Welcome to Microprocessor Systems

An introduction to CPT210

Module Information

About the module instructor, teaching organisation.

Teaching Organisation

- Module instructor: Jianjun Chen
  - Email address: Jianjun.Chen@xjtlu.edu.cn
  - Office: SD 541
  - Office hours: please check the module page online
    - If you wish to see me at other times, please email me first.

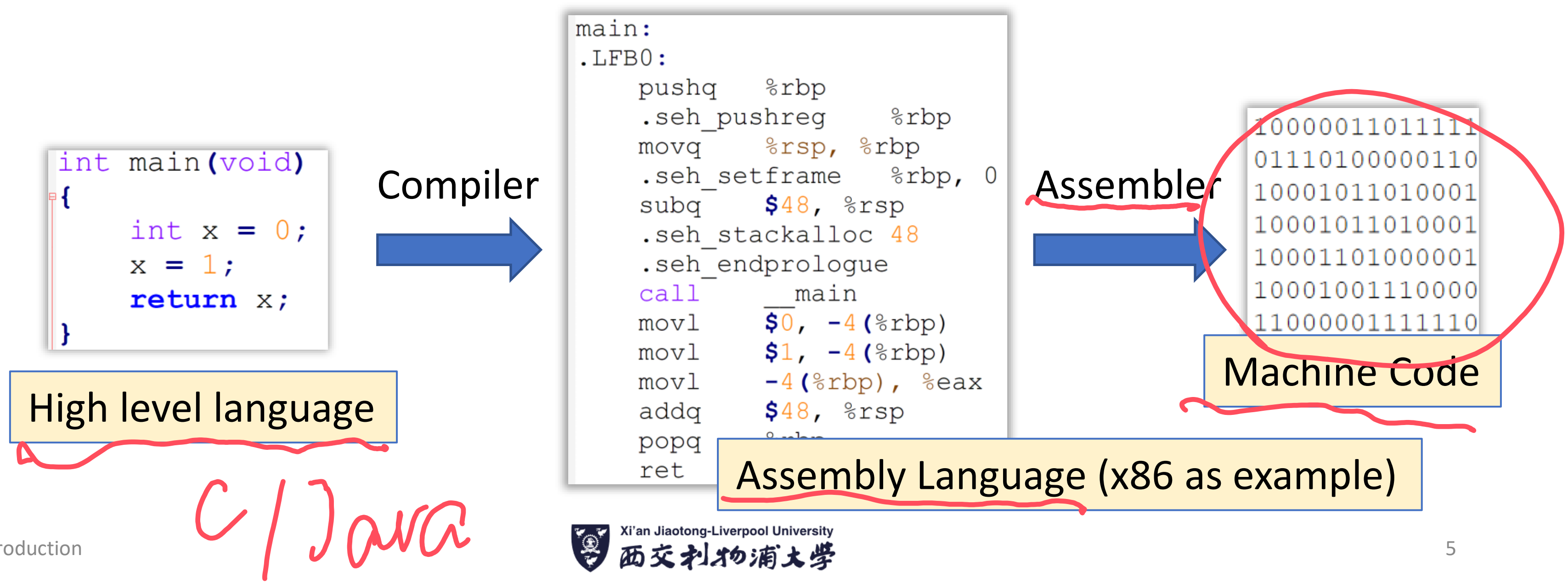
- Module instructor: Kok Hoe Wong
  - Email address: kh.wong@xjtlu.edu.cn
  - Office: SD 431
  - Office hours: please check the module page online

Teaching Organisation

- Support:
  - I have office hours every week.
  - Work in groups with your friends during labs. Discussions with classmates enable you to better understand concepts and terminologies.
  - I will add one or more notice boards on the Learning Mall about issues (like coursework, exam, labs) related to this module, I will also send notifications to you about any updates. Please check emails frequently.
- Grades: Please check the information about coursework and exams on e-bridge.
  - You are encouraged to discuss ideas (Not solutions!) with others when doing coursework.
  - But your assignment submissions must be your own works.

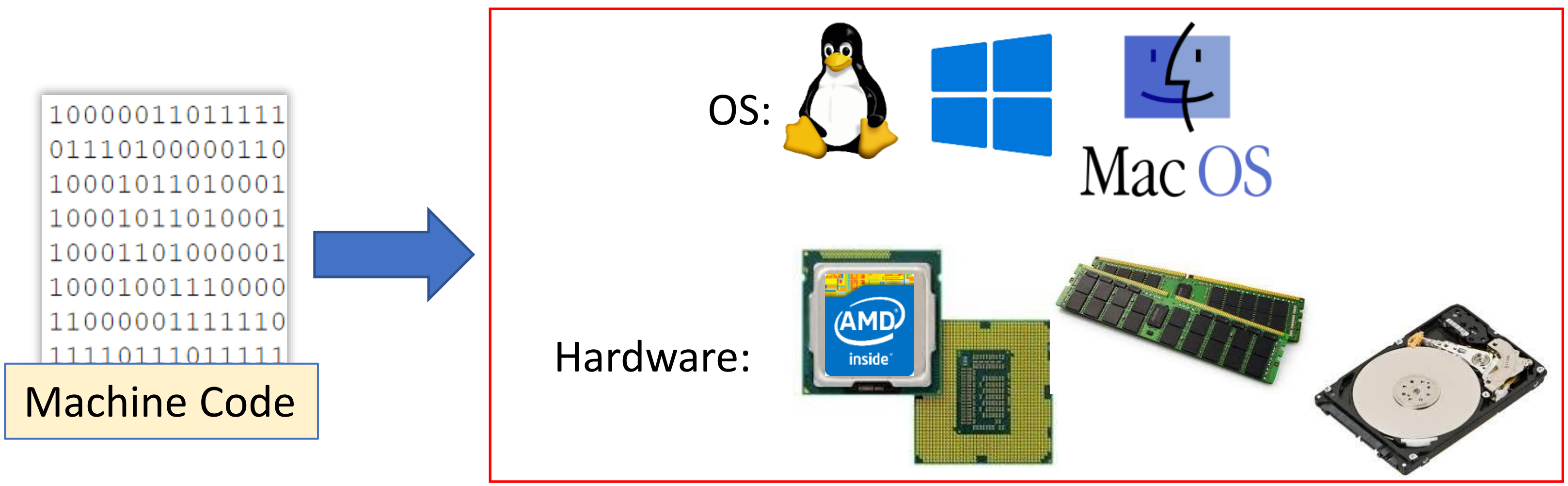
About this Module

- This module is a "bridge" between digital electronics (hardware) and high level languages (software).
- You will learn basic concepts required to understand how computers work
  - What programs are? How does your C/C++ code become the instructions that your computer understands?
  - How these instructions are executed in hardware?



About this Module

- This module is a “bridge” between digital electronics (hardware) and high level languages (software).
- You will learn basic concepts required to understand how computers work
  - What programs are? How does your C/C++ code become the instructions that your computer understands?
  - How are these instructions executed in hardware?



Focus of this Module

- We will focus on Instruction Set Architectures and their use through Assembly Language Programming.

- The architecture studied in this module will be ARM.

Introduction to Microprocessor Systems

Evolution of technology

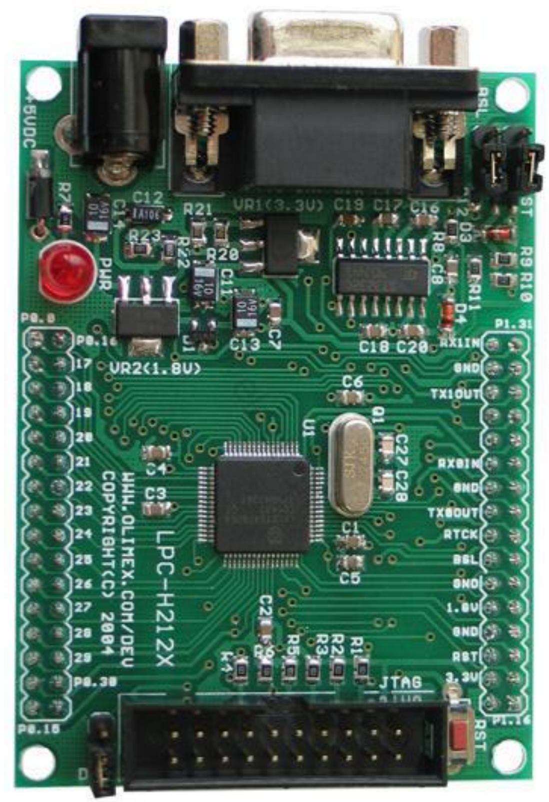
Design of ISA

The Evolution of Computers

- Electronic Delay Storage Automatic Computer (EDSAC)
  - First practical programmable computer, by UK. In Cambridge 1949.
  - On average, processes 650 instructions per second.
  - 1024 17-bit words of memory in mercury ultrasonic delay lines.
  - 3000 valves, 12 kW power consumption, occupied a room 5m by 4m.
  - Early use to solve problems in meteorology, genetics and X-ray crystallography.

The Evolution of Computers

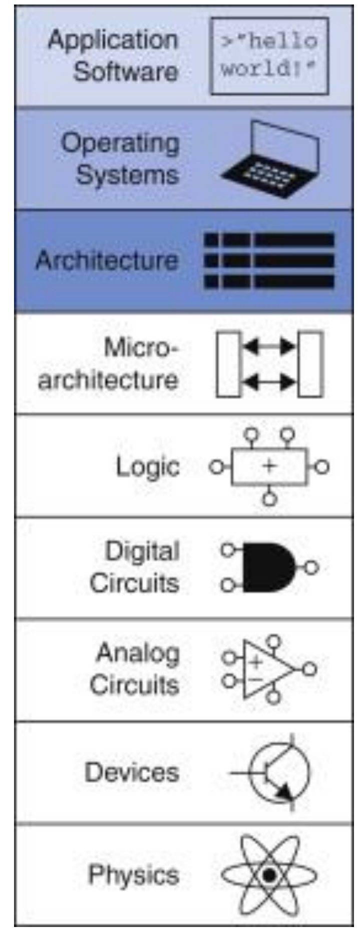
- ARM7
  - Up to 130 million instructions per second. 1995-2011
  - One of the most successful embedded processor.
- The picture shows LPC-H2124 header board.
  - 256KB of program flash
  - 16KB of RAM.
  - The square chip in the middle is the microcontroller
- Original ARM design:
  - Steve Furber, Acro RISC Machine
  - Cambridge, 1985
- Evolution of computer architecture



What is Computer Architecture?

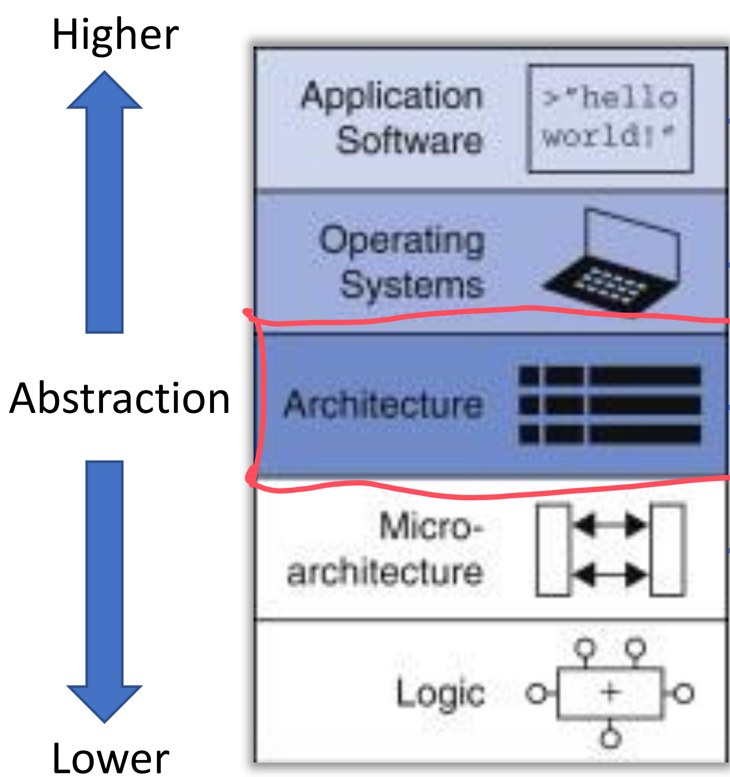
- *Computer architectures* represent the means of interconnectivity for a computer's hardware components as well as the mode of data transfer and processing exhibited.
  - *Paul J. Fortier, Howard E. Michel, in [Computer Systems Performance Evaluation and Prediction](), 2003*

- The *architecture* is the programmer's view of a computer. It is defined by the instruction set (language) and operand locations (registers and memory). Many different architectures exist, such as ARM, x86, MIPS, SPARC, and PowerPC.
  - *Sarah L. Harris, David Money Harris, in [Digital Design and Computer Architecture](), 2016*



What is Computer Architecture?

Architecture is about design, not just hardware.



- Web browsers, Games, Office
- Linux, Windows, macOS
- ISA: like ARM, x86, MIPS
- Cortex-A77, Core i7, AMD Ryzen7

Also called Instruction Set Architecture (ISA)

Instruction Set Architecture (ISA)

- Instruction (or Operation Code) set.
  - ADD, SUB, RSB, etc.
- Organisation of programmable storage.
  - Where data is stored.
- Mode of addressing and accessing data items and instructions.
  - How data is stored/retrieved.
- Behaviour on exceptional conditions, e.g.:
  - Hardware divide by zero.
  - An interrupt occurred

- ISA is about what the computer does (not how it does).

Instruction Set Architecture (ISA)

- Examples of ISAs:
  - **8086/Pentium (x86) ISA**: Widely used, compatible with a lot of operating systems and software.
  - **ARM ISA**: Supports highly optimising compiler & operating system software for embedded applications.

x86 instruction	
ADD	Add numbers
AND	Logical AND
MOV	Copy data between memory blocks/registers

ARM instruction	
ADD	Add numbers
AND	Logical AND
LDR	Load from memory (can't save to memory)
STR	Store to memory (can't load from memory)

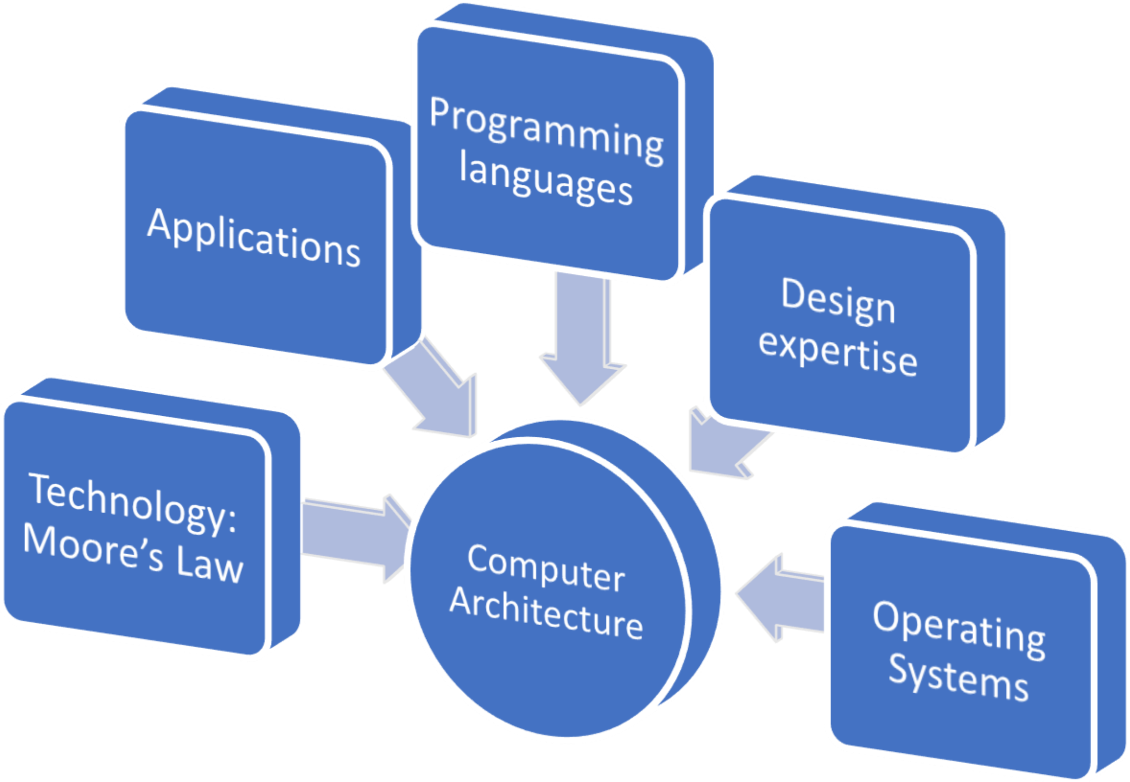
- Different implementations of the same ISA can run identical software.
  - Your compiled C program can run on any x86 computer as long as the operating system is the same and libraries are satisfied.

Discussion

- Discuss the questions below based on your understanding and knowledge so far.

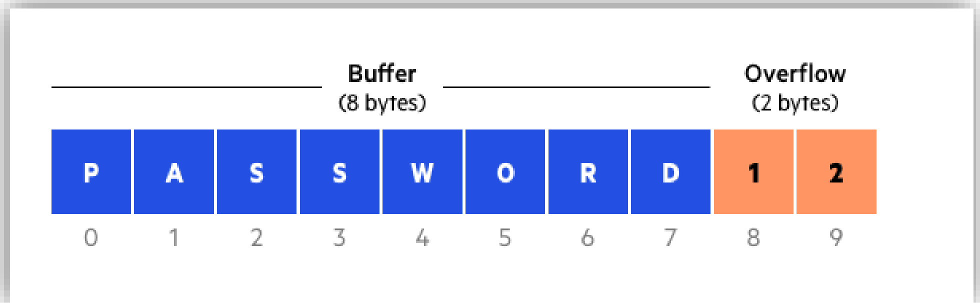
- Computer architecture affects:
  1. CPU design?
  2. Compiler design?
  3. Energy efficiency of the computer?
  4. Maximum memory size?
  5. UI design of the operating system (OS)?
  6. Security of the OS?

What Factors Influence Computer Architecture?



Example: Buffer Overflow Attack

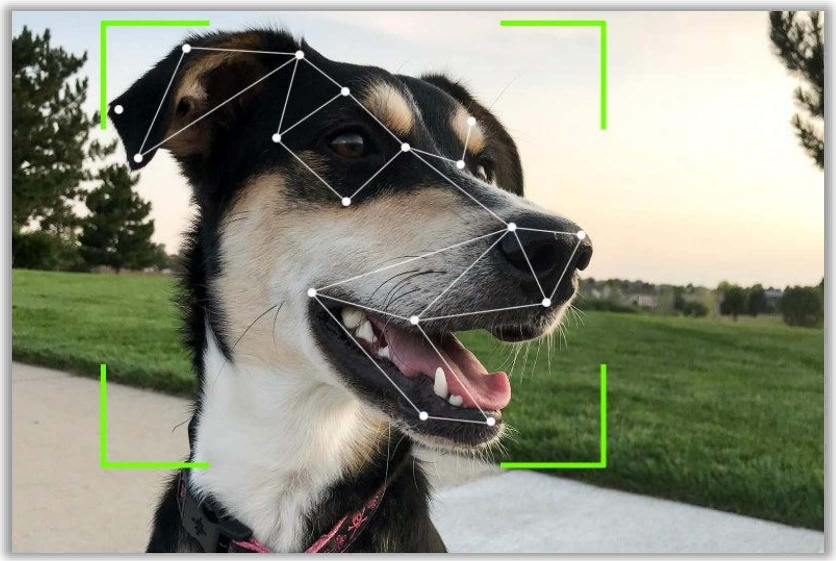
- This attack targets programs that accept input data from untrusted sources and do not verify the length of the stored data.



- Overloaded buffer can override executable code with malicious code.
- Execution Disable Bit (EDB), introduced by Intel
  - Classifies areas in memory where a code can execute or not execute.
  - Stops an attack from running code in a non-executable region.
  - Does not shutdown the buffer overflow attack completely, but can help reduce the impact.

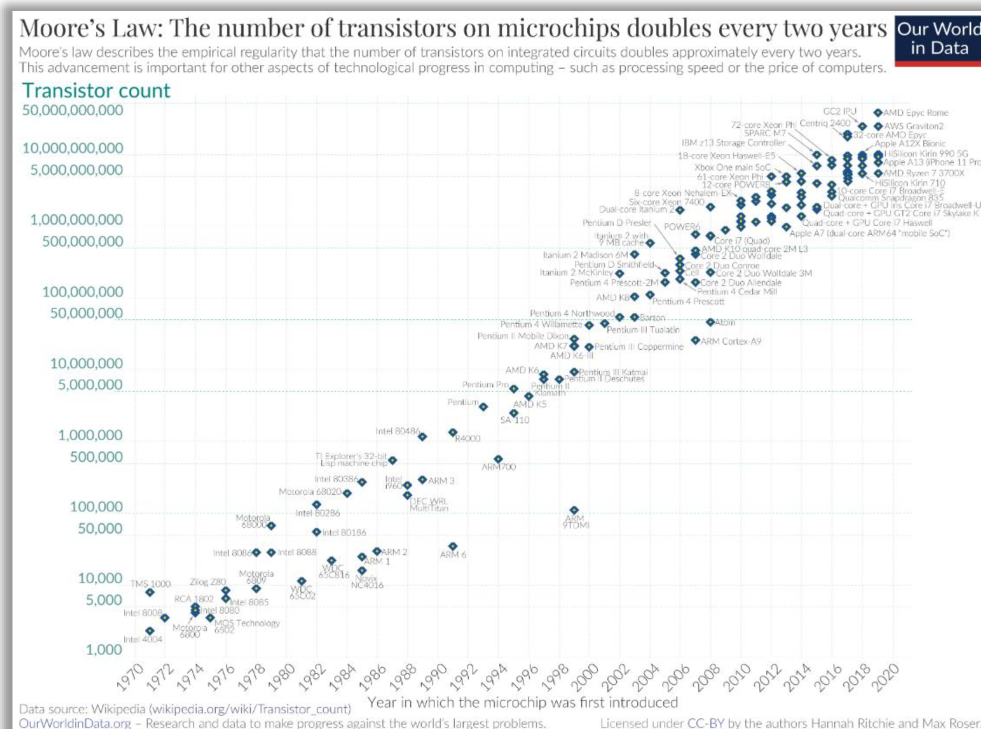
Example: Application

- Special units for AI learning in CPU.
  - Implemented in recent chips like Snapdragon from Qualcomm, Apple Silicon, Kirin chips, Exynos chips.
- Video processing unit.
  - Intel chips provide video encoding/decoding accelerating
  - Can greatly speed up video decoding and encoding.



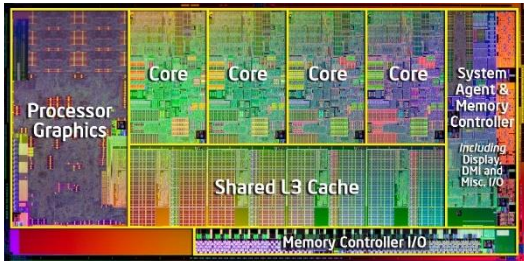
Example: Technology (Moore's Law)

- In 1965, Gordon E. Moore, co-founder of Intel, postulated that the number of transistors that can be packed into a given unit of space will double about every two years. (See on Wikipedia)



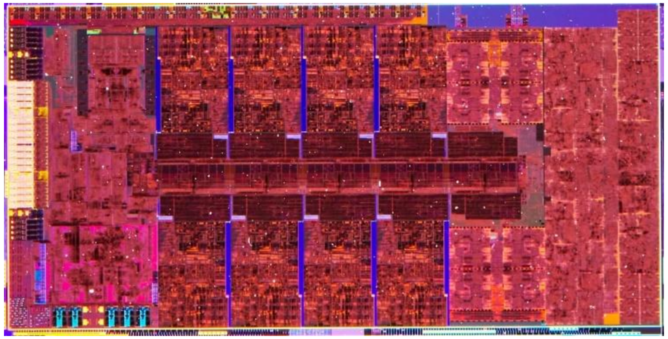
Example: Technology (Moore's Law)

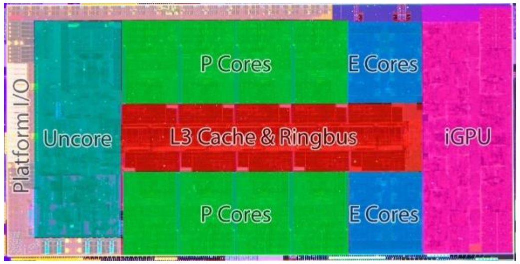
- VLSI (very large scale integration) has increased speed and density of CPUs by shrinking dimensions
  - This is limited by size of atoms (Quantum effects), so will stop.
- Moore's Law predicted a doubling of computing power every two years.
  - This has held true but will stop soon without a change in technology or architecture (Quantum computing).



i7-2600k: 4 cores

i7-12700k:
8 P cores
8 E cores





Xi'an Jiaotong-Liverpool University
西交利物浦大学

Micro-architectures of ARM

- There are many architectures developed by ARM.
- Each architecture has many implementations (micro-architectures).
- Each micro-architecture has its own targeted purpose.
  - Cortex have A, M and R series.

Cortex-A	Cortex-R	Cortex-M
High performance	Real-time response	Low power usage
PCs, laptops, servers, networking equipment, and supercomputers.	medical equipment, vehicle steering, braking and signaling.	IoT and embedded devices, such as wearables and small sensors.

Micro-architectures of ARM

ISA	Cortex-A	Cortex-M	Cortex-R
ARMv7	**Processor cores:** 1. Cortex-A7 2. Cortex-A9 3. Cortex-A17 **Products:** 1. Nvidia Tegra 2 2. HiSilicon K3V2	**Processor cores:** 1. Cortex-M0 2. Cortex-M3 **Products:** 1. Texas Instruments F28 2. Realtek RTL8710	**Processor cores:** 1. Cortex-R4 2. Cortex-R8
ARMv8	**Processor cores:** 1. Cortex-A53 2. Cortex-A57 3. Cortex-A73 **Products:** 1. Apple A7 (iPhone 5) 2. Exynos 5433 (Galaxy Note 4) 3. Nvidia Tegra X1	**Processor cores:** 1. Cortex-M23 2. Cortex-M33 **Products:** 1. Nuvoton M2351 2. GigaDevice GD32E230 3. NXP LPC5500 4. ST STM32 L5	**Processor cores:** 1. Cortex-R52 2. Cortex-R82
ARMv9	**Processor cores:** 1. Cortex-A715 **Products:** 1. MediaTek Dimensity 9200 2. Qualcomm Snapdragon 8 Gen 2		

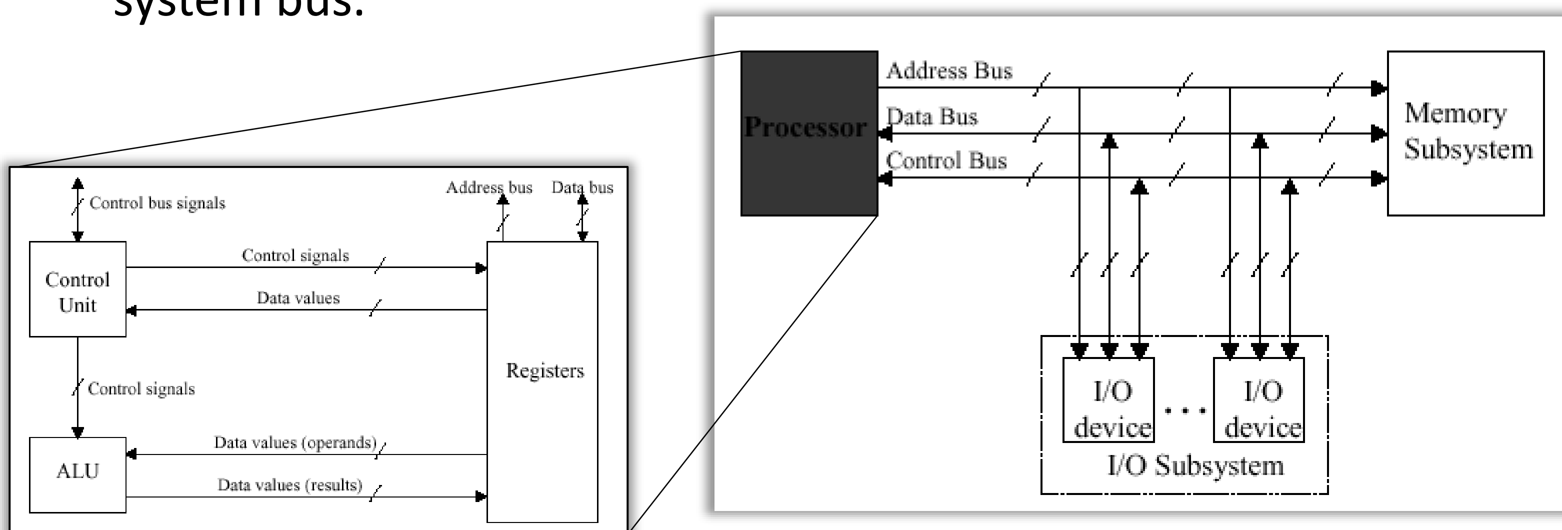
https://en.wikipedia.org/wiki/Template:Embedded_ARM-based_chips

The Organisation of Microprocessor Systems

Some basic terms and knowledge

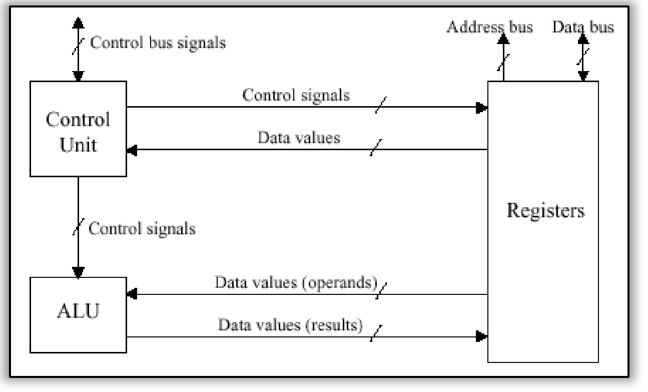
Von Neumann machine

- Below is the typical organization of a modern Von Neumann machine.
  - Data and program are mostly stored in the computer memory separate from the process.
  - Registers in the processor [datapath]() can also store small amount of data.
  - The address bus, data bus and control bus are also referred to as system bus.



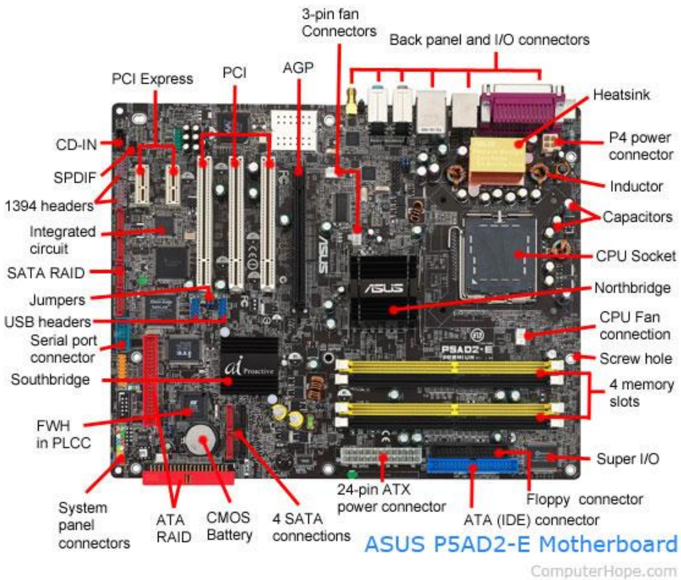
CPU

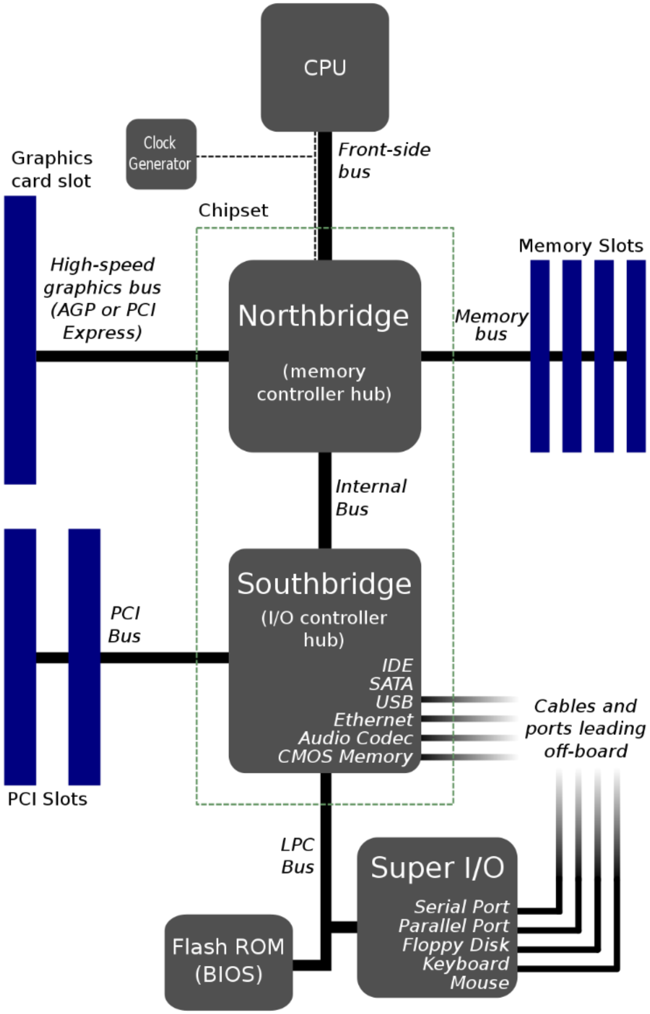
- **Registers**: Very fast but limited memory space embedded inside the CPU.
- **Arithmetic logic unit (ALU)**: Carries out arithmetic and bitwise operations.
  - Arithmetic: + - * /
  - Bitwise: & | ^ ~
- **Control unit**: Coordinator of memory, ALU, registers, I/O devices.
- CPU example workflow:
  1. Fetch the instruction
  2. Fetch the operands
  3. Execute the instruction
  4. Write the results.
  5. Back to 1.



Internal Organisation of PC

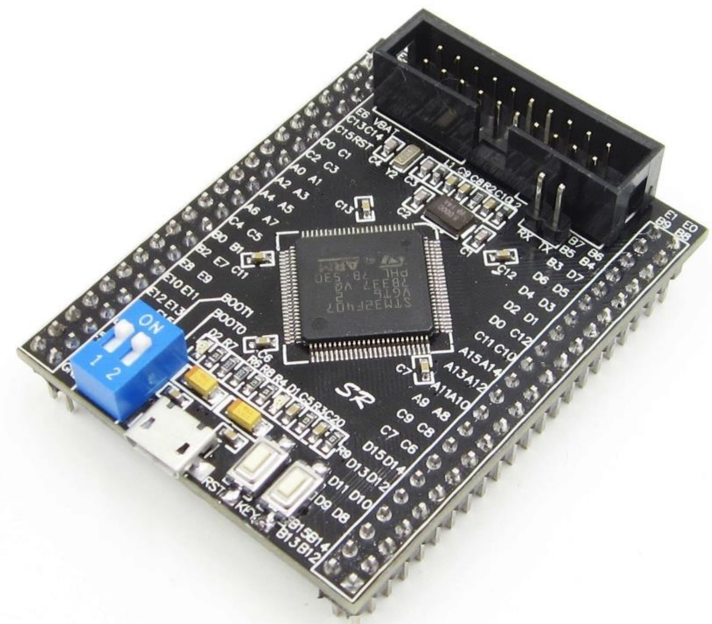
- The image on the right shows the structure of a PC we see in daily life.
  - You can install operating systems on it.



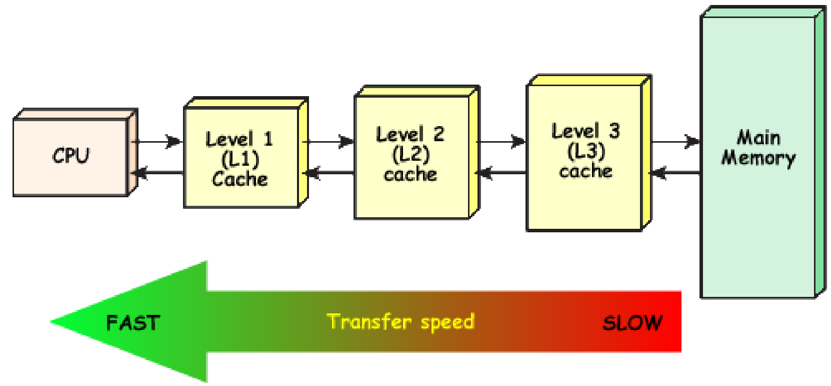


MCU Board

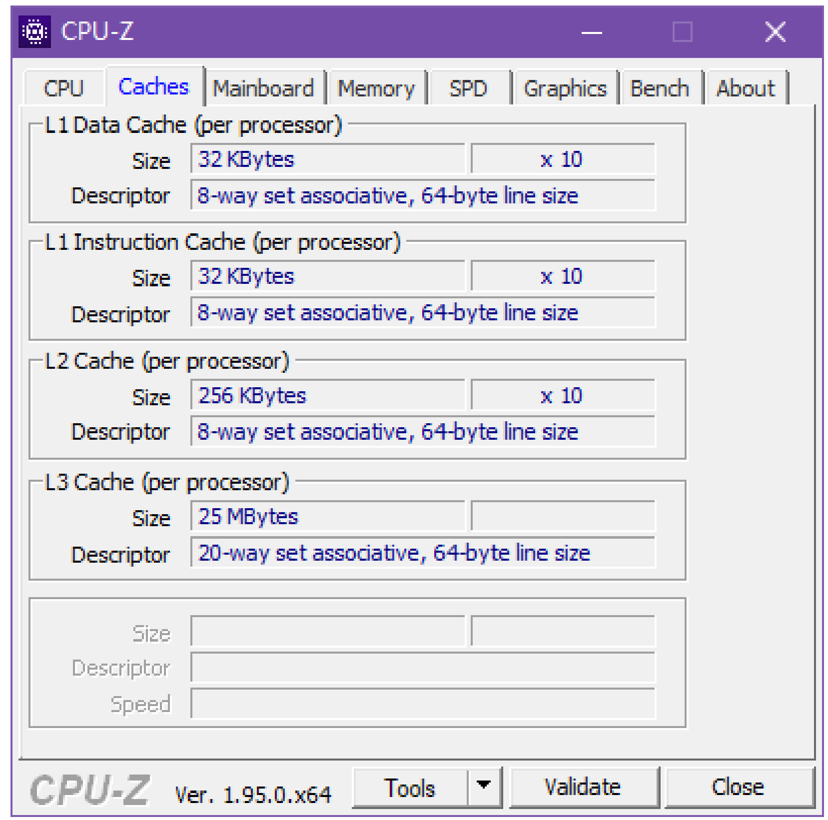
- The image on the right is an ARM MCU board
  - **MCU**: Microcontroller Unit
  - It only runs one single program.
- You cannot install full operating systems on MCU.
  - Lacks memory management unit (MMU).
  - Some details will be covered later in this module.



CPU Cache



- Another Component of CPU.
- Lies in between CPU registers and memory.
  - Access speed: Register > cache > memory
  - Manufacture cost: Register > cache > memory
  - Storage capacity: Memory > cache > register



- Before looking for a certain piece of data in memory, CPU will first look for it in the cache.

Memory


- Data and Program instructions are stored in the primary memory.
  - A byte is a memory unit for storage
  - A memory chip is full of such bytes.
  - Secondary memory refers to external storages
    - CD, Floppy, Hard disk, USB storage etc.
- Technology limitations affects memory size.
- For 32-bit CPUs
  - Address bus is 32-bit wide.
  - Each memory location refers to 1 Byte (8 bits)
  - Can address up to $2^{32} = 4294967296 = 4\ \mathrm{GB}$
    - The lowest address is $0\mathrm{x}00000000$
    - The highest address is $0\mathrm{xFFFFFFFF}$

1 byte

Address	Content
0xFFFFFFFF	0x8b
0xFFFFFFFE	0x11
.	.
.	.
.	.
.	.
0x00000001	0xAA
0x00000000	0xCD

Accessing Memory

4 bytes

- **Data** and instructions are accessed and manipulated in fixed-length chunks.

0 ... 0 1

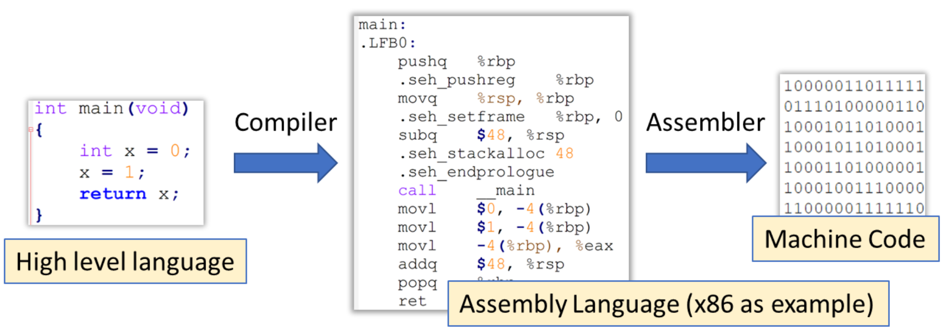
- **Data sizes in C**: 1 byte (char), 2 bytes (short), 4 bytes (int, float), 8 bytes (long)

$4 \times 8 = 32$ bit

  - For data that is less than the size of its `sizeof(type)`, leading zeros/ones will be added.
  - A `char` with value 1 is stored as 000000001 in the memory.
  - A `char` with value -12 is stored as 11110100.
  - A `short` with value -12 is 1111111111110100.
    - Negative binary numbers will be taught later.
- **Data sizes in ARMv7 assembly**: byte, half-word (2 bytes), word (4 bytes)
  - No data type associated.
  - No sign/unsigned number indicator.

Accessing Memory

- <u>Data</u> and <u>**instructions**</u> are accessed and manipulated in fixed-length chunks.
- <u>**Instruction**</u>: each instruction (usually) has a length of one **word**.
  - Words in a 32-bit system are usually 32 bits long.
  - Words in a 64-bit system are usually 64 bits long.
    - Somehow related to the width of the data and address bus.
    - BUT, some ARM instructions are half-word long.
    - In x86, the length of instructions may vary.
  - Usually, word size = address size (pointer size) = register size.



Question

- Data in memory has no associated data type information.
- How does a C program know the data type associated with a piece of memory and calculate correct results?

```
int x = 1;
float y = 1.1;
x = x * 2;
y = y * x;
```

Byte Ordering

0x1 2 3 4

| AA | BB | CC | DD | Big ~

| DD | CC | BB | AA | Little

- Assume we have a 32-bit int value:
  - 0x AA BB CC DD

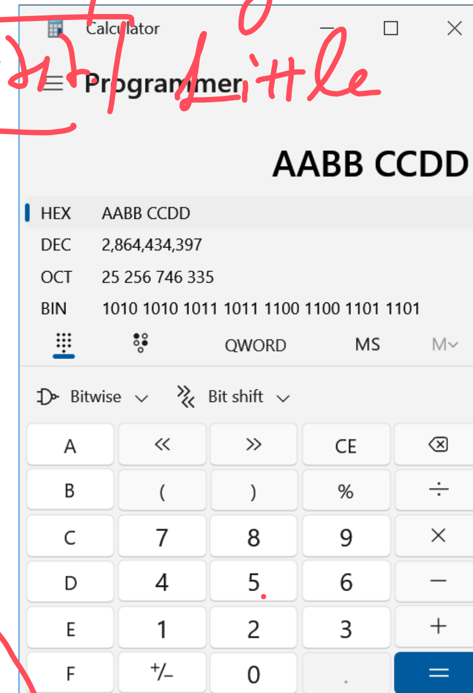
MSB LSB 4 byte

- How should its bytes be ordered in memory?
  - Arrangement 1

Value	DD	CC	BB	AA
Address	0x1	0x2	0x3	0x4

  - Arrangement 2

Value	AA	BB	CC	DD
Address	0x1	0x2	0x3	0x4



- The ordering of bytes is called **endianness**.
  - The first arrangement method is called **little endian**.
  - The second arrangement method is called **big endian**.

Byte Ordering

- Big-endian architectures:
  - SPARC, z/Architecture
  - Least significant byte has highest address.
    - DD at 0x04

- Little-endian architectures:
  - x86, x86-64
  - Least significant byte has lowest address.
    - DD at 0x01

- Both endianness are supported:
  - ARM, MIPS, PowerPC
  - Can be specified

Important Notes about Byte Ordering

- Byte ordering only affects basic data units (word), composite data units like arrays or struct/class are NOT affected by byte ordering.

```
int y[] = {0x41424344, 0x45464748};
```

- Big-endian

Value	0x41	0x42	0x43	0x44	0x45	0x46	0x47	0x48
Address	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08

y[0] y[1]

- Little-endian

Value	0x44	0x43	0x42	0x41	0x48	0x47	0x46	0x45
Address	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08

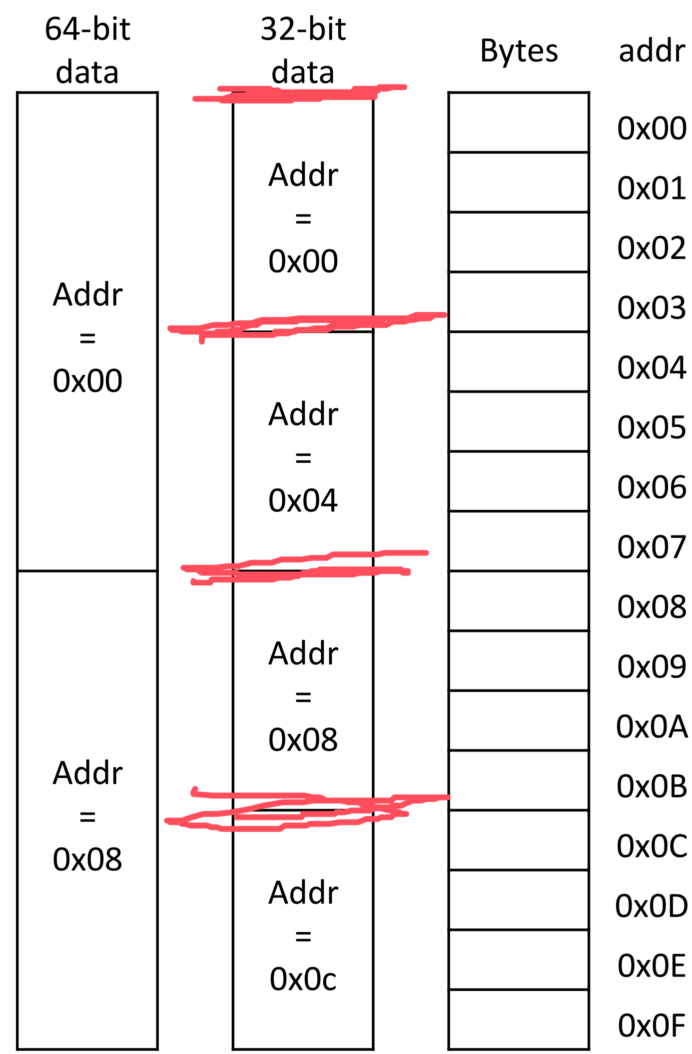
Important Notes about Byte Ordering

"Byte ordering only affects basic data units (word), composite data units like arrays or struct/class are NOT affected by byte ordering."

- How would you prove this?
  - Assume we use x86, it should be little-endian.
  - Write a piece of C/C++ code

Alignment

- Instructions that read/write data from/to memory views the memory as a series of chunks of fixed-sized data.
- Each chunk has:
  - The address (of the first byte) of that chunk of data.
  - The length of data.
- If an instruction loads 4 bytes in one go, then the address of the 4-byte data must be a multiply of 4.
- This is called memory **alignment**.
  - <u>Aligned vs unaligned access</u>.



Alignment: A Case Study

- How much memory does struct t1 occupy?

```
struct t1 {
    int a;
    char b1;
    char b2;
    char b3;
    char b4;
    int c;
};
```

4

4

```
struct t2 {
    int a;
    char b1;
    char b2;
    int c;
};
```

4

4

- How about struct t2?

Alignment: A Case Study

- If t1 and t2 are initialised as follows:

```
struct t1 a = {1, 2, 3, 4, 5, 6};
struct t2 b = {1, 2, 3, 4};
```

- How is the arrangement of these numbers in the memory?
- Try the code below for ‘a’:

```
char * c = (char *) &a;
printf("showing struct t1 a\n");
for (int i = 0; i < 12; i++) {
    printf("byte %d: %d\n", i, *c);
    c++;
}
```

Alignment: A Case Study

- The second chunk of 32 bits of memory in struct t2 only stores 2 characters, the rest of it is left empty.
- If bytes 6~9 are used to store int, the access will be unaligned

```
showing struct t1 a        showing struct t2 b
byte 0: 1                  byte 0: 1
byte 1: 0                  byte 1: 0
byte 2: 0                  byte 2: 0
byte 3: 0                  byte 3: 0
byte 4: 2                  byte 4: 2
byte 5: 3                  byte 5: 3
byte 6: 4                  byte 6: 0
byte 7: 5                  byte 7: 0
byte 8: 6                  byte 8: 4
byte 9: 0                  byte 9: 0
byte 10: 0                 byte 10: 0
byte 11: 0                 byte 11: 0
```

C C

Extended Exploration

Which of the following questions pique your curiosity?

- What's the difference between the Von-Neumann architecture and the Harvard architecture?
- How do CPUs carry out instructions so fast?
- How are function calls in C/C++ translated into assembly?
- Do I waste memory by declaring local variables in functions but then never use them?
- Why I sometimes see values like `3.150000000000004` but it should really be `3.15` when using `printf()`?
- What happens when I assign an object to another object in C++?
- How I get virus after visiting some malicious website?