# **Embedded programming: Comparing the performance and development workflows for architectures**

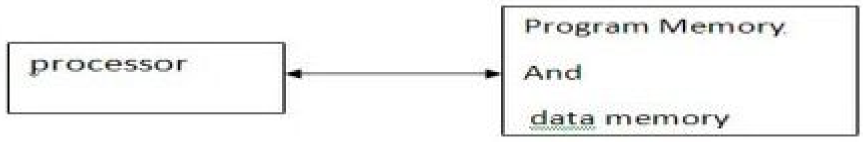
Embedded programming week
FABLAB BRIGHTON 2018

## What do we mean by architecture?

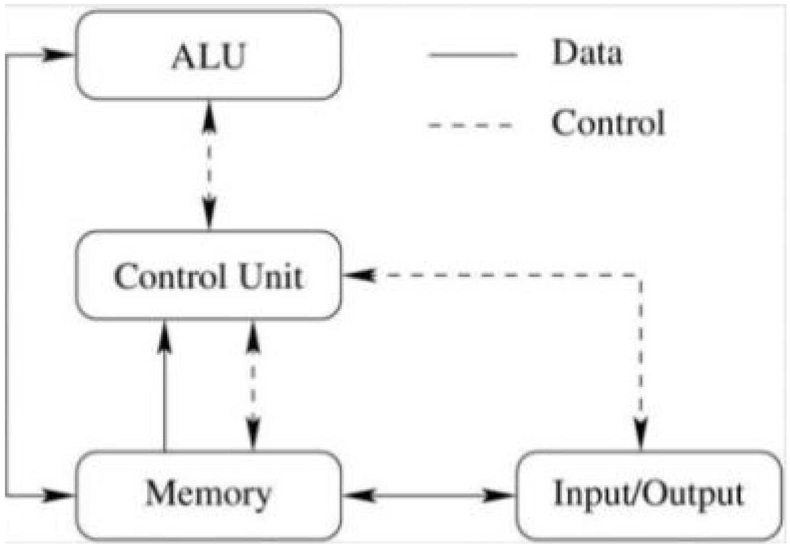
The architecture of microprocessors and microcontrollers are classified based on the way memory is allocated (memory architecture). There are two main ways of doing this:

### Von Neumann architecture (also known as Princeton)

Von Neumann uses a single unified cache (i.e. the same memory) for both the code (instructions) and the data itself,

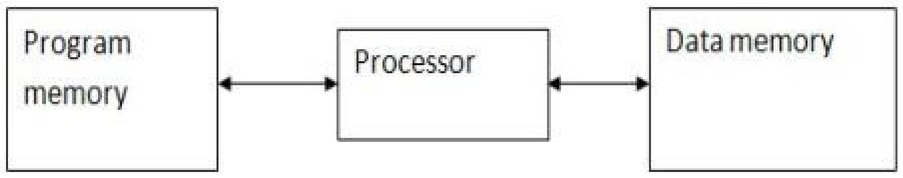


Under pure von Neumann architecture the CPU can be either reading an instruction or reading/writing data from/to the memory. Both cannot occur at the same time since the instructions and data use the same bus system.

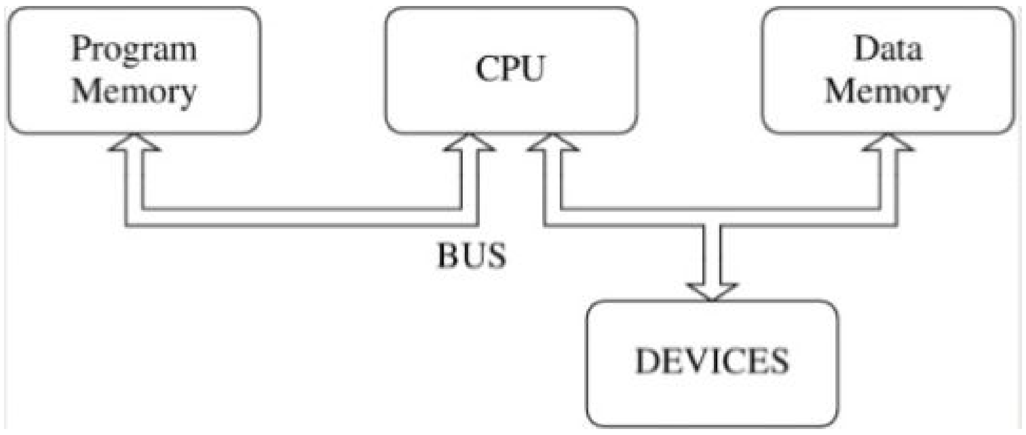


### Harvard architecture

Harvard architecture uses different memory allocations for the code (instructions) and the data, allowing it to be able to read instructions and perform data memory access simultaneously.



The best performance is achieved when both instructions and data are supplied by their own caches, with no need to access external memory at all.

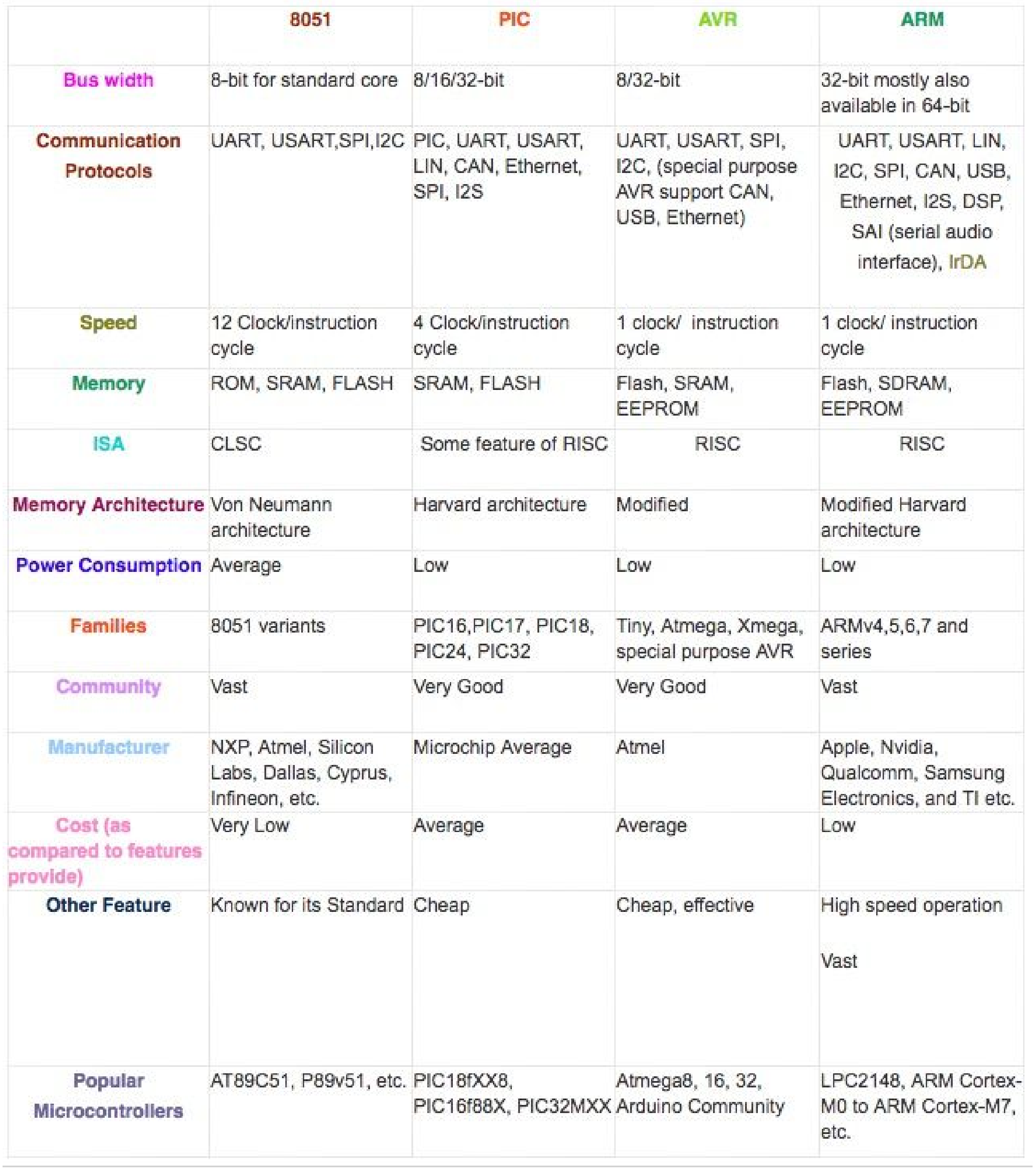


## How does this relate to microcontrollers/microprocessors?

We found this page to be a good introduction to the topic of microcontrollers and microprocessors, the architectures they use and the difference between some of the common types. First though, it's worth looking at the difference between a microprocessor and a microcontroller. **Microprocessors (e.g. ARM)** generally consist of just the Central Processing Unit (CPU), which performs all the instructions in a computer program, including arithmetic, logic, control and input/output operations. **Microcontrollers (e.g. AVR, PIC or 8051)** contain one or more CPUs with RAM, ROM and programmable input/output peripherals. Microprocessors tend to operate at much greater clock speeds on general application tasks, like gaming, photo editing and software development, whereas microcontrollers are designed for more specific tasks or development in smaller embedded systems like keyboards, mice, electronic toys and vending machines.

It's worth noting that Raspberry Pi computers use ARM processors, while Arduino development boards use AVR microcontrollers (ATtiny, ATmega etc).

We found the following table from this site to be a useful summary comparing the various microcontrollers (8081, PIC and AVR) with the ARM microprocessor.



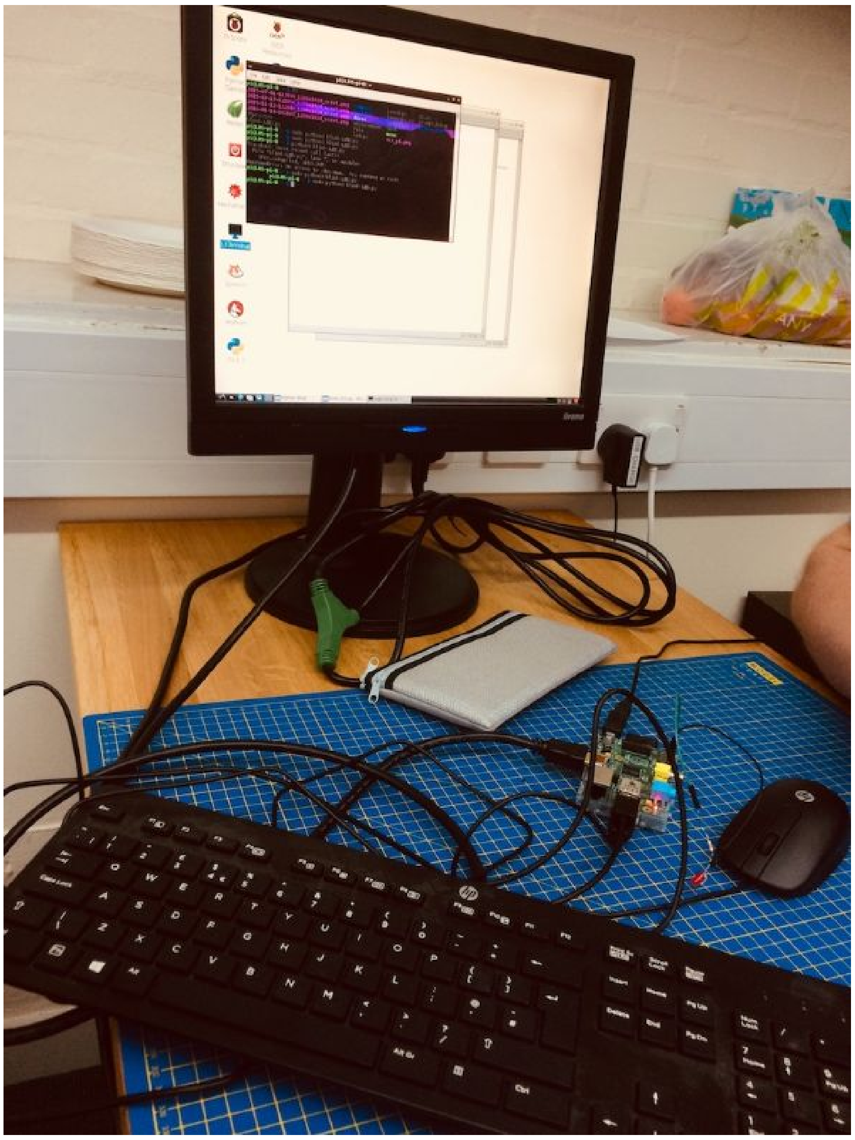
| | 8051 | PIC | AVR | ARM |
|---|---|---|---|---|
| Bus width | 8-bit for standard core | 8/16/32-bit | 8/32-bit | 32-bit mostly also available in 64-bit |
| Communication Protocols | UART, USART,SPI,I2C | PIC, UART, USART, LIN, CAN, Ethernet, SPI, I2S | UART, USART, SPI, I2C, (special purpose AVR support CAN, USB, Ethernet) | UART, USART, LIN, I2C, SPI, CAN, USB, Ethernet, I2S, DSP, SAI (serial audio interface), IrDA |
| Speed | 12 Clock/instruction cycle | 4 Clock/instruction cycle | 1 clock/ instruction cycle | 1 clock/ instruction cycle |
| Memory | ROM, SRAM, FLASH | SRAM, FLASH | Flash, SRAM, EEPROM | Flash, SDRAM, EEPROM |
| ISA | CLSC | Some feature of RISC | RISC | RISC |
| Memory Architecture | Von Neumann architecture | Harvard architecture | Modified | Modified Harvard architecture |
| Power Consumption | Average | Low | Low | Low |
| Families | 8051 variants | PIC16,PIC17, PIC18, PIC24, PIC32 | Tiny, Atmega, Xmega, special purpose AVR | ARMv4,5,6,7 and series |
| Community | Vast | Very Good | Very Good | Vast |
| Manufacturer | NXP, Atmel, Silicon Labs, Dallas, Cyprus, Infineon, etc. | Microchip Average | Atmel | Apple, Nvidia, Qualcomm, Samsung Electronics, and TI etc. |
| Cost (as compared to features provide) | Very Low | Average | Average | Low |
| Other Feature | Known for its Standard | Cheap | Cheap, effective | High speed operation<br><br>Vast |
| Popular Microcontrollers | AT89C51, P89v51, etc. | PIC18fXX8, PIC16f88X, PIC32MXX | Atmega8, 16, 32, Arduino Community | LPC2148, ARM Cortex-M0 to ARM Cortex-M7, etc. |

## Using a Raspberry Pi to run an LED flashing programme in Python

We've all now had some experience using an AVR microcontroller (ATtiny85) on our own boards and using Arduinos, and how to interface with this using the Arduino IDE, so as part of our class activity we spent some time together using a Raspberry Pi (which uses and ARM microprocessor) to replicate the blinking LED programme that we'd achieved with our own boards using the Arduino IDE. Many thanks to Mike for running the session!

Firstly we went through the concept of General Purpose Input and Output pins (GPIO), and found that this site gives a good introduction with some good resources.

Here's the Raspberry pi setup we used with keyboard and mouse plugged into the USB ports and the monitor plugged in to the HDMI. The LED was plugged into the GPIO pin on the Raspberry pi (pin 7).



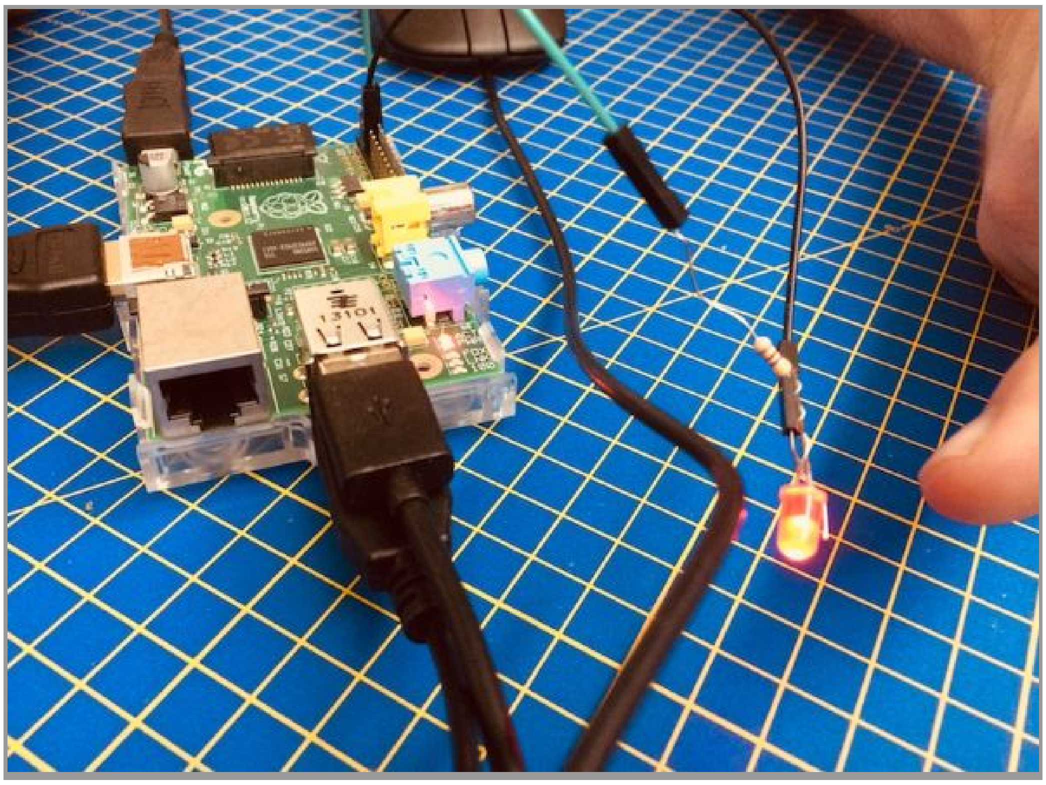
We then created a python programme using the code below, and saved this as **blink-LED.py** in the root directory.

```
import RPi.GPIO as GPIO
import time
GPIO.setmode (GPIO.BOARD)
led = 7
GPIO.setup(led, GPIO.OUT)
for blink in range (1,111):
        GPIO. output(led,1)
        time.sleep(0.1)
        GPIO.output(led,0)
        time.sleep(0.1)
GPIO.cleanup()
```

In the LX Terminal programme (on the desktop), we typed:

```
Sudo python3 blink-LED.py
```

Which ran the programme in Python3 and the LED blinked!



## References:

1. https://www.elprocus.com/difference-between-avr-arm-8051-and-pic-microcontroller/
2. https://www.quora.com/What-is-the-difference-between-the-Von-Neumann-architecture-and-the-Harvard-architecture
3. http://www.edgefxkits.com/blog/difference-between-von-neumann-and-harvard-architecture
4. https://learn.sparkfun.com/tutorials/raspberry-gpio