# Introduction to Microcontroller Concepts

You should be able to describe:

* The differences between microcontrollers and microprocessors.
* In which application areas microcontrollers are typically found.
* What is in a CPU, and what else it needs to be useful?
* How many bits in, e.g., a KibiByte
* How many busses are needed to connect to memory?
* What the bus width relates to in the case of the address bus and the data bus.
* Pros and cons of a parallel bus over a serial bus.
* Why microcontroller pins can do many different tasks, and the programmer sets what each pin does.

Terminology

You should be able to define the following terminology.

* *Microcontroller* - an integrated circuit that contains a microprocessor, memory, and peripheral electronics.
* *Embedded System* - a processor plus associated electronics that is used for a specific purpose.
* *Bit* - A single binary digit capable of representing the logic states ‘0’ and ‘1’.
* Nibble - A group of 4 bits.
* Byte - A group of 8 bits.
* Word - A general term for a group of one or more bits.
* Word length - The number of bits in a word.
* Radix - The default base number system for the values in your program.
* Digit - Base 10 number system, any of the numerals from 0 to 9.
* Parallel Bus - A collection of wires that transfer a word of data simultaneously.
* Memory depth – the number of individual memory locations within the memory.
* Memory width – the size in bits of the data within an individual memory location.
* Memory capacity – the total number of bits that can be stored in the memory.

## Base, Radix

We will uses numbers that have different bases, e.g. binary=base 2, decimal=base 10, hexadecimal is base 16. Another name for “base” is “radix”. The term radix is used in Microchip’s assembler MPASM.

“The **radix** (aka. base) is the number of unique digits, including zero.”

## Digits, Bits, Bytes, and Words

The term digit is often used for the decimal system, for the numbers 0-9. For binary numbers, we have “**bi**nary digi**t**” – which is shortened to the word “**bit**”. We can also use the term “hexadecimal digit” for base 16 numbers.

* The **bit** (b) is the smallest unit of information in the binary numeration system and is represented by the digits 0 and 1.
* The **byte** (B) is a binary character string typically operated upon as one unit. We will define it simply **as a collection of 8 bits**
* A **word** is the number of bits used by default by the system.
  + A word of bits used the 32 bit ARM Cortex-M4 is therefore 32 bits, 4 bytes.
  + The Microchip PIC18F microcontroller that we will be studying has uses 8-bit data. So a word in this PIC18F is 8 bits.

For a definition see JEDEC: https://www.jedec.org/standards-documents/dictionary?

## Background concepts for microcontrollers

A microcontroller combines a CPU, memory, and peripheral electronics, within a single integrated circuit. Microcontrollers are often used in embedded systems.

### What is an Embedded System?

An embedded system is one that is designed for a specific purpose, rather than for general purpose computing. An embedded contains a processor plus associated electronics that is used for a specific purpose, for example, a web camera or a vending machine.

Compare this to a PC that can perform many tasks. However sometimes cheap, relatively low power “embedded PCs” are used by constraining them to run a single program as can be seen when cash machines or Railway timetable screens go wrong and show a Microsoft Windows crash screen (the “blue screen of death”).

In a 2013 Embedded market survey,

* “What 32-bit chip will you use next?” **Answer**: Microchip (23%), STMicro ARM (22%), TI ARM (22%, 17%), NXP ARM (17%) (note %>100%...)
* “How are embedded systems programmed?” **Answer**: C 60%, C++ 20%, Assembly 5%, Java 3%

Note that you are not learning assembly to program embedded systems, you are learning it to understand how processors work.

In what types of applications are embedded systems/microcontrollers used?

The top 3 are Industrial Automation, Consumer Electronics, and Communication Networks.

### Addressable memory and word size

Data and programs need to be kept somewhere. Data in memory is not normally used sequentially, but the processor will jump around the memory getting and replacing the information within the memory. Every memory location has a unique address which enables the processor to select different specific areas within the memory.

Different processors are designed to be good at working on a specific sized number, e.g. the PIC18F is good at working with 8-bit numbers, while the ARM Cortex M4 is good at working with 32-bit numbers. The default data size of the processor is the word size. So the word size of the PIC18F is 8.

### Registers

Registers are small bits of memory associated with fast data processing. They hold one piece of data. They are located near the maths processing part of the processor. Since they hold one piece of data, in general, the register size is the same as the word size of the processor.

The PIC18F has only one register that is used for processing data. It is called the Working register.

### A Processor

The Central Processing Unit (CPU), also known as the “processor”, comprises:

* Logic to break the machine code instruction into its constituent parts – this is the **instruction decoder**. You will later learn that the machine code parts are the opcode (command) and the operand (a data value or a memory address).
* Logic to control the action of the system to carry out the current machine code instruction - “**Control Unit**”. The control unit orchestrates all the logic within the processor.
* Somewhere to store temporary numbers - **Registers**,
* A register to remember current the place in the machine code program – **Program Counter**.
* Some logic with the ability to maths – called the **Arithmetic and Logic Unit** or “**ALU”**, or combinational logic. The ALU is sometimes also called Combinational Logic.
* Registers, which are small, fast bits of memory for temporally storing numbers.

To operate, the processor also needs:

* some way get the number in/out of the computer - “**Peripheral Interface**”,
* somewhere to store our program - “**Program Memory**”,
* somewhere to store the in/out numbers once we get them. “**Data memory**”.

In a microprocessor, these three items are normally external to the chip, whereas In a microcontroller they are located within the chip.

### How programs work

When microcontroller starts up will fetch the first machine code instruction from address 0x000000. We want to make sure that our program is placed in program memory, so the first line of our program is placed at program memory location 0x000000. This is what the **org 0x000** code in the header of your assembly program does - it sets the **or**i**g**in of the program.

Once the first line of machine code has been completed, the next line of machine code is taken from program memory, and this line is then executed. This process continues forever unless the current machine code program tells the process to jump to a new location within the program.

The current place within the program is remembered by the processor by having a **Program Counter** register.

### Programs never stop

The end statement at the end of your assembly program is not a machine code instruction; it is just to help the compiler. The end statement, therefore, does not stop the program and in fact, the processor would try to continue processing any commands in the program memory that come after your program. You must stop it doing this!

To stop a program you can get it to loop back to somewhere else within your program. The simplest way to do this is just to keep jumping back to the same statement. The assembly command “BRA L1” means jump to the place in the program where the label ‘L1’ is. So if your line of assembly is “L1 BRA L1”, then the label L1 is on the same place in program memory as the jump command, and the processor will just keep repeating this line again and again. That way you stop the processor from going any further.

## Microcontrollers

A micro**processor** is designed to perform fast calculations, with the majority of memory and peripherals on separate chips. A microcontroller is designed to be cheap and low power and have its CPU, memory and peripherals on the same integrated circuit.

A micro**controller** is designed to be small. Being small often means that it has very few pins, yet it must be flexible enough to be used in many different applications. To do this, each of its pins can perform many different functions (e.g. turning on LEDs, measuring a voltage, performing serial communication with another device. However in any particular application, the pin will likely only perform one task, e.g. it will be connected to an LED. It is up to the programmer (i.e. YOU) to tell the microcontroller what each pin should do.

### Microcontroller pin capabilities

To be small, microcontrollers only have a few pins, typically between 8 and 100 pins. The manufacturer of the microcontroller wants the same chip to go in too many different products. Each product may require the microcontroller’s pins to do different things. For example, a microcontroller for a motor application might need a pin to create a PWM signal, while for a temperature measurement application, a pin that can measure voltage is required.

The microcontroller can be used for many different applications while only having a few pins but giving each pin the capability to do different things. You will see all the things the PIC18F pins can do in the MCE1 Lab Handbook. You will investigate this in the Lab, and you should complete the tables as pre-lab work (i.e. In your own time before the lab).

It is up to the programmer to configure the pin depending on what the pin is connected to. You will connect the microcontroller board to your I/O board so you will need to set the function for each microcontroller pin connected to LEDs, switches, light sensors etc.

You will need to consider if the particular microcontroller pin is an input or output:

* If the microcontroller is measuring a voltage (like a multimeter), then the pin should be set as an input.
* If the microcontroller is controlling a voltage, e.g. to turn on/off an LED, then the pin should be set as an output.
* If the microcontroller just need to know if the input value is on or off, then the pin should be set as a digital input
* If the microcontroller wants to measure the voltage at its pin, then the pin should be set as an analogue input.

## Microcontrollers Vs Microprocessors

A microprocessor is designed to perform fast calculations, with the majority of memory and peripherals on separate chips. A microcontroller is designed to be small, cheap and low power. To do this it has a CPU, memory and peripherals all on the same integrated circuit.

|  |  |
| --- | --- |
| **Microprocessor** | **Microcontroller (MCU)** |
| Many pins - address and data buses | Few pins, no memory busses. |
| Pins just to one job | Pins can do multiple things, the programmer decides. |
| Little Peripheral IO logic | Peripheral IO on chip |
| Has some fast (cache) memory but needs additional memory (DRAM). | All the required data memory is on-chip. SRAM (volatile) for data/registers. Some have EEPROM (non-volatile) to store small amounts of data |
| Multiple programs, all stored off-chip | Single program, stored within the chip, using EEPROM |
| Expensive (Intel Core i7-5960X £800) | Cheap (ARM M4 around £4) |
| Most are powered via the mains, especially PCs. | Most are for battery operation. |
| For number crunching, and performing many tasks at the same time. | Used for hardware control, and for performing only a few tasks at the same time. |

### Comparing common microcontrollers

The table below compares four common microcontrollers. However it contains terminology that you will learn about later in the course, so remember to come back here at the end of the Unit!

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **AT89C51 (8051)** | **Microchip**  **PIC18F8** | **AVR ATmega168**  **(Arduino1)** | **ARM**  **STM32F4** |
| **Microarchitecture - Memory** | Von Neumann architecture | Harvard architecture | Modified Harvard architecture | Modified Harvard architecture |
| **ISA** | CISC | Partially RISC | RISC | RISC |
| **Clocks / instruction cycle** | 12 | 4- | 1 | 1 |
| **Memory** | ROM, SRAM, FLASH | S-RAM, FLASH | S-RAM, FLASH | Flash, SRAM, EEPROM |
| **Data Bus width** | 8-bit | 8-bit | ATmega168 8-bit | 32-bit |

**1**The term “Arduino” refers to the programming API, not the processor underneath. There are many types of processors that run Arduino - see https://www.arduino.cc/en/Products/Compare

## Parallel Bus

An electronic parallel bus is a collection of wires that transfer a word of data simultaneously. Each wire contains 1 bit of the data. Therefore there are as many wires as there are bits in the word.

In a microcontroller, components within the CPU are connected with a data bus to pass the data values around. The source/destination of where this data is stored in the data memory is set by another bus, called the data memory address bus.

Electronic memory requires two busses: one for carrying the data value (the data bus) and another for selecting which part of memory to access (the address bus).

A parallel bus can transfer data faster than a serial bus simply because it has more wires. However, more wires mean more space is required and this can also make the routing of wires more difficult.

When drawing a parallel bus on a schematic, rather than drawing multiple wires, the bus is drawn as a single line but with the number of wires within it written next to it. The example below shows an 8-wire bus going to the D pins (D0-D7) of the IC, and a 12 but bus connected to the A pins (A0-A11).

D[0..7]

8

A[0..11]

12

## Bus Width, memory capacity and literals

The memory address bus controls the memory. The width of the memory address bus dictates how many individual memory locations there can be, e.g. if the address bus is 12 bits wide, then there can be at most 4096 different memory locations.

Each of these individual memory locations can hold a single value. The number of bits in this value is determined by the size of the data-bus. For instance, an 8-bit data bus can hold values from 0 to 255.

**It is the size of the data-bus that gives the processor its name,** e.g. an “8-bit processor” means it has an 8-bit data bus and therefore the default range values it can manipulate are between 0 and 255. It does not say anything about any other bus - an 8-bit processor could have, for example, a 14-bit data memory address bus, a 16-bit program memory address bus, and a 32-bit program memory data/instruction bus.

The program memory’s data bus is also called the **instruction-bus,** and it dictates the size of the longest machine code instruction. The data memory **data-bus** provides data values, and its size depends on how many bits are in the largest allowable data value.

The total number of bits that can be stored in the memory is called the memory capacity. To work out the memory capacity, you multiply the number of addresses the memory has by the number of bits in the data value. E.g. A memory with a 12-bit address but and an 8-bit data bus, has a memory capacity of 4096x8=32768 bits

The combinatorial logic (ALU) combines two numbers, so it has two data buses feeding into it. The width of each of these buses is same as the data word size. The output data bus of the ALU is also the same data word size so it is possible that the result of the addition is bigger than the bus can hold and the answer will be wrong!

Some machine code instructions contain data values within themselves. These data values are called “Literals”. Since the machine code instruction can contain data, the width (number of bits) to encode the instruction must be bigger than the number of bits in the default data size. Therefore the instruction value bus width must be bigger than the data bus width. In the PIC18F, the instructions bus width is 16-bits while the data bus width is 8-bits.

Does the data memory address bus width have to be wider than the data bus? It is in the PIC18F, but it does not have to be. For instance, a 64-bit processor would have a 64-bit data bus (by definition), but it would be unlikely to require 2^64 = 16 million Terabytes of memory. So, in this case, memory address bus width would be smaller than the data-bus width.

## Memory Dimensions and Sizes

Virtually all semiconductor memory devices are organised as a set of *memory locations*, with each memory location storing an identical number of bits. The process of selecting one unique location in the memory is known as *memory addressing*.

Shown below is a component diagram of a generic memory device. The memory has an *n*-bit data input **D**, an *n*-bit data output **Q**, a *k*-bit address selection input **A**. The chip selection (CS) and write enable (WE) control signals will be discussed later.

A memory device with a *k*-bit address selection input will contain 2*k* memory locations. The binary pattern on the input A will uniquely select one of these locations. Each memory location in the device will contain *n* bits. This figure implies that the organisation of the memory device a set of 2*k* memory locations, with each location storing *n* bits.

Q

D

n

n

A

k

CS

WE

There are various notations used for specifying the capacity of a memory device. The most common notation specifies the total number of memory locations in the device (termed the ***depth***) and the number bits in each memory location (termed the ***width***). For the generic memory device shown above, the size would be specified as *depth* by *width*, “2k by n”.

Two other common notations state the total capacity of the memory in bits or bytes. For the memory device shown above, the **memory capacity** in bits is 2k × n and the memory capacity in bytes is 2k × n / 8.

Compare the **n** and **k** in the diagram above to the diagram on the next page.

We discuss the memory size in binary units (Kibi), not SI units (kilo).

0

1

2

3

4

2k - 1

2k - 2

Address =

***n*** bits per word (width)

data word

***k*** bit address

2***k*** memory locations (depth)

Address =

bit 0

bit n-1

## Orders of Magnitude –SI convention

Orders of magnitude of decimals - the SI (international system of units)

For most physical quantities we use powers of 10, so, thousands, millions, billions and to write down the measurements we use the SI system of powers of 10: kilo, Mega, Giga, e.g. 1 kilogram (1 kg), one megavolt (1 MV)

Note SI requires a space after the number. See section 5.3.3 in http://www.bipm.org/en/publications/si-brochure/section5-3.html

1 MV is correct, 1MV, 1M V, are both wrong,

2 km is correct, 2km, 2k m, are both wrong

## Orders of Magnitude – IEC convention

The terms *kilobyte* and *megabyte* are often a source of confusion.

In the International System of Units (SI), the prefix *kilo* means 103 = 1000, and the prefix *mega* means 106 = 1,000,000. SI (International System of Units) are in powers of 10, e.g. kilo = 1000 = 103, however binary numbers use powers of 2. To address the confusion, some standardisation bodies have agreed on alternative abbreviations for the binary multiples.

In the field of computer science, the prefix *Kibi* means 210 = 1024, and the prefix *Mebi* means 220 = 1048576. These prefixes are abbreviated to Ki (capital letter) and Mi.

Please follow the **IEC** convention below.

For bits:

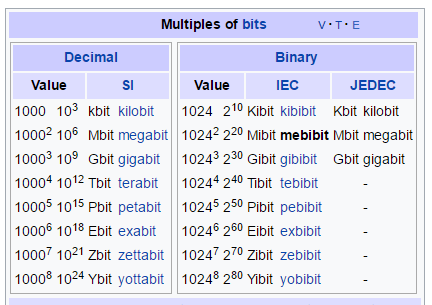


Figure 1 From Wikipedia (Mebibit)

For bytes:

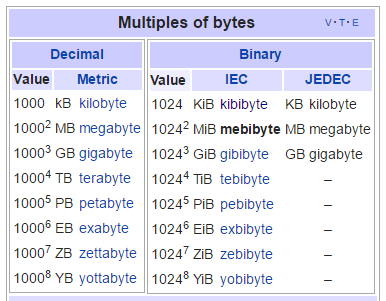


Figure 2 From Wikipedia (Mebibyte)

**Base 2 orders of magnitude**

210 = 1024 kibi (Ki)

220 = 1,048,576 mebi (Mi)

230 = 1,073,741,824 gibi (Gi)

240 = 1,099,511,627,776 tebi (Ti)

## References

BBC Computer Studies - <http://www.bbc.co.uk/bitesize/standard/computing/computer_systems/low_level_machines/revision/5/>

## Example Exam Questions

Exam Qu. 4 – 2016

1. ATMEL use an 8-bit CPU core with a Harvard architecture in their ATMEGA series. The ATMEGA8 microcontroller effectively has a 12-bit program memory address bus width.

What is:

The largest number that can be put onto the data memory data bus? Provide the value in decimal representation.

The number of accessible program memory locations?

1. The ATMEL device in part (b) has a 16-bit wide instruction set with matching program memory instruction bus width.

What is:

* 1. The capacity of the program memory in bits?
  2. The capacity of the program memory in KiBytes?