Microcontroller architecture

ISA and Microarchitecture

Daniel Giovanni Martínez Sandoval

UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

I7266 - Programación de Sistemas Embebidos - 2024B

Table of Contents

Table of Contents

Definition of architecture

“Architecture” is a broad term in computer science, but we will stick to this definition:

Definition of architecture

“Architecture” is a broad term in computer science, but we will stick to this definition:

Architecture

Architecture describes the internal organization of a computer in an abstract way; that is, it defines the capabilities of the computer and its programming model.[1]

Definition of architecture

“Architecture” is a broad term in computer science, but we will stick to this definition:

Architecture

Architecture describes the internal organization of a computer in an abstract way; that is, it defines the capabilities of the computer and its programming model.[1]

Basically, it is an abstraction that outlines how a computer’s hardware components interact and work together to process data. Of course it is important. But why?

Importance of architecture

In embedded systems, architecture plays a crucial role because these systems are mostly designed for specific tasks rather than general-purpose computing. Some things to take into account when choosing an architecture are:

Importance of architecture

In embedded systems, architecture plays a crucial role because these systems are mostly designed for specific tasks rather than general-purpose computing. Some things to take into account when choosing an architecture are:

- **Power consumption:** Many embedded systems operate on battery power. A microcontroller packed with features but with a high current consumption is not always feasible for battery-operated systems.

Importance of architecture

In embedded systems, architecture plays a crucial role because these systems are mostly designed for specific tasks rather than general-purpose computing. Some things to take into account when choosing an architecture are:

- **Power consumption:** Many embedded systems operate on battery power. A microcontroller packed with features but with a high current consumption is not always feasible for battery-operated systems.
- **Cost and size:** Choosing a microcontroller with just enough features to develop the intended functionality is key to saving money, but also in some cases the size, which in turn also reduces the cost.

Importance of architecture

In embedded systems, architecture plays a crucial role because these systems are mostly designed for specific tasks rather than general-purpose computing. Some things to take into account when choosing an architecture are:

- **Power consumption:** Many embedded systems operate on battery power. A microcontroller packed with features but with a high current consumption is not always feasible for battery-operated systems.
- **Cost and size:** Choosing a microcontroller with just enough features to develop the intended functionality is key to saving money, but also in some cases the size, which in turn also reduces the cost.
- **Performance and memory:** Depending on the application, some devices might only run certain OS which is targeted for a specific arquitechture, or maybe they need to be compatible with external memory.

Importance of architecture (cont.)

So, just like buying any other thing, when buying a microcontroller to design an embedded system, you should take into account the size, your budget, the style, among many other things. But the thing with architecture in computer science and embedded systems is that saying “architecture” is really, really broad.

Importance of architecture (cont.)

So, just like buying any other thing, when buying a microcontroller to design an embedded system, you should take into account the size, your budget, the style, among many other things. But the thing with architecture in computer science and embedded systems is that saying “architecture” is really, really broad.

Why? you may ask... Well, in the beginning we had very simple circuits compared to those nowadays. We also had one way to program them: 1’s and ’0s. But today we have a huge variety of variables, even with circuits themselves, since we’re not limited to Microcontrollers(µCs or MCUs) and Microprocessors(µPs), we also have Digital Signal Processors (DSPs), Field Programmable Gate Arrays (FPGAs), System-on-Chips (SoCs), and Application Specific Integrated Circuits (ASICS).

Different kinds of architectures

I said that architecture is a broad term, and is not only because of the current complexity of computational systems, but also because of the many variants of architecture or the many ways that we can organize components inside a microcontroller in order to understand its functionality.

Different kinds of architectures

I said that architecture is a broad term, and is not only because of the current complexity of computational systems, but also because of the many variants of architecture or the many ways that we can organize components inside a microcontroller in order to understand its functionality.
So, let us explore some different kinds of architectures.

ISAs

Instruction Set Architecture

An Instruction Set Architecture (ISA) is part of the abstract model of a computer that defines how the CPU is controlled by the software. The ISA acts as an interface between the hardware and the software, specifying both what the processor is capable of doing as well as how it gets done. [2]

ISAs

Instruction Set Architecture

An Instruction Set Architecture (ISA) is part of the abstract model of a computer that defines how the CPU is controlled by the software. The ISA acts as an interface between the hardware and the software, specifying both what the processor is capable of doing as well as how it gets done. [2]

The ISA provides the only way through which a user is able to interact with the hardware. It can be viewed as a programmer's manual because it is the portion of the machine that's visible to the assembly language programmer, the compiler writer, and the application programmer.

ISAs (cont.)

Some popular ISAs are:

ISAs (cont.)

Some popular ISAs are:

- **ARM:** Used in smartphones. Based in RISC.

ISAs (cont.)

Some popular ISAs are:

- **ARM:** Used in smartphones. Based in RISC.
- **x86:** Used in desktop and laptop computers. Is a CISC.

ISAs (cont.)

Some popular ISAs are:

- **ARM:** Used in smartphones. Based in RISC.
- **x86:** Used in desktop and laptop computers. Is a CISC.
- **RISC-V:** An *open-source* ISA.

ISAs (cont.)

Some popular ISAs are:

- **ARM:** Used in smartphones. Based in RISC.
- **x86:** Used in desktop and laptop computers. Is a CISC.
- **RISC-V:** An *open-source* ISA.
- **MIPS:** Based in RISC and used in many network systems.

ISAs (cont.)

Some popular ISAs are:

- **ARM:** Used in smartphones. Based in RISC.
- **x86:** Used in desktop and laptop computers. Is a CISC.
- **RISC-V:** An *open-source* ISA.
- **MIPS:** Based in RISC and used in many network systems.
- **AVR:** 8-bit RISC, which we will explore and use in this course.

Microarchitecture

Microarchitecture

Microarchitecture is the implementation of the ISA in hardware. It defines how a processor's internal components are organized and interact to execute the instructions described by the ISA.

ISAs can be executed in different implementations of hardware, so, a microarchitecture (or µarch) is the description of that exact thing.

Microarchitecture (cont.)

Microarchitectures are usually represented as diagrams that describe the interconnections of the various microarchitectural elements of the machine, which may include gates, registers, arithmetic ALUs and even larger elements. These diagrams generally also include the buses.

Memory hierarchy

Memory hierarchy architecture refers to the structured levels of memory in a computer system, from the fastest (but smallest) to the slowest (but largest) storage. The hierarchy helps balance speed, cost, and capacity. In a typical modern microcontroller we can usually find:

Memory hierarchy

Memory hierarchy architecture refers to the structured levels of memory in a computer system, from the fastest (but smallest) to the slowest (but largest) storage. The hierarchy helps balance speed, cost, and capacity. In a typical modern microcontroller we can usually find:

- Registers

Memory hierarchy

Memory hierarchy architecture refers to the structured levels of memory in a computer system, from the fastest (but smallest) to the slowest (but largest) storage. The hierarchy helps balance speed, cost, and capacity. In a typical modern microcontroller we can usually find:

- Registers
- Cache

Memory hierarchy

Memory hierarchy architecture refers to the structured levels of memory in a computer system, from the fastest (but smallest) to the slowest (but largest) storage. The hierarchy helps balance speed, cost, and capacity. In a typical modern microcontroller we can usually find:

- Registers
- Cache
- RAM

Memory hierarchy

Memory hierarchy architecture refers to the structured levels of memory in a computer system, from the fastest (but smallest) to the slowest (but largest) storage. The hierarchy helps balance speed, cost, and capacity. In a typical modern microcontroller we can usually find:

- Registers
- Cache
- RAM
- Flash

Memory hierarchy

Memory hierarchy architecture refers to the structured levels of memory in a computer system, from the fastest (but smallest) to the slowest (but largest) storage. The hierarchy helps balance speed, cost, and capacity. In a typical modern microcontroller we can usually find:

- Registers
- Cache
- RAM
- Flash
- EEPROM

Memory hierarchy (cont.)

- **Registers**: The fastest type of memory located within the CPU, used to store temporary data and intermediate values during computation.

Memory hierarchy (cont.)

- **Registers**: The fastest type of memory located within the CPU, used to store temporary data and intermediate values during computation.
- **Cache**: These are small storage regions used for frequently accessed information. Can be located inside or really close to the CPU and it is really fast. Generally, we find three levels of cache:

Memory hierarchy (cont.)

- **Registers**: The fastest type of memory located within the CPU, used to store temporary data and intermediate values during computation.
- **Cache**: These are small storage regions used for frequently accessed information. Can be located inside or really close to the CPU and it is really fast. Generally, we find three levels of cache:
  - **L1**: The fastest, and also the smallest. Usually found inside the CPU. Stores recently used instructions or data that are most likely to be requested again soon.
  - **L2 & L3**: Typically external. Store additional information that may be needed by the processor in order to run efficiently; they are slightly slower than Level 1 but still provide quick access when necessary.

Memory hierarchy (cont.)

- **Flash**: A type of non-volatile memory in which data persists even after the power is turned off, it is cheap, small and reliable.

Memory hierarchy (cont.)

- **Flash**: A type of non-volatile memory in which data persists even after the power is turned off, it is cheap, small and reliable.
- **EEPROM**: Electrically Erasable Programmable Read-Only Memory. Offers faster byte-level writes and erases, making it suitable for applications with occasional updates (e.g., configuration settings). Still, flash provides higher storage density, better endurance through wear-leveling, and is more cost-effective for larger storage capacities and applications that require frequent data updates.

Real architectures

We have studied some basics about architectures, so...

Real architectures

We have studied some basics about architectures, so...
How do architectures really look?

Von-Neumann Architecture

- Single memory space: Program instructions and data share the same memory.

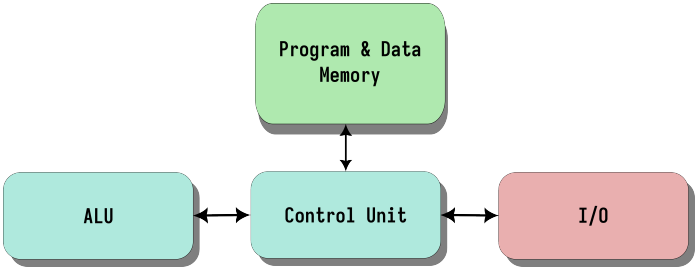
Von-Neumann Architecture

- Single memory space: Program instructions and data share the same memory.
- Unified bus: A single bus is used for transferring both instructions and data between memory and the CPU.

Von-Neumann Architecture

- Single memory space: Program instructions and data share the same memory.
- Unified bus: A single bus is used for transferring both instructions and data between memory and the CPU.
- Execution bottleneck: Since instructions and data share the same bus, the system can experience delays (known as the Von Neumann bottleneck) when fetching instructions and data alternately.

Von-Neumann Architecture



Harvard Architecture

- Separated memory spaces: instruction memory (Flash) and data memory (SRAM).

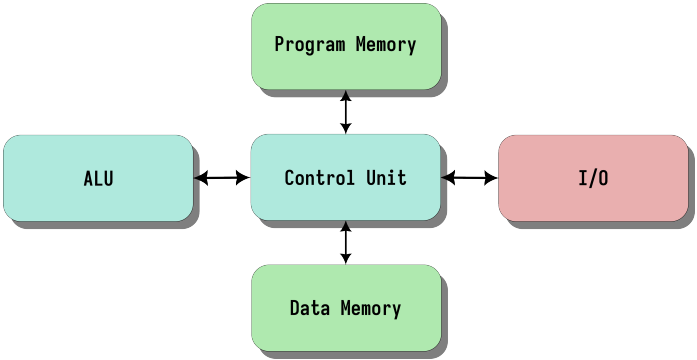
Harvard Architecture

- Separated memory spaces: instruction memory (Flash) and data memory (SRAM).
- Separate buses: The CPU can fetch instructions and access data simultaneously because it uses two separate buses for instruction and data transfer.

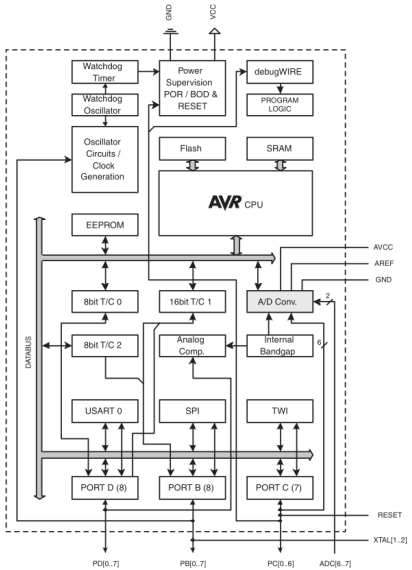
Harvard Architecture

- Separated memory spaces: instruction memory (Flash) and data memory (SRAM).
- Separate buses: The CPU can fetch instructions and access data simultaneously because it uses two separate buses for instruction and data transfer.
- Faster execution: This separation reduces the bottleneck since fetching instructions does not interfere with data access.

Harvard Architecture



ATmega328P architecture



The instruction cycle

The instruction cycle is the process by which a microprocessor (or microcontroller) achieves the execution of each instruction. It is the fundamental operation that allow these devices to perform tasks. It constists of the following steps:

Each step consists of multiple other steps. But to understand these steps we should also understand what instructions and microinstructions are.

The instruction cycle

The instruction cycle is the process by which a microprocessor (or microcontroller) achieves the execution of each instruction. It is the fundamental operation that allow these devices to perform tasks. It constists of the following steps:

1. Fetch: Retrieving the instruction from memory.

Each step consists of multiple other steps. But to understand these steps we should also understand what instructions and microinstructions are.

The instruction cycle

The instruction cycle is the process by which a microprocessor (or microcontroller) achieves the execution of each instruction. It is the fundamental operation that allow these devices to perform tasks. It constists of the following steps:

1. Fetch: Retrieving the instruction from memory.
2. Decode: Interpreting what the instruction is meant to do.

Each step consists of multiple other steps. But to understand these steps we should also understand what instructions and microinstructions are.

The instruction cycle

The instruction cycle is the process by which a microprocessor (or microcontroller) achieves the execution of each instruction. It is the fundamental operation that allow these devices to perform tasks. It constists of the following steps:

1. Fetch: Retrieving the instruction from memory.
2. Decode: Interpreting what the instruction is meant to do.
3. Execute: Performing the operation specified by the instruction.

Each step consists of multiple other steps. But to understand these steps we should also understand what instructions and microinstructions are.

Instructions and microinstructions

A CPU is capable of executing the instructions that the ISA describes and the µarch provides hardware for. There might be instructions, for example, a multiplication instruction, which may involve multiple steps, like to move the data to multiply to a register close to the ALU, and then operate and then move the result to another register. This is: a single ISA instruction might be comprised of multiple microinstructions, however, microinstructions are internal to CPU and they are out of our reach, but they still play a big role in microcontroller architectures.

The instruction cycle (cont.)

Fetch:

The instruction cycle (cont.)

Fetch:

1. The *progam counter* (PC) is previously loaded with the address of the next instruction to be fetched.

The instruction cycle (cont.)

Fetch:

1. The *progam counter* (PC) is previously loaded with the address of the next instruction to be fetched.
2. The address is copied to the *memory address register* (MAR)

The instruction cycle (cont.)

Fetch:

1. The *progam counter* (PC) is previously loaded with the address of the next instruction to be fetched.
2. The address is copied to the *memory address register* (MAR)
3. The data from the MAR is sent across the address bus, and the contents of the address are accessed.

The instruction cycle (cont.)

Fetch:

1. The *progam counter* (PC) is previously loaded with the address of the next instruction to be fetched.
2. The address is copied to the *memory address register* (MAR)
3. The data from the MAR is sent across the address bus, and the contents of the address are accessed.
4. The data from that location in memory is sent down the data bus to the *memory data register* (MDR).

The instruction cycle (cont.)

Fetch:

1. The *progam counter* (PC) is previously loaded with the address of the next instruction to be fetched.
2. The address is copied to the *memory address register* (MAR)
3. The data from the MAR is sent across the address bus, and the contents of the address are accessed.
4. The data from that location in memory is sent down the data bus to the *memory data register* (MDR).
5. There is now an instruction to decode and the PC increments.

The instruction cycle (cont.)

Decode:

The instruction cycle (cont.)

Decode:

1. The instruction is placed into the *current instruction register* (CIR).

The instruction cycle (cont.)

Decode:

1. The instruction is placed into the *current instruction register* (CIR).
2. The data is split into the *opcode* and the *operand*.

The instruction cycle (cont.)

Decode:

1. The instruction is placed into the *current instruction register* (CIR).
2. The data is split into the *opcode* and the *operand*.
3. Those are sent to the *control unit* (CU).

The instruction cycle (cont.)

Execute:

The instruction cycle (cont.)

Execute:

1. This part is really dependant on the instruction we are executing.

The instruction cycle (cont.)

Execute:

1. This part is really dependant on the instruction we are executing.
2. Maybe more data is read and then, processed.

The instruction cycle (cont.)

Execute:

1. This part is really dependant on the instruction we are executing.
2. Maybe more data is read and then, processed.
3. Then the ALU would do something with it.

The instruction cycle (cont.)

Execute:

1. This part is really dependant on the instruction we are executing.
2. Maybe more data is read and then, processed.
3. Then the ALU would do something with it.
4. The result should be stored into a register.

The instruction cycle (cont.)

Execute:

1. This part is really dependant on the instruction we are executing.
2. Maybe more data is read and then, processed.
3. Then the ALU would do something with it.
4. The result should be stored into a register.
5. Other registers or flags might be afected and in turn, signal for the next instruction.

RISC Architecture

- Small, simple instruction set: RISC processors use a small set of simple instructions that can be executed quickly, usually in one clock cycle.

RISC Architecture

- Small, simple instruction set: RISC processors use a small set of simple instructions that can be executed quickly, usually in one clock cycle.
- Load/store architecture: Only load and store instructions can access memory, while other instructions operate on registers.

RISC Architecture

- Small, simple instruction set: RISC processors use a small set of simple instructions that can be executed quickly, usually in one clock cycle.
- Load/store architecture: Only load and store instructions can access memory, while other instructions operate on registers.
- Pipelining: RISC designs are highly pipelined, meaning multiple instructions can be processed simultaneously at different stages (fetch, decode, execute).

RISC Architecture

- Small, simple instruction set: RISC processors use a small set of simple instructions that can be executed quickly, usually in one clock cycle.
- Load/store architecture: Only load and store instructions can access memory, while other instructions operate on registers.
- Pipelining: RISC designs are highly pipelined, meaning multiple instructions can be processed simultaneously at different stages (fetch, decode, execute).
- Energy efficiency: Simple instructions and faster execution lead to more efficient use of power, which is critical for embedded systems.

CISC Architecture

- Large, complex instruction set: CISC processors have a large number of complex instructions, some of which can perform multiple tasks in a single instruction.

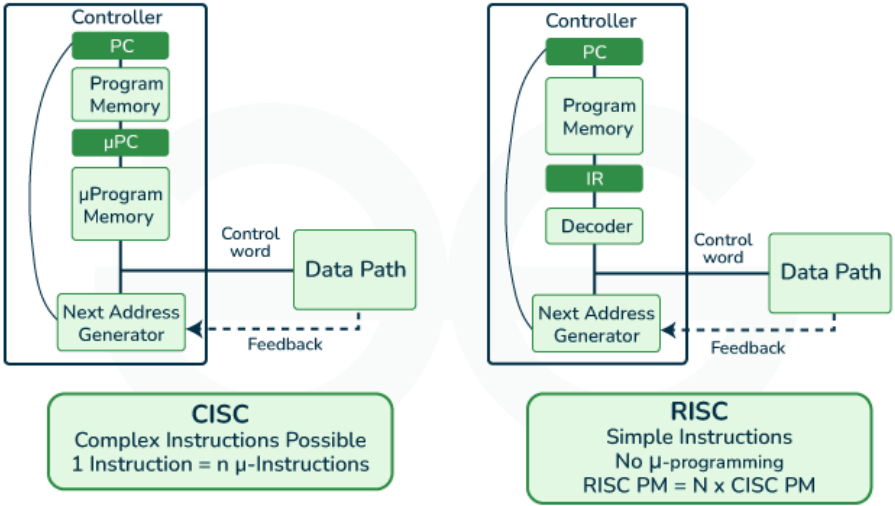
CISC Architecture

- Large, complex instruction set: CISC processors have a large number of complex instructions, some of which can perform multiple tasks in a single instruction.
- Direct memory access: CISC instructions can directly operate on memory, reducing the need for multiple load/store instructions.

CISC Architecture

- Large, complex instruction set: CISC processors have a large number of complex instructions, some of which can perform multiple tasks in a single instruction.
- Direct memory access: CISC instructions can directly operate on memory, reducing the need for multiple load/store instructions.
- Longer instruction execution time: Due to the complexity of CISC instructions, they generally take more clock cycles to execute than RISC instructions.

ATmega328P architecture



Introduction to ATmega328P

- ATmega328P is an 8-bit AVR microcontroller. (*What does this mean?*)

Introduction to ATmega328P

- ATmega328P is an 8-bit AVR microcontroller. (*What does this mean?*)
- It is widely used in Arduino boards (e.g., Arduino Uno).

Introduction to ATmega328P

- ATmega328P is an 8-bit AVR microcontroller. (*What does this mean?*)
- It is widely used in Arduino boards (e.g., Arduino Uno).
- Features a RISC architecture with 32 general-purpose registers.

Core Features

- **CPU**: 8-bit AVR with Harvard architecture.

Core Features

- **CPU**: 8-bit AVR with Harvard architecture.
- **Clock Speed**: Up to 20 MHz.

Core Features

- **CPU**: 8-bit AVR with Harvard architecture.
- **Clock Speed**: Up to 20 MHz.
- **Flash Memory**: 32 KB for program storage.

Core Features

- **CPU**: 8-bit AVR with Harvard architecture.
- **Clock Speed**: Up to 20 MHz.
- **Flash Memory**: 32 KB for program storage.
- **SRAM**: 2 KB for data storage.

Core Features

- **CPU**: 8-bit AVR with Harvard architecture.
- **Clock Speed**: Up to 20 MHz.
- **Flash Memory**: 32 KB for program storage.
- **SRAM**: 2 KB for data storage.
- **EEPROM**: 1 KB for permanent data storage.

Core Features

- **CPU**: 8-bit AVR with Harvard architecture.
- **Clock Speed**: Up to 20 MHz.
- **Flash Memory**: 32 KB for program storage.
- **SRAM**: 2 KB for data storage.
- **EEPROM**: 1 KB for permanent data storage.
- **I/O**: 23 GPIO pins for external interfacing.

AVR RISC Architecture

- Reduced Instruction Set Computing (RISC).

AVR RISC Architecture

- Reduced Instruction Set Computing (RISC).
- Only 131 instructions.

AVR RISC Architecture

- Reduced Instruction Set Computing (RISC).
- Only 131 instructions.
- Most instructions execute in a single clock cycle.

AVR RISC Architecture

- Reduced Instruction Set Computing (RISC).
- Only 131 instructions.
- Most instructions execute in a single clock cycle.
- Efficient use of general-purpose registers (32 in total).

Memory Organization

- **Program Memory**: 32 KB Flash for storing code.
- **Data Memory**: 2 KB SRAM for runtime data.
- **EEPROM**: 1 KB for permanent data storage.
- **I/O Memory**: Special function registers (SFR) for controlling peripherals.

Interrupt System

- 24 Interrupt vectors with varying priorities.
- External interrupts: INT0, INT1.
- Internal interrupts: Timer, USART, ADC, etc.
- Can trigger service routines to handle events like I/O changes, timer overflow, etc.

Peripherals Overview

- **Timers/Counters**: Three timers (Timer0, Timer1, Timer2) for timing applications.
- **USART**: Supports serial communication.
- **ADC**: 10-bit ADC for analog-to-digital conversion.
- **SPI/I2C**: Interfaces for communicating with other devices.

Summary

- ATmega328P is an efficient and widely used 8-bit microcontroller.
- Implements RISC and Harvard architecture for high performance.
- Provides ample I/O options and peripherals for various embedded applications.

Table of Contents

Any Questions?

References

[1] Alan Clements. *Principles of Computer Hardware*. Oxford University Press, 2006.

[2] Arm Ltd. *What is Instruction Set Architecture (ISA)? — arm.com*. URL: https://www.arm.com/glossary/isa (visited on 09/24/2024).