Computer models and ISA

Computer Architecture

UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CS3501 – 2024II
PROF.: JGONZALEZ@UTEC.EDU.PE
SRC: HARRIS, HARRIS - DIGITAL DESIGN AND COMPUTER ARCHITECTURE
STALLINGS – COMPUTER ARCHITECTURE

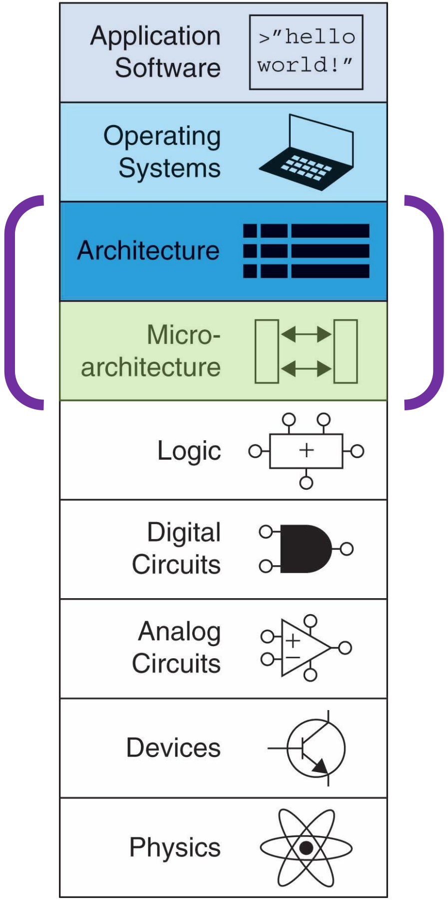
Executive Summary

- Motivation: Processors are present in everyday activities within computing systems.
- Problem: We need to understand the processor operation and design to propose future paradigms.
- Overview:
  - Computer definitions
  - Architecture models
  - ARM Instructions.
- Conclusion: We can start building a complex processor using design principles and fundamental building blocks.

Outline

Moving Towards the Abstraction Levels

- **We have covered the lower levels on the previous lectures.**
  - From bits to logic circuits (combinational and sequential).
- **New domain:**
  - **Architecture:** programmer's view of computer
    - Defined by **instructions** and **operand locations**
  - **Microarchitecture:** how to implement an architecture in hardware (more in later lectures).



Who is he?

Who is he?

- Why?

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

Stanford Encyclopedia of Philosophy

Browse About Support SEP

Search SEP

Entry Contents

Bibliography

Academic Tools

Friends PDF Preview

Author and Citation Info

Back to Top

Turing Machines

First published Mon Sep 24, 2018

Turing machines, first described by Alan Turing in Turing 1936–7, are simple abstract computational devices intended to help investigate the extent and limitations of what can be computed. Turing's 'automatic machines', as he termed them in 1936, were specifically devised for the computing of real numbers. They were first named 'Turing machines' by Alonzo Church in a review of Turing's paper (Church 1937). Today, they are considered to be one of the foundational models of computability and (theoretical) computer science.[1]

- 1. Definitions of the Turing Machine
  - 1.1 Turing's Definition
  - 1.2 Post's Definition
  - 1.3 The Definition Formalized

UTEC UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

So, given some Turing machine T which is in state q_i scanning the symbol S_j, its ID is given by Pq_iS_jQ where P and Q are the finite words to the left and right hand side of the square containing the symbol S_j. Figure 1 gives a visual representation of an ID of some Turing machine T in state q_i scanning the tape.

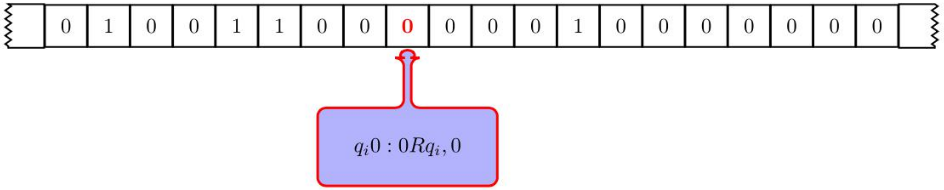


FIGURE 1: A complete configuration of some Turing machine T

The notation thus allows us to capture the developing behavior of the machine and its tape through its consecutive IDs. Figure 2 gives the first few consecutive IDs of T_{Simple} using a graphical representation.

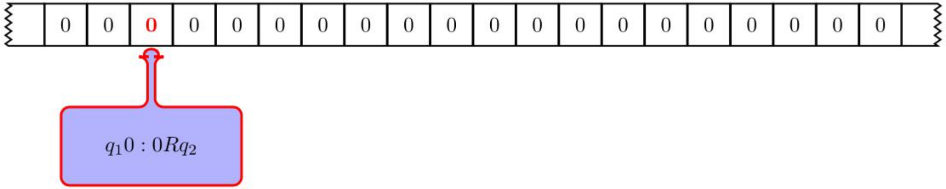
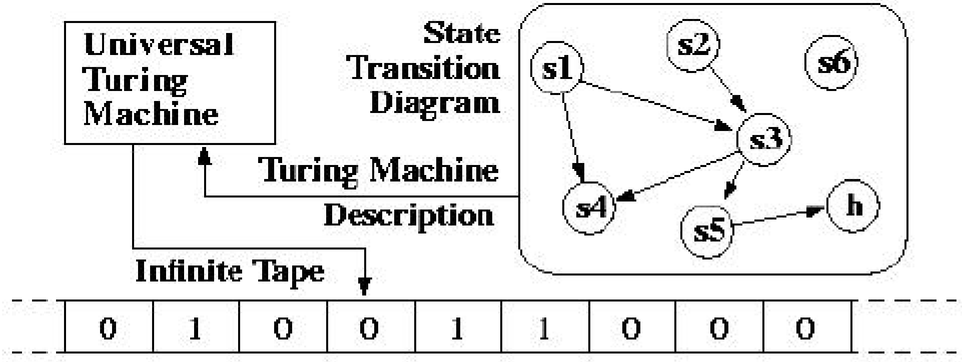


FIGURE 2: The dynamics of T_{Simple} graphical representation

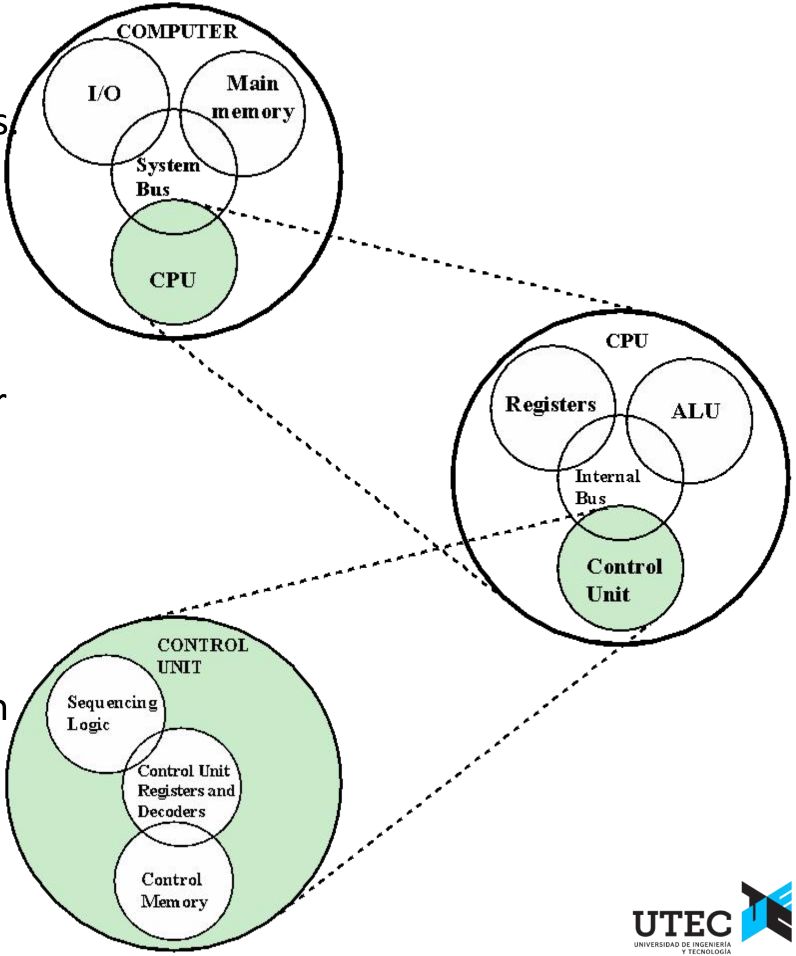
- State Machine : FSM
- Tape: Memory

https://web.mit.edu/manoli/turing/www/turing.html



Computer Functions and Components

1. **Data processing**
   - Multiple types of data and processing requirements.
2. **Data storage**
   - Short-term
   - Long-term
3. **Data movement**
   - **Input-output (I/O):** when data are received from or delivered to a device (peripheral) that is directly connected to the computer
   - **Data communications:** when data are moved over longer distances, to or from a remote device
4. **Control**
   - A control unit manages the computer's resources in response to commands (instructions).



UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

ARM Architecture

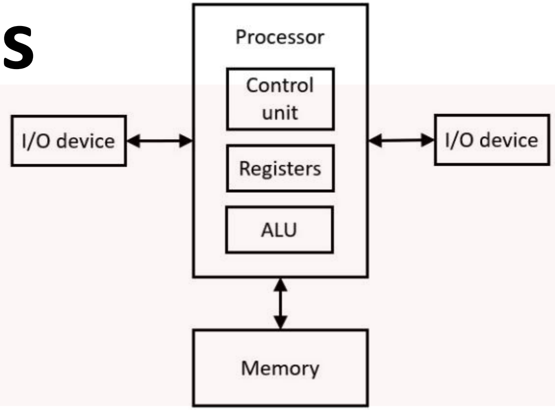
- **Developed in the 1980's by** Advanced RISC Machines – now called ARM Holdings
- **Nearly 10 billion ARM processors sold/year**
- **Almost all cell phones and tablets have multiple ARM processors**
- **Over 75% of humans use products with an ARM processor**
- Used in servers, cameras, robots, cars, pinball machines, etc.

Once you've learned one architecture, it's easier to learn a different one ☺

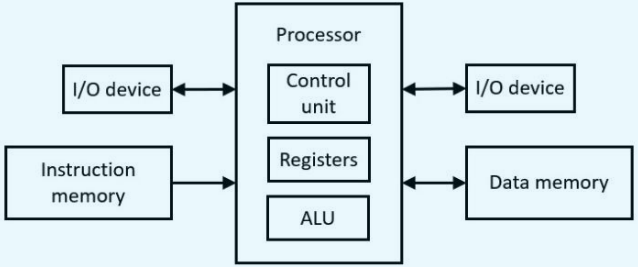
Outline

Computer Architecture Models

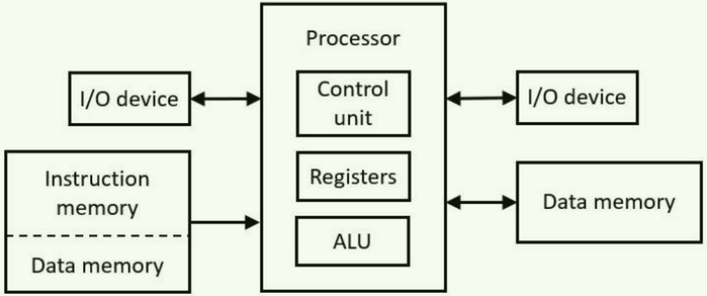
- **Von Neumann:**
  - Single area of memory for program instructions and the data.
  - Easier for circuit designers.



- **Harvard:**
  - Complete separation of code and data memory regions.
  - Rarely used in modern computers.



- **Modified Harvard:**
  - Some degree of separation between program instructions and data.
  - Modern processors.

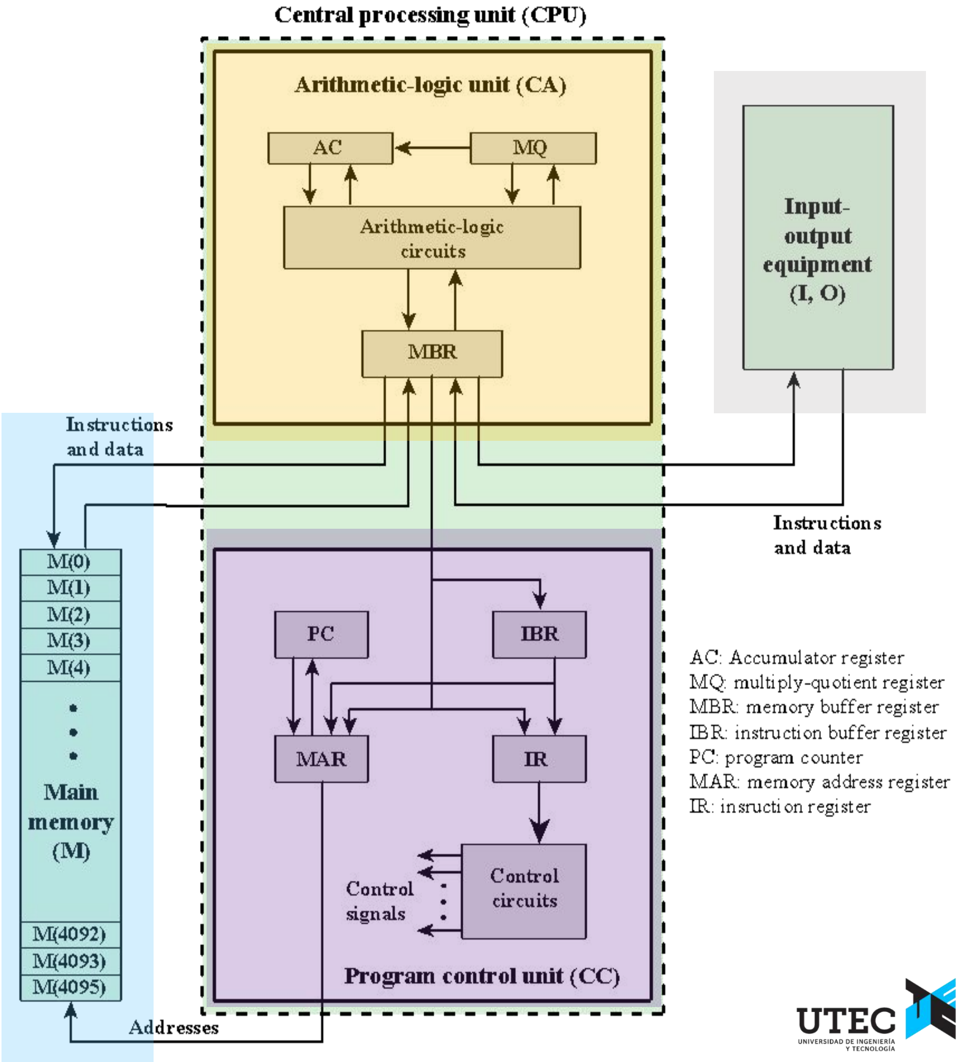


Von Neumann Model

- Proposed by [Burks, Goldstein and von Neumann, 1946] in **"Preliminary discussion of the logical design of an electronic computing instrument"**

- **5 parts:**
  1. Memory
  2. Processing Unit
  3. Input
  4. Output
  5. Control Unit

- **Bottleneck:** total time for access data can be higher than time for working in the actual program.



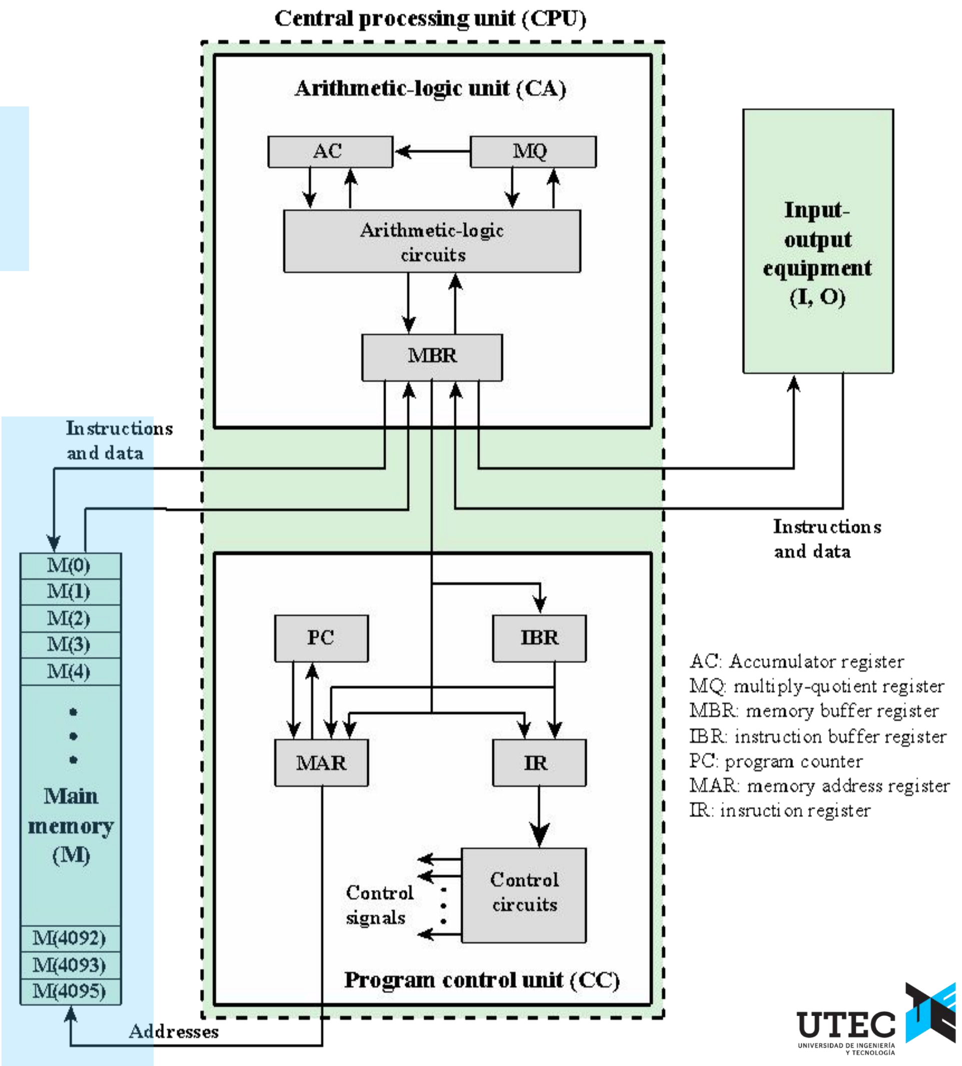
Memory stores:

1) **Data**
2) **Programs (instructions)**

- **Both are represented as bits**
- **Processor receives bits, control (FSM) determines if it is data or program.**

Memory

- **The memory stores**
  - Data
  - Programs
- The memory contains bits
  - Bits are grouped into bytes (8 bits) and words (e.g., 8, 16, 32 bits)
- How the bits are accessed determines the addressability
  - E.g., word-addressable
  - E.g., 8-bit addressable (or byte-addressable)
- The total number of addresses is the address space
  - In ARM, the address space is 2^{32}
    - 32-bit addresses
  - In x86-64, the address space is (up to) 2^{48}
    - 48-bit addresses



UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

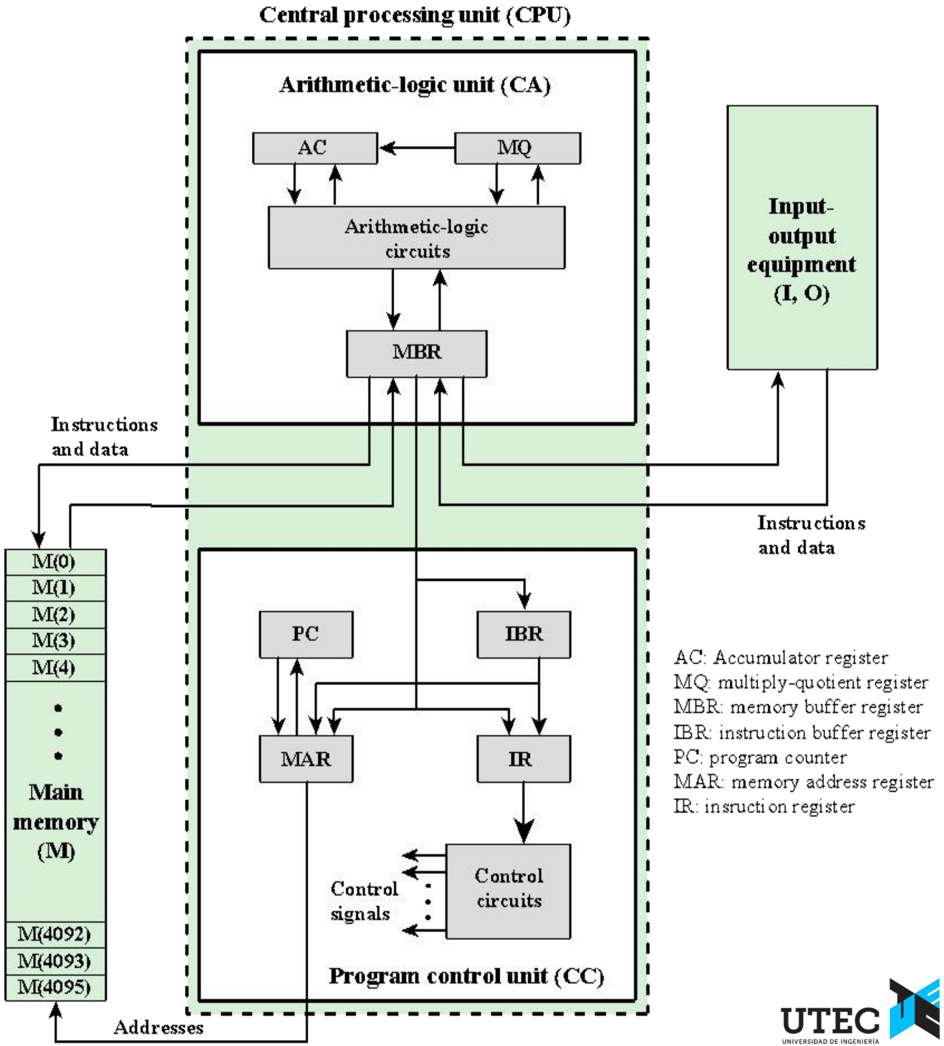
Accessing Memory

- There are two ways of accessing memory
  - Reading or loading
  - Writing or storing

- Two registers are necessary to access memory
  - Memory Address Register (MAR)
  - Memory Data Register (MDR)

- To read
  - Step 1: Load the MAR with the address
  - Step 2: Data is placed in MDR

- To write
  - Step 1: Load the MAR with the address and the MDR with the data
  - Step 2: Activate Write Enable signal



Word-Addressable Memory

- Each data word has a unique address
  - In ARM, a unique address for each 32-bit data word

ARM memory

Word Address	Data	
⋮	⋮	⋮
00000003	D 1 6 1 7 A 1 C	Word 3
00000002	1 3 C 8 1 7 5 5	Word 2
00000001	F 2 F 1 F 0 F 7	Word 1
00000000	8 9 A B C D E F	Word 0

Byte-Addressable Memory

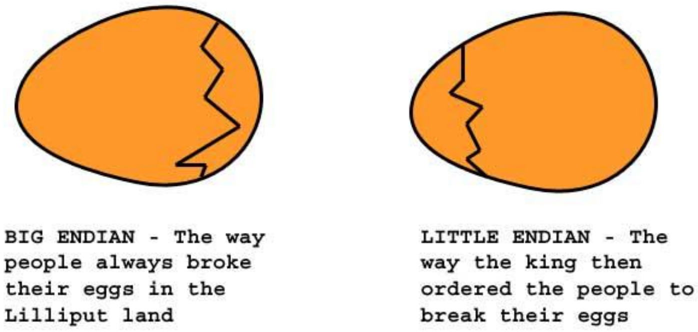
- Each byte has a unique address
  - Actually, ARM is byte-addressable

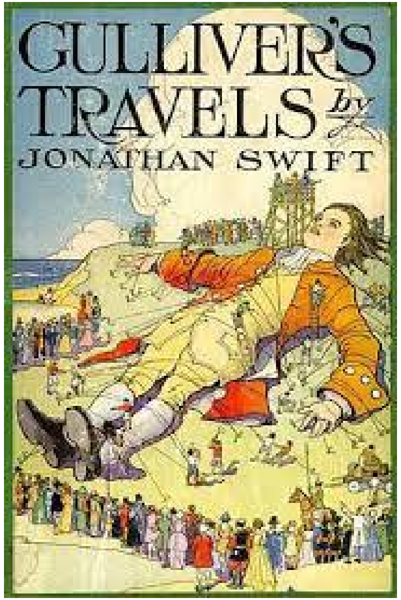
ARM memory

Byte Address of the Word	Data				
⋮	⋮				⋮
0000000C	D1	61	7A	1C	Word 3
00000008	13	C8	17	55	Word 2
00000004	F2	F1	F0	F7	Word 1
00000000	How are these four bytes addressed?				Word 0

Big Endian vs Little Endian

- Jonathan Swift's Gulliver's Travels
  - **Little Endians** broke their eggs on the little end of the egg
  - **Big Endians** broke their eggs on the big end of the egg





- How to number bytes within a word?
  - **Little-endian:** byte numbers start at the little (least significant) end
  - **Big-endian**: byte numbers start at the big (most significant) end

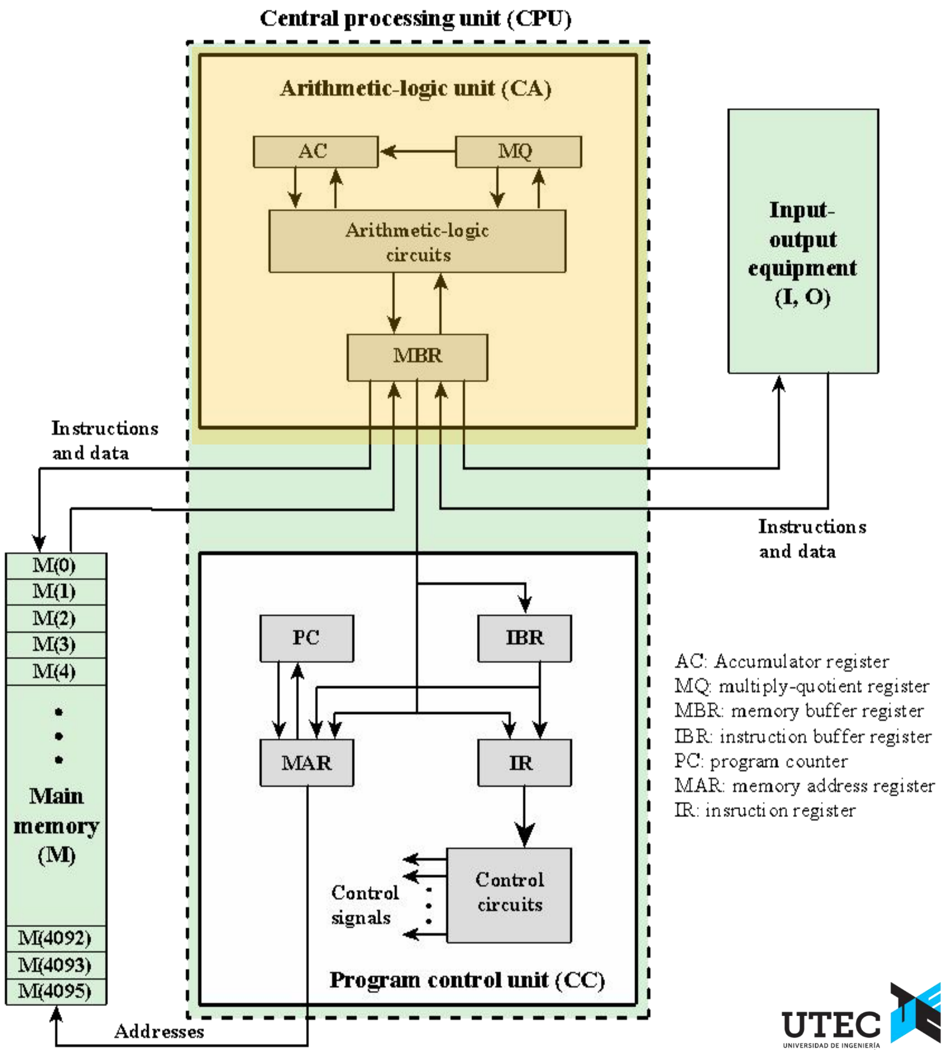
Big Endian vs Little Endian

Big Endian Byte Address				Word Address	Little Endian Byte Address			
C	D	E	F	C	F	E	D	C
8	9	A	B	8	B	A	9	8
4	5	6	7	4	7	6	5	4
0	1	2	3	0	3	2	1	0
MSB (Most Significant Byte)			LSB (Least Significant Byte)		MSB			LSB

It doesn't really matter which addressing type used – except when two systems share data

Processing Unit

- The processing unit can consist of many functional units
- We start with a simple **Arithmetic and Logic Unit (ALU)**, which executes computations
  - ARM: add, sub, mult, and, nor, …
- The ALU processes quantities that are referred to as words
  - Word length In ARM it is 32 bits
- Temporary storage: Registers
  - E.g., to calculate (A+B)*C, the intermediate result of A+B is stored in a register



Registers

- **Memory** is big but slow

- **Registers**
  - Registers **are faster than memory**
  - Ensure fast access to operands
  - Typically one register contains **one word**

- **Register set or file**
  - ARM has 16 registers
  - Each register is 32 bits
  - ARM is called a “32-bit architecture” because it operates on 32-bit data

Name	Use
R0	Argument / return value / temporary variable
R1-R3	Argument / temporary variables
R4-R11	Saved variables
R12	Temporary variable
R13 (SP)	Stack Pointer
R14 (LR)	Link Register
R15 (PC)	Program Counter

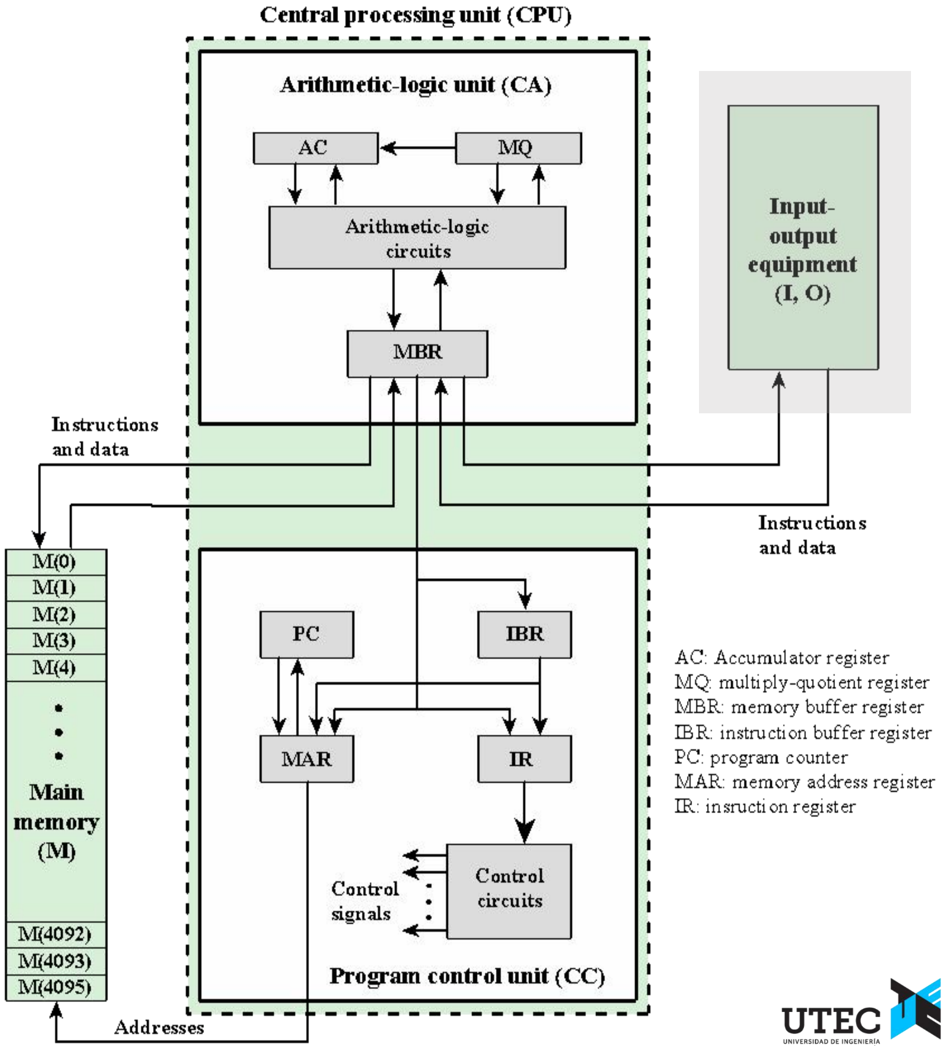
Register Set

- **Registers naming:**
  - R before number, all capitals
  - **E.g.:** “R0” or “register zero” or “register R0”
- **Used for specific purposes:**
  - **Saved registers:** R4-R11 hold variables
  - **Temporary registers:** R0-R3 and R12, hold intermediate values
  - Discuss others later

Name	Use
R0	Argument / return value / temporary variable
R1-R3	Argument / temporary variables
R4-R11	Saved variables
R12	Temporary variable
R13 (SP)	Stack Pointer
R14 (LR)	Link Register
R15 (PC)	Program Counter

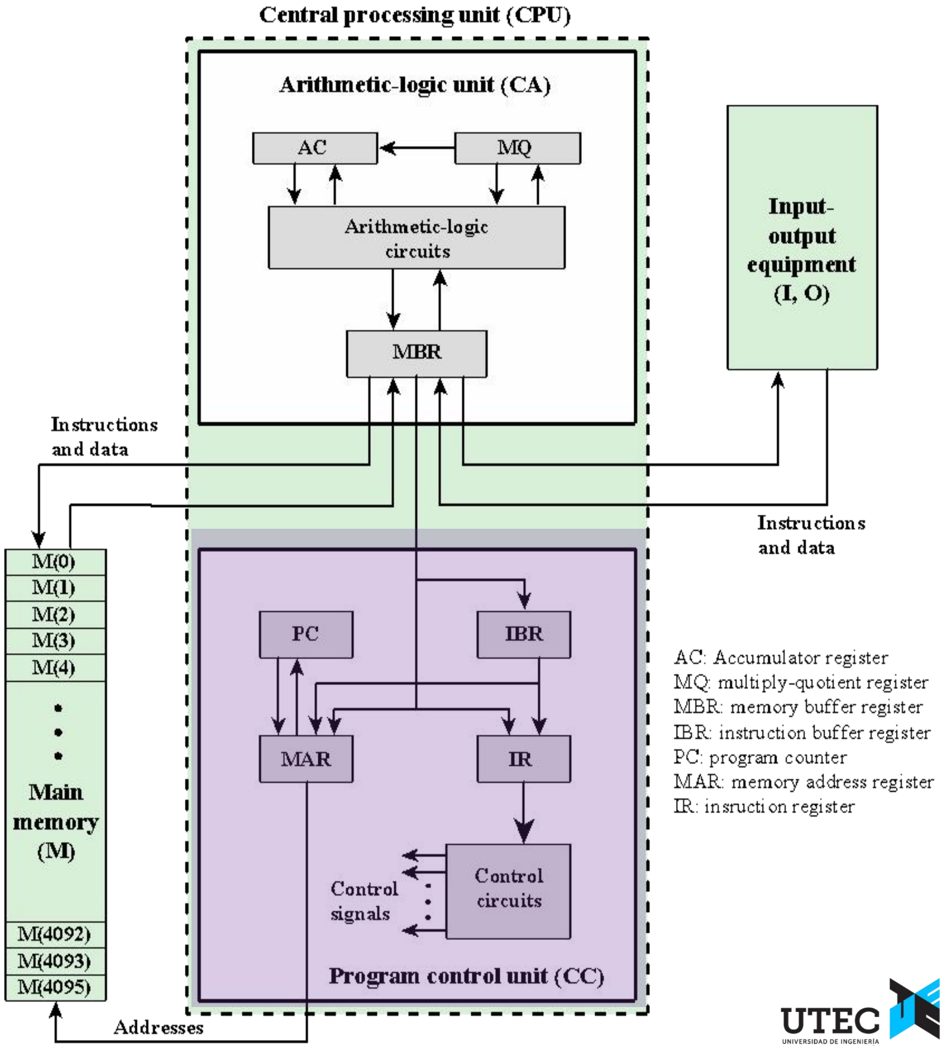
Input and Output

- Many devices can be used for input and output
- They are called peripherals
  - Input
    - Keyboard
    - Mouse
    - Scanner
    - Disks
    - Etc.
  - Output
    - Monitor
    - Printer
    - Disks
    - Etc.

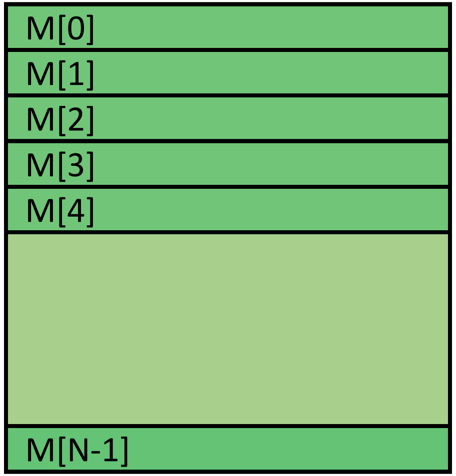


Control Unit

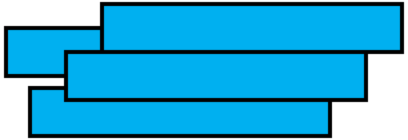
- The control unit is similar to the conductor of an orchestra
- It conducts the **step-by-step process of executing (every instruction in) a program**
- It keeps track of the instruction being executed with an **instruction register** (IR), which contains the instruction
- Another register contains the address of the next instruction to execute. It is called **program counter** (PC) or **instruction pointer** (IP)



Programmer Visible (Architectural) State



Memory
array of storage locations
indexed by an address



Registers
- given special names in the ISA (as opposed to addresses)
- general vs. special purpose

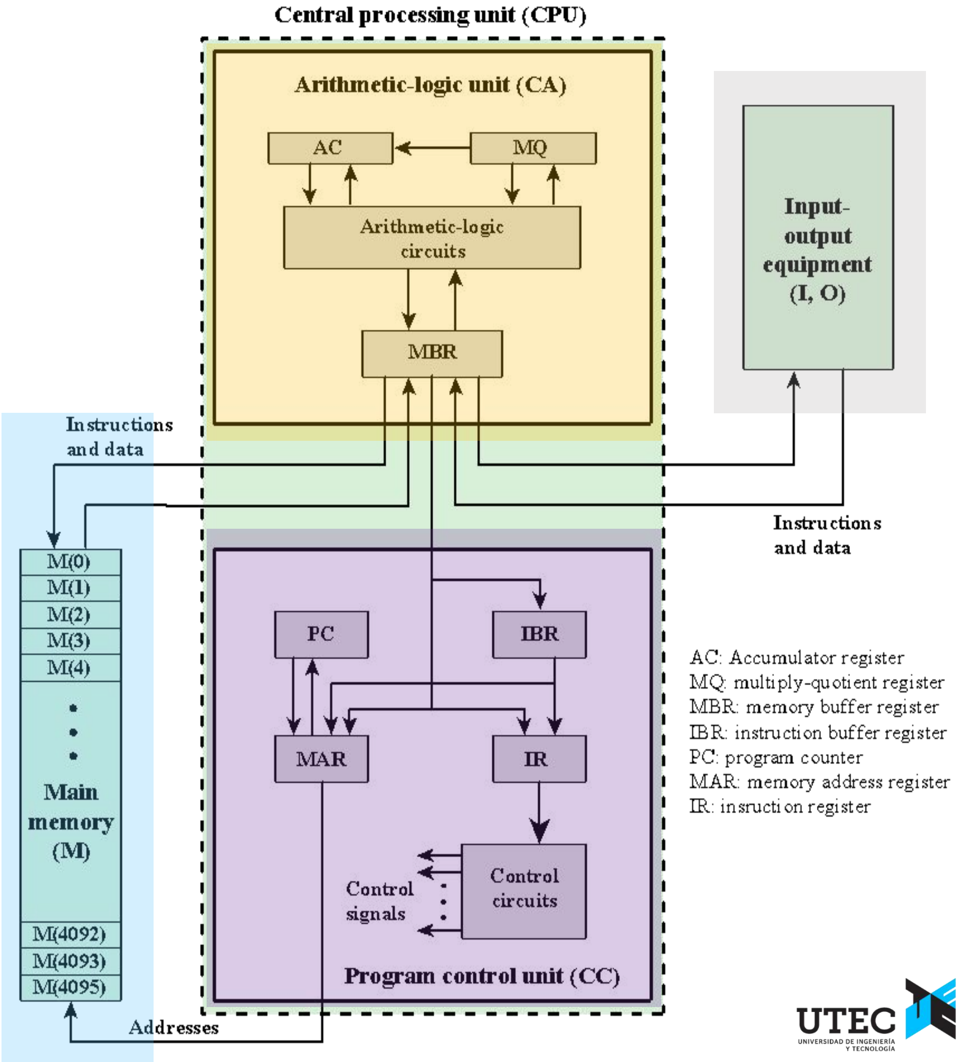
Program Counter
memory address
of the current instruction

Instructions (and programs) specify **how to transform the values of programmer visible state**

Von Neumann Model

- Also called stored program computer **(instructions in memory)**. It has two key properties:

1. Stored program
    - Instructions stored in a linear memory array
    - Memory is unified between instructions and data
        - The interpretation of a stored value depends on the control signals
2. Sequential instruction processing
    - One instruction processed (fetched, executed, completed) at a time
    - Program counter (instruction pointer) identifies the current instruction
    - Program counter is advanced sequentially except for control transfer instructions



Outline

Instructions

- An instruction the **most basic unit of computer processing**
  - Instructions are words in the language of a computer
  - **Instruction Set Architecture (ISA)** is the vocabulary

- ARM is a **Reduced Instruction Set Computer (RISC)**, with a small number of simple instructions
- Other architectures, such as Intel's x86, are **Complex Instruction Set Computers (CISC)**

Sequential Execution

- Instructions and data are stored in memory
  - Typically the instruction length is the word length
- **Instructions: Commands in a computer's language**
  - **Assembly language:** human-readable format of instructions
  - **Machine language:** computer-readable format (1's and 0's)
- The processor fetches instructions from memory sequentially (first one instruction, then the next one ...)
- The address of the current instruction is stored in the program counter (PC)
  - If word-addressable memory, the processor increments the PC by 1.
  - If byte-addressable memory, the processor increments the PC by the word length (4 in ARM)
    - E.g: OS sets the PC to 0x00400000 (start of a program), increment from this address value.

A Sample Program Stored in Memory

- A sample program
  - 4 instructions stored in consecutive words in memory
    - No need to understand the program now. We will get back to it

Program assembly

```
lw   $t2, 32($0)
add  $s0, $s1, $s2
addi $t0, $s3, -12
sub  $t0, $t3, $t5
```

Machine code

```
0x8C0A0020
0x02328020
0x2268FFF4
0x016D4022
```

Address	Instructions	
⋮	⋮	
0040000C	0 1 6 D 4 0 2 2	
00400008	2 2 6 8 F F F 4	
00400004	0 2 3 2 8 0 2 0	
00400000	8 C 0 A 0 0 2 0	← PC
⋮	⋮	

Example 1: ADD Instruction

C Code

```
a = b + c;
```

ARM Assembly Code

```
ADD a, b, c
```

- **ADD:** mnemonic – indicates operation to perform
- **b, c:** source operands
- **a:** destination operand

Example 2: SUB Instruction

Similar to addition - only mnemonic changes

C Code	ARM assembly code
`a = b - c;`	`SUB a, b, c`

- **SUB:** mnemonic
- **b, c:** source operands
- **a:** destination operand

Example 3: Multiple Instructions

More complex code handled by multiple ARM instructions

C Code

```
a = b + c - d;
```

ARM assembly code

```
ADD t, b, c  ; t = b + c
SUB a, t, d  ; a = t - d
```

Instructions with Registers

Revisit `ADD` instruction

C Code

```
a = b + c
```

ARM Assembly Code

```
; R0 = a, R1 = b, R2 = c

ADD R0, R1, R2
```

Operands: Constants and Immediates

- Many instructions can use constants or *immediate* operands
- For example: ADD and SUB
- value is *immediate*ly available from instruction

C Code

```
a = a + 4;
b = a - 12;
```

ARM Assembly Code

```
; R0 = a, R1 = b
ADD R0, R0, #4
SUB R1, R0, #12
```

Generating Constants

Generating small constants using move (`MOV`):

C Code

```
//int: 32-bit signed word
int a = 23;
int b = 0x45;
```

ARM Assembly Code

```
; R0 = a, R1 = b
MOV R0, #23
MOV R1, #0x45
```

Constant must have < 8 bits of precision

Note: `MOV` can also use 2 registers: `MOV R7, R9`

Generating Constants

Generate larger constants using move (MOV) and or (ORR):

C Code

```
int a = 0x7EDC8765;
```

ARM Assembly Code

```
# R0 = a
MOV R0, #0x7E000000
ORR R0, R0, #0xDC0000
ORR R0, R0, #0x8700
ORR R0, R0, #0x65
```

Ada Lovelace

- British mathematician , **1815-1852**
- Wrote **the first computer program**
- **Her program calculated the Bernoulli numbers on Charles Babbage's Analytical Engine**
- She was a child of the poet Lord Byron

At her time, no high-level languages:

- e.g., C, Java, Python
- Written at higher level of abstraction

Types of Instructions and Programming Blocks

Three main types of instructions:

1. **Data-processing Instructions**
2. **Branches**
3. **Memory**

- **High-level Constructs:**
    - if/else statements
    - for loops
    - while loops
    - arrays
    - function calls

We study how to implement high-level programs using low-level definitions

How are These Instructions Executed?

- By using instructions we can speak the language of the computer
- Thus, we now know how to tell the computer to
  - Execute computations in the ALU by using, for instance, an addition
  - Access operands from memory by using the load word instruction
- But, how are these instructions executed on the computer?
  - The process of executing an instruction is called is the instruction cycle

More about instruction execution on next lecture.

Outline

Introduction

Computer Architecture

Instructions

Conclusion

Conclusions

- **We introduced computer architecture models and detailed the relation between them.**
- **We detailed the Von Neuman model and its operation.**
- **We introduced the ARM instruction set architecture.**
- We conclude that a **processor operates through a defined set of instructions.**

Instruction Set Architecture

Computer Architecture

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CS3501 - 2024II

PROF.: JGONZALEZ@UTEC.EDU.PE