EE3220 System-on-Chip Design

Lecture Note 2

Introduction to Modern Microprocessor

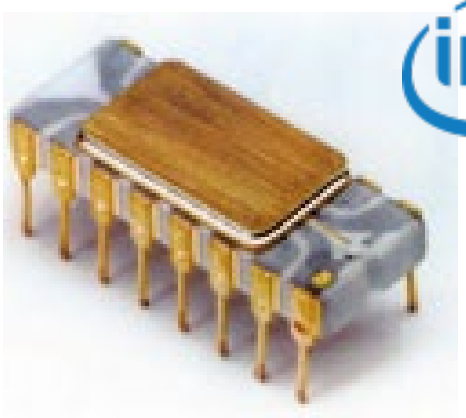
CityU Department of Electrical Engineering
香港城市大學
City University of Hong Kong

Outline

- Introduction to the First Microprocessor
- Concept of Computation
- Microcontroller vs. Microprocessor
- ARM Chip & Die
- Stored Program Concept
- RISC Philosophy
- Instruction Set Architecture
- Pipelined Instruction Execution
- Compilation and Assembling
- ARM Processor Family

The First Microprocessor

- In 1971 Intel introduces the first commercially available microprocessor – the 4004
- 4-bit data bus, 45 instructions
- Required many support chips to build a functioning system
- 2,300 transistors on one IC



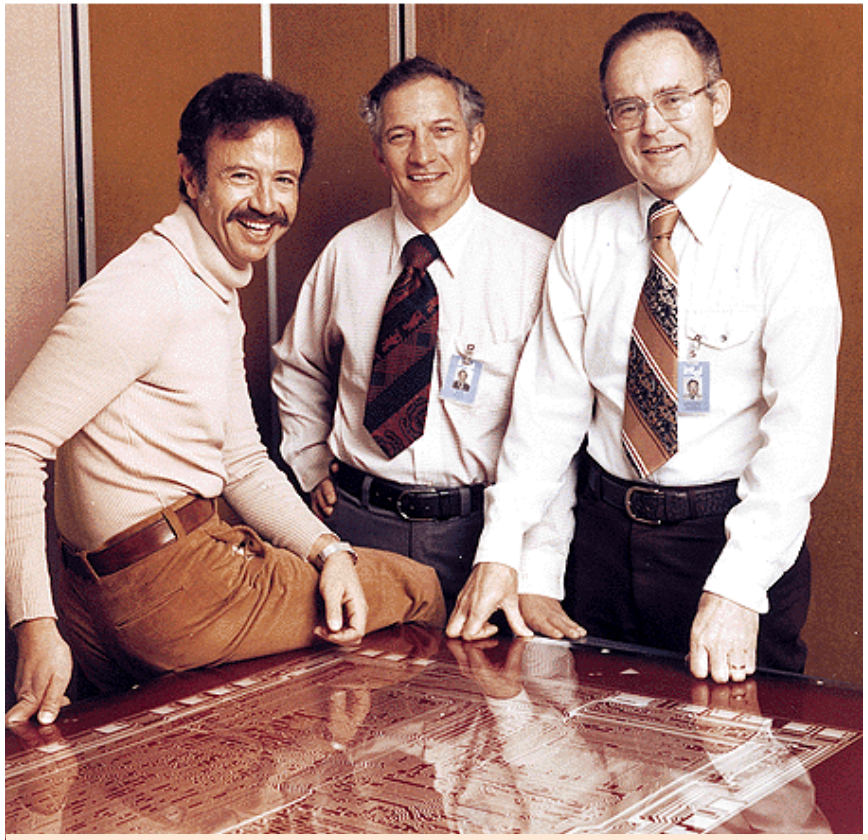
Since 1970s, we have been observing great developments in both computer architecture and integrated circuit fabrication.

Microprocessors become more powerful but at the same cheaper!

Moore's Law

The number of transistors on a chip will roughly double every two years.

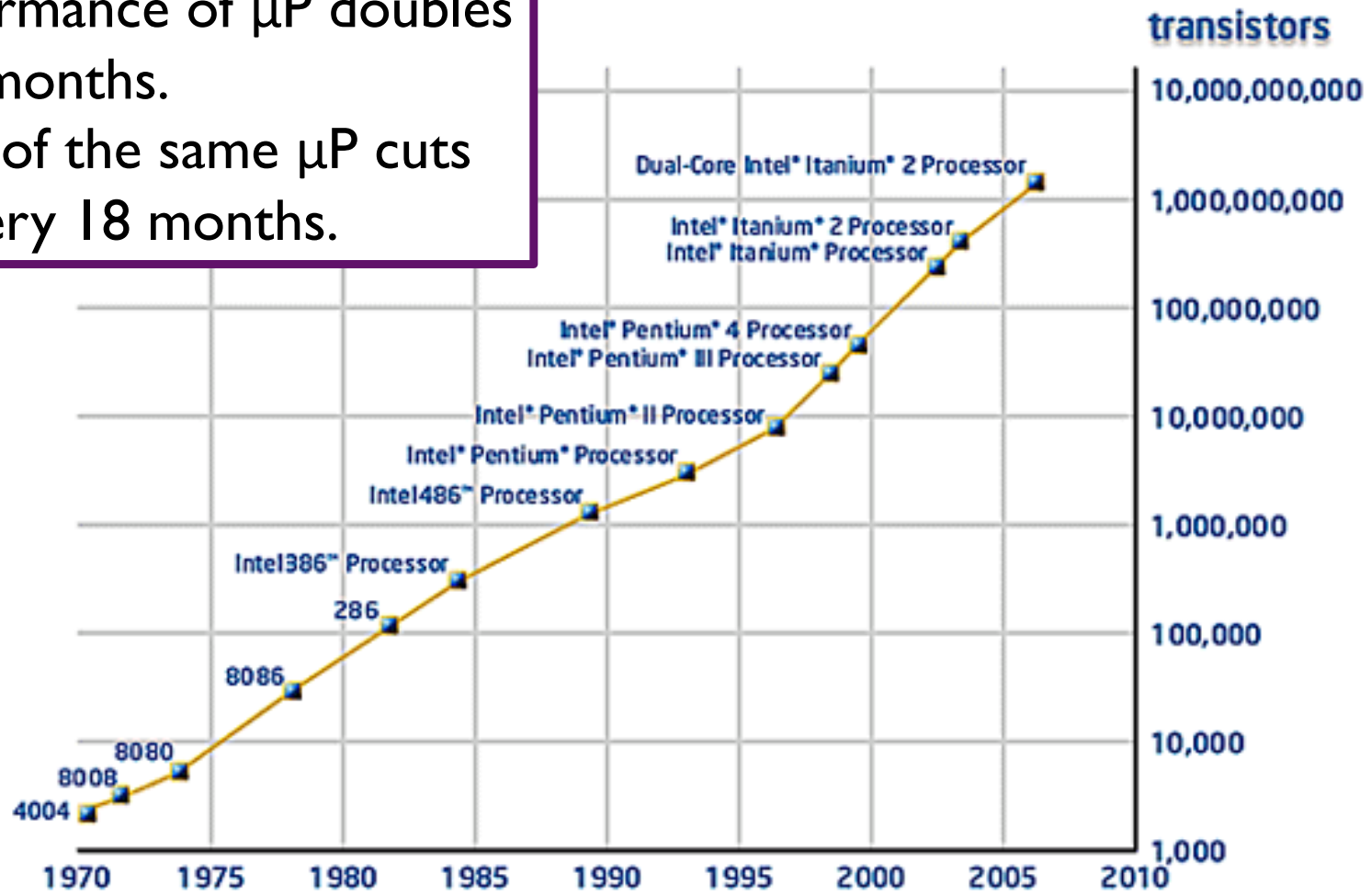
- Gordon Moore's prediction from 1965!
- He has been proved right so far...



Founders of Intel: Andy Grove, Robert Noyce and Gordon Moore

Moore's Law (Cont')

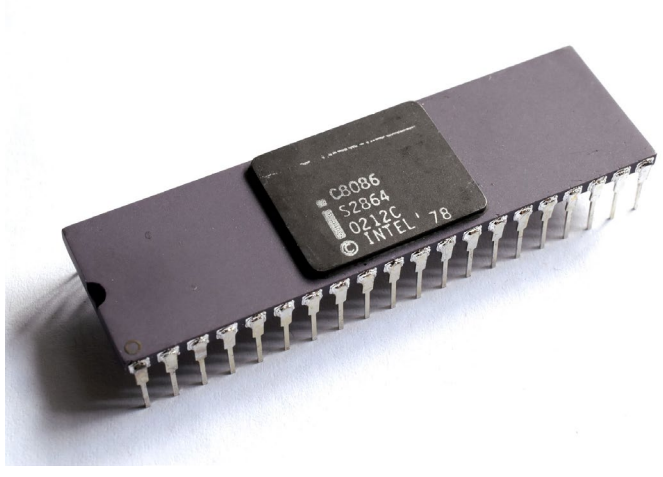
The performance of µP doubles every 18 months.
The price of the same µP cuts by half every 18 months.



CityU Department of Electrical Engineering
香港城市大學 City University of Hong Kong

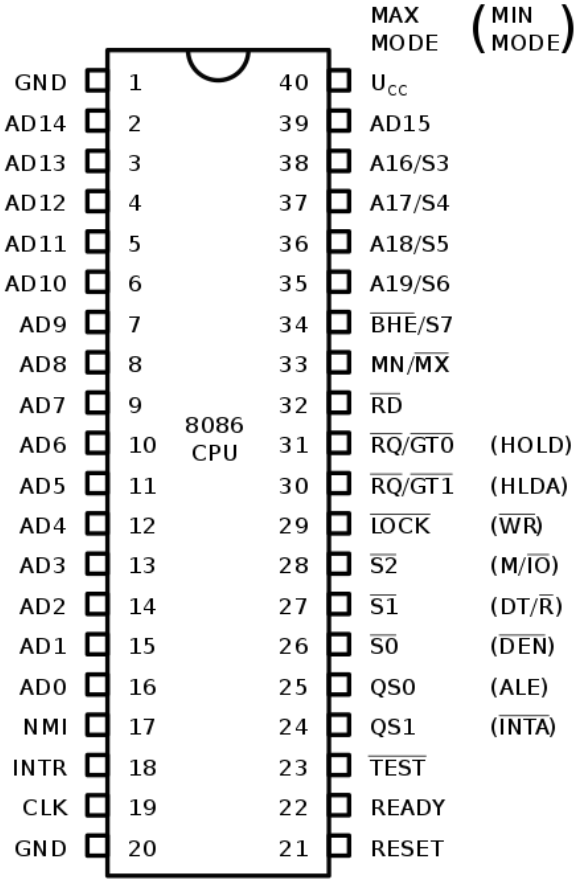
Classic: Intel 8086 Processor & Chip

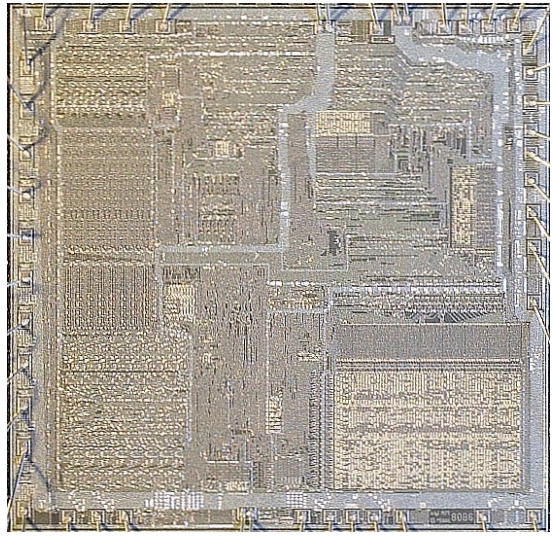
- Launched in 1978, 10MHz, 16-bits, 3um.
- 16 general registers
- You can learn more from the computer architecture course in EE department.



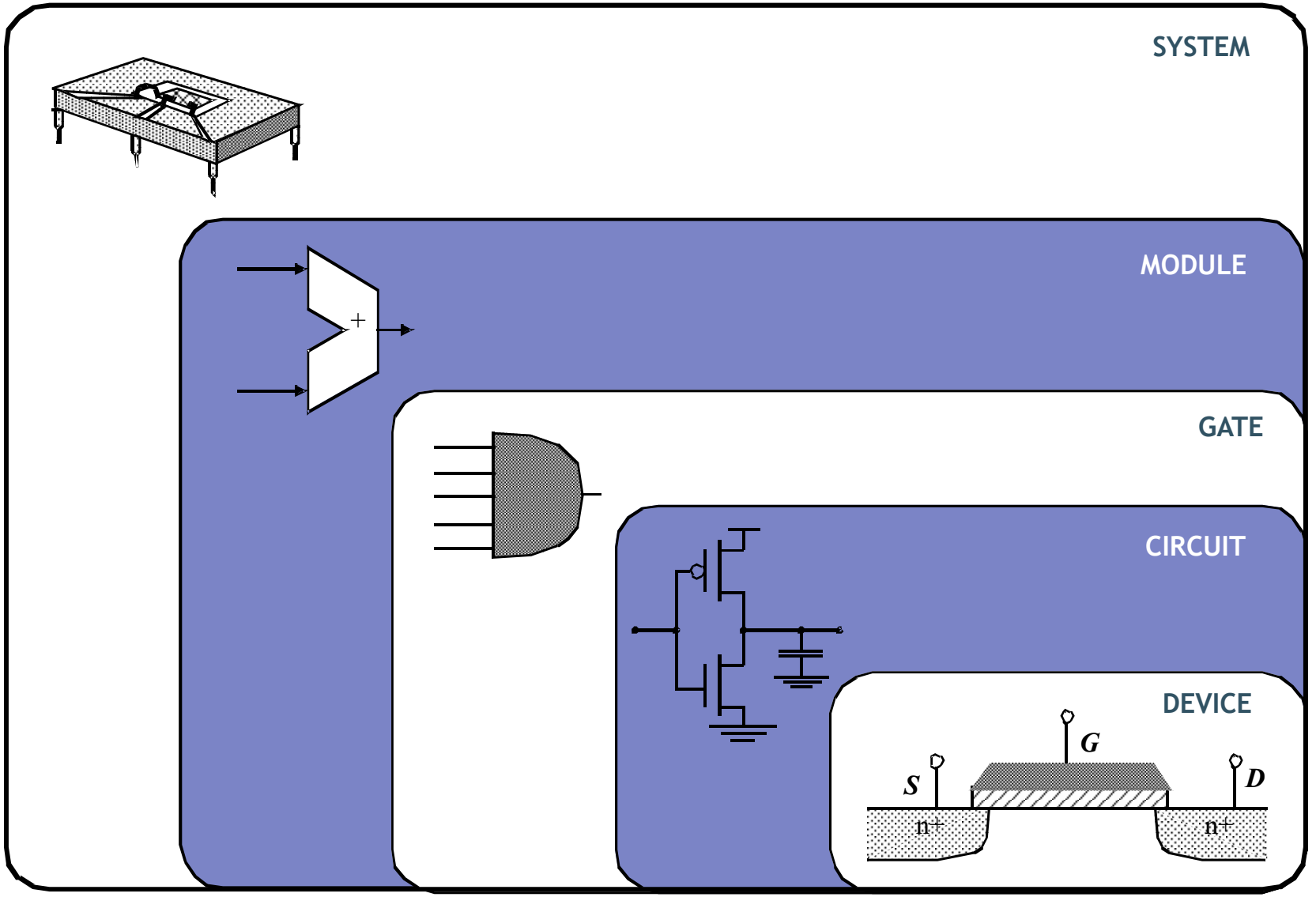
Intel 8086 registers

1_9 1_8 1_7 1_6 1_5 1_4 1_3 1_2 1_1 1_0 0_9 0_8 0_7 0_6 0_5 0_4 0_3 0_2 0_1 0_0 *(bit position)*			
Main registers			
	AH	AL	**AX** (primary accumulator)
0 0 0 0	BH	BL	**BX** (base, accumulator)
	CH	CL	**CX** (counter, accumulator)
	DH	DL	**DX** (accumulator, extended acc)
Index registers			
0 0 0 0	SI		**S**ource **I**ndex
0 0 0 0	DI		**D**estination **I**ndex
0 0 0 0	BP		**B**ase **P**ointer
0 0 0 0	SP		**S**tack **P**ointer
Program counter			
0 0 0 0	IP		**I**nstruction **P**ointer
Segment registers			
CS		0 0 0 0	**C**ode **S**egment
DS		0 0 0 0	**D**ata **S**egment
ES		0 0 0 0	**E**xtra **S**egment
SS		0 0 0 0	**S**tack **S**egment
Status register			
- - - - O D I T S Z - A - P - C			Flags



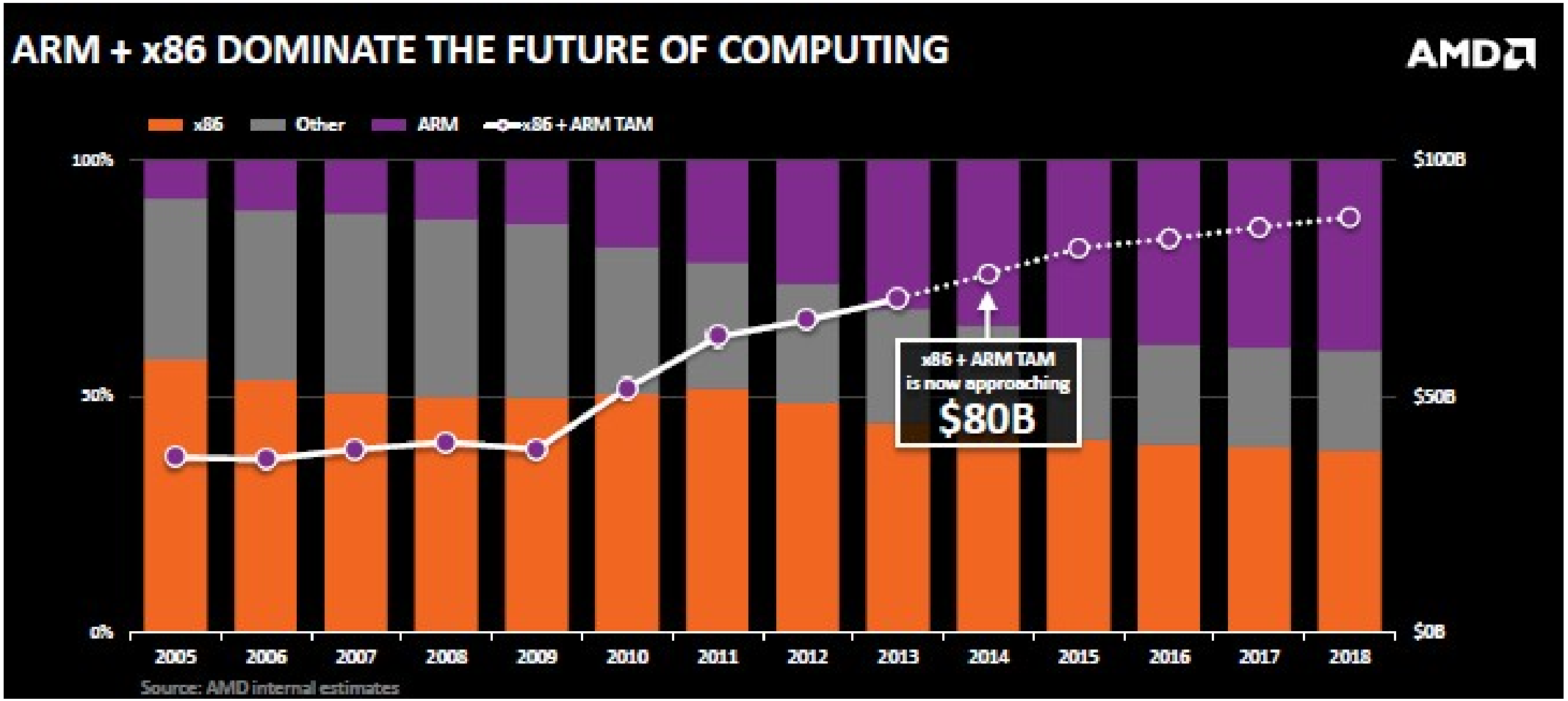


Design Abstraction Levels



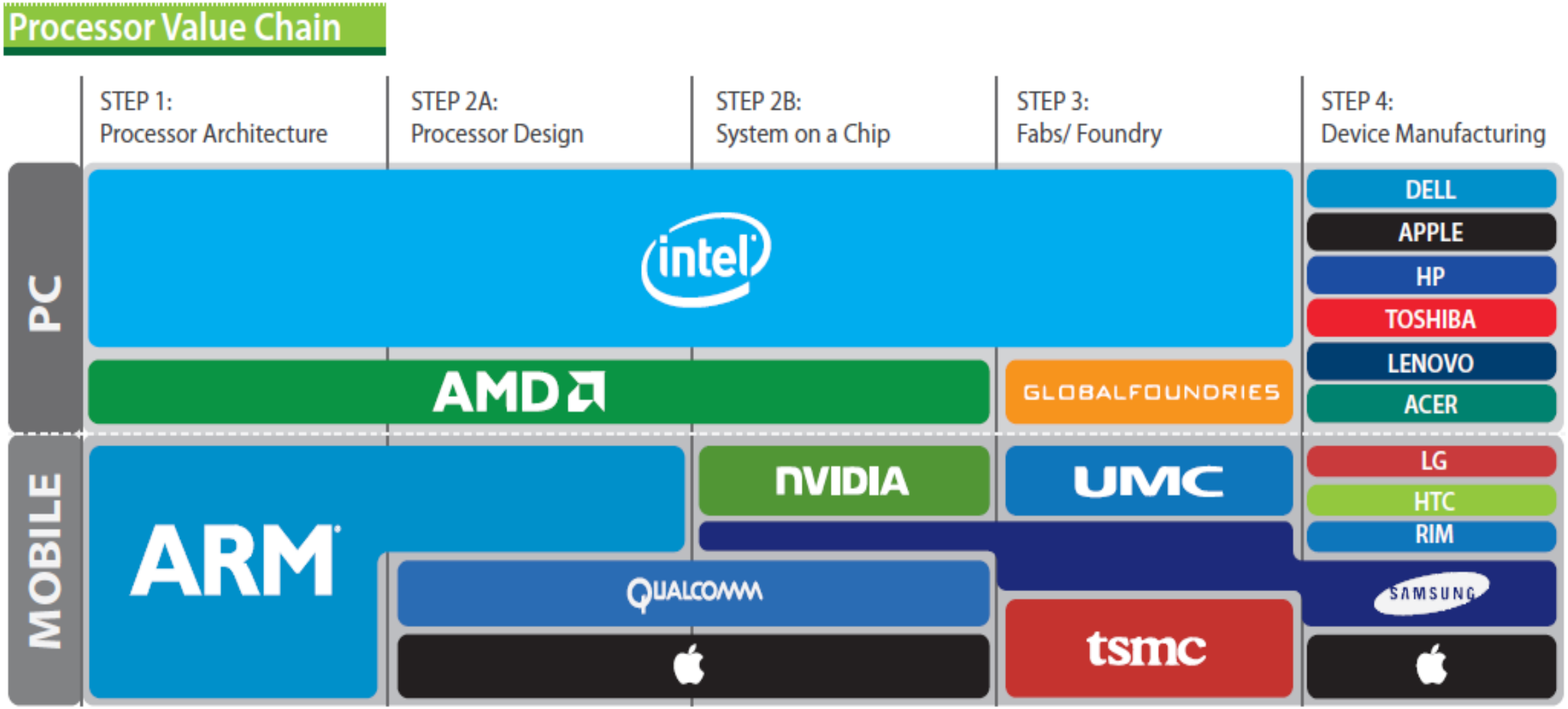
x86 vs. ARM

- Mobile computing market is continuously growing.



Department of Electrical Engineering
香港城市大學
City University of Hong Kong

Mobile Processor vs. PC Processor



CityU Department of Electrical Engineering
香港城市大學 City University of Hong Kong

What is a Computer?

- A computer is a machine that can perform simple calculations
- But a computer can also process algorithms where it ...
  - performs a sequence of calculations;
  - makes decisions based on the results of calculations; and
  - repeats the sequence if wanted.

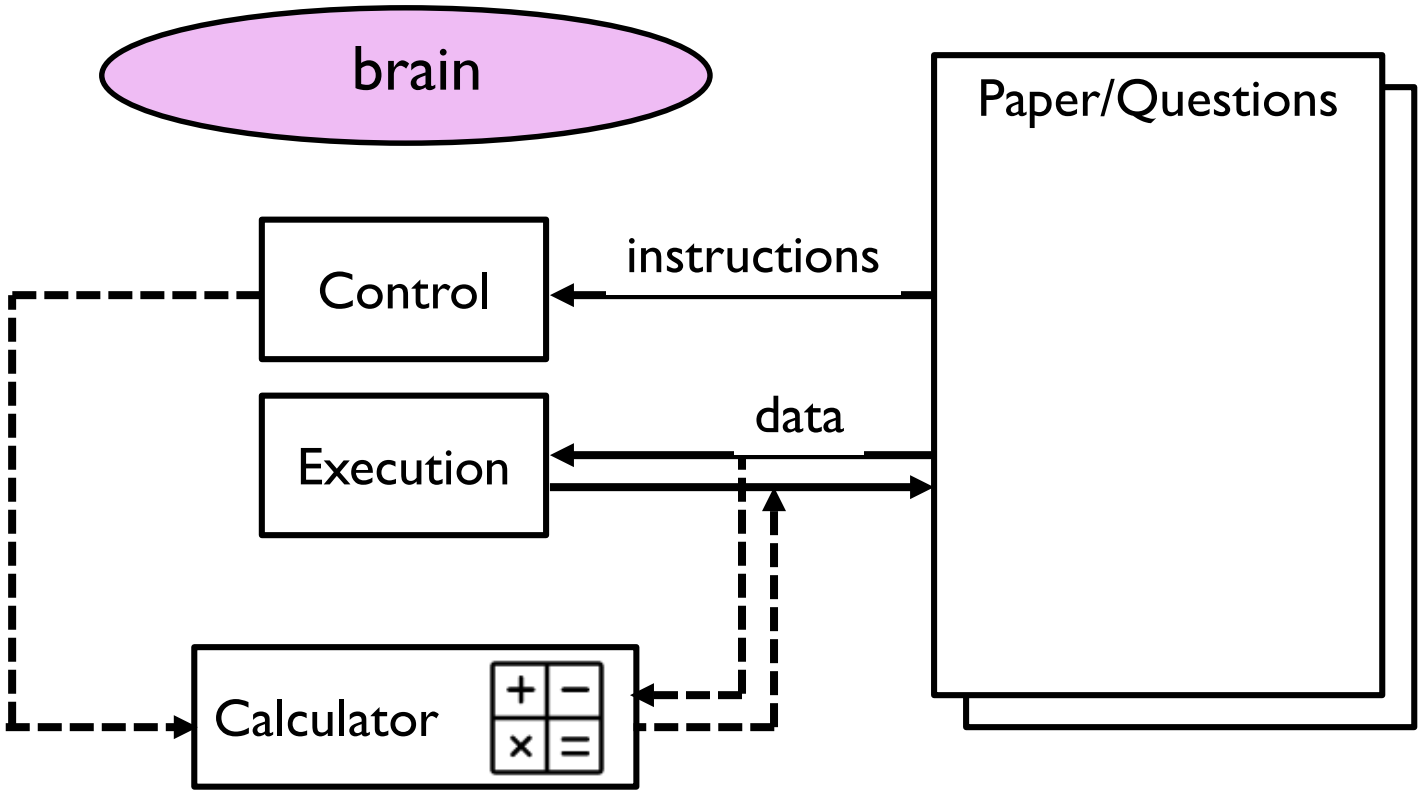
Math Quiz!

1. 1 + 2 + 3 + 4 + 5 = ?
2. 16 x 8 + 29 = ?
3. A circle has a radius of 5 cm, what is its area?

Let's think: how did you answer the above questions?

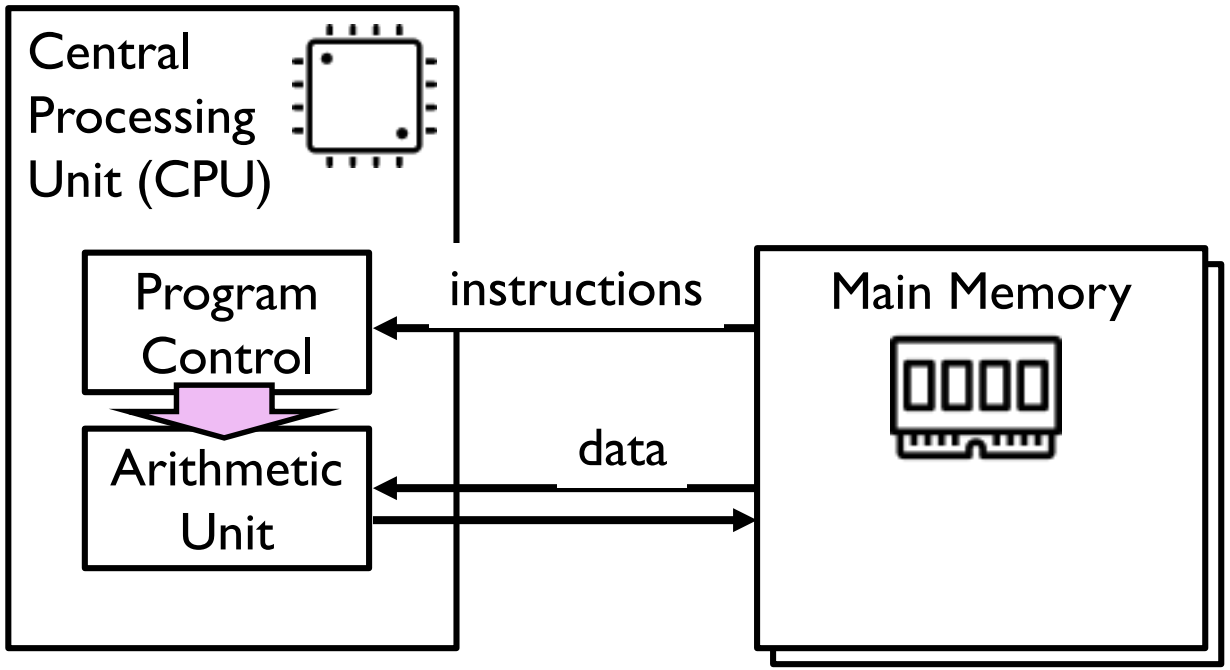
Concept of Computation

Let's start with the way we handle computations...



Concept of Computation (Cont')

How about machine computations?

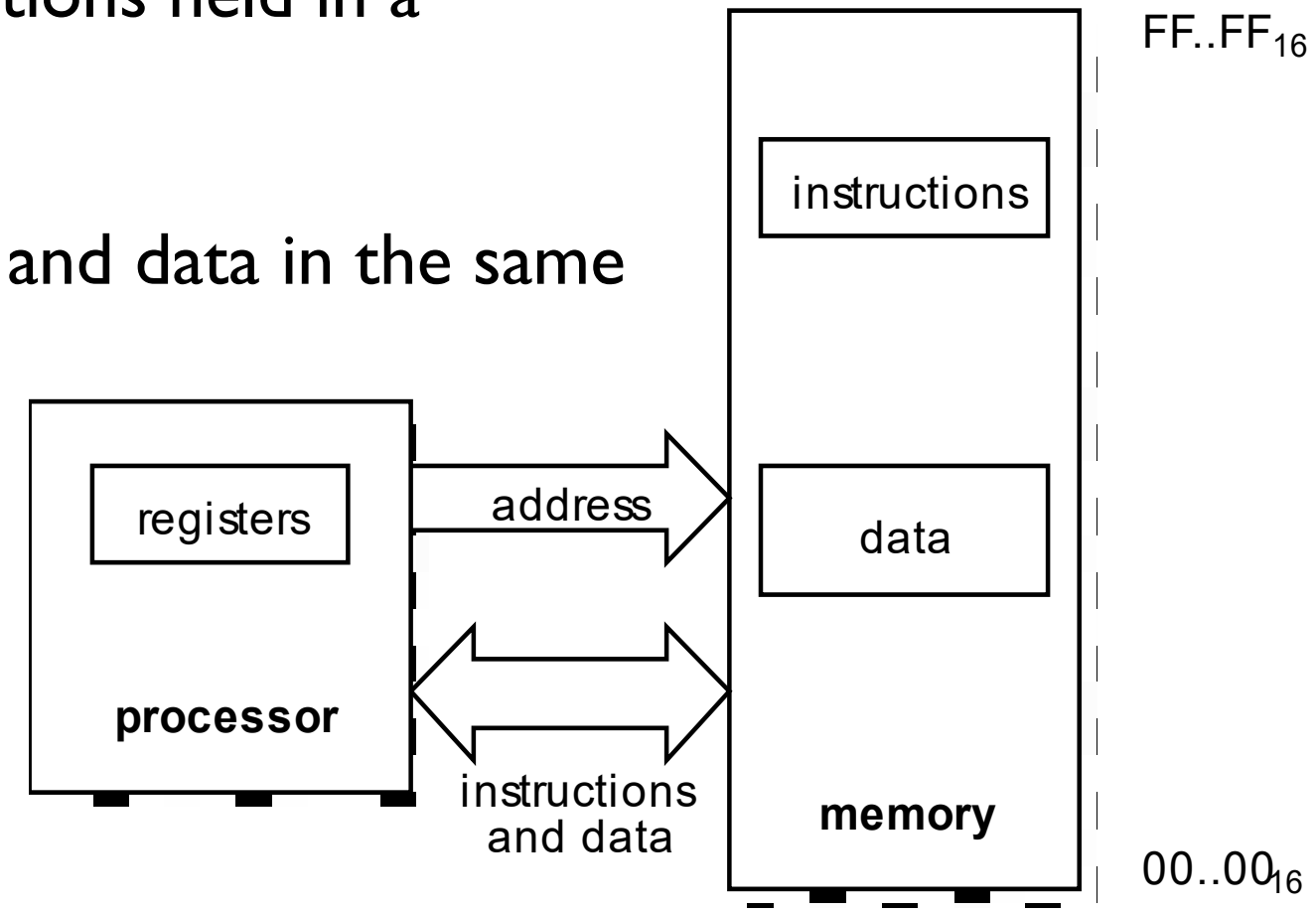


Is there anything missing?

The state in a stored-program digital computer

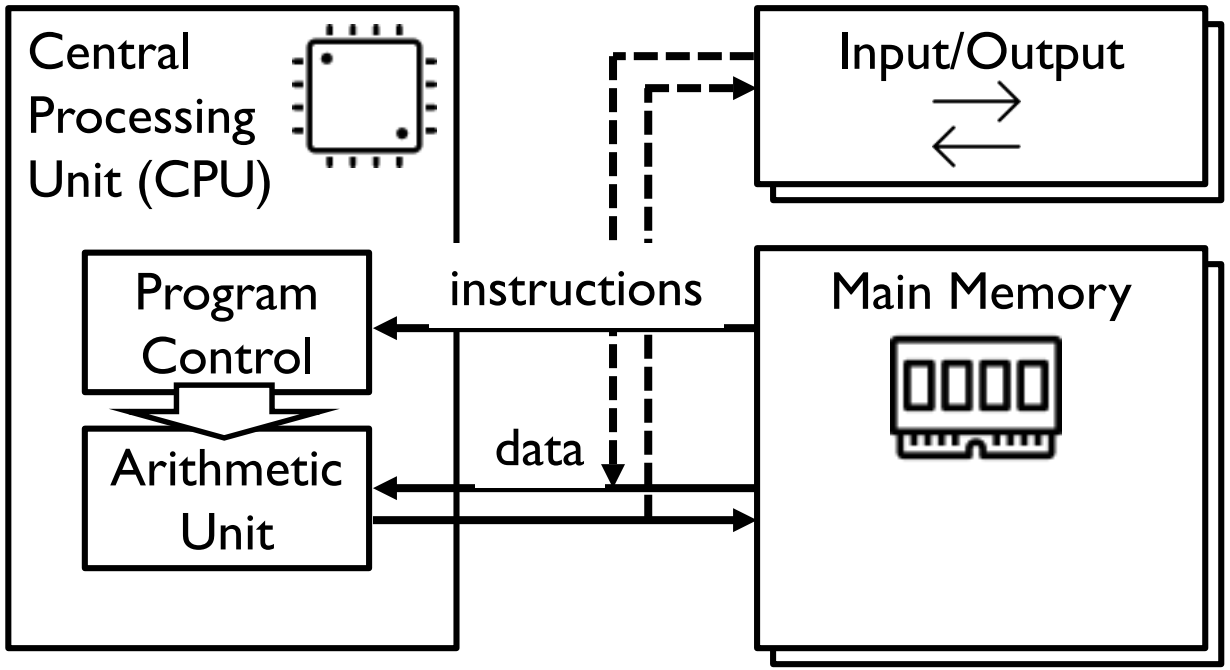
A processor is a finite-state automation that executes instructions held in a memory.

Keep its instructions and data in the same memory system



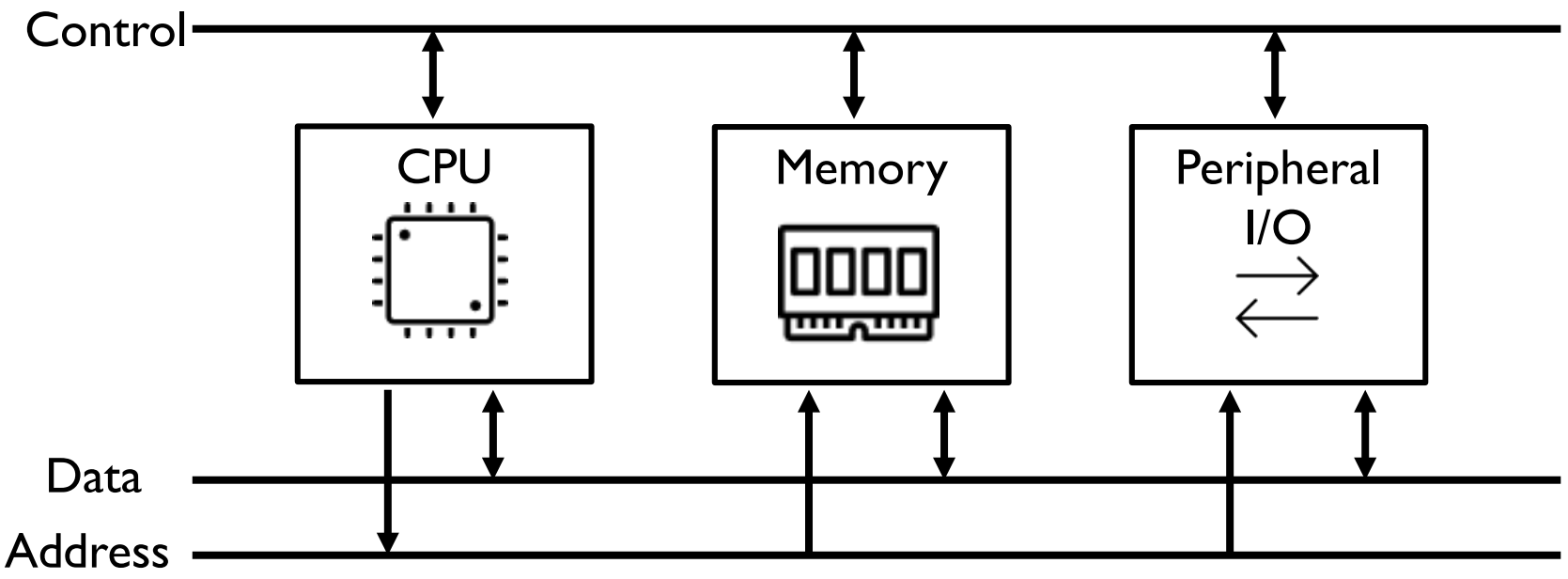
Concept of Computation (Cont')

I/O provides extra data and allow interactions.



Do we have a simple model to describe a computer?

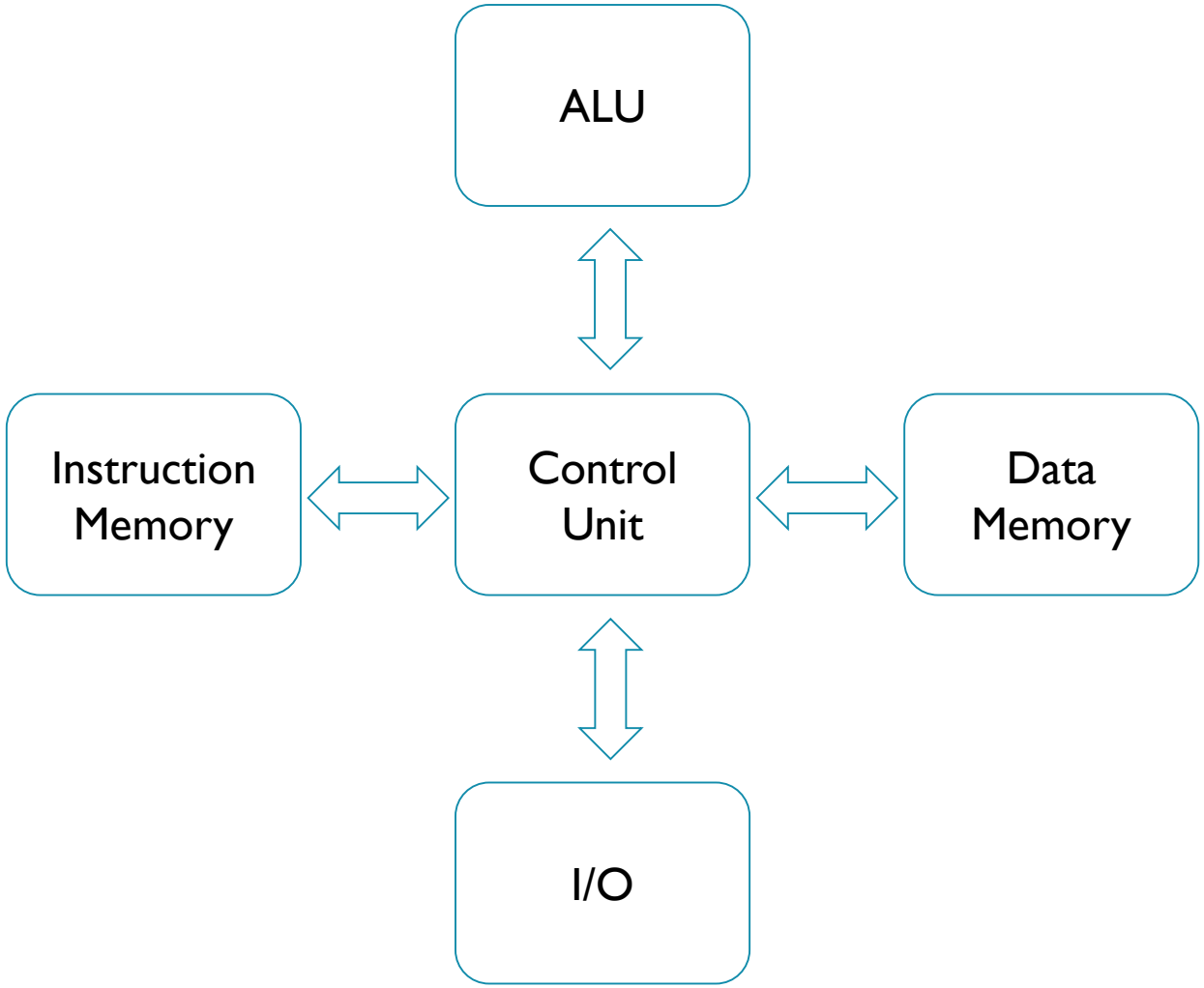
Von Neumann Architecture



Component	Functions
Central Processing Unit (CPU)	controls the system and performs calculations
Main Memory	stores both programs and data
Peripheral Input/Output (I/O)	allows data to be input to system allows results to be output from system

Harvard Architecture

- It is a computer architecture with separate storage and signal pathways for instructions and data.
- Modern processors are mostly von Neumann machines, with the program code stored in the same main memory as the data.
  - Main memory, Cache, TLB, Storage.
  - Speed and Performance.
  - Advanced topics: Read / Write Hazards.

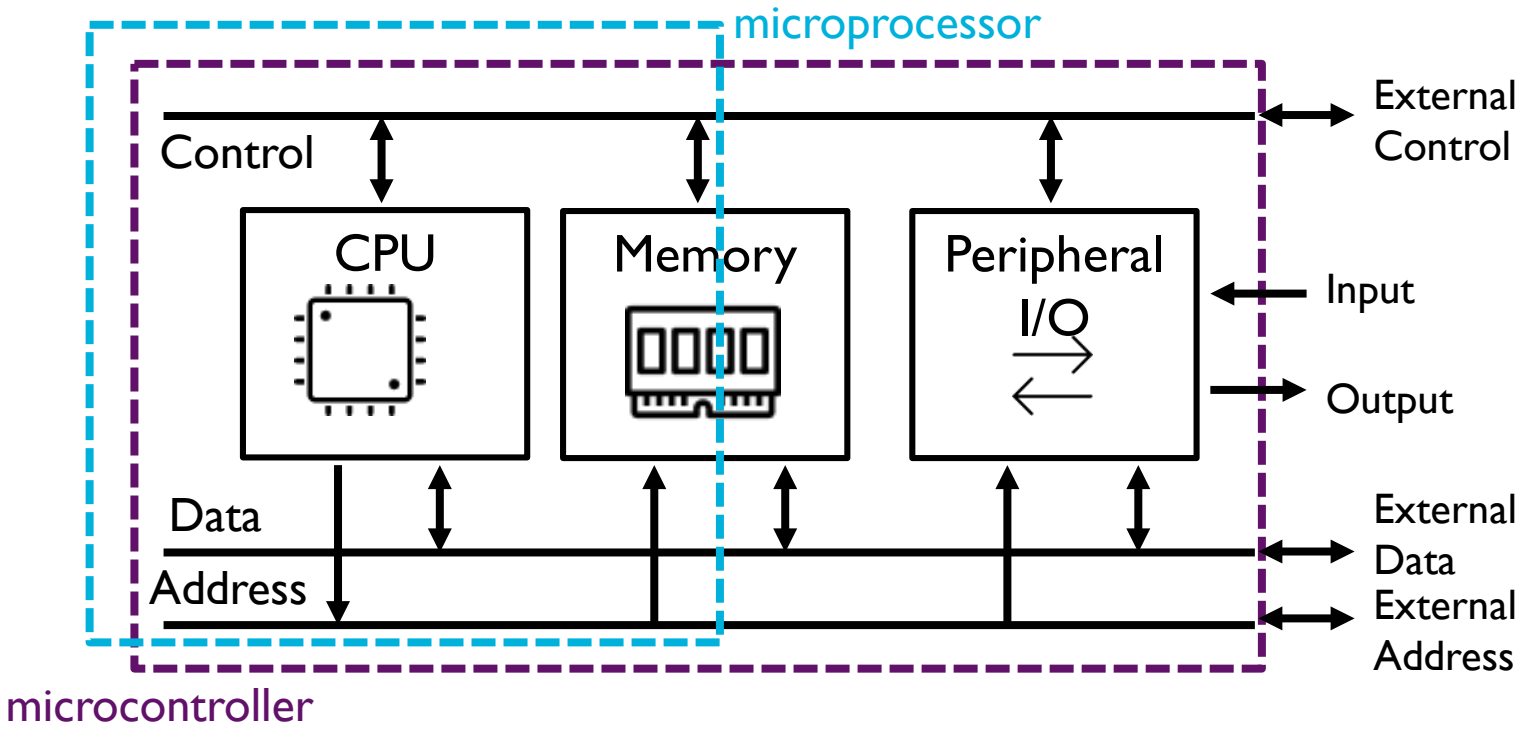


Microcontroller vs. Microprocessor

- Both have a CPU core to execute instructions
- Microcontroller has peripherals for concurrent embedded interfacing and control
  - Analog
  - Non-logic level signals
  - Timing
  - Events
  - Clock generators
  - Communications
  - Reliability and safety

Microprocessor vs. Microcontroller

A microprocessor mainly refers to the CPU with some memories.



A microcontroller unit (MCU) is a microprocessor integrated with both memory and I/O. It is a general-purpose device that is designed to fetch data, perform limited calculations and control the environment.

Stored Program Concept

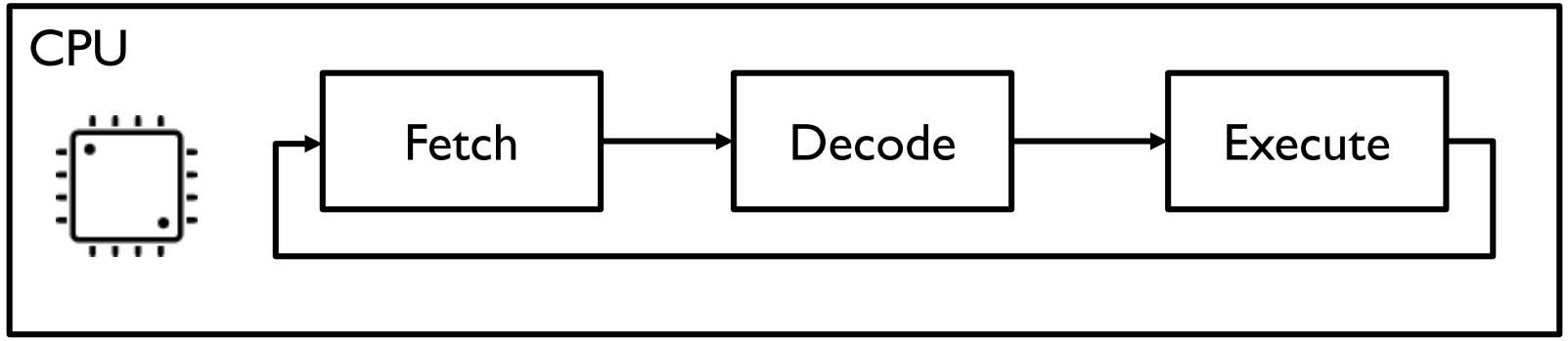
What is inside my computer program?

Stored Program Concept

- The CPU executes instructions stored in memory.
  - So called the "stored program" concept
- Recall there are two kinds of memory to store programs and data that are in use.
  - RAM – Random Access Memory
    Can be used for both programs & data
  - ROM - Read-Only Memory
    Can be used for fixed programs & constant data
- Different computers have different styles in arranging the memories – computer architectures

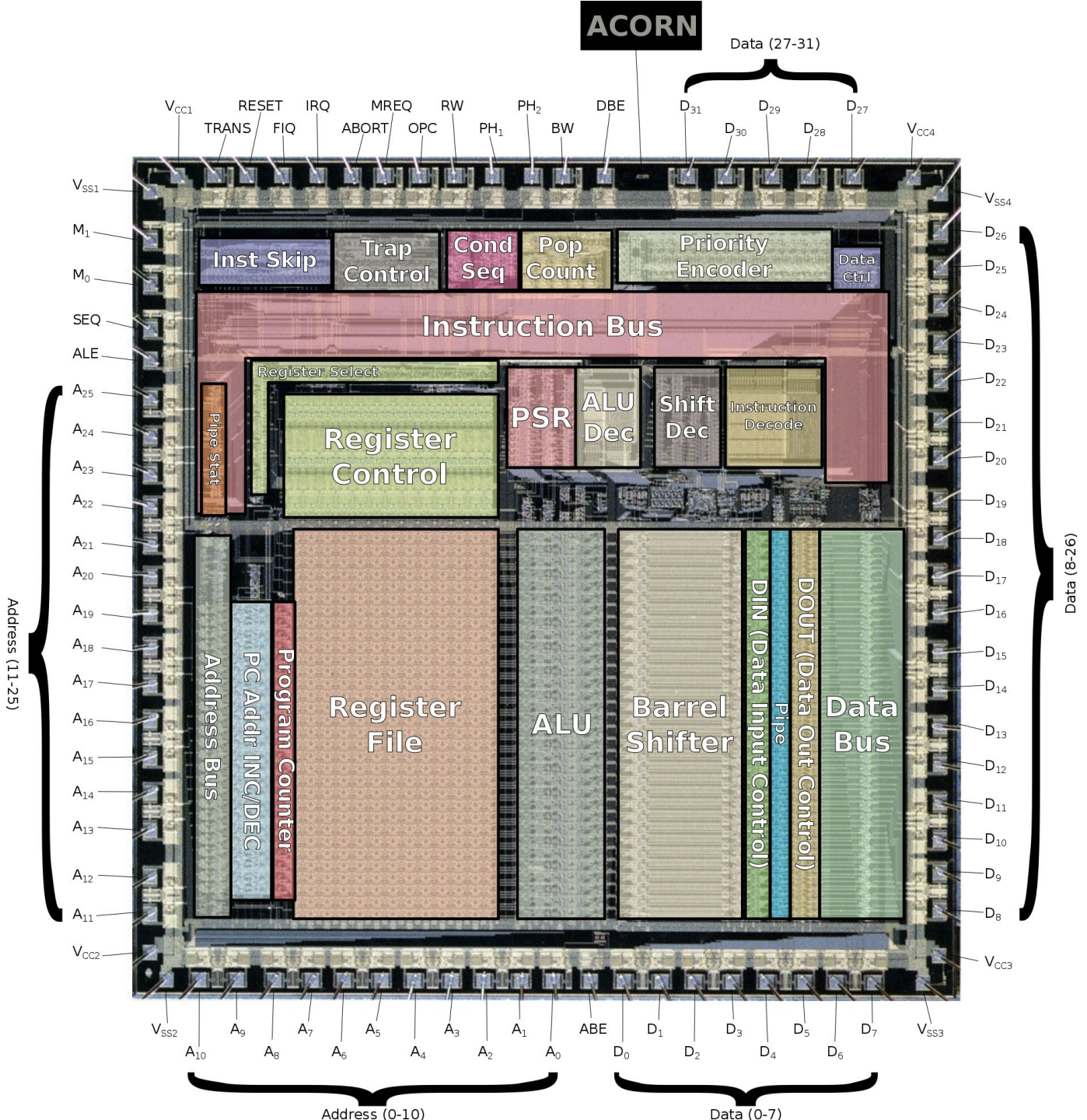
Fetch-Decode-Execute Cycle

- The CPU is a finite state machine (FSM) which runs the programs stored in the memory by the user.
- It repeatedly performs three operations:
  - Fetch – retrieve an instruction from memory
  - Decode – interpret the instruction
  - Execute – control appropriate hardware to carry out the instruction

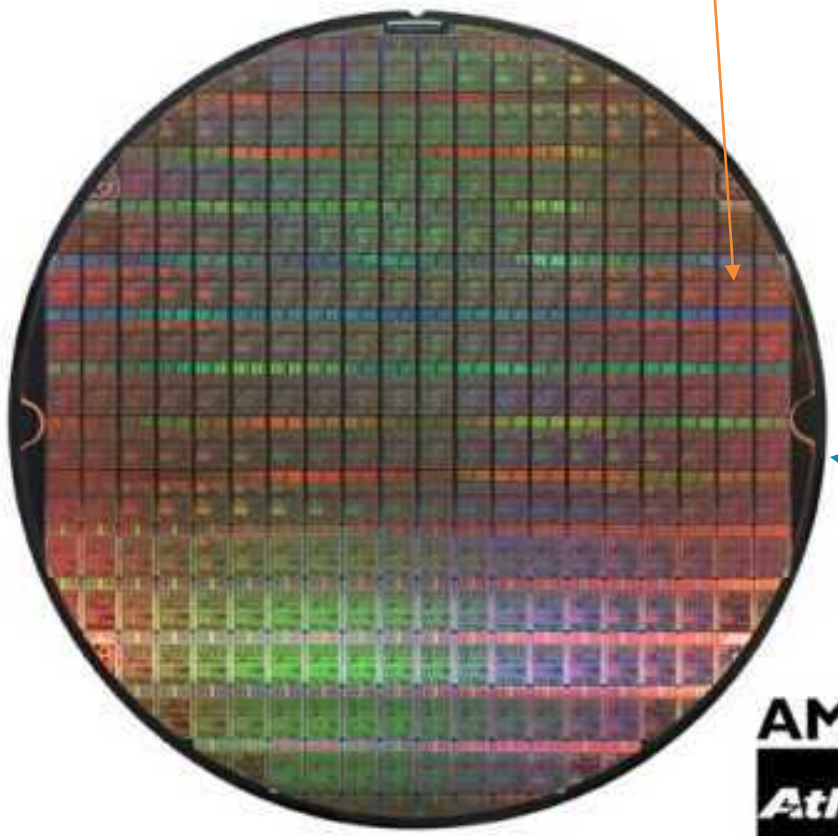


ARM Die Photo

- ARM1 - Microarchitectures – Acorn
- ARM1 was the first ARM microarchitecture designed and realized by Acorn Computers for the BBC Computer Literacy Project.
- ARM1 was introduced in 1985 and was extended to be used as a coprocessor in the Acorn's BBC Micro microcomputers.



Die Cost



Going up to 12" (30cm)

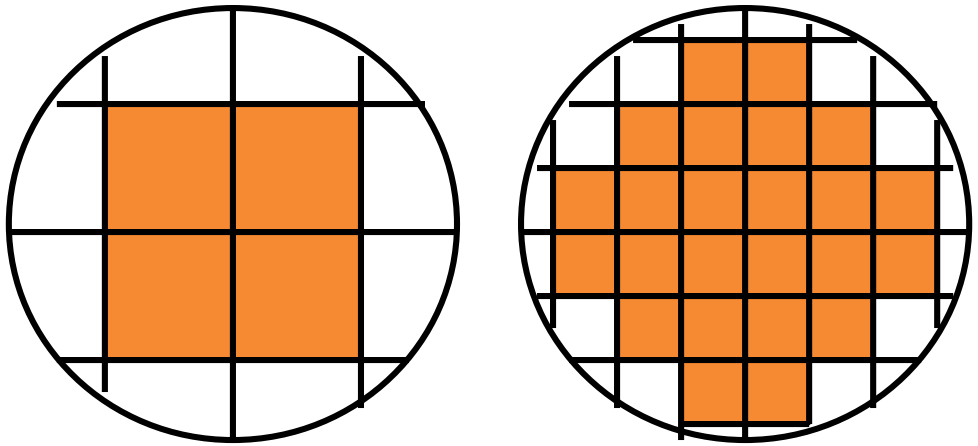
From http://www.amd.com

Yield

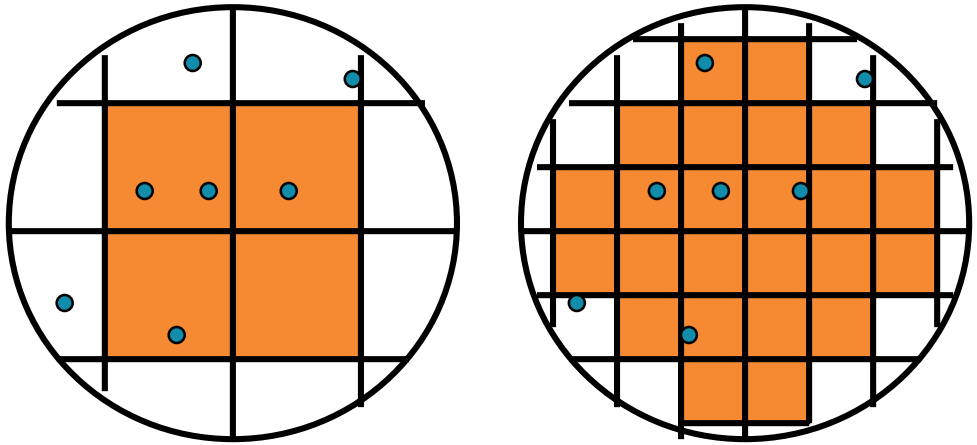
$$Y = \frac{\text{No. of good chips per wafer}}{\text{Total number of chips per wafer}} \times 100\%$$

$$\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per wafer} \times \text{Die yield}}$$

$$\text{Dies per wafer} = \frac{\pi \times (\text{wafer diameter}/2)^2}{\text{die area}} - \frac{\pi \times \text{wafer diameter}}{\sqrt{2 \times \text{die area}}}$$



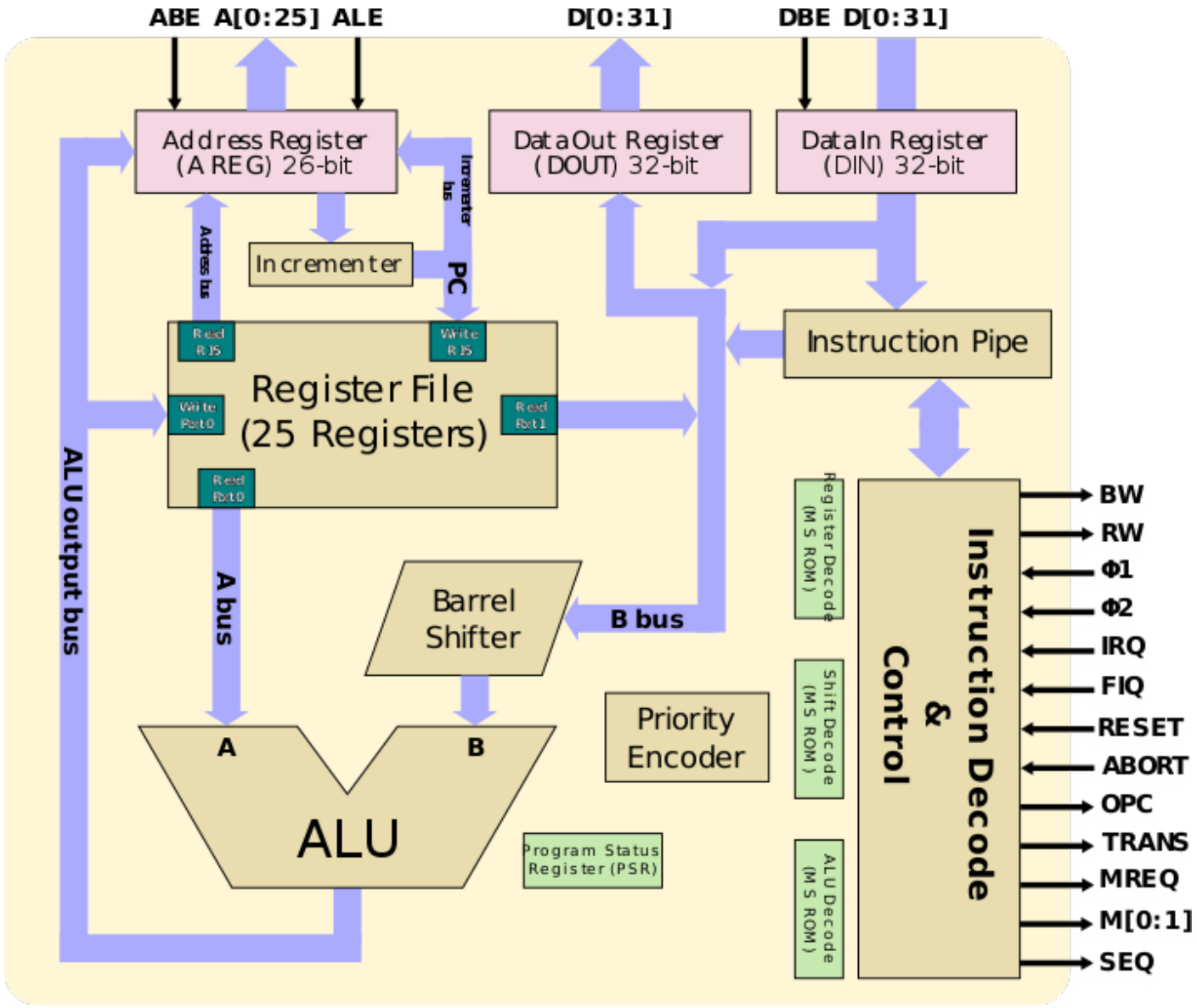
Defects



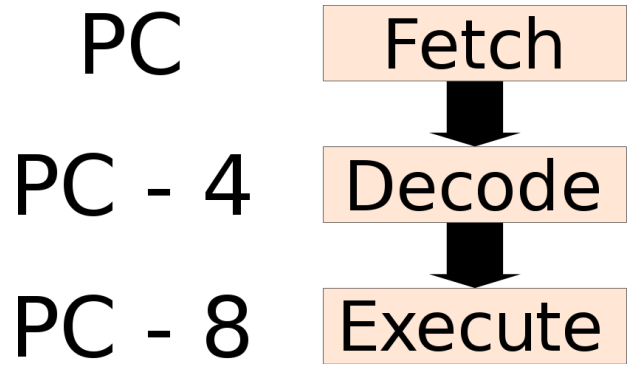
$$\text{die yield} = \left(1 + \frac{\text{defects per unit area} \times \text{die area}}{\alpha}\right)^{-\alpha}$$

α is approximately 3

ARM1 Block Diagram



- Scalar, Pipelined Processor
- 3-pipeline
  - Fetch, Decode, Execute
- Instruction Set Architecture (ISA): ARMv1



What is inside a Program?

- We had written "computer programs" in C or Java.
  - These were compiled and executed.
- If we do not forget a μP/μC is a digital electric circuit (hardware - HW), then the program must be stored as strings of '0' or '1' bits in the memory.
- Assembly programming is considered the closest and the lowest level of programming to the HW.
  - We write instructions that the machine readily understands and executes.

Instructions: Computer’s Language

To command computer's hardware, you must speak its language.

- The words of a computer's language are called instructions, and its vocabulary is called an instruction set.
- It is interesting that there are different dialects of computer languages. It is easy to pick up others if you once learn it.
- The functionalities of a computer are revealed with its instruction set.

Instructions: Operations

Every computer must be able to perform simple arithmetic:

```
ADD r3, r1, r2    ;r3 = r1 + r2
```

instruct a computer to add two variables r1 and r2 and to put their sum back in r3.

The words to the right of the semi-colon (;) are comments for the human reader.

Q: Why is an instruction usually simple?
A: Simplicity favours regularity.

30

Instructions: Operands

The operands of instructions are restricted. They must be from a limited number of special locations in HW called registers.

```
ADD r3, r1, r2    ;r3 = r1 + r2
```

In this example, all r1, r2 and r3 are registers.

In writing instructions, we often need to assign variables to registers. You should note that some registers are dedicated for a special purpose, e.g. r15 is the program counter.

Q: Why is the number of registers small and limited?
A: Smaller is faster. A very large number of registers may increase the clock cycle time.

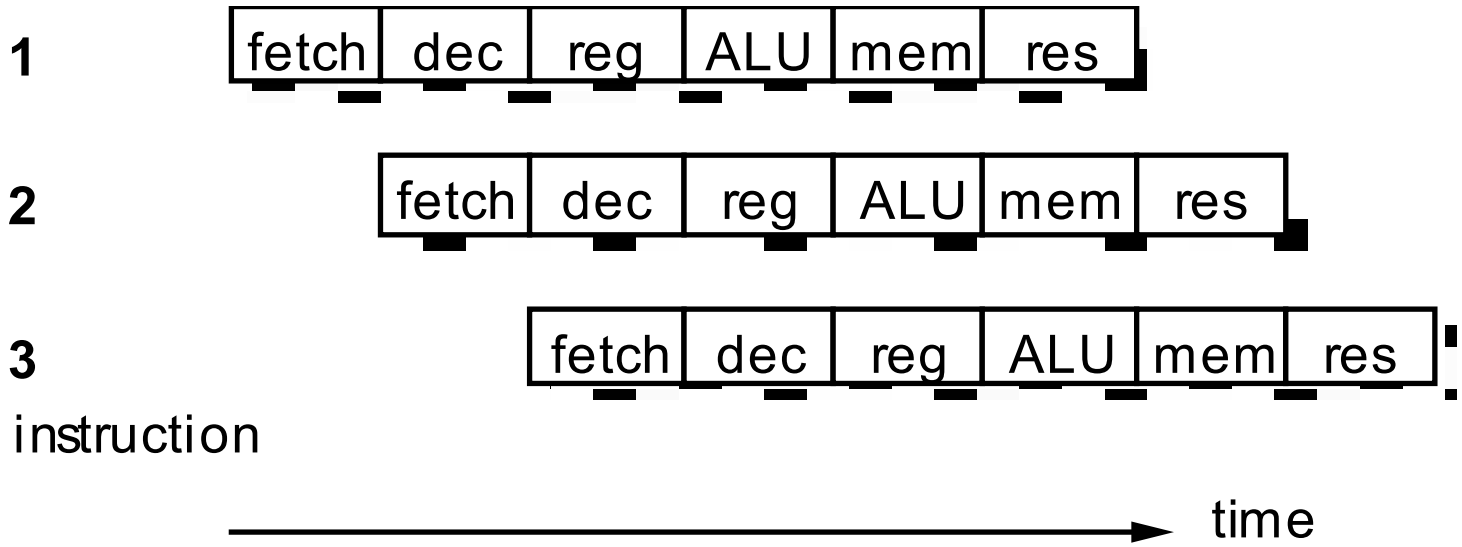
RISC Philosophy

- RISC, or *Reduced Instruction Set Computer*:
  - A type of microprocessor architecture that utilizes a small, highly-optimized set of instructions, rather than a more complex set of instructions.
- History:
  - The first RISC projects came from IBM, Stanford, and UC-Berkeley in the late 70s and early 80s. The IBM 801, Stanford MIPS, and Berkeley RISC 1 and 2 were all designed with a similar philosophy which has become known as RISC.

RISC Philosophy

- MIPS Technologies, Inc.:
  - Founded in 1984 upon the Stanford research from which the first MIPS chip resulted. Started with MIPS R2000 processor (1986).
  - MIPS: Microprocessor without Interlocked Pipelined Stages.
  - In a pipelined processor, there must exist a mechanism to enforce dependencies between instructions. Data dependency occurs when an instruction depends on the results of a previous instruction.
  - In early processors, pipelined interlocks were implemented by software.
  - Pipelining is a standard feature in RISC processors.

Pipelined instruction execution



RISC vs. CISC

- A brief comparison:

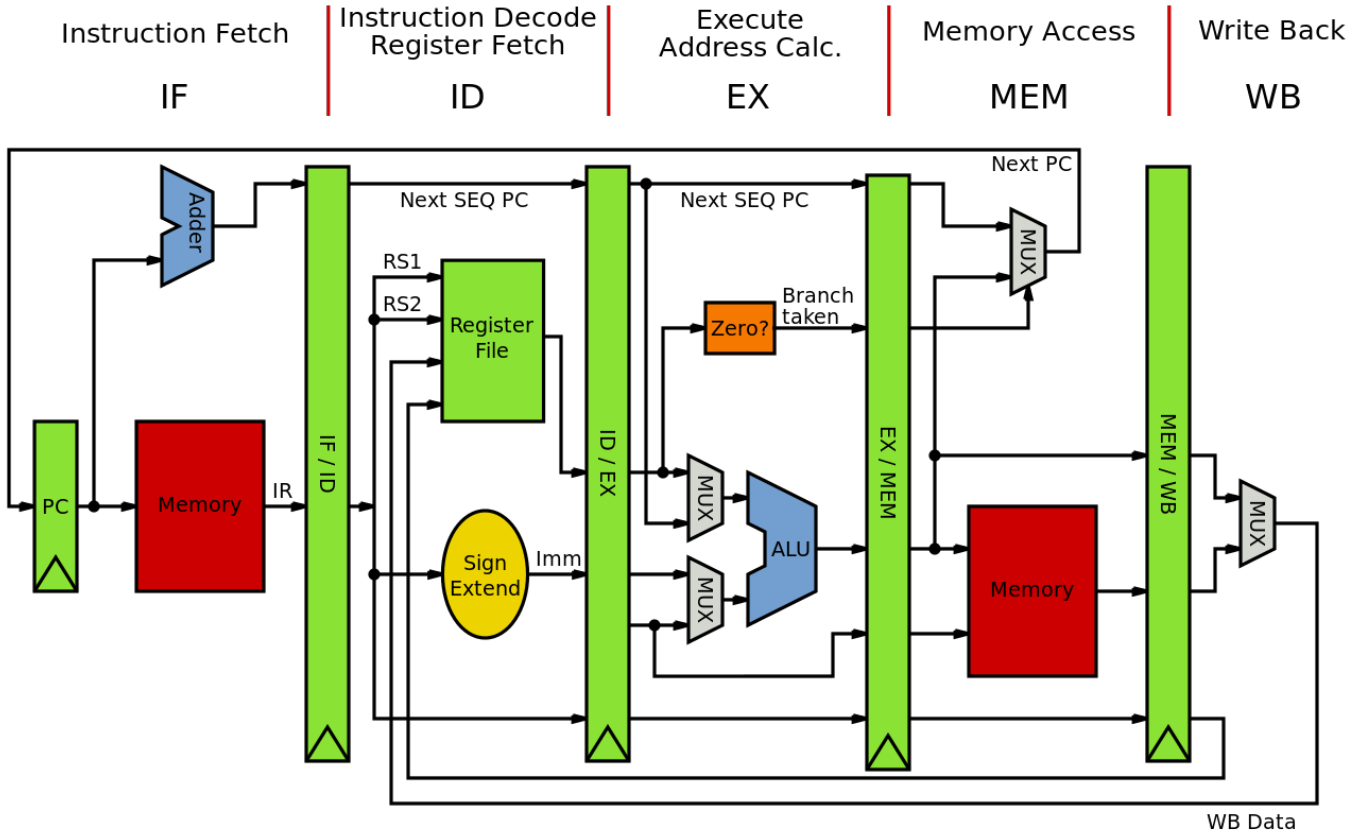
Reduced Instruction Set Computing (RISC)	Complex Instruction Set Computers (CISC)
Emphasis on software.	Emphasis on hardware.
Single-clock, simple instruction set.	Includes multi-clock complex instructions
Register to register: "LOAD" and "STORE" are independent instructions.	Memory-to-memory: "LOAD" and "STORE" incorporated in Instructions.
Low cycles per second, large code sizes.	Small code sizes, high cycles per second.
Spends more transistors on memory registers.	Transistors used for storing complex instructions.

RISC vs. CISC

- Which is better?
  - A topic of intense debate in the 1980s and 1990s.
  - But now understood that decoding and processing is easy with a RISC ISA;
  - it also has been shown that hardware can translate complex CISC-style instructions into RISC-style instructions and process them.

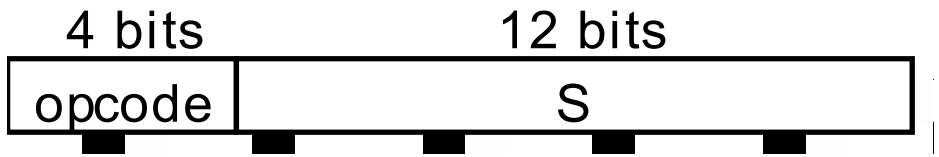
MIPS ISA

- MIPS instruction set architecture (ISA):
  - Contains a set of simple arithmetic, logical, memory-access, branch, and jump instructions.
  - Emphasizes simplicity and excludes instructions that might take longer than the most common instructions.
  - There are 32 general-purpose registers in the MIPS architecture.
  - Each register is 32 bits wide. Registers are often referred to as $0, $1, $2, …, and $31.



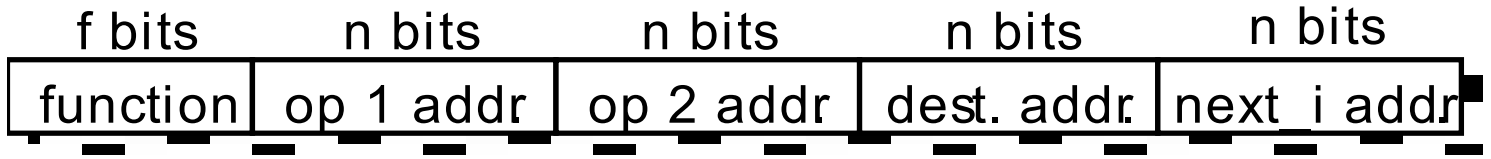
37

Instruction Set Architecture (ISA)

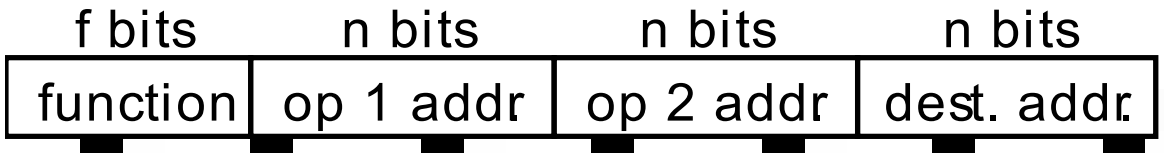


Instruction	Opcode	Effect
LDA S	0000	ACC := mem_{16}[S]
STO S	0001	mem_{16}[S] := ACC
ADD S	0010	ACC := ACC + mem_{16}[S]
SUB S	0011	ACC := ACC - mem_{16}[S]
JMP S	0100	PC := S
JGE S	0101	if ACC >= 0 PC := S
JNE S	0110	if ACC !=0 PC := S
STP	0111	stop

Different types of Instruction Format

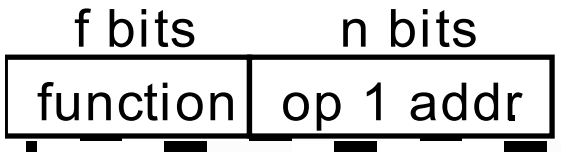


4-address instruction format

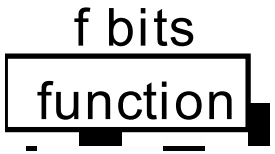


3-address instruction format

2-address instruction format



1-address instruction format



0-address instruction format

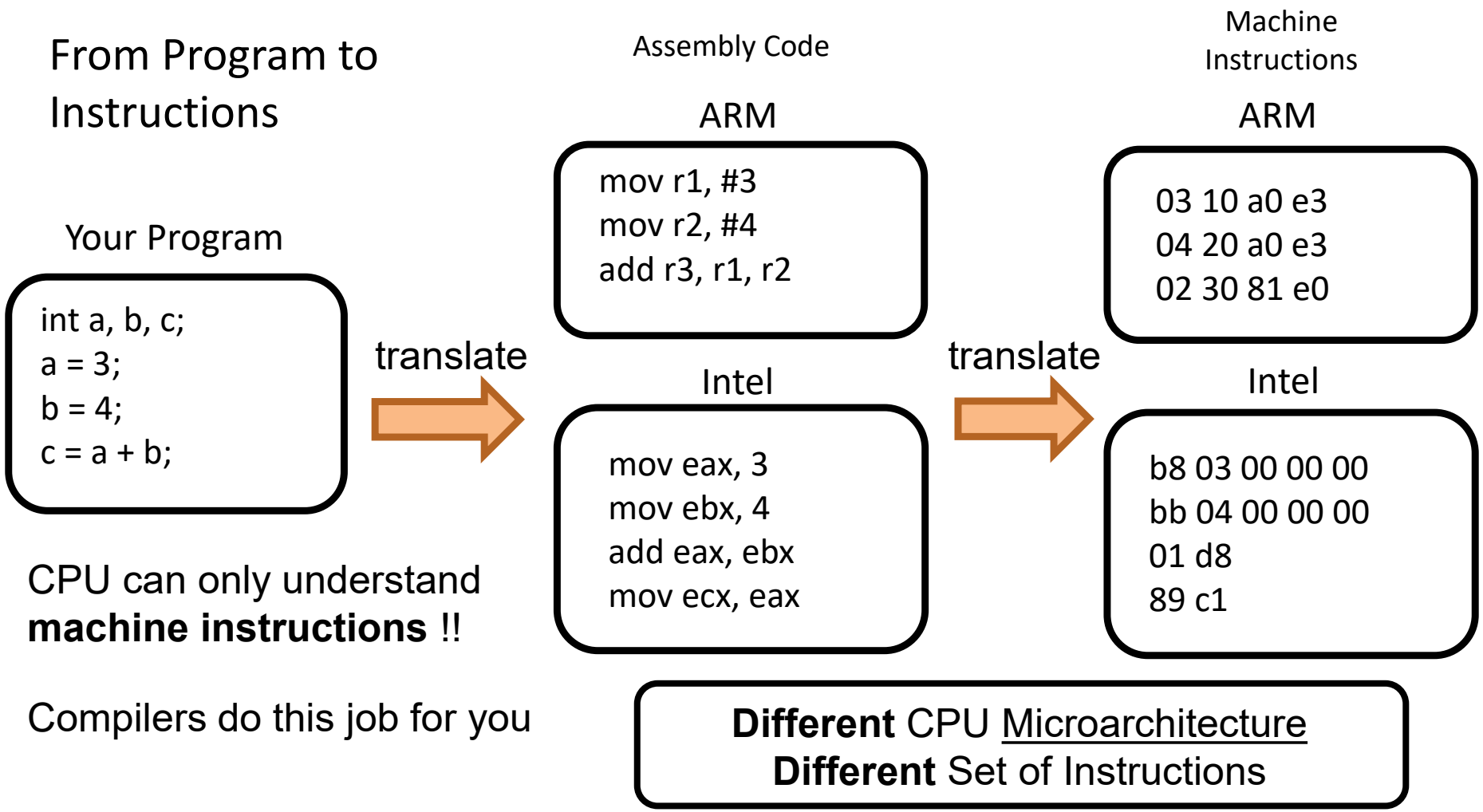
MIPS ISA

Instruction	Example	Meaning
Add (with overflow)	add $s1, $s2, $s3	$s1=$s2+$s3
Add immediate (with overflow)	addi $s1, $s2, k	$s1 = $s2 + k k is a 16-bit constant.
Add unsigned (no overflow)	addu $s1, $s2, $s3	$s1 = $s2 + $s3
Add immediate unsigned (no overflow)	addiu $s1, $s2, k	$s1 = $s2 + k k is unsigned 16-bit constant
Subtract (with overflow)	sub $s1, $s2, $s3	$s1 = $s2 - $s3
Subtract unsigned (no overflow)	subu $s1, $s2, $s3	$s1 = $s2 - $s3

Typical dynamic instruction usage

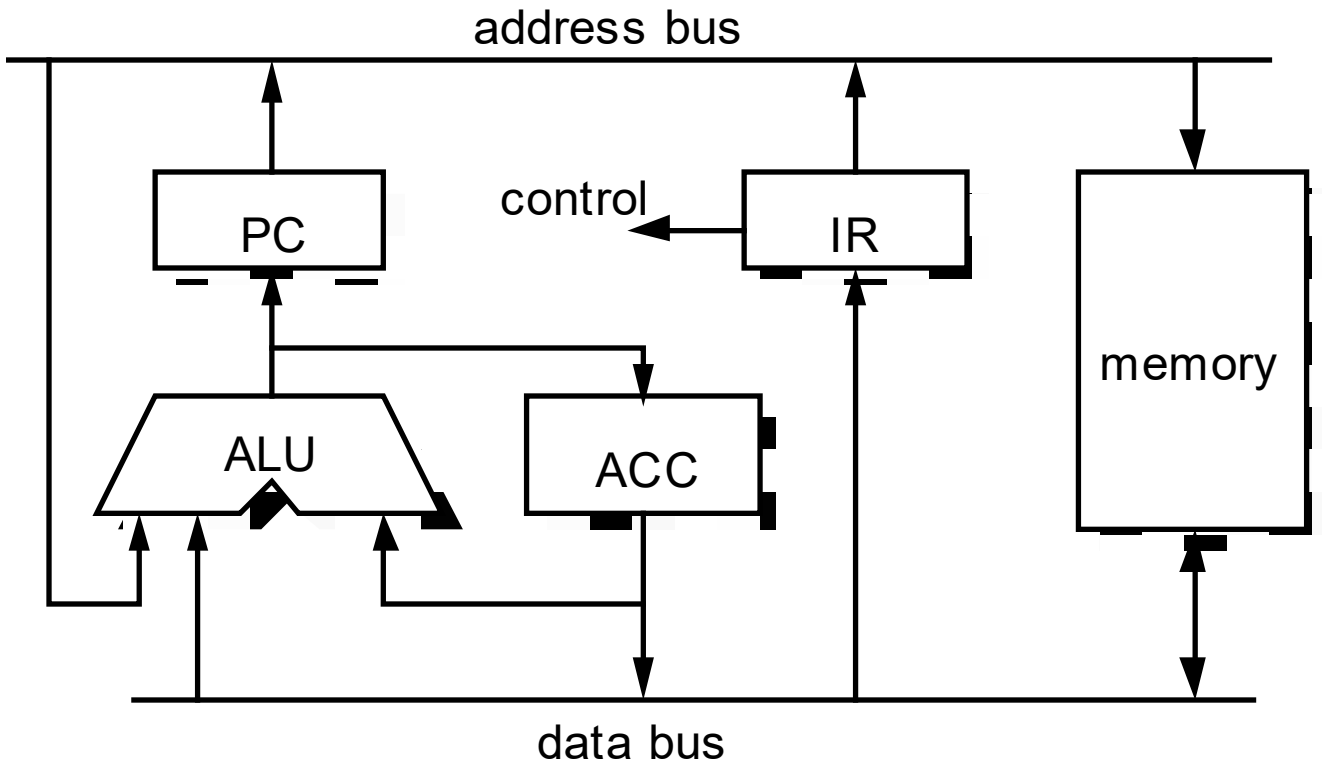
Instruction type	Dynamic usage
Data movement	43%
Control flow	23%
Arithmetic operations	15%
Comparisons	13%
Logical operations	5%
Other	1%

From Program to Instructions

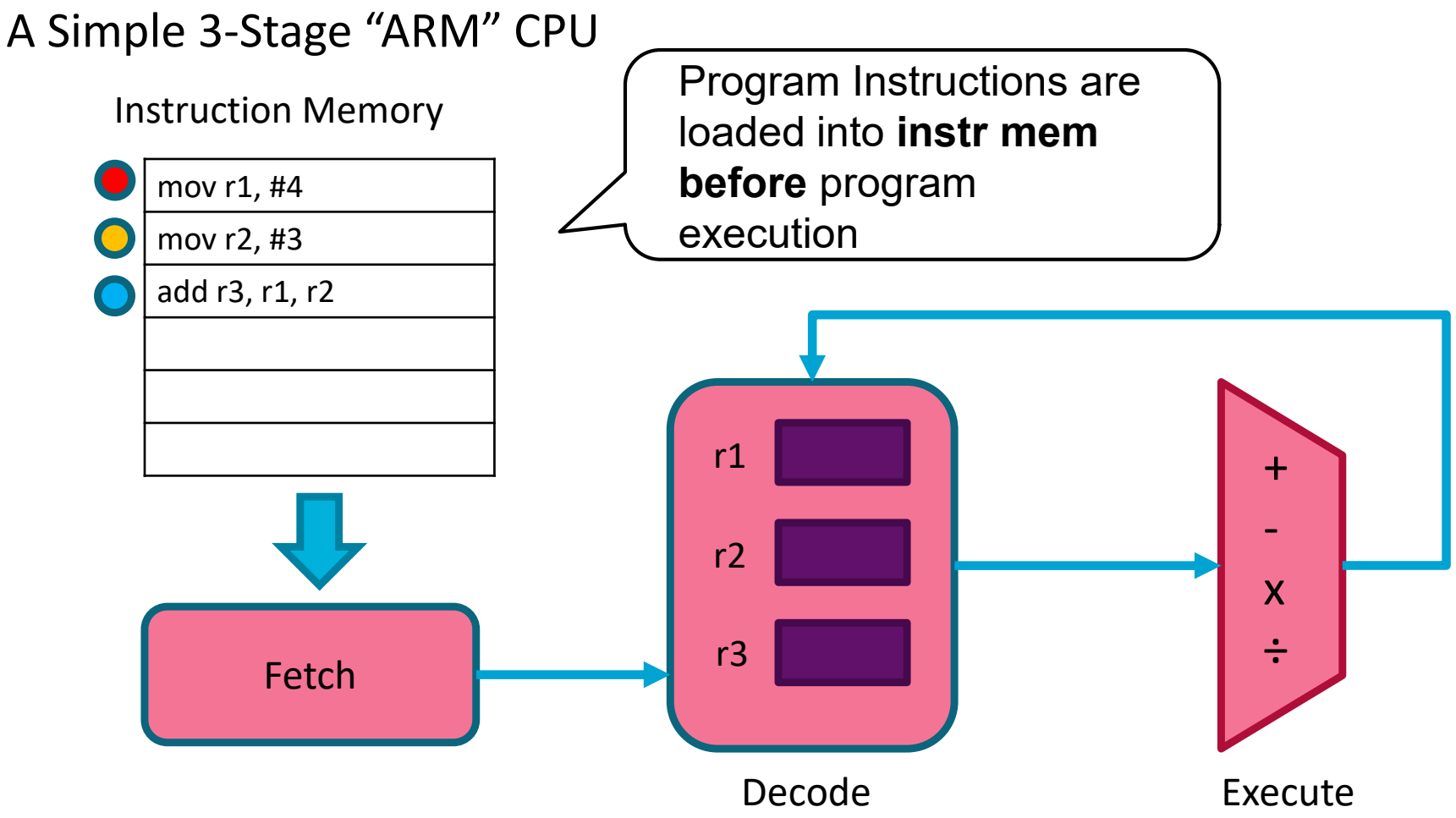


Example: A Typical Datapath

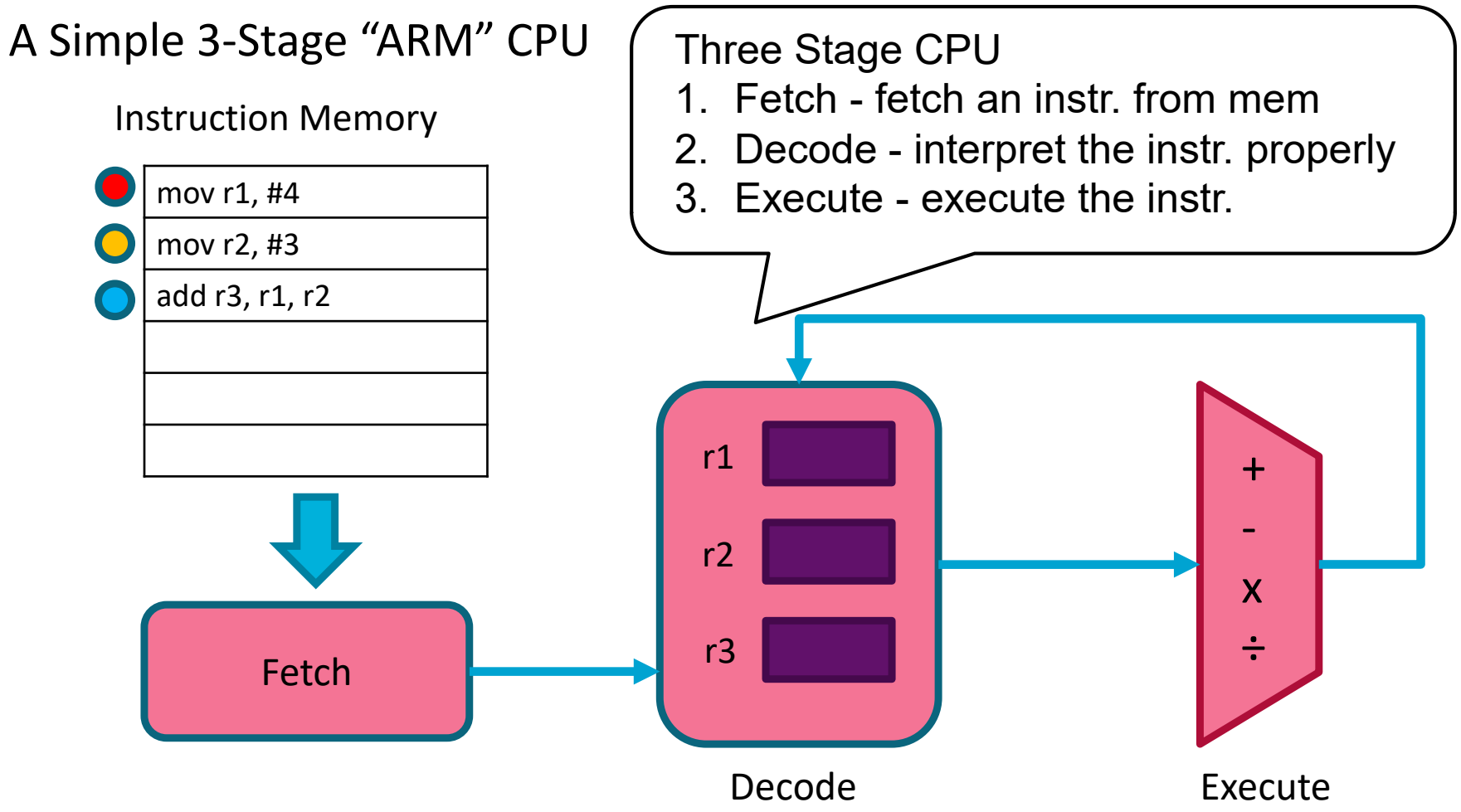
An instruction triggers the movement of data of each component in a datapath. The question is how optimized the datapath is?



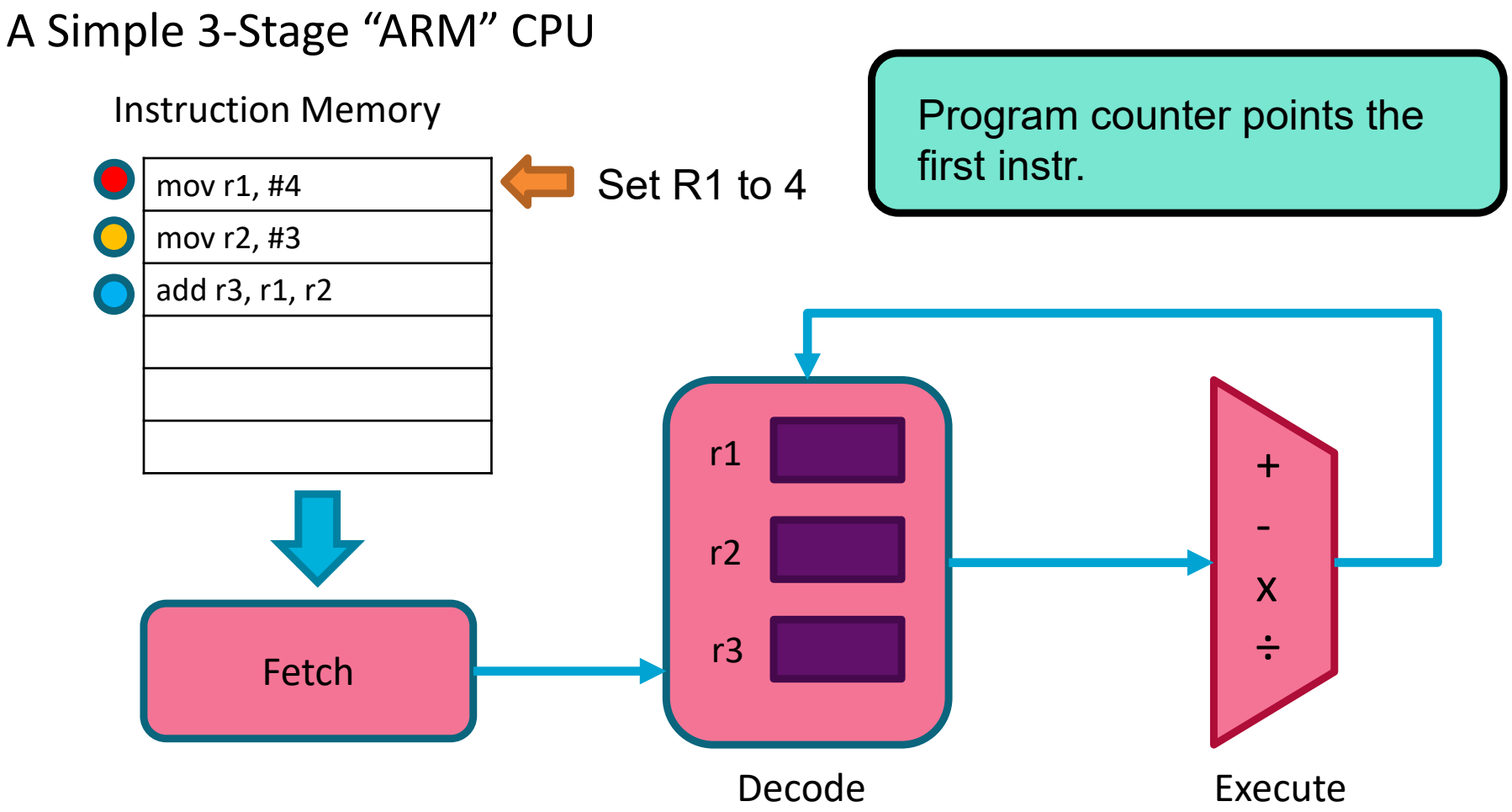
A 3-Stage CPU



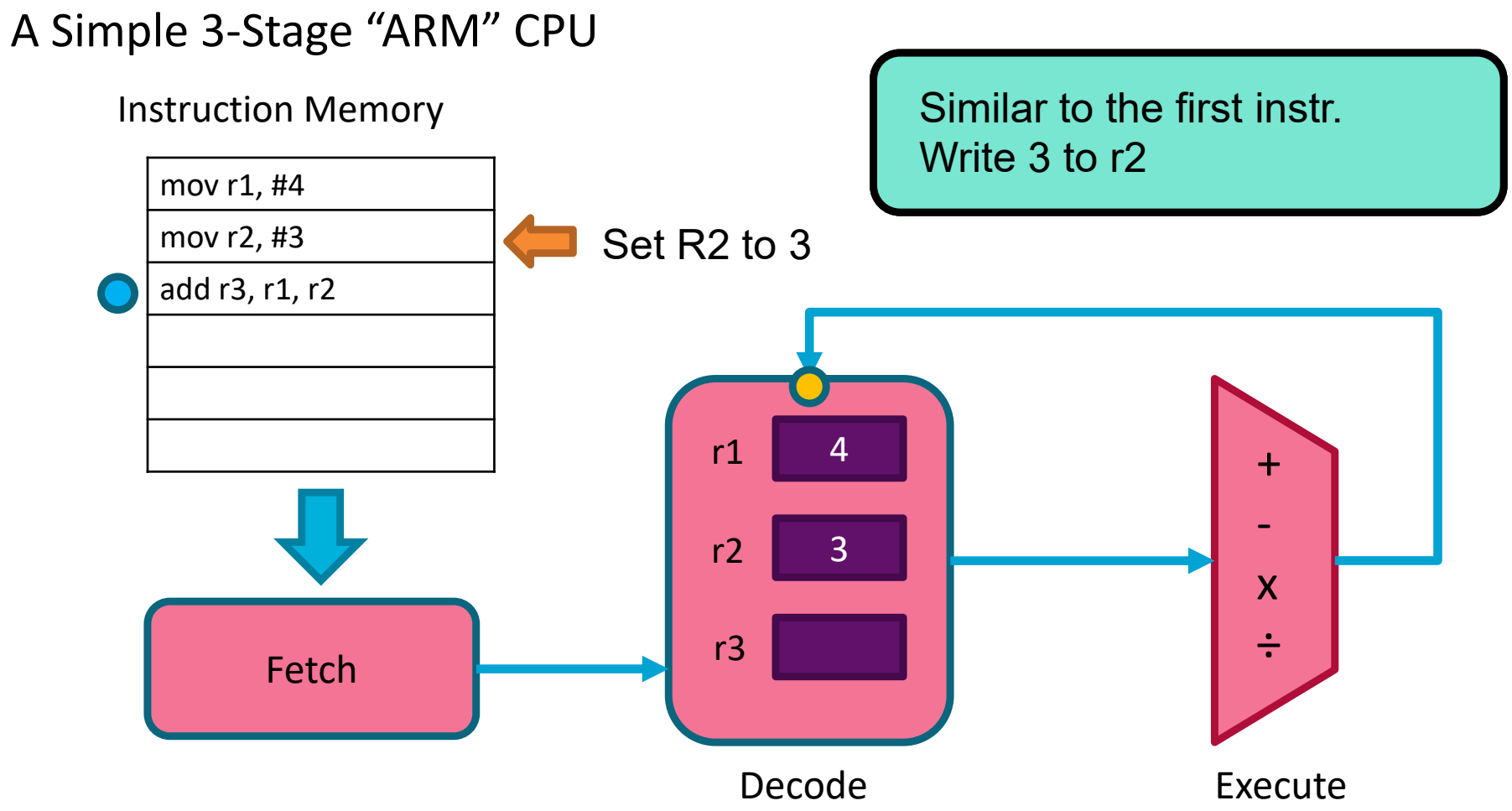
A 3-Stage CPU



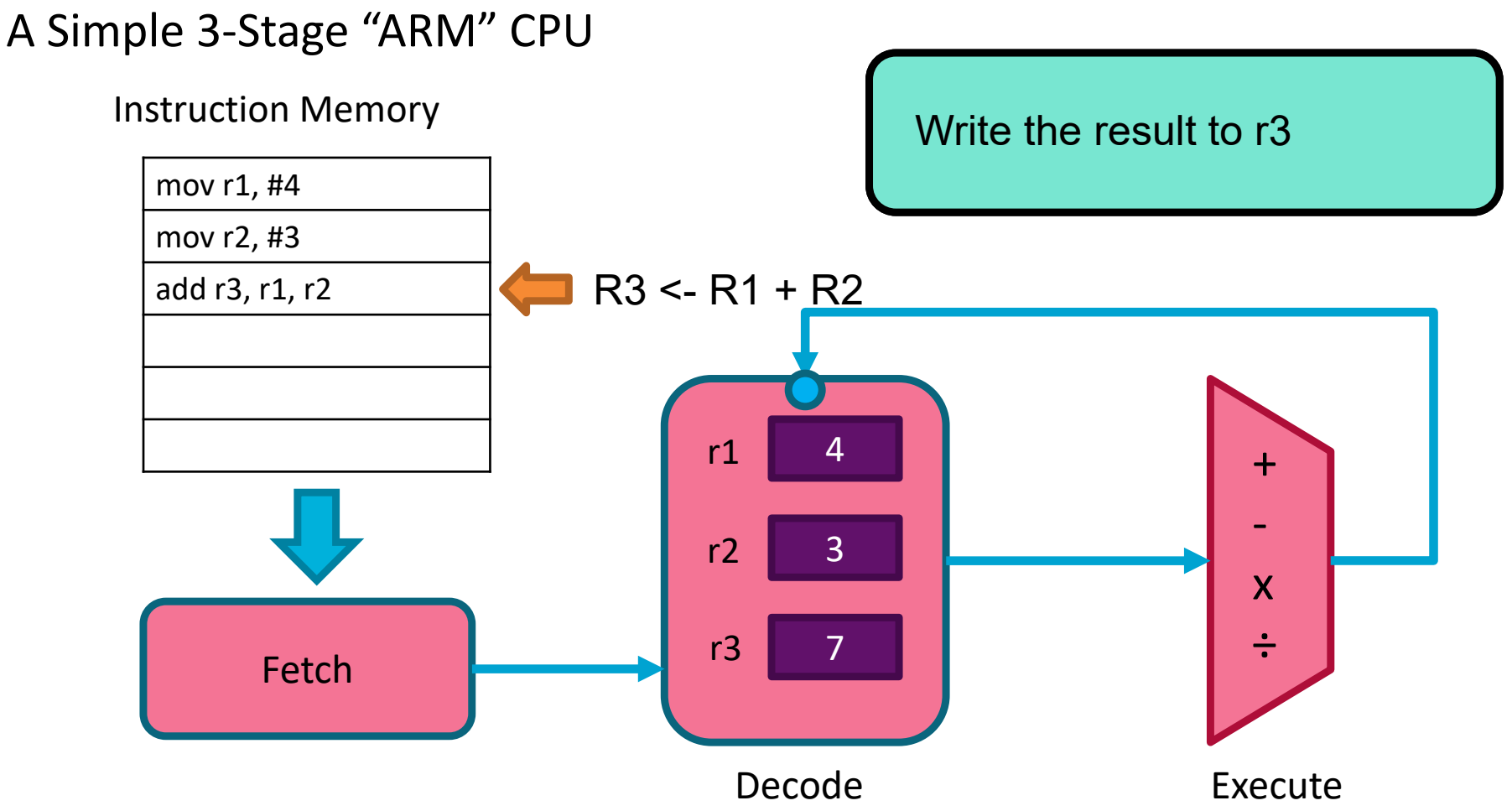
A 3-Stage CPU



A 3-Stage CPU

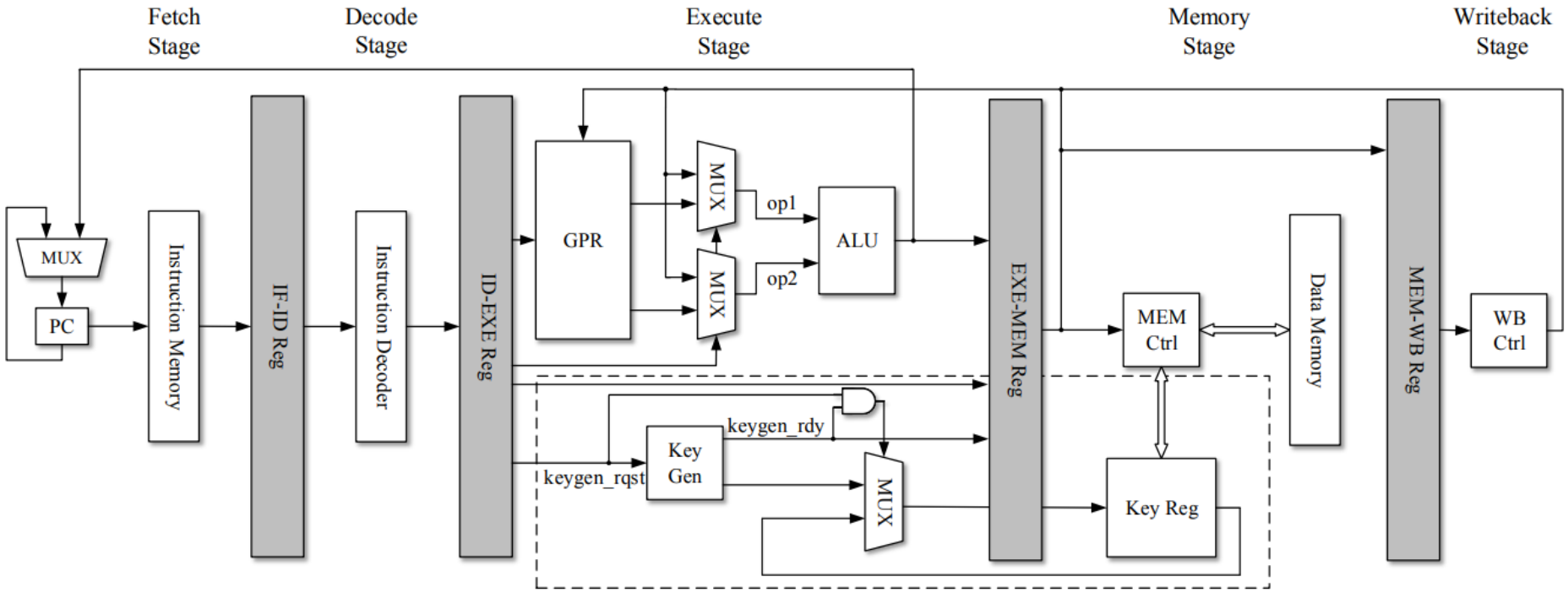


A 3-Stage CPU



Processor Micro-Architecture

- Machine Code running on this architecture
- Compile a high-level language to machine code



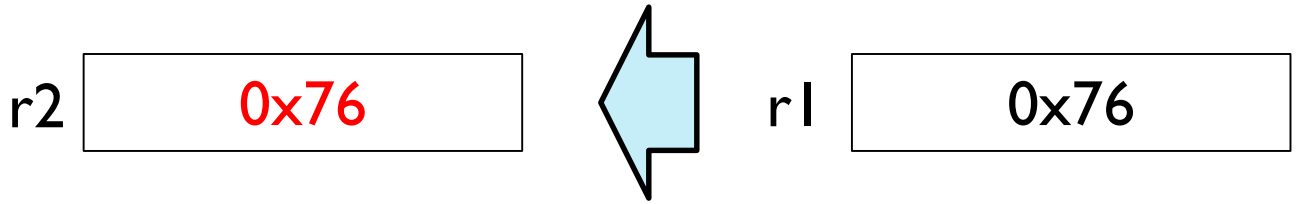
CityU - Secure Microprocessor

Data Transfer between Registers

If arithmetic operations occur only on registers, then it should be allowed to transfer data between registers.

MOV r2, r1 ;r2 = r1

instruct a computer to copy/move the value stored in register r1 to the register r2.



We'll see how to transfer data between register and memory (load & store).

Compiling Two C Statements

Translate the following C-style statement into instructions:

w = x – (y + z);

We can translate into two instructions

```
ADD r3, r1, r2        ; y: r1, z: r2, w: r3
SUB r3, r0, r3        ; x: r0
```

This is what a C compiler will do.
It analyses our statements and work out the sequence of machine instructions.

We will study the principles of compilation into assembly in details later in the module.

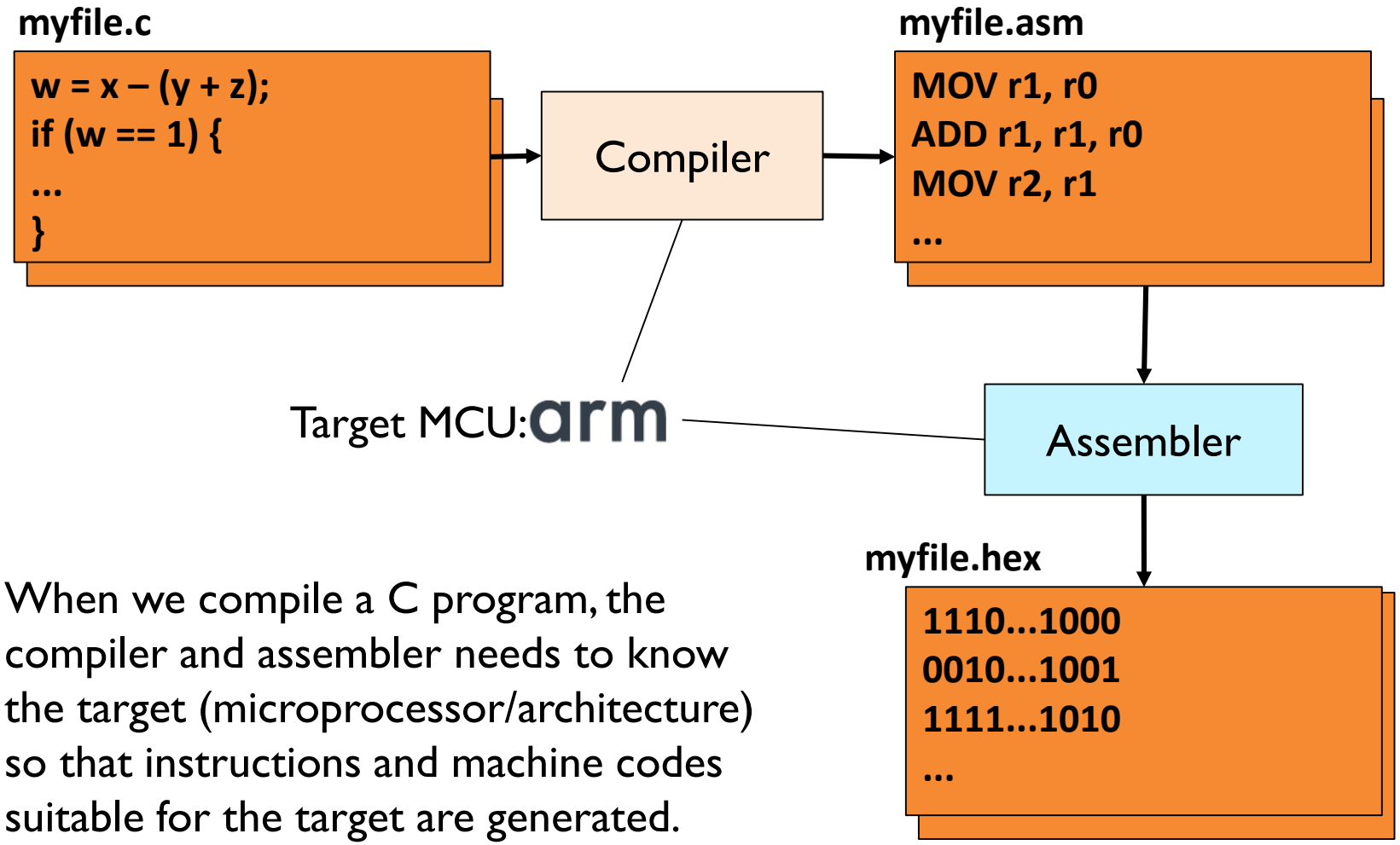
Assembly Language

To make the machine executes our instruction, an assembler will translate our symbolic notation into machine code:

MOV r2, r1 → 1110 0001 1010 0000 0010 0000 0000 0001

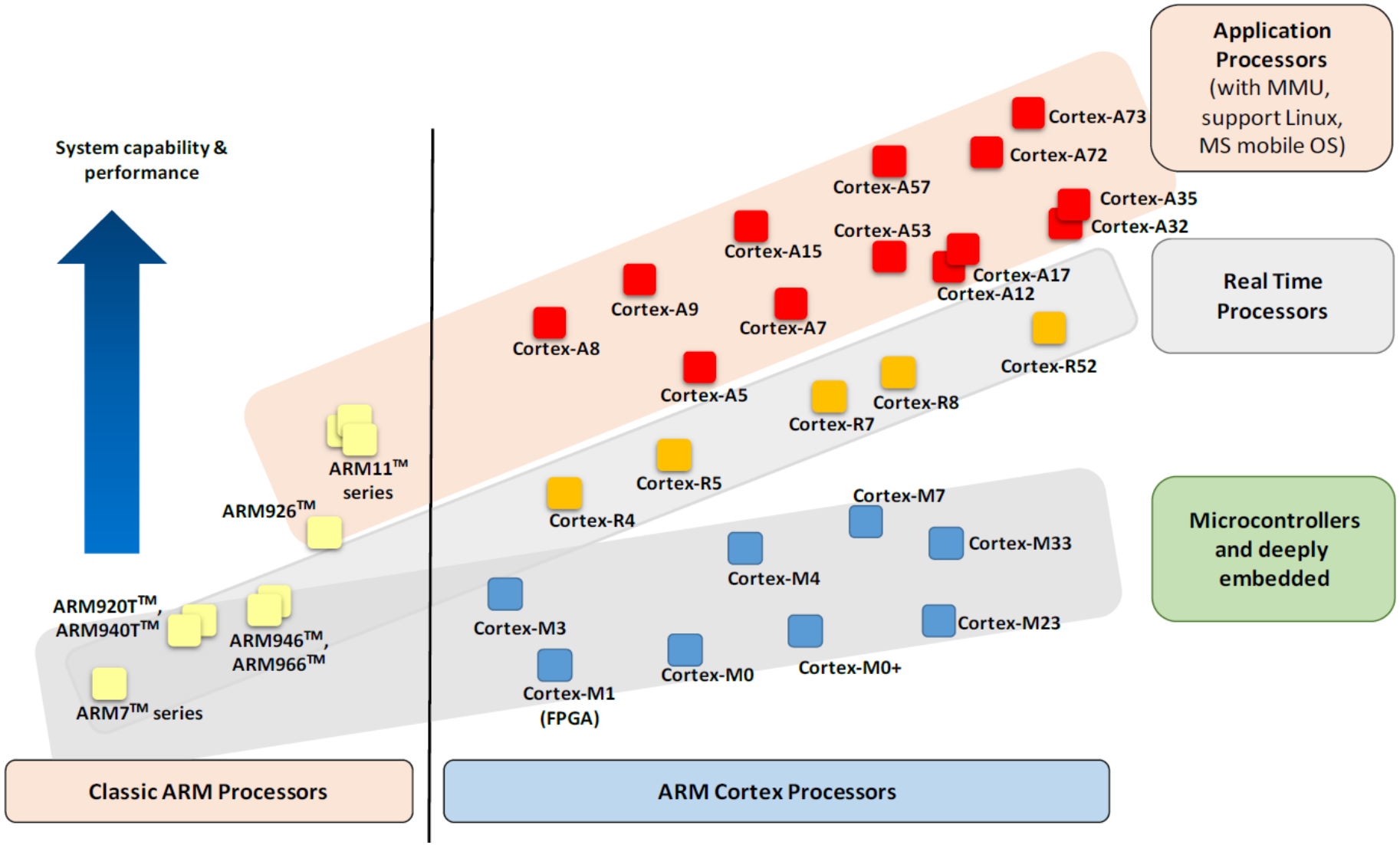
- Writing in binary is far too difficult for us.
- We prefer to use the symbolic language that called the assembly language (ADD, MOV, etc.).
- Assembly language requires the programmer to write one line for every instruction that the machine will follow, forcing the programmer to think like the machine.

Compilation and Assembling



When we compile a C program, the compiler and assembler needs to know the target (microprocessor/architecture) so that instructions and machine codes suitable for the target are generated.

ARM Processor Family

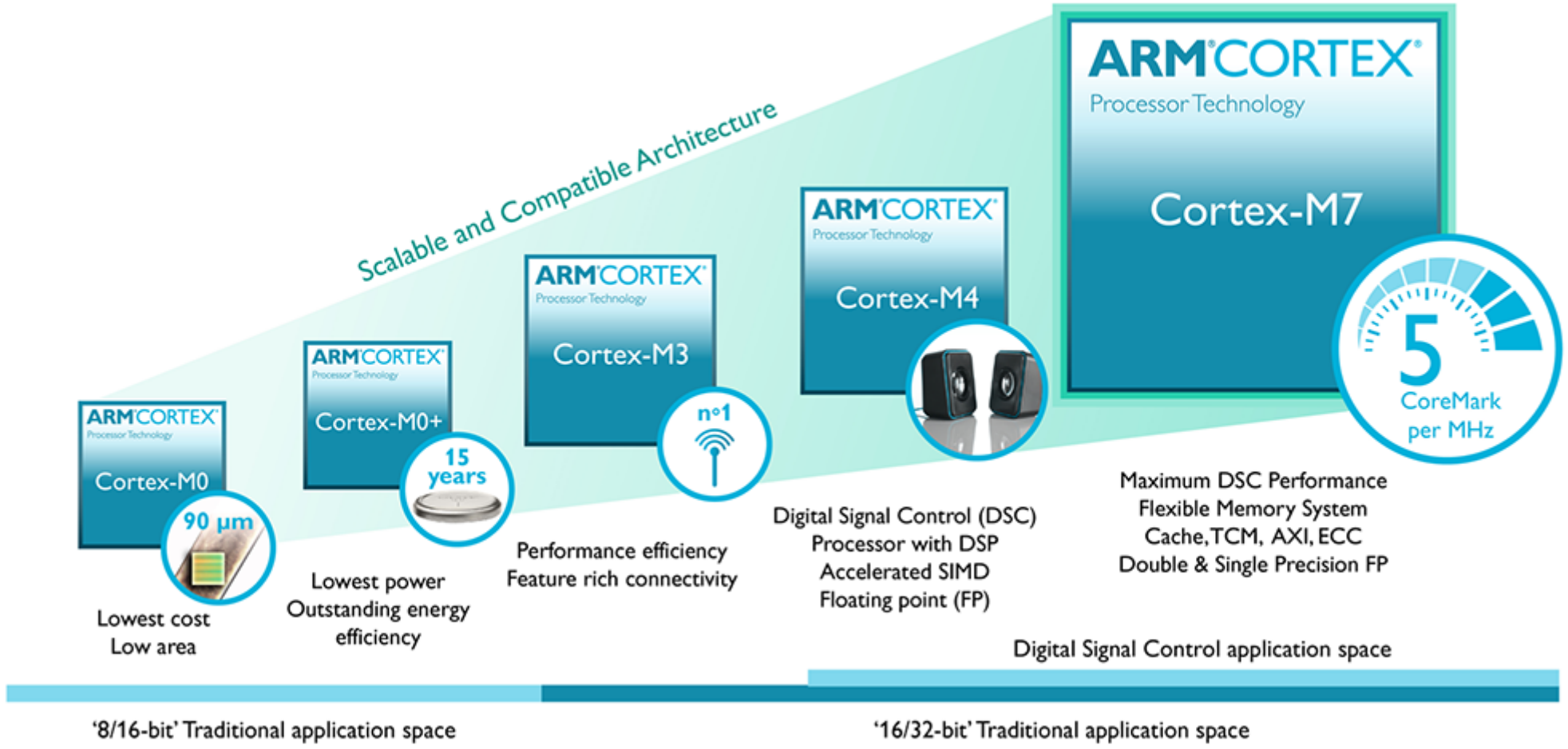


ARM Processor Family

- **ARM Processors** are divided between the classic ARM processors and the newer Cortex processors
- **Application Processors** – High-end processors for mobile computing, smart phone, servers
  - These processors run at higher clock frequency (over 1GHz), and support Memory Management Unit (MMU)
- **Real-time Processors** – very high-performance processors for real-time applications such as hard disk controller, automotive power train and base band control in wireless communications
  - Most of these processors do not have MMU, and usually have Memory Protection Unit (MPU), cache, and other memory features designed for industrial applications
  - They can run at a fairly high clock frequency (e.g. 200MHz to >1GHz)
  - very low response latency
- **Microcontroller Processors** – much lower silicon area, and much high-energy efficiency
  - shorter pipeline, and usually lower maximum frequency
  - running at over 200MHz
  - designed to be very easy to use (Cortex-M)
  - very popular in the microcontroller and deeply embedded systems market

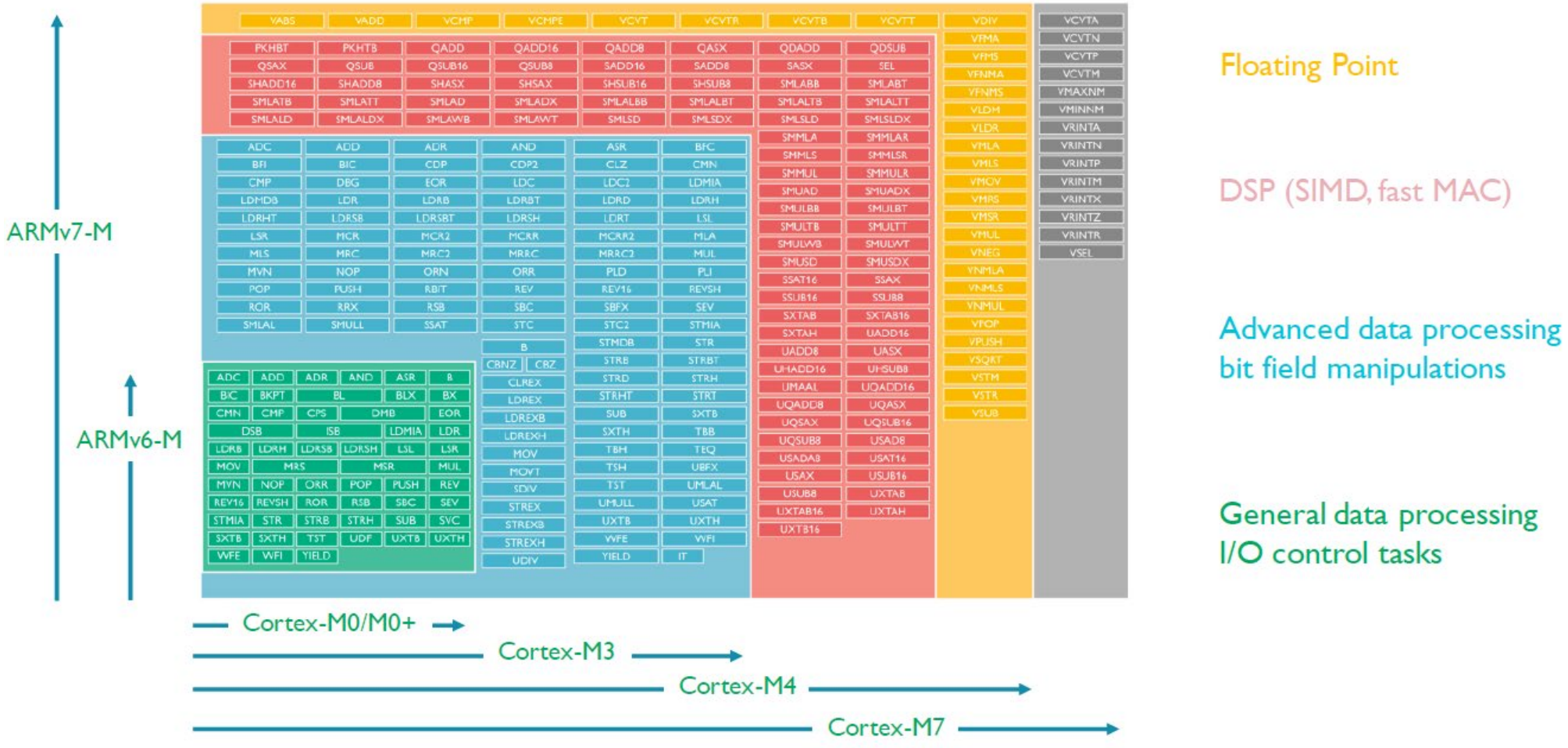
ARM Cortex-M Series

- We will how to measure performance, and choose the right microprocessor.



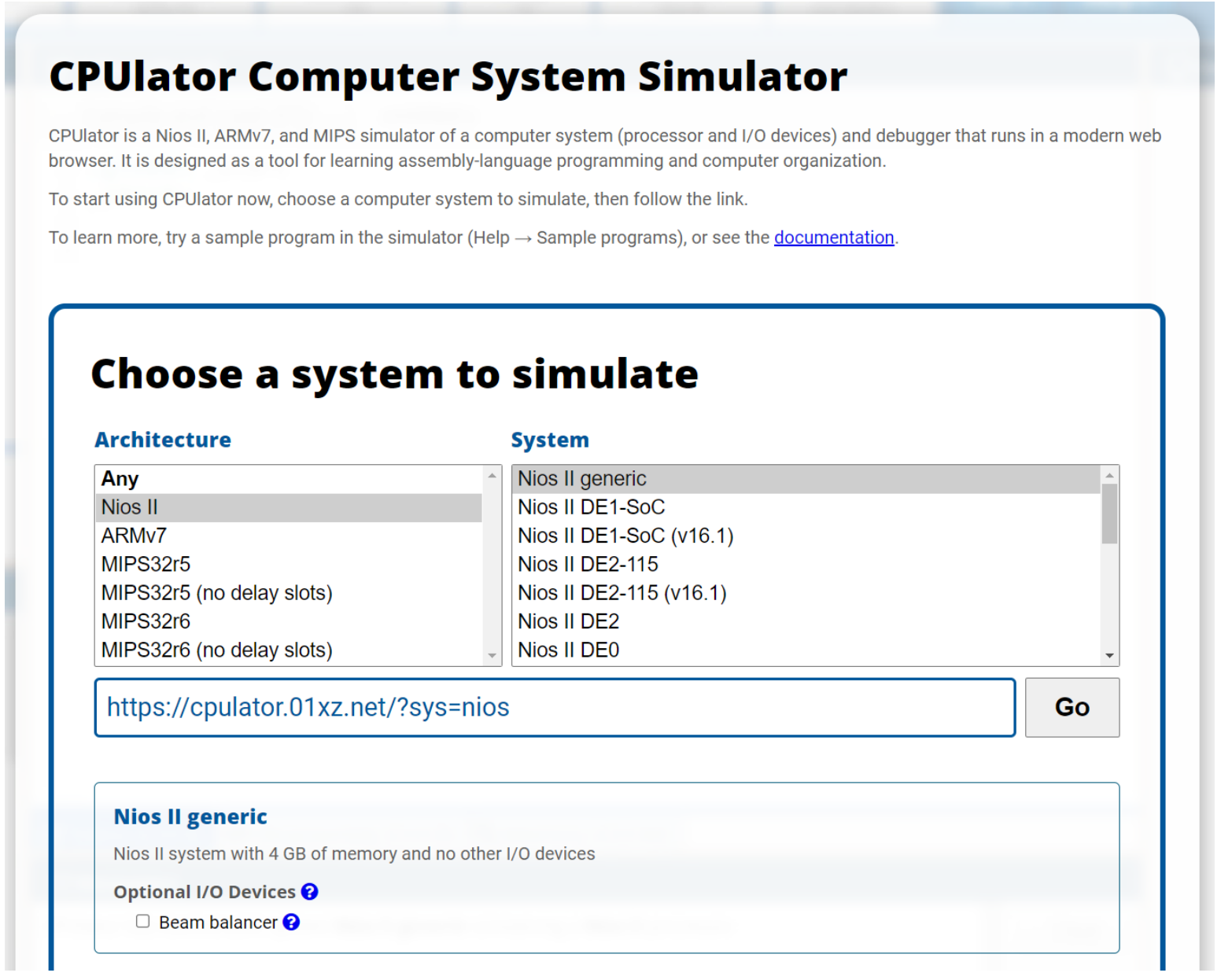
Instruction Set support in the Cortex-M processors

- All Cortex-M processors support an instruction set called Thumb



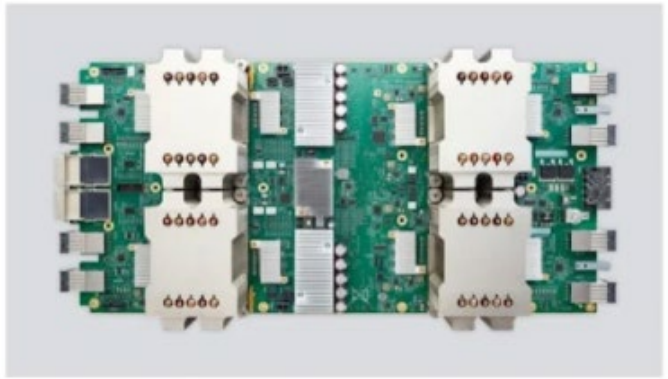
Online CPU Simulator

https://cpulator.01xz.net/



Google TPU Processor

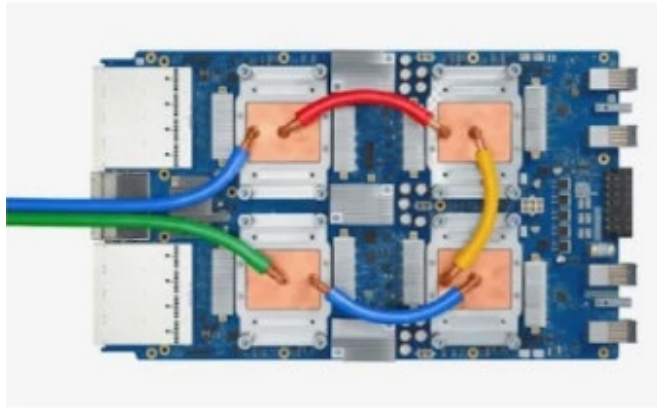
https://cloud.google.com/tpu/



Cloud TPU v2

180 teraflops

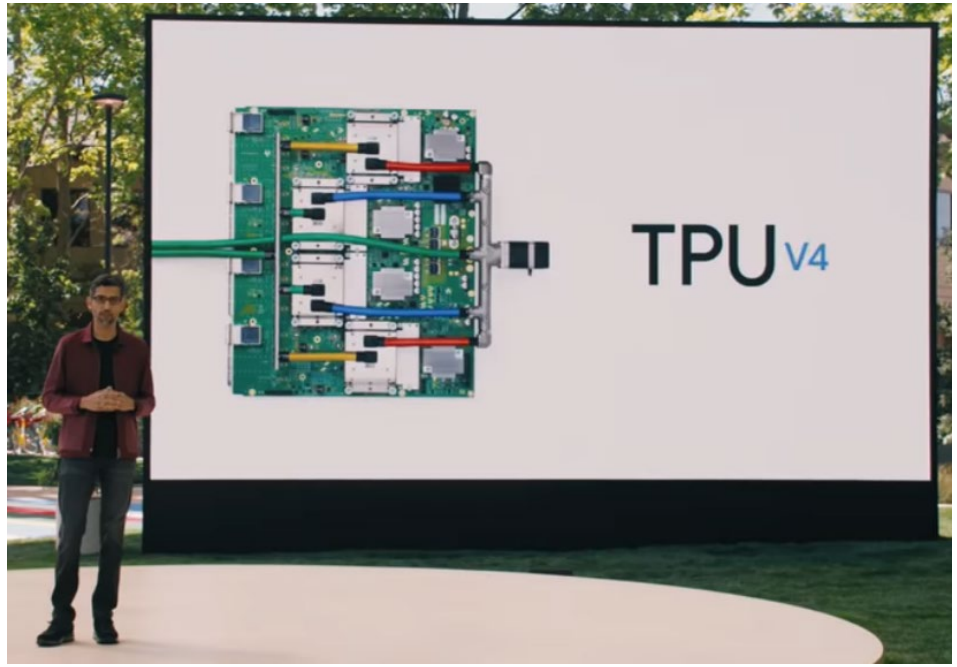
64 GB High Bandwidth Memory (HBM)

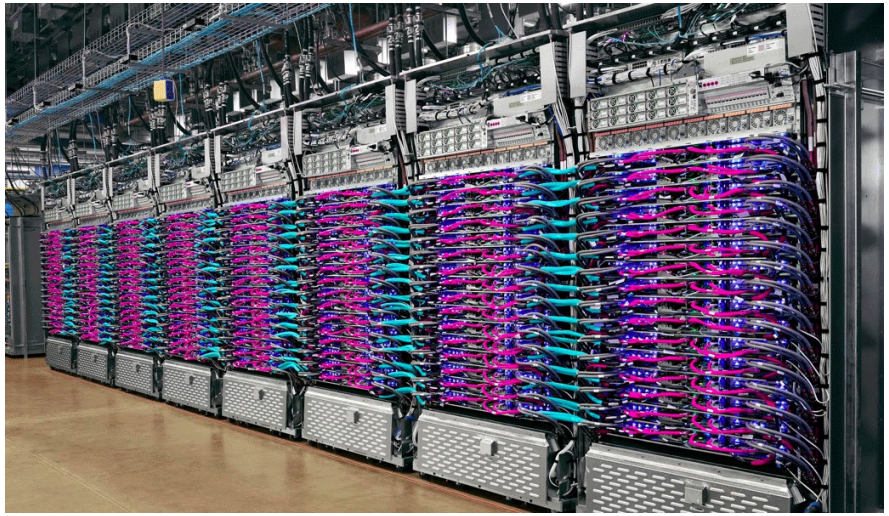


Cloud TPU v3

420 teraflops

128 GB HBM

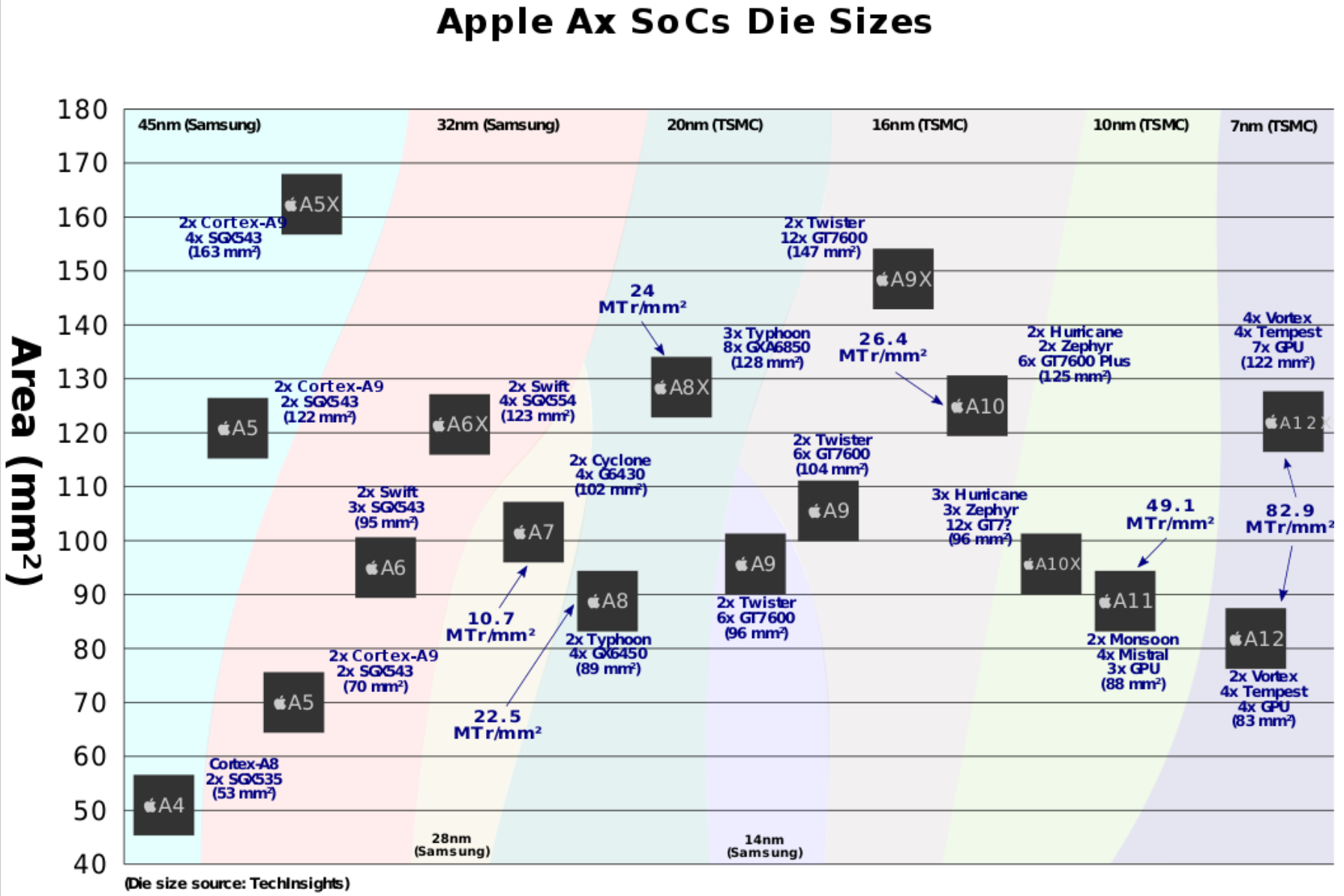




- Custom high-speed network offers over 100 petaflops of performance in a single pod
- Robust performance and low cost in Machine Learning training.
- Offer state-of-art models for different usage.

Apple Ax Processor Family

- Focus on the Area, and the processor technology.



Summary

- Introduction to the First Microprocessor and Moore's Law
- Concept of Computation, and the meaning of finite-state automation
- Microcontroller vs. Microprocessor
  - CPU with some memories
  - Microprocessor with both memory and I/O
- Stored Program Concept – Fetch-Decode-Execute Cycle
- RISC Philosophy, as compared to CISC
- Instruction Set Architecture and Instruction format
- Pipelined Instruction Execution
- Compilation and Assembling
- ARM Processor Family, Cortex-A, Cortex-R, Cortex-M series

Coming Next

- Software Design Basics
- Concurrency
- Software Engineering for Embedded Systems
- Event-Triggered Scheduling using Interrupts
- RTOS in Embedded Systems
- The ARM Cortex-M4 Processor Architecture