Embedded Systems: Tier A

0x00 Intro-Embedded Systems:

- **embedded characteristics:**

1. **Performs specific task:** Embedded systems perform some specific function or tasks.
2. **Low Cost:** The price of an embedded system is not so expensive.
3. **Time Specific:** It performs the tasks within a certain time frame.
4. **Low Power:** Embedded Systems don't require much power to operate.
5. **High Efficiency:** The efficiency level of embedded systems is so high.
6. **Minimal User interface:** These systems require less user interface and are easy to use.
7. **Less Human intervention:** Embedded systems require no human intervention or very less human intervention.
8. **Highly Stable:** Embedded systems do not change frequently mostly fixed maintaining stability.
9. **High Reliability:** Embedded systems are reliable they perform tasks consistently well.
10. **Use microprocessors or microcontrollers:** Embedded systems use microprocessors or microcontrollers to design and use limited memory.
11. **Manufacturable:** The majority of embedded systems are compact and affordable to manufacture. They are based on the size and low complexity of the hardware.

- **Microcontroller Vs. Microprocessor:**
    - A microprocessor is suitable for applications that require more processing, operates faster, requires more memory, is more extensive, and consumes more power.
    - A microcontroller performs a particular task, operates at low clock speed, and requires less memory than a microprocessor.

	Microprocessor	Microcontroller
Applications	Advanced data processing, video, computer vision, personal computer, multi-core computations.	Embedded devices, control systems, smart phones, consumer electronics.
Processing power	Higher	Lower
Memory	External-Flexible	Internal-Limited Size
Power Consumption	Higher	Lower
Size	Larger	Smaller
Price	Expensive	Cheaper
I/O	Need external peripherals with I/O pins	Programmable digital and analog I/O pins

Types of memory:

1. RAM

- **A. Static RAM (SRAM)**

  Definition and Properties of SRAM

  Static Random Access Memory (SRAM) is a type of volatile memory used in embedded systems. SRAM stores data using flip-flop circuits, ensuring that the data remains intact as long as power is supplied. This type of memory offers high-speed access and low power consumption, making it ideal for fast data retrieval and temporary storage.

- **Common Use Cases of SRAM in Embedded Systems**

  In embedded systems, SRAM is often used for critical data paths and cache memory due to its fast access time. It is also common to find SRAM in combination with other memory types, such as DRAM, to create a balance between performance and cost. Some typical applications include microcontrollers, digital signal processors, and high-speed data buffers.

- **B. Dynamic RAM (DRAM)**

  Dynamic Random Access Memory (DRAM) is another type of volatile memory commonly used in embedded systems. Unlike SRAM, DRAM stores data in capacitors, which require periodic refreshing to maintain their charge. Due to this refresh requirement, DRAM has a shorter data lifetime and slower access time compared to SRAM.

- **DRAM Controllers and Their Function**

  DRAM controllers are essential components that make DRAM more usable by periodically refreshing the data stored in the memory. By refreshing the data before it expires, the contents of the memory can be preserved for as long as needed, effectively making DRAM as useful as SRAM for certain applications.

- **Common Use Cases of DRAM in Embedded Systems**

  DRAM is often used in embedded systems that require large memory capacities, such as multimedia devices, communication systems, and data storage applications. In many cases, embedded systems include a combination of SRAM for critical data paths and DRAM for larger storage requirements, striking a balance between performance and cost.

2. Read-Only Memory (ROM)

- **A. Masked ROM**

  Definition and properties: Masked ROM is a type of non-volatile memory that contains preprogrammed data or instructions. The contents of the memory are specified before chip production, allowing the data to be permanently stored in the device. This type of ROM is also referred to as hardwired memory.

- **B. Programmable ROM (PROM)**

  Definition and properties: Programmable ROM (PROM) is a type of non-volatile memory that can be programmed after manufacturing. It is purchased in an unprogrammed state and can be written using a special device programmer. Once programmed, the data stored in a PROM cannot be changed, making it a one-time programmable (OTP) device.

- **C. Erasable and Programmable ROM (EPROM)**

  Definition and properties: Erasable and Programmable ROM (EPROM) is similar to PROM but can be erased and reprogrammed multiple times. EPROM devices can be erased using a strong source of ultraviolet light, resetting the entire chip to its initial unprogrammed state.

  3. Hybrid Memory Devices

Hybrid memory devices are a unique class of memory components that combine features of both RAM and ROM. These devices can be read and written as desired, like RAM, but maintain their contents without electrical power, just like ROM. Hybrid memory devices include Electrically Erasable Programmable ROM (EEPROM), Flash Memory, and Non-Volatile RAM (NVRAM).

A. Electrically Erasable Programmable ROM (EEPROM)

Definition and properties: EEPROM is a non-volatile memory that can be electrically erased and programmed. It is similar to EPROM but offers byte-by-byte erasure and reprogramming capabilities, allowing for more precise control over memory contents.

B. Flash Memory

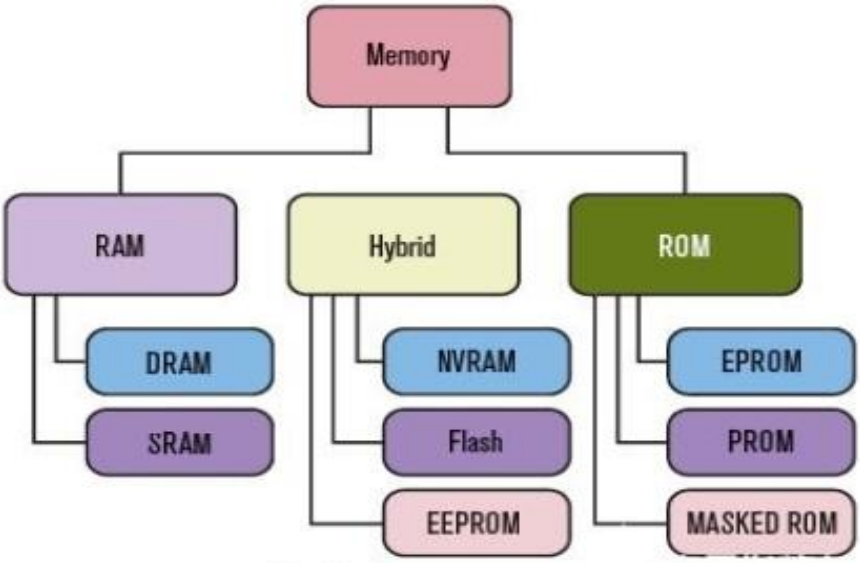
Definition and properties: Flash memory is a high-density, non-volatile memory that is electrically reprogrammable and offers fast read access. It is a popular choice for embedded systems due to its combination of high capacity, low cost, and fast read performance.

Popularity in embedded systems: Flash memory has become increasingly popular in embedded systems as it combines the best features of various memory types. Its high density, low cost, and fast read performance make it

an ideal choice for storing code and data in a wide range of applications.

C. Non-Volatile RAM (NVRAM)

Definition and properties: NVRAM is a modified version of SRAM that maintains its data even without power. It is commonly used to store persistent data in embedded systems.



• Introduction to ISA:

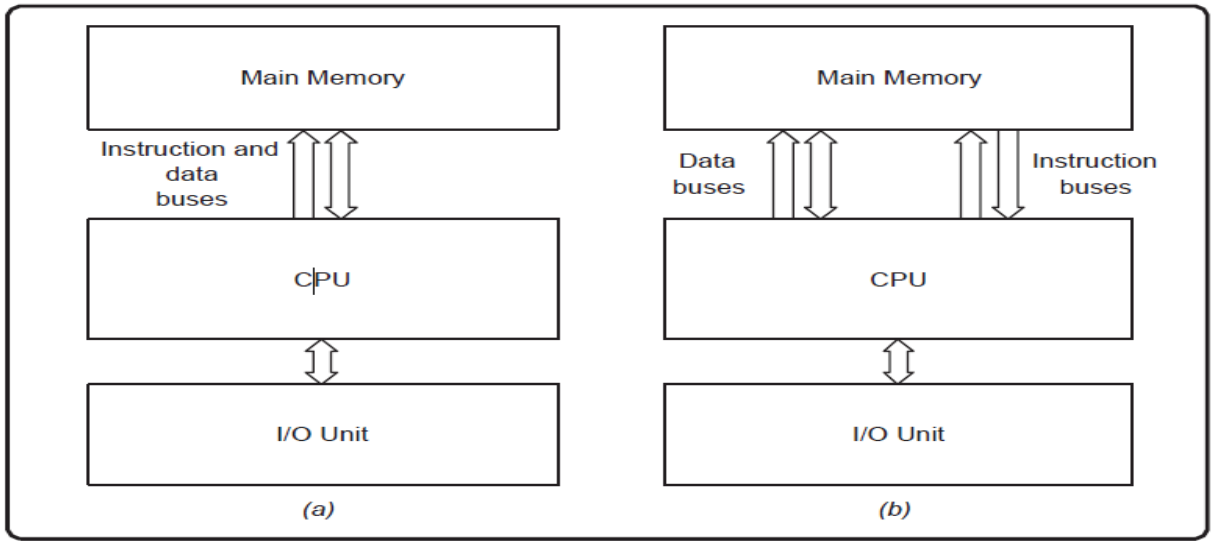
Instruction set architecture (ISA) is a bridge between the software and hardware of a computer. It functions as a programmer's viewpoint on a machine. Computers can only comprehend binary language (0 and 1), but humans can comprehend high-level language (if-else, while, conditions,

and the like). Consequently, ISA plays a crucial role in user-computer communications by translating high-level language into binary language. In addition, ISA outlines the architecture of a computer in terms of the fundamental activities it must support. It's not involved with implementation-specific computer features. Instruction set architecture dictates that the computer must assist:

- **Arithmetic/logic instructions:** These instructions execute various mathematical or logical processing elements solely on a single or maybe more operands (data inputs).
- **Data transfer instructions:** These instructions move commands from the memory or into the processor registers, or vice versa.
- **Branch and jump instructions:** These instructions are essential to interrupt the logical sequence of instructions and jump to other destinations.

• computer architecture:

Computer architecture refers to the end-to-end structure of a computer system that determines how its components interact with each other in helping to execute the machine's purpose (i.e., processing data), often avoiding any reference to the actual technical implementation.



Computers are an integral element of any organization's infrastructure, from the equipment employees use at the office to the cell phones and wearables they use to work from home. All computers, regardless of their size, are founded on a set of principles describing how hardware and software connect to make them function. This is what constitutes computer architecture.

Computer architecture is the arrangement of the components that comprise a computer system and the engine at the core of the processes that drive its functioning. It specifies the machine interface for which programming languages and associated processors are designed. Complex instruction set computer (CISC) and reduced instruction set computer (RISC) are the two predominant approaches to the architecture that influence how computer processors function.

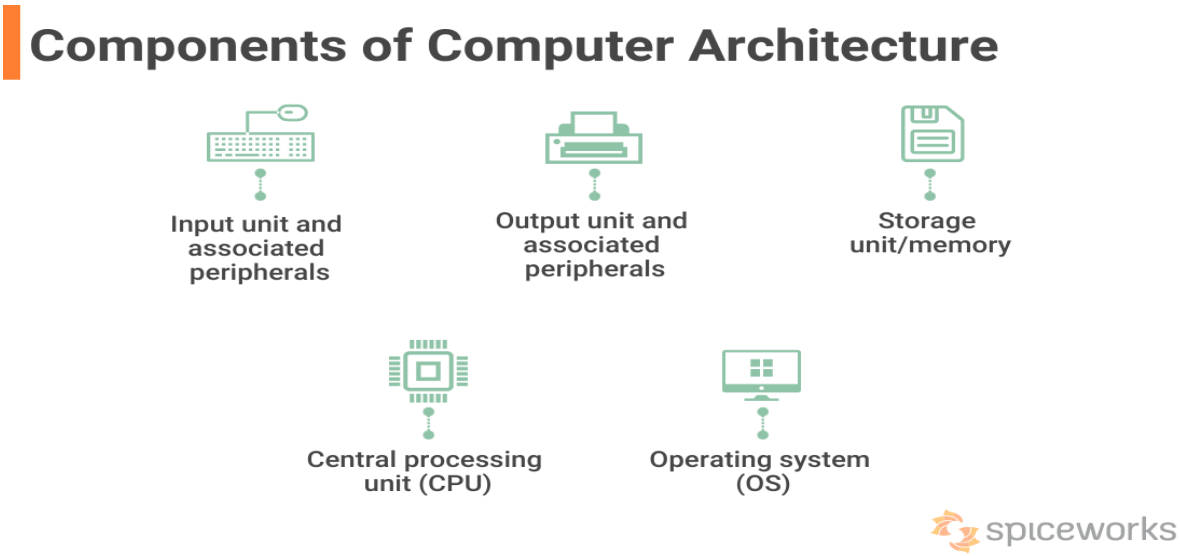
CISC processors have one processing unit, auxiliary memory, and a tiny register set containing hundreds of unique commands. These processors execute a task with a single instruction, making a programmer's work simpler since fewer lines of code are required to complete the operation.

This method utilizes less memory but may need more time to execute instructions.

A reassessment led to the creation of high-performance computers based on the **RISC** architecture. The hardware is designed to be as basic and swift as possible, and sophisticated instructions can be executed with simpler ones.

Components of Computer Architecture

Depending on the method of categorization, the parts of a computer architecture can be subdivided in several ways. The main components of a computer architecture are the CPU, memory, and peripherals. All these elements are linked by the system bus, which comprises an address bus, a data bus, and a control bus. Within this framework, the computer architecture has eight key components, as described below.



- **Extract Info from datasheet:**

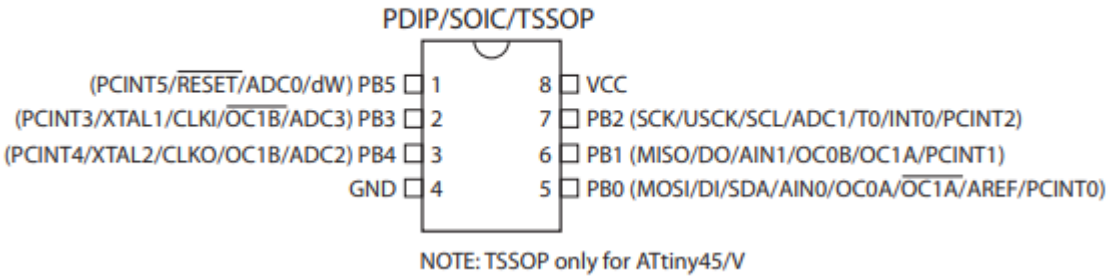
ATtiny85 Microcontroller tutorial: GPIO as output:

If you are making a hardware project that requires a microcontroller, don't just go for MCU that's lying around. Choose Microcontroller that's fit for the job. Don't go for a 32 bit MCU when all you need to do is take input from a sensor. In this tutorial we are going to discuss ATtiny85 which is a good thing that comes in small package. This microcontroller is loaded with functionality but comes in a small package as DIP8 thus enabling you to reduce your board size still offering great performance.
As a beginner tutorial we will make a LED blink using ATtiny85 Micrcontroller. This tutorial will cover

1. Features of ATtiny85
2. Core registers used in configuring ATtiny85 microcontroller

ATtiny85 :

Pinout ATtiny25/45/85



NOTE: TSSOP only for ATtiny45/V

DDRB register:

Bit	7	6	5	4	3	2	1	0	
0x17	–	–	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

This is the first register that need to be configured to use PB5 pin as GPIO. As you can see in the above register diagram, you can observe there are 5 readable as well as writeable bits in this register. Each one of these bits DDB1, DDB2…DDB5 controls the direction of physical pins PB1, PB2 ….PB5. Writing "1" to these pins will configure the GPIO pin as output and "0" will configure it as input. In our case we need to configure PB5 as output, therefore we need to write the DDB5 bit as logic "1" through code. This register is port addressable so to write 1 in DDB5 we use DDRB=0b00100000

PORTB register:

Bit	7	6	5	4	3	2	1	0	
0x18	–	–	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Now that we have set our PB5 pin as output we can proceed to next step. This PORTB register is used to toggle the output state of pins in our controller. Similar to DDRB register each bit is responsible for output of pins PB1…PB5. Our point of interest is PB5 pin therefore to make PB5 output high we write PORTB5 logic “1” and logic “0” when we need PB5 to turn low. So to make a blinker we need to write logic 0 and logic 1 alternatively with specific time delay in PORTB5 of this register. This is port addressable where to toggle PB5 to output you need to write PORTB=0b00100000.

PINB register:

Bit	7	6	5	4	3	2	1	0	
0x16	–	–	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	N/A	N/A	N/A	N/A	N/A	

The structure of this register is quite similar to the above two registers. PINB1 to PINB5 responsible for output of PB1 to PB5. You might ask why there are two registers to control output of PORTB pins. This register is bit addressable meaning you can address any single bit in this register and toggle its value without disturbing its other bits whereas above two registers are PORT or byte addressable where you cannot instruct a single bit individually. So here if you need to activate PB5 you can simply instruct the bit PINB5=1 in the above register. This value will toggle the corresponding bit of PORTB register automatically.

GPIO as input:

DDRB register:

DDRB – Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x17	–	–	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The configuration of ATtiny85 GPIO input pins starts with DDRB register, you can refer the datasheet here. We are looking to use pin 2 of attiny85 as Input. Now according to the datasheet the pin 2 is PB3 of PORTB. In order to make it as an Input we need to write the DDB3 register bit as logic ‘low’ or 0 in the code. The below piece of code will do that.

```
DDRB &= ~(1 << PB3); // Set the pin PB3 as input
```

DDRB – Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x17	–	–	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Similarly we need to define the pin 3 or PB4 as an output pin. Therefore the DDB4 register bit will be logic ‘high’ or 1.

```
DDRB |= (1 << PB4); // Set the pin PB4 as output
```

PORTB register:

Now that we have configured the pins, we need to read the logic input from the input pin PB3. For that purpose we use PORTB register. PB3 is an Input pin, therefore we can not leave the pin floating when neither high nor low logic level are connected to it. So we need to activate the internal pull up resistor of this pin. To do that,

PORTB – Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x18	–	–	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

We need to set the PORTB3 register to 1. This register is generally used to set the output state of that pin high or low, but as we declare this pin as input, this will activate the internal pull up resistor.

```
PORTB |= (1 << PB3);  //activate pull-up resistor for PB3
```

Alternatively you can use an external 10k resistor to pull-up or pull-down this pin.

PORTB – Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x18	–	–	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PORTB register also enables to toggle the LED state high or low. Writing ‘1’ or ‘0’ to PORTB4 toggles the LED to ON state or OFF state.

PINB register:

PINB register is where we can read the input of any pin that is configured as inpit. Here we will look for PINB3 bit value in this register without disturbing other bits.

PINB – Port B Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x16	–	–	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	N/A	N/A	N/A	N/A	N/A	

We do that using the below line

```
buttonState = PINB & (1 << PB3);
```

Upon reading the input state we use this input to toggle LED ON and OFF. Like this,

```
 if (buttonState) {
   PORTB |= (1 << PB4); //write to PORTB register to set the LED state to HIGH
 }
 else {
   PORTB &= ~(1 << PB4); //write to PORTB register to set the LED state to LOW
 }
```