MEMORY ORGANIZATION OF 8051

8051 operates with 4 different memories:

Internal ROM
External ROM

Internal RAM
External RAM

Being based on **Harvard Model**, 8051 stores **programs and data in separate memory spaces**.
Programs are stored in ROM, whereas data is stored in RAM.

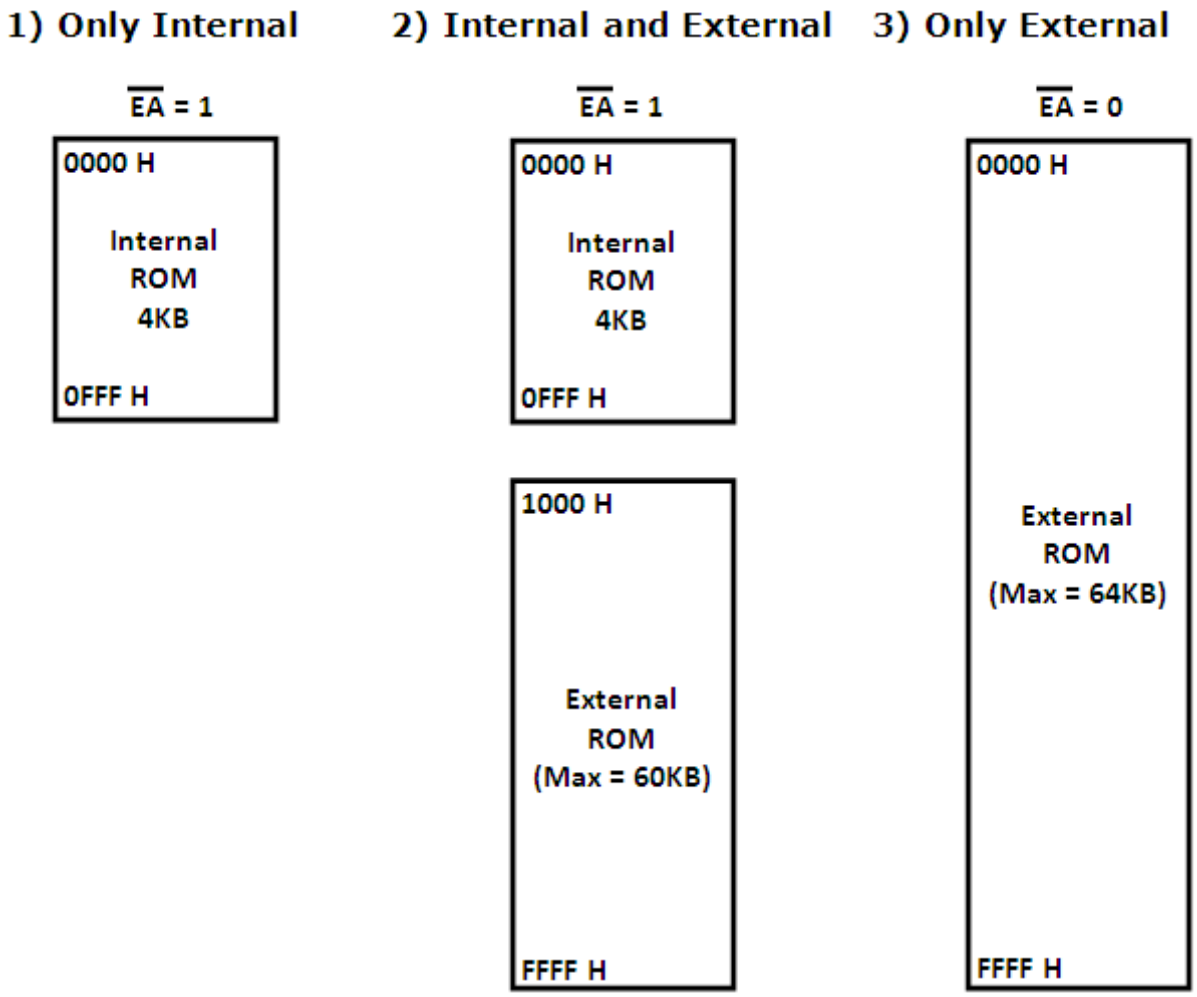
Microcontrollers are used in **appliances**.
Washing machines, remote controllers, microwave ovens are some of the examples.
Here **programs are generally permanent** in nature and very rarely need to be modified.
Moreover, the programs must be **retained** even after the device is completely **switched off**.
Hence programs are stored in permanent (non-volatile) memory like ROM.

Data on the other hand is continuously **changed at runtime**.
For example current temperature, cooking time etc. in an oven.
Such data is not permanent in nature and will certainly be modified in every usage of the device.
Hence Data is stored in writeable memory like RAM.

However, sometimes there is **permanent data**, such as ASCII codes or 7-segment display codes.
Such data is stored in **ROM**, in the form of **Look up tables** and is accessed using a dedicated addressing mode called **Indexed** Addressing mode. We will discover this in more depth in further topics.

We are now going to take a closer look at all four memories.

ROM ORGANIZATION / CODE MEMORY / PROGRAM MEMORY



We can implement ROM in three different ways in 8051.

ONLY INTERNAL ROM

8051 has 4 KB internal ROM.
In many cases this size is sufficient and there is no need for connecting External ROM.
Such systems use only Internal ROM of 8051.
All addresses from 0000H… 0FFFH will be accessed from Internal ROM.
Any address beyond that will be invalid.
In such systems **$\overline{EA}$** will be "1" as Internal ROM is being used.

(PS: Read the whole answer to understand $\overline{EA}$ clearly)

INTERNAL AND EXTERNAL ROM

8051 has 4 KB internal ROM.
In many cases this size is may be insufficient and we may need to add some External ROM.
Such systems use a combination of Internal ROM and External ROM.
The **"total"** ROM that can be accessed is 64 KB.
Since we are using the Internal ROM of 4 KB, the maximum amount of External ROM that can be connected is 60 KB.
All addresses from 0000H… 0FFFH will be accessed from Internal ROM.
Addresses 1000H… FFFH will be accessed from External ROM.
In such systems **$\overline{EA}$** will be "1" as Internal ROM is being used.

(PS: Read the whole answer to understand $\overline{EA}$ clearly)

ONLY EXTERNAL ROM

This is the most interesting case.
Though 8051 has 4 KB of Internal ROM, the user may choose the discard it and connect only External ROM.
This may happen due to several reasons.
The program stored in the Internal ROM may have become invalid or outdated, or the system may need to be upgraded etc.
Such systems use only External ROM, and the Internal ROM is discarded.
Here we can connect up to 64 KB of External ROM.
All addresses from 0000H… FFFFH will be accessed from External ROM.
But do keep in mind, that the Internal ROM is still present in 8051.
We need to clearly indicate to 8051 that the Internal ROM must be ignored and every address from 0000H… FFFFH must be accessed externally. This is indicated by us to 8051 using **$\overline{EA}$.**

By making **$\overline{EA}$ = 0**, we inform 8051 that the Internal ROM must be discarded and all ROM must be accessed externally.

Note: Use of $\overline{EA}$ pin of 8051.

The $\overline{EA}$ pin of 8051 decides whether the Internal ROM will be used or not.

If the Internal ROM has to be used we must make **$\overline{EA}$** = 1.

Now 8051 will Access the internal ROM for all addresses from 0000H to 0FFFH and will only access external ROM for addresses 1000H and beyond.

But if **$\overline{EA}$** = 0, then the Internal ROM is completely discarded.

Now 8051 will access the External ROM for all addresses from 0000H to FFFFH, hence discarding the internal ROM.

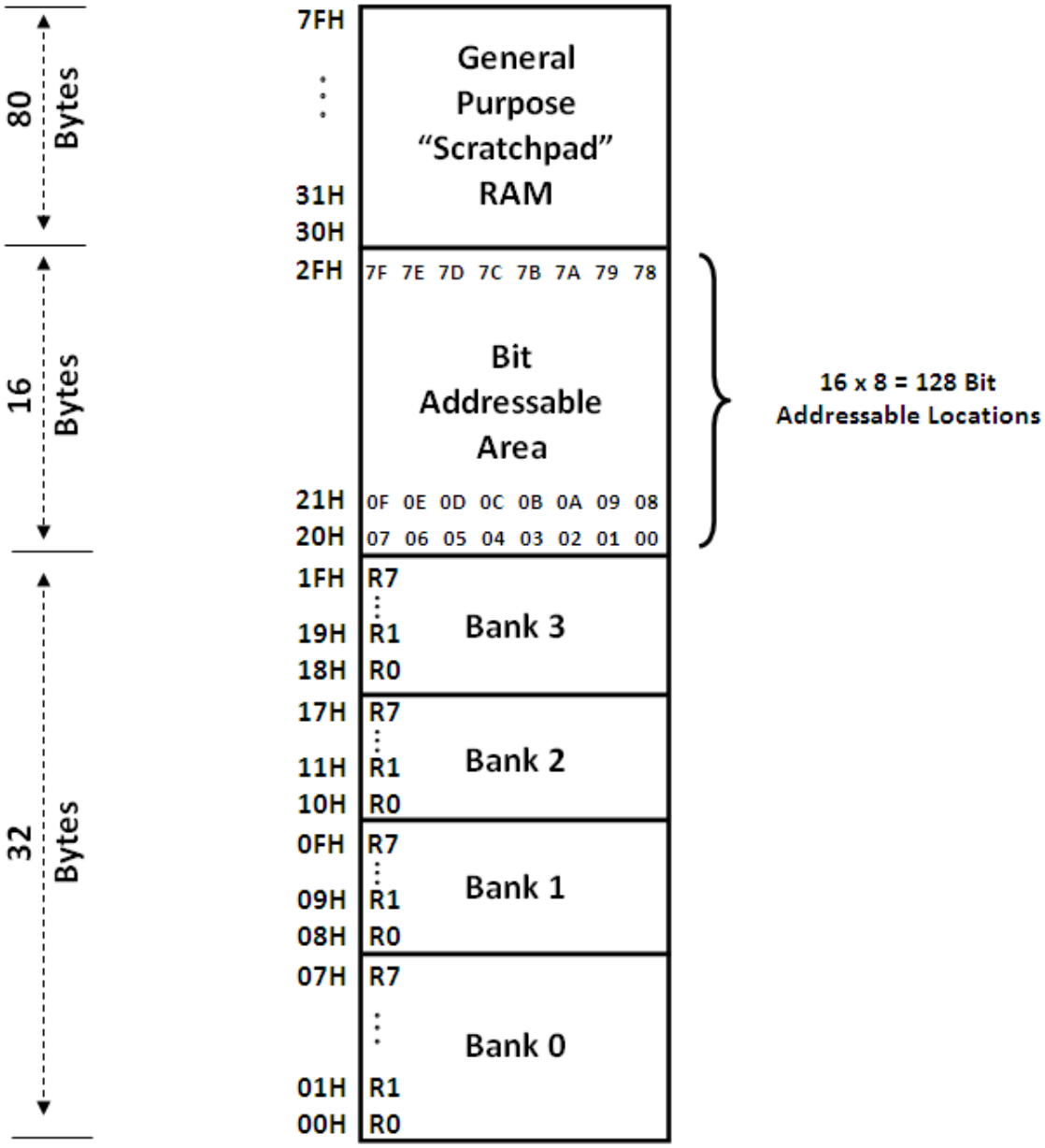
8051 checks **$\overline{EA}$** pin during every ROM operation where the address is 0000H… 0FFFH.

If **$\overline{EA}$ = 1,** this location is accessed from internal ROM.

If **$\overline{EA}$ = 0,** this location is accessed from external ROM.

If the address is 1000H or more, 8051 does not check **$\overline{EA}$** as this location can only be present in External ROM.

STRUCTURE OF INTERNAL RAM OF 8051



8051 has a 128 Bytes of internal RAM.
These are 128 locations of 1 Byte each.
The address range is 00H… 7FH.
This RAM is used for storing data.
It is divided into three main parts: Register Banks, Bit addressable area and a general purpose area.

REGISTER BANKS

1) The **first 32 locations (Bytes)** of the Internal RAM from **00H… 1FH**, are used by the programmer as general purpose registers.
2) Having so many general purpose registers makes programming easier and faster.
3) But as a downside, this also vastly increases the number of opcodes (refer my class lectures for detailed understanding of this).
4) **Hence the 32 registers are divided into 4 banks, each having 8 Registers R0… R7.**
5) The first 8 locations 00H… 07H are registers R0… R7 of bank 0.
6) Similarly locations 08H… 0FH are registers R0… R7 of bank 1 and so on.
   A register can be addressed using its name, or by its address.
   Eg: Location 00H can be accessed as R0, if Bank 0 is the active bank.
   MOV A, R0; "A" register gets data from register R0.
   It can also be accessed as Location 00H, irrespective of which bank is the active bank.
   MOV A, 00H; "A" register gets data from Location 00H.
7) The appropriate bank is selected by the **RS1, RS0 bits of PSW**.
   Since PSW is available to the programmer, any Bank can be selected at run-time.
8) **Bank 0** is selected by **default**, on reset.

BIT ADDRESSABLE AREA

1) The **next 16-bytes** of RAM, from **20H… 2FH**, is available as **Bit Addressable** Area.
2) We can perform ordinary byte operations on these locations, as well as bit operations.
   #Please refer Bharat Sir's Lecture Notes for detailed explanation on this …
3) As each location has 8-bits, we have a total of ➔ 16 × 8 = **128 Addressable Bits.**
4) These bits can be addressed using their individual address **00H … 7FH**.
   SETB 00H; Will store a "1" on the LSB of location 20H
   CLR 07H; Will store a "0" on the MSB of location 20H
5) Normal **"BYTE"** operations can also be performed at the addresses: **20H … 2FH**.
   MOV 20H, #00H; Will store a "0" on all 8-bits of location 20H.
6) Here is something very interesting to know and will also help you understand further topics.
   The entire internal RAM is of 128 bytes so the address range is 00H… 7FH.
   The bit addressable area has 128 bits so its bit addresses are also 00h… 7FH.
7) This means every address 00H… 7FH can have two meanings, it could be a byte address or a bit address.
8) This does not lead to any confusion, because the instruction in which we use the address, will clearly indicate whether it is a bit operation or a byte operations.
   SETB, CLR etc. are bit ops whereas ADD, SUB etc. are byte operations.
9) **SETB 00H;** This is a bit operation.
   It will make Bit location 00H contain a value "1".
10) **MOV A, 00H;** This is a byte operation.
   "A" register will get 8-bit data from byte location 00H.

GENERAL PURPOSE AREA

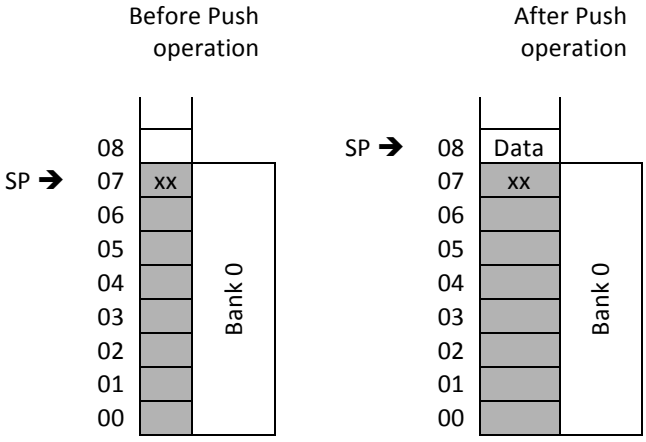
1) The general-purpose area ranges from location 30H … 7FH.
2) This is an 80-byte area which can be used for general data storage.

STACK OF 8051

1) Another important element of the Internal RAM is the **Stack**.
2) Stack is a set of memory locations operating in Last In First Out (LIFO) manner.
3) It is used to store return addresses during ISRs and also used by the programmer to store data during programs.
4) In 8051, the **Stack** can only be present in the **Internal RAM**.
5) This is because, **SP** which is an **8-bit register**, can only contain an 8-bit address and External RAM has 16-bit address. (#Viva)
6) On **reset SP** gets the value **07H**.
7) Thereafter SP is changed by every PUSH or POP operation in the following manner:

PUSH:	**POP:**
SP ← SP + 1	Data ← [SP]
[SP] ← New data	**SP ← SP – 1**

8) The reset value of SP is 07H because, on the first PUSH, SP gets incremented and then data is pushed on to the stack. This means the very first data will be stored at location 08H.
9) This does not affect the default bank (0) and still gives the stack, the maximum space to grow. (#Viva)



10) The programmer can relocate the stack to any desired location by simply putting a new value into SP register.