Assembly Language and Microcomputer Interface

Introduction

Ming Cai

cm@zju.edu.cn

College of Computer Science and Technology
Zhejiang University

Course Information

- **Text book**
  - Barry B. Brey 《INTEL Microprocessors》 8th Edition
- **QQ group: 891509207 (rename to "ID + name")**
- **TA: Jixin Yu 617139596@qq.com**

Course Resources

References

- Intel® 64 and IA-32 Architectures Software Developer Manuals:
  https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html
- AMD64 Architecture Programmer's Manual Volumes 1-5:
  https://www.amd.com/system/files/TechDocs/40332.pdf
- Irvine K R. Assembly language for X86 processors[M]. 2015.
- Kusswurm D. Modern X86 Assembly Language Programming[M]. Apress, 2018.
- Jo Van Hoey. Beginning X64 Assembly Programming[M]. Apress, 2019.
- Intel Intrinsics Guide: https://software.intel.com/sites/landingpage/IntrinsicsGuide/

Recommended videos

- 汇编语言程序设计(华中)：www.bilibili.com/video/BV1Nt411V7fa?p=27&t=138
- 微机原理与接口技术 (西交大)：www.bilibili.com/video/BV1DA411t7NN?p=1
- 汇编语言程序设计(清华)：www.bilibili.com/video/BV1G7411Z7VP?p=1

Course Resources

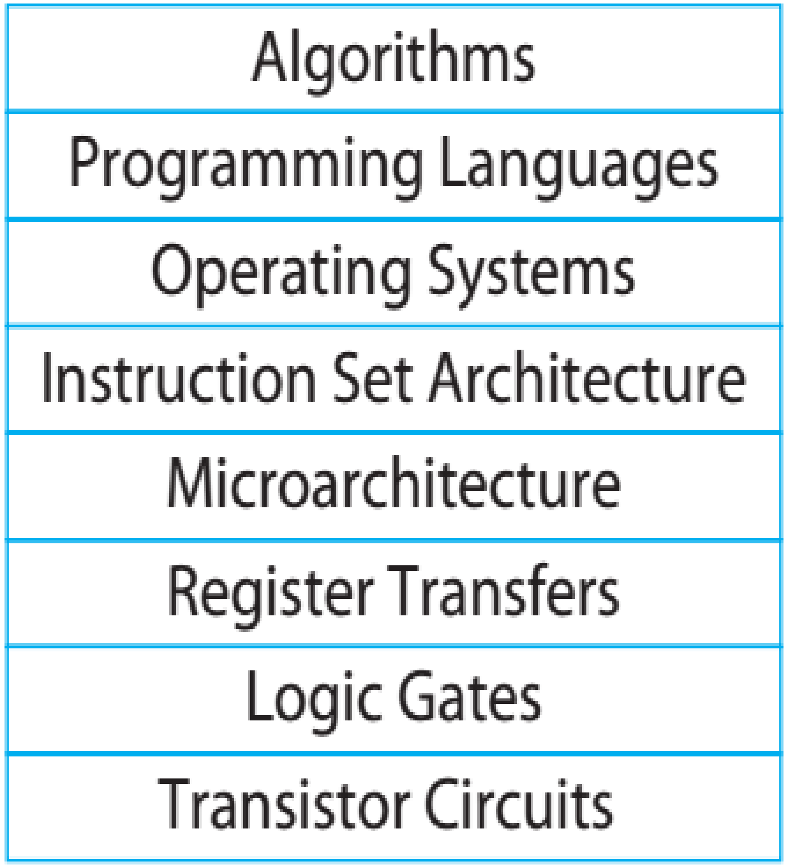
- **Microsoft Macro Assembler (MASM) references**
  - Directives Reference: https://docs.microsoft.com/en-us/cpp/assembler/masm/directives-reference?view=msvc-160
  - Symbols reference: https://www.amd.com/system/files/TechDocs/40332.pdf
  - Operators reference: https://docs.microsoft.com/en-us/cpp/assembler/masm/operators-reference?view=msvc-160
- **Useful software links**
  - EMU8086 - Microprocessor Emulator: https://emu8086-microprocessor-emulator.en.softonic.com
  - Godbolt compiler explorer: https://www.godbolt.org

    https://www.bilibili.com/video/BV1Uv4y1Z7pe?from=search&seid=8220486653503703739
  - x86 Processor information: www.sandpile.org

Assembler Directives

- Assembler directives are special instructions that provide information to the assembler but do not generate any code. Examples include the LABEL, EQU, ASSUME, and PROC. These mnemonics are not valid 80x86 instructions. They are messages to the assembler, nothing else.
- Three types of assembler directives in MASM:
  - Directives: .CODE, .DATA, .386, .586, LABEL, ORG, BYTE(DB), WORD(DW), PROC, STRUCT, IF, ELSE, etc.
  - Symbols: ?, @data, @code, @stack, $, @FileName, etc.
  - Operators: DUP, ‘’, ;, SEG, OFFSET, ==, >, !=, &&, ZERO?, SIGN?, CARRY?, OVERFLOW?, etc.

Abstraction Layers in Computer Systems



Algorithms
Programming Languages
Operating Systems
Instruction Set Architecture
Microarchitecture
Register Transfers
Logic Gates
Transistor Circuits

- **greedy, heuristic, LP, DP**
- **C/C++, Java, Python**
- **Linux, Windows, Android**
- **X86, ARM, RISC-V**
- **Pipeline, OOE, Multiprocessing**
- **Register, Datapath, Control Unit**
- **AND, OR, NOT, NAND, NOR**
- **BJT, JFET, IGFET**

Reasons to learn assembly language (1/4)

- Whether you should learn assembly language depends on what your goals are. For most developers, the answer is "no."
- Only a relative handful of the world's engineers and computer scientists actually use assembly language. Some specific areas where assembly language gets used are:
  - Operating systems
  - Firmware
  - Device drivers
  - Compiler design
  - Embedded systems
  - Hardware design
  - Advanced cryptography
  - Theoretical computer science

Reasons to learn assembly language (2/4)

- To gain a better understanding of how a compiler works.

```
int Sum1ToN(int n) {
        int sum = 0;
        for (int i = 0; i < n; i++) {
          sum += i;
        }
        return sum;
}
```

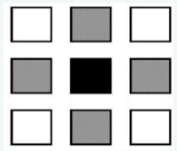
Adding the numbers from 1 to n-1

$$sum = \frac{(1 + n - 1) * (n - 1)}{2} = \frac{n * (n - 1)}{2}$$

- using compiler explorer (https://www.godbolt.org)
- using x86-64 gcc with different optimization options (-O1, -O3)
- using x86-64 clang with optimization option (-O3)
- using x86-64 icc with optimization option (-O3)

Reasons to learn assembly language (3/4)

- Assembly language enables us to exploit the single-instruction multiple-data (SIMD) capabilities of a processor.



```
void blur(const Image &in, Image &blurred) {
 Image tmp(in.width(), in.height());

 for (int y = 0; y < in.height(); y++)
  for (int x = 0; x < in.width(); x++)
   tmp(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;

 for (int y = 0; y < in.height(); y++)
  for (int x = 0; x < in.width(); x++)
   blurred(x, y) = (tmp(x, y-1) + tmp(x, y) + tmp(x, y+1))/3;
}
```

Clean C++ : 9.94 ms per megapixel

multithreading

tiling

vectorization

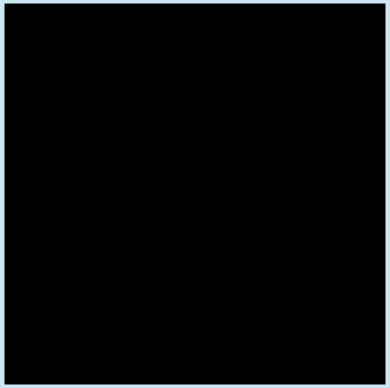
```
void fast_blur(const Image &in, Image &blurred) {
 __m128i one_third = _mm_set1_epi16(21846);
 #pragma omp parallel for
 for (int yTile = 0; yTile < in.height(); yTile += 32) {
  __m128i a, b, c, sum, avg;
  __m128i tmp[(256/8)*(32+2)];
  for (int xTile = 0; xTile < in.width(); xTile += 256) {
   __m128i *tmpPtr = tmp;
   for (int y = -1; y < 32+1; y++) {
    const uint16_t *inPtr = &(in(xTile, yTile+y));
    for (int x = 0; x < 256; x += 8) {
     a = _mm_loadu_si128((__m128i*)(inPtr-1));
     b = _mm_loadu_si128((__m128i*)(inPtr+1));
     c = _mm_load_si128((__m128i*)(inPtr));
     sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
     avg = _mm_mulhi_epi16(sum, one_third);
     _mm_store_si128(tmpPtr++, avg);
     inPtr += 8;
   }}
   tmpPtr = tmp;
   for (int y = 0; y < 32; y++) {
    __m128i *outPtr = (__m128i *)(&(blurred(xTile, yTile+y)));
    for (int x = 0; x < 256; x += 8) {
     a = _mm_load_si128(tmpPtr+(2*256)/8);
     b = _mm_load_si128(tmpPtr+256/8);
     c = _mm_load_si128(tmpPtr++);
     sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
     avg = _mm_mulhi_epi16(sum, one_third);
     _mm_store_si128(outPtr++, avg);
}}}}}
```

Fast C++ (for x86) : 0.90 ms per megapixel

Reasons to learn assembly language (4/4)

- Assembly language enables us to exploit the single-instruction multiple-data (SIMD) capabilities of a processor.

Clean C++

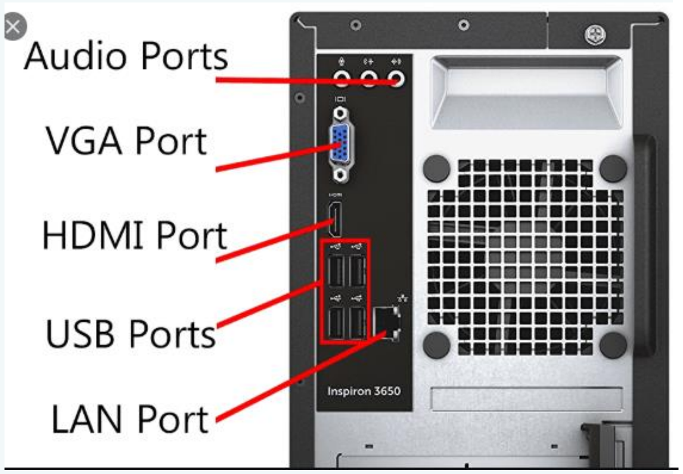


multithreading & tiling

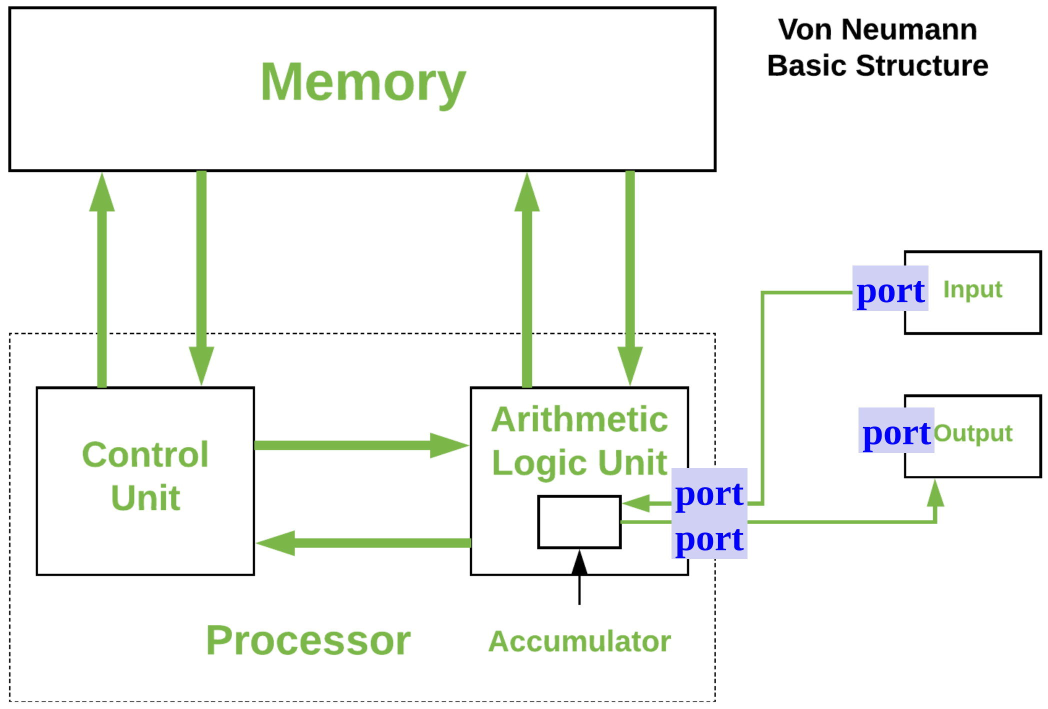
vectorization

Fast C++ (for x86)

Computer interface and their functions (1/3)

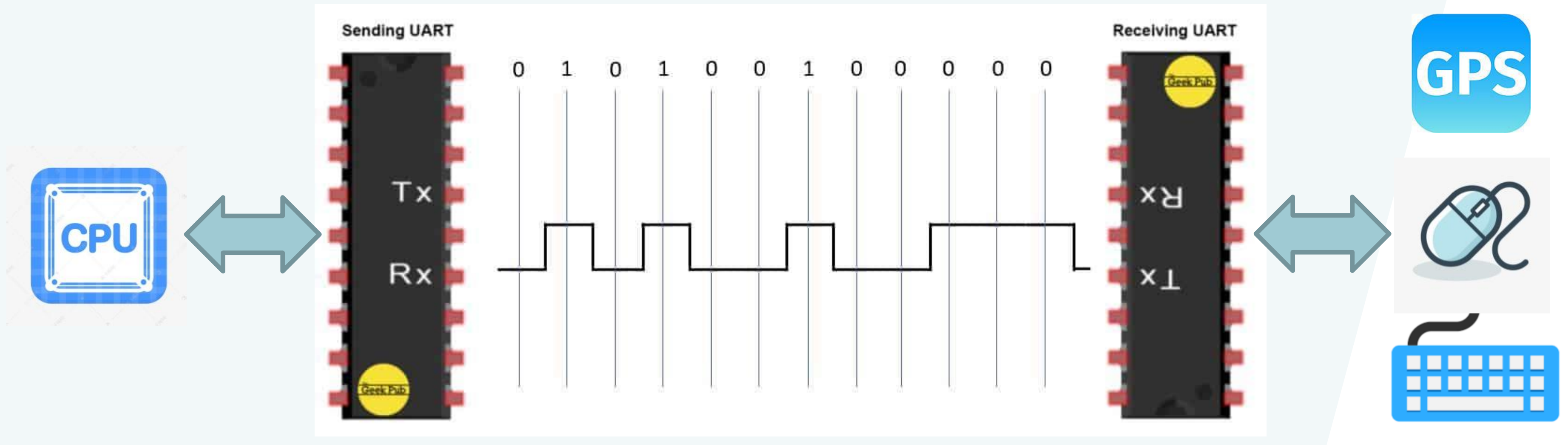


Computer ports



Computer interface and their functions (2/3)

- Interface or port is a point of connection that links together computer and peripheral devices, so that they can work together.



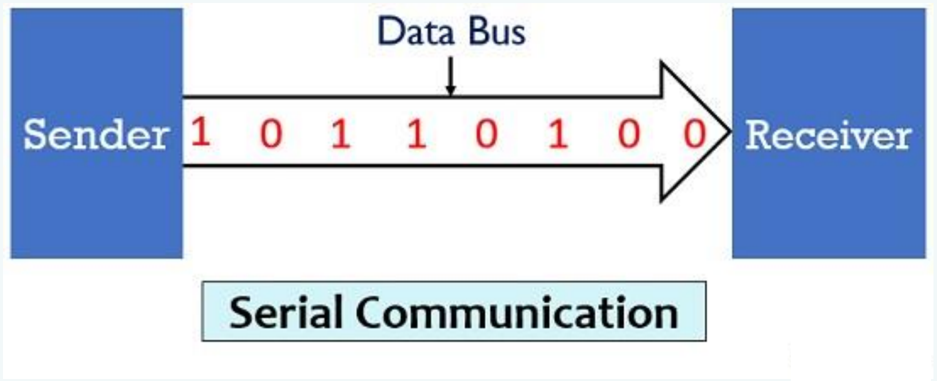
Computer

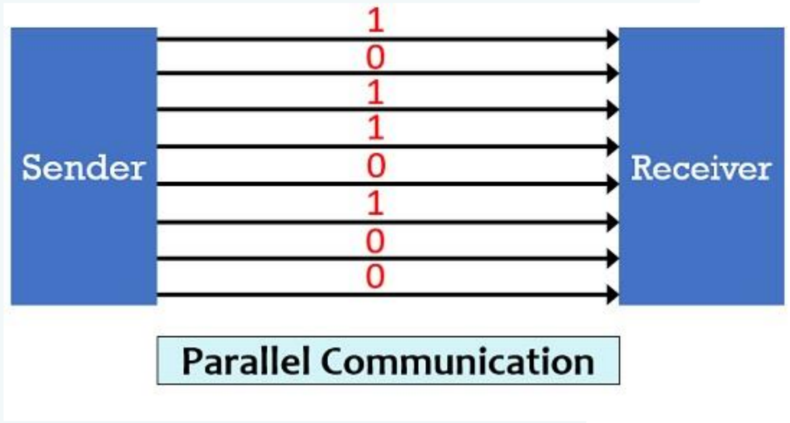
Serial interface

Peripheral

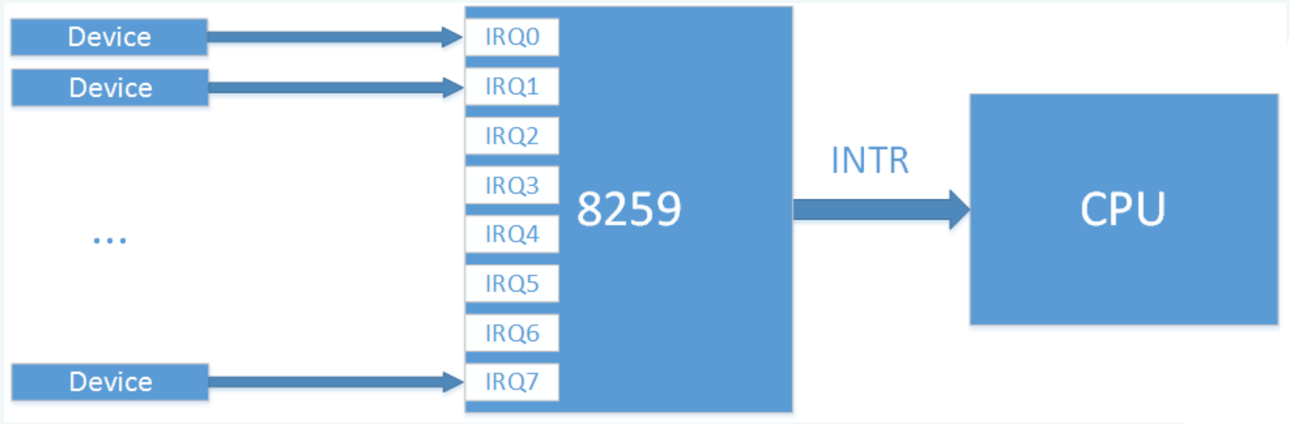
Computer interface and their functions (3/3)

- The function of interface:
  - Communication: serial/parallel communication





  - Control: interrupt controller, timer



Interrupt controller

Topics covered

- **Topics**
  - **The microprocessor and its architecture (Chap. 1-2)**
  - **Addressing modes (Chap. 3)**
  - **Data movement instructions (Chap. 4)**
  - **Arithmetic and logic instructions (Chap. 5)**
  - **Program control instructions (Chap. 6)**
  - **Using assembly language with c/c++ (Chap. 7)**
  - **Basic I/O interface (Chap. 11)**
  - **Interrupts (Chap. 12)**
  - **Arithmetic coprocessor and SIMD (chap. 14/++)**

Experiment (1/5)

- Preliminary experiment (optionally)
  - Programming——Environment setup
  - Programming——Branching
  - Programming——Loops
  - Programming——Mixing assembly and C/C++
  - Programming——x64 assembly programming
  - Programming——Basics of SIMD programming
  - I/O interface

Experiment (2/5)

- Exploratory research (mandatory)
  - Instruction exploration
  - SIMD exploration
  - I/O interface exploration (tentative)

- Why instruction exploration?
  - LLVM compiler: 1,115 contributors, 2.5 million lines
  - Only generates < 1000 out of 3,600 x86 instructions
  - Too many instructions and problems are worth to be explored!

Experiment (3/5)

- Topics for the instruction exploration
- From a CPU (Intel/ARM/RISC-V…) standpoint
  - NOP, LEA, XCHG, CMOV……
  - Intel CAT(Cache Allocation Technology)
  - prefetch in SSE instruction
- From a compiler or virtual machine standpoint
  - Directive: .code, org, record, ……
  - Code generation: switch, while, volatile, ……
  - Different optimization level: loop, ……
  - Python Virtual Machine (PVM), Java Virtual Machine (JVM), ...
- From your standpoint
  - Based on your experience

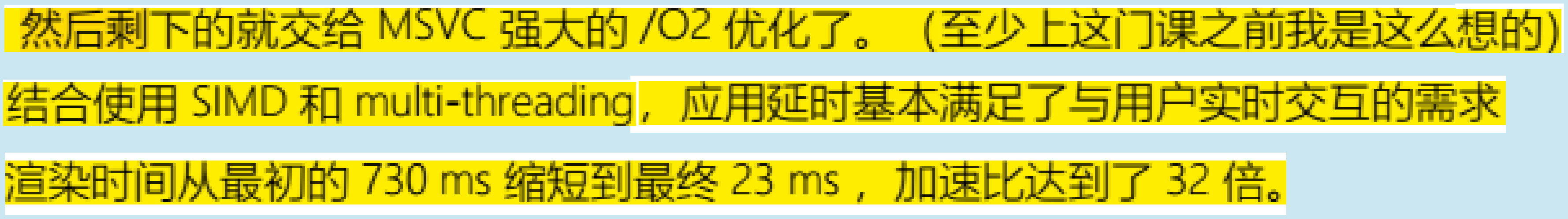
Experiment (4/5)

- How to conduct an exploratory research?
  - Narrow down your research question
  - Observation/motivation -> assumption -> method -> evaluation
  - Use tools for analyzing
  - Get your hands dirty with code
- Requirements
  - Work independent
  - Report
  - Oral Presentation (<=8 mins, before oct.31)
- Your research will be the source of our final examination!

Experiment (5/5)

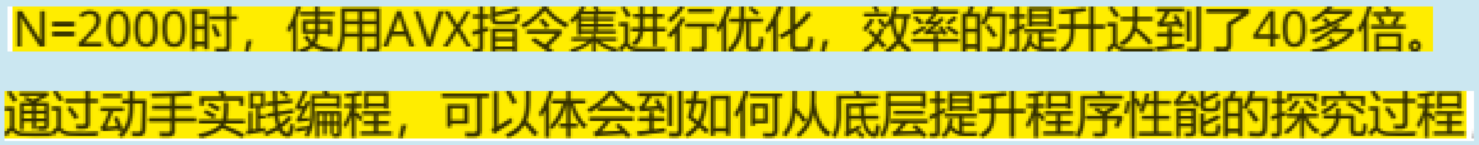
- Topics for the SIMD exploration
- From your standpoint (e.g., SRTP, project)
  - Practicing SIMD to maximize performance (>= 10X speedup)
- From a compiler (GCC/LLVM/…) standpoint
  - Automatic vectorization in compiler (GCC, LLVM, ...)
  - Vectorization in programming language
- From a CPU standpoint
  - X86: SSE and AVX
  - ARM: NEON and SVE
  - RISC-V: “V” and “P” extension
- From a framework/library standpoint
  - NumPy, Eigen, OpenBLAS
  - OpenCV, Tencent/NCNN ⇒ ARM NEON assembly level of careful optimization

Comments on exploratory research

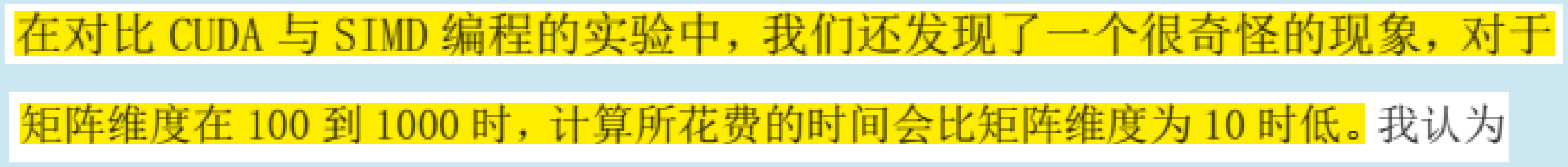

然后剩下的就交给 MSVC 强大的 /O2 优化了。（至少上这门课之前我是这么想的）

结合使用 SIMD 和 multi-threading，应用延时基本满足了与用户实时交互的需求

渲染时间从最初的 730 ms 缩短到最终 23 ms，加速比达到了 32 倍。


N=2000时，使用AVX指令集进行优化，效率的提升达到了40多倍。

通过动手实践编程，可以体会到如何从底层提升程序性能的探究过程


在对比 CUDA 与 SIMD 编程的实验中，我们还发现了一个很奇怪的现象，对于

矩阵维度在 100 到 1000 时，计算所花费的时间会比矩阵维度为 10 时低。我认为

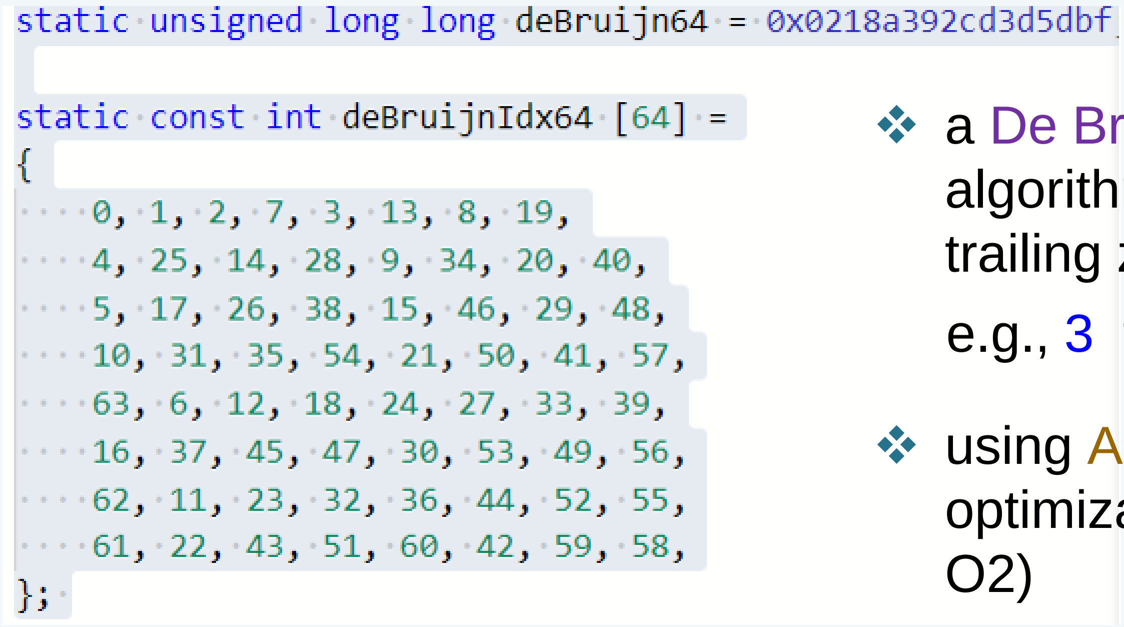
Course Assessment

- **Exploratory research: 30%**

- **Final: 70%**
    - Score of Final Examination: ≥ 50

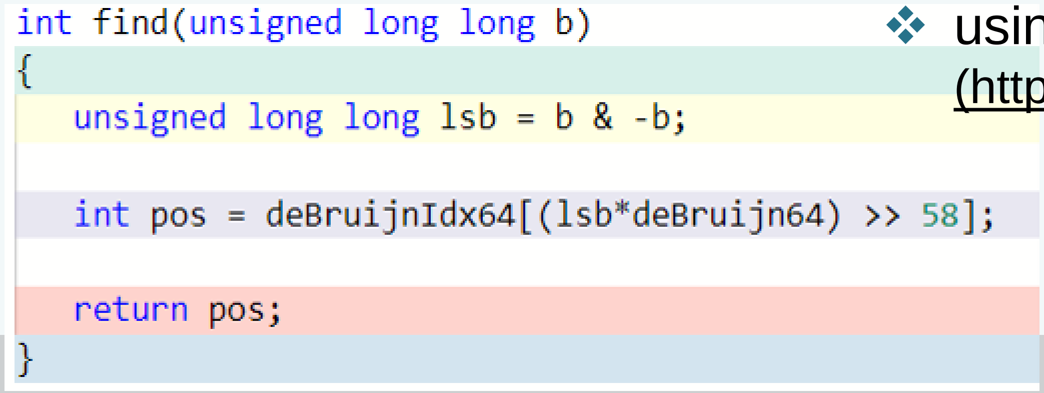
Appendix A: Peek into compiler’s code

- To gain a better understanding of how a compiler works.



```
static unsigned long long deBruijn64 = 0x0218a392cd3d5dbf

static const int deBruijnIdx64 [64] =
{
    0, 1, 2, 7, 3, 13, 8, 19,
    4, 25, 14, 28, 9, 34, 20, 40,
    5, 17, 26, 38, 15, 46, 29, 48,
    10, 31, 35, 54, 21, 50, 41, 57,
    63, 6, 12, 18, 24, 27, 33, 39,
    16, 37, 45, 47, 30, 53, 49, 56,
    62, 11, 23, 32, 36, 44, 52, 55,
    61, 22, 43, 51, 60, 42, 59, 58,
};
```



```
int find(unsigned long long b)
{
   unsigned long long lsb = b & -b;

   int pos = deBruijnIdx64[(lsb*deBruijn64) >> 58];

   return pos;
}
```

- a De Bruijn sequence based algorithm to count number of trailing zeros,

  e.g., 3 for 01111000

- using ARM64 gcc with different optimization options (-O0, -O1, -O2)

- using compiler explorer (https://www.godbolt.org)

Reasons to learn assembly language (3/5)

- You will be better equipped to analyze bugs in programs.

```
struct __attribute__((__packed__)) packed_struct
{ char c; short i; int j; };
```

variable
assignment

```
struct  unpacked_struct { char c; short i; int j; };
```

- Debug edition (-O0)

LDNW: Load nonaligned word from memory

- Release edition (-O3)

LDW: Load word from memory being aligned