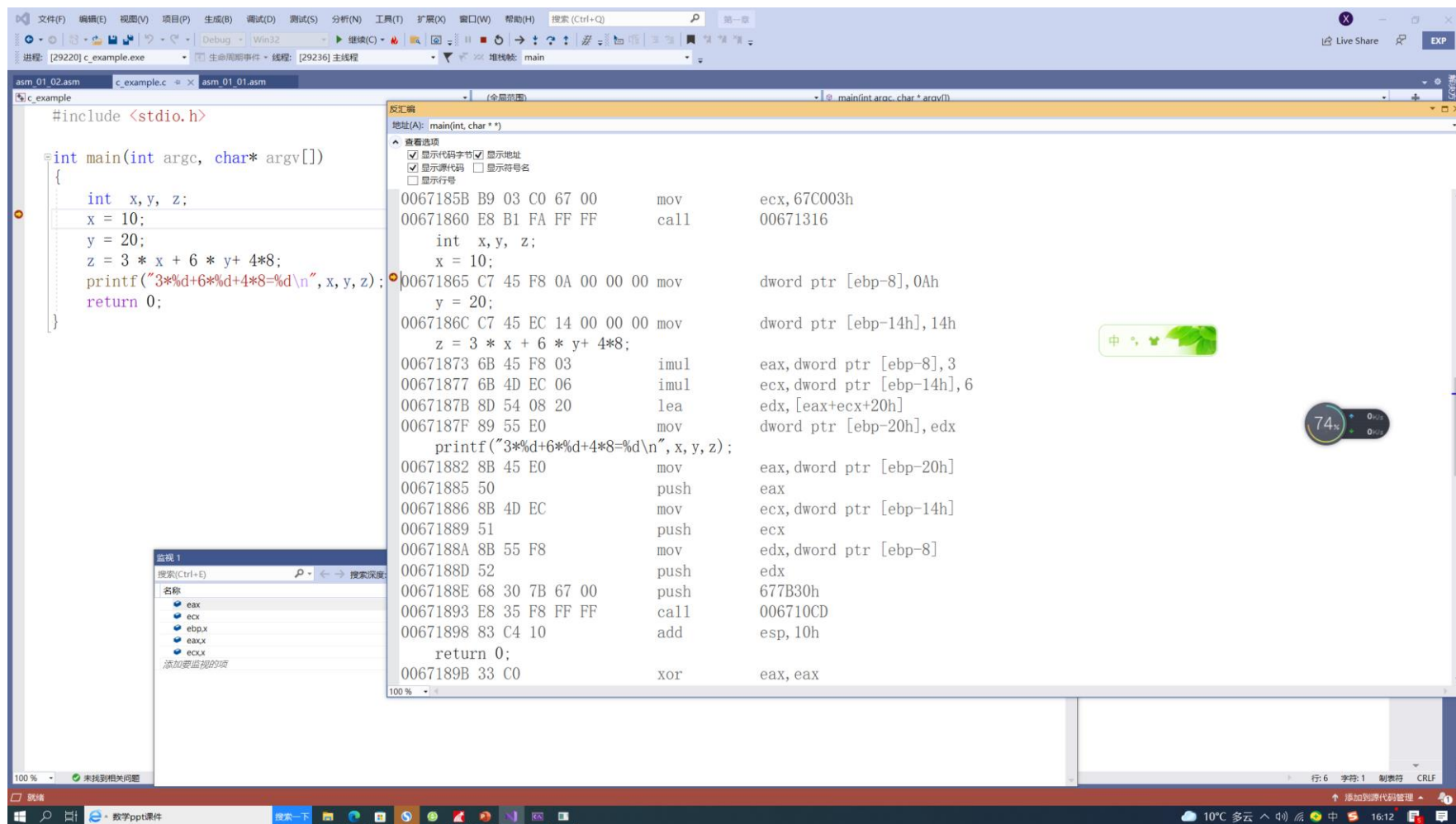
IA-32+Windows+C+VS2019

+Linux+C+gcc

+QEMU (ARMv8)

+IDA 反汇编调试工具 (OLLYDBG,W32Dasm 等)

课程基本信息



VS2019 社区版

许向阳.《x86汇编语言程序设计》，第 19 章

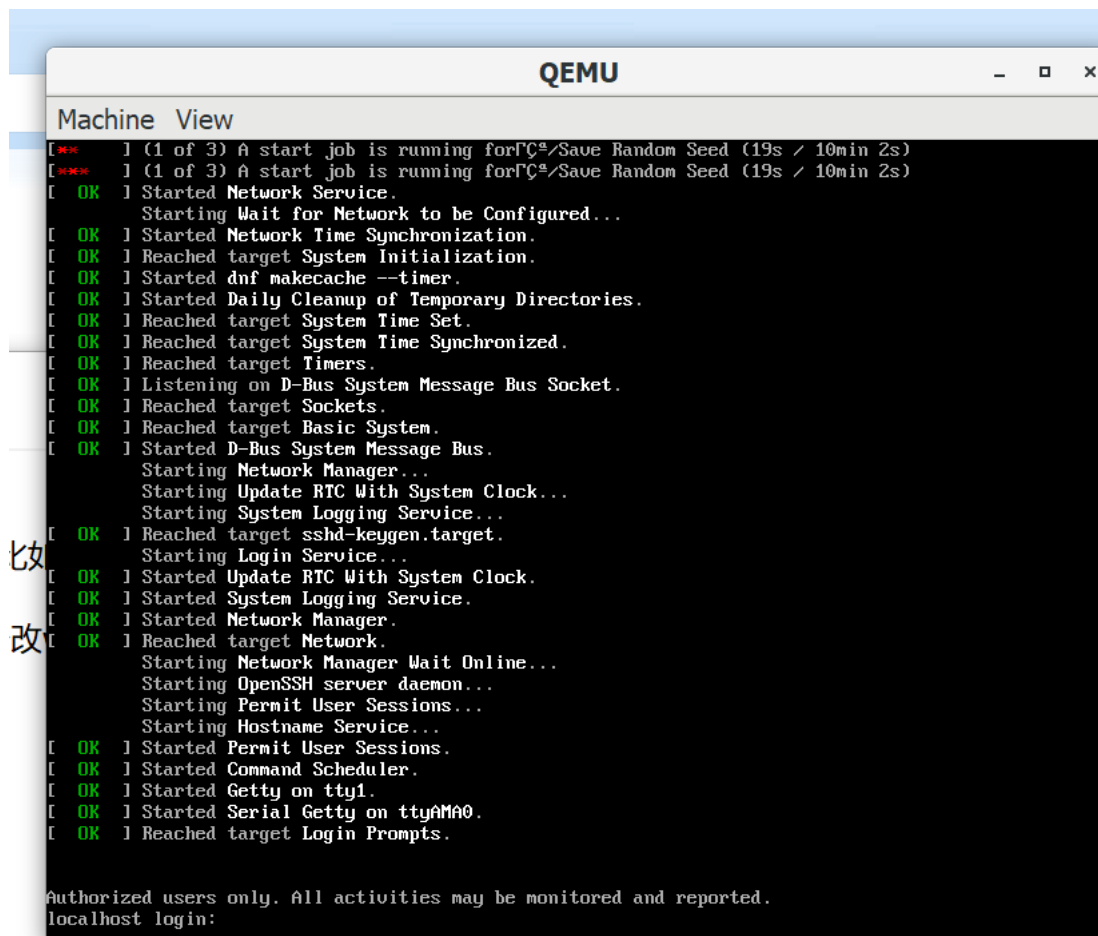
课程基本信息

```
root@LAPTOP-CJLSTBTI: /home
Microsoft Windows [版本 10.0.19043.1526]
(c) Microsoft Corporation。保留所有权利。

C:\WINDOWS\system32>bash
root@LAPTOP-CJLSTBTI:/mnt/c/WINDOWS/system32# cd /
root@LAPTOP-CJLSTBTI:/# ls
bin  boot  dev  etc  home  init  lib  lib64  media  mnt  opt  proc
root@LAPTOP-CJLSTBTI:/# cd home
root@LAPTOP-CJLSTBTI:/home# ls
test  test.asm  test.c  test.i  test.o  test.s  test_intel.s
root@LAPTOP-CJLSTBTI:/home# gcc test.c -o test
root@LAPTOP-CJLSTBTI:/home#
```

win10 下安装和配置Ubuntu 及gcc_gdb .docx

课程基本信息



```
Machine View
[***] (1 of 3) A start job is running forFC#/Save Random Seed (19s / 10min 2s)
[***] (1 of 3) A start job is running forFC#/Save Random Seed (19s / 10min 2s)
[ OK ] Started Network Service.
       Starting Wait for Network to be Configured...
[ OK ] Started Network Time Synchronization.
[ OK ] Reached target System Initialization.
[ OK ] Started dnf makecache --timer.
[ OK ] Started Daily Cleanup of Temporary Directories.
[ OK ] Reached target System Time Set.
[ OK ] Reached target System Time Synchronized.
[ OK ] Reached target Timers.
[ OK ] Listening on D-Bus System Message Bus Socket.
[ OK ] Reached target Sockets.
[ OK ] Reached target Basic System.
[ OK ] Started D-Bus System Message Bus.
       Starting Network Manager...
       Starting Update RTC With System Clock...
       Starting System Logging Service...
[ OK ] Reached target sshd-keygen.target.
       Starting Login Service...
[ OK ] Started Update RTC With System Clock.
[ OK ] Started System Logging Service.
[ OK ] Started Network Manager.
[ OK ] Reached target Network.
       Starting Network Manager Wait Online...
       Starting OpenSSH server daemon...
       Starting Permit User Sessions...
       Starting Hostname Service...
[ OK ] Started Permit User Sessions.
[ OK ] Started Command Scheduler.
[ OK ] Started Getty on tty1.
[ OK ] Started Serial Getty on ttyAMA0.
[ OK ] Reached target Login Prompts.

Authorized users only. All activities may be monitored and reported.
localhost login:
```

win10 下, QEMU
ARM模拟环境直接释放免安装包.rar

参考资源

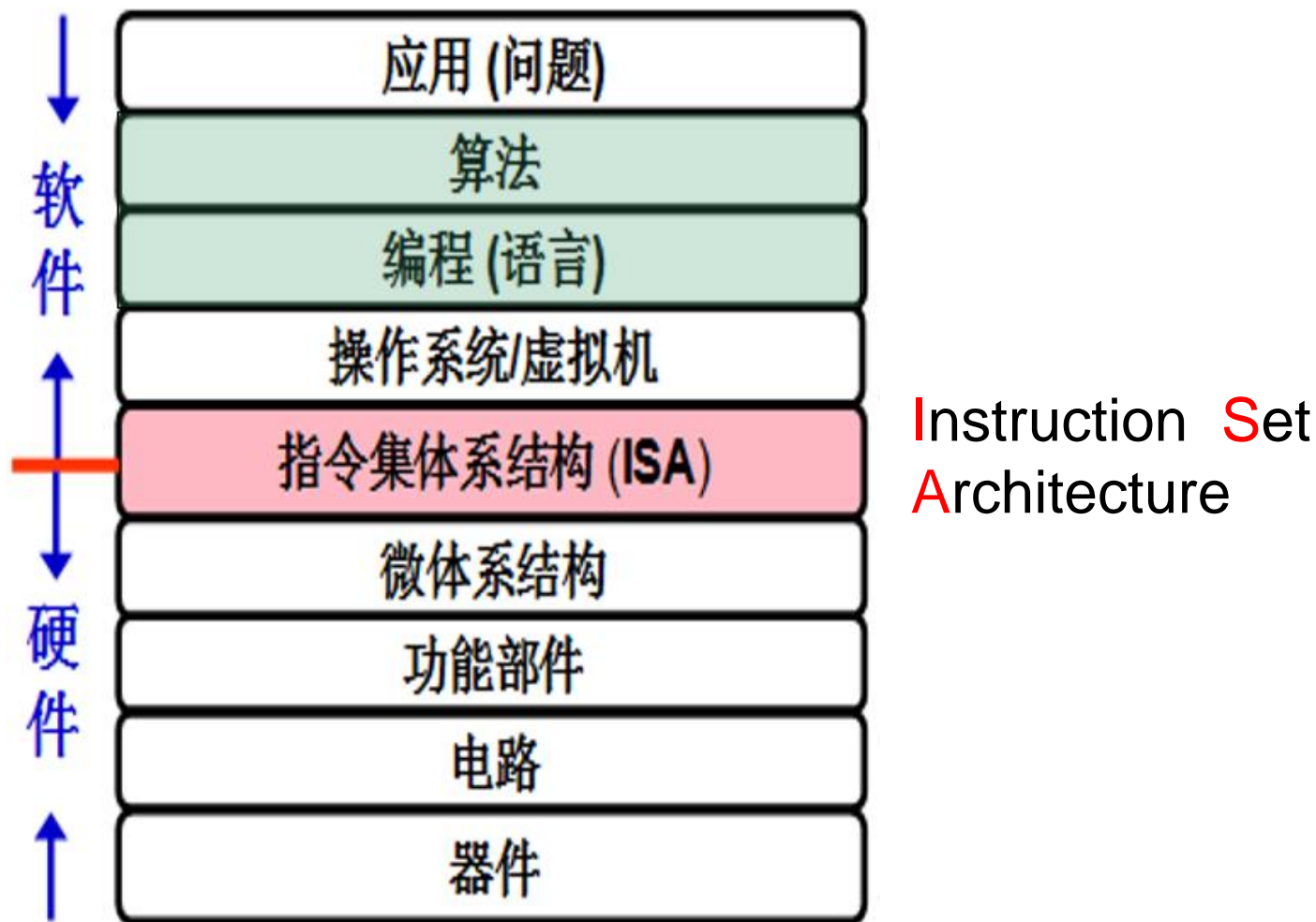
- MOOC网站（4门系列课程）

- <https://www.icourse163.org/course/NJU-1001625001>
程序的表示、转换与链接
- <https://www.icourse163.org/course/NJU-1001964032>
程序的执行和存储访问
- <https://www.icourse163.org/course/NJU-1002532004>
异常、中断和输入/输出
- <http://www.icourse163.org/course/NJU-1449521162>
编程与调试实践

课程参考网站（南京大学）

http://cslab.nju.edu.cn/ics/index.php/lcs:Main_page

什么是计算机系统？



什么是计算机系统？



计算机视觉、自然语言处理

计算机算法、数据结构

C、C++、JAVA、python

操作系统 编译原理

汇编语言

计算机组成原理

数字逻辑

模拟电子技术

什么是计算机系统？

2022全国大学生**计算机系统能力**大赛**操作系统设计**大赛

OS 功能挑战赛

赛题中很大一部分来自企业，甚至是企业目前正在寻求解决的真实技术问题。

OS 内核实现

硬件平台可以选用 K210开发板、HiFive Unmatched开发板。构思并实现一个综合性的操作系统，以展示参赛队面向特定目标平台的操作系统构造与优化能力。

什么是计算机系统？

2022全国大学生**计算机系统能力**大赛**编译系统设计**大赛

- 构思并实现一个综合性的编译系统，以展示面向特定目标平台的编译器构造与编译优化的能力。
- 开发支持特定语言、面向ARM平台的综合性编译系统。
- 能够将符合自定义程序设计语言SysY2022的测试程序，编译为ARM(ARMv7 32位)汇编语言程序。

什么是计算机系统？

2022全国大学生**计算机系统能力**大赛 **CPU设计** 大赛

- 开发支持MIPS基准指令集的MIPS微系统。
- 自行决定CPU微架构、演示方案等。
- 综合运用各种知识（流水线、超标量、预测、Cache等）
充分利用实验板硬件资源以尽可能提高CPU运行性能。
- 以FPGA运行基准测试程序所需时间为评价依据。

计算机系统基础——学什么？

- 表示 (Representation)
 - 数据 如何表示和存储？
(不同数据类型：带符号整数、无符号整数、浮点数、数组、结构等)
 - 指令 如何表示和编码 (译码) ？
 - 存储地址 (指针) 如何表示？
 - 如何生成复杂数据结构中数据元素的地址？
- 转换 (Translation) 和链接 (Link)
 - 高级语言程序对应的机器级代码是怎样的？如何合并成可执行文件？
- 执行控制流 (Control flow)
 - 计算机能理解的“程序”是如何组织和控制的？
 - 如何在计算机中组织多个程序的并发执行？
 - 逻辑控制流中的异常事件及其处理
 - I/O操作的执行控制流 (用户态→内核态)

为什么要学习“计算机系统基础”？

培养系统思维能力

Q：程序执行结果
取决于哪些因素？

算法
程序编写
语言处理系统
操作系统
ISA
微体系结构



为什么要学习“计算机系统基础”？

Q：什么是系统思维？

全局思维、动态思维、结构思维



站得更高，看得更远，看得更全！

为什么要学习“计算机系统基础”？

- 系统思维

- 从计算机系统角度出发分析问题和解决问题
- 首先取决于对计算机系统有多了解，“知其然并知其所以然”

基本认识

- 高级语言语句都要转换为机器指令才能在计算机上执行
- 机器指令是一串0/1序列，能被机器直接理解并执行
- 计算机系统是模运算系统，字长有限，高位被丢弃
- 运算器不知道参加运算的是有符号数还是无符号数
- 在计算机世界， $x \cdot x$ 可能小于0， $(x+y)+z$ 不一定等于 $x+(y+z)$
- 访问内存需几十到几百个时钟，而访问磁盘要几百万个时钟
- 进程具有独立的逻辑控制流和独立的地址空间
- 过程调用使用栈存放参数和局部变量等，递归过程有大量额外指令，增加时间开销，并可能发生栈溢出
-

只有先理解系统，才能驾驭系统、优化系统、研制国产系统！

为什么要学习“计算机系统基础”？

- 强化“系统思维”
- 更好地理解计算机系统，从而编写出更好的程序
- 编写程序时少出错
- 在程序出错时很快找到出错的地方
- 编写出运行更快的程序
- 明白程序是怎样在计算机上执行的
- 为后续课程的学习打下良好基础
-

深入理解计算机系统

培养系统性思维能力



东九楼 到 韵苑食堂 路段的交通变迁 的启示

深入理解计算机系统

求整型数组中len个元素的和

```
int sum_1(int a[ ], unsigned len)
{
    int i, sum = 0;
    for (i = 0; i <= len-1; i++)
        sum += a[i];
    return sum;
}
```

warning C4018: “<=” :
有符号/无符号不匹配

```
int main()
{
    int a[3]={10,20,30};
    int result;
    result=sum_1(a,0);
    printf(“%d”,result);
}
```

Q: 运行时会有什么?

深入理解计算机系统

```
result = sum_1(a, 0);
```

```
6 int sum_1(int a[], unsigned int len)
7 {
8     int i, sum = 0;
9     for (i = 0; i <= len-1; i++)
10         sum += a[i];
11     return sum;
12 }
```

已引发异常

引发了异常: 读取访问权限冲突。
a 是 0x18D4112。

[复制详细信息](#) | [启动 Live Share 会话...](#)

异常设置

☒ 引发此异常类型时中断

从以下位置引发时除外:

☐ 运算等价性.exe

[打开异常设置](#) | [编辑条件](#)

```
19     return sum;
20 }
```

监视 1		
搜索(Ctrl+E)		
名称	值	类型
a[0]	10	int
a[1]	20	int
a[2]	30	int
i	4492	int
sum	1519703788	int
len	0	unsigned int

Q: 什么是“读取访问权限冲突”？
为什么会有“读取访问权限冲突”？
“访问权限”的实现机理是什么？

参考: 教材第6章
层次结构存储系统
6.5.4 节 存储保护
6.6 节 地址转换

深入理解计算机系统

操作系统： 内存管理

计算机组成原理： 中断和异常

```
#include <iostream>
#include <Windows.h>
using namespace std;
int sum_1_exception(int a[], unsigned int len)
{
    int i, sum = 0;
    __try {
        for (i = 0; i <= len - 1; i++)
            sum += a[i];
    }
    __except (EXCEPTION_EXECUTE_HANDLER )
    {
        cout << "exception occur" << endl;
        sum = 0;
    }
    return sum;
}
```


深入理解计算机系统

```
int sum_1(int a[ ], unsigned len)
{
    int i, sum = 0;
    for (i = 0; i <= len-1; i++)
        sum += a[i];
    return sum;
}
```

```
int main()
{
    int a[3]={10,20,30};
    int result;
    // result=sum_1(a,0);
    result = sum_2(a,0);
    printf("%d",result);
}
```

```
int sum_2(int a[ ], unsigned len)
{
    int i, sum = 0;
    for (i = 0; i < len; i++)
        sum += a[i];
    return sum;
}
```

改为右边的形式后，运行正常。显示结果0

$i \leq len-1 \Leftrightarrow i < len ?$
 $x < y \Leftrightarrow x-y < 0 ?$

深入理解计算机系统

```
int sum_1(int a[ ], unsigned len)
{
    int i, sum = 0;
    for (i = 0; i <= len-1; i++)
        sum += a[i];
    return sum;
}
```

```
int sum_3(int a[ ], int len)
{
    int i, sum = 0;
    for (i = 0; i <= len-1; i++)
        sum += a[i];
    return sum;
}
```

```
int sum_2(int a[ ], unsigned len)
{
    int i, sum = 0;
    for (i = 0; i < len; i++)
        sum += a[i];
    return sum;
}
```

result=sum_3(a, 0); 运行正常

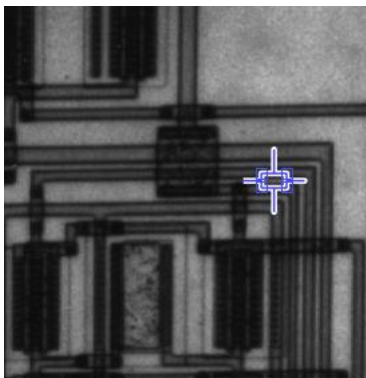
result=sum_3(a, 0x90000000);
返回的结果为0

result=sum_1(a, 0x90000000);
result=sum_2(a, 0x90000000);
运行异常

有符号数(int)，无符号数(unsigned int)

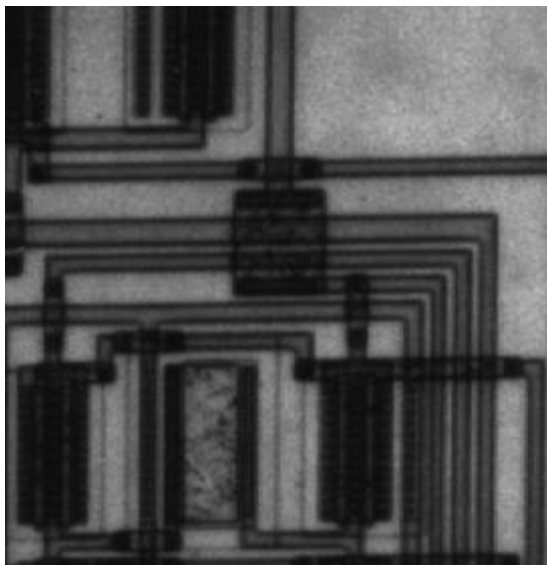
矩阵减法 性能研究

```
int a[2000][3000], b[2000][3000], c[2000][3000];  
c= b-a;
```

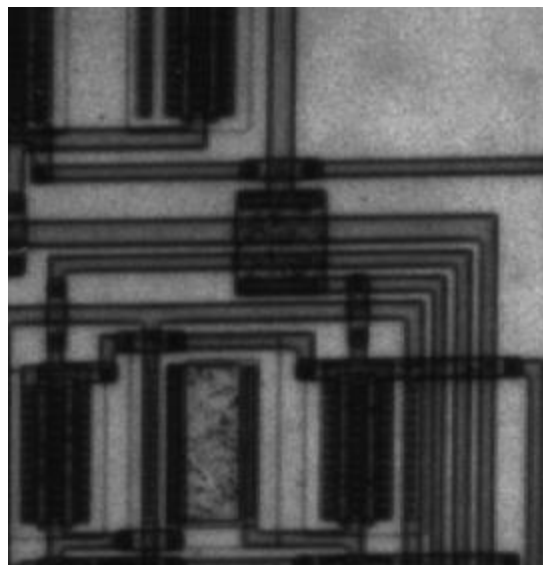


60	59	59	59	58	57	58	59	58	58	55	55	59	59	60	58	55	57	58
48	47	46	46	45	45	44	43	43	44	45	42	45	47	46	46	44	46	46
37	36	35	37	37	35	34	32	32	34	35	35	36	36	35	37	34	36	38
38	39	43	45	45	43	42	39	41	43	42	44	45	43	42	45	42	44	49
57	58	65	67	68	65	64	64	67	68	68	69	72	73	71	68	71	71	77
80	77	81	86	83	79	81	84	91	91	85	83	86	95	97	93	96	95	97
81	79	77	80	79	76	77	79	87	90	79	76	78	85	90	90	87	87	87
59	59	56	56	57	55	54	55	60	62	56	54	56	58	60	60	55	56	57
39	41	41	37	37	40	39	40	41	40	37	36	36	38	38	40	36	36	35
42	45	45	39	41	47	47	46	45	45	43	44	43	43	44	45	43	41	38
59	61	59	55	60	62	62	61	62	62	61	61	61	63	64	62	59	59	57

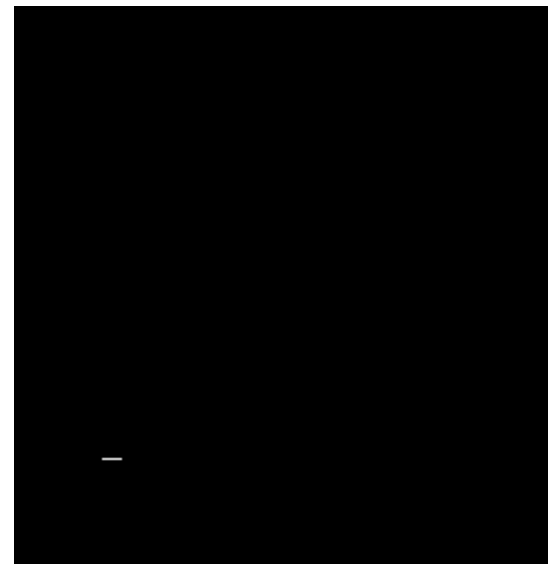
矩阵减法 性能研究



生成的电路板

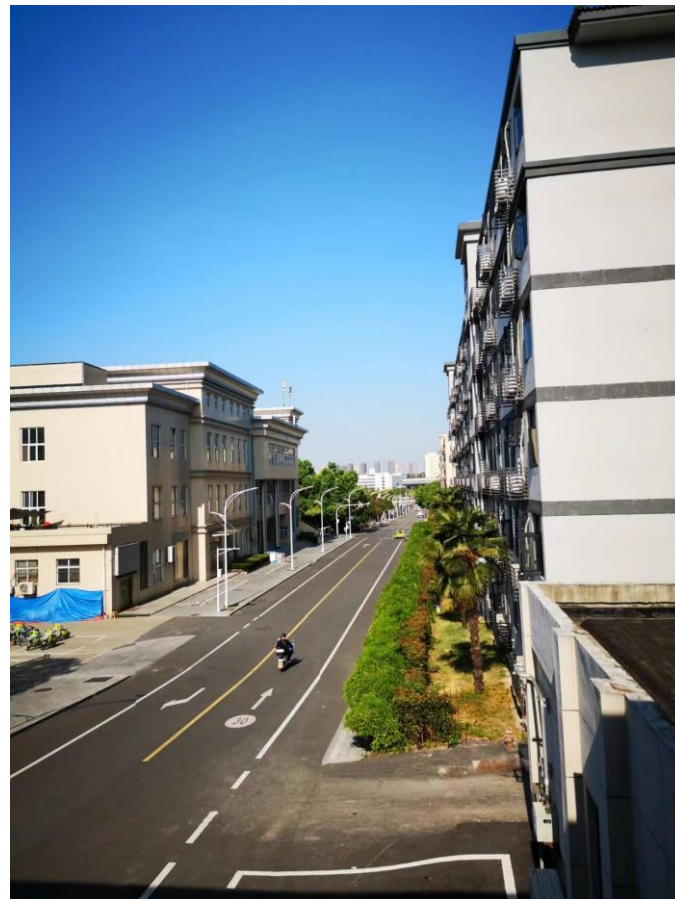


电路 模板



瑕疵 检测

矩阵减法 性能研究



R: 43	R: 43	R: 43	R: 43	R: 43	R: 43
G: 109	G: 109	G: 109	G: 109	G: 109	G: 109
B: 195	B: 195	B: 195	B: 195	B: 195	B: 195

R: 43	R: 43	R: 43	R: 43	R: 43	R: 43
G: 109	G: 109	G: 109	G: 109	G: 109	G: 109
B: 195	B: 195	B: 195	B: 195	B: 195	B: 195

R: 43	R: 43	R: 43	R: 43	R: 43	R: 43
G: 109	G: 109	G: 109	G: 109	G: 109	G: 109
B: 195	B: 195	B: 195	B: 195	B: 195	B: 195



矩阵减法 性能研究

```
void arraysubtract_rowsfirst()
{
    int i, j;
    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            c[i][j] = b[i][j] - a[i][j];
}
```

```
void arraysubtract_onedim()
{
    int i;
    int* pa,*pb,*pc;
    pa = &a[0][0];
    pb = &b[0][0];
    pc = &c[0][0];
    for (i = 0; i < M * N; i++)
        pc[i] = pb[i] - pa[i];
}
```

```
void arraysubtract_colsfirst()
{
    int i, j;
    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            c[i][j] = b[i][j] - a[i][j];
}
```

```
#define M 2000
#define N 3000
int a[M][N];
int b[M][N];
int c[M][N];
```

矩阵减法 性能研究

```
int main()
{
    int start, finish, duration, i;
    void (*funcp[3])() = { arraysubtract_colsfirst, arraysubtract_rowsfirst,
                           arraysubtract_onedim };
    char funcname[3][50] = {"arraysubtract_colsfirst ",
                           "arraysubtract_rowsfirst", "arraysubtract_onedim" };
    printf("please input : 0,1,2 :");
    scanf("%d", &i);
    start = GetTickCount(); // windows.h。得到系统运行的时间精确到毫秒
    funcp[i]();
    finish = GetTickCount();
    duration = finish - start;
    printf("%s 用时:  %d 毫秒\n", funcname[i], duration);
    system("pause");
    return 0;
}
```

函数指针、函数指针数组、函数入口地址表

矩阵减法 性能研究

C:\教学\本科教学\计算机系统基础\计算机系统基础_程序\C00_宏论计算机系统基础\Debug\数组减法比较.exe

```
please input : 0,1,2 :1  
arraysubtract_rowsfirst 用时: 63 毫秒  
请按任意键继续. . .
```

C:\教学\本科教学\计算机系统基础\计算机系统基础_程序\C00_宏论计算机系统基础\Debug\数组减法比较.exe

```
please input : 0,1,2 :0  
arraysubtract_colsfirst 用时: 171 毫秒  
请按任意键继续. . .
```

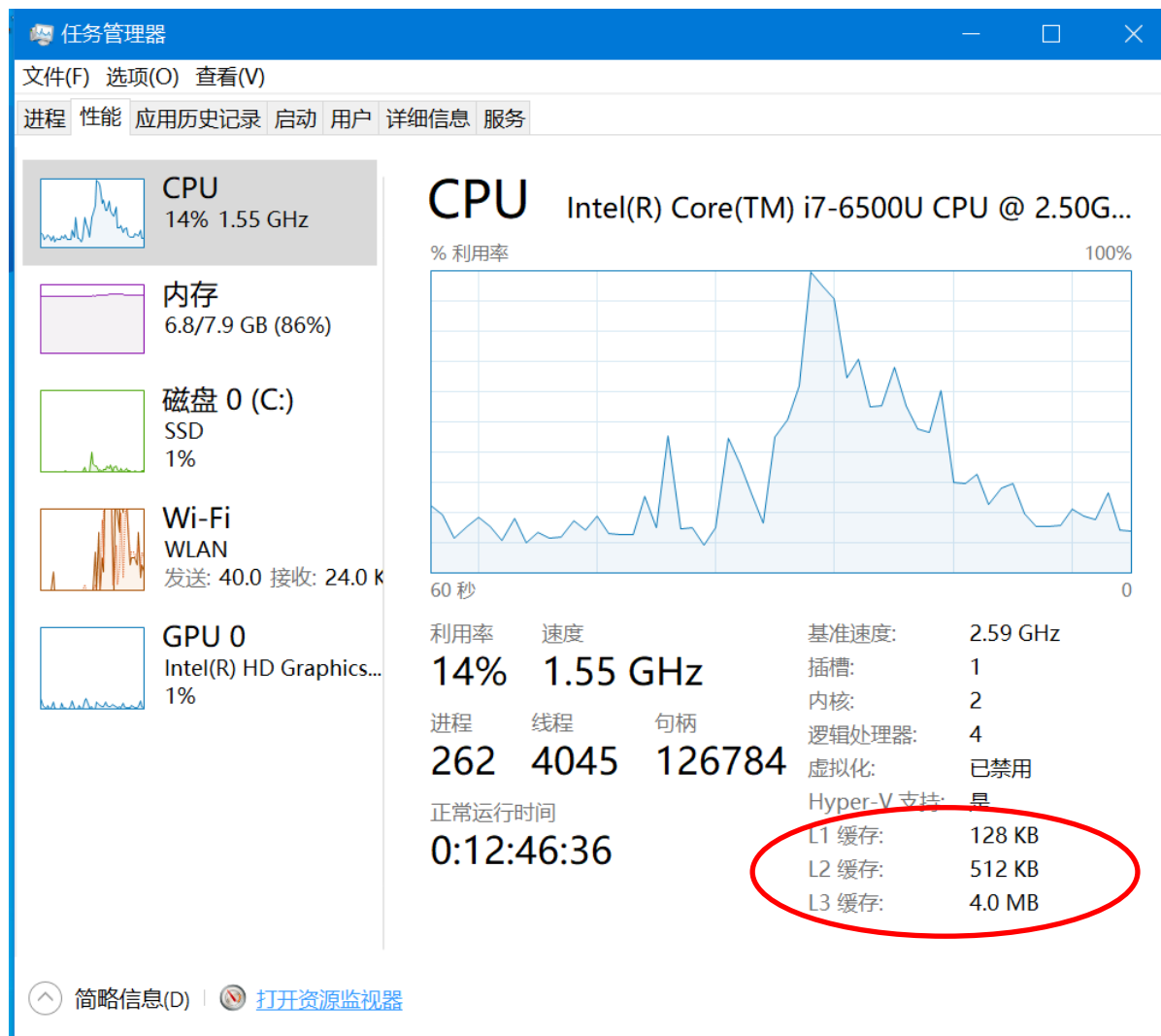
C:\教学\本科教学\计算机系统基础\计算机系统基础_程序\C00_宏论计算机系统基础\Debug\数组减法比较.exe

```
please input : 0,1,2 :2  
arraysubtract_onedim 用时: 78 毫秒  
请按任意键继续. . .
```

DEBUG 版本下，各函数的运行时间

矩阵减法 性能研究

Q: 为什么按行序优先执行快?



Q: 为什么有
Cache,
行序优先
执行快?

L1缓存: 128KB
L2缓存: 512KB
L3缓存: 4.0MB

矩阵减法 性能研究

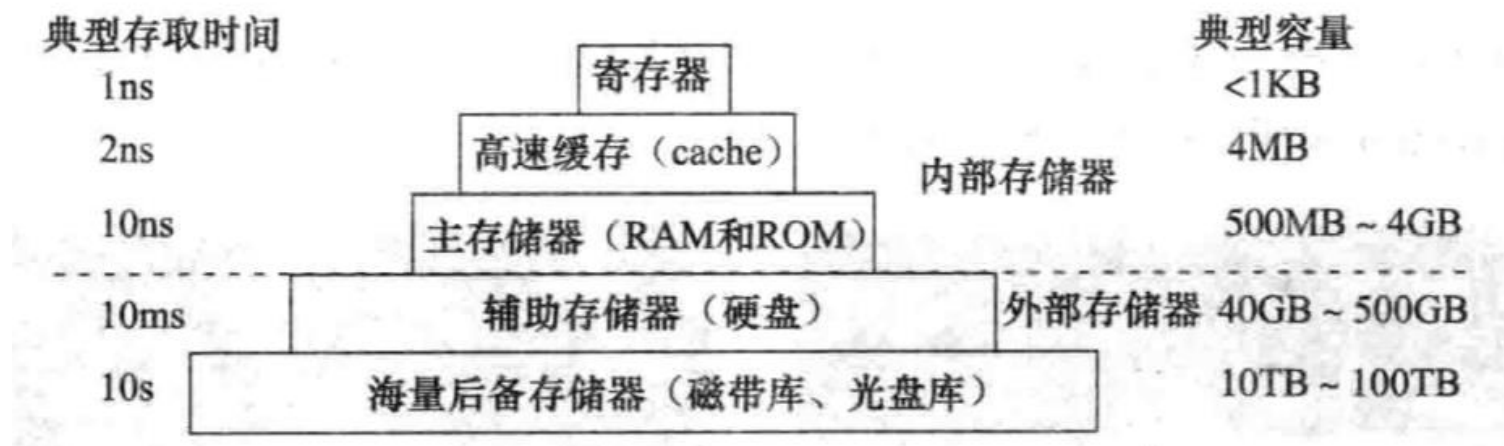
Q: Cache 是什么? 高速缓冲存储器 (SRAM)

Cache 的工作原理?

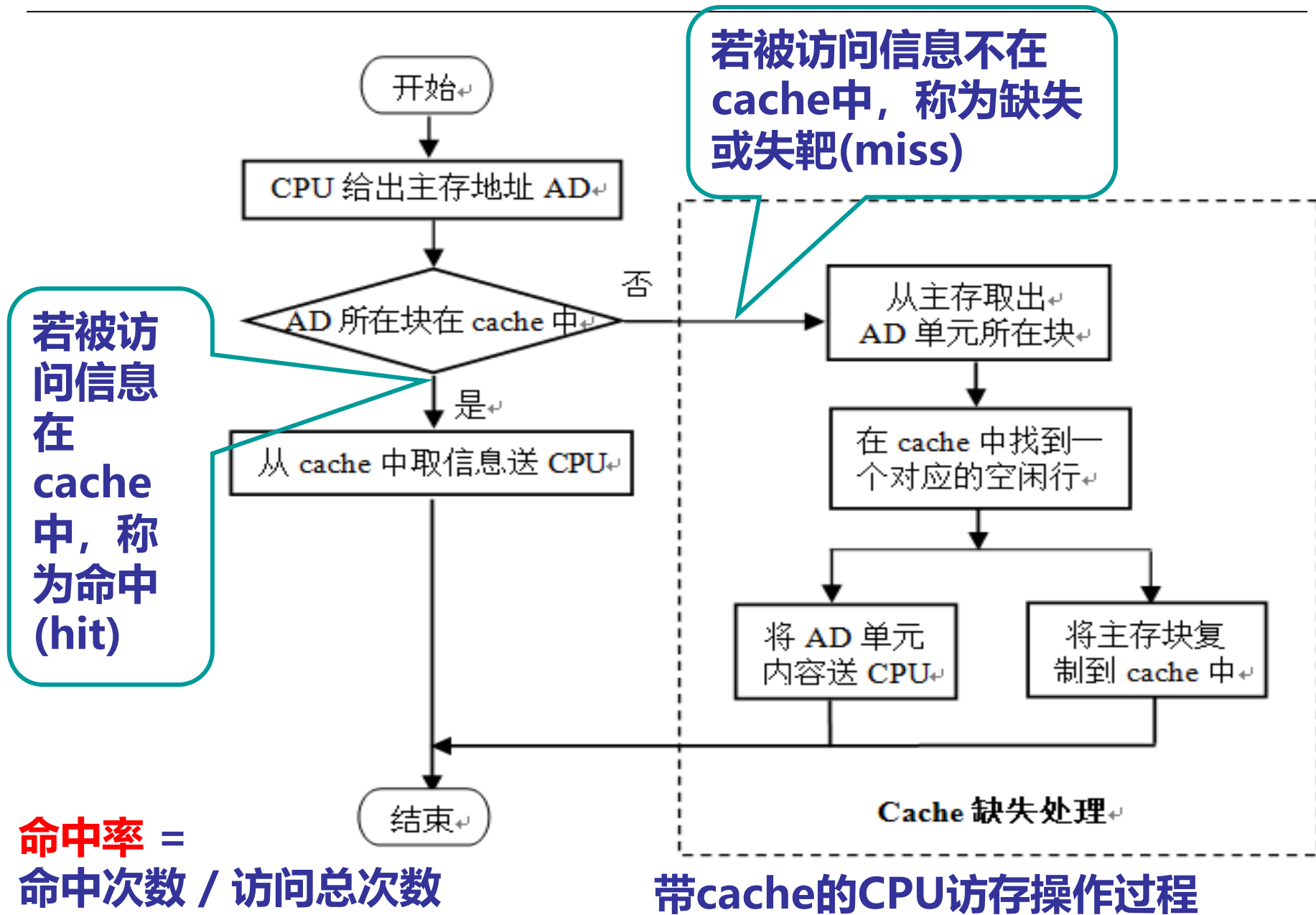
为什么 Cache 的访问速度快?

Cache 的发展历史?

参考: 教材第6章 层次结构存储系统
6.4 节 高速缓冲存储器

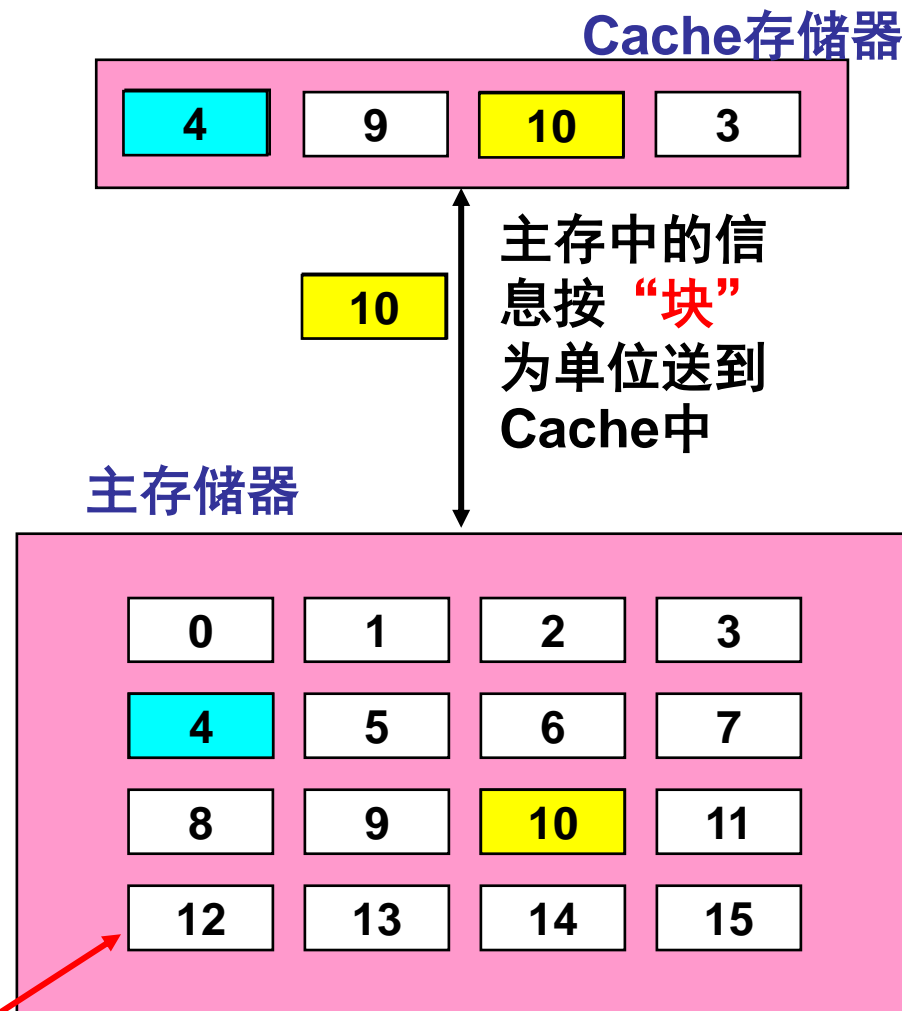


矩阵减法 性能研究



矩阵减法 性能研究

- Cache是一种小容量高速缓冲存储器，它由SRAM组成。
- Cache直接制作在CPU芯片内，速度几乎与CPU一样快。
- 程序运行时，CPU使用的一部分数据/指令会预先成批拷贝在Cache中，Cache的内容是主存储器中部分内容的副本。
- 当CPU需要从内存读(写)数据或指令时，先检查Cache，若有，就直接从Cache中读取，而不用访问主存储器。



矩阵减法 性能研究

问题：要实现Cache机制需要解决哪些问题？

如何分块？

主存块和Cache之间如何映射？

Cache已满时，怎么办？

写数据时怎样保证Cache和MM的一致性？

如何根据主存地址访问到cache中的数据？

Cache对程序员(编译器)是否透明？为什么？

矩阵减法 性能研究

Q: 如何充分利用 cache?

Cache 命中率

Cache 抖动

挖掘程序访问的局部性:

时间局部性、空间局部性

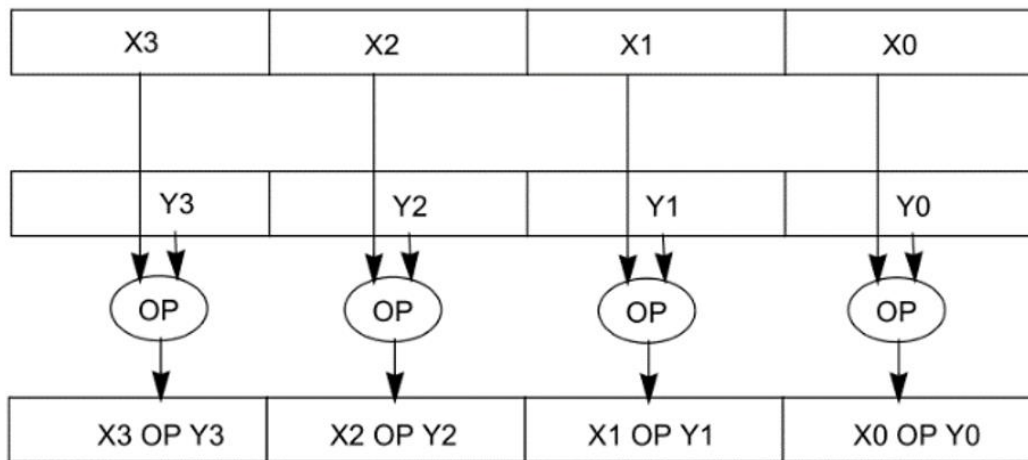
类比: 内存命中率

内存抖动

矩阵减法 性能研究

CPU 支持 SIMD ， 能否用 多媒体成组运算指令提高性能？

Single Instruction Multiple Data, 单指令多数据流



	10 (000AH)	<u>20</u> (0014H)	<u>30</u> (001EH)	40 (0028H)
+	25 (0019H)	-1 (0FFFFH)	-35 (0FFDDH)	35 (0023H)
	35 (0023H)	<u>19</u> (0013H)	-5 (0FFFBH)	75 (004BH)

矩阵减法 性能研究

Single Instruction Multiple Data, 单指令多数据流

```
#include <emmintrin.h>    //128 位
void arraysubtract_simd_128()
{
    __m128i *pa , *pb, *pc;
    int LEN4;
    pa = (__m128i*) & a[0][0];
    pb = (__m128i*) & b[0][0];
    pc = (__m128i*) & c[0][0];
    LEN4 = M * N / 4;
    for (int i = 0; i < LEN4; i++) {
        *pc = _mm_sub_epi32((__m128i)*pb, (__m128i)*pa);
        pa += 1;      // 一次 4个成对的整型数相减
        pb += 1;
        pc += 1;
    }
}
```

arraysubtract_simd_128 用时: 32 毫秒

矩阵减法 性能研究

Single Instruction Multiple Data, 单指令多数据流

```
#include <immintrin.h>    //256 位
void arraysubtract_simd_256()
{
    __m256i *pa, *pb, *pc;
    int LEN8;
    pa = (__m256i*) & a[0][0];
    pb = (__m256i*) & b[0][0];
    pc = (__m256i*) & c[0][0];
    LEN8 = M * N / 8;
    for (int i = 0; i < LEN8; i++) {
        *pc = _mm256_sub_epi32((__m256i)*pb, (__m256i) * pa);
        pa += 1;    // 一次 8个成对的整型数相减
        pb += 1;
        pc += 1;
    }
}
```

arraysubtract_simd_256 用时: 16 毫秒

矩阵减法 性能研究

CPU 支持 多线程，能否用多线程提高性能？

```
#include <iostream>
#include <thread>
using namespace std;
#define M 2000
#define QM 250 // 2000 / 8

void arraysubtract_rowsfirst(int no)
{
    int i, j;
    for (i = QM*no; i < QM*(no+1); i++)
        for (j = 0; j < N; j++)
            c[i][j] = b[i][j] - a[i][j];
    cout << " thread over " << no << endl;
}
```

矩阵减法 性能研究

CPU 支持 多线程，用多线程提高性能

```
int start, finish, duration;
thread subf[8];
init();
start = GetTickCount();

for (int i = 0; i < 8; i++)
    subf[i] = thread(arraysubtract_rowsfirst, i);

for (int i = 0; i < 8; i++)
    subf[i].join();

finish = GetTickCount();

duration = finish - start;
printf("用时:  %d  毫秒\n",  duration);
```

矩阵减法 性能研究

CPU 支持 多线程，用多线程提高性能

```
thread over 4 thread over 0  
thread over 2  
thread over 3  
thread over 1  
  
thread over 6  
thread over 7  
thread over 5  
用时： 16 毫秒
```

Q: 为什么用多线程，能够提高性能？
是不是线程越多越好？

矩阵减法 性能研究

Q: RELEASE 版本下，各函数的运行时间有何变化？

C:\教学\本科教学\计算机系统基础\计算机系统基础_程序\C00_宏论计算机系统基础\Release\数组减法比较.exe

```
please input : 0,1,2 :1  
arraysubtract_rowsfirst 用时: 47 毫秒  
请按任意键继续. . .
```

63->47

C:\教学\本科教学\计算机系统基础\计算机系统基础_程序\C00_宏论计算机系统基础\Release\数组减法比较.exe

```
please input : 0,1,2 :0  
arraysubtract_colsfirst 用时: 46 毫秒  
请按任意键继续. . .
```

171->46

C:\教学\本科教学\计算机系统基础\计算机系统基础_程序\C00_宏论计算机系统基础\Release\数组减法比较.exe

```
please input : 0,1,2 :2  
arraysubtract_onedim 用时: 47 毫秒  
请按任意键继续. . .
```

78->47

矩阵减法 性能研究

```
movups xmm0, XMMWORD PTR ?a@@@3PAY0HFDA@HA[eax]
movups xmm1, XMMWORD PTR ?b@@@3PAY0HFDA@HA[eax]
psubd  xmm1, xmm0
movups xmm0, XMMWORD PTR ?a@@@3PAY0HFDA@HA[eax+16]
movups XMMWORD PTR ?c@@@3PAY0HFDA@HA[eax], xmm1
movups xmm1, XMMWORD PTR ?b@@@3PAY0HFDA@HA[eax+16]
psubd  xmm1, xmm0
movups xmm0, XMMWORD PTR ?a@@@3PAY0HFDA@HA[eax+32]
movups XMMWORD PTR ?c@@@3PAY0HFDA@HA[eax+16], xmm1
movups xmm1, XMMWORD PTR ?b@@@3PAY0HFDA@HA[eax+32]
psubd  xmm1, xmm0
movups xmm0, XMMWORD PTR ?a@@@3PAY0HFDA@HA[eax+48]
movups XMMWORD PTR ?c@@@3PAY0HFDA@HA[eax+32], xmm1
movups xmm1, XMMWORD PTR ?b@@@3PAY0HFDA@HA[eax+48]
psubd  xmm1, xmm0
movups XMMWORD PTR ?c@@@3PAY0HFDA@HA[eax+48], xmm1
add     eax, 64 ; 00000040H
sub     ecx, 1
jne     SHORT $LL7@arraysubtr
```

优化：按行序执行；双重循环展开为单循环；使用 SIMD 指令

深入理解计算机系统

仅仅依靠编译器的优化，能否包打天下？

Release 版是否一定比**Debug** 版快？

Release 版是否一定能优化得很好？

深入理解计算机系统

一个字符串中小写字母变成大写字母

```
void change()  
{  
    int i;  
    int start, finish, duration;  
    start = GetTickCount();  
    for (i = 0; i < strlen(block); i++) {  
        if (block[i] >= 'a' && block[i] <= 'z')  
            block[i] = block[i] - 'a' + 'A';  
    }  
    finish = GetTickCount();  
    duration = finish - start;  
    cout << "time used: " << duration << endl;  
}
```

```
#define SIZE 4096*50  
char block[SIZE];  
  
block[SIZE-10]=0;
```


深入理解计算机系统

C:\教学\本科教学\计算机系统基础\计算机系统基础_程序\C00_宏论计算机系统基础\Debug\未能优化.exe

```
time used: 7781  
请按任意键继续. . .
```

C:\教学\本科教学\计算机系统基础\计算机系统基础_程序\C00_宏论计算机系统基础\Release\未能优化.exe

```
time used: 19765  
请按任意键继续. . .
```

Microsoft Visual Studio 调试控制台

```
time used: 0  
请按任意键继续. . .
```

Debug : 7781 毫秒
Release : 19765 毫秒

人工优化: 0 毫秒

深入理解计算机系统

```
void change()
```

```
{  
    int i;  
    int start, finish, duration;  
    start = GetTickCount();  
    for (i = 0; i < strlen(block); i++) {  
        if (block[i] >= 'a' && block[i] <= 'z')  
            block[i] = block[i] - 'a' + 'A';  
    }  
    finish = GetTickCount();  
    duration = finish - start;  
    cout << "time used: " << duration << endl;  
}
```

如何优化?

```
    int len = strlen(block);  
    for (i = 0; i < len; i++) { .....
```

测验

```
short x = 0x7fff;  
unsigned short y = 0x7fff;  
cout << "x=" << x << " y=" << y << endl;  
x++;    y++;  
cout << "x=" << x << " y=" << y << endl;  
x++;    y++;  
cout << "x=" << x << " y=" << y << endl;  
cout << "x=" << (unsigned short)x << " y=" << (short)y << endl;
```

Microsoft Visual Studio 调试控制台

```
x=32767 y=32767  
x=-32768 y=32768  
x=-32767 y=32769  
x=32769 y=-32767
```

运行结果是什么？

内存 1

地址: 0x004FFC10 列: 自动

0x004FFC10	01 80 cc cc cc cc cc cc cc cc cc cc cc	. € ??????????
0x004FFC1C	01 80 cc cc cc cc cc cc 44 fc 4f 00	. € ??????D?0.
0x004FFC28	33 34 15 00 01 00 00 00 c0 54 77 00	34.....?Tw.
0x004FFC34	50 7b 77 00 01 00 00 00 c0 54 77 00	P{w.....?Tw

监视 1

搜索(Ctrl+E) 搜索深度: 3

名称	值
&y	0x004ffc10 {32769}
&x	0x004ffc1c {-32767}

测验

若x和y为int型，当 $x=65535$ 时， $y=x*x$ ；y的值为多少？

$y=-131071$ 。Why?

现实世界中， $x^2 \geq 0$ ，但在计算机世界并不一定成立。

对于任何int型变量x和y， $(x > y) == (-x < -y)$ 总成立吗？

当 $x=-2147483648$ ，y任意（除-2147483648外）时不成立

Why?

在现实世界中成立，

但在计算机世界中并不一定成立。

理解该问题需要知道：

机器级数据的表示

机器指令的执行

计算机内部的运算电路

测验

main.c

```
int d=100;
int x=200;
int main()
{
    p1( );
    printf ( "d=%d, x=%d\n" , d, x );
    return 0;
}
```

p1.c

```
double d;

void p1( )
{
    d=1.0;
}
```

打印结果是什么?

d=0, x=1 072 693 248

Why?

理解该问题需要知道:
机器级数据的表示
变量的存储空间分配
数据的大端/小端存储方式
链接器的符号解析规则

.....

测验

以下是一段C语言代码：

```
#include <stdio.h>
```

```
main()
```

```
{
```

```
    double a = 10;
```

```
    printf("a = %d\n", a);
```

```
}
```

理解该问题需要知道：

IEEE 754 的表示

X87 FPU的体系结构

IA-32和x86-64中过程
调用的参数传递

计算机内部的运算电路

.....

在IA-32上运行时，打印结果为a=0

在x86-64上运行时，打印出来的a是一个不确定值

为什么？

测验

```
double fun(int i)
{
    volatile double d[1] = {3.14};
    volatile long int a[2];
    a[i] = 1073741824; /* Possibly out of bounds */
    return d[0];
}
```

i=0~4时, fun(i)分别返回什么值?

fun(0) → 3.14
fun(1) → 3.14
fun(2) → 3.1399998664856
fun(3) → 2.00000061035156
fun(4) → 3.14, 然后存储保护错

Why?

理解该问题需要知道:

机器级数据的表示

过程调用机制


栈帧中数据的布局

.....

测验

```
void copyij (int src[2048][2048],
             int dst[2048][2048])
{
    int i,j;
    for (i = 0; i < 2048; i++)
        for (j = 0; j < 2048; j++)
            dst[i][j] = src[i][j];
}
```

```
void copyji (int src[2048][2048],
             int dst[2048][2048])
{
    int i,j;
    for (j = 0; j < 2048; j++)
        for (i = 0; i < 2048; i++)
            dst[i][j] = src[i][j];
}
```



以上两个程序功能完全一样，算法完全一样，因此，时间和空间复杂度完全一样，执行时间一样吗？

理解该问题需要知道：

数组的存放方式

Cache机制

访问局部性

.....

测验

C/C++ code



```
1  #include "stdafx.h"
2  int main(int argc, char* argv[])
3  {
4      int a=10;
5      double *p=(double*)&a;
6      printf("%f\n", *p);
7      printf("%f\n", (double(a)));
8
9      return 0;
10 }
```

11 为什么printf("%f", *p)和printf("%f", (double)a)结果不一样呢？

理解该问题需要知道：

数据的表示

编译（程序的转换）

局部变量在栈中的位置

.....

//结果为0.000000

//结果为10.000000

不都是强制类型转换吗？怎么会不一样

关键差别在于一条指令：

fldl 和 **fildl**

总结

理解程序的执行结果要从系统层面考虑！

学完“计算机系统基础”就会对计算机系统有清晰的认识，
以后再学其他相关课程就容易多了。

把许多概念和知识联系起来就是李国杰院士所提出的“系统思维”。
即：站在“计算机系统”的角度考虑问题！