Differences between a Microprocessor and a Microcontroller

Himadri Barman

A microprocessor (abbreviated as µP or uP) is a computer electronic component made from miniaturized transistors and other circuit elements on a single semiconductor integrated circuit (IC) (microchip or just chip). The central processing unit (CPU) is the most well known microprocessor, but many other components in a computer have them, such as the Graphics Processing Unit (GPU) on a video card. In the world of personal computers, the terms microprocessor and CPU are used interchangeably. At the heart of all personal computers and most workstations sits a microprocessor. Microprocessors also control the logic of almost all digital devices, from clock radios to fuel-injection systems for automobiles.

Microcontroller is a computer-on-a-chip optimised to control electronic devices. It is designed specifically for specific tasks such as controling a specific system. A microcontroller (sometimes abbreviated µC, uC or MCU) is basically a specialized form of microprocessor that is designed to be self-sufficient and cost-effective. Also, a microcontroller is part of an embedded system, which is essentially the whole circuit board.
An embedded system is a computer system designed to perform one or a few dedicated functions often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts.

Examples of microcontrollers are Microchip's PIC, the 8051, Intel's 80196, and Motorola's 68HCxx series. Microcontrollers which are frequently found in automobiles, office machines, toys, and appliances are devices which integrate a number of components of a microprocessor system onto a single microchip:

- The CPU core (microprocessor)
- Memory (both ROM and RAM)
- Some parallel digital I/O

The microcontroller sees the integration of a number of useful functions into a single IC package. These functions are:

- The ability to execute a stored set of instructions to carry out user defined tasks.
- The ability to be able to access external memory chips to both read and write data from and to the memory.

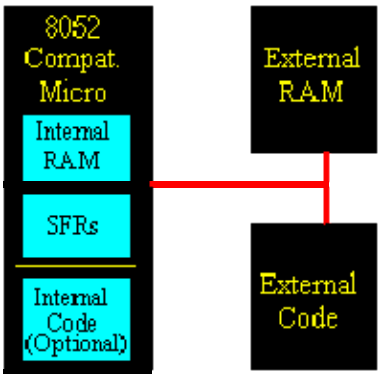
The difference between the two is that a microcontroller incorporates features of microprocessor (CPU, ALU, Registers) along with the presence of added features like presence of RAM, ROM, I/O ports, counter, etc. Here a microcontroller controls the operation of a machine using fixed programs stored in ROM that doesn't change with lifetime.

From another view point, the main difference between a typical microprocessor and a micro controller leaving there architectural specifications is the application area of both the devices. Typical microprocessors like the Intel Core family or Pentium family processors or similar processors are in computers as a general purpose programmable device. In its life

period it has to handle many different tasks and programs given to it. On the other hand a micro controllers from 8051 family or PIC family or any other have found there applications in small embedded systems like some kind of robotic system or a traffic signal control system. Also these devices handle same task or same program during there complete life cycle. (Best example is of traffic signal control system).The other difference is that the micro controllers usually has to handle real time tasks while on the contrary the microprocessors in a computer system may not handle a real time task at all times.

A Dossier on 8051 Memory

The 8051 has three very general types of memory. The memory types are illustrated in the following graphic. They are: On-Chip Memory, External Code Memory, and External RAM.



On-Chip Memory refers to any memory (Code, RAM, or other) that physically exists on the microcontroller itself. On-chip memory can be of several types, but we'll get into that shortly.

External Code Memory is code (or program) memory that resides off-chip. This is often in the form of an external EPROM.

External RAM is RAM memory that resides off-chip. This is often in the form of standard static RAM or flash RAM.

Code Memory

Code memory is the memory that holds the actual 8051 program that is to be run. This memory is limited to 64K and comes in many shapes and sizes: Code memory may be found *on-chip*, either burned into the microcontroller as ROM or EPROM. Code may also be stored completely *off-chip* in an external ROM or, more commonly, an external EPROM. Flash RAM is also another popular method of storing a program. Various combinations of these memory types may also be used--that is to say, it is possible to have 4K of code memory *on-chip* and 64k of code memory *off-chip* in an EPROM.

When the program is stored on-chip the 64K maximum is often reduced to 4k, 8k, or 16k. This varies depending on the version of the chip that is being used. Each version offers

specific capabilities and one of the distinguishing factors from chip to chip is how much ROM/EPROM space the chip has.

However, code memory is most commonly implemented as off-chip EPROM. This is especially true in low-cost development systems and in systems developed by students.

External RAM

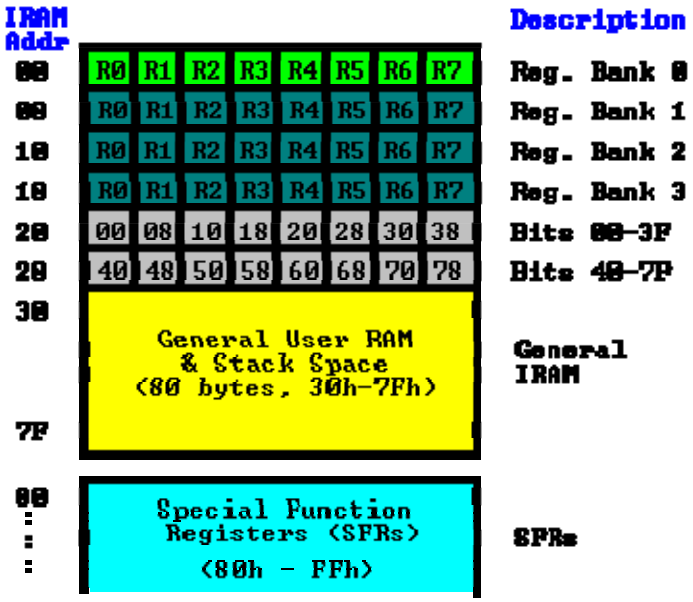
As an obvious opposite of *Internal RAM*, the 8051 also supports what is called *External RAM*.

As the name suggests, External RAM is any random access memory which is found *off-chip*. Since the memory is off-chip it is not as flexible in terms of accessing, and is also slower. For example, to increment an Internal RAM location by 1 requires only 1 instruction and 1 instruction cycle. To increment a 1-byte value stored in External RAM requires 4 instructions and 7 instruction cycles. In this case, external memory is 7 times slower!

What External RAM loses in speed and flexibility it gains in quantity. While Internal RAM is limited to 128 bytes (256 bytes with an 8052), the 8051 supports External RAM up to 64K.

On-Chip Memory

As mentioned at the beginning of this chapter, the 8051 includes a certain amount of on-chip memory. On-chip memory is really one of two types: Internal RAM and Special Function Register (SFR) memory. The layout of the 8051's internal memory is presented in the following memory map:



As is illustrated in this map, the 8051 has a bank of 128 bytes of *Internal RAM*. This Internal RAM is found *on-chip* on the 8051 so it is the fastest RAM available, and it is also the most flexible in terms of reading, writing, and modifying its contents. Internal RAM is volatile, so when the 8051 is reset, this memory is cleared.

The 128 bytes of internal ram is subdivided as shown on the memory map. The first 8 bytes (00h - 07h) are "register bank 0". By manipulating certain SFRs, a program may choose to use register banks 1, 2, or 3. These alternative register banks are located in internal RAM in addresses 08h through 1Fh. They "live" and are part of internal RAM.

Bit Memory also lives and is part of internal RAM. Keep in mind that bit memory actually resides in internal RAM, from addresses 20h through 2Fh.

The 80 bytes remaining of Internal RAM, from addresses 30h through 7Fh, may be used by user variables that need to be accessed frequently or at high-speed. This area is also utilized by the microcontroller as a storage area for the operating *stack*. This fact severely limits the 8051s stack since, as illustrated in the memory map, the area reserved for the stack is only 80 bytes--and usually it is less since this 80 bytes has to be shared between the stack and user variables.

Register Banks

The 8051 uses 8 "R" registers which are used in many of its instructions. These "R" registers are numbered from 0 through 7 (R0, R1, R2, R3, R4, R5, R6, and R7). These registers are generally used to assist in manipulating values and moving data from one memory location to another. For example, to add the value of R4 to the Accumulator, we would execute the following instruction:

ADD A,R4

Thus if the Accumulator (A) contained the value 6 and R4 contained the value 3, the Accumulator would contain the value 9 after this instruction was executed.

However, as the memory map shows, the "R" Register R4 is really part of Internal RAM. Specifically, R4 is address 04h. This can be see in the bright green section of the memory map. Thus the above instruction accomplishes the same thing as the following operation:

ADD A,04h

This instruction adds the value found in Internal RAM address 04h to the value of the Accumulator, leaving the result in the Accumulator. Since R4 is really Internal RAM 04h, the above instruction effectively accomplished the same thing.

But watch out! As the memory map shows, the 8051 has four distinct register banks. When the 8051 is first booted up, register bank 0 (addresses 00h through 07h) is used by default. However, your program may instruct the 8051 to use one of the alternate register banks; i.e., register banks 1, 2, or 3. In this case, R4 will no longer be the same as Internal RAM address 04h. For example, if your program instructs the 8051 to use register bank 3, "R" register R4 will now be synonymous with Internal RAM address 1Ch. We can switch to other banks by use of the PSW (program status word) register. **[This is the fundamental concept of Switching of Register Banks]**

The concept of register banks adds a great level of flexibility to the 8051, especially when dealing with interrupts. However, always remember that the register banks really reside in the first 32 bytes of Internal RAM.

Bit Memory

The 8051, being a communications-oriented microcontroller, gives the user the ability to access a number of *bit variables*. These variables may be either 1 or 0.

There are 128 bit variables available to the user, numberd 00h through 7Fh. The user may make use of these variables with commands such as SETB and CLR. For example, to set bit number 24 (hex) to 1 you would execute the instruction:

```
SETB 24h
```

It is important to note that Bit Memory is really a part of Internal RAM. In fact, the 128 bit variables occupy the 16 bytes of Internal RAM from 20h through 2Fh. Thus, if you write the value FFh to Internal RAM address 20h youve effectively set bits 00h through 07h. That is to say that:

```
MOV 20h,#0FFh
```

is equivalent to:

```
SETB 00h
SETB 01h
SETB 02h
SETB 03h
SETB 04h
SETB 05h
SETB 06h
SETB 07h
```

As illustrated above, bit memory isn't really a new type of memory. Its really just a subset of Internal RAM. But since the 8051 provides special instructions to access these 16 bytes of memory on a bit by bit basis it is useful to think of it as a separate type of memory. However, always keep in mind that it is just a subset of Internal RAM--and that operations performed on Internal RAM can change the values of the bit variables.

Bit variables 00h through 7Fh are for user-defined functions in their programs. However, bit variables 80h and above are actually used to access certain SFRs on a bit-by-bit basis. For example, if output lines P0.0 through P0.7 are all clear (0) and you want to turn on the P0.0 output line you may either execute:

```
MOV P0,#01h
```

or you may execute:

```
SETB 80h
```

Both these instructions accomplish the same thing. However, using the SETB command will turn on the P0.0 line without effecting the status of any of the other P0 output lines. The MOV command effectively turns off all the other output lines which, in some cases, may not be acceptable.

Special Function Register (SFR) Memory

Special Function Registers (SFRs) are areas of memory that control specific functionality of the 8051 processor. For example, four SFRs permit access to the 8051s 32 input/output lines. Another SFR allows a program to read or write to the 8051s serial port. Other SFRs allow the user to set the serial baud rate, control and access timers, and configure the 8051s interrupt system.

When programming, SFRs have the illusion of being Internal Memory. For example, if you want to write the value "1" to Internal RAM location 50 hex you would execute the instruction:

MOV 50h,#01h

Similarly, if you want to write the value "1" to the 8051s serial port you would write this value to the **SBUF**[1] SFR, which has an SFR address of 99 Hex. Thus, to write the value "1" to the serial port you would execute the instruction:

MOV 99h,#01h

As you can see, it appears that the SFR is part of Internal Memory. This is not the case. When using this method of memory access (its called direct address), any instruction that has an address of 00h through 7Fh refers to an Internal RAM memory address; any instruction with an address of 80h through FFh refers to an SFR control register.

[1] SBUF: Serial Data Buffer Register. May also mean Serial Data Buffer which is an assembly language directive You may also come across SCON which means Serial Port Control Register

