# Computer Architecture
## (ENE1004)

Lec – 1: Computer Abstractions and Technology (Chapter 1)

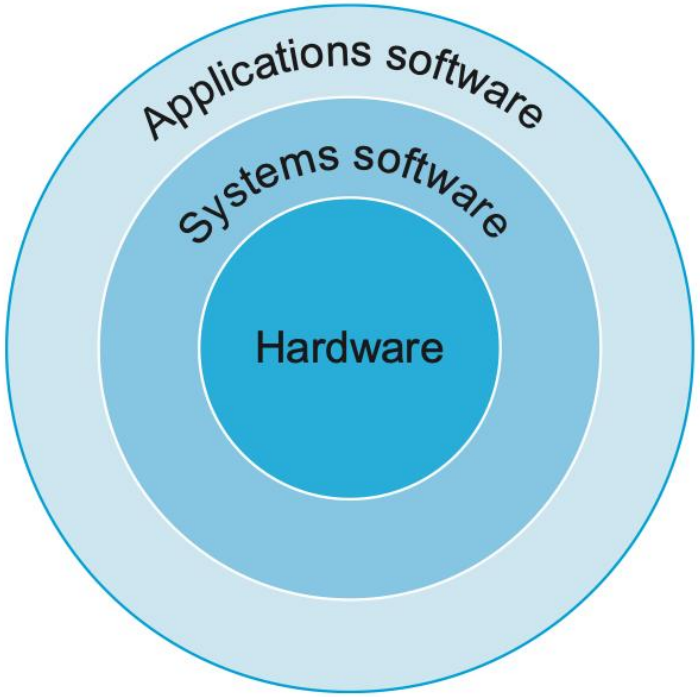
# What You Can Learn …

- By the time we end this course, you should be able to answer:
  - How are programs written in a high-level languages translated into the language of the hardware? How does the hardware execute the resulting program?
  - What is the interface between the software and the hardware? How does software instruct the hardware to perform needed functions?
  - What determines the performance of a program? How can a programmer improve the performance?
- You can improve the performance of your program if you are aware of …

*Our interest*

| Hardware or software component | How this component affects performance | Where is this topic covered? |
|---|---|---|
| Algorithm | Determines both the number of source-level statements and the number of I/O operations executed | Other books! |
| Programming language, compiler, and architecture | Determines the number of computer instructions for each source-level statement | Chapters 2 and 3 |
| Processor and memory system | Determines how fast instructions can be executed | Chapters 4, 5, and 6 |
| I/O system (hardware and operating system) | Determines how fast I/O operations may be executed | Chapters 4, 5, and 6 |

# Review: Layers of Software



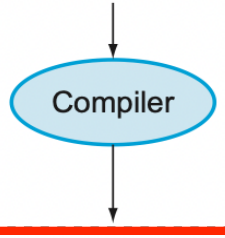

- System software helps your application software to run on hardware
- Two kinds of key system software
- (1) Operation system
  - Allocating memory and storage
  - Handling input/output operations
  - Coordinating concurrent applications
- (2) Compiler
  - Translating a program written in a high-level language to instructions that the hardware can execute
  - Your program is based on a high-level language: c = a + b;
  - Hardware works based on binary values: 100111010

# Review: from High-Level Language to Machine Language

High-level language program (in C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

- High-level language (e.g., C code)
- Text composed of words and algebraic notation



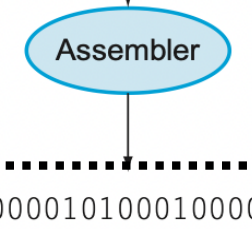

- High-level language -> Assembly language

Assembly language program (for MIPS)

```
swap:
      multi $2, $5,4
      add   $2, $4,$2
      lw    $15, 0($2)
      lw    $16, 4($2)
      sw    $16, 0($2)
      sw    $15, 4($2)
      jr    $31
```

- Assembly language (e.g., MIPS code)
- A symbolic representation of machine instructions

Assignment #1




- Assembly language -> Machine language

Binary machine language program (for MIPS)

```
00000000101000100000000100011000
00000000100000100001000000100001
10001101111000100000000000000000
10001110000100100000000000000100
```

- Machine language
- A binary representation of machine instructions

# Review: from High-Level Language to Machine Language

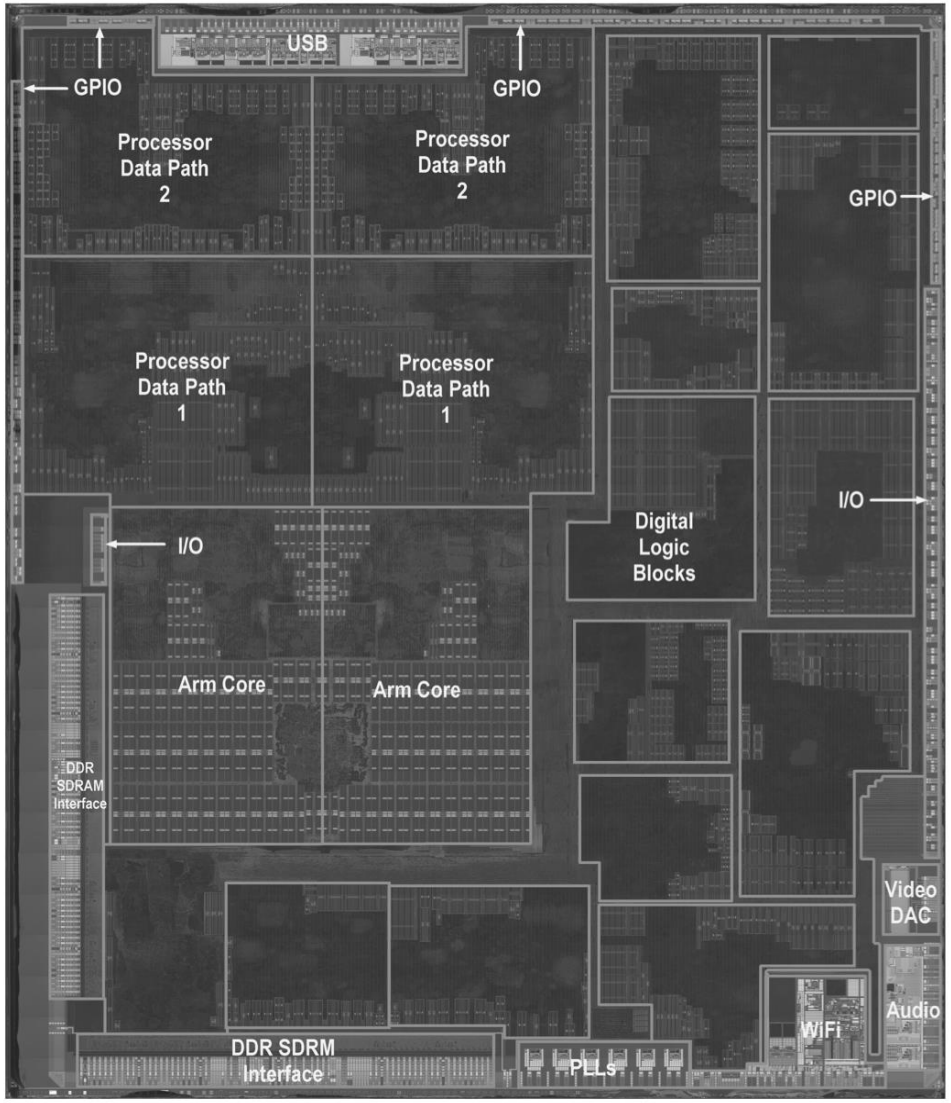
- Regarding to the conversion, there are many terms
  - Compiling, linking, building, … ; inclusion relationship, used interchangeably
  - Compilers can cut out the intermediate process and produce binary directly
- Recognition that a program can be written in a more powerful language and then translated into computer instructions was a great breakthrough
  - High-level languages offer several important benefits
    - It allows programmers to think in a more natural language, using English words and algebraic notation, resulting in programs that look much more like text
    - It improves programmer productivity; it takes less time to develop programs
    - It allows programs to be independent of the computer on which they were developed
  - Compiler has been evolving towards producing very efficient machine code optimized for the target machine
    - Study of compiler has a long history
    - CGO (https://conf.researchr.org/program/cgo-2023/program-cgo-2023/)
- Again, in this course, we are interested in how target machine executes the machine code

# Example: Device Components

- Apple iPad 2
  - Display
  - Battery
  - Logic board


- Logic board of Apple iPad 2
  - A5 chip
  - Flash memory
  - Power controller chip
  - I/O controller chip

# Example: Device Components




- Inside the A5 chip
  - Two ARM processors
  - GPU with four datapaths
  - DRAM interfaces
  - I/O controller chip
- Integrated circuit (chip, package)
- Processors (a.k.a., central processor unit, CPU)
  - (1) Datapath: performs arithmetic operations
  - (2) Control: tells datapath, memory, I/O device what to do
  - Chapter 4 will cover the two components in detail
- Interfaces & controllers
  - DRAM, I/O, etc
- Cache memory (SRAM)
  - Temporarily keeps data

# Example: Device Components


- Storage
  - Non-volatile: it keeps data after power loss
  - Hard disk drives (HDDs) or Solid state drives (SSDs) based on flash memory
  - Slow, low cost per bit
- Main memory
  - Volatile: data is lost when power is removed
  - Dynamic random access memory (DRAM)
  - Faster than storage, high cost per bit
- Cache memory
  - Volatile as well
  - Static random access memory (SRAM)
  - Faster than DRAM, more expensive than DRAM
- Chapter 5 will cover the memory systems in detail

# Review: Two Performance Metrics

- We say one computer has better performance than another; what do we mean?
  - Performance of airplanes (cruising speed vs passenger throughput)

| Airplane | Passenger capacity | Cruising range (miles) | Cruising speed (m.p.h.) | Passenger throughput (passengers × m.p.h.) |
|---|---|---|---|---|
| Boeing 777 | 375 | 4630 | 610 | 228,750 |
| Boeing 747 | 470 | 4150 | 610 | 286,700 |
| BAC/Sud Concorde | 132 | 4000 | 1350 | 178,200 |
| Douglas DC-8-50 | 146 | 8720 | 544 | 79,424 |

- Response time (execution time) - s, ms, us, ns, …
  - The time between the start and the completion of a task
  - Important to individual users
  - Used in evaluating the performance of embedded and laptop computers
- Throughput (bandwidth) – MB/s, GB/s, …
  - The total amount of work done in a given time
  - Important to datacenter managers
  - Used in evaluating the performance of servers

# Review: Measuring Execution Time

- Relative Performance
  - If computer A runs a program in 10 seconds and computer B runs the same program in 15 se conds, how much faster is A than B?
  - A is *n* times faster than B if $execution\_time_B / execution\_time_A = n$
  - 15/10 = 1.5; so, A is 1.5 times faster than B

- Two types of execution time
  - (1) Elapsed time
    - The total time to complete a task
    - It includes disk accesses, memory accesses, I/O activities, operating system overhead
  - (2) CPU time
    - The time the CPU spends computing for this task
    - It does not include time spent for other activities
  - Once you can analyze both the elapsed time & CPU time of a task, you can find the perfor mance bottleneck and thus where to optimize