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")**
  password: assembly
- **TA1: Zhining Kang kang264@zju.edu.cn**
- **TA2: Jiajun Qin hobbitqia@zju.edu.cn**

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

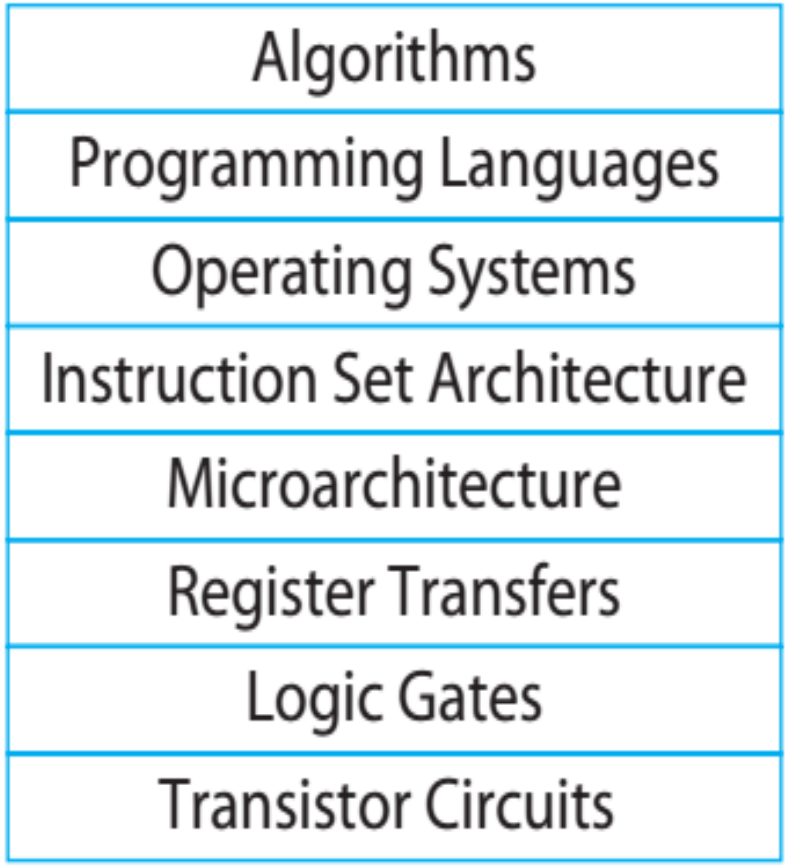
Useful software links

- 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
- Online x86 / x64 Assembler and Disassembler: https://defuse.ca/online-x86-assembler.htm#disassembly

Abstraction Layers in Computer Systems



- 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)

- Lacking knowledge of assembly prevents us from understanding valuable information on how a program runs, and limits understanding of what the code is actually doing.
- Consider the following example from "Dive Into Systems" (Chap 12 Code Optimization) with a = INT_MAX:

```
int  silly (int a)  {
     return (a + 1) > a;
}
```

inconsistent output

(unspecified behavior)

- using x86-64 clang with optimization (-O0)

return 0

```
.....
mov   eax,  DWORD PTR [rbp-0x4]
add   eax,  0x1
cmp   eax,  DWORD PTR [rbp-0x4]
.....
```

- using x86-64 clang with optimization (-O1)

return 1

```
mov   eax,  0x1
ret
```

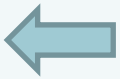
Reasons to learn assembly language (3/4)

- To gain a deeper insight of how optimization 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

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



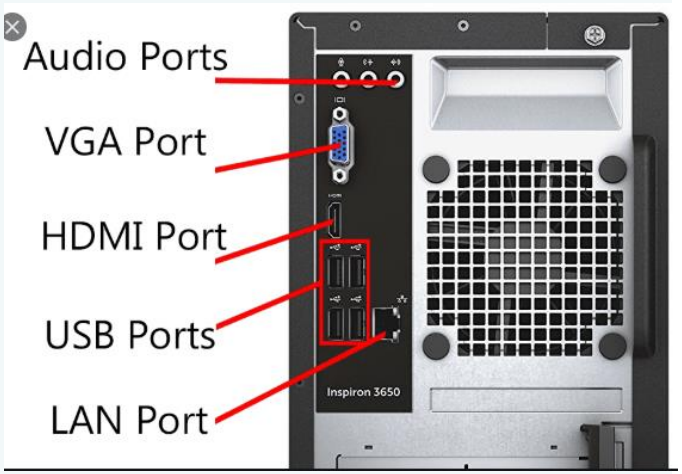
- using compiler explorer: https://www.godbolt.org/z/xTYPhr14n
- using x86-64 gcc with different optimization options (-O1, -O3)
- using x86-64 icc with optimization option (-O3)
- using x86-64 clang with optimization option (-O3)

(idiom recognition)

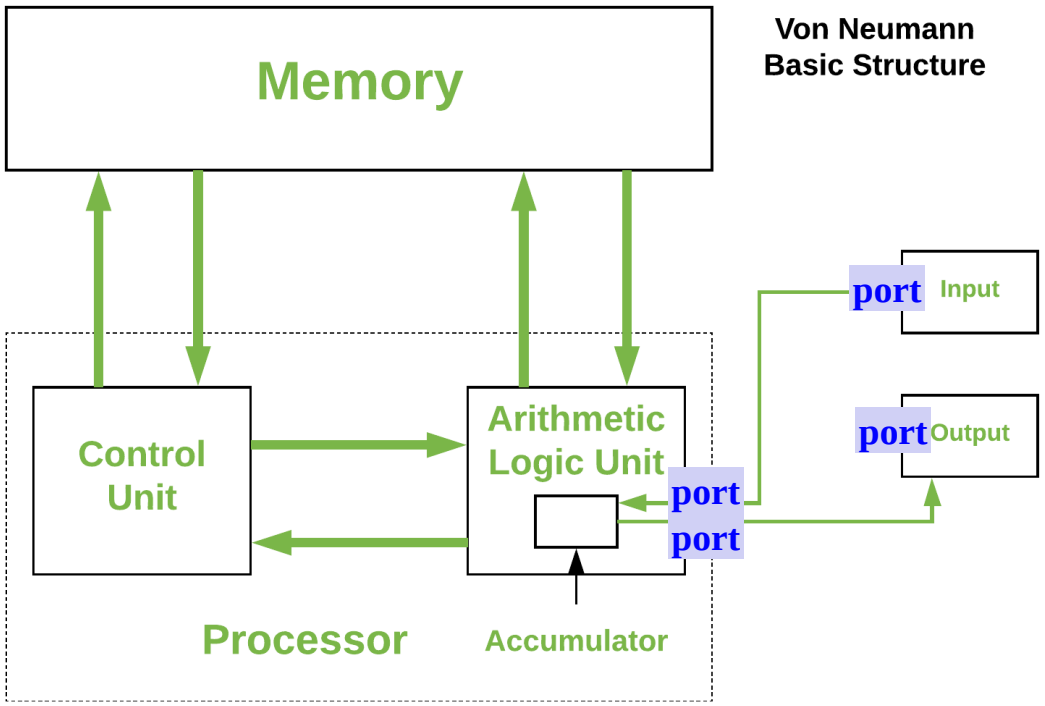
Reasons to learn assembly language (4/4)

- To develop highly efficient and lightweight applications.
  - **NCNN** (Tencent) is a high-performance neural network inference computing framework for mobile platforms featured by ARM NEON assembly level of careful optimization.
  - **MNN** (Alibaba) is a highly efficient and lightweight deep learning framework. It supports inference and training of deep learning models on-device. It implements core operations by relying on large amounts of handwritten assembly code to make full use of the ARM CPU.

Computer interface and their functions (1/3)

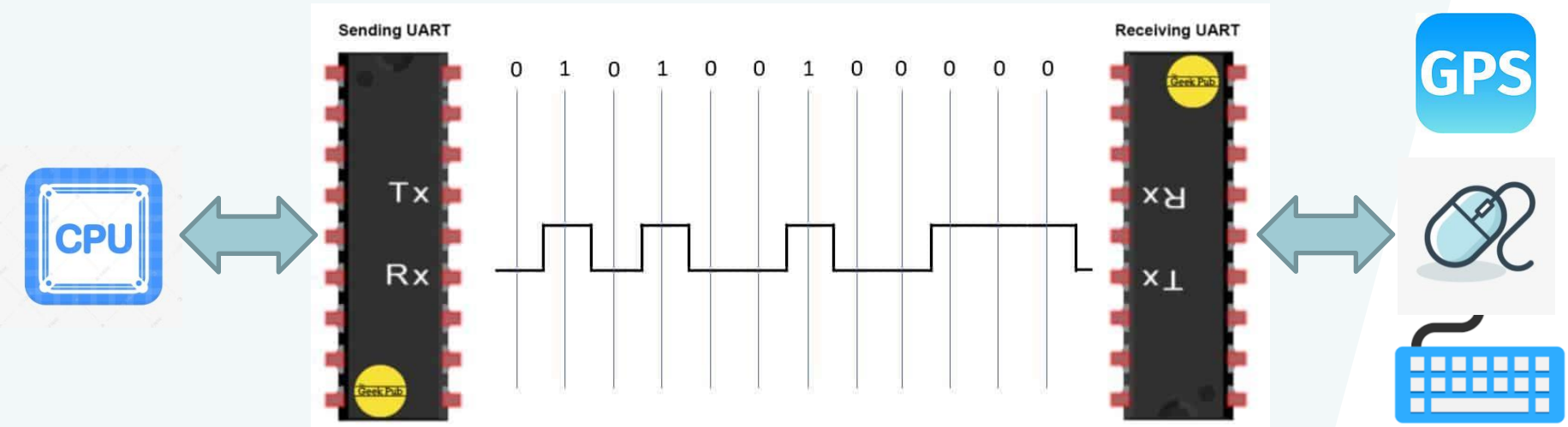


Computer ports



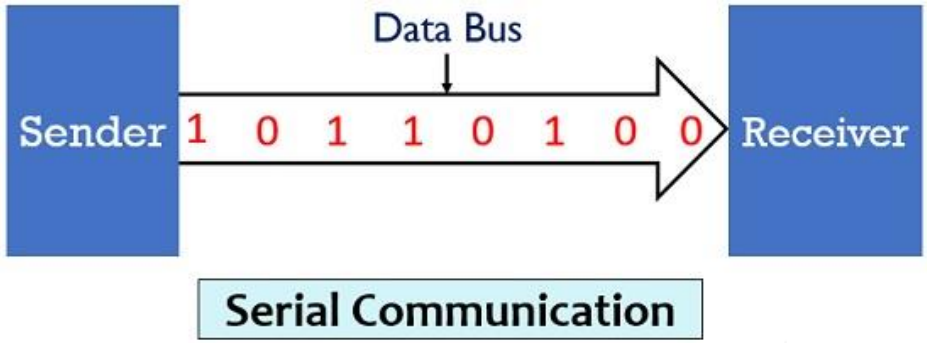
Computer interface and their functions (2/3)

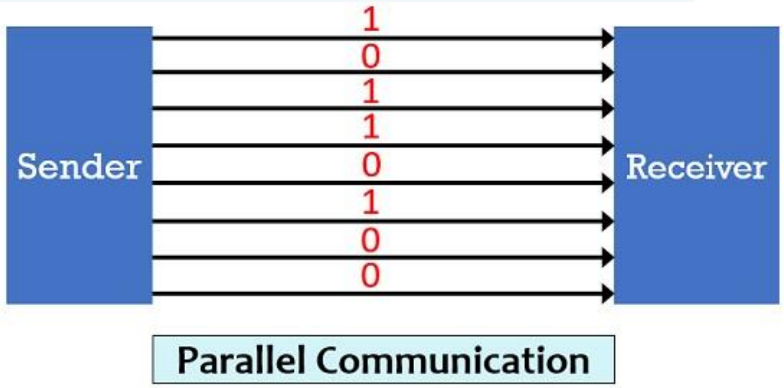
- Interface or port is a point of connection that links together computer and peripheral devices, so that they can work together.



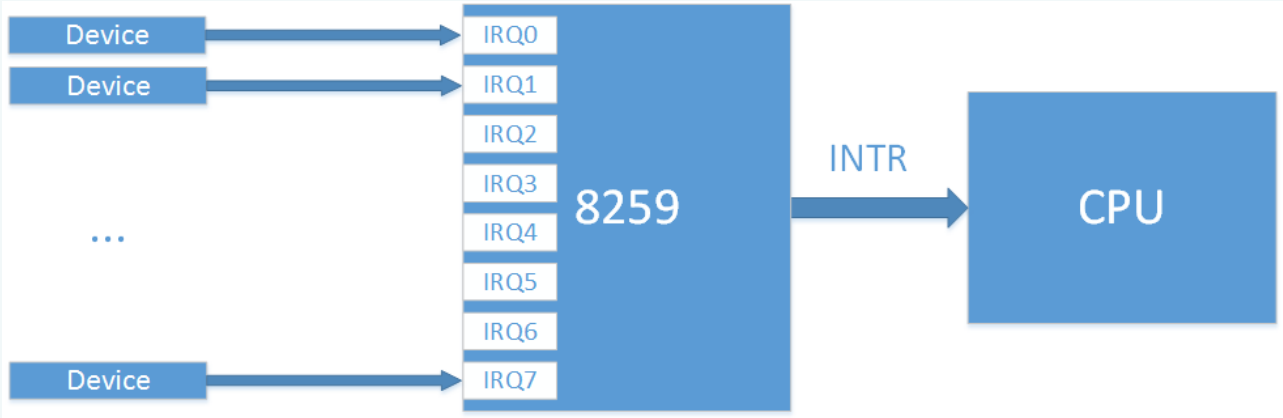
Computer interface and their functions (3/3)

- 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
  - I/O interface

Experiment (2/5)

- Exploratory research (mandatory)
  - Project 1: Floating-point Operations in Programming Language
  - Project 2: Exploring the LLMs' Potential in Assembly Code Understanding, Generation, and Optimization

Project 1: Floating-point Operations

- We know very little about floating-point arithmetic.

- For example, which is true of the following boolean expression, given that x is a variable of type double?

  3.0 == x * (3.0 / x)

  - A. It will always evaluate to false.
  - B. It may evaluate to false for some values of x.
  - C. It will evaluate to false only when x is zero.
  - D. It will evaluate to false only when x is very large or very close to zero.
  - E. It will always evaluate to true.

Project 1: Floating-point Operations

- Key Points to Consider:
  - Floating-Point Precision: In most programming languages, the double type represents a floating-point number with finite precision. Operations involving floating-point numbers can result in small rounding errors.
  - Very Large or Very Small Values of x: For very large or very small values of x, the result of x * (3.0/x) may suffer from significant precision loss, potentially causing the expression to evaluate to false.

- Involving very large or very small floating-point calculations may also lead to a significant performance drop.
  - E.g., abnormal execution time in the POW function

Project 1: Floating-point Operations

- In the project, you are required to explore how programming languages handle floating-point arithmetic.

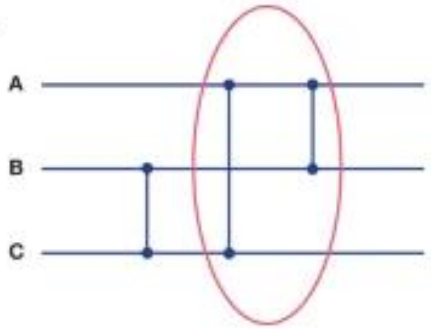
- Recommended topics (but not limited to):
  - Exploration of Anomalies in POW Function Execution Time
  - Cross-Language Floating-Point Precision Benchmark and Analysis
  - Profiling Floating-Point Performance on Different Architectures
  - Decimal Floating-Point Arithmetic
  - Custom Floating-Point Format and Emulator

Project 2: LLMs' Potential in Assembly

- Faster sorting algorithms discovered using deep reinforcement learning. Nature 618, 257–263 (2023).

Fig. 3: Sorting networks and algorithmic improvements discovered by AlphaDev.

a



b Original

```
Memory[0] = A
Memory[1] = B
Memory[2] = C

mov Memory[0] P  // P = A
mov Memory[1] Q  // Q = B
mov Memory[2] R  // R = C

mov R S
cmp P R
cmovg P R  // R = max(A, C)
cmovl P S  // S = min(A, C)
mov S P    // P = min(A, C)
cmp S Q
cmovg Q P  // P = min(A, B, C)
cmovg S Q  // Q = max(min(A, C), B)

mov P Memory[0]  // = min(A, B, C)
mov Q Memory[1]  // = max(min(A, C), B)
mov R Memory[2]  // = max(A, C)
```

c AlphaDev

```
Memory[0] = A
Memory[1] = B
Memory[2] = C

mov Memory[0] P  // P = A
mov Memory[1] Q  // Q = B
mov Memory[2] R  // R = C

mov R S
cmp P R
cmovg P R  // R = max(A, C)
cmovl P S  // S = min(A, C)

cmp S Q
cmovg Q P  // P = min(A, B)
cmovg S Q  // Q = max(min(A, C), B)

mov P Memory[0]  // = min(A, B)
mov Q Memory[1]  // = max(min(A, C), B)
mov R Memory[2]  // = max(A, C)
```

Project 2: LLMs' Potential in Assembly

- Meta Large Language Model Compiler: Foundation Models of Compiler Optimization. arXiv:2407.02524 (2024).
- Built on the foundation of Code Llama, LLM Compiler enhances the understanding of compiler intermediate representations (IRs), assembly language, and optimization techniques.
- The model showcases its enhanced capabilities in optimizing code size and disassembling from x86_64 and ARM assembly back into LLVM-IR. These capabilities achieve 77% of the optimization potential of an autotuning search and 45% disassembly round-trip efficiency.

Steps for an exploratory research

- How to conduct an exploratory research?
  - Narrow down your research question
  - Observation/motivation → assumption → method → evaluation
  - Integrate your previously learned knowledge for a comprehensive understanding
  - Use tools for analyzing effectively
  - Get your hands dirty with code
- Requirements
  - Work independent
  - Report
  - Deadline

✓Project 1: Nov.10 ✓ Project 2: Dec.31

Course Assessment

- **Exploratory research: 30%**
- **Final: 70%**
  - Score of Final Examination: ≥50