Lab 11 - Embedded Systems Architecture

Note: Before going to embedded systems architecture, we will cover an important remaining topic from previous lab which is "Extended Addition and Subtraction".

11.1 Extended Addition and Subtraction

Extended precision addition and subtraction is the technique of adding and subtracting numbers having an almost unlimited size. In C++, no standard operator permits you to add two 1024-bit integers. But in assembly language, the ADC (add with carry) and SBB (subtract with borrow) instructions are well suited to this type of operation.

11.1.1 ADC Instruction

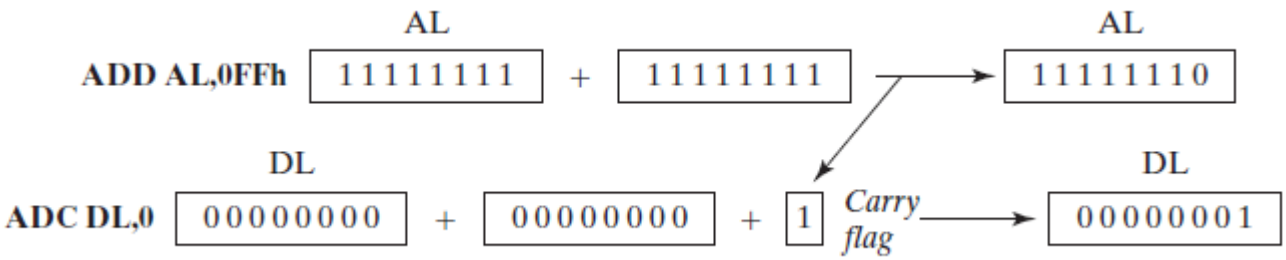
The ADC (add with carry) instruction adds both a source operand and the contents of the Carry flag to a destination operand. Syntax of ADC is:

```
ADC destination , source        ; destination = destination + source + Carry Flag
```

The instruction formats of ADC are the same as for the ADD instruction

```
ADC reg,reg
ADC mem,reg
ADC reg,mem
ADC mem,imm
ADC reg,imm
```

```
Include Irvine32.inc
.code
main PROC
      mov AL,0FFh
      add AL,0FFh        ; AL = FEh
      mov DL,0
      adc DL,0           ; DL:AL = 01FEh
exit
main ENDP
END main
```



```
Include Irvine32.inc

.data
	op1 QWORD 0A2B2A40674981234h
	op2 QWORD 8010870000234502h
	sum BYTE 9 DUP(0)		; = 0122C32B0674BB5736h
.code
main PROC
	mov ESI,OFFSET op1		; first operand
	mov EDI,OFFSET op2		; second operand
	mov EBX,OFFSET sum		; sum operand
	mov ECX,SIZEOF op1		; number of bytes
	call Extended_Add

; Displaying the sum
	mov ESI,OFFSET sum
	mov ECX,LENGTHOF sum
	call Display_Sum
	call Crlf

	exit
main ENDP

;-----------------------------------------------------
Extended_Add PROC
;
; Calculates the sum of two extended integers stored
; as arrays of bytes.
; Receives: ESI and EDI point to the two integers,
; EBX points to a variable that will hold the sum, and
; ECX indicates the number of bytes to be added.
; Storage for the sum must be one byte longer than the
; input operands.
; Returns: nothing
;-----------------------------------------------------
	pushad
	clc				; clear the Carry flag
L1:
	mov	AL,[ESI]		; get the first integer
	adc	AL,[EDI]		; add the second integer
	pushfd				; save the Carry flag
	mov	[EBX],AL		; store partial sum
	add	ESI,1			; advance all 3 pointers
	add	EDI,1
	add	EBX,1
	popfd				; restore the Carry flag
LOOP L1					; repeat the loop

	mov	BYTE PTR [EBX],0	; clear high byte of sum
	adc	BYTE PTR [EBX],0	; add any leftover carry
	popad
	ret
Extended_Add ENDP
```

```
;-----------------------------------------------------------
Display_Sum PROC
;
; Displays a large integer that is stored in little endian
; order (LSB to MSB). The output displays the array in
; hexadecimal, starting with the most significant byte.
; Receives: ESI points to the array, ECX is the array size
; Returns: nothing
;-----------------------------------------------------------
	pushad

	; point to the last array element
	add ESI,ECX
	sub ESI,TYPE BYTE
	mov EBX,TYPE BYTE
L1:
	mov AL,[ESI]		; get an array byte
	call WriteHexB		; display it
	sub ESI,TYPE BYTE	; point to previous byte
LOOP L1

	popad
	ret
Display_Sum ENDP
END main
```

11.1.2 SBB Instruction

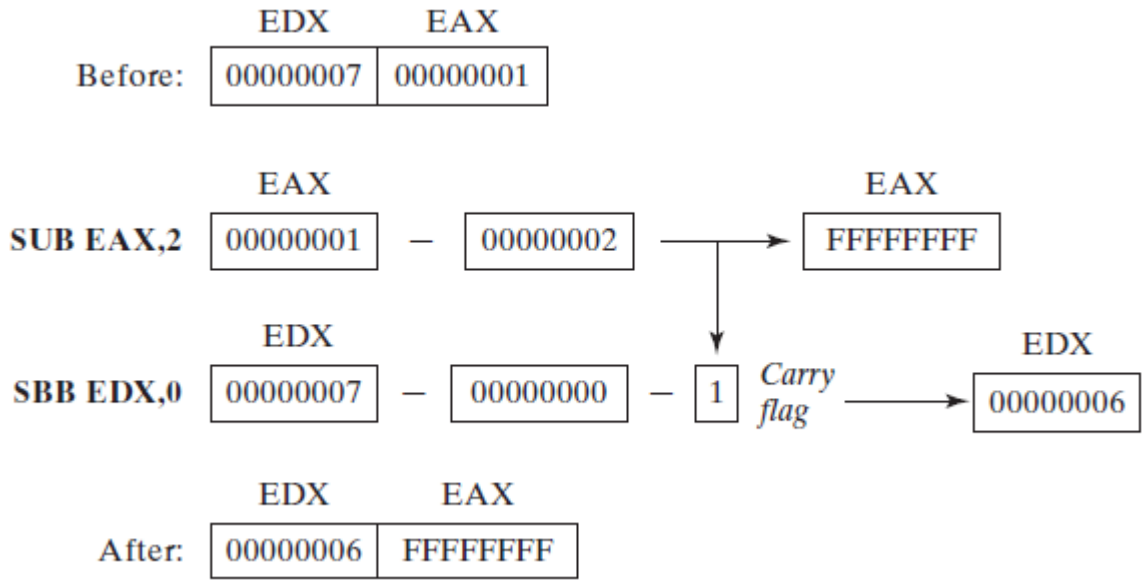
The SBB (subtract with borrow) instruction subtracts both a source operand and the value of the Carry flag from a destination operand. Syntax of ADC is:

```
	SBB destination , source		; destination = destination – source – Carry Flag

	Include Irvine32.inc
	.code
	main PROC
		mov EAX,0

		mov EDX,7	; upper half
		mov EAX,1	; lower half
		sub EAX,2	; subtract 2
		sbb EDX,0	; subtract upper half

		mov EAX,EDX
		call WriteDec
		call Crlf
		exit
	main ENDP
	END main
```

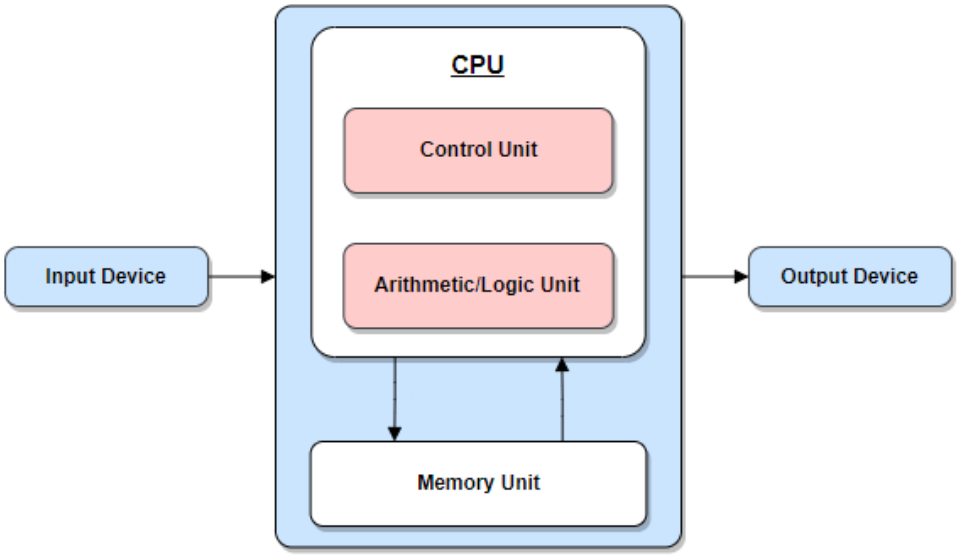


11.2 Computer Architectures

11.2.1 Von-Neumann Architecture

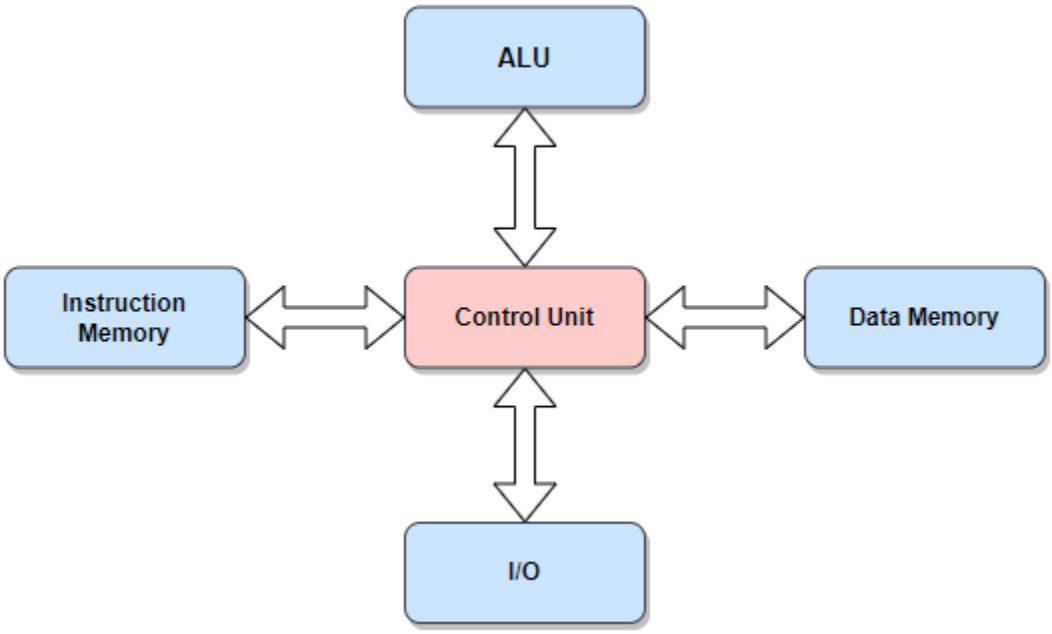
Until now we have worked on x86 Intel processor which is based on the "Von-Neumann Architecture". It is also known as "Princeton Architecture". It renders a unique design for the electronic digital systems having the following components:

- A Central Processing Unit (CPU) with arithmetic and logic unit (ALU) and registers.
- A memory that can store data and instructions.
- External mass storage or secondary storage.
- A Control Unit (CU) with the ability to hold instructions in the program counter (PC) or instruction register (IR).
- Input and output mechanisms and peripherals.



11.2.2 Harvard Architecture

Harvard Architecture consists of code and data stored in distinct memory sections. It requires a separate memory block for data and instruction. It has solely contained data storage within the Central Processing Unit (CPU). A single collection of clock cycles is needed. Data accessibility in one memory is done by a single memory location in the case of Harvard architecture.



11.2.3 Modified Harvard Architecture

The modified Harvard architecture is a variation of the Harvard computer architecture that allows the contents of the instruction memory to be accessed as data. Most modern computers that are documented as Harvard architecture are, in fact, modified Harvard architecture.

Those modifications are various ways to loosen the strict separation between code and data, while still supporting the higher performance concurrent data and instruction access of the Harvard architecture.

Its applications are:

- Cache-less DSP (Digital Signal Processor)
- Microcontrollers (AVR and ARM)
- Most modern processors have a CPU cache which partitions instruction and data

There are also processors which are Harvard machines by the most rigorous definition (that program and data memory occupy different address spaces) and are only modified in the weak sense that there are operations to read and/or write program memory as data. For example, LPM (Load Program Memory) and SPM (Store Program Memory) instructions in the **Atmel AVR** implement such a modification. Similar solutions are found in other microcontrollers such as the **PIC** and **Z8Encore!**.

11.3 Difference between Von-Neumann and Harvard Architecture

Parameters	Von Neumann Architecture	Harvard Architecture
Definition	The Von Neumann Architecture is an ancient type of computer architecture that follows the concept of a stored-program computer.	Harvard Architecture is a modern type of computer architecture that follows the concept of the relay-based model by Harvard Mark I.
Physical Address	It uses one single physical address for accessing and storing both data and instructions.	It uses two separate physical addresses for storing and accessing both instructions and data.
Buses	One common signal path (bus) helps in the transfer of both instruction and data.	It uses separate buses for the transfer of both data and instructions.
Number of Cycles	It requires two clock cycles for executing a single instruction.	It executes any instruction using only one single cycle.
Cost	It is comparatively cheaper in cost than Harvard Architecture.	It is comparatively more expensive than the Von Neumann Architecture.
Access to CPU	The CPU is not able to read/write data and access instructions at the same time.	The CPU can easily read/write data as well as access the instructions at any given time.
Uses	This method comes to play in the case of small computers and personal computers.	This architecture is best for signal processing as well as microcontrollers.
Requirement of Hardware	Von Neumann Architecture requires lesser hardware. It is because it only needs to reach one common memory.	This one requires more hardware. It is because it requires separate sets of data as well as address buses for individual memory.
Requirement of Space	This architecture basically requires less space.	This architecture comparatively requires more space.
Usage of Space	This architecture does not waste any space. It is because the instruction memory can utilize the left space of the data memory. It can also happen vice-versa.	This type of architecture can result in space wastage. It is because the instruction memory cannot utilize the leftover space in the data memory. It also cannot happen vice-versa.
Execution Speed	The speed of execution of the Von Neumann Architecture is comparatively slower. It is because it is not capable of fetching the instructions and data both at the same time.	The overall speed of execution of Harvard Architecture is comparatively faster. It is because the processor can fetch both instructions and data at the very same time.
Controlling	The process of controlling becomes comparatively simpler with this architecture. It is because it fetches either instructions or data at any given time.	The process of controlling becomes comparatively complex with this architecture. It is because it basically fetches both instructions and data simultaneously at the very same time.

11.4 Subcategories of Computer Architecture

The discipline of computer architecture has three main subcategories:

11.4.1 Instruction Set Architecture (ISA)

It defines the machine code that a processor reads and acts upon as well as the word size, memory address modes, processor registers, and data type. A device that executes instructions described by that ISA, such as a central processing unit (CPU), is called an *implementation*.

11.4.1.1 Classification of ISAs

A common classification of ISA is by its architectural complexity. These are some types of ISA:

Complex Instruction Set Computer (CISC): It has many specialized instructions, some of which may only be rarely used in practical programs. The primary goal of CISC architecture is to complete a task in as few lines of assembly as possible. This is achieved by building processor hardware that is capable of understanding and executing a series of operations.

Reduced Instruction Set Computer (RISC): It simplifies the processor by efficiently implementing only the instructions that are frequently used in programs, while the fewer common operations are implemented as subroutines, having their resulting additional processor execution time offset by infrequent use.

Some other types include:

- Very Long Instruction Word (VLIW) Architectures
- Closely Related Long Instruction Word (LIW) Architecture
- Explicitly Parallel Instruction Computing (EPIC) Architecture

These architectures seek to exploit instruction-level parallelism with less hardware than RISC and CISC by making the compiler responsible for instruction issue and scheduling.

11.4.2 Microarchitecture

It is also known as "Computer Organization", this describes how a particular processor will implement the ISA. The size of a computer's CPU cache for instance, is an issue that generally has nothing to do with the ISA. Simply, it is a logical form of all electronic elements and data pathways present in the microprocessor, designed in a specific way. It allows for the optimal completion of instructions.

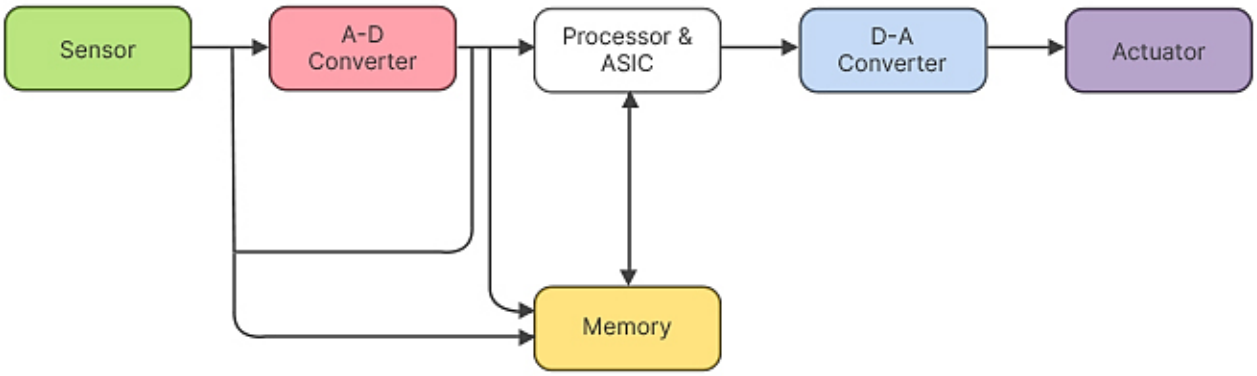
11.4.3 Systems Design

It includes all the other hardware components within a computing system, such as data processing other than the CPU (e.g., direct memory access), virtualization, and multiprocessing.

11.5 Architecture of an Embedded System

Typical embedded system mainly has two parts, embedded hardware and embedded software. Embedded hardware is based around microprocessors and microcontrollers, also include memory, bus, Input/Output, Controller, whereas embedded software includes embedded operating systems, different applications and device drivers

Architecture of the Embedded System includes Sensor, Analog to Digital Converter, Memory, Processor, Digital to Analog Converter, and Actuators etc.



11.5.1 Important Characteristics of an Embedded System:

- **Low Cost:** The price of an embedded system is not so expensive.
- **Time Specific:** It performs the tasks within a certain time frame.
- **Low Power:** Embedded Systems don't require much power to operate.
- **High Efficiency:** The efficiency level of embedded systems is so high.
- **Minimal User interface:** These systems require less user interface and easy to use.
- **Less Human intervention:** Embedded systems require no or very less human intervention.
- **Highly Stable:** Embedded systems do not change frequently mostly fixed maintaining stability.
- **High Reliability:** Embedded systems are reliable they perform the tasks consistently well.
- **Manufacturable:** Most embedded systems are compact and affordable to manufacture. They are based on the size and low complexity of the hardware.

11.6 Microcontroller

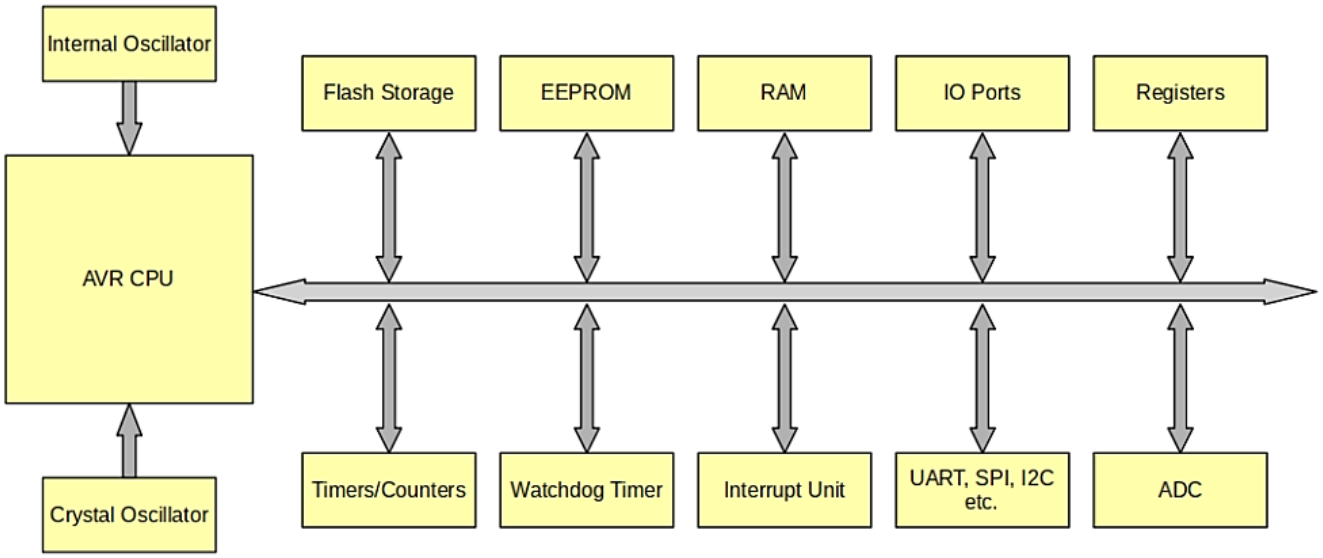
A micro-controller can be comparable to a little standalone computer; it is an extremely powerful device, which is able of executing a series of pre-programmed tasks and interacting with extra hardware devices. A microcontroller contains one or more CPUs (processor cores) along with memory and programmable input/output peripherals.

In modern terminology, a microcontroller is similar to, but less sophisticated than, a **system on a chip (SoC)**. An SoC may connect the external microcontroller chips as the motherboard components, but an SoC usually integrates the advanced peripherals like graphics processing unit (GPU) and Wi-Fi interface controller as its internal microcontroller unit circuits.

Some microcontrollers may use four-bit words and operate at frequencies as low as 4 kHz for low power consumption (single-digit milliwatts or microwatts). They generally have the ability to retain functionality while waiting for an event such as a button press or other interrupt; power consumption while sleeping (CPU clock and most peripherals off) may be just nanowatts, making many of them well suited for long lasting battery applications. Other microcontrollers may serve performance-critical roles, where they may need to act more like a digital signal processor (DSP), with higher clock speeds and power consumption.

A microcontroller can be considered a self-contained system with a processor, memory and peripherals and can be used as an embedded system. The majority of microcontrollers in use today are embedded in other machinery, such as automobiles, telephones, appliances, and peripherals for computer systems.

Typical input and output devices include switches, relays, solenoids, LED's, small or custom liquid-crystal displays, radio frequency devices, and sensors for data such as temperature, humidity, light level etc. Embedded systems usually have no keyboard, screen, disks, printers, or other recognizable I/O devices of a personal computer and may lack human interaction devices of any kind.



11.7 Features of Microcontroller

11.7.1 Central Processing Unit

The CPU performs arithmetic operations, manages data flow, and generates control signals in accordance with the sequence of instructions created by the programmer. The extremely complex circuitry required for CPU functionality is not visible to the designer.

11.7.2 Memory

Microcontrollers have both nonvolatile and volatile memories.

Nonvolatile memory: is used to store the microcontroller's program. You will typically see the word "Flash".

Volatile memory: (i.e., RAM) is used for temporary data storage. This data is lost when the microcontroller loses power. Internal registers also provide temporary data storage, but we don't think of these as a separate functional block because they are integrated into the CPU.

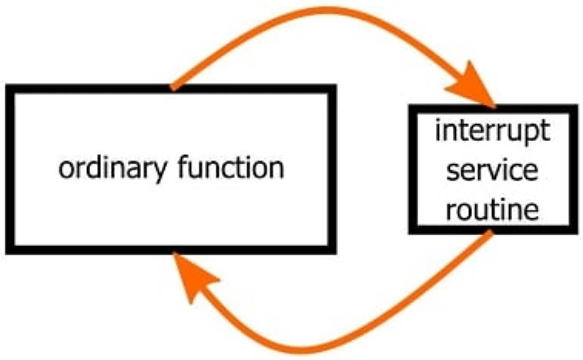
11.7.3 Peripherals

We use the word "peripheral" to describe the hardware modules that help a microcontroller to interact with the external system. The following points identify the various categories of peripherals:

- **Data Converters:** Analog-to-digital converter, digital-to-analog converter, reference-voltage generator.
- **Clock Generation:** Internal oscillator, crystal-drive circuitry, phase-locked loop.
- **Timing Operations:** General-purpose timer, real-time clock, external-event counter, pulse-with modulation.
- **Analog signal processing:** Operational amplifier, analog comparator.
- **Input/Output:** General-purpose digital input and output circuitry, parallel memory interface.
- **Serial Communication:** UART, SPI, I2C, USB

11.7.4 Interrupts

Interrupts are an extremely valuable aspect of microcontroller functionality. Interrupts are generated by external or internal hardware-based events, and they cause the processor to immediately respond to these events by executing a specific group of instructions.



11.8 Difference between AVR, ARM, 8051 and PIC Microcontrollers

11.8.1 8051 Microcontroller

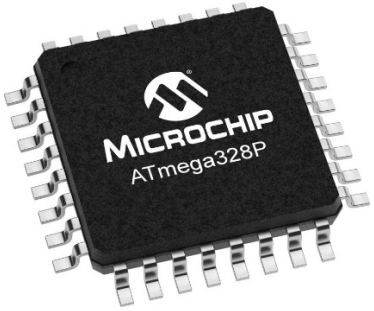
8051 microcontroller is an 8-bit family of microcontroller is developed by the Intel in the year 1981. This is one of the popular families of microcontroller are being used all across the world. It has 128 bytes of RAM, 4Kbytes of a ROM, 2 timers, 1 serial port, and 4 ports on a single chip.

11.8.2 PIC Microcontroller

Peripheral Interface Controller (PIC) is microcontroller developed by a Microchip, PIC microcontroller is fast and simple to implement program when we contrast other microcontrollers like 8051. The ease of programming and simple to interfacing with other peripherals PIC become successful microcontroller. PIC mostly used to modify Harvard architecture and also supports RISC (Reduced Instruction Set Computer).

11.8.3 AVR Microcontroller

AVR microcontroller was developed in the year of 1996 by Atmel Corporation. The structural design of AVR was developed by the Alf-Egil Bogen and Vegard Wollan. AVR derives its name from its developers and stands for Alf Egil Bogen Vegard Wollan RISC microcontroller, also known as Advanced Virtual RISC. The AT90S8515 was the initial microcontroller, which was based on the AVR architecture, though the first microcontroller to hit the commercial market was AT90S1200 in the year 1997.

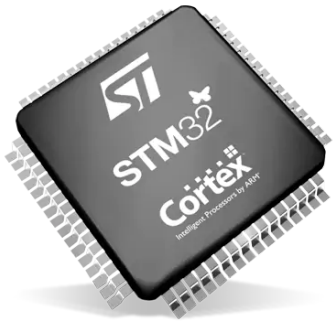


AVR Microcontrollers are available in three categories

- **TinyAVR:** Less memory, small size, appropriate just for simpler applications.
- **MegaAVR:** These are the mainly popular ones having a good quantity of memory (up to 256 KB), higher number of inbuilt peripherals and appropriate for modest to complex applications.
- **XmegaAVR:** Used in commercial for complex applications, which need large program memory and high speed.

11.8.4 ARM Processor

An ARM processor is also one of a family of CPUs based on the RISC (reduced instruction set computer) architecture developed by Advanced RISC Machines (ARM).



An ARM makes at 32-bit and 64-bit RISC multi-core processors. RISC processors are designed to perform a smaller number of types of computer instructions so that they can operate at a higher speed, performing extra millions of instructions per second (MIPS).

ARM processors are widely used in customer electronic devices such as smartphones, laptops, tablets, smartwatches, multimedia players and other mobile devices.

Note: “Qualcomm Snapdragon”, “MediaTek” and “Apple Silicon” SoC processors are also ARM based.

	8051	PIC	AVR	ARM
Bus width	8-bit **for stand**ard 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	Harvard architecture	Von Neumann 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	Very Low	Average	Average	Low
Other Feature	Known for its Standard	Cheap	Cheap, effective	High speed operation Vast
Popular Microcontrollers	AT89C51, P89v51, etc.	PIC18fXX8, PIC16f88X, PIC32MXX	Atmega8, 16, 32, Arduino Community	LPC2148, ARM Cortex-M0 to ARM Cortex-M7, etc.

11.9 Microcontroller's Memory Types

Writing assembly programs requires an understanding of the AVR memory types and their uses. This topic covers a generic overview, but you will need to consult the datasheet for your microcontroller to get specific values.

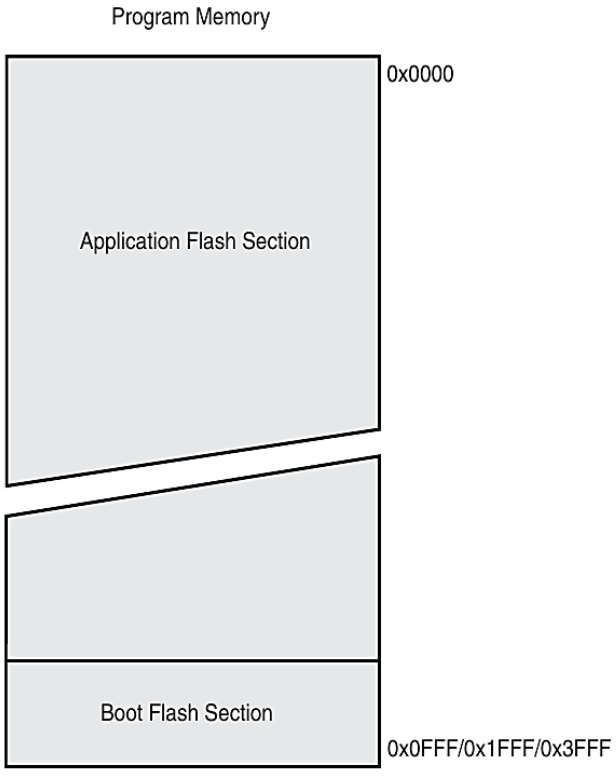
There are three types of memory in the AVR architecture:

- Program Memory
- Data Memory
- EEPROM

11.9.1 Program Memory

Program Memory, also referred to as flash, is where your code is stored. Depending on settings in the fuse bits, the entire flash memory can be used as application storage, or a portion can be code-blocked off for a bootloader program.

Flash memory is non-volatile meaning its data does not go away when the microcontroller loses power. Flash cannot be accessed as quickly as Data Memory and individual bytes cannot be modified at a time, so it is not a good place to store variables which constantly change. However, in addition to storing your application code, it is a good place to put constants like lookup tables or strings.



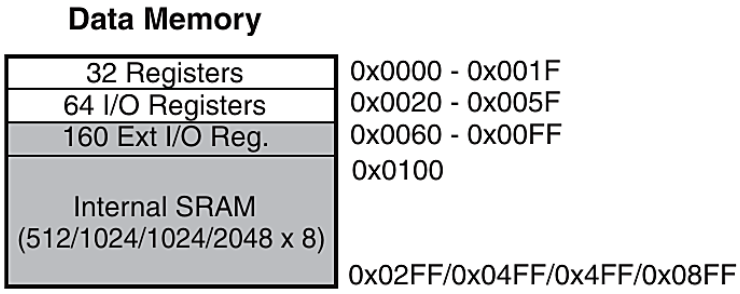
11.9.2 Data Memory

Data Memory is composed of four parts:

- Register File
- I/O Registers
- Extended I/O Registers
- Internal SRAM

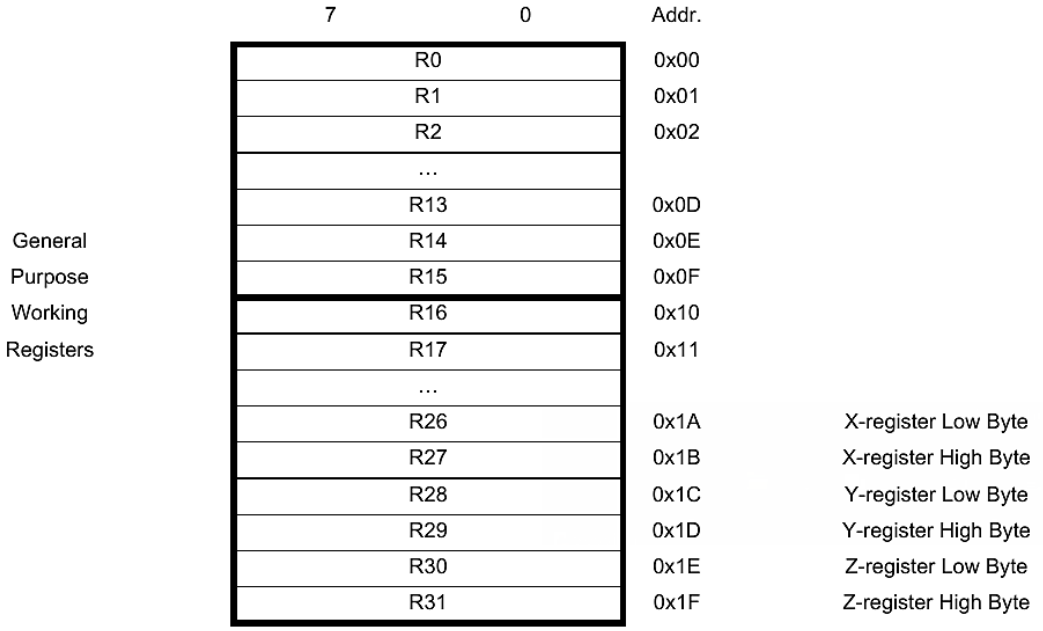
Unlike flash, Data Memory is not composed of a single memory type. Rather, it is a contiguous mapping of addresses across multiple memory types.

The breakdown of data memory is shown below. The location and size of the general-purpose working registers, I/O registers, and extended I/O registers are the same across all chips (even if some of the extended I/O features are not implemented). However, the size of internal SRAM varies between different models.



11.9.2.1 Register File

It is volatile memory. The register file begins at the start of Data Memory and contains 32 general purpose working registers that are directly connected to the arithmetic logic unit (ALU). The general-purpose working registers are used as operands for most instructions. The general-purpose working registers are extremely important and useful. Any data manipulation in the AVR requires using one of these registers.



Registers R26:R27, R28:R29, and R30:R31 are given the names X, Y, and Z respectively. They are often referred to as the X, Y and Z pointers as they can be used to store or retrieve data from memory by placing an address in them.

11.9.2.2 I/O Registers

The I/O Registers are locations in Data Memory which control certain microcontroller functions. All the data direction registers and ports (e.g., DDRB and PORTB) are located in the I/O Register portion of Data Memory. Because they are needed so often, there are special instructions which allow quick access and modification of the I/O Registers.

The I/O Registers are non-volatile. They will initialize to default values that may or may not need to be changed by your program.

11.9.2.3 Extended I/O Registers

Most microcontrollers have more functions and peripherals than can be supported by the 64 bytes allocated to the I/O Registers. Thus, there is an Extended I/O Register section. The Extended I/O section only differs from the regular I/O section in that does not support all of the same instructions to set, clear, or copy its contents. However, its function and purpose are exactly the same.

Like the regular I/O Registers, the Extended I/O Registers are non-volatile.

11.9.2.4 Internal SRAM

It is volatile memory. Static Random Access Memory - referred to as SRAM - is the last part of Data Memory. SRAM is volatile so its contents cannot be guaranteed at startup. However, unlike flash, single bytes can be written to SRAM and its access time is quicker, making it much better for storing temporary values.

SRAM is a very convenient runtime memory type. It is great when you have more variables than can fit in the general-purpose working registers and need a place to put them while they are not being used.

11.9.3 EEPROM

EEPROM is another non-volatile storage location in the AVR architecture. Unlike flash, individual bytes may be written to it, although its access time is much slower. EEPROM is best served for configuration settings, i.e., things that need to be available at startup but may be changed at runtime and used the next time the chip starts up.