

EE3220 System-on-Chip Design

Lecture Note 2

Introduction to Modern Microprocessor

Recap on Lecture 1

The background features a large, light blue circle with a subtle gradient. To the left of the circle is a smaller, solid light blue sphere. To the right is a vertical rectangular shape with rounded corners and thin, vertical white lines. At the bottom right, a portion of another light blue sphere is visible. The overall aesthetic is clean and modern.

Navigation icons: back, forward, search, and other presentation controls.

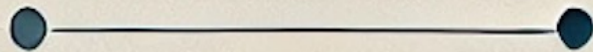


Lecture I Recap

- Course Overview
- Review Background
- Preview Design Boards and Tools
- Intro to Computer System
- Intro to Chip Design
- Intro to Embedded System
- Intro to ARM Architecture

This is simply an introduction. We will examine the details of “Embedded System” and “ARM Architecture” in subsequent lectures

Quick Quiz on Lecture 1



Questions on Acronyms in Lecture I

- What do these acronyms stand for? What do they mean?
 - CPU, MCU, SOC, ASIC, IC, ASSP
 - RAM, ROM, ADC, DAC, DMA
 - USB, LCD, UART, RF, IO
 - RTOS, HDL
 - KMAP, SOP, POS, HA, FA
 - IOT, IOE
 - HLS, FPGA, PLD, CPLD
 - SISD, SIMD, MISD, MIMD
 - CMOS, BJT, ECL
 - LSB, MSB

Questions on Logic Design

- How to determine if a two-complement number is negative?
 - Check the LSB. (T / F)
 - Check the MSB. (T / F)

Questions on Logic Design

- A 64-bit adder may be implemented with or without pipeline. Check if the following statements are correct.
 - A pipelined adder can provide higher bandwidth. (T / F)
 - A pipelined adder can provide lower latency. (T / F)
 - A pipelined adder may consume more power. (T / F)
 - A pipelined adder may run at a higher clock rate. (T / F)
 - A pipelined adder may consume more area. (T / F)

Questions on Logic Design

- What are the key Verilog / VHDL features absent in the C programming language?
 - Floating Point Representation (T / F)
 - Different Signal Levels (T / F)
 - Analog Representation (T / F)
 - Reactivity (T / F)
 - Concurrency (T / F)
 - Timing Control (T / F)
 - Object Oriented Design (T / F)
 - Clocking (T / F)

Questions on SOC

- Which of the following are essential components in a SOC design?
 - CPU (T / F)
 - RAM (T / F)
 - ROM (T / F)
 - SSD / HDD (T / F)
 - Internal Bus (T / F)
 - USB Port (T / F)
 - Ethernet Port (T / F)
 - Wi-Fi Port (T / F)

Questions on Processor Architecture

- Most low-cost IoT chips are based on which architectural type?
 - SISD (T / F)
 - SIMD (T / F)
 - MISD (T / F)
 - MIMD (T / F)

Questions on Processor Architecture

- x86 chips are based on which architectural type?
 - SISD (T / F)
 - SIMD (T / F)
 - MISD (T / F)
 - MIMD (T / F)

Questions on Processor Architecture

- GPU chips are based on which architectural type?
 - SISD (T / F)
 - SIMD (T / F)
 - MISD (T / F)
 - MIMD (T / F)

Questions on Processor Architecture

- Which of the following statements regarding Cache are correct.
 - Cache is used to improve Processor Performance. (T / F)
 - Cache is used to reduce Processor Power. (T / F)
 - L1 Cache is larger than L2 Cache. (T / F)
 - L1 Cache is faster than L2 Cache. (T / F)
 - Cache and Register File are both CPU Memory (T / F)
 - Cache design is like Register File design. (T / F)

Questions on SOC Design

- What are the key purposes of functional verification?
 - Check for any performance issues. (T / F)
 - Check for any set-up time issues. (T / F)
 - Check for any hold time issues. (T / F)
 - Check for any hardware logic issues. (T / F)
 - Check for any firmware logic issues. (T / F)
 - Check for any reliability issues. (T / F)
 - Check for any cost issues. (T / F)
 - Check for any production issues. (T / F)

Questions on SOC Design

- What are the key advantages of using an FPGA versus an ASIC in an embedded system?
 - Reduce Unit Cost? (T / F)
 - Reduce Development Cost? (T / F)
 - Reduce Development Time? (T / F)
 - Reduce Form Factor? (T / F)
 - Improve Performance? (T / F)
 - Improve Reliability? (T / F)
 - Reduce Power Consumption? (T / F)

Questions on SOC Design

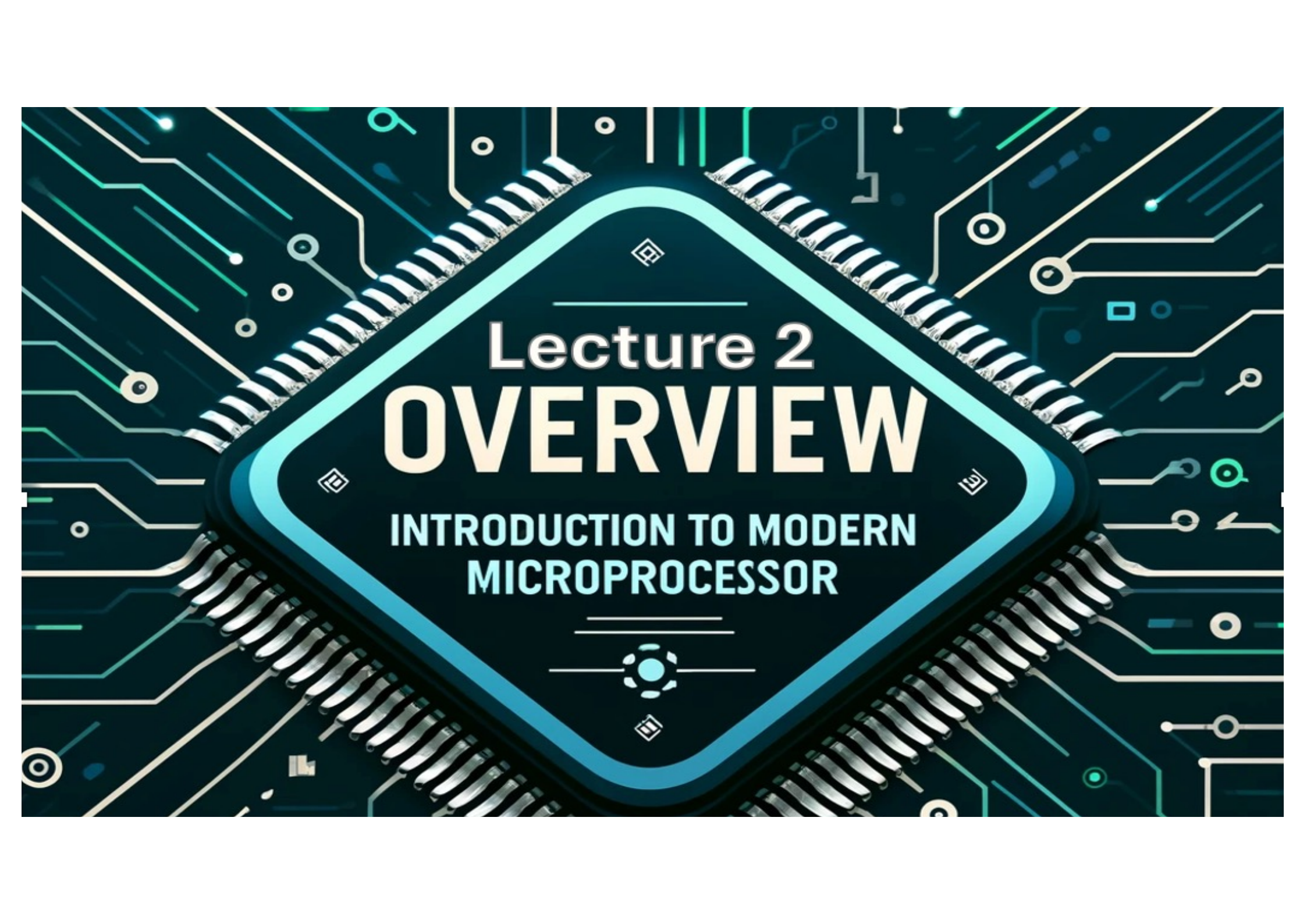
- Using the negative binomial yield model with the defect density = 1 defect per 100mm².

Check if the following statements are correct.

- Yield can be improved with smaller die size. (T / F)
- Yield can be improved with lower defect density. (T / F)
- Yield = 0 if die size > 100mm². (T / F)
- Yield = ~0.42 if die size = 100mm². (T / F)
- Yield is lower in 3nm process than in 5nm process. (T / F)
- Yield can affect the total costs. (T / F)

Questions on SOC Design

- Which of the following statements regarding SOC performance are correct.
 - Bandwidth (MB/s) is a key performance metric. (T / F)
 - Latency (us) is a key performance metric. (T / F)
 - Bandwidth is more important than latency. (T / F)
 - Higher performance usually means higher power. (T / F)
 - Higher performance usually means larger chip size. (T / F)
 - Faster clock usually means higher performance. (T / F)

The background of the slide is a dark teal color with a complex pattern of glowing blue and white circuit lines. In the center, there is a large, diamond-shaped microprocessor chip with a glowing blue border. The chip has many pins extending from its edges. The text is centered on the chip.

Lecture 2 **OVERVIEW**

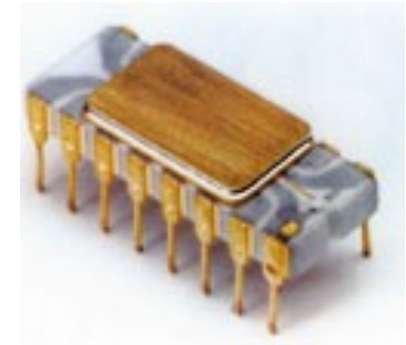
INTRODUCTION TO MODERN
MICROPROCESSOR

Lecture 2 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 have become **MORE POWERFUL** and **CHEAPER**.



Moore's Law

The number of transistors on a chip will roughly **DOUBLE** every **TWO YEARS**.

- Gordon Moore's prediction from 1965!
- He had been right for more than 50 years...
- **WHY?**
- How often do you change your phone or computer?

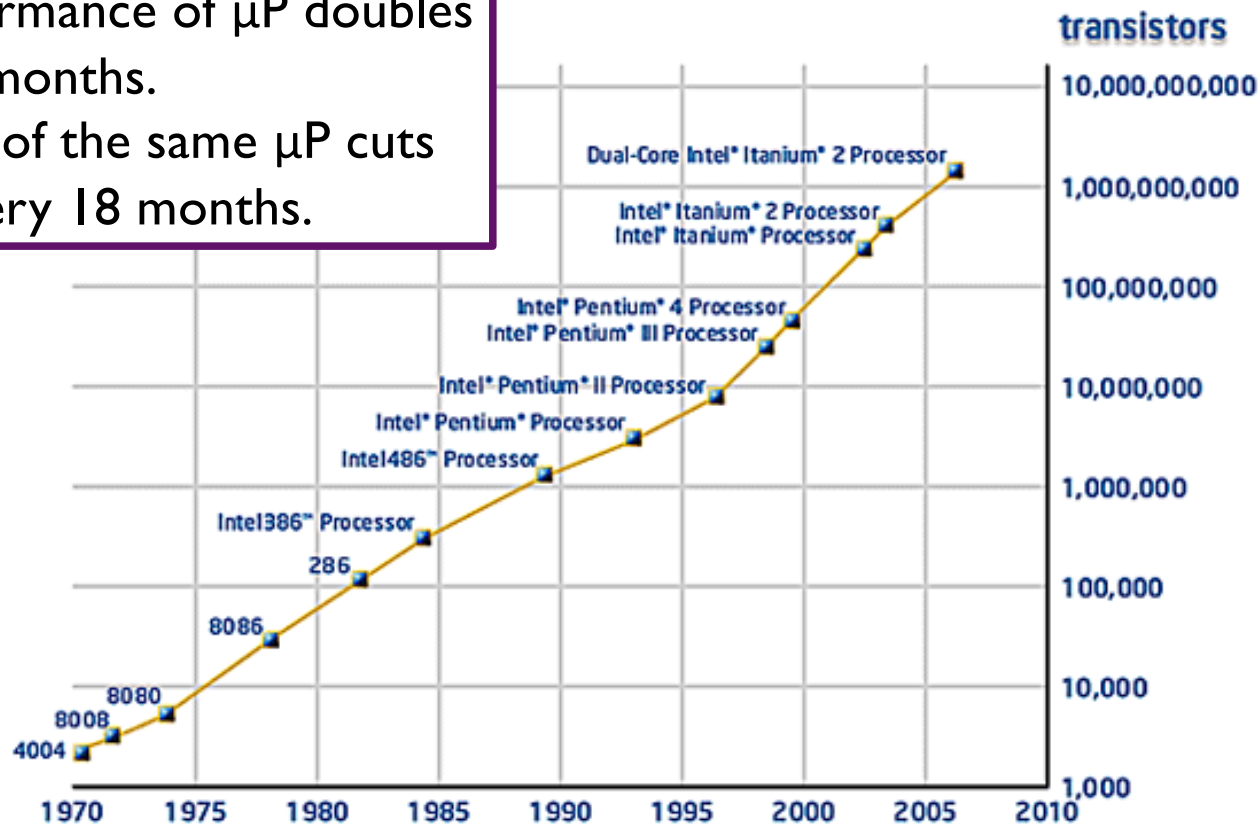


Founders of Intel: Andy Grove, Robert Noyce and Gordon Moore

Moore's Law

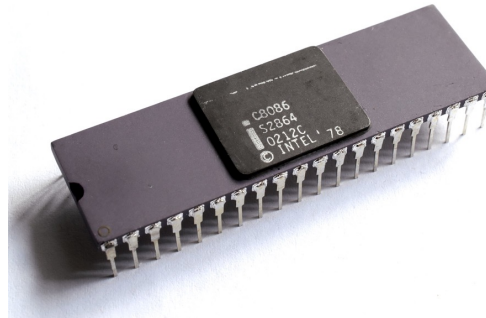
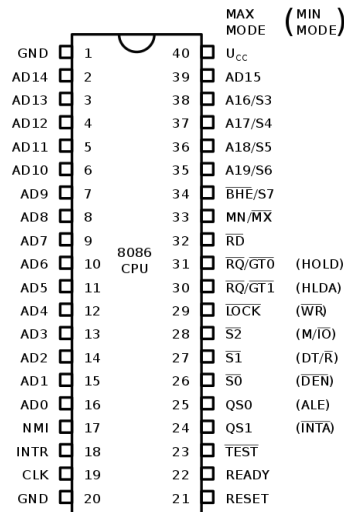
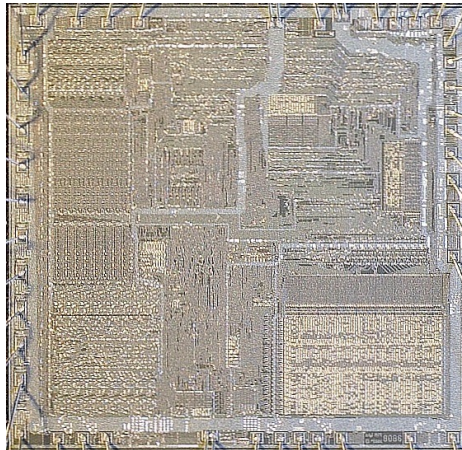
The performance of μ P doubles every 18 months.

The price of the same μ P cuts by half every 18 months.



Classic: Intel 8086 Processor & Chip

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



Intel 8086 registers

1 9 1 8 1 7 1 6 1 5 1 4 1 3 1 2 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	Source Index
0 0 0 0	DI	Destination Index
0 0 0 0	BP	Base Pointer
0 0 0 0	SP	Stack Pointer

Program counter

0 0 0 0	IP	Instruction Pointer
---------	----	---------------------

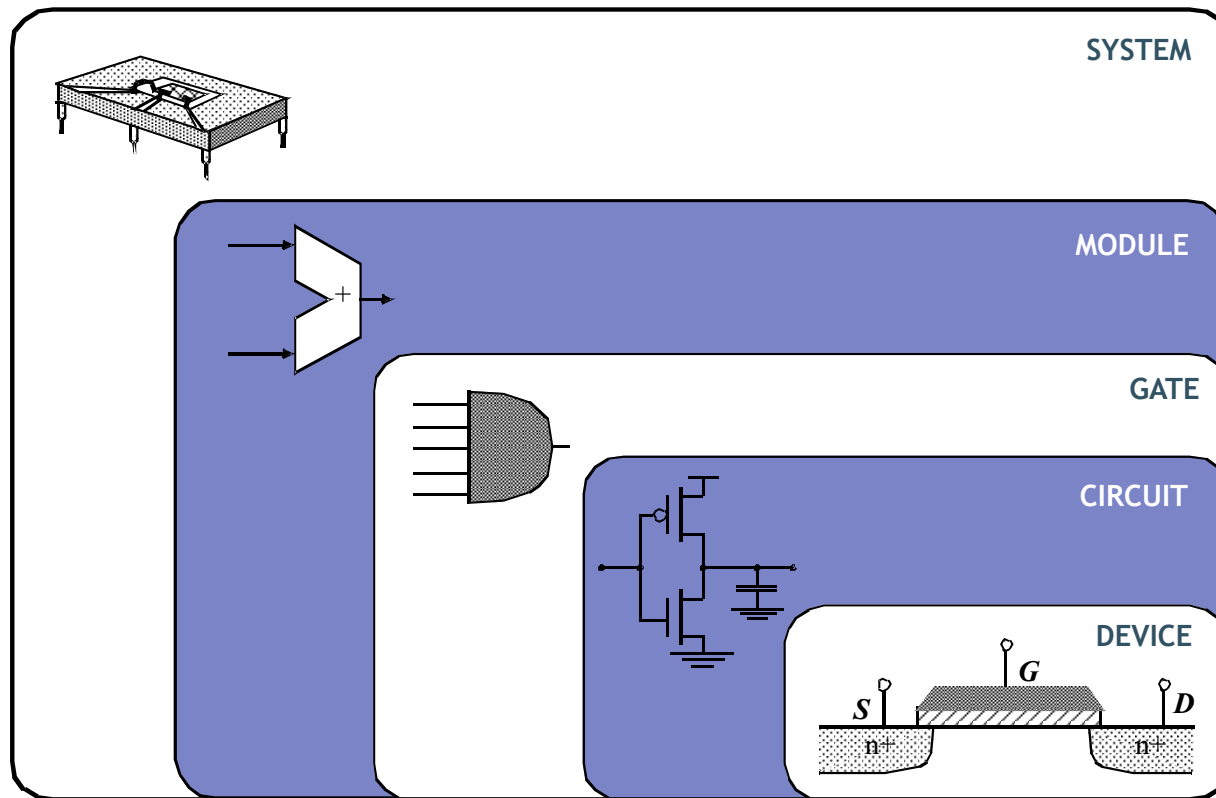
Segment registers

CS	0 0 0 0	Code Segment
DS	0 0 0 0	Data Segment
ES	0 0 0 0	Extra Segment
SS	0 0 0 0	Stack Segment

Status register

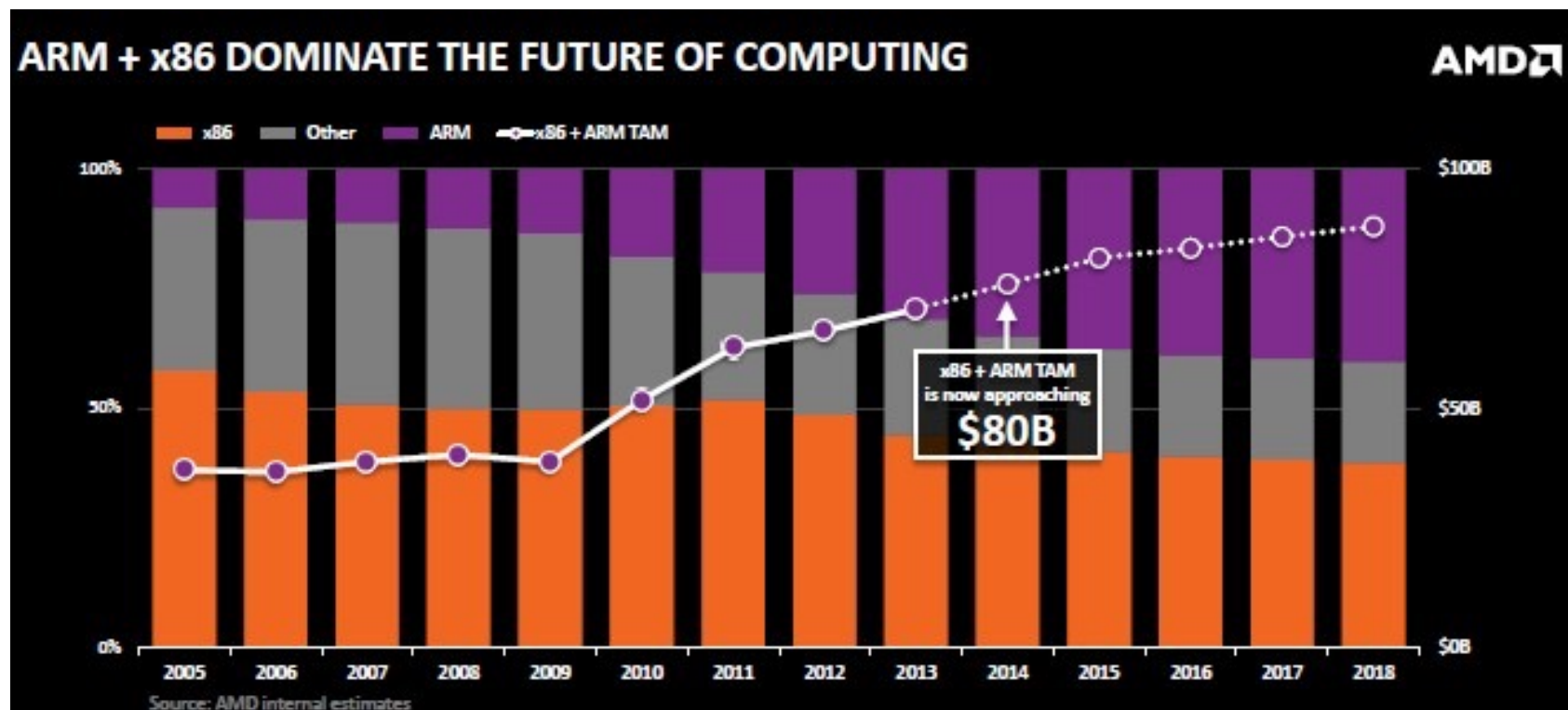
- - - - O D I T S Z - A - P - C	Flags
---------------------------------	-------

Design Abstraction Levels



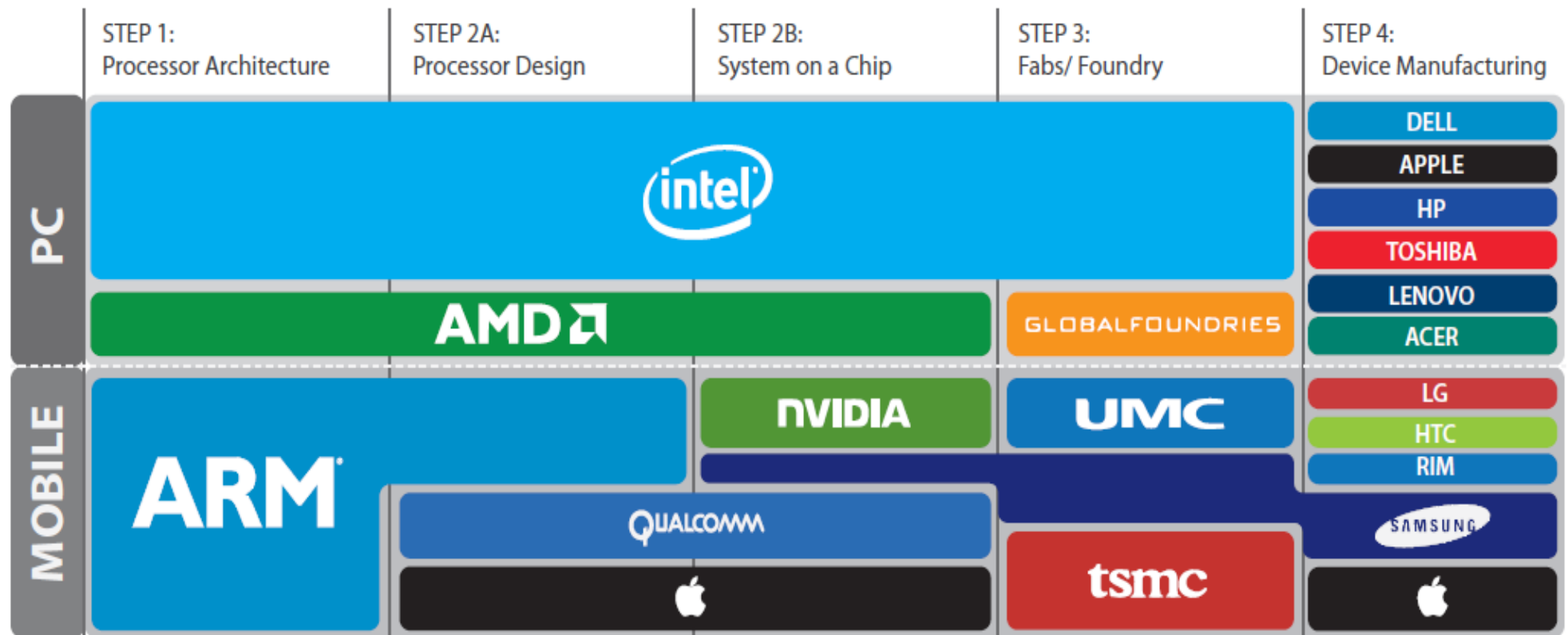
x86 vs.ARM

- Mobile computing market is continuously growing.

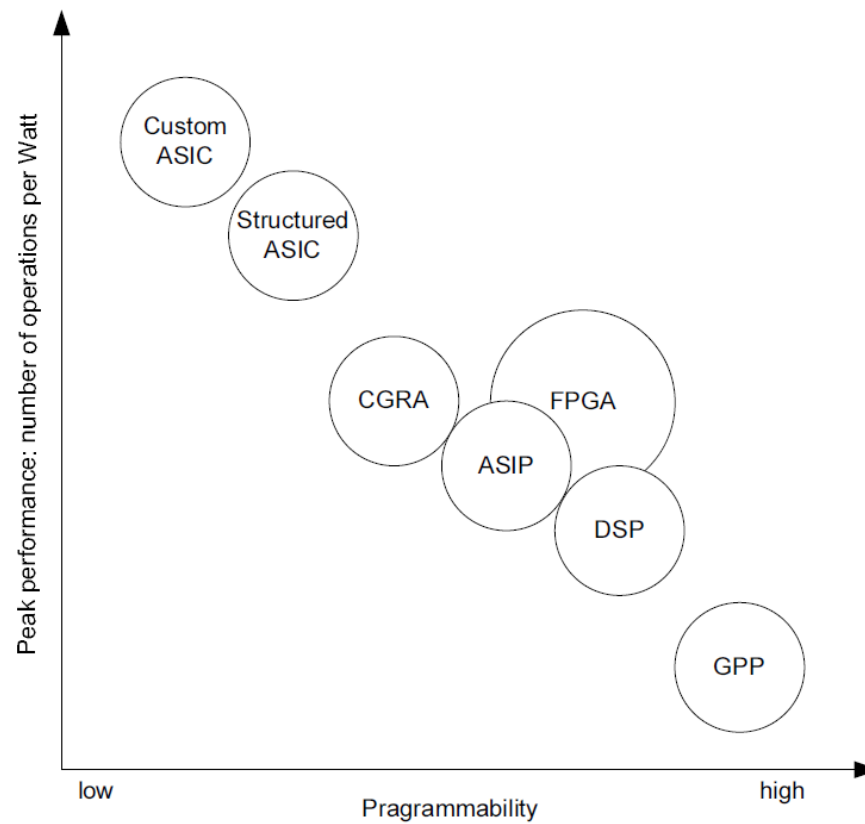


Mobile vs PC Processors

Processor Value Chain



Technology Comparison: Programmability vs Performance



What is Computer?

- A Computer is a Machine that can perform simple calculations
- 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.

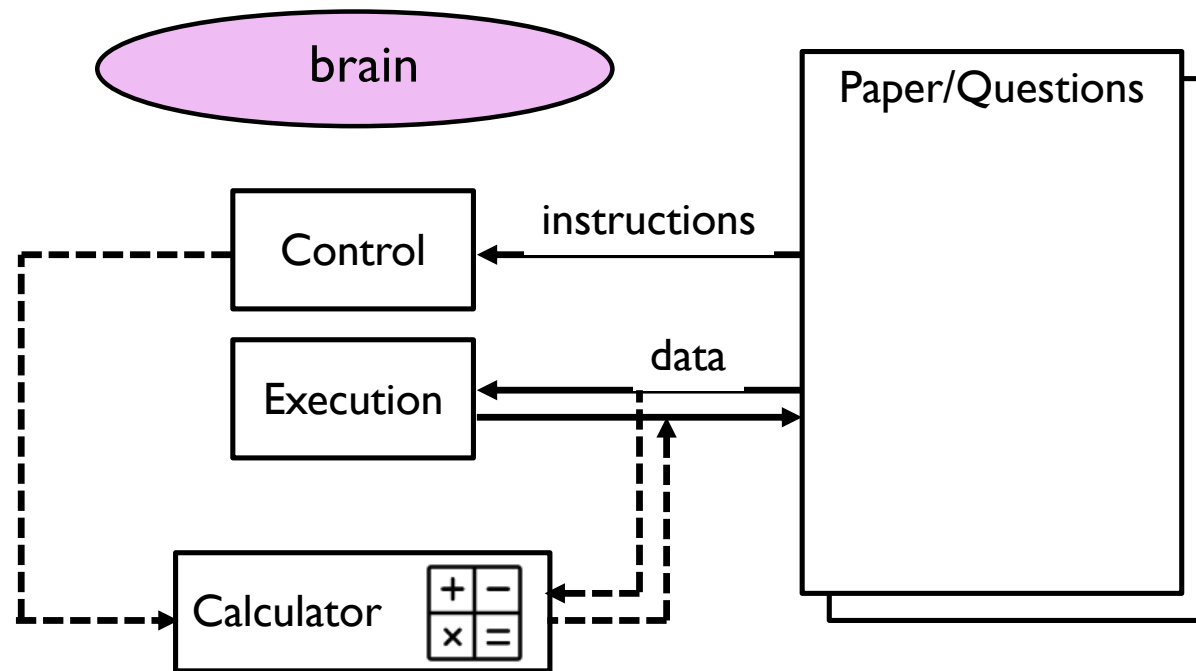
Solving different TYPES of Math problems

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



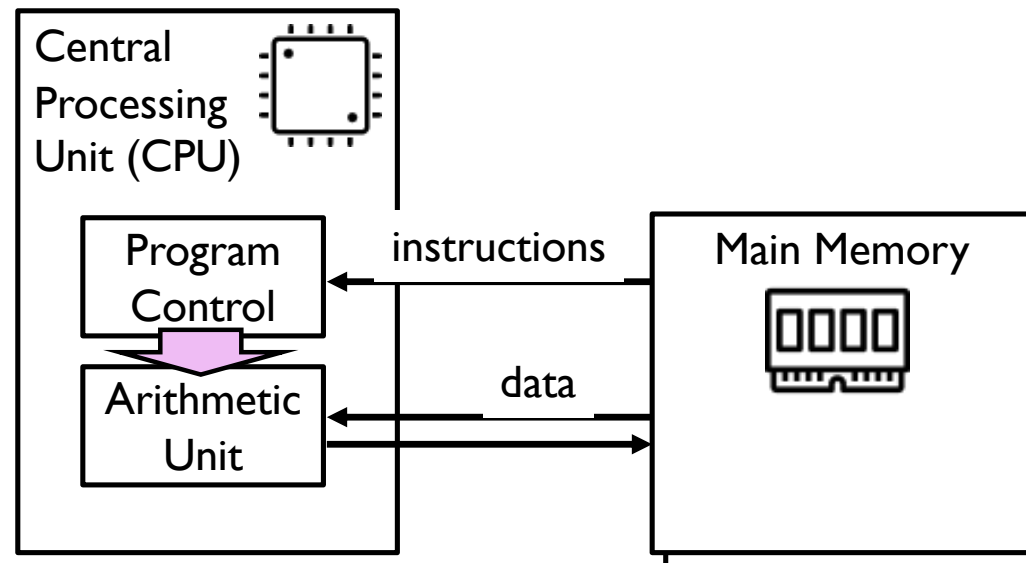
Concept of Computation

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



Concept of Computation (Cont')

How about machine computations?

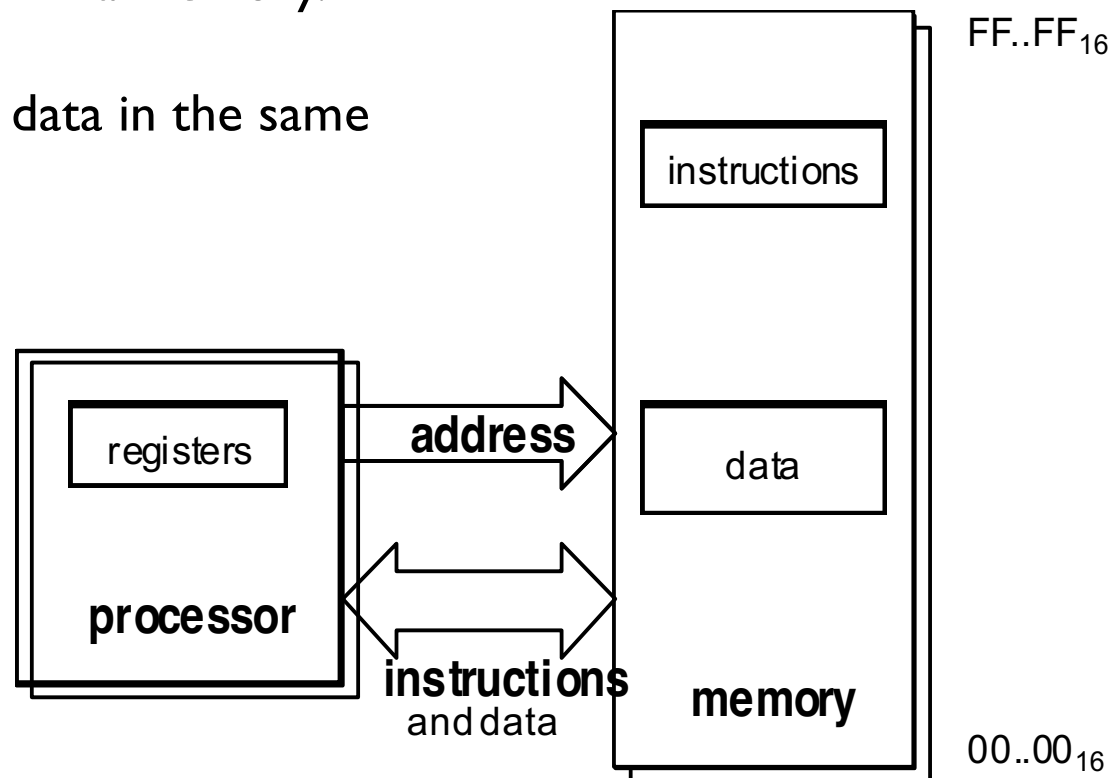


Is there anything missing?

The state in a stored-program digital computer

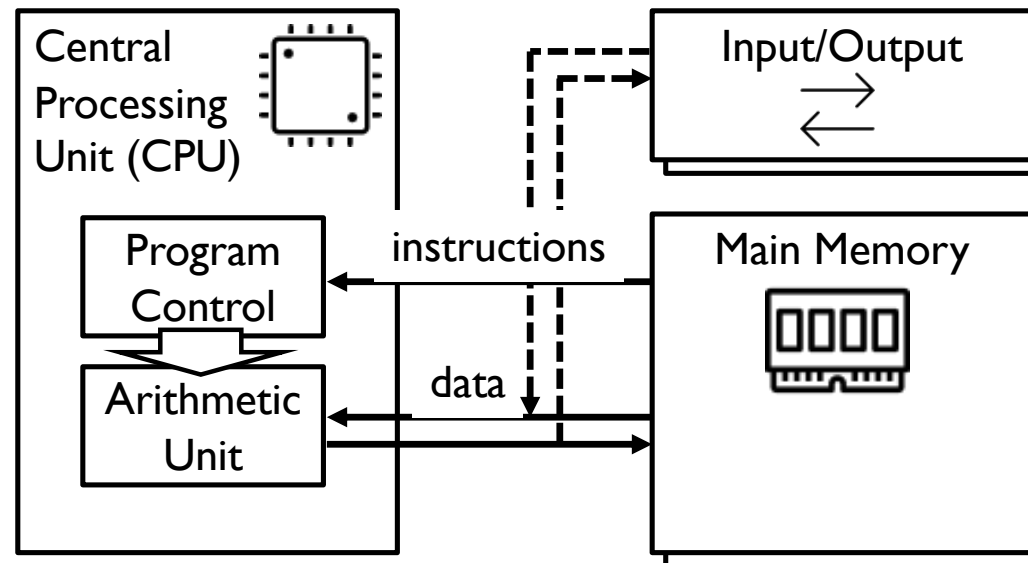
A processor is a finite-state automation that executes instructions held in a memory.

Keep its instructions and data in the same memory system



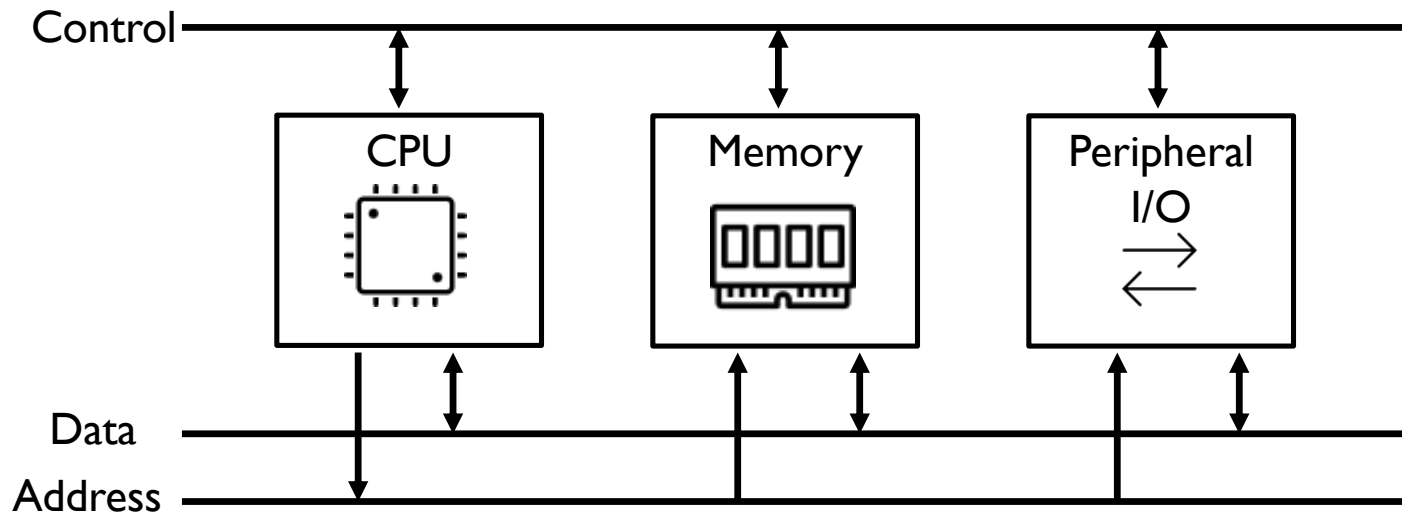
Concept of Computation (Cont')

I/O provides extra data and allow interactions.



Do we have a simple model to describe a computer?

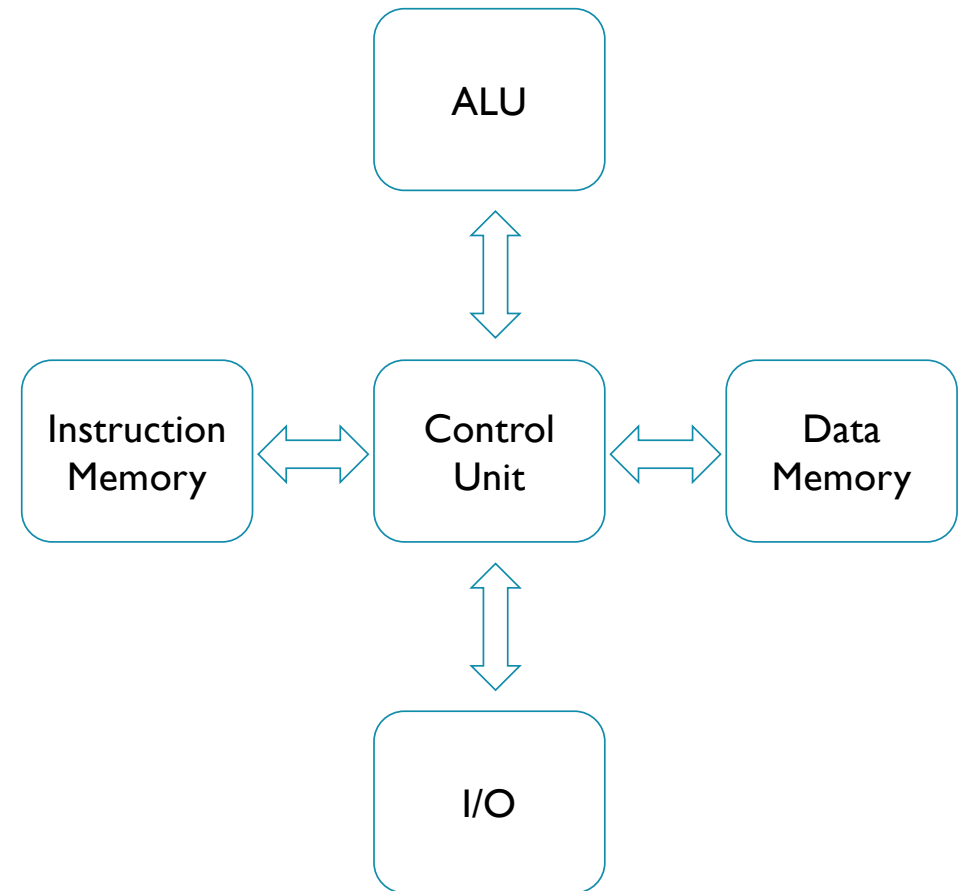
Von Neumann Architecture



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.

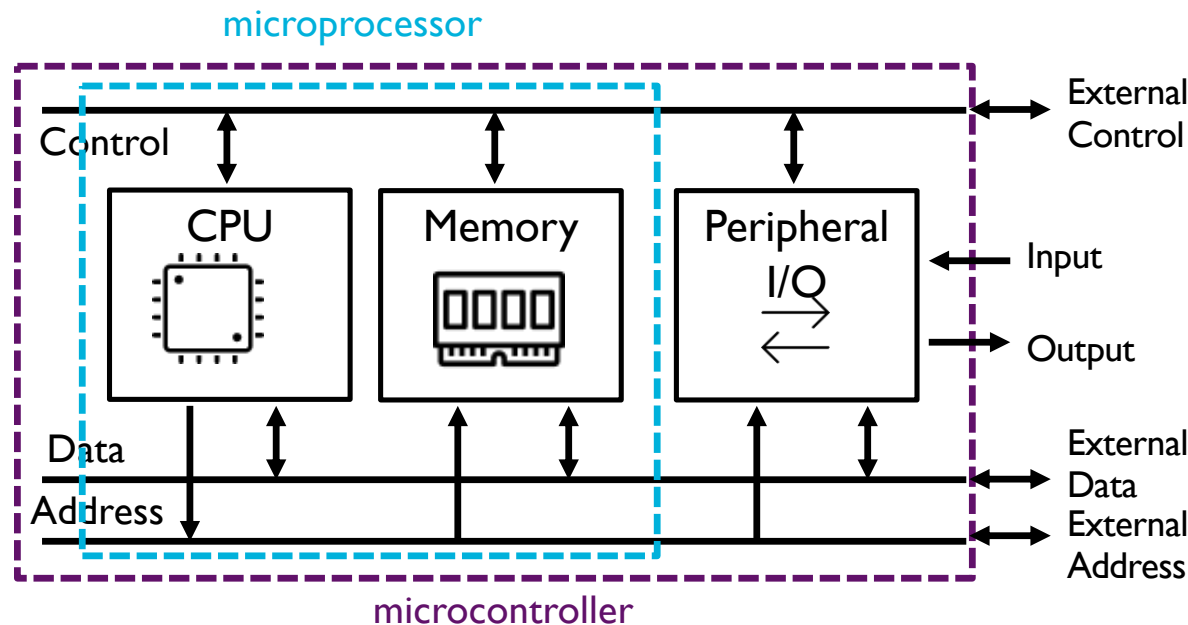


Microcontroller vs Microprocessor

- 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

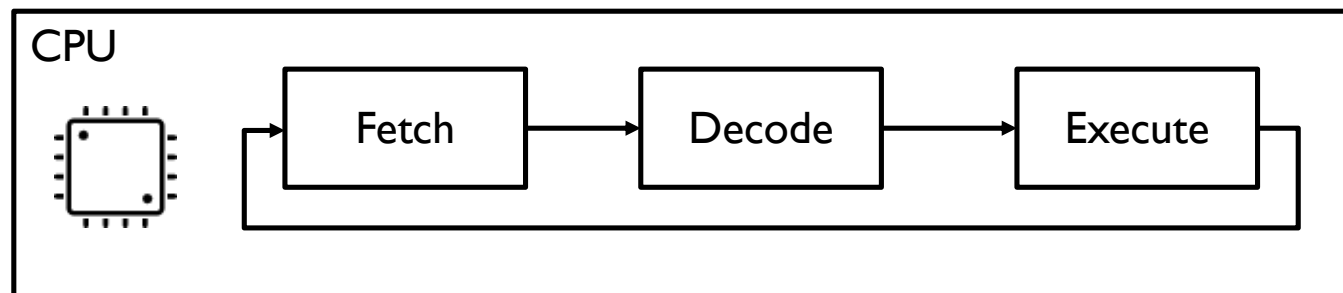
What is inside my computer program?

Stored Program Concept

- CPU executes Instructions stored in Memory.
 - Known as “Stored Program” Concept
- Recall there are 2 two kinds of memory to store programs and data.
 - RAM – Random Access Memory
 - Can be used for both Program and Data
 - ROM – Read-Only Memory
 - Can be used for fixed Program & Constant Data
- Different computers have different ways in arranging Memory (Computer Architecture)

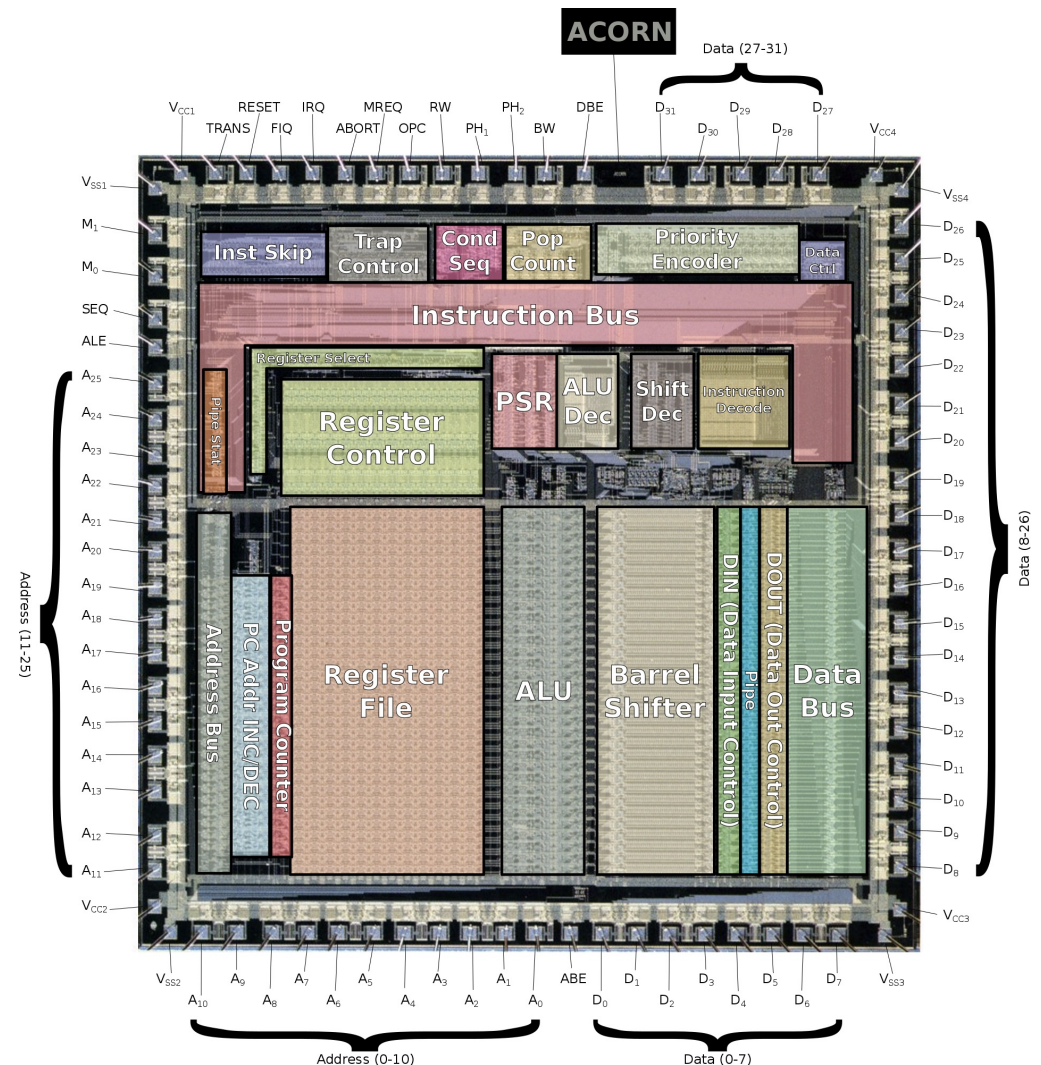
Fetch-Decode-Execute Cycle

- 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

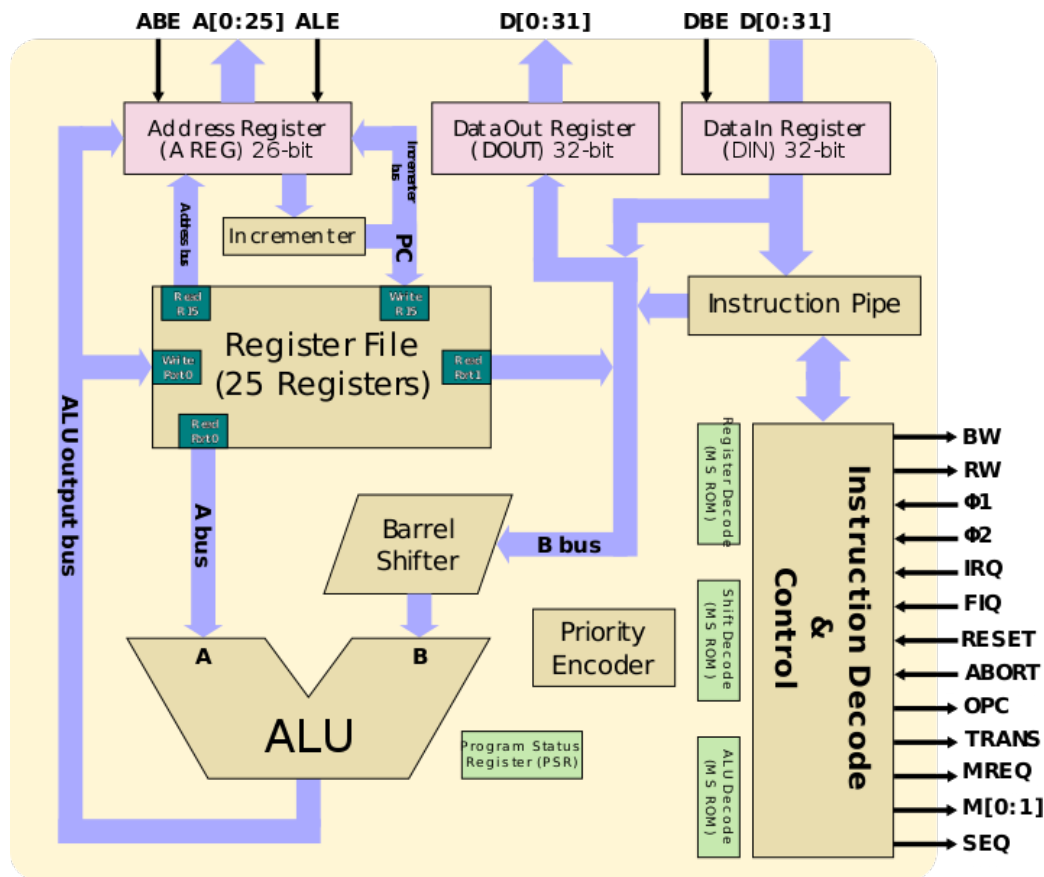


ARM Die Photo

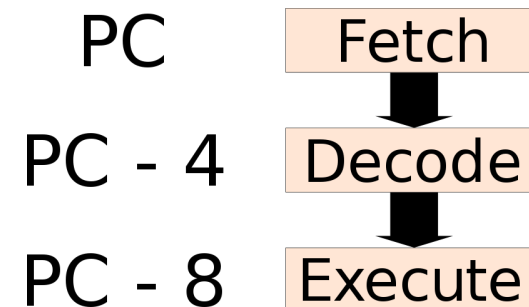
- ARMI - Microarchitectures – Acorn
- ARMI was the first ARM microarchitecture designed and realized by Acorn Computers for the BBC Computer Literacy Project.
- ARMI was introduced in 1985 and was extended to be used as a coprocessor in the Acorn's BBC Micro microcomputers.



ARMI Block Diagram



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



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 $\mu\text{P}/\mu\text{C}$ 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 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 the 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 and flexibility and is better for general-purpose applications.

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.

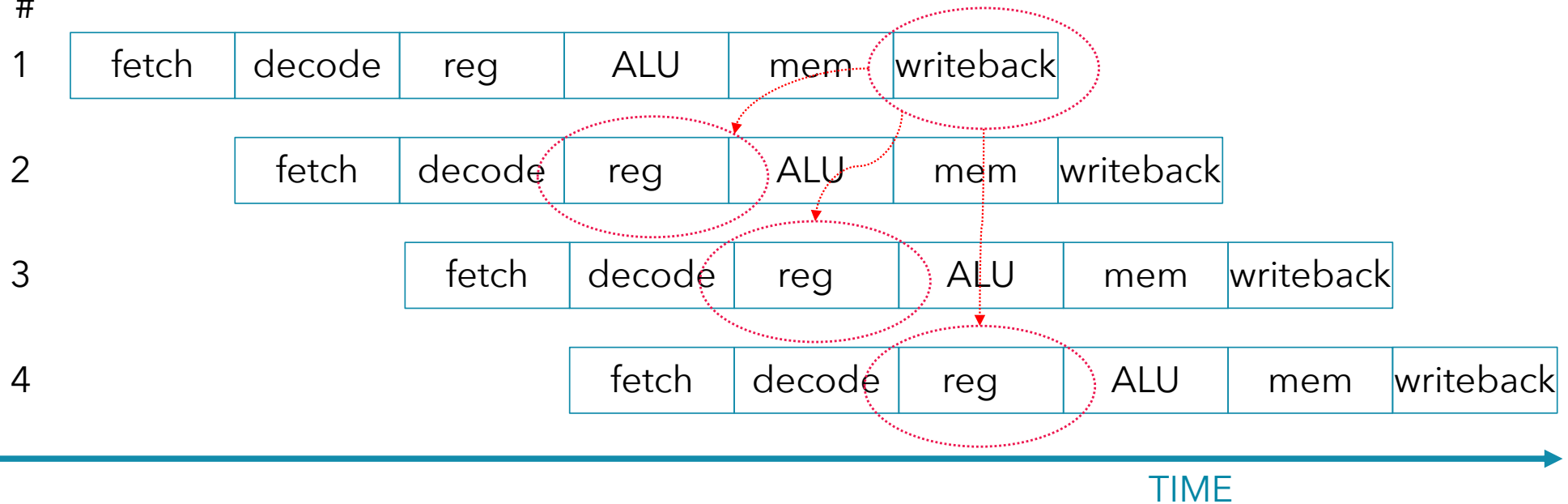
RISC Philosophy

- 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

Instruction

#



Check for data dependencies (data hazards):

RAR (Read-After-Read), **RAW** (Read-After-Write), **WAR** (Write-After-Read), **WAW** (Write-After-Write)

RISC vs. CISC

- Brief Comparison:

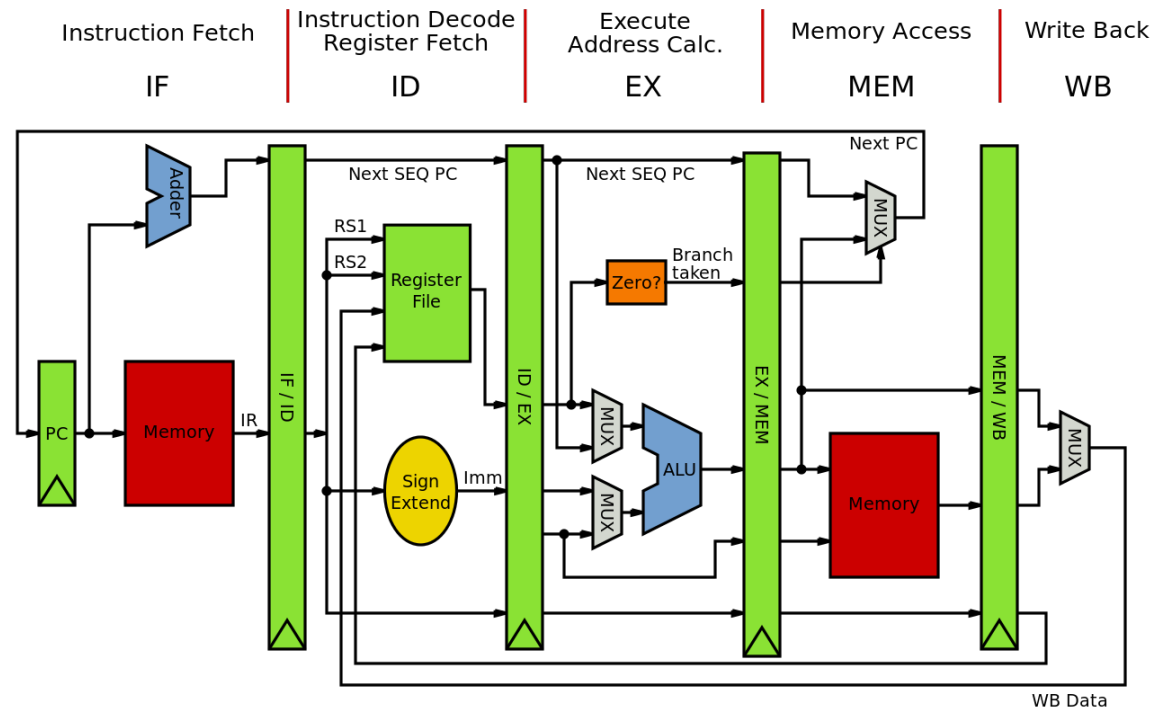
Reduced Instruction Set Computing (RISC)	Complex Instruction Set Computers (CISC)
Emphasis on software.	Emphasis on hardware.
Single-clock, simple instruction set.	Includes multi-clock complex instructions
Register to register: "LOAD" and "STORE" are independent instructions.	Memory-to-memory: "LOAD" 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

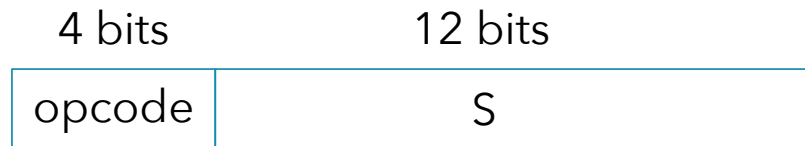
- Which is better?
 - Intense debate in the 1980s and 1990s.
 - RISC can support faster **CLOCK SPEED**.
 - RISC can support more **COMPILER OPTIMIZATION**.
 - CISC can provide better **CODE DENSITY**.
 - Hardware can also translate CISC-style instructions into RISC-style instructions.
- Performance depends on:
 - Instruction numbers per Task.
 - Instruction executed per Clock.
 - Clock Speed.

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.



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 $ACC \geq 0$ $PC := S$
JNE S	0110	if $ACC \neq 0$ $PC := S$
STP	0111	stop

Different types of Instruction Format

f bits	n bits	n bits	n bits	n bits
function	op 1 addr.	op 2 addr.	dest addr.	next i addr.

4-address instruction format

f bits	n bits	n bits	n bits
function	op 1 addr.	op 2 addr.	dest addr.

3-address instruction format

f bits	n bits	n bits
function	op 1 addr.	dest addr.

2-address instruction format

f bits	n bits
function	op 1 addr.

1-address instruction format

f bits
function

0-address instruction format

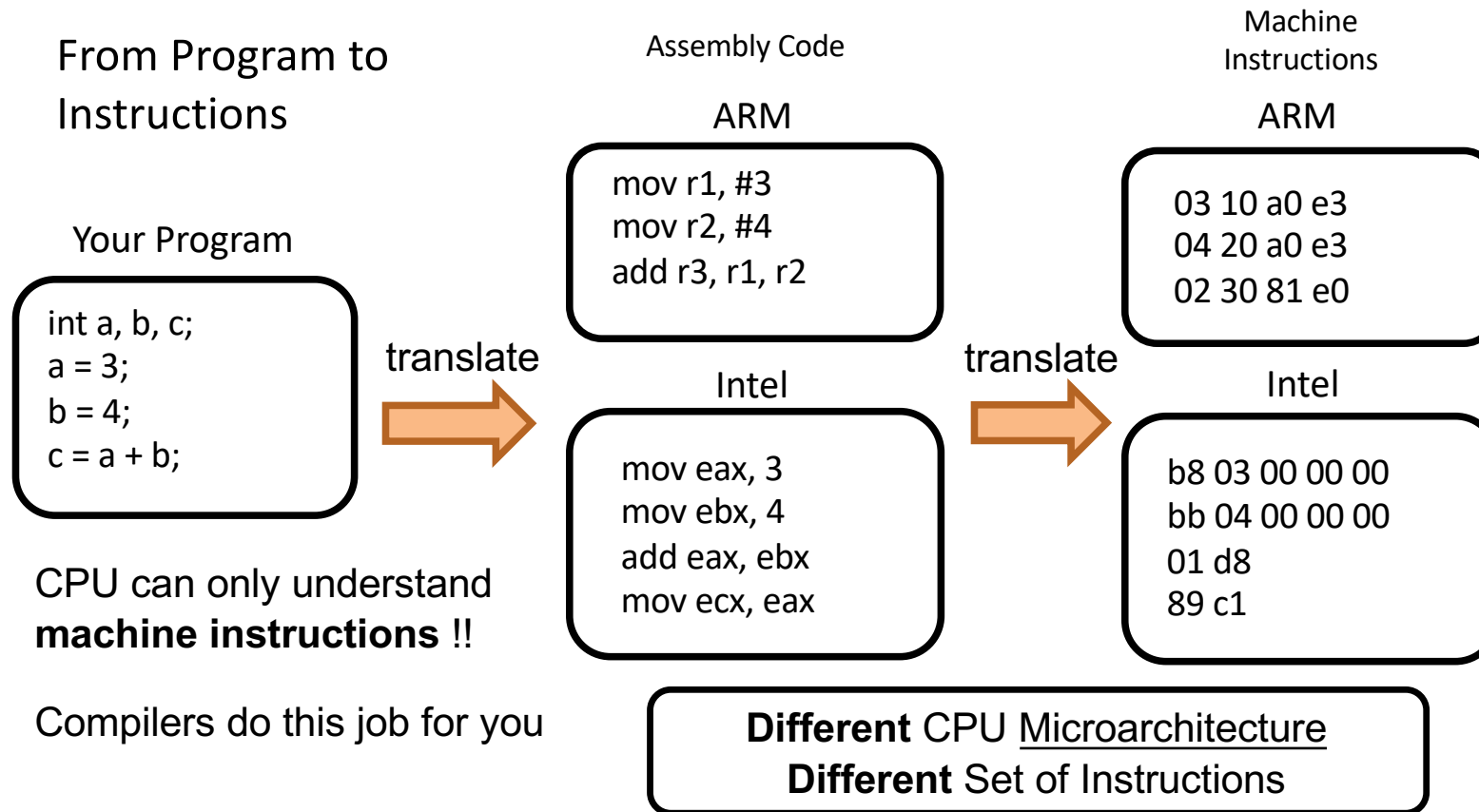
MIPS ISA

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

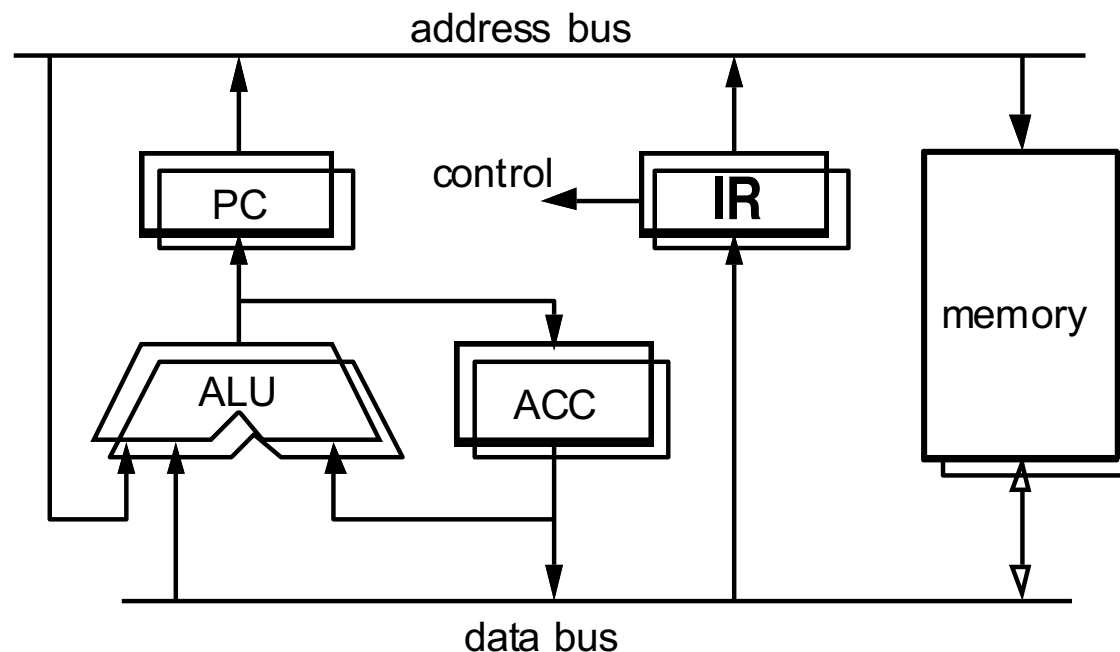
Instruction type	Dynamic usage
Data movement	43%
Control flow	23%
Arithmetic operations	15%
Comparisons	13%
Logical operations	5%
Other	1%

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?





```
graph TD; A[SOC DESIGN] --> B[HARDWARE DESIGN]; A --> C[FIRMWARE DESIGN];
```

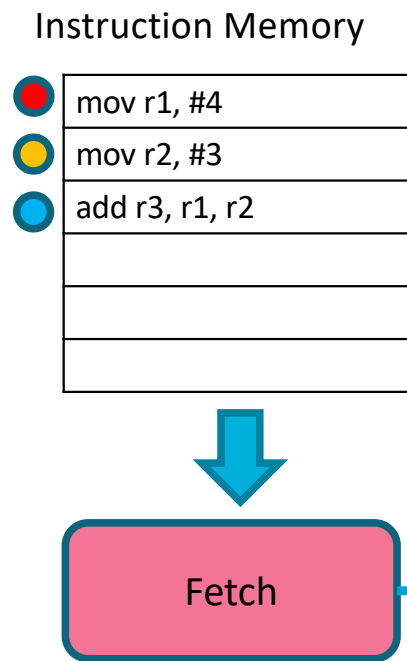
SOC DESIGN

HARDWARE DESIGN

FIRMWARE DESIGN

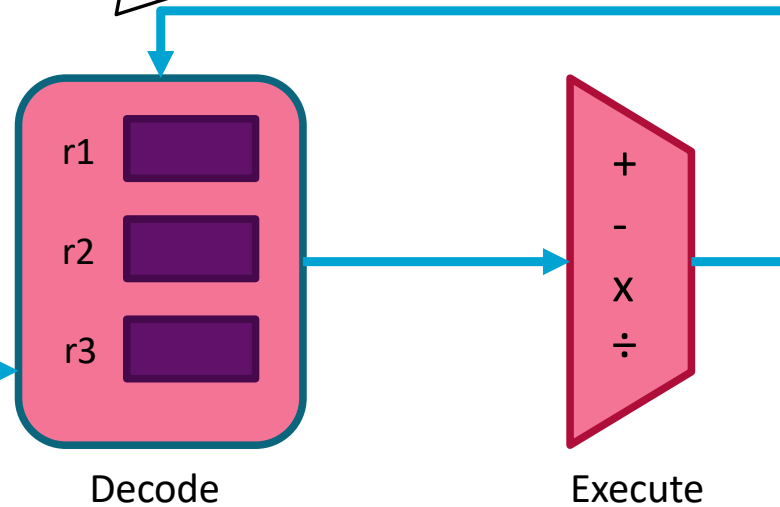
A 3-Stage CPU

A Simple 3-Stage “ARM” CPU



Three Stage CPU

1. Fetch - fetch an instr. from mem
2. Decode - interpret the instr. properly
3. Execute - execute the instr.



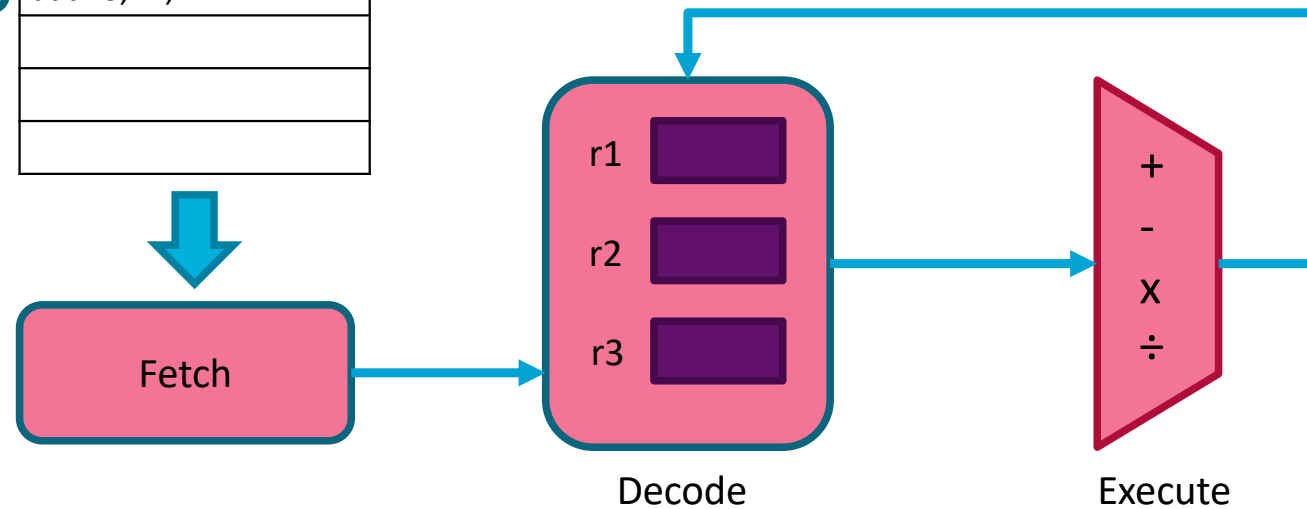
A 3-Stage CPU

A Simple 3-Stage “ARM” CPU

Instruction Memory

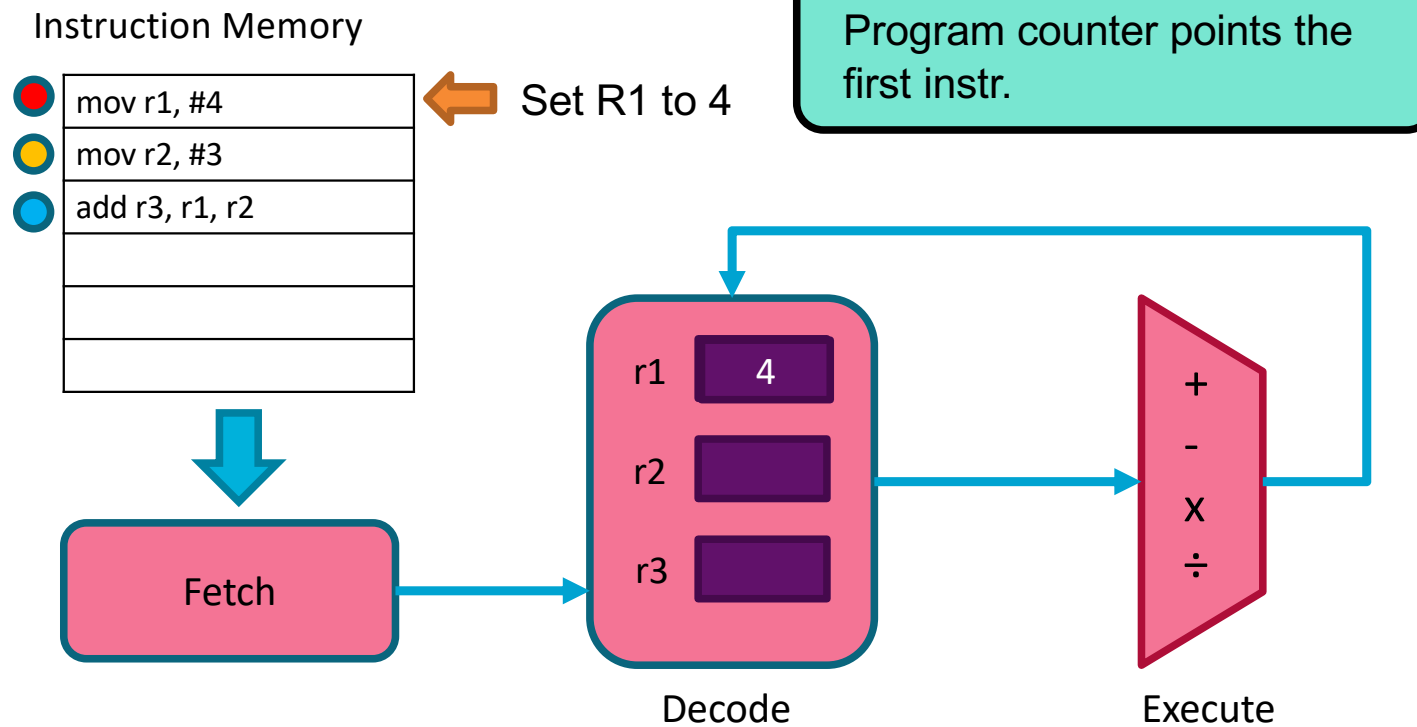
●	mov r1, #4
●	mov r2, #3
●	add r3, r1, r2

Program Instructions are loaded into **instr mem** **before** program execution



A 3-Stage CPU

A Simple 3-Stage “ARM” CPU



A 3-Stage CPU

A Simple 3-Stage “ARM” CPU

Instruction Memory

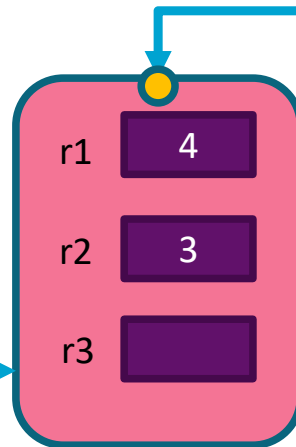
mov r1, #4
mov r2, #3
add r3, r1, r2



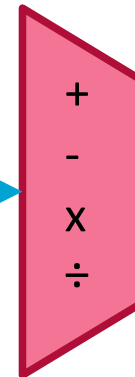
Set R2 to 3



Fetch



Decode



Execute

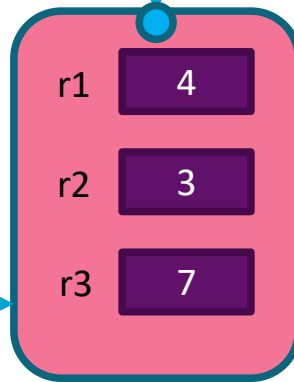
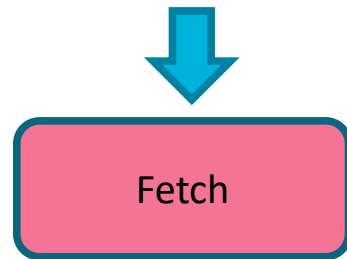
Similar to the first instr.
Write 3 to r2

Processor Micro-Architecture

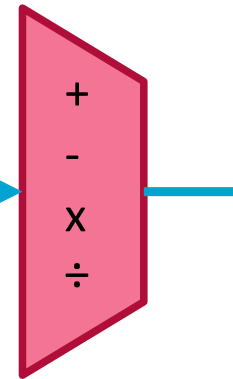
A Simple 3-Stage “ARM” CPU

Instruction Memory

mov r1, #4
mov r2, #3
add r3, r1, r2



Decode



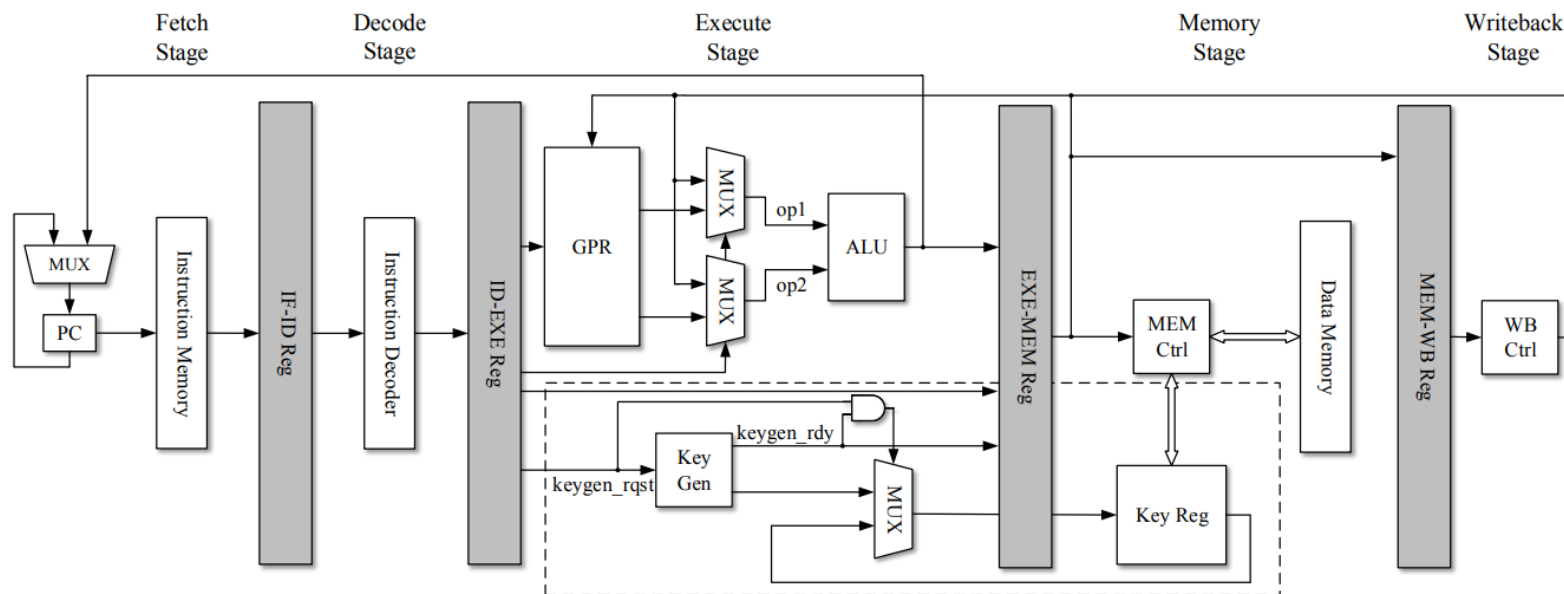
Execute

Write the result to r3

← $R3 \leftarrow R1 + R2$

Processor Micro-Architecture

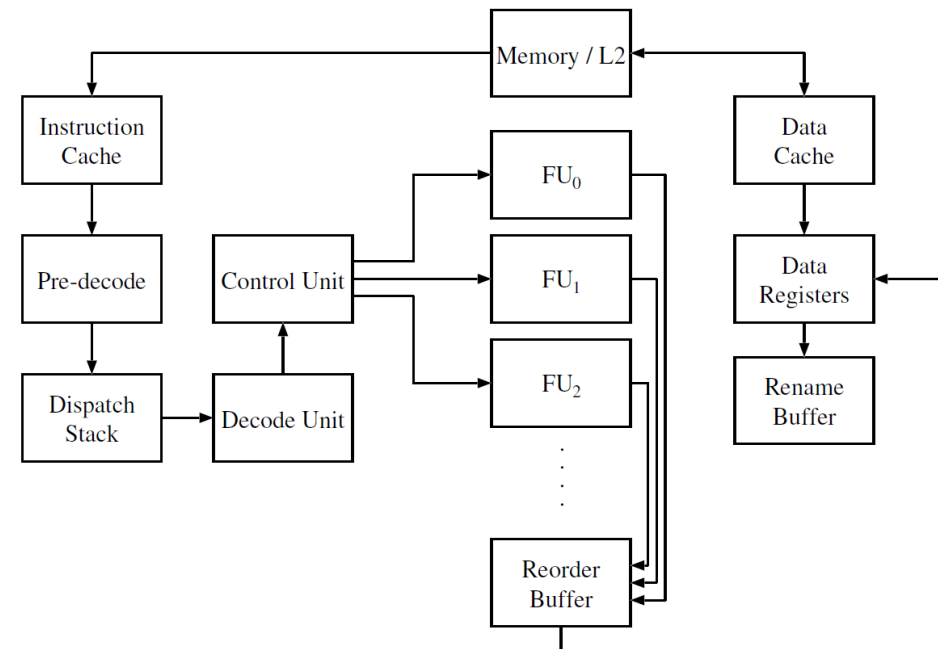
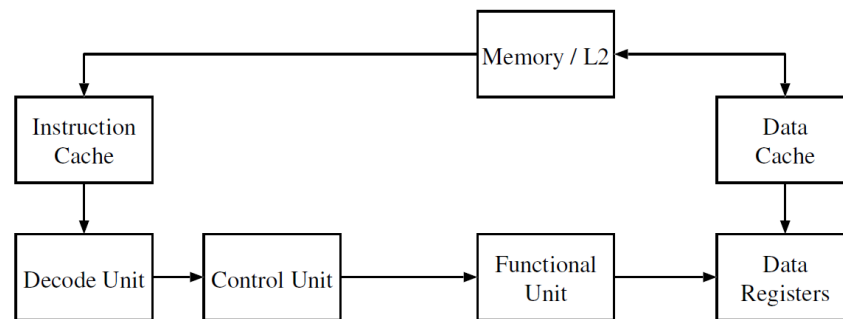
- Machine Code running on this architecture
- Compile a high-level language to machine code



CityU - Secure Microprocessor

Sequential Scalar vs Out-of-order Superscalar Processor

- Scalar vs Superscalar Issue & Execution
- Sequential vs Out-of-order Execution

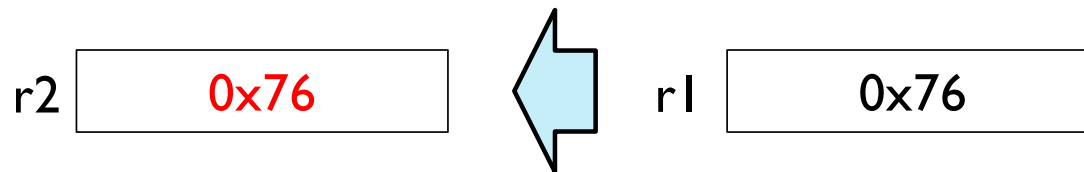


Data Transfer between Registers

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.

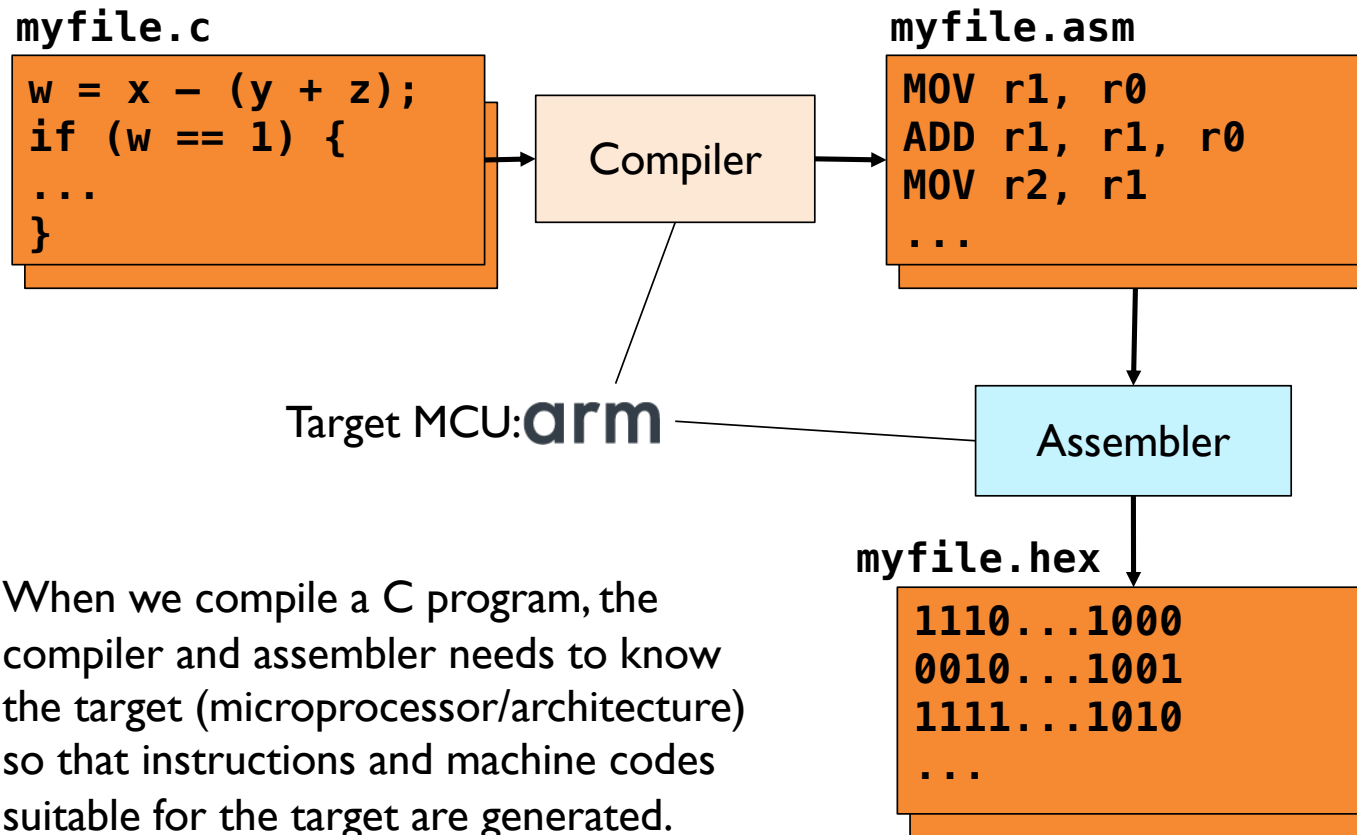
Assembly Language

To make the machine executes our **INSTRUCTIONS**, an assembler will translate our symbolic notation into **MACHINE CODES**:

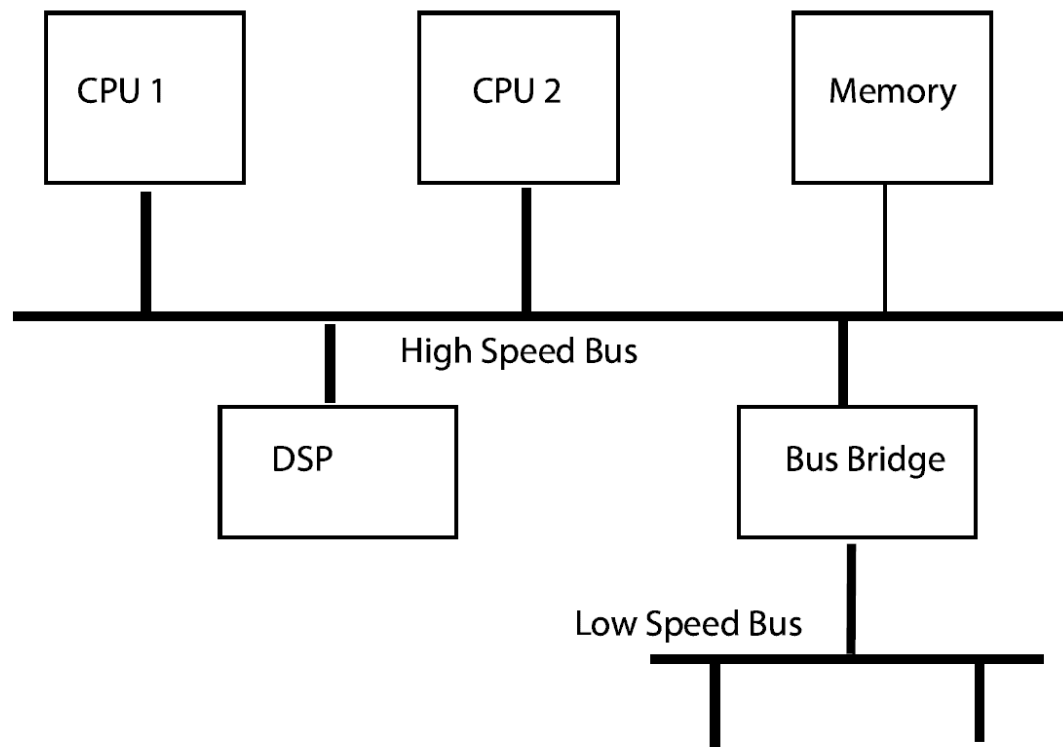
MOV r2, r1 → 1110 0001 1010 0000 0010 0000 0000 0001

- Programming binary machine codes is far **TOO TEDIOUS**.
- Use the Symbolic Language that called 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.

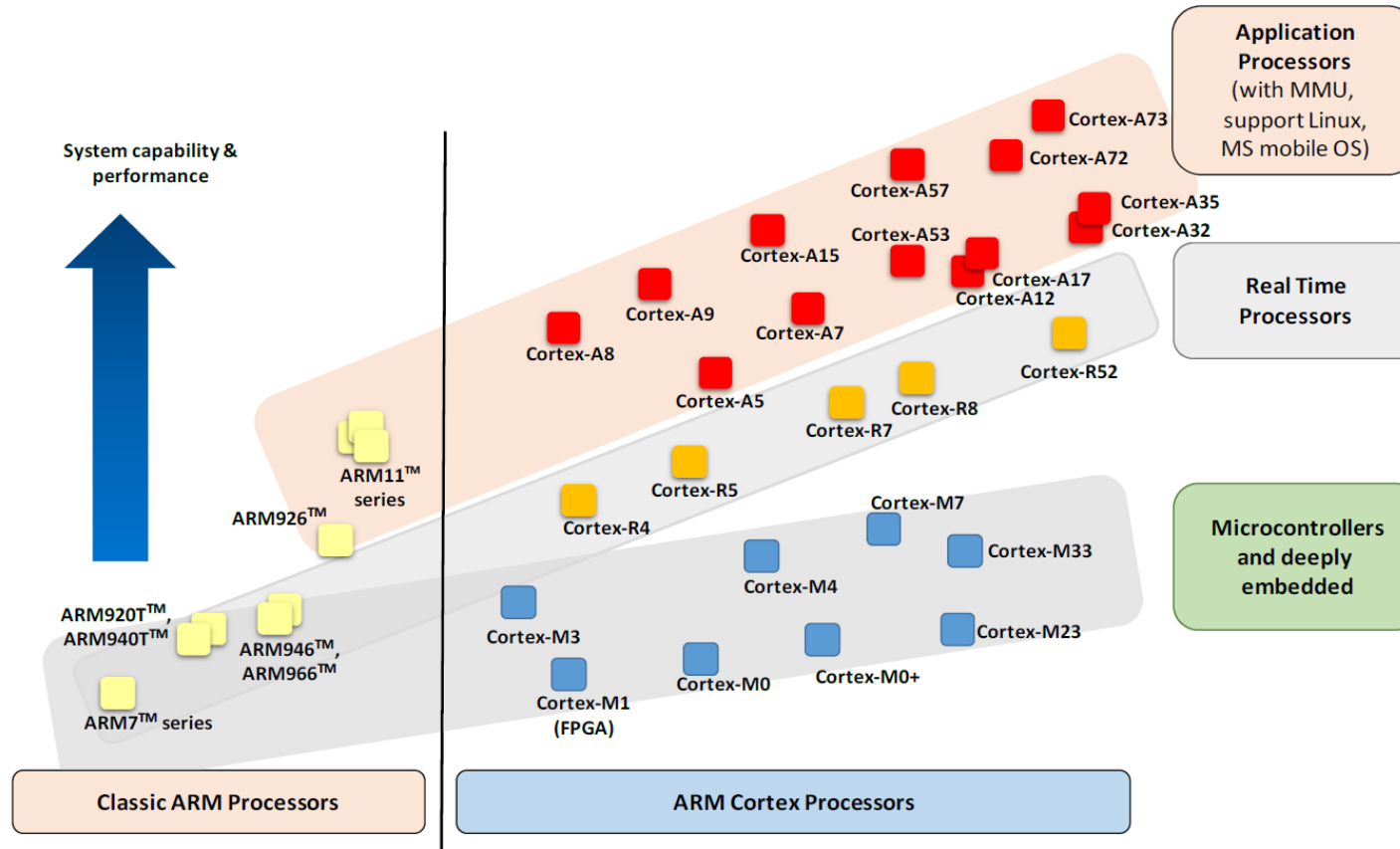
Compilation and Assembling



SoC: System level interconnection: Bus-based



ARM Processor Family

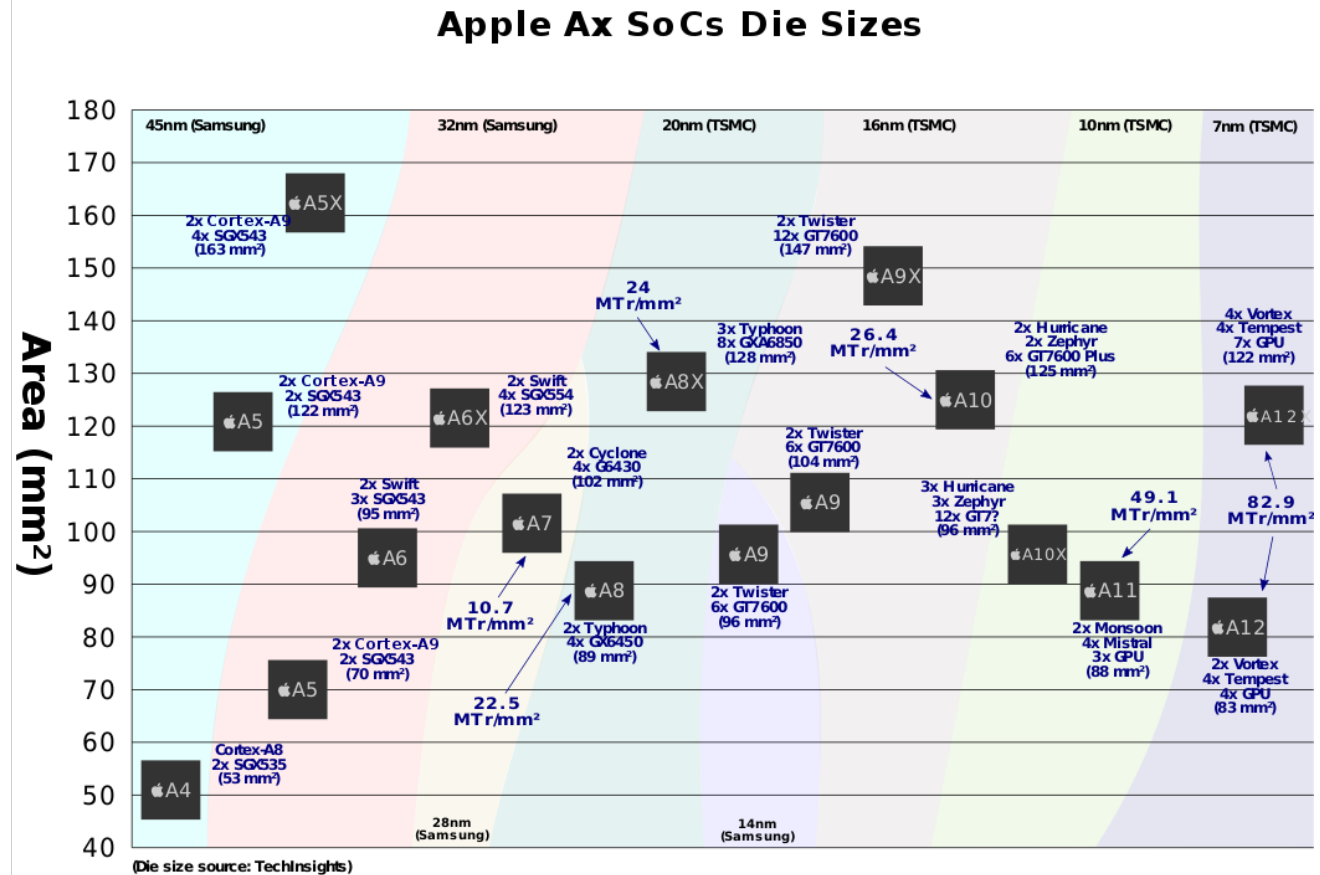


ARM Processor Family

- **ARM PROCESSORS** are divided between the classic ARM processors and 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

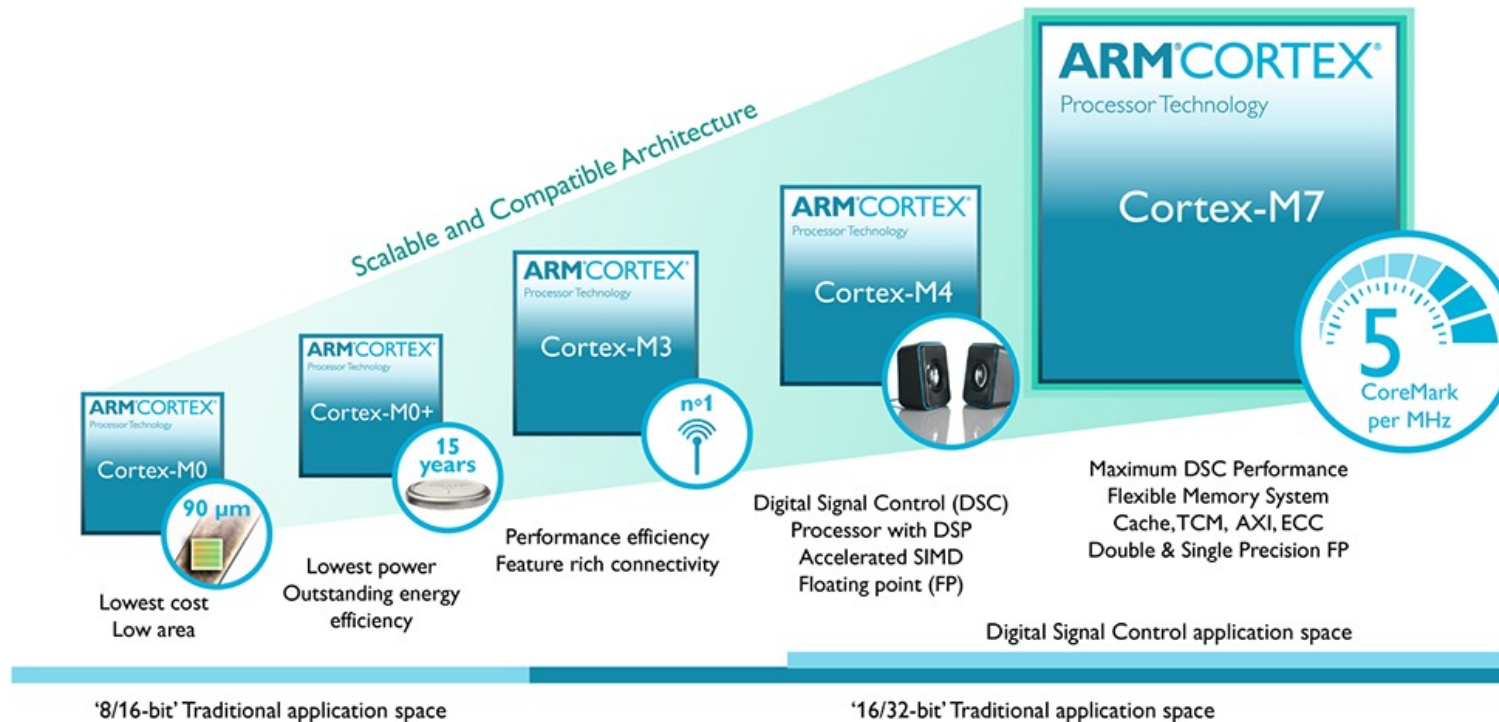
Apple Ax Processor Family

- Focus on the Area, and the processor technology.

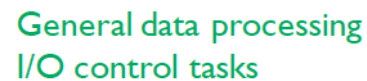


ARM Cortex-M Series

- How to measure performance and choose the right microprocessor?



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



Online CPU Simulator

<https://cpulator.01xz.net/>

CPULATOR Computer System Simulator

CPULATOR is a Nios II, ARMv7, and MIPS simulator of a computer system (processor and I/O devices) and debugger that runs in a modern web browser. It is designed as a tool for learning assembly-language programming and computer organization.

To start using CPULATOR now, choose a computer system to simulate, then follow the link.

To learn more, try a sample program in the simulator (Help → Sample programs), or see the [documentation](#).

Choose a system to simulate

Architecture

- Any
- Nios II
- ARMv7
- MIPS32r5
- MIPS32r5 (no delay slots)
- MIPS32r6
- MIPS32r6 (no delay slots)

System

- Nios II generic
- Nios II DE1-SoC
- Nios II DE1-SoC (v16.1)
- Nios II DE2-115
- Nios II DE2-115 (v16.1)
- Nios II DE2
- Nios II DE0

<https://cpulator.01xz.net/?sys=nios>

Go

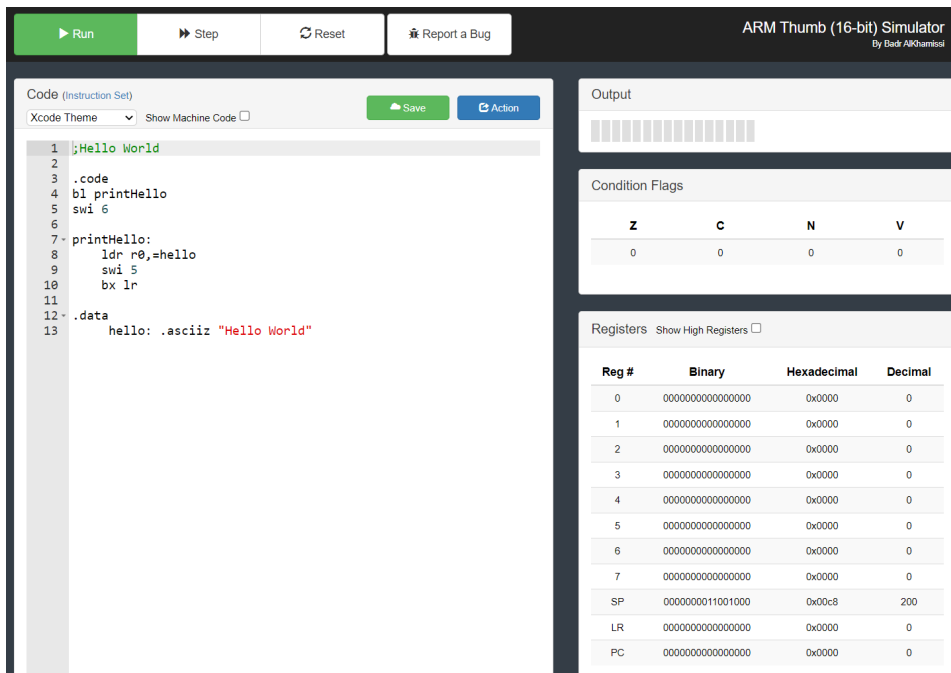
Nios II generic

Nios II system with 4 GB of memory and no other I/O devices

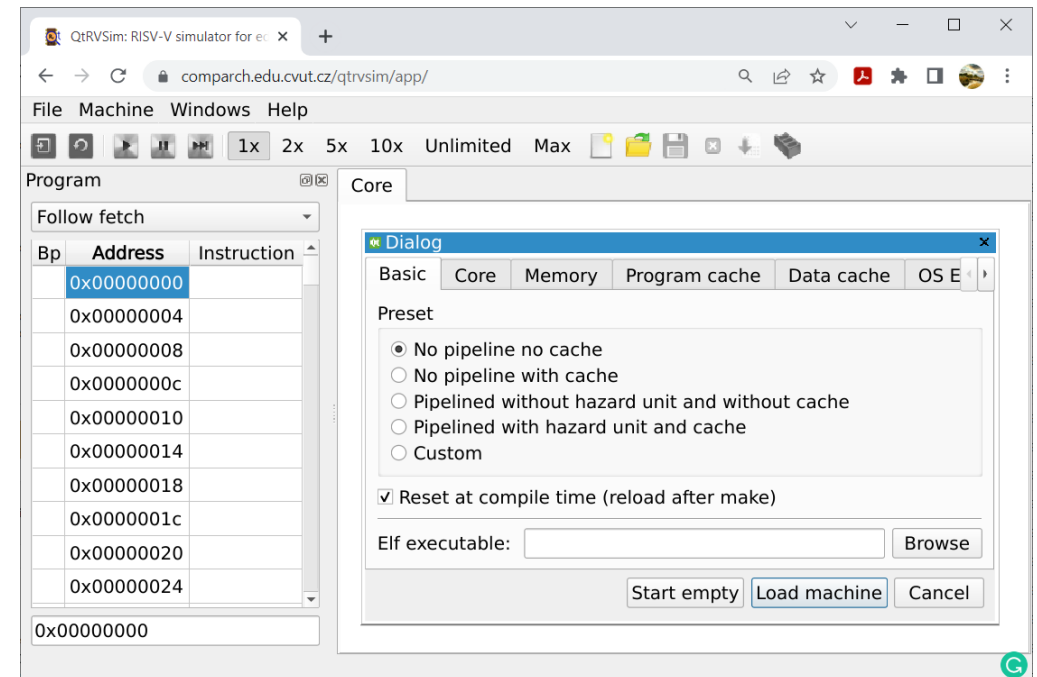
Optional I/O Devices

☐ Beam balancer

Other CPU Simulators: ARM vs. RISC-V



https://bkhmsi.github.io/ARMThumb_Sim/#/



<https://comparch.edu.cvut.cz/qtrvsim/app/>

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

