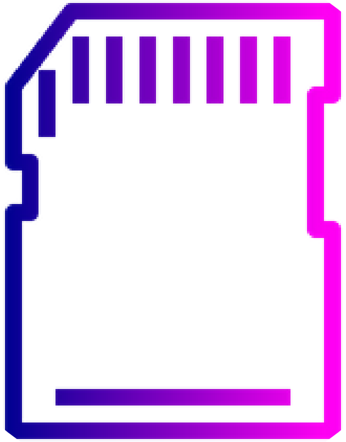
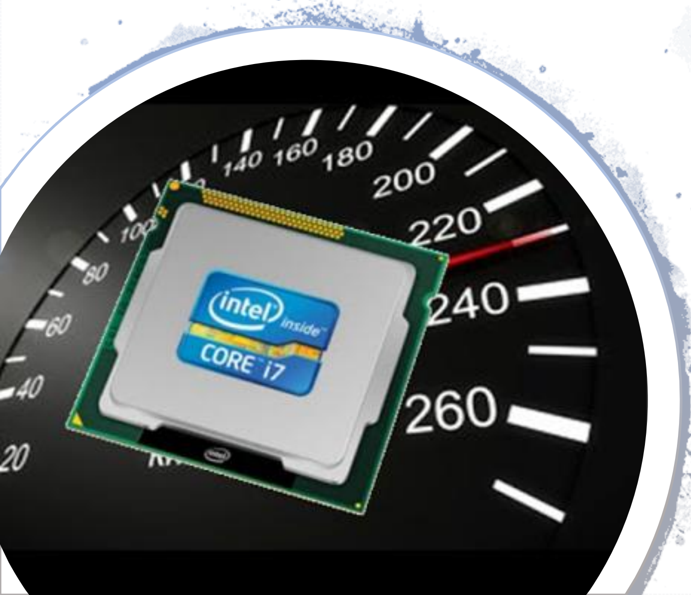
Introduction to Embedded Systems

Sem IV

Mr. Umesh Koyande

Mr. Kiran Datar



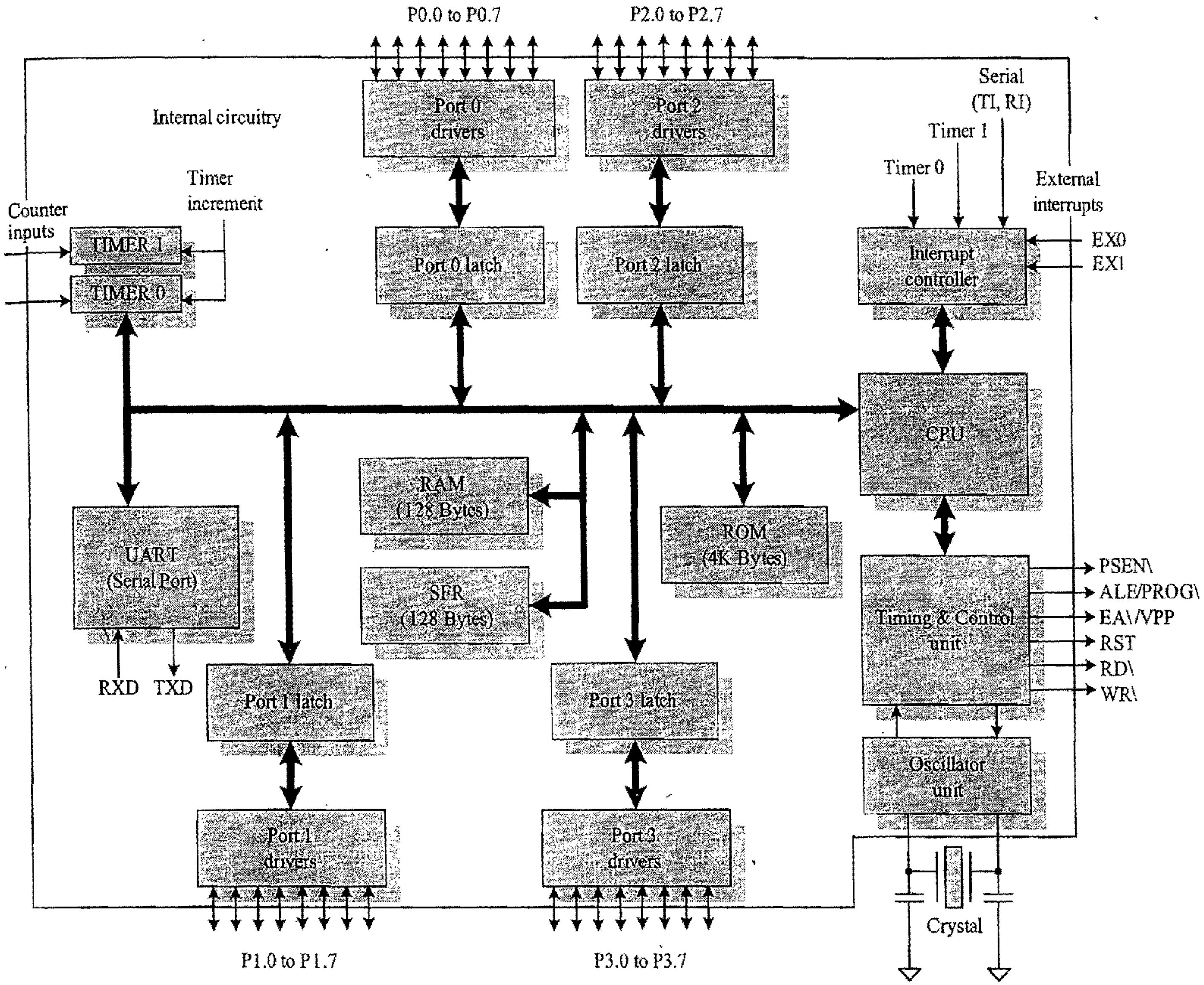


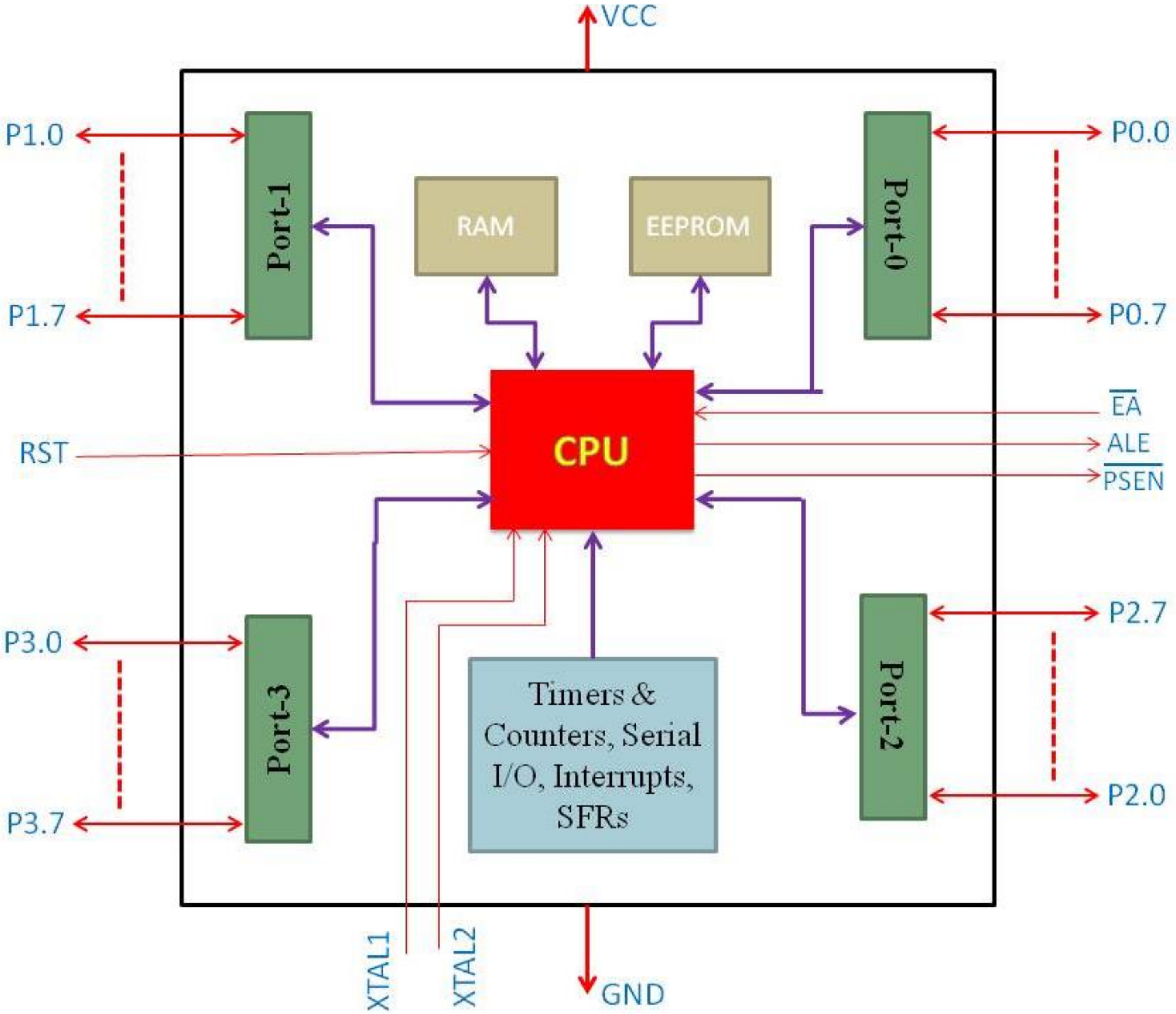
Factors To Be Considered In Selecting A Controller

- Speed of Operation
- Code Memory Space
- Data Memory Space
- Development Support
- Availability
- Power Consumption
- Cost

Why 8051 Microcontroller

- 6 interrupts (2 external interrupts , 2 timer interrupts and 2 serial interrupts)
- Two 16 bit timers/counters
- 32 I/O lines and programmable full duplex serial interface
- Available through more than 20 vendors with more than 100 variants of 8051
- Supports CAN, USB, SPI and TCP/IP interfaces
- Integrated ADC/DAC
- Cost is very low

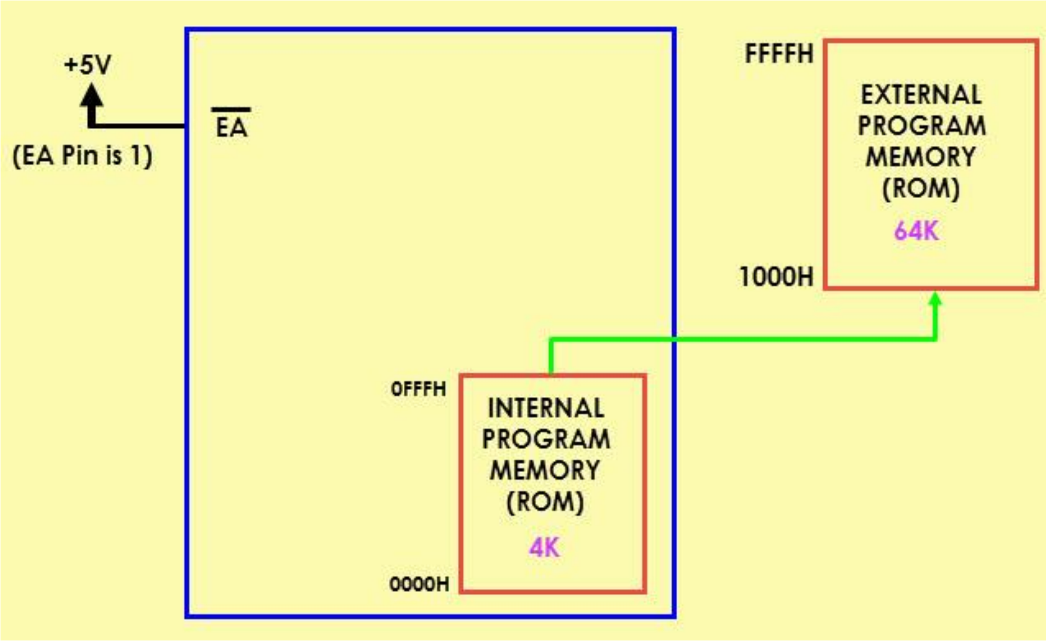

P0.0 to P0.7
P2.0 to P2.7
Internal circuitry
Port 0 drivers
Port 2 drivers
Serial (TI, RI)
Timer 1
Timer 0
External interrupts
Timer increment
Counter inputs
TIMER 1
TIMER 0
Port 0 latch
Port 2 latch
Interrupt controller
EX0
EX1
CPU
RAM (128 Bytes)
SFR (128 Bytes)
ROM (4K Bytes)
UART (Serial Port)
RXD
TXD
Timing & Control unit
PSEN\
ALE/PROG\
EA\/VPP
RST
RD\
WR\
Port 1 latch
Port 3 latch
Oscillator unit
Port 1 drivers
Port 3 drivers
Crystal
P1.0 to P1.7
P3.0 to P3.7

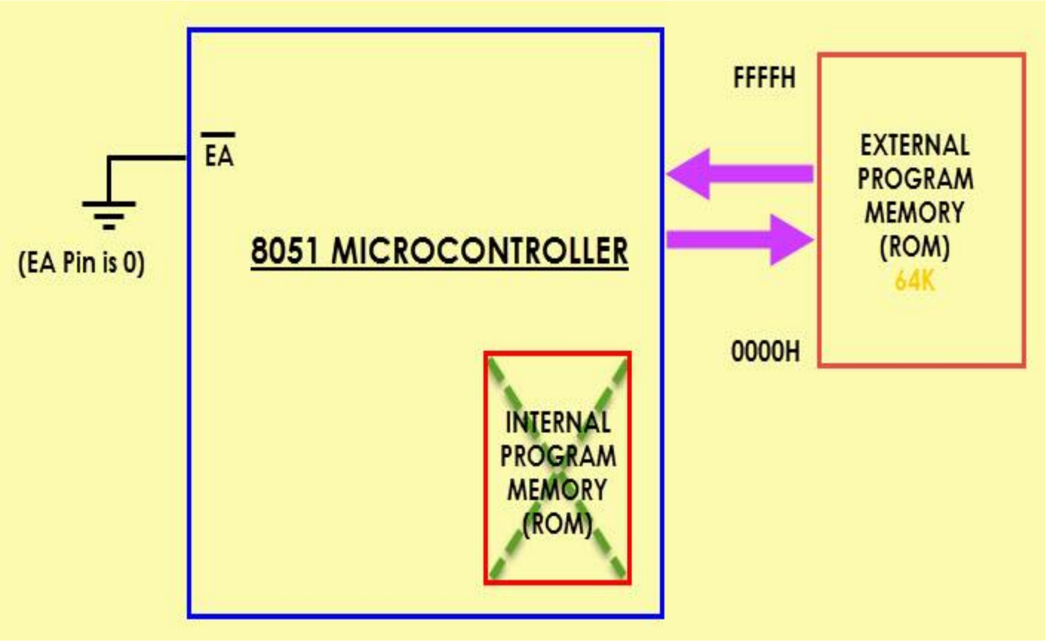


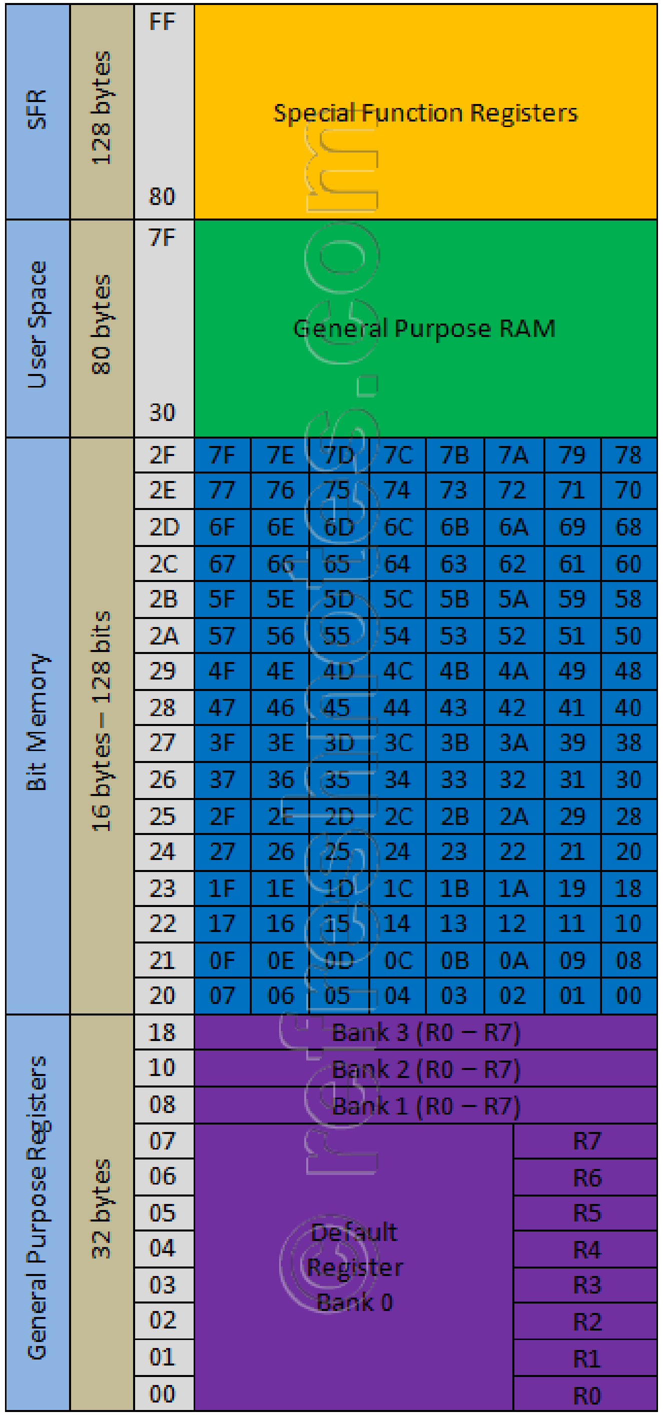
Designing with 8051

Memory Organisation

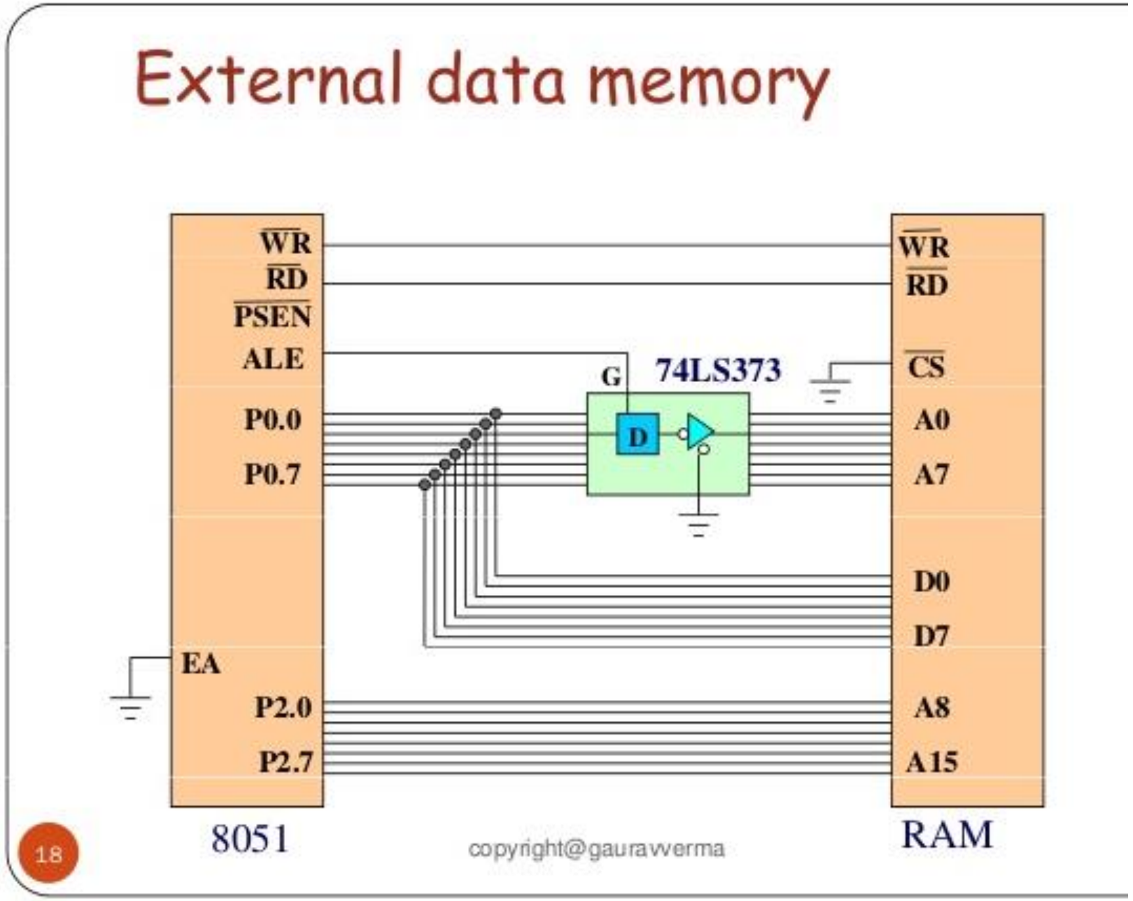
- 4K bytes of program memory as On-Chip memory
- External Access EA pin







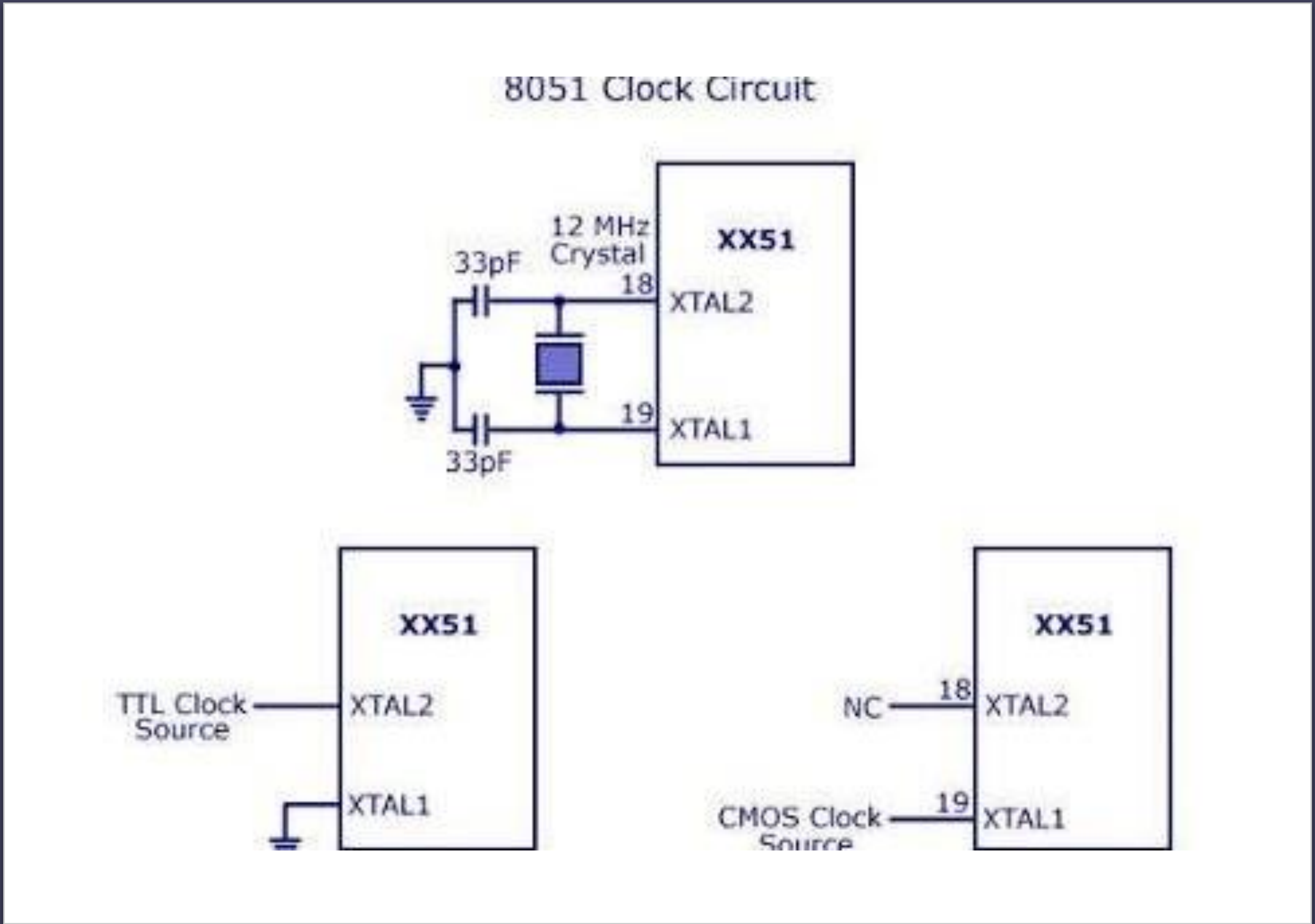
SFR	128 bytes	FF 80	Special Function Registers							
User Space	80 bytes	7F 30	General Purpose RAM							
Bit Memory	16 bytes – 128 bits	2F	7F	7E	7D	7C	7B	7A	79	78
		2E	77	76	75	74	73	72	71	70
		2D	6F	6E	6D	6C	6B	6A	69	68
		2C	67	66	65	64	63	62	61	60
		2B	5F	5E	5D	5C	5B	5A	59	58
		2A	57	56	55	54	53	52	51	50
		29	4F	4E	4D	4C	4B	4A	49	48
		28	47	46	45	44	43	42	41	40
		27	3F	3E	3D	3C	3B	3A	39	38
		26	37	36	35	34	33	32	31	30
		25	2F	2E	2D	2C	2B	2A	29	28
		24	27	26	25	24	23	22	21	20
		23	1F	1E	1D	1C	1B	1A	19	18
		22	17	16	15	14	13	12	11	10
		21	0F	0E	0D	0C	0B	0A	09	08
		20	07	06	05	04	03	02	01	00
General Purpose Registers	32 bytes	18	Bank 3 (R0 – R7)							
		10	Bank 2 (R0 – R7)							
		08	Bank 1 (R0 – R7)							
		07	Default Register Bank 0							R7
		06								R6
		05								R5
		04								R4
		03								R3
		02								R2
		01								R1
		00								R0



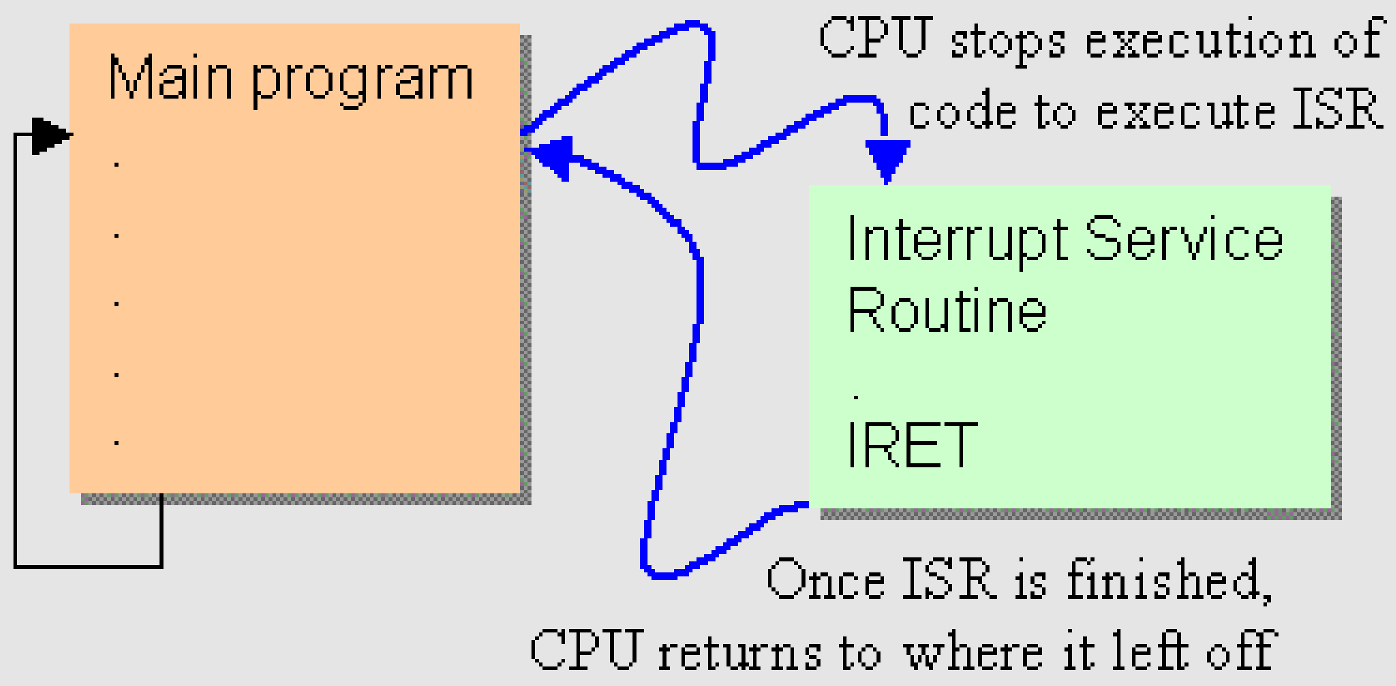
External data memory

copyright@gauraverma

Oscillator Unit



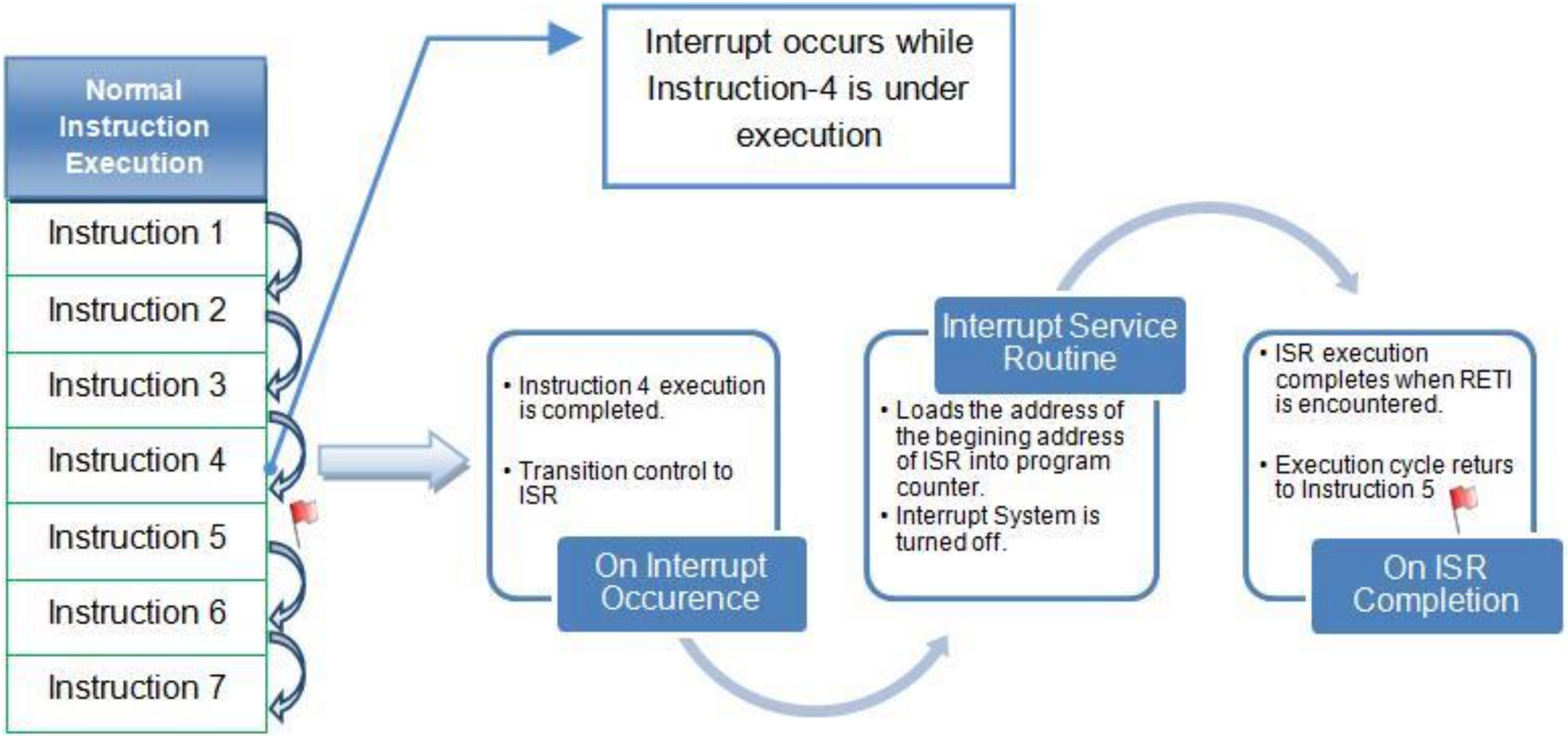
Interrupts



Main program
.
.
.
.
.
CPU stops execution of code to execute ISR
Interrupt Service Routine
.
IRET
Once ISR is finished, CPU returns to where it left off

- I/O data transfer between peripheral devices and processor/controller
- Timing applications
- Handling emergency situations
- Context switching/Multitasking/ Real-Time application programming
- Event driven programming

Interrupts



Normal Instruction Execution

Instruction 1

Instruction 2

Instruction 3

Instruction 4

Instruction 5

Instruction 6

Instruction 7

Interrupt occurs while Instruction-4 is under execution

On Interrupt Occurence

- Instruction 4 execution is completed.
- Transition control to ISR

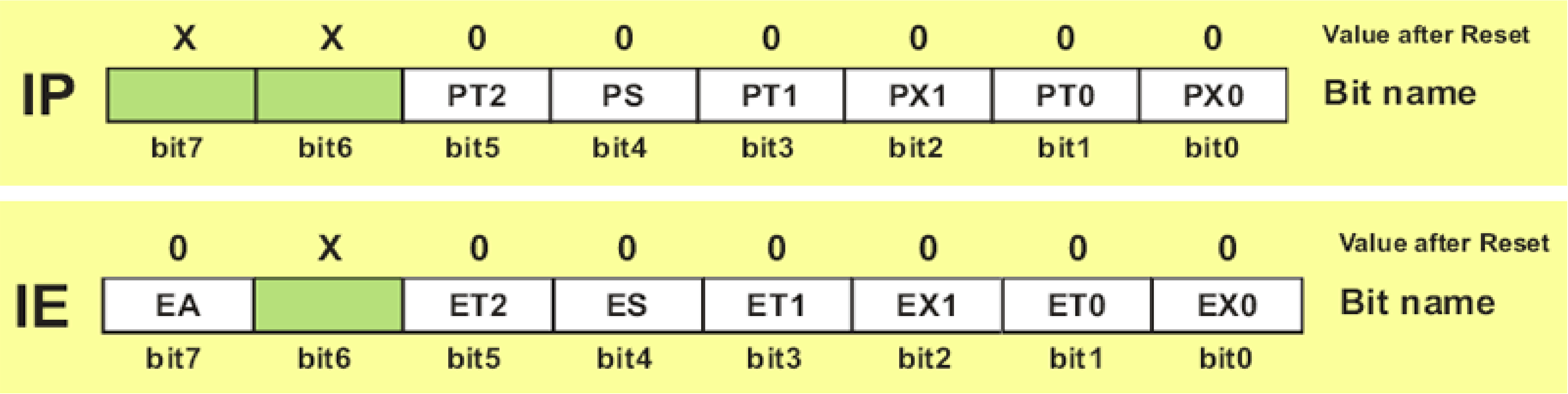
Interrupt Service Routine

- Loads the address of the begining address of ISR into program counter.
- Interrupt System is turned off.

On ISR Completion

- ISR execution completes when RETI is encountered.
- Execution cycle returs to Instruction 5

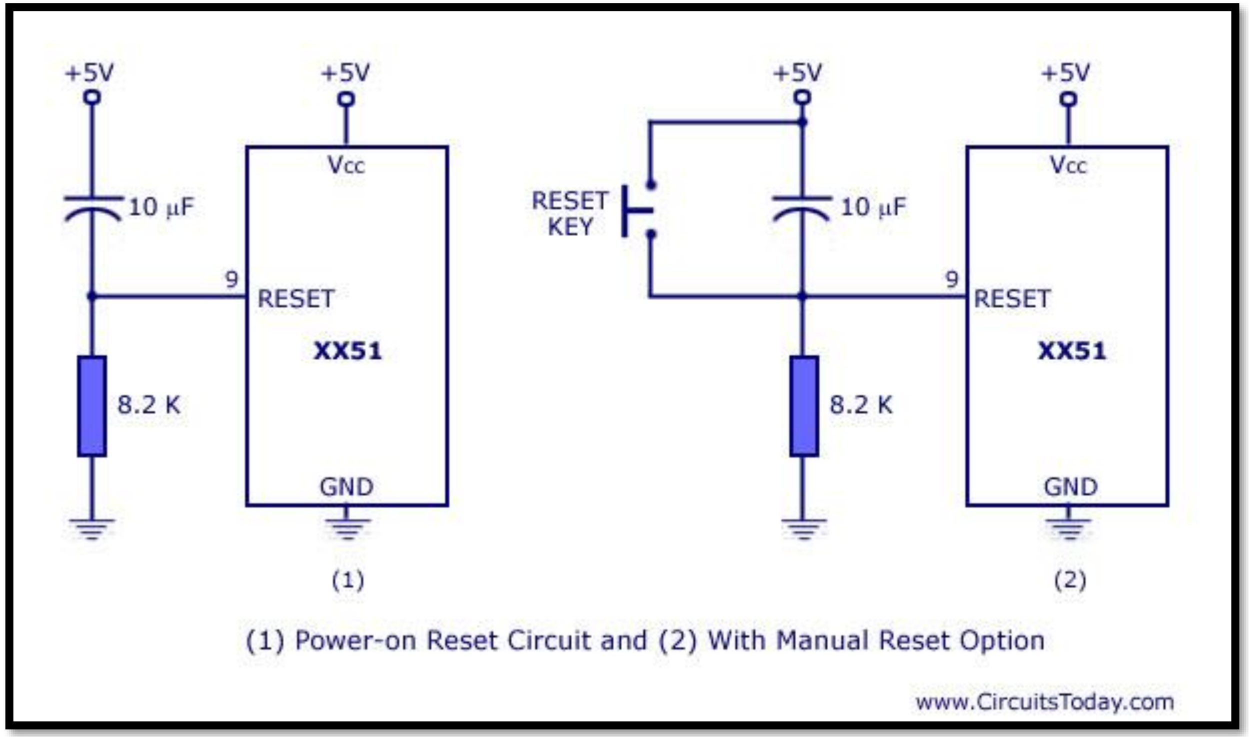
Interrupts



	X	X	0	0	0	0	0	0	Value after Reset
IP	🟩	🟩	PT2	PS	PT1	PX1	PT0	PX0	Bit name
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	

	0	X	0	0	0	0	0	0	Value after Reset
IE	EA	🟩	ET2	ES	ET1	EX1	ET0	EX0	Bit name
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	

Reset Circuitry



(1) Power-on Reset Circuit and (2) With Manual Reset Option

Necessary to bring internal hardware circuitry to a known state

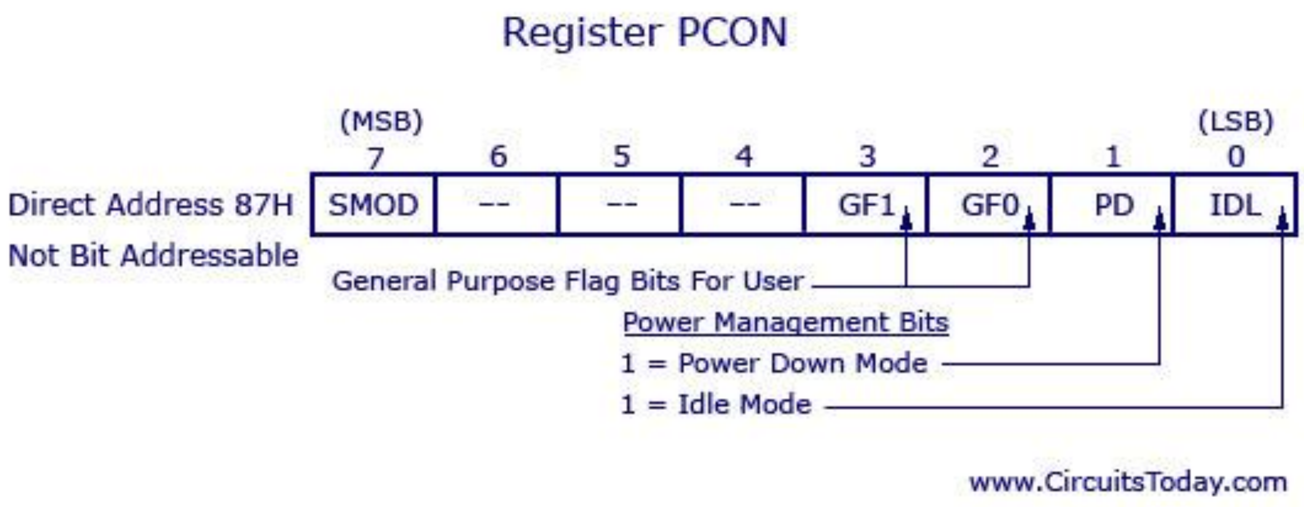
Reset can be of two types:

- Hardware reset
- Software reset

Reset signal must be kept active until all the three of the following conditions are met

- Power supply must be in the specified range .
- Oscillator must reach a minimum oscillation level to ensure a good noise to signal ratio and correct internal duty cycle generation.
- Reset pulse width duration must be at least two machine cycles (24 Clock periods) when conditions 1 and 2 are met.

Power Saving Modes



Register PCON

	(MSB) 7	6	5	4	3	2	1	(LSB) 0
Direct Address 87H Not Bit Addressable	SMOD	--	--	--	GF1	GF0	PD	IDL

General Purpose Flag Bits For User

Power Management Bits

1 = Power Down Mode

1 = Idle Mode

www.CircuitsToday.com

Power Saving Modes

IDLE Mode

- internal clock to the processor is temporarily suspended
- By setting IDLE bit in PCON =1
- Stops program execution and contents of internal RAM are preserved
- Oscillator continues to run, but clock is disconnected from CPU
- Timer and serial port operates normally
- Come out by activation of any Interrupt or RESET, this will make IDLE=0
- After execution of ISR, program resume from instruction after set idle.

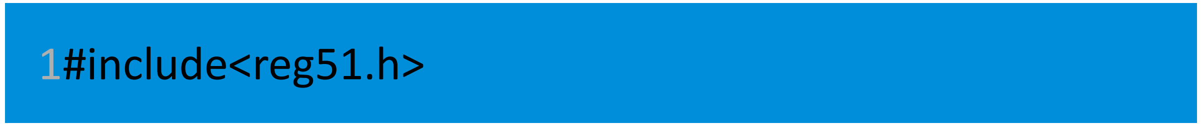
Power Saving Modes

Power Down Mode	By setting PWDN=1
	Stops on chip oscillator
	Program execution, timers and serial port operation also stops
	Content of internal RAM are preserved
	Come out of by RESET

Structure of Embedded Program

Preprocessor directive

- Indication to the compiler that it must look into this file for symbols that are not defined in the program.
- Usually represented using #include… or #define….
- To indicate a header file specific to the microcontroller, which contains all the SFRs and the bits in those SFRs.



```
#include<reg51.h>
```

Structure of Embedded Program

Comments

- Comments are readable text that are written to help us (the reader) understand the code easily
- Ignored by the compiler and do not take up any memory in the final code (after compilation)
- Denoted by // or multiline comments denoted by /*….*/

Structure of Embedded Program

Global Variables:

- Global Variables, as the name suggests, are Global to the program i.e. they can be accessed anywhere in the program

Local Variables:

- Local Variables, in contrast to Global Variables, are confined to their respective function

Main Function:

- Every C or Embedded C Program has one main function, from where the execution of the program begins

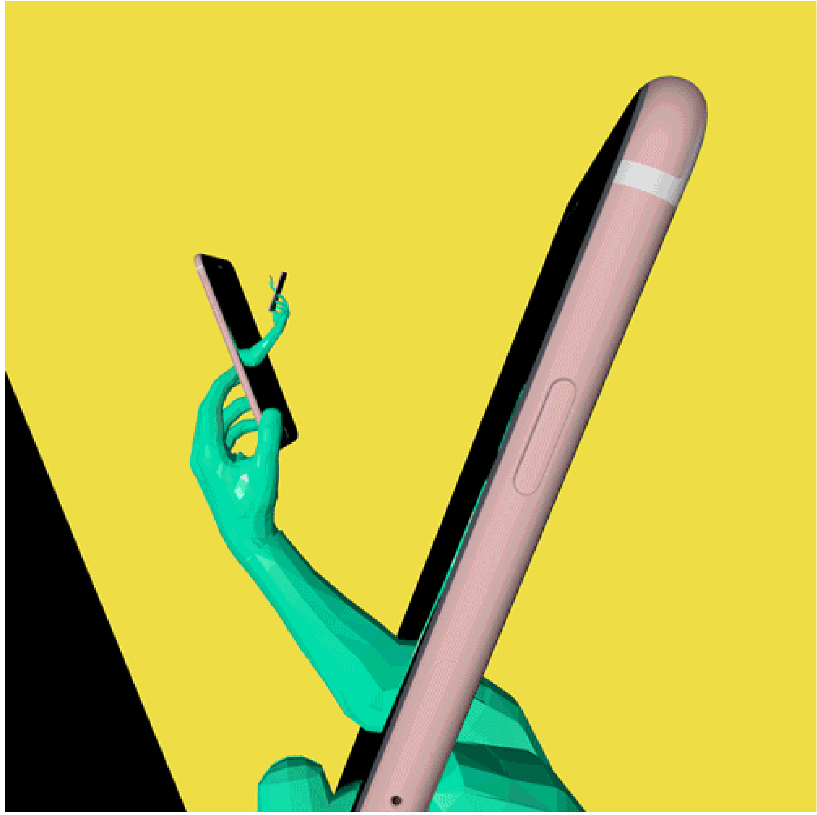
```
#include<reg51.h> // Preprocessor Directive
void delay (int);    // Delay Function Declaration
void main(void)    // Main Function
{
P1 = 0x00;          /* Making PORT1 pins LOW. All the LEDs are OFF. (P1 is PORT1, as defined in reg51.h) */
while(1)             // infinite loop
{
P1 = 0xFF;       // Making PORT1 Pins HIGH i.e. LEDs are ON.
delay(1000);    /* Calling Delay function with Function parameter as 1000. This will cause a delay of 1000mS i.e. 1 second */
P1 = 0x00;      // Making PORT1 Pins LOW i.e. LEDs are OFF.
delay(1000);
}
}
void delay (int d) // Delay Function Definition
{
unsigned int i=0; // Local Variable. Accessible only in this function. /*
for(;d>0;d–)
{
for(i=250;i>0;i – -);
for(i=248;i>0;i – -);
}
}
```

Infinite loop

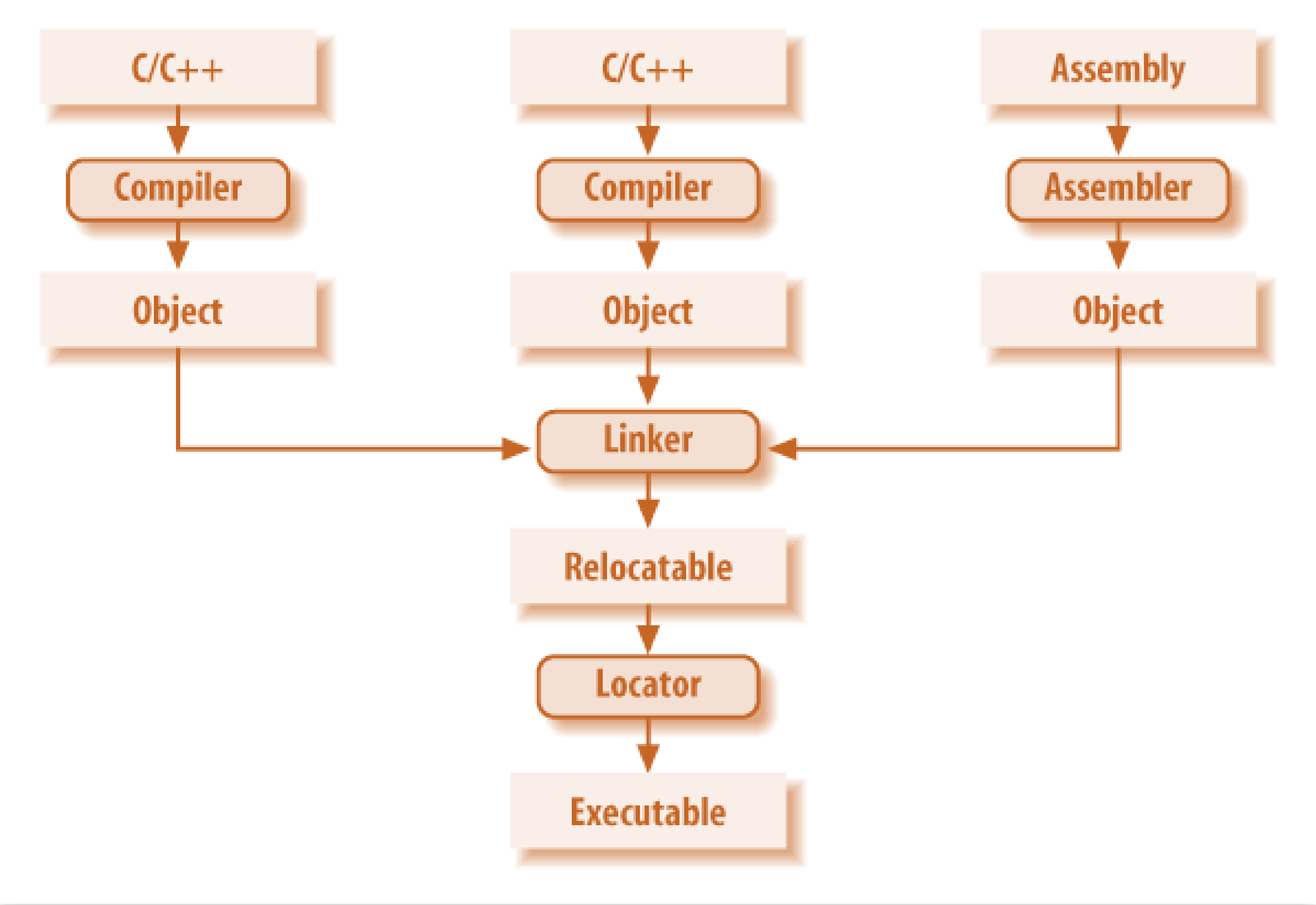
- Piece of coding that lacks a functional exit so that it repeats indefinitely
- No break statement so as to get out of the loop, it just keeps looping over the statements within the block defined.
- Example:

```
While(Boolean True) OR for(;;);
{
//Code
}
```

- Embedded systems need infinite loops for repeatedly processing/monitoring the state of the program.



Build Process

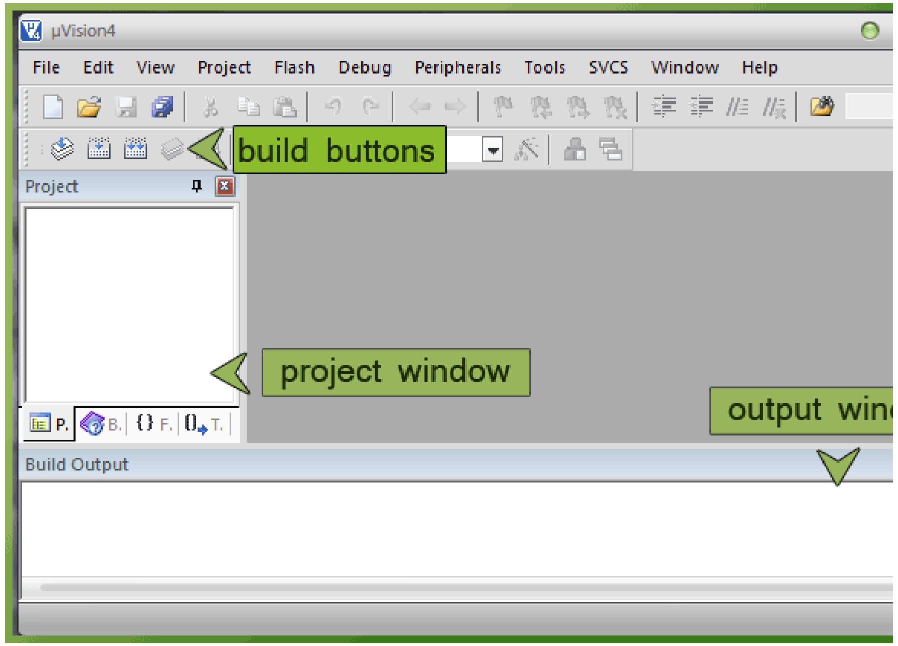


Build Process

Process of converting the source code representation of your embedded software into an executable binary image

- Each of the source files must be compiled or assembled into an object file.
- All of the object files that result from the first step must be linked together to produce a single object file, called the relocatable program.
- Physical memory addresses must be assigned to the relative offsets within the relocatable program in a process called relocation.

Result of the final step is a file containing an executable binary image that is ready to run on the embedded system



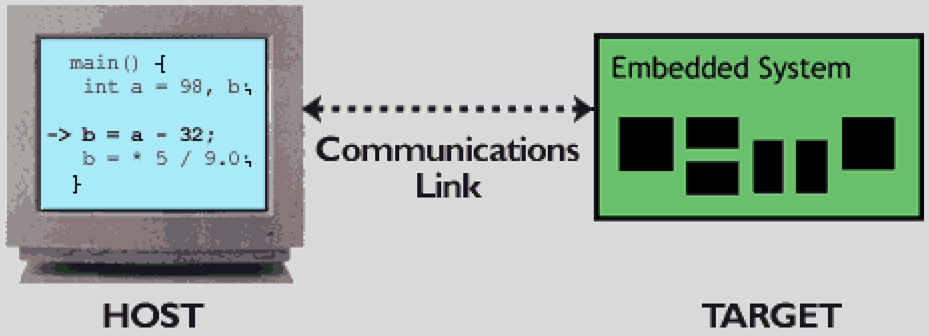
Split between host and target

Host Platform

- Software Development is performed on a Host computer
- Development Platform
- General Purpose
- Development Tools -Editor –Compiler -Debugger

Target Platform

- Target hardware (processor, memory, I/O)
- Runtime environment (Operating System/Kernel)
- Target hardware platform contains only what is needed for final deployment
- Target hardware platform does not contain development tools (editor, compiler, debugger)



```
  main() {
   int a = 98, b;

-> b = a - 32;
   b = * 5 / 9.0;
  }
```

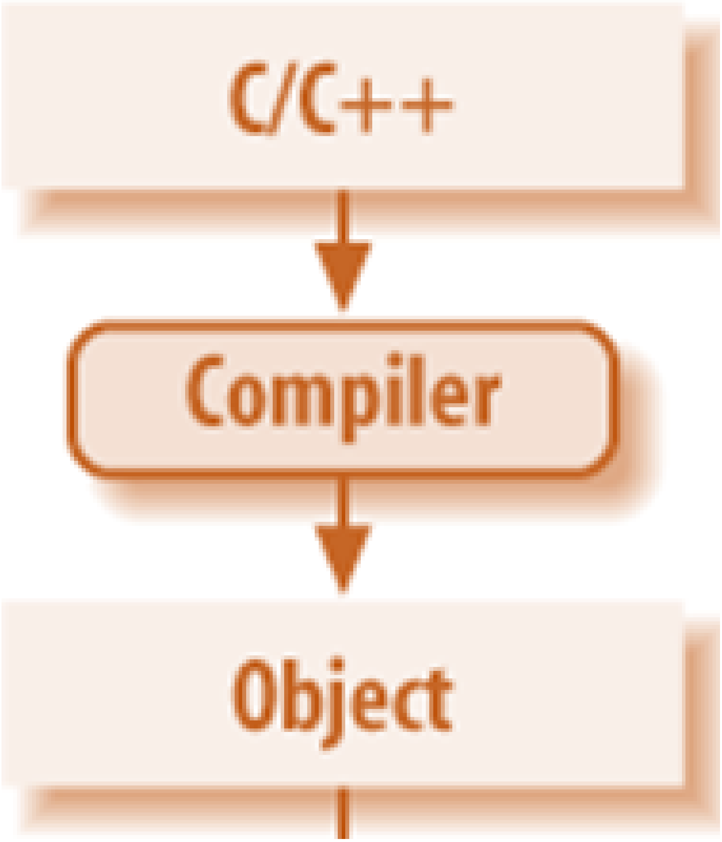
Compiler

Compiler translates program written in human-readable language into machine language

- Source Code --> Object file
- Object file is binary file that contains set of machine-language instructions (opcodes) and data resulting from language translation process

Machine-language instructions are specific to a particular processor

A Cross-compiler runs on one computer platform and produces code for *another* computer platform



Compiler

Format of the object file

- Common Object File Format (COFF)
- Extended Linker Format (ELF)

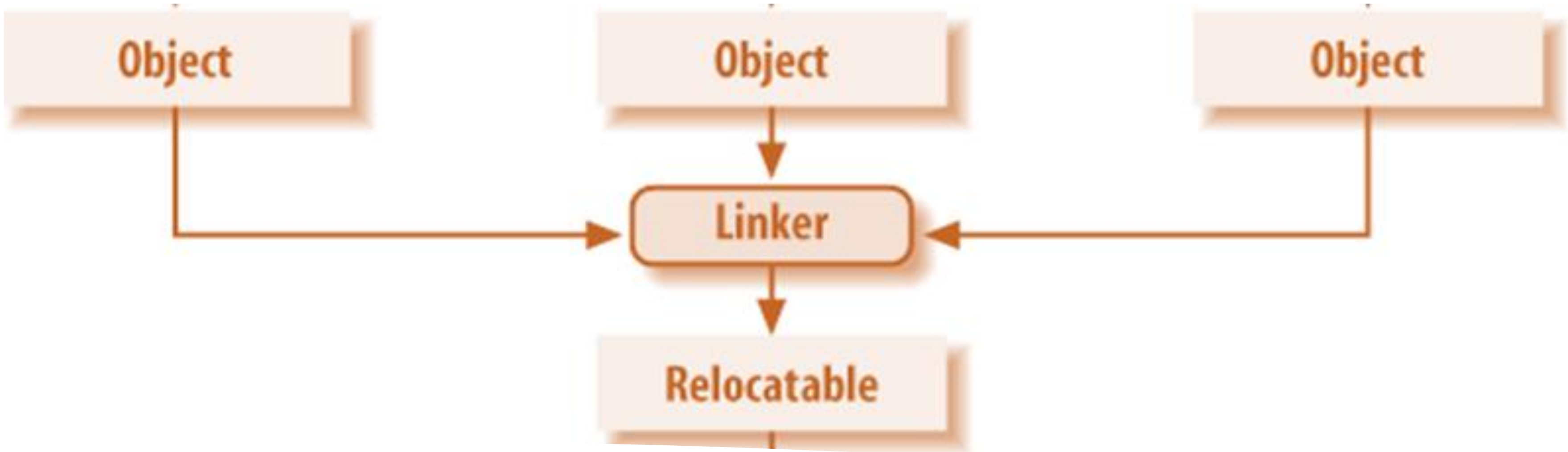
Structure of object file

- Text section – code blocks
- Data – initialized global variables
- Bss – uninitialized global variables

Unresolved references

- Some of the variables and functions which are defined in one source file, whose reference may be found in some other source file.

Linker



Output of the linker is a new object file that contains all of the code and data from the input object files and is in the same object file format.

Unresolved symbols

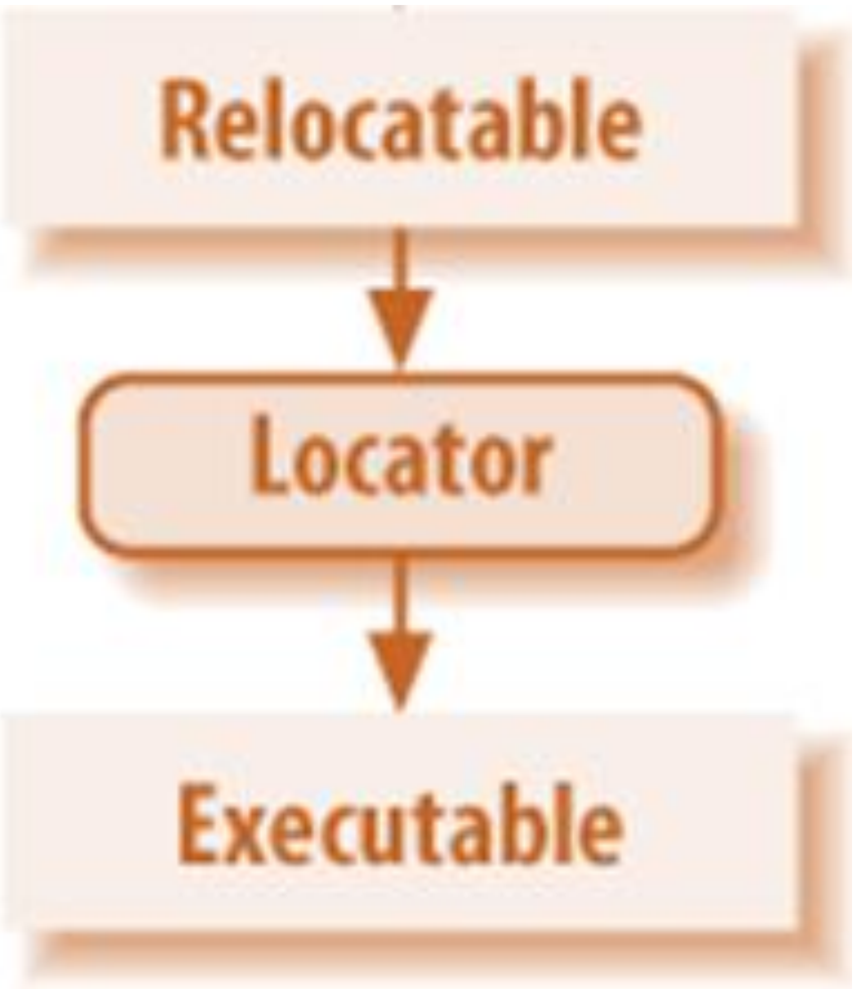
- Will be replaced with a reference to the actual variable
- If the same symbol is declared in more than one object file, the linker is unable to proceed
- If a symbol reference remains unresolved, linker will try to resolve the reference on its own

Output of linker will be a special "relocatable" copy of the program

Entire embedded application-including the operating system-is almost always statically linked together and executed as a single binary image

Locator

- A Locator is the tool that performs the conversion from relocatable program to executable binary image
- The Locator assigns physical memory addresses to code and data sections within the relocatable program
- The Locator produces a binary memory image that can be loaded into the target ROM.
- In many cases, the locator is a separate development tool. However, in some of the cases this functionality is built right into the linker.



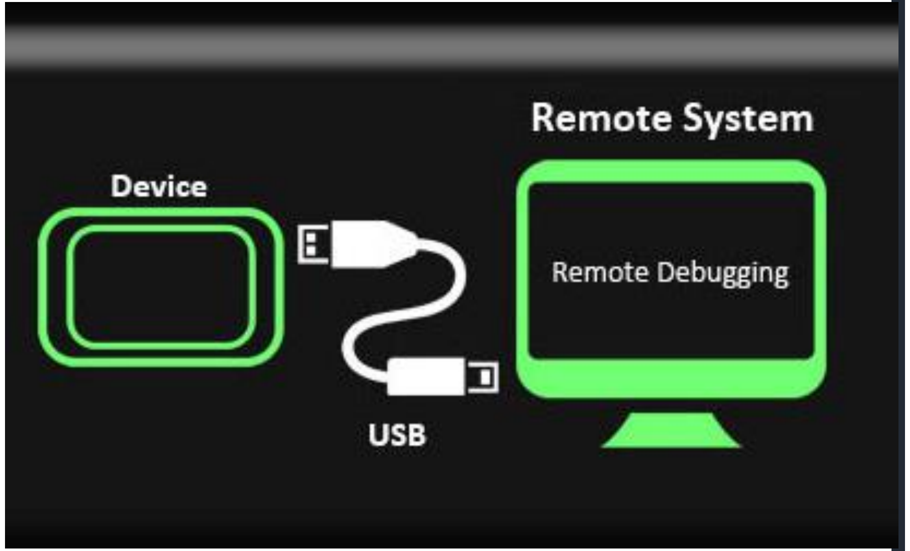
To develop software for an embedded system

- – Create source file (on Host)
- – Type in C code (on Host)
- – Compile/Assemble: translate into machine code (on Host)
- – Link: combine all object files and libraries, resolve all symbols (on Host)
- – Locate: assign memory addresses to code and data (on Host)
- – Download: copy executable image into Target processor memory
- – Execute: reset Target processor

Downloading in ROM

- **A device programmer** is a computer system that is capable of programming memory devices of all sorts.
- Connected to host computer, through which the files that contain executable binary images could be transferred to it for ROM programming.
- Process can take from a few seconds to several minutes
- After you program the ROM, it is ready to be inserted into its socket on the board.
- The power should be turned off and then reapplied only after the chip has been carefully inserted.
- When the processor is reset, it begins by fetching and executing whatever is stored at particular physical address.

Remote Debuggers



Remote debugger can be used to download, execute, and debug embedded software

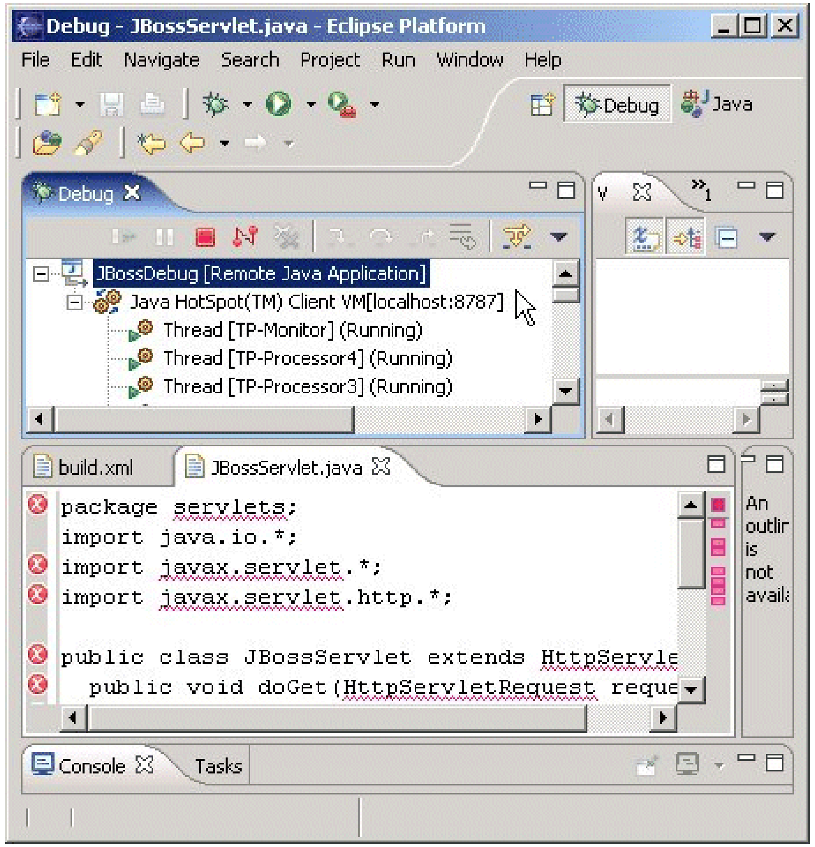
Frontend of a remote debugger usually has a text or GUI-based

Debugger and the software being debugged are executing on two different computer systems.

There is also a hidden backend that runs on the target processor and communicates with the frontend over a communications link

Backend provides for low-level control of the target processor and is usually called the debug monitor.

Remote Debuggers



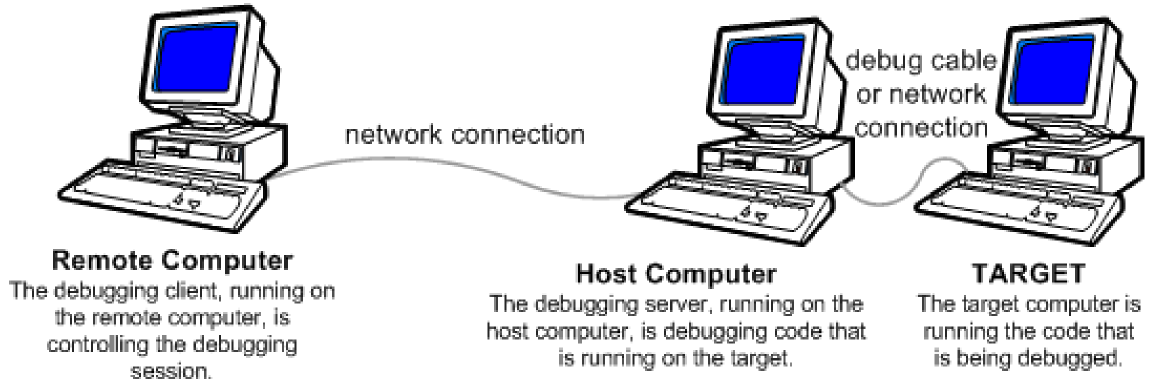
Debug - JBossServlet.java - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Debug
Java
Debug
build.xml
JBossServlet.java
Console
Tasks

```
JBossDebug [Remote Java Application]
  Java HotSpot(TM) Client VM[localhost:8787]
    Thread [TP-Monitor] (Running)
    Thread [TP-Processor4] (Running)
    Thread [TP-Processor3] (Running)
```

```
package servlets;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class JBossServlet extends HttpServle
  public void doGet(HttpServletRequest reque
```

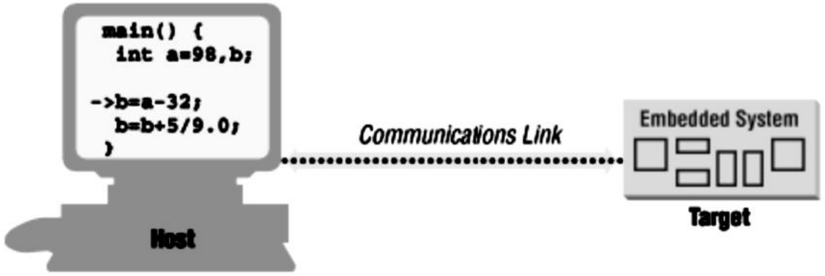
An outlir is not avail



Remote Computer
The debugging client, running on the remote computer, is controlling the debugging session.

Host Computer
The debugging server, running on the host computer, is debugging code that is running on the target.

TARGET
The target computer is running the code that is being debugged.



```
main() {
 int a=98,b;

->b=a-32;
 b=b+5/9.0;
}
```

Fig. **:** A remote debugging session

Thank You!