
Computer Systems

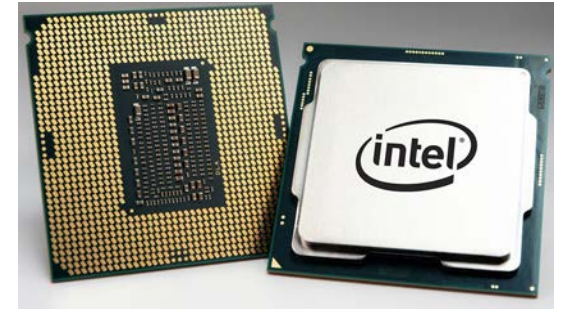
Dae Hyun Kim

EECS

Washington State University

Primary Hardware Components

- CPU (Central Processing Unit)
- Main memory
- Disk drives
 - Hard disk drive (HDD)
 - Solid state drive (SSD)
- Input
 - Keyboard, mouse, ...
- Output
 - Monitor, printer, ...



Primary Software Components

- Operating System
 - Compiler
 - Linker
 - Loader
 - Application
-

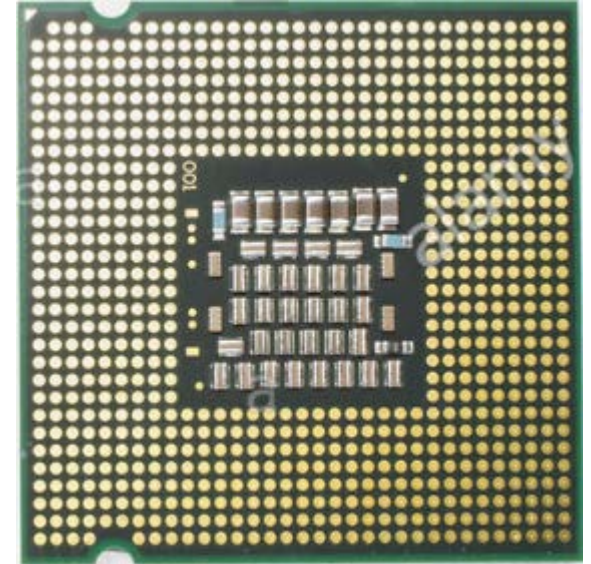
How Each Component Works



Hardware – CPU

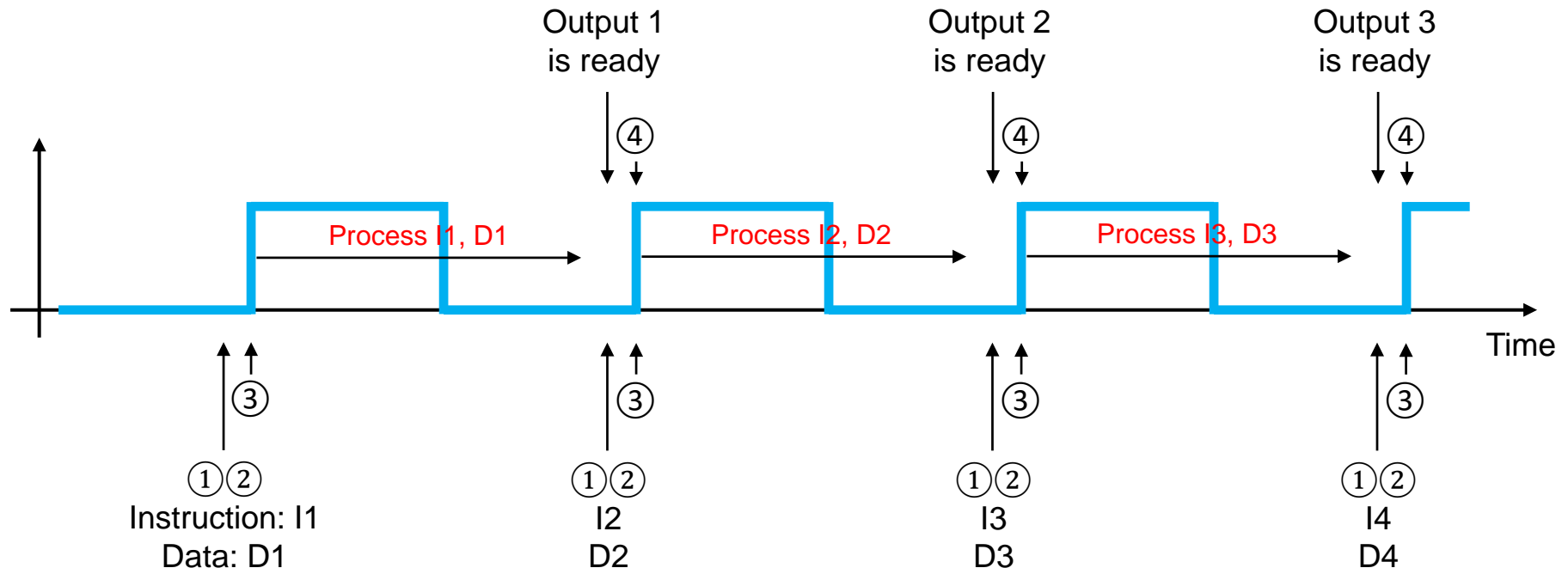
○ Pins

- Input (signals going into the CPU)
- Output (signals coming out from the CPU)
- Power (VDD, GND)
 - VDD: High voltage (e.g., 1V)
 - GND: Low voltage (e.g., 0V)



Hardware – CPU

- A very simple, conceptual model for CPU operations
 1. Place an input data on its data signal pins (input).
 2. Place an instruction on its instruction signal pins (input).
 3. Provide a clock (then the CPU will run the instruction).
 4. Read the data placed on its data signal pins (output).



Hardware – CPU

○ How do the instructions and data look like?

- 10010001000101010101010001000011

○ What is it?

- | | | | | | |
|---------------|---------|--------------|----------------------------|--------------------|-------------------------|
| 1 | 0010001 | 00 | 010101010101 | 00010 | 00011 |
| ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| 64bit
data | ADD | Some
code | Immediate
(value 1,365) | Source
Register | Destination
Register |

- Add 1,365 and the value stored in the source register.
 - (A register is a temporary memory space. We will study that later.)
- Store the result in the destination register.

○ If an instruction is given, how can we find out what it means?

- 00000100110101111110001110000011
-

Hardware – CPU

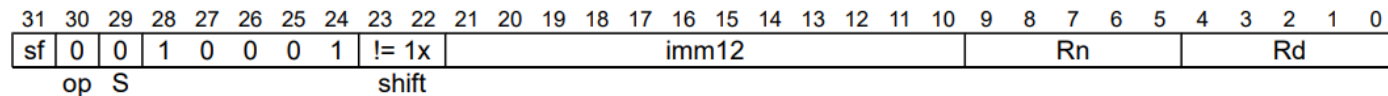
○ Instruction Set Architecture (ISA)

- The ISA of a CPU defines
 - All the instructions the CPU can perform (addition, subtraction, ...)
 - The formats of the instructions and their machine codes (0110001...)

ADD (immediate)

Add (immediate) adds a register value and an optionally-shifted immediate value, and writes the result to the destination register.

This instruction is used by the alias [MOV \(to/from SP\)](#).



32-bit (sf == 0)

```
ADD <Wd|WSP>, <Wn|WSP>, #<imm>{, <shift>}
```

64-bit (sf == 1)

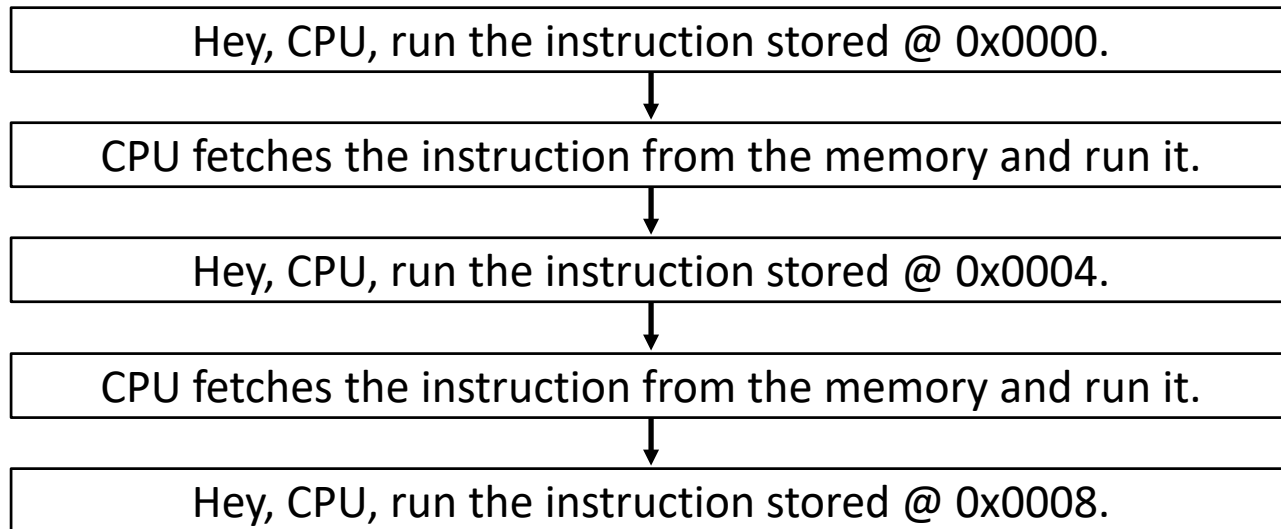
```
ADD <Xd|SP>, <Xn|SP>, #<imm>{, <shift>}
```

```
integer d = UInt(Rd);
integer n = UInt(Rn);
integer datasize = if sf == '1' then 64 else 32;
bits(datasize) imm;

case shift of
  when '00' imm = ZeroExtend(imm12, datasize);
  when '01' imm = ZeroExtend(imm12:Zeros(12), datasize);
  when '10' SEE "ADDG, SUBG";
  when '11' ReservedValue();
```


Hardware – CPU

- Yes, we can run a program like that.
- However, we want to automate the whole process. How?
 - Instructions, data: Store them in memory.
 - Tell the CPU to fetch the instruction stored at a certain memory address and run it.
 - Then, tell the CPU where the next instruction is.



Hardware – Memory (DRAM)

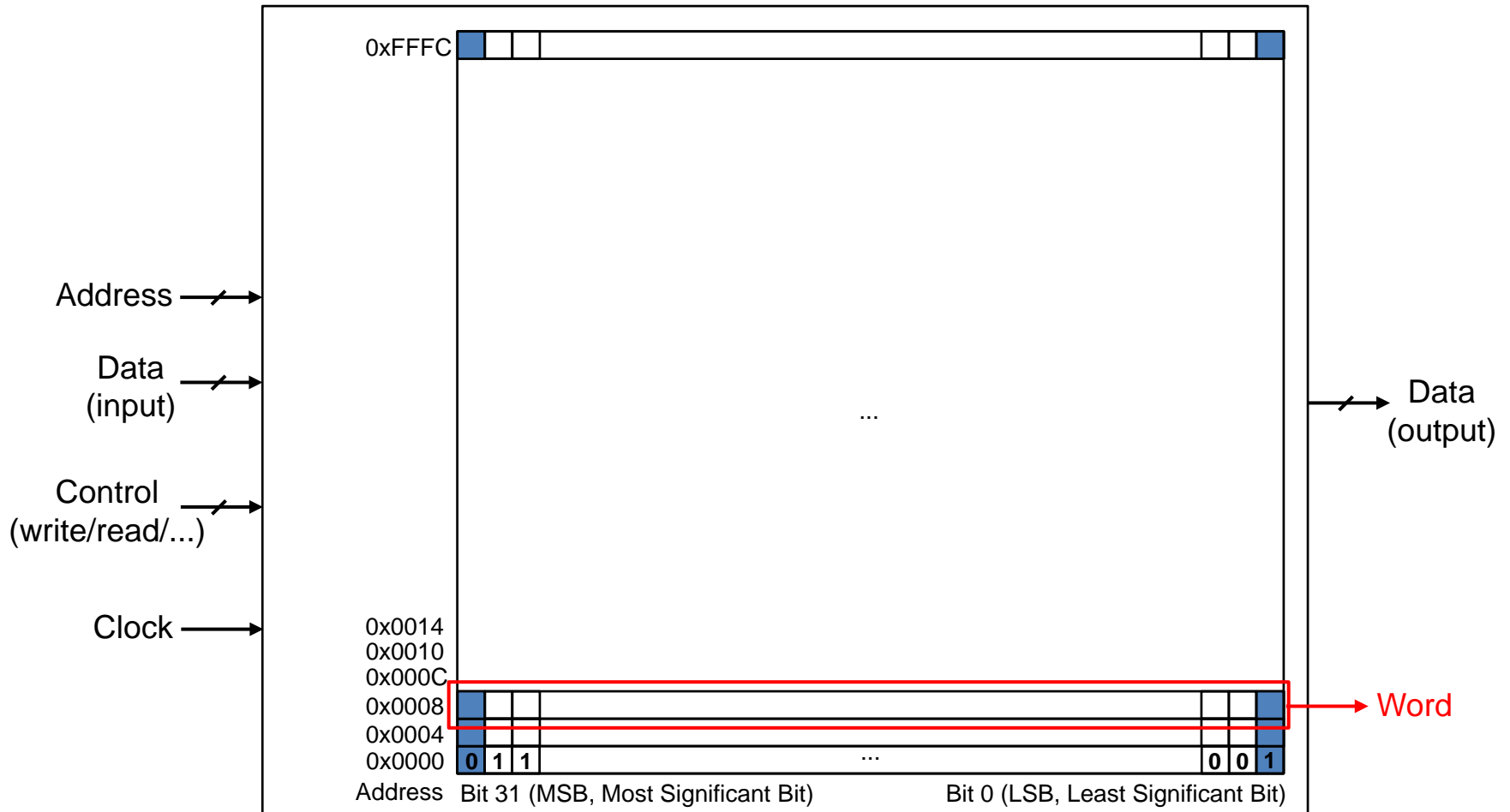
○ Pins

- Input (signals going into the memory)
- Output (signals coming out from the memory)
- Power (VDD, GND)
 - VDD: High voltage (e.g., 1V)
 - GND: Low voltage (e.g., 0V)



Hardware – Memory

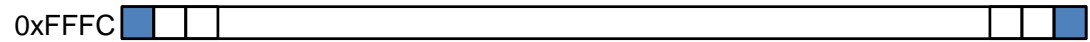
○ Abstract model



Hardware – Memory

○ How to calculate the size of a given memory chip

- Word (data) width: w (bits)



- # words: k

- Size: $k \cdot w$ (bits) = $\frac{k \cdot w}{8}$ (Bytes)

- Example

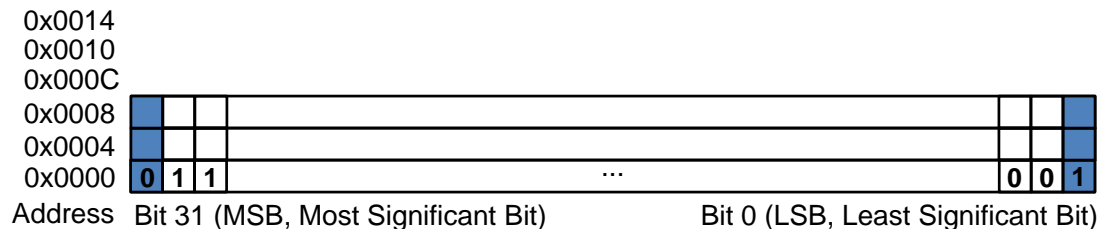
- $w = 32$

- $k = \frac{2^{16}}{4} = 2^{14}$

- $s = 2^{19}$ bits = 2^{16} Bytes = 2^6 KBytes = 32 KBytes ...

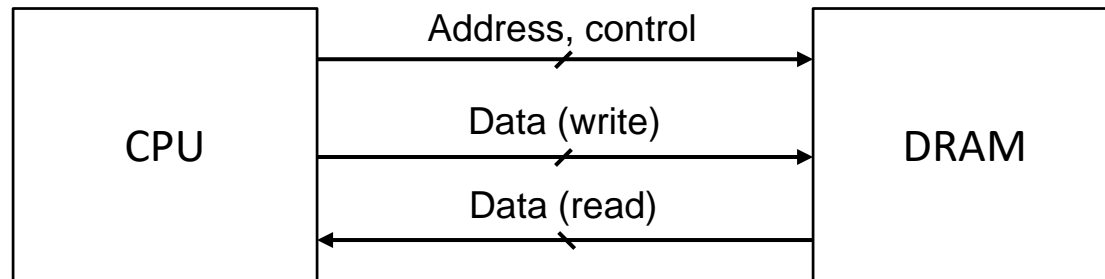
○ If the address bus has 32 bits, what's the max. memory capacity it can support?

- Answer: 4GB

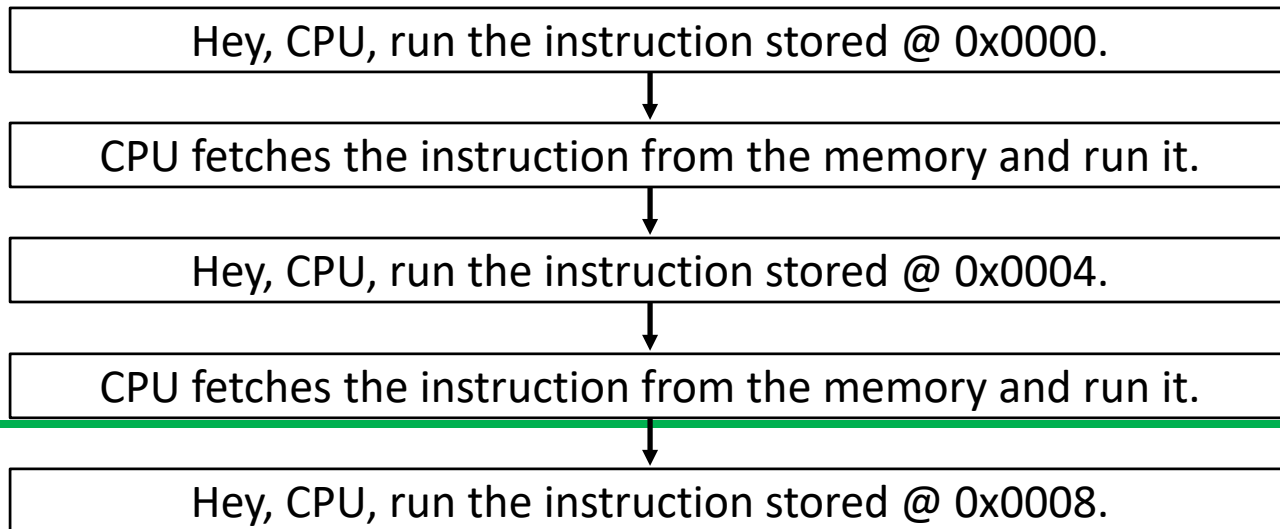


Hardware – CPU and Memory (DRAM)

- Now, we know how a CPU chip and a memory chip look like.
- Let's connect them.



- Finally, we can automatically run a program.
 - Load your program and data into the main memory.
 - Then



Hardware – CPU and Memory (DRAM)

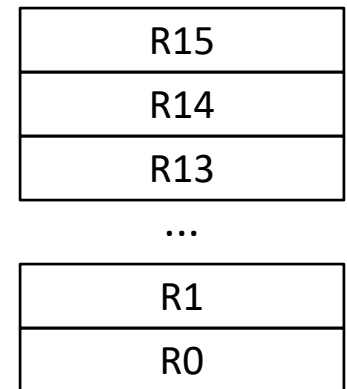
- CPUs are fast (high-speed).
 - Running an instruction takes one clock cycle (~300ps).
 - Memory is slow (low-speed).
 - Loading a data takes several hundreds of clock cycles.

 - Question
 - Why can't we integrate a large amount of memory in a CPU?

 - Solution
 - Let's have a small, but fast memory inside a CPU.
-

Hardware – CPU

- Register File (RF)
 - A register is a memory space that can store a word.
 - A register file is a set of registers.
- The input/output behavior of an RF is very similar to that of the abstract model of a general memory chip.
- Since RF should be very fast, RF has only a few registers (16~32).



Hardware – CPU

○ Comparison with vs. without RF

- Instructions: Add data1 (stored @ memory address 0x4000), data2 (@ 0x4004), data3 (@ 0x4008), and data4 (@ 0x400C), then store the result @ 0x5000.
- Let's assume that a memory instruction takes 200 clock cycles.

Load data @ 0x4000	+200 cycles
Load data @ 0x4004	+200
Add the two	+1
Store @ 0x8000	+200

Load data @ 0x4008	+200
Load data @ 0x400C	+200
Add the two	+1
Store @ 0x8004	+200

Load data @ 0x8000	+200
Load data @ 0x8004	+200
Add the two	+1
Store @ 0x5000	+200

Load data @ 0x4000 to R1	+200
Load data @ 0x4004 to R2	+200
Add the two and store to R3	+1

Load data @ 0x4008 to R1	+200
Load data @ 0x400C to R2	+200
Add the two and store to R4	+1

Add R3, R4 and store to R1	+1
----------------------------	----

Store R5 @ 0x5000	+200
-------------------	------

Without RF (1,803 cycles)

With RF (1,003 cycles)

Hardware – CPU and Memory (DRAM)

○ Summary

- A CPU has a Register File (RF) consisting of 16~32 registers.
 - To perform a task in a CPU, you should
 - Load the data from the memory
 - Store it in the RF
 - Perform a task (addition, subtraction, ...) on the registers
 - Store the result back to the RF
 - Store the result back to the memory (if necessary)
 - RFs are small, RFs should be used only for temporary space.
 - All the instructions and data should be stored in the main memory.
 - We will study this later.
 - For now, we will assume that a program and its data are stored in the memory.
 - A CPU has an instruction set architecture (ISA).
 - We will study this later.
 - How to run a program automatically
 - Hey, CPU, the next instruction is @ X. Do it. Then, the next instruction is @ Y. ...
-

Software – Application Programming

○ Let's make a small program that adds two values stored @ 0x4000 and @ 0x4004 and store the result @ 0x5000.

○ How would you do it?

- Make a pseudo code.

Load data @ 0x4000 to R1
Load data @0x4004 to R2
Add the two and store to R3
Store R3 @ 0x5000

- Translate it into a machine code. (You need an ISA reference for this.)

Load data @ 0x4000 to R1
Load data @0x4004 to R2
Add the two and store to R3
Store R3 @ 0x5000




```
00011000100110110001010101111000
00011000100110101001010101001011
00010111110000011010101111111000
01100100001011100000111100000000
```

○ Now, it is very error-prone, inefficient, and hard to maintain.

Software – Assembly

- An assembly language is a human-friendly programming language that makes the machine code programming much easier.
- An assembly code has a kind of one-to-one correspondence with a machine code.

Assembly		Machine code
LDR R1, #0x4000 (Load data @ 0x4000 to R1)		00011000100110110001010101111000

- The translator (from an assembly code to a machine code) is an **assembler**.
 - The backward translator is a disassembler.
-

Software – Application Programming

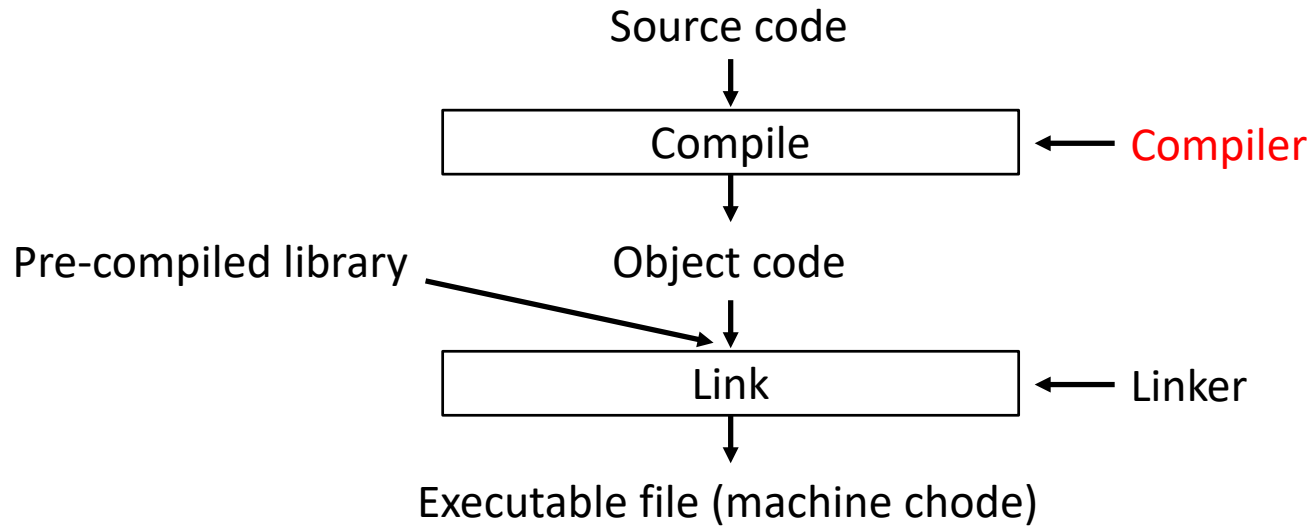
- Yes, coding with an assembly language looks much easier than coding directly with the machine language.
 - However, should we need to remember all the instructions and the syntax?
 - Yes, we do...
 - But we can remember only 20~30 instructions.
 - ADD (addition)
 - SUB (subtraction)
 - MUL (multiplication)
 - AND (logical AND)
 - OR (logical OR)
 - EOR (logical exclusive-OR)
 - LSR/LSL (logical shift right/left)
 - MOV (move data)
 - CMP (compare)
 - LDR/STR (load/store)
-

Software – High-Level Programming Languages and Compilers

- Unfortunately, assembly programming is still extremely hard, time-consuming, error-prone, unportable, and human-unfriendly.
 - High-level programming languages (C/C++, ...)
 - Easy
 - Efficient
 - Human-friendly
 - High-performance
 - Portable (independent of CPUs)
 - Different CPUs have different ISAs.
 - Assembly languages are dependent on CPUs.
-

Software – High-Level Programming Languages and Compilers

- Application programming



Software – Operating System

- If you run a single program, managing the program is easy. Just load your program to the main memory starting from 0x0000 and run it from 0x0000.

0x01C8	B ...
0x01C4	BNE ...
	...
0x0008	ADD ...
0x0004	LDR ...
0x0000	LDR ...

Software – Operating System

- What if you want to
 - run multiple programs at the same time?
 - manage input/output devices
 - manage the main memory dynamically (on-the-fly)
 - ...

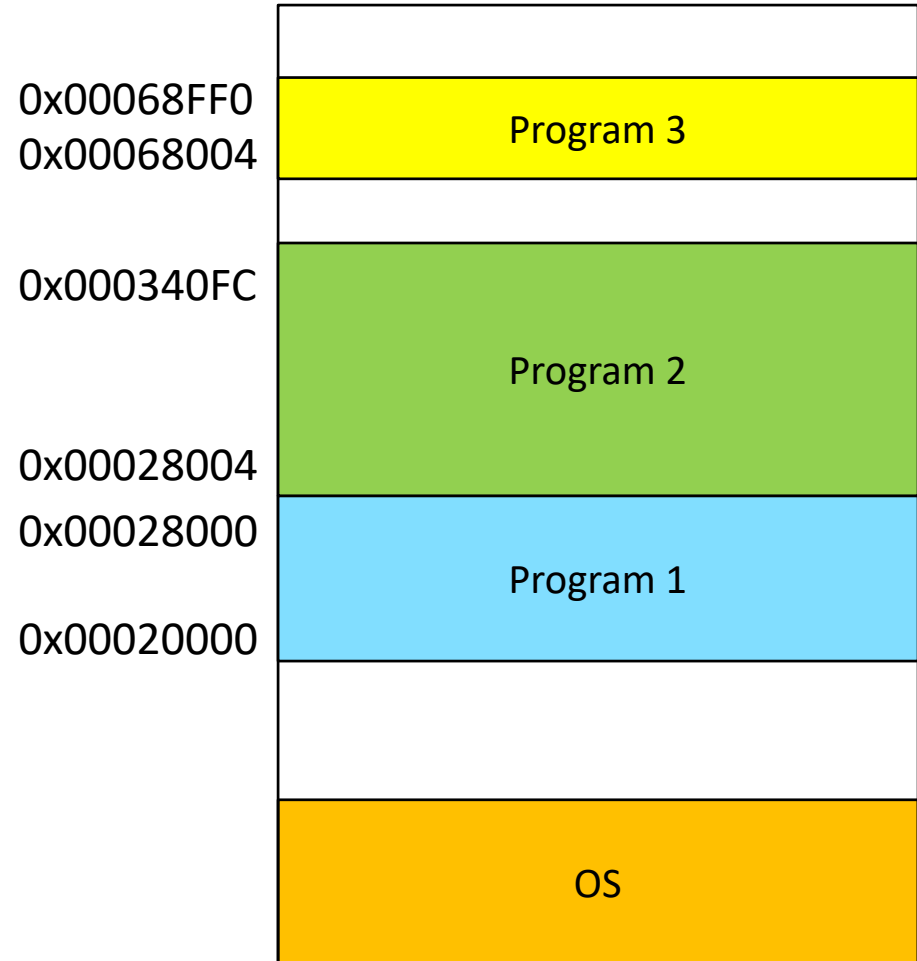
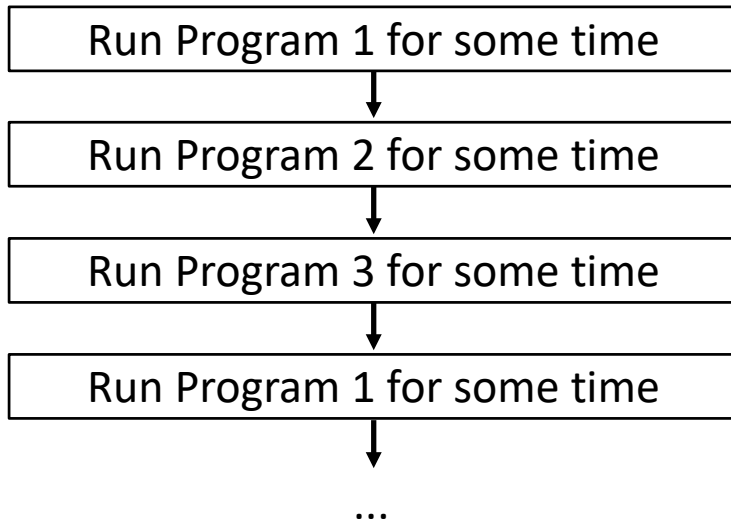
 - Operating system (OS)
 - Process management
 - Memory management
 - Storage management
 - I/O device management
 - File system
 - Security
 - ...
-

Software – Operating System

- An I/O device has a special communication protocol.
 - OS has device drivers.
 - Application programmers don't need to develop device drivers.
 - Multiple programs ask for resources (storage, memory, CPU, ...).
 - OS manages them intelligently to maximize the system performance, usability, etc.
 - How does an OS manage the main memory?
-

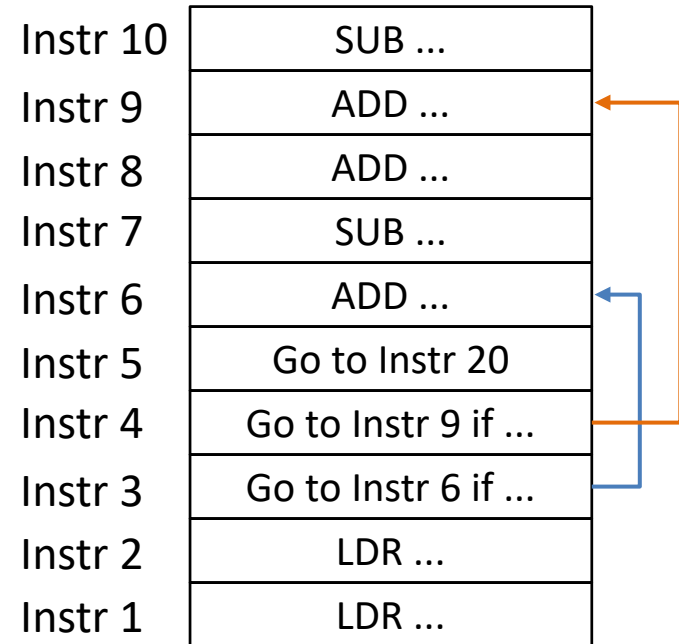
Software – Operating System

- Running multiple programs
 - Load them into the memory.
 - Then



Software – Operating System

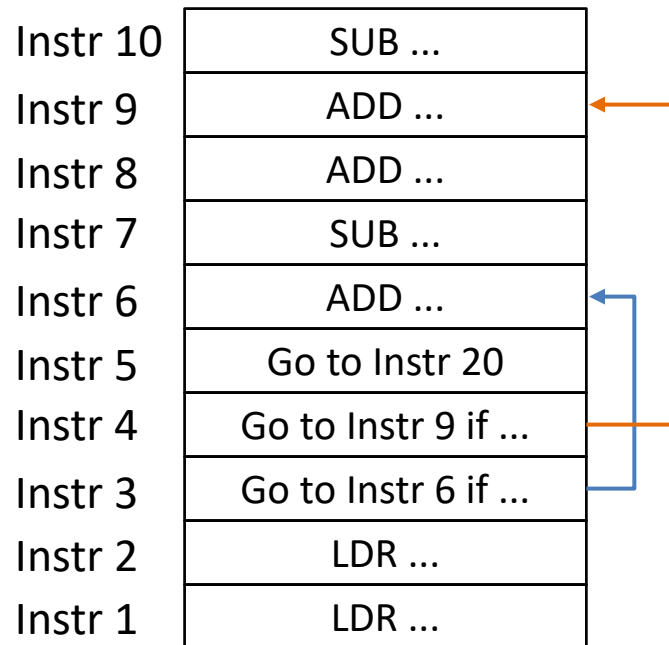
- An issue when you run multiple programs
 - Some codes use relative memory addresses.
 - If Instr 1 is located at 0x0000
 - Instr 6 is @ 0x0014
 - Instr 9 is @ 0x0020
 - If Instr 1 is located at 0x4000
 - Instr 6 is @ 0x4014
 - Instr 9 is @ 0x4020
 - The relative memory addresses are determined when the program is loaded into the memory.



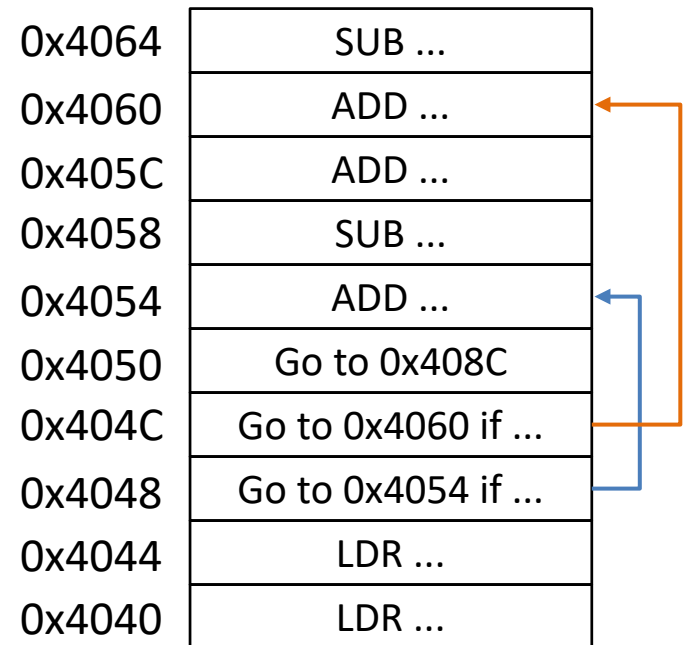
Software – Loader

○ Solution

- The compiled machine code has relative memory addresses.
- When the program is loaded into the memory, fill in the relative addresses.
 - Who does it? **Loader**



Compiled code



Loaded code

Software

○ Summary

- Machine language: 00011000001010101101100001010101
 - Assembly language: ADD R1, R2, R3
 - High-level programming language: $a = b + c$;
 - Compiler
 - Linker
 - Loader
 - Operating system
-