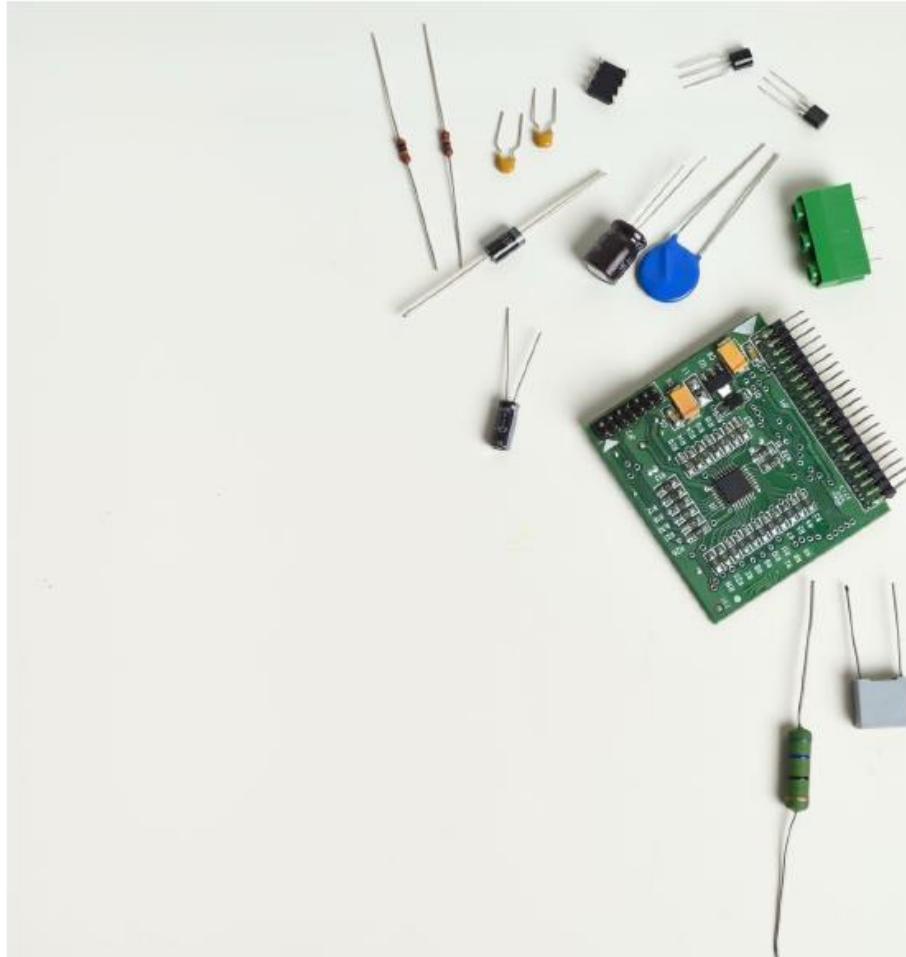


# Embedded System



---

**DR MANOJ TOLANI (PHD-IIIT ALLAHABAD)**  
**ASSISTANT PROFESSOR (DEPARTMENT OF ICT)**



# MY INTRODUCTION

- First, let me introduce myself. My name is **Manoj Tolani**. I joined MIT, Manipal (MAHE) in **April 2023**.
- I am currently **Assistant Professor in Information and Communication Technology Department**.
- I received an **MTech degree (Specialization: Digital Systems)** from the **Madan Mohan Malviya Engineering College, Gorakhpur** (Now MMMUT) in 2012.
- After MTech, I joined **PSIT Engineering College** as an **Assistant Professor in 2012**. I worked for **3.5 Years at PSIT**.
- I attempted **GATE Exam in 2015**, and my **AIR was 1214**. I also qualified for UGC-NET also in 2015.
- In **2016**, I joined **IIIT Allahabad** as a research scholar for a PhD Degree.
- I **completed my Ph.D. in Feb 2021**. I worked **6 months at IIIT-Allahabad as Teaching-cum-Research Associate**.
- After that, I joined **Atria Institute of Technology, Bangalore** as an Assistant Professor. I worked for **2 Years at AIT**, and after that, I joined this college (**MIT, Manipal**) in **April 2023**.
- My research interest includes **Wireless Sensor Networks, IoT, Photonics, and Machine Learning**.
- I have published **18+ research papers in SCI/Scopus Journals/Conferences** (IEEE, Elsevier, Springer etc.)

# Lecture-1

**Course Objectives**

- Familiarise with the building blocks of microcontroller and its applications in embedded system
- Understand ARM- Cortex M instruction set to implement assembly language programs
- Understand the key concepts of ARM-Cortex M microcontroller architecture
- Design and program an embedded system for real world applications

.

**Course Outcomes**

At the end of this course, students will be able to

1. Demonstrate the salient features of embedded systems.
2. Explain addressing modes and assembly language instructions
3. Illustrate the architecture of ARM Cortex- M microcontroller.
4. Develop the application software for ARM Cortex-M microcontroller-based Systems.
5. Design real world systems using ARM Cortex-M microcontroller

# Syllabus

## **Introduction to Embedded Systems and ARM Cortex-M Microcontroller**

ALU, Register, Control unit, Flag register, BUS interface, Memory systems, Input output, microprocessor versus microcontroller, CISC vs. RISC, choosing a microcontroller, ARM history, ARM Cortex M Architecture

**[5 Hours]**

## **Assembly language programming**

Addressing modes, Data transfer instructions, Data format and directives, Arithmetic and logical instructions, Shift and rotate instructions, Branch and conditional branch instructions, Function call and return, Stack, Recursive functions, Conditional execution, assembly language programs.

**[10 Hours]**

## **Input/output (IO) programming**

Pin connect block, Pin function select registers, General Purpose Input and Output (GPIO) registers, GPIO configuration, GPIO programming using ARM C language, Interfacing: LEDs, Seven segment, multiplexed seven segments, LCD, keyboard, DC motor, Stepper motor.

**[13 Hours]**

## **Timer/ Counter programming**

Timer versus counter, timer registers, timer architecture and operation

**[6 Hours]**

## **Interrupt programming**

Nested Vectored Interrupt Controller (NVIC), external hardware interrupts, IO interrupts, timer/counter interrupts,

**[5 Hours]**

## **ADC, DAC, PWM, Serial Interfacing**

Analog to Digital Converter (ADC), ADC registers, ADC programming. DAC Registers, DAC programming, PWM registers, PWM programming, General introduction to serial interfacing, RS232, MAX 232, UART, UART programming, data acquisition system

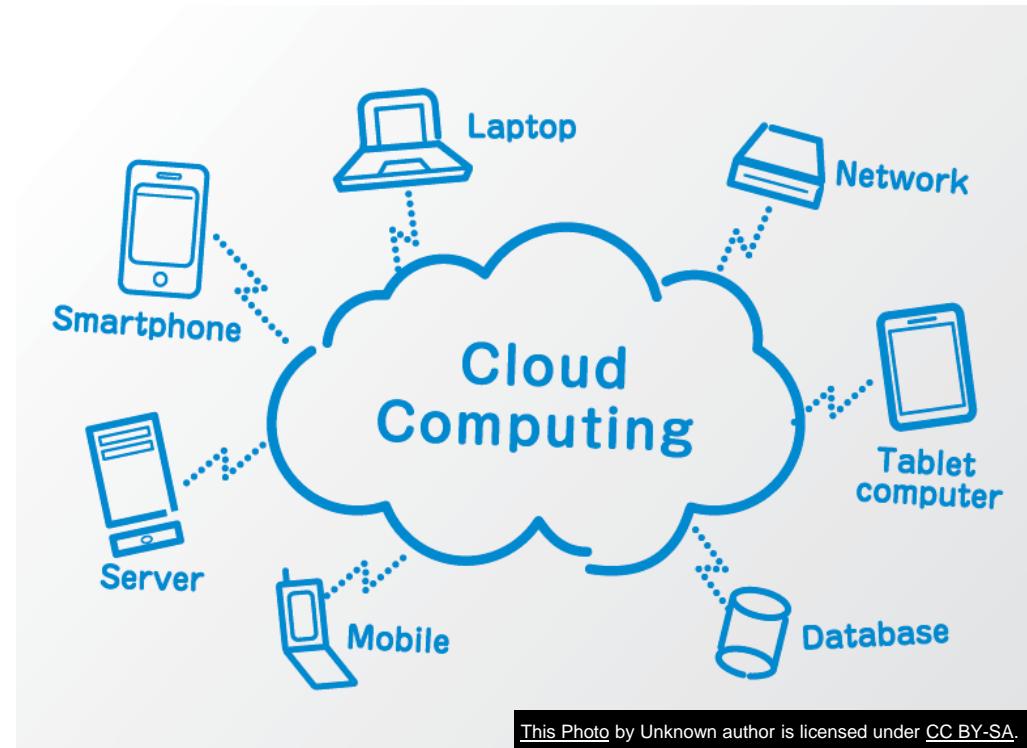
**[9 Hours]**

## References

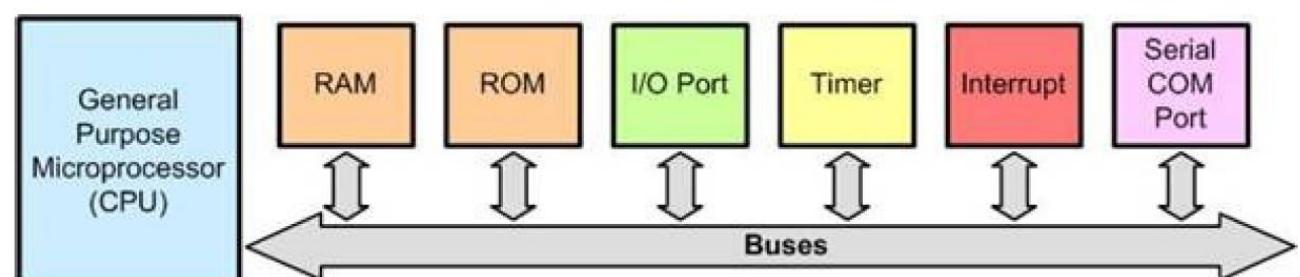
1. Muhammad Ali Mazidi, *Arm Assembly Language Programming & Architecture: Volume 1 (2e)*, Microdigitaled.com August 2016
2. Jonathan W.V., *Embedded systems: Real-time interfacing to ARM Cortex-M microcontrollers*, 8<sup>th</sup> Edition, ISBN: 978-1463590154 Createspace Independent Publishing Platform, July 2021.
3. Wilmshurst T., *Fast and Effective Embedded System Design applying the ARM mbed*, Elsevier, 2017.
4. Jonathan W.V., *Embedded systems: Introduction to Arm(r) Cortex-M Microcontrollers*, 6<sup>th</sup> Edition, ISBN: 978-1477508992, Createspace Independent publishing platform, Jan 2019.
5. UM10360, *LPC 176x/5x User Manual*, NXP Semiconductors, Rev. 3.1, 2014.
6. Joseph V., *A definitive Guide to ARM Cortex-M3 and Cortex-M4 processors*, 3<sup>rd</sup> Edition, Elsevier, 2014.
7. Muhammad A.M, Sarmad N., Sepehr N., Shujen C., *ARM Assembly Language Programming & Architecture*, 2<sup>nd</sup> Edition, Wiley, 2016.

# Introduction

---



- We have been brought up in the age of computing.
- Computers are everywhere (some we see, some we do not see).
- Types of computers we are familiar with:
  - Desktops and Laptops
  - Servers
  - Mobile phones



# Embedded System

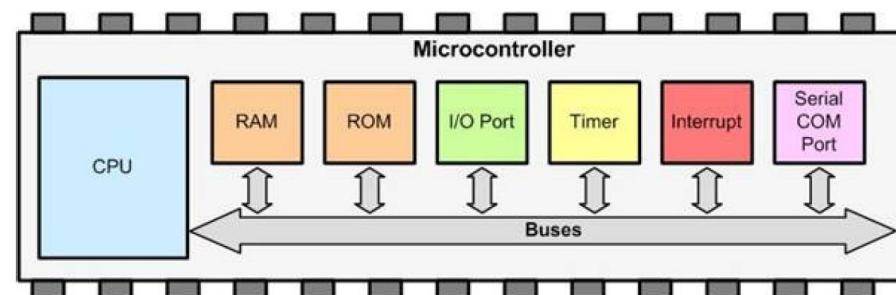


What are Embedded Systems?

- Computers are embedded within other systems:
- What is “other systems”? — Hard to define.
- Any computing system other than desktop / laptop server.
- Typical examples: Washing machine, refrigerator, camera, vehicles, airplane, missile, printer.

Processors are often very simple and inexpensive (depending on application of course).

Billions of embedded system units produced yearly, versus millions of desktop units.

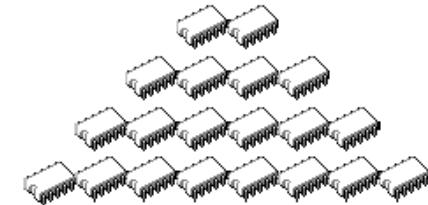


SoC (System on Chip)

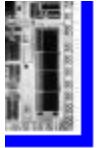


# Embedded systems overview

- Embedded computing systems
  - Computing systems embedded within electronic devices
  - Nearly any computing system other than a desktop computer
  - Billions of units produced yearly, versus millions of desktop units



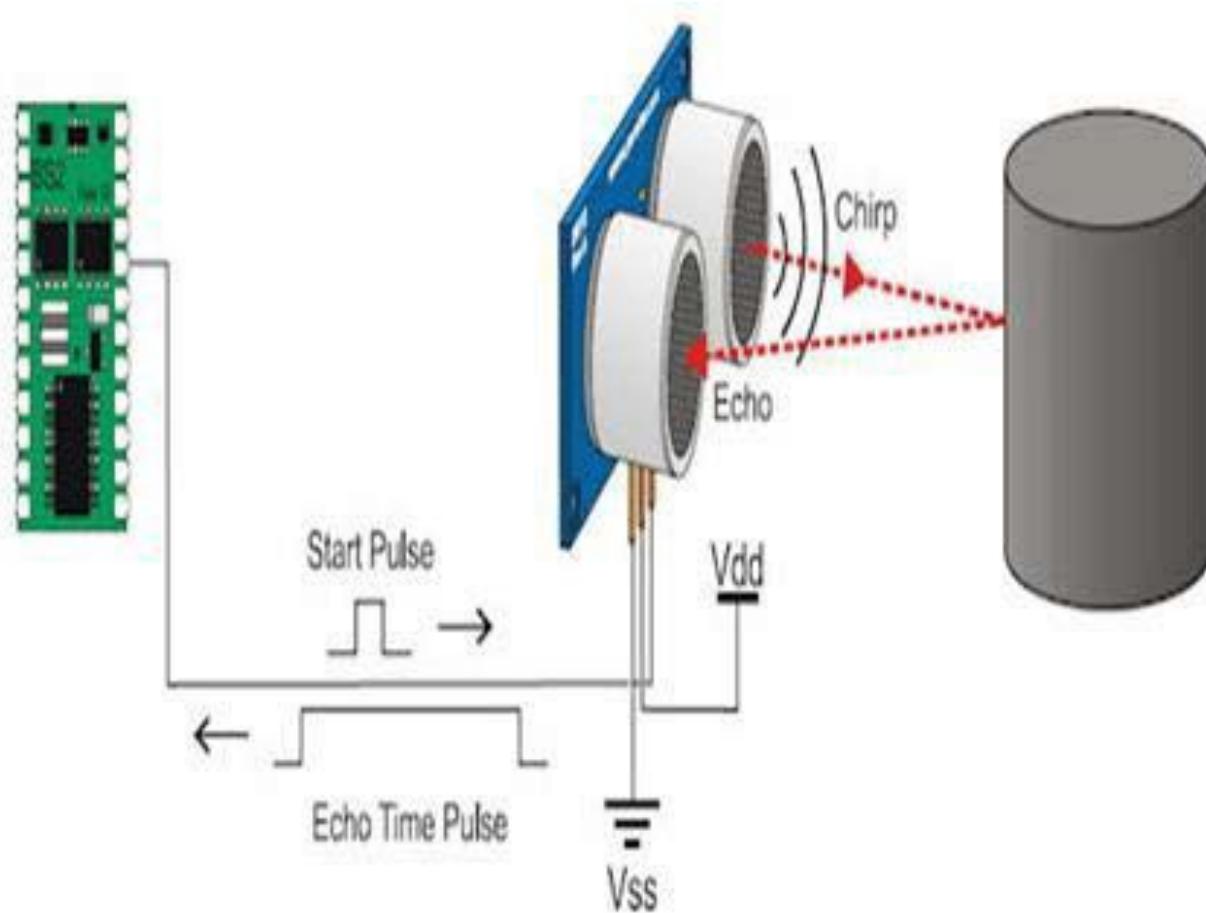
Lots more of these,  
though they cost a lot  
less each.



# Some common characteristics of embedded systems

- Single-functioned
  - Executes a single program, repeatedly
- Tightly-constrained
  - Low cost, low power, small, fast, etc.
- Reactive and real-time
  - Continually reacts to changes in the system's environment
  - Must compute certain results in real-time without delay

# Example



# The evolution of Microprocessors and Microcontrollers

---

In early computers, CPUs were designed using a number of vacuum tubes. The vacuum tube was bulky and consumed a lot of electricity.

The invention of transistors, followed by the IC (Integrated Circuit), provided the means to put a CPU on printed circuit boards.

Some of the microprocessors are the x86 family of Intel used widely in desktop computers, and the 68000 of Motorola

The microprocessors do not contain RAM, ROM, or I/O peripherals. As a result, they must be connected externally to RAM, ROM and I/O

# A Brief History of the Microcontrollers

---

In the 1980s and 1990s, Intel and Motorola dominated the field of microprocessors and microcontrollers

Intel had the x86 (8088/86, 80286, 80386, 80486, and Pentium).

Motorola (now Freescale) had the 68xxx (68000, 68010, 68020, etc.).

Many embedded systems used Intel's 32-bit chips of x86 (386, 486, Pentium)

Motorola's 32-bit 68xxx for high-end embedded products such as routers. Cisco routers used 68xxx for the CPU

With the introduction of PIC from Microchip and AVR from Atmel

---

In the late 1990s, the ARM microcontroller started to challenge the dominance of Intel and Motorola in the 32-bit market.

Although both Intel and Motorola used RISC features to enhance the performance of their microprocessors, due to the need to maintain compatibility with legacy software, they could not make a clean break and start over.

Intel used massive amounts of gates to keep up the performance of x86 architecture and that in turn increased the power consumption of the x86 to a level unacceptable for battery-powered embedded products.

# The ARM Family History

---

The ARM came out of a company called Acorn Computers in United Kingdom in the 1980s

Professor Steve Furber of Manchester University worked with Sophie Wilson to define the ARM architecture and instructions.

The VLSI Technology Corp. produced the first ARM chip in 1985 for Acorn Computers and was designated as Acorn RISC Machine (ARM).

Unable to compete with x86 (8088, 80286, 80386, ...) PCs from IBM and other personal computer makers

when Apple Corp. got interested in using the ARM chip for the PDA (personal digital assistants) products. This renewed interest in the chip led to the creation of a new company called ARM

---

Currently the ARM Corp. receives its entire revenue from licensing the ARM to other companies since it does not own state of the art chip fabrication facility

This business model of making money from selling IP (intellectual property) has made ARM one of the most widely used CPU architectures in the world.

# ARM and Apple

---

When Steve Jobs came back to run the Apple in 1996, the company was in decline

It had lost the personal computer race that had started 20 years earlier

The introduction of iPod in 2001 changed the fortune of that company more than anything else

Apple had tried to sell a PDA called Newton in the 1990s but was not successful

The iPod used an enhanced version of ARM called ARM7 and became an instant success. iPod brought the attention to the ARM chip that it deserved.

# ARM family variations

---

Although the ARM7 family is the most widely used version, ARM is determined to push the architecture into the low end of the microcontroller market where 8- and 16-bit microcontrollers have been traditionally dominating

For this reason they have come up with a microcontroller version of **ARM called Cortex.**

# One CPU many peripherals

---

ARM has defined the details of architecture, registers, instruction set, memory map, and timing of the ARM CPU and holds the copyright to it

The various design houses and semiconductor manufacturers license the IP (intellectual property) for the CPU and can add their own peripherals as they please

details of peripherals such as I/O ports, serial port UART, timer, ADC, SPI, DAC, I2C, and so on.

---

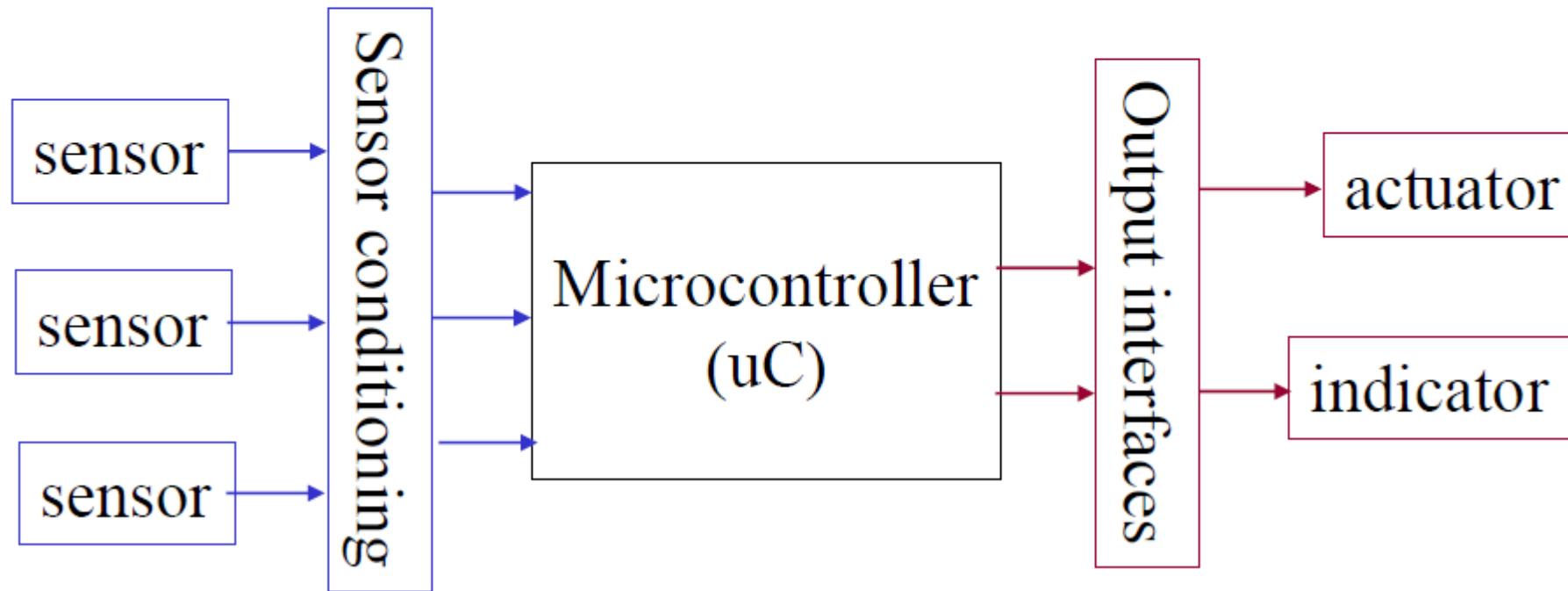
As a result while the CPU instructions and architecture are same across all the ARM chips made by different vendors, their peripherals are not compatible.

That means if you write a program for the serial port of an ARM chip made by TI (Texas Instrument), the program might not necessarily run on an ARM chip sold by NXP.

**This is the only drawback of the ARM microcontroller**

# Embedded System

## General Block Diagram



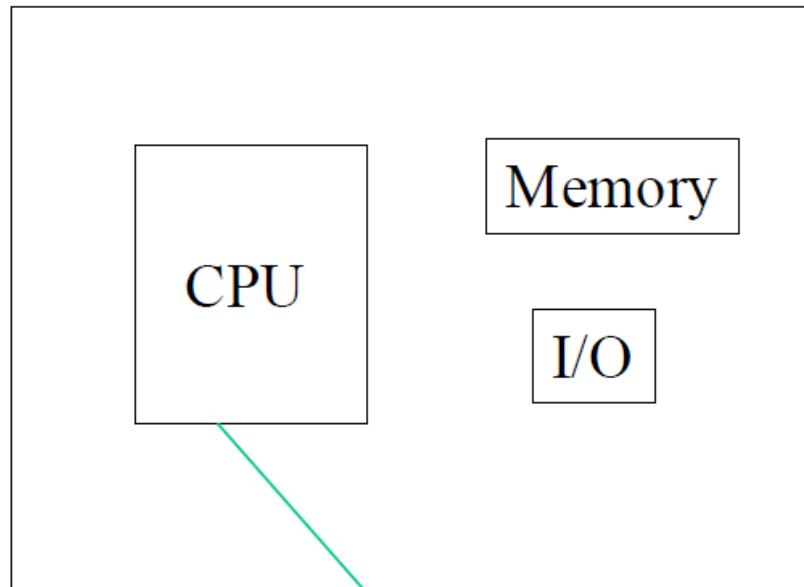
# Lecture-2



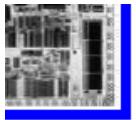
# Basic Components of Computer

- CPU
- Memory
- I/O

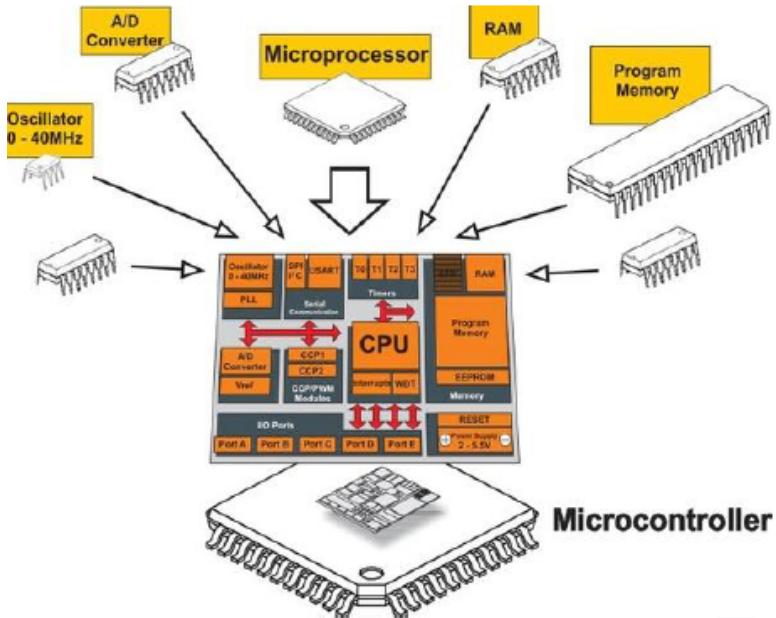
Motherboard



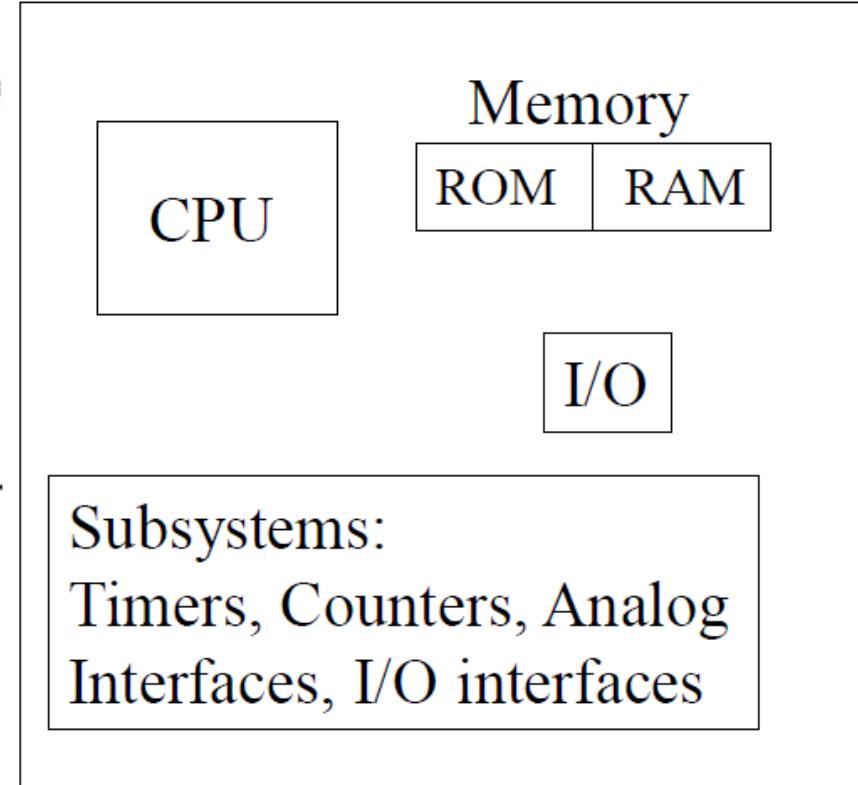
Microprocessor



# Microcontrollers



A single chip  
(SoC)



# Microprocessor vs Microcontroller

Sl.no	Microprocessor	Microcontroller
1	General purpose processor	Specific application controller
2	Contains no RAM, no ROM, no I/O ports on chip itself.	Contains RAM, ROM, I/O ports on chip itself
3	Size of RAM/ROM can vary	Size of RAM/ROM is fixed
4	Makes the system bulkier	Make the system compact
5	More expensive	Less expensive
6	It has less bit handling instructions	It has more bit handling instructions
7	Less number of pins have multiplexed functions	more number of pins have multiplexed functions
8	More flexible in designer point of view	Less flexible in designer point of view
9	Limited power saving options compared to microcontrollers	Includes lot of power saving features
10	Eg: Desktop PC, 8086,i7	Eg: Digital Camera,8051,msp430
11	Execution faster	Compared to µp slower
12	More general purpose registers	Less number of gen purpose registers
13	More addressing modes	Less addressing modes
14	Design time is more	Application design time less
15	Microprocessors are based on von Neumann model/architecture where program and data are stored in same memory module	Micro controllers are based on Harvard architecture where program memory and Data memory are separate
16	Cannot be used in compact systems and hence inefficient	Can be used in compact systems and hence it is an efficient technique
17	Example code: ADD AX,BX ADD AX,CX ADD AX,DX	Example code: MOV A, #2fh MOV B, #2fh ADD A,B
18	It cannot be used as stand alone	Can be used as stand alone.
19	May or may not be real-time application oriented	Real-time application oriented
20	Fig : a	Fig :b

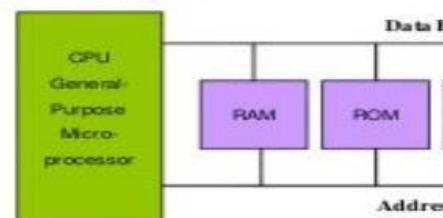


Fig : (a) Microprocessor

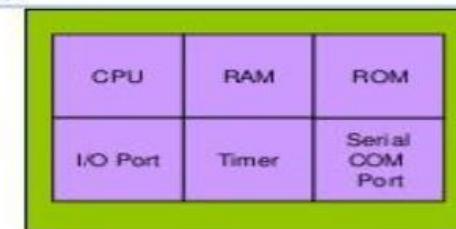
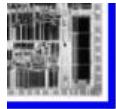
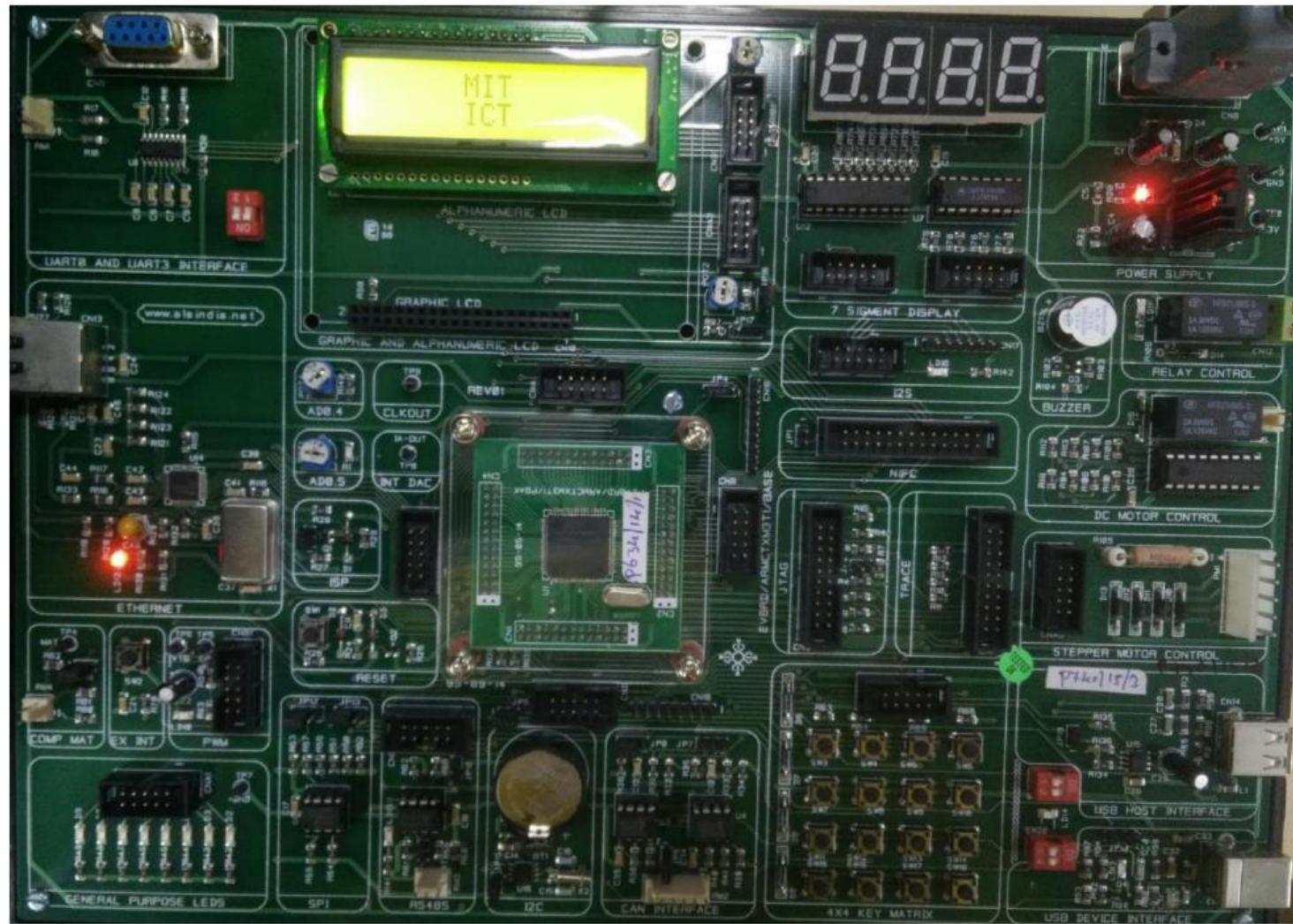


Fig : (b) Microcontroller

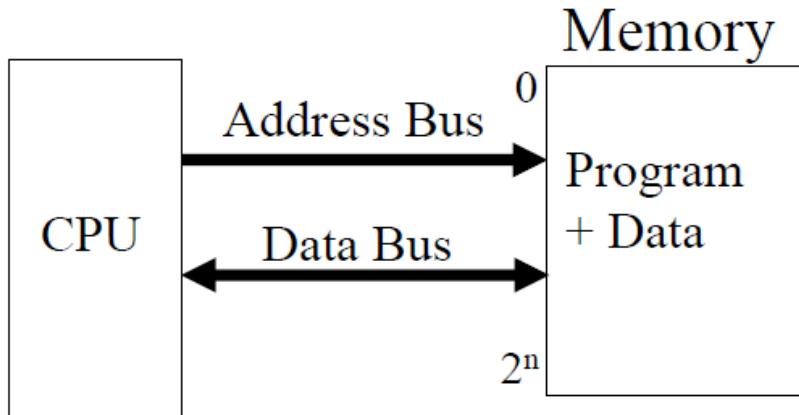


# LPC 1768

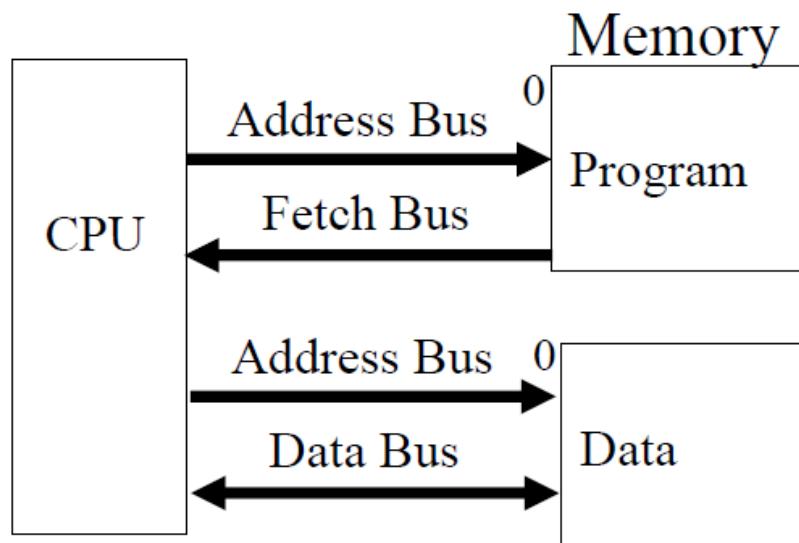




# Microcontroller Architectures

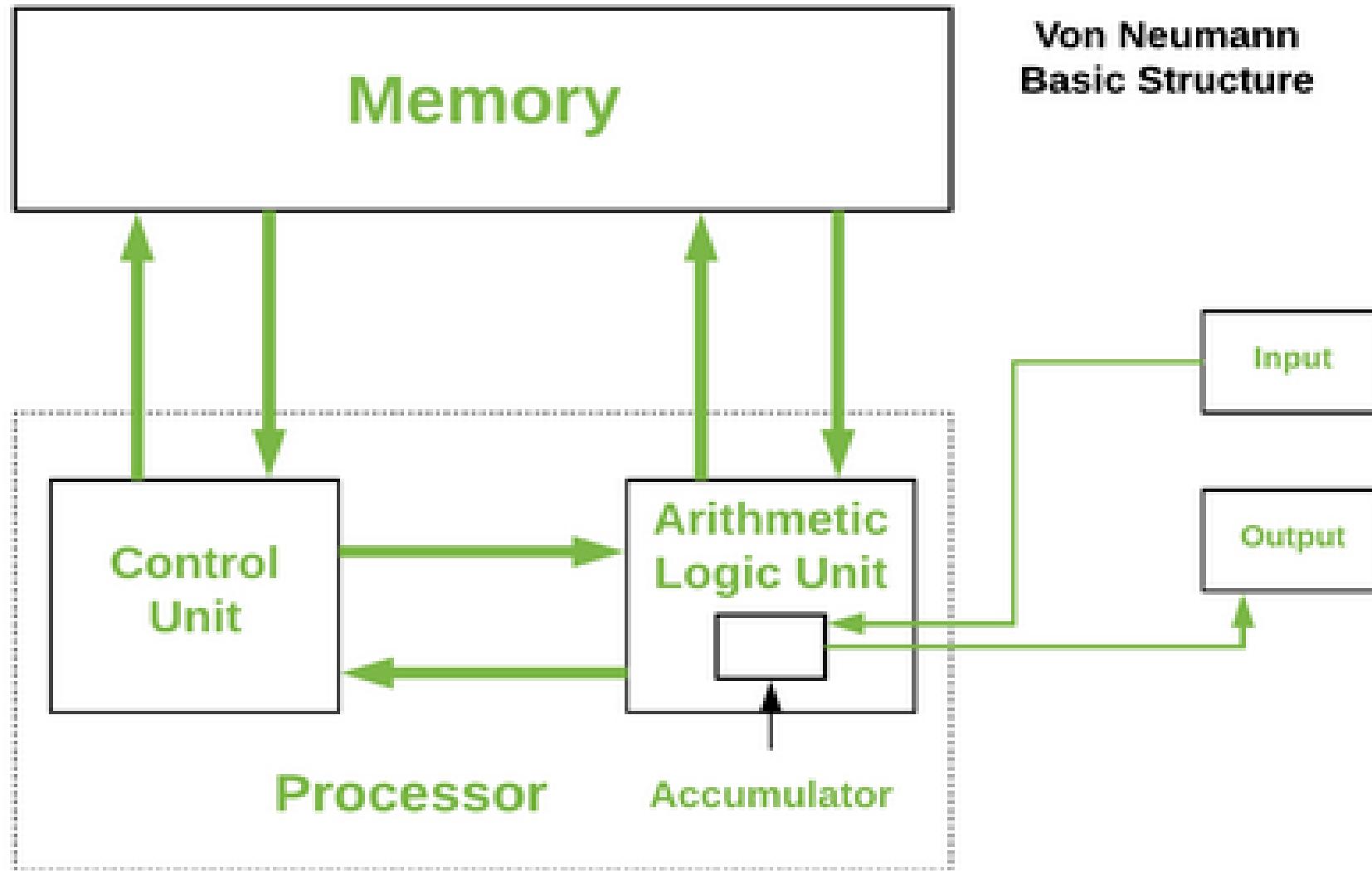


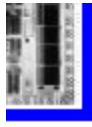
Von Neumann  
Architecture in CISC



Harvard  
Architecture in RISC

## Von Neumann Basic Structure





# RISC vs CISC

## What is RISC?

- A reduced instruction set computer is a computer that only uses simple commands that can be divided into several instructions that achieve low-level operation within a single CLK cycle, as its name proposes “Reduced Instruction Set”.

## What is CISC?

- A complex instruction set computer is a computer where single instructions can perform numerous low-level operations like a load from memory, an arithmetic operation, and a memory store or are accomplished by multi-step processes or addressing modes in single instructions, as its name proposes “Complex Instruction Set”.

# RISC vs CISC

RISC – Reduced Instruction Set Computer

CISC – Complex Instruction Set Computer

1. One of the major characteristics of RISC architecture is a large number of registers. All RISC architectures have at least 8 or 16 registers. Of these 16 registers, only a few are assigned to a dedicated function. One advantage of a large number of registers is that it avoids the need for a large stack to store parameters.
2. RISC processors have a fixed instruction size. In a CISC microprocessors such as the x86, instructions can be 1, 2, 3, or even 5 bytes. For example, look at the following instructions in the x86:

# RISC vs CISC

3. RISC processors have a small instruction set. RISC processors have only basic instructions such as ADD, SUB, MUL, LOAD, STORE, AND, OR, EOR, CALL, JUMP, and so on.

The limited number of instructions is one of the criticisms leveled at the RISC processor because it makes the job of Assembly language programmers much more tedious and difficult compared to CISC Assembly language programming.

This is one reason that RISC is used more commonly in high-level language environments such as the C programming language rather than Assembly language environments.

# RISC vs CISC

4. The most important characteristic of the RISC processor is that more than 99% of instructions are executed with only one clock cycle, in contrast to CISC instructions.
  
5. RISC processors have separate buses for data and code.

# RISC vs CISC

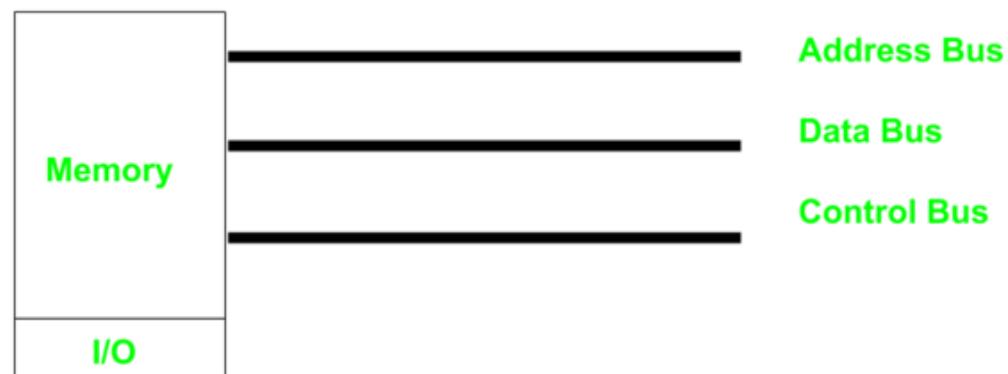
6. Because CISC has such a large number of instructions, each with so many different addressing modes, microinstructions (microcode) are used to implement them.

RISC instructions, however, due to the small set of instructions, are implemented using the hardwire method.

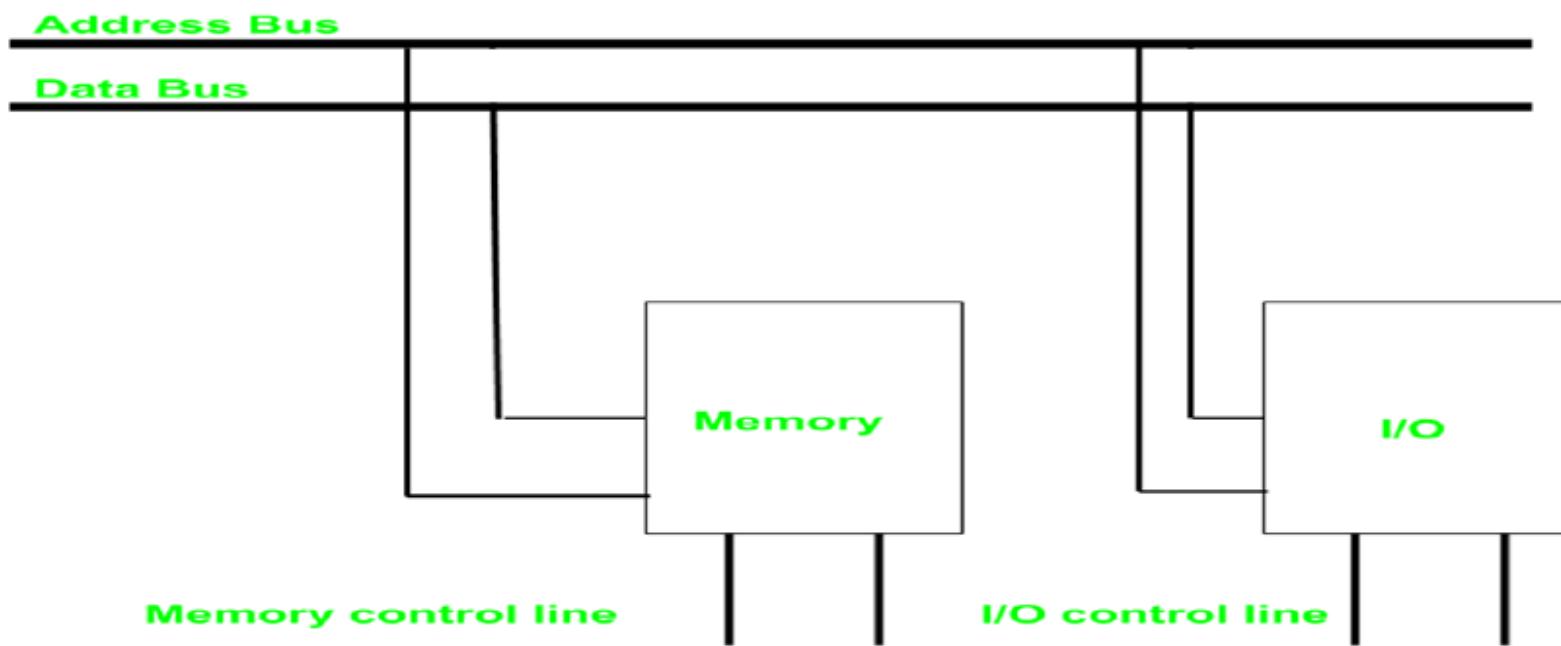
7. RISC uses load/ store architecture. In CISC microprocessors, data can be manipulated while it is still in memory.
8. Memory Mapped IO in RISC and IO mapped IO in CISC

RISC	CISC
<b>1. RISC stands for Reduced Instruction Set Computer.</b>	1. CISC stands for Complex Instruction Set Computer.
<b>2. RISC processors have simple instructions taking about takes few cycles</b>	2. CSIC processor has complex instructions that take up multiple clocks for execution. take 2-10 cycles.
<b>3. It has no memory unit and uses separate hardware to implement instructions..</b>	3. It has a memory unit to implement complex instructions.
<b>4. It has a hard-wired unit of programming.</b>	4. It has a microprogramming unit.
<b>5. The instruction set is reduced i.e. it has only a few instructions in the instruction set. Many of these instructions are very primitive.</b>	5. The instruction set has a variety of different instructions that can be used for complex operations.
<b>6 Supports few addressing modes.</b>	6. CISC has many different addressing modes and can thus be used to represent higher-level programming language statements more efficiently.
<b>7. Complex addressing modes are synthesized using the software.</b>	7. CISC already supports complex addressing modes
<b>8. Multiple register sets are present</b>	8. Only has a single register set
<b>9. RISC processors are highly pipelined</b>	9. They are normally not pipelined or less pipelined
<b>10. The complexity of RISC lies with the compiler that executes the program</b>	10. The complexity lies in the microprogram
<b>11. Execution time is very less</b>	11. Execution time is very high
<b>12. The decoding of instructions is simple.</b>	12. Decoding of instructions is complex
<b>13. It does not require external memory for calculations</b>	13. It requires external memory for calculations
<b>14. The most common RISC microprocessors are Alpha, ARC, ARM, AVR, MIPS, PA-RISC, PIC, Power Architecture, and SPARC.</b>	14. Examples of CISC processors are the System/360, VAX, PDP-11, Motorola 68000 family, AMD, and Intel x86 CPUs.
<b>15 RISC architecture is used in high-end applications such as video processing, telecommunications, and image processing.</b>	15. CISC architecture is used in low-end applications such as security systems, home automation, etc.

- **Memory Mapped I/O**
- In this case every bus is common due to which the same set of instructions work for memory and I/O.

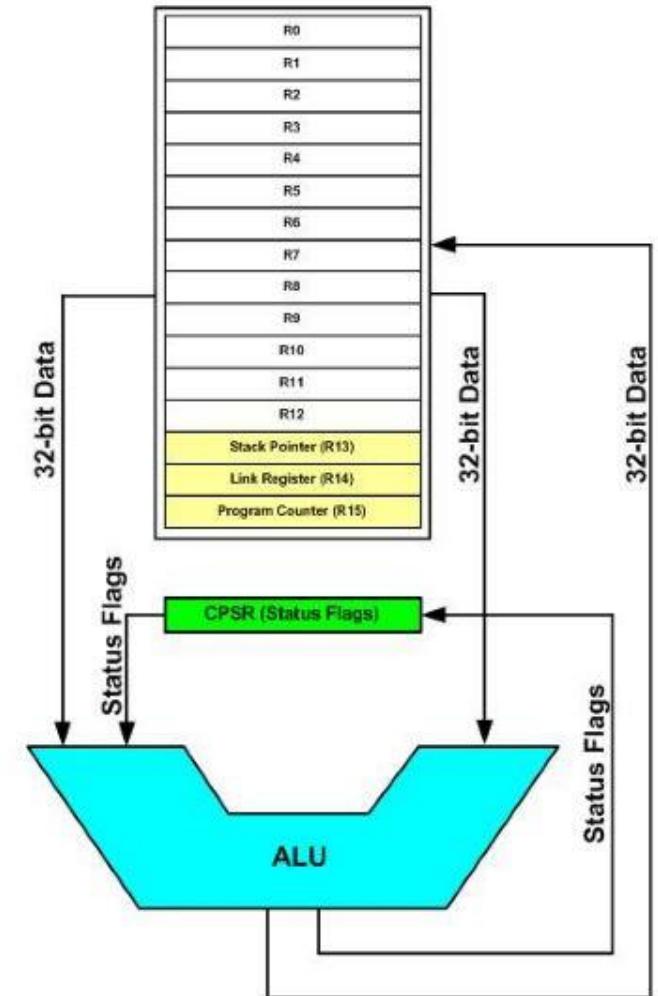
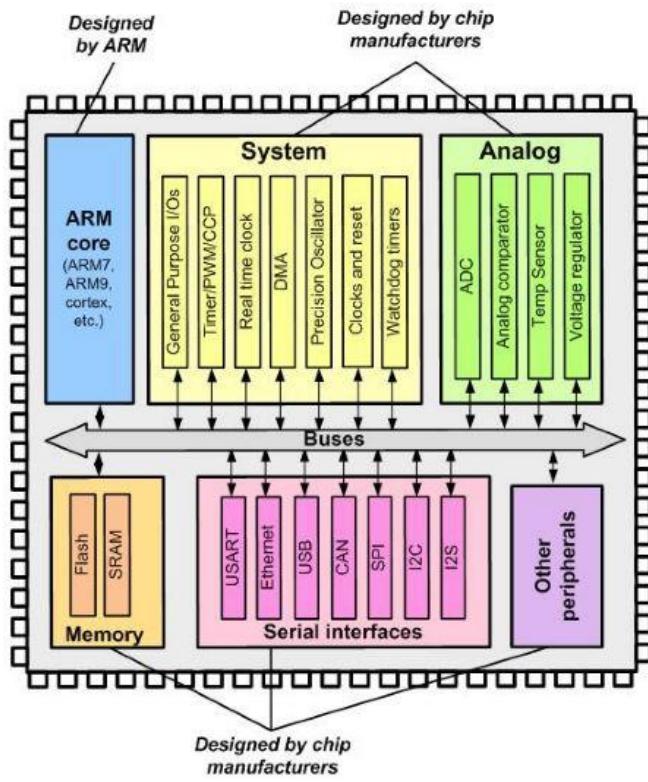


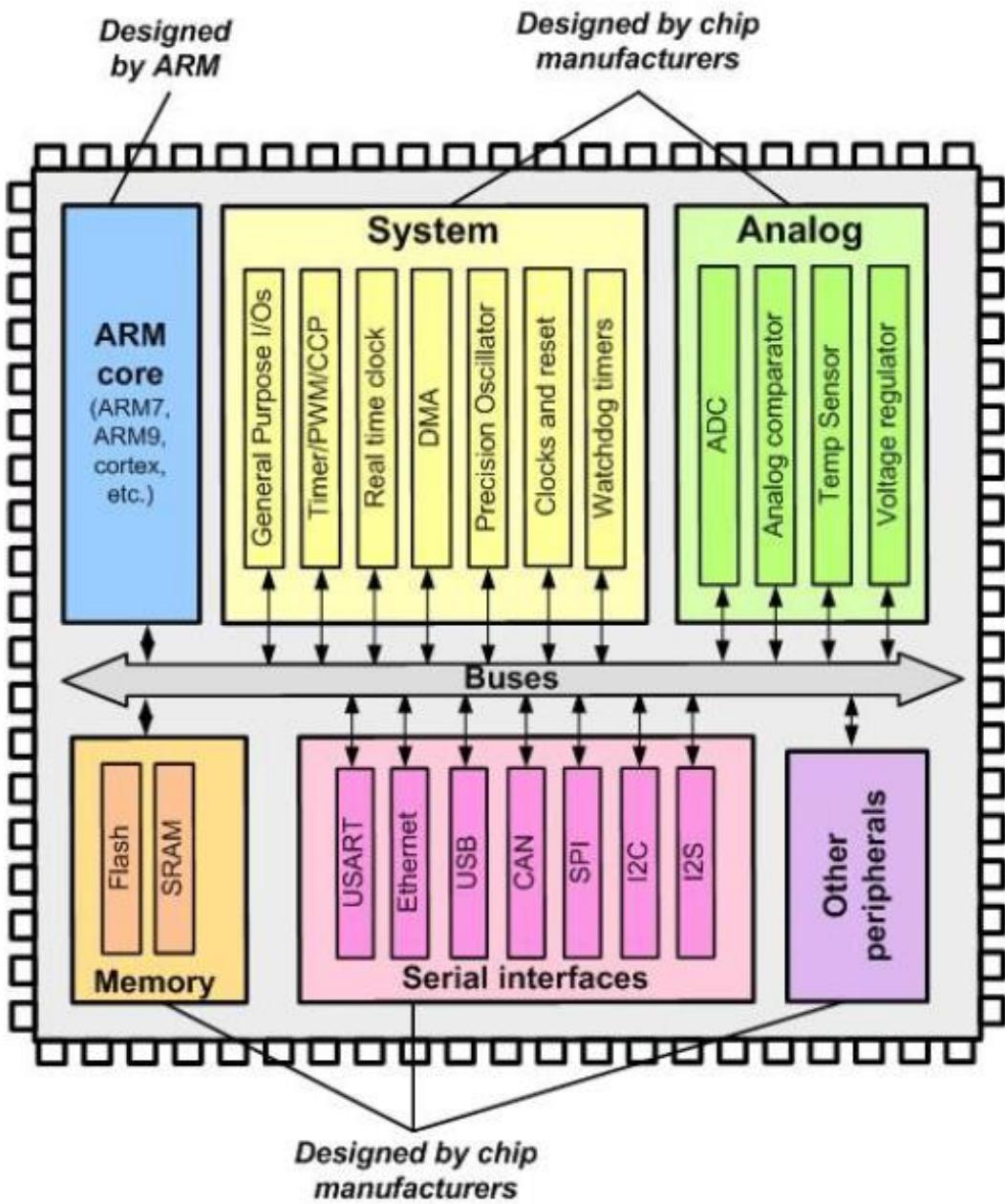
**IO mapped IO (Isolated I/O)** In which we have common bus(data and address) for I/O and memory but separate read and write control lines for I/O.



# ARM Architecture

N,Z,C,V flags  
Bit No. 31, 30, 29, 28





Actel	Analog Devices	Atmel
Broadcom	Cypress	Ember
Dust Networks	Energy	Freescale
Fujitso	Nuvoton	NXP
Renesas	Samsung	ST
Toshiba	Texas Instruments	Triad Semiconductor

Table 1- 3: ARM Vendors

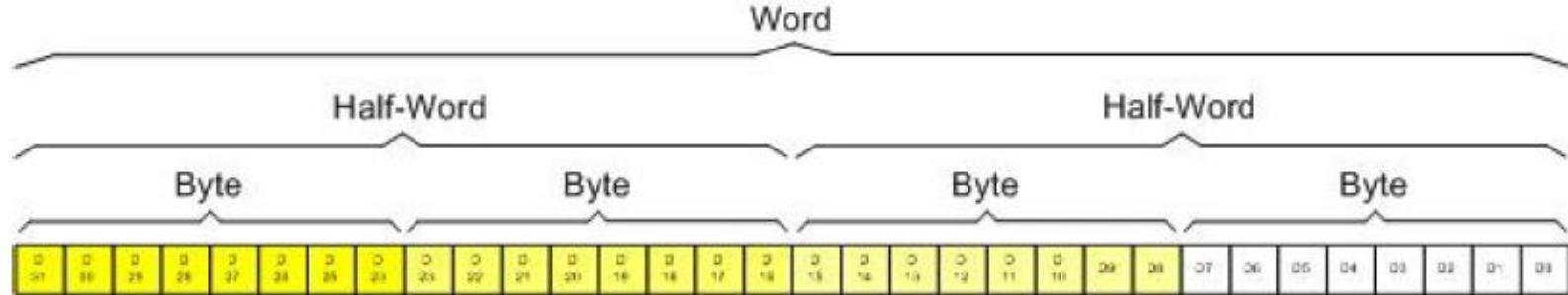


Figure 2- 1: ARM Registers Data Size

Name	Functions (and banked registers)
R0	General-purpose register
R1	General-purpose register
R2	General-purpose register
R3	General-purpose register
R4	General-purpose register
R5	General-purpose register
R6	General-purpose register
R7	General-purpose register
R8	General-purpose register
R9	General-purpose register
R10	General-purpose register
R11	General-purpose register
R12	General-purpose register
R13 (MSP)	Main Stack Pointer (MSP), Process Stack Pointer (PSP)
R13 (PSP)	
R14	Link Register (LR)
R15	Program Counter (PC)

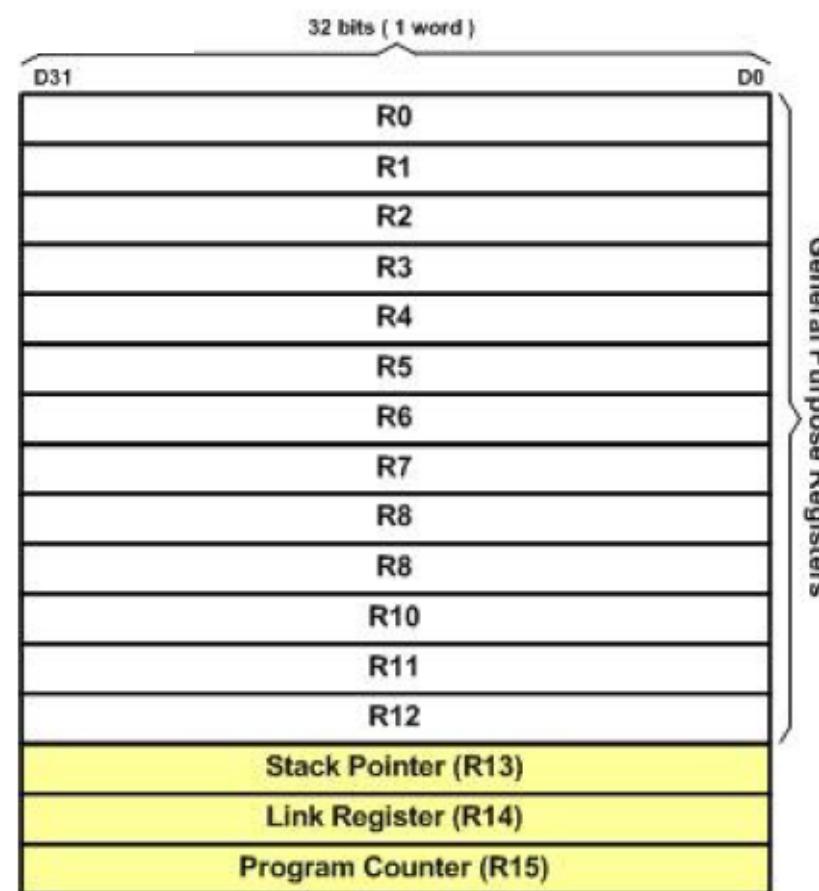


Figure 2- 2: ARM Registers

N,Z,C,V flags

Bit No. 31, 30, 29, 28

Name	Functions	
xPSR	Program status registers	
PRIMASK		
FAULTMASK	Interrupt mask registers	
BASEPRI		
CONTROL	Control register	

The Cortex-M3 processor also has a number of special registers. They are as follows:

1. Program Status registers (PSRs)
2. Interrupt Mask registers (PRIMASK, FAULTMASK, and BASEPRI)
3. Control register (CONTROL)

# Special function registers in ARM

---

In ARM the R13, R14, R15, and CPSR (current program status register) registers are called **SFRs**  
*(special function registers)*

*R13-> Stack pointer*

*R14-> Link register*

*R15-> Program counter*

*CPSR-> stores the status of the program after execution of a given instruction.*

Since each one is dedicated to a specific function.

# Program Counter in the ARM

---

One of the most important register in the ARM microcontroller is the PC (program counter) .

the R15 is the program counter

The program counter is used by the CPU to point to the address of the next instruction to be executed

As the CPU fetches the opcode from the program memory, the program counter is incremented automatically to point to the next instruction

The wider the program counter, the more memory locations a CPU can access

a 32-bit program counter can access a maximum of 4G ( $2^{32} = 4G$ ) bytes of program memory locations

# Memory space allocation in the ARM

---

The ARM has 4G bytes of directly accessible memory space. This memory space has addresses 0 to 0xFFFFFFFF.

The 4G bytes of memory space can be divided into five sections.

**On-chip peripheral and I/O registers:**

**On-chip data SRAM:**

**On-chip EEPROM**

**On-chip Flash ROM**

**Off-chip DRAM space**

# Memory address calculation

---

Starting address is calculated by writing all zero

Ending address by writing all 1s

Finally represent it in Hexadecimal

example

# On-chip peripheral and I/O registers

---

This area is dedicated to general purpose I/O (GPIO) and special function registers (SFRs) of peripherals

such as timers serial communication, ADC, and so on.

In other words, ARM uses memory-mapped I/O.

The function and address location of each SFR is fixed by the chip vendor at the time of design because it is used for port registers of peripherals

# On-chip data SRAM

---

A RAM space ranging from a few kilobytes to several hundred kilobytes is set aside mainly **for data storage**

Even within the same family, the size of the data SRAM space varies from chip to chip.

The data RAM space is used for data variables and stack and is accessed by the microcontroller instructions.

one can also buy or design an ARM-based system in which the RAM space is used for both data and program codes. Example: x86 PCs

Microsoft Windows 8 uses such a system for ARM-based Tablet computers.

The data RAM space is read/write memory used by the CPU for storage of data variables, and stack

# On-chip EEPROM

---

A block of memory from 1K bytes to several thousand bytes is set aside for EEPROM memory  
EEPROM used most often for saving critical data.

# On-chip Flash ROM

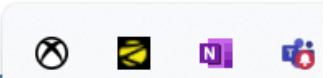
---

A block of memory from a few kilobytes to several hundred kilobytes is set aside for program space

The program space is used for the program code.

The Flash memory of code ROM is under the control of the PC (program counter)

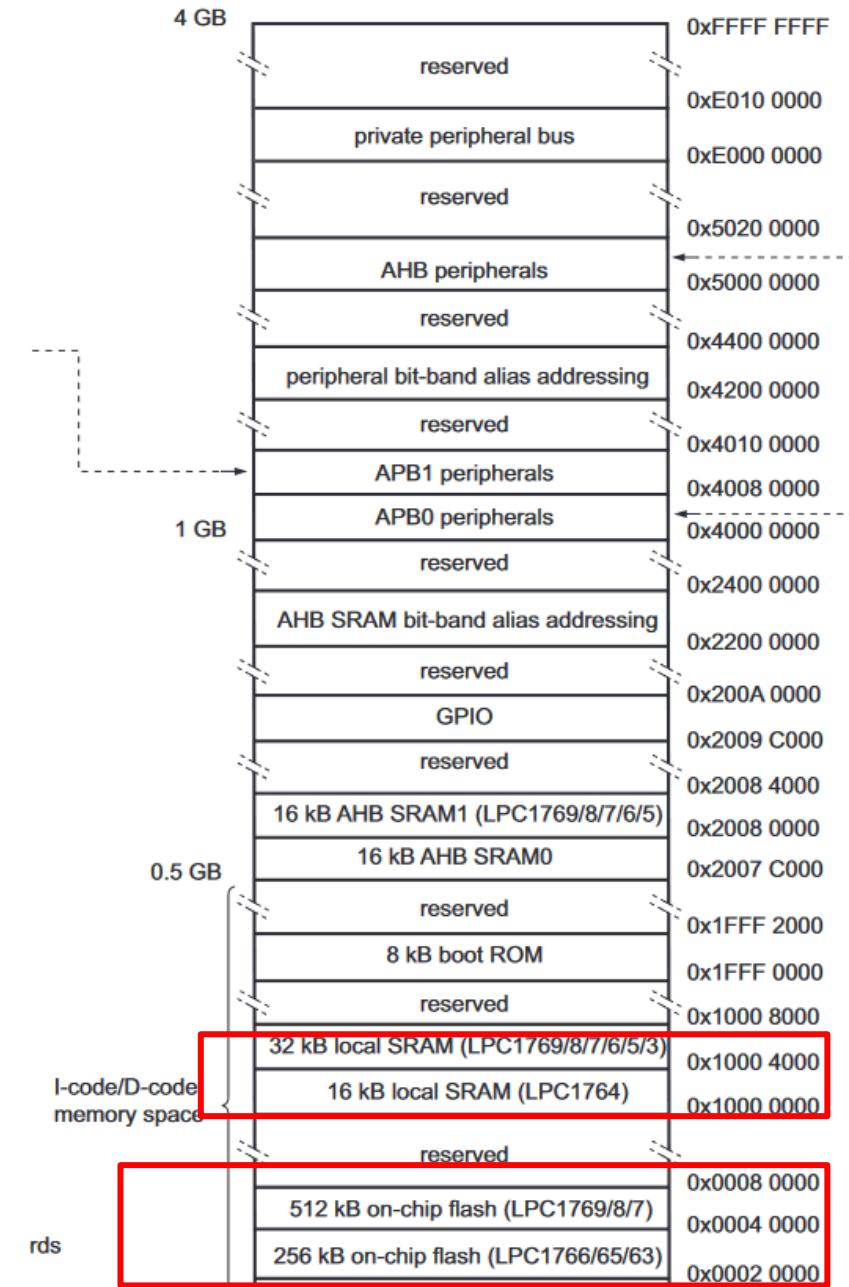
Directive	Description
<b>AREA</b>	Instructs the assembler to assemble a new code or data section
<b>END</b>	Informs the assembler that it has reached the end of a source file.
<b>ENTRY</b>	Declares an entry point to a program.
<b>EQU</b>	Gives a symbolic name to a numeric constant, a register-relative value or a PC-relative value.
<b>INCLUDE</b>	It adds the contents of a file to our program.



# Lecture-3

# Memory

- On-chip **Flash memory** system
  - Up to 512 kB of on-chip flash memory
  - Flash memory accelerator maximizes performance for use with the two fast advanced high-performance bus AHB-Lite buses
  - Can be used for both code and data storage
- On-chip Static RAM (**SRAM**)
  - Up to 64 kB of on-chip static RAM memory
  - Up to 32 kB of SRAM, accessible by the CPU and all three DMA (direct memory access) controllers are on a higher-speed bus
  - Devices with more than 32 kB SRAM have two additional 16 kB SRAM blocks



## Memory 1

Address: 0x10000000

0x10000000:	00 00 00 00
0x10000004:	00 00 00 00
0x10000008:	00 00 00 00
0x1000000C:	00 00 00 00
<b>0x10000010:</b>	<b>67 45 23 01</b>
0x10000014:	00 00 00 00
0x10000018:	00 00 00 00
0x1000001C:	00 00 00 00
0x10000020:	00 00 00 00
0x10000024:	00 00 00 00
0x10000028:	00 00 00 00
0x1000002C:	00 00 00 00
0x10000030:	00 00 00 00
0x10000034:	00 00 00 00
0x10000038:	00 00 00 00
<b>0x1000003C:</b>	<b>00 00 00 00</b>
0x10000040:	00 00 00 00
0x10000044:	00 00 00 00
0x10000048:	00 00 00 00
0x1000004C:	00 00 00 00
0x10000050:	00 00 00 00

Data: 0x01234567

0x00000010	0x00000011	0x00000012	0x00000013
------------	------------	------------	------------

Big Endian

Data: 0x01234567

0x00000010	0x00000011	0x00000012	0x00000013
------------	------------	------------	------------

Little Endian

Little Endian

Big  
Endian

0x10000000:	00000000
0x10000004:	00000000
0x10000008:	00000000
0x1000000C:	00000000
<b>0x10000010:</b>	<b>01234567</b>
0x10000014:	00000000
0x10000018:	00000000
0x1000001C:	00000000
0x10000020:	00000000
0x10000024:	00000000
0x10000028:	00000000
0x1000002C:	00000000
0x10000030:	00000000
0x10000034:	00000000
0x10000038:	00000000
<b>0x1000003C:</b>	<b>00 00 00 00</b>
0x10000040:	00000000
0x10000044:	00000000
0x10000048:	00000000
0x1000004C:	00000000
0x10000050:	00000000

Data: 0x01234567

0x000000010

0x000000011

0x000000012

0x000000013



Big Endian

Data: 0x67452301

0x000000013

0x000000012

0x000000011

0x000000010



Big Endian

Data: 0x67452301

0x000000013

0x000000012

0x000000011

0x000000010



Little Endian

Data: 0x01234567

0x000000010

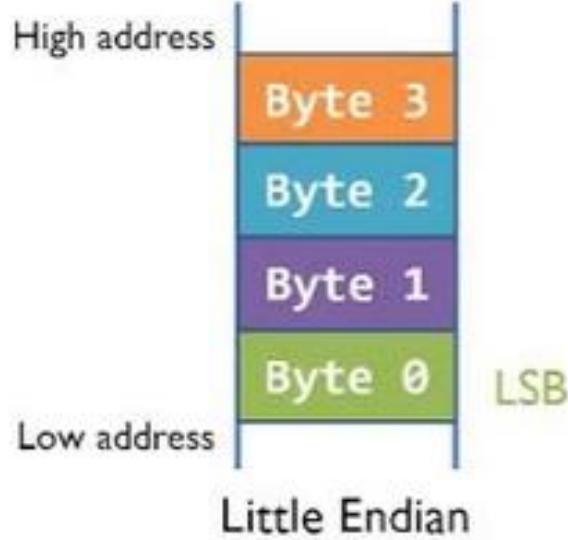
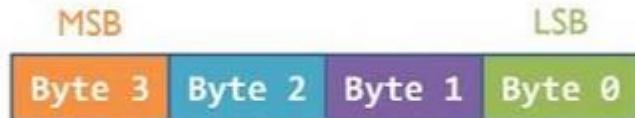
0x000000011

0x000000012

0x000000013



Little Endian



*LSB is at least address!*



*MSB is at least address!*

## Memory space allocation in the ARM

The ARM has 4G bytes of directly accessible memory space. This memory space has addresses 0 to 0xFFFFFFFF. The 4G bytes of memory space can be divided into five sections. They are as follows:

- **On-chip peripheral and I/O registers:** This area is dedicated to general-purpose I/O (GPIO) and special function registers (SFRs)
- **On-chip data SRAM:** A RAM space ranging from a few kilobytes to several hundred kilobytes is set aside mainly for data storage.
- **On-chip EEPROM:** A block of memory from 1K bytes to several thousand bytes is set aside for EEPROM memory.
- **On-chip Flash ROM:** A block of memory from a few kilobytes to several hundred kilobytes is set aside for program space.
- **Off-chip DRAM space:** A DRAM memory ranging from few megabytes to several hundred mega bytes can be implemented for external memory connection.

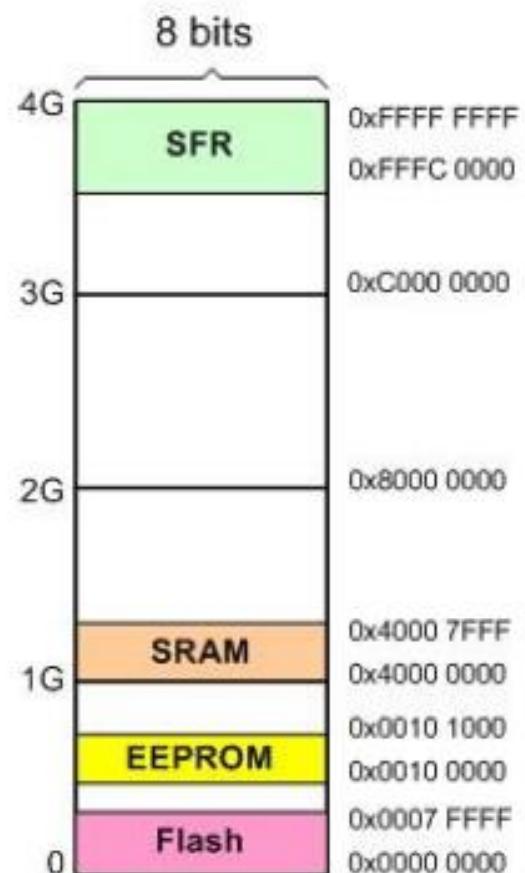
### **Example 2-1**

A given ARM chip has the following address assignments. Calculate the space and the amount of memory given to each section.

- (a) Address range of 0x00100000 – 0x00100FFF for EEPROM
- (b) Address range of 0x40000000 – 0x40007FFF for SRAM
- (c) Address range of 0x00000000 – 0x0007FFFF for Flash
- (d) Address range of 0xFFFFC0000 – 0xFFFFFFFF for peripherals

**Solution:**

- (a) With address space of 0x00100000 to 00100FFF, we have  $00100FFF - 00100000 = 0FFF$  bytes. Converting 0FFF to decimal, we get  $4,095 + 1$ , which is equal to 4K bytes.
- (b) With address space of 0x40000000 to 0x40007FFF, we have  $40007FFF - 40000000 = 7FFF$  bytes. Converting 7FFF to decimal, we get  $32,767 + 1$ , which is equal to 32K bytes.
- (c) With address space of 0000 to 7FFFF, we have  $7FFFF - 0 = 7FFFF$  bytes. Converting 7FFFF to decimal, we get  $524,287 + 1$ , which is equal to 512K bytes.
- (d) With address space of FFFC0000 to FFFFFFFF, we have  $FFFFFFF - FFFC0000 = 3FFF$  bytes. Converting 3FFF to decimal, we get  $262,143 + 1$ , which is equal to 256K bytes.



## Initial Instructions for Programming

```
AREA reset, DATA, READONLY
EXPORT __Vectors
__Vectors
    DCD 0x10001000
    DCD Reset_Handler
    ALIGN
    AREA mycode, CODE, READONLY
    EXPORT Reset_Handler
    ENTRY
Reset_Handler
    LDR R1, N1
STOP
    B STOP
N1 DCD 0x20000001
END
```

The screenshot shows a debugger interface with two main panes: Registers and Disassembly.

**Registers Pane:**

Register	Value
R0	0x00000000
R1	0x20000001
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x0000000A
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	2
Sec	0.00000017

**Disassembly Pane:**

```
13: B STOP
>0x0000000A E7FE B 0x0000000A

P1_1.s
1 AREA reset, DATA, READONLY
2 EXPORT __Vectors
3 __Vectors
4 DCD 0x10001000
5 DCD Reset_Handler
6 ALIGN
7 AREA mycode, CODE, READONLY
8 EXPORT Reset_Handler
9 ENTRY
10 Reset_Handler
11 LDR R1, N1
12 STOP
13 B STOP
14 N1 DCD 0x20000001
15 END
16
```

# Addressing Modes

The way in which an operand is specified in an instruction is called Addressing Mode

1. Register - MOV R0, R1
2. Immediate – MOV R0, #1
3. Register Indirect- LDR R0, [R1]  
STR R0, [R1]
4. Indexed

Indexed Addressing Mode	Syntax	Pointing Location in Memory	Rm Value After Execution
<b>Preindex</b>	LDR Rd, [Rm,#k]	Rm+#k	Rm
<b>Preindex with WB*</b>	LDR Rd, [Rm,#k]!	Rm + #k	Rm + #k
<b>Postindex</b>	LDR Rd, [Rm],#k	Rm	Rm + #k

\*WB means Writeback  
\*\*Rd and Rm are any of registers and #k is a signed 12-bit immediate value between -4095 and +4095

```
1      AREA RESET, DATA, READONLY
2      EXPORT __Vectors
3 __Vectors
4      DCD 0x10001000
5      DCD Reset_Handler
6      ALIGN
7      AREA mycode, CODE, READONLY
8      ENTRY
9      EXPORT Reset_Handler
10 Reset_Handler
11      MOV R0, #0x23
12      MOV R1, #23
13      MOV R2, #2_10101
14      MOV R3, #4_123
15 STOP B STOP
16 END
```

```
1      AREA RESET, DATA, READONLY
2      EXPORT __Vectors
3 __Vectors
4      DCD 0x10001000
5      DCD Reset_Handler
6      ALIGN
7      AREA mycode, CODE, READONLY
8      ENTRY
9      EXPORT Reset_Handler
10 Reset_Handler
11      MOV R0, #0x23
12      MOV R1, #23
13      MOV R2, #2_10101
14      MOV R3, #4_123
15 STOP B STOP
16      END
17
```

Register	Value
Core	
R0	0x00000023
R1	0x00000017
R2	0x00000015
R3	0x0000001B
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x00000018
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP

Proj1.s

```
1 AREA RESET, DATA, READONLY
2 EXPORT __Vectors
3 __Vectors
4 DCD 0x10001000
5 DCD Reset_Handler
6 ALIGN
7 AREA mycode, CODE, READONLY
8 ENTRY
9 EXPORT Reset_Handler
10 Reset Handler
11 MOV R0, #0x23
12 MOV R1, #-0x23
13 MOV R2, #0x345
14 MOV R3, #0x4567
15 MOV R4, #-0x345
16 STOP B STOP
17 END
```

Registers

Register	Value
Core	
R0	0x00000023
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x0000000C
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Starlr	MSP

```
1 AREA RESET, DATA, READONLY
2 EXPORT __Vectors
3 __Vectors
4 DCD 0x10001000
5 DCD Reset_Handler
6 ALIGN
7 AREA mycode, CODE, READONLY
8 ENTRY
9 EXPORT Reset_Handler
10 Reset_Handler
11 MOV R0, #0x23
12 MOV R1, #-0x23
13 MOV R2, #0x345
14 MOV R3, #0x4567
15 MOV R4, #-0x345
16 STOP B STOP
17 END
```

Register	Value
Core	
R0	0x00000023
R1	0xFFFFFD
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x00000010
+ xPSR	0x01000000
+ Banked	
+ System	
- Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP

Proj1.s

```

1 AREA RESET, DATA, READONLY
2 EXPORT __Vectors
3 __Vectors
4 DCD 0x10001000
5 DCD Reset_Handler
6 ALIGN
7 AREA mycode, CODE, READONLY
8 ENTRY
9 EXPORT Reset_Handler
Reset_Handler
10
11 MOV R0, #0x23
12 MOV R1, #-0x23
13 MOV R2, #0x345
14 MOV R3, #0x4567
15 MOV R4, #-0x345
16 STOP B STOP
17 END
18

```

Registers

Register	Value
Core	
R0	0x00000023
R1	0xFFFFFFFDD
R2	0x00000345
R3	0x00004567
<b>R4</b>	<b>0xFFFFFCBB</b>
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFF
<b>R15 (PC)</b>	<b>0x0000001C</b>
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP

Project | Registers

Proj1.s

```
1 AREA RESET, DATA, READONLY
2 EXPORT __Vectors
3 __Vectors
4 DCD 0x10001000
5 DCD Reset_Handler
6 ALIGN
7 AREA mycode, CODE, READONLY
8 ENTRY
9 EXPORT Reset_Handler
10 Reset_Handler
11 LDR R0, SRC
12 LDR R1, =0x23456780
13 LDR R2, =SRC
14 LDR R3, [R2]
15 STOP B STOP
16 SRC DCD 0x12345678
17 END
```

Registers

Register	Value
R0	0x12345678
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x0000000A
xPSR	0x01000000

Memory 1

Address	Value
0x00000000	00 10 00 10
0x00000004	09 00 00 00
0x00000008	02 48 03 49
0x0000000C	03 4A 13 68
0x00000010	FE E7 00 00
0x00000014	78 56 34 12
0x00000018	80 67 45 23
0x0000001C	14 00 00 00
0x00000020	00 00 00 00
0x00000024	00 00 00 00
0x00000028	00 00 00 00
0x0000002C	00 00 00 00
0x00000030	00 00 00 00
0x00000034	00 00 00 00
0x00000038	00 00 00 00



# Embedded System and Design

---

**DR MANOJ TOLANI (PHD-IIIT ALLAHABAD)**  
**ASSISTANT PROFESSOR (DEPARTMENT OF ICT)**

# Addressing Modes

The way in which an operand is specified in an instruction is called Addressing Mode

1. Register - MOV R0, R1
2. Immediate – MOV R0, #1
3. Register Indirect- LDR R0, [R1]  
STR R0, [R1]
4. Indexed

Indexed Addressing Mode	Syntax	Pointing Location in Memory	Rm Value After Execution
<b>Preindex</b>	LDR Rd, [Rm,#k]	Rm+#k	Rm
<b>Preindex with WB*</b>	LDR Rd, [Rm,#k]!	Rm + #k	Rm + #k
<b>Postindex</b>	LDR Rd, [Rm],#k	Rm	Rm + #k

\*WB means Writeback  
\*\*Rd and Rm are any of registers and #k is a signed 12-bit immediate value between -4095 and +4095

Proj1.s

```
1 AREA RESET, DATA, READONLY
2 EXPORT __Vectors
3 __Vectors
4 DCD 0x10001000
5 DCD Reset_Handler
6 ALIGN
7 AREA mycode, CODE, READONLY
8 ENTRY
9 EXPORT Reset_Handler
10 Reset_Handler
11 LDR R0, SRC
12 LDR R1, =0x23456780
13 LDR R2, =SRC
14 LDR R3, [R2]
15 STOP B STOP
16 SRC DCD 0x12345678
17 END
```

Registers

Register	Value
R0	0x12345678
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x0000000A
xPSR	0x01000000

Memory 1

Address	Value
0x00000000	00 10 00 10
0x00000004	09 00 00 00
0x00000008	02 48 03 49
0x0000000C	03 4A 13 68
0x00000010	FE E7 00 00
0x00000014	78 56 34 12
0x00000018	80 67 45 23
0x0000001C	14 00 00 00
0x00000020	00 00 00 00
0x00000024	00 00 00 00
0x00000028	00 00 00 00
0x0000002C	00 00 00 00
0x00000030	00 00 00 00
0x00000034	00 00 00 00
0x00000038	00 00 00 00

Proj1.s

```
1 AREA RESET, DATA, READONLY
2 EXPORT __Vectors
3 __Vectors
4 DCD 0x10001000
5 DCD Reset_Handler
6 ALIGN
7 AREA mycode, CODE, READONLY
8 ENTRY
9 EXPORT Reset_Handler
10 Reset_Handler
11 LDR R0, SRC
12 LDR R1, =0x23456780
13 LDR R2, =SRC
14 LDR R3, [R2]
15 STOP B STOP
16 SRC DCD 0x12345678
17 END
```

Registers

Register	Value
R0	0x12345678
R1	0x23456780
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x0000000C
xPSR	0x01000000

Memory 1

Address	Value
0x00000000	00 10 00 10
0x00000004	09 00 00 00
0x00000008	02 48 03 49
0x0000000C	03 4A 13 68
0x00000010	FE E7 00 00
0x00000014	78 56 34 12
0x00000018	80 67 45 23
0x0000001C	14 00 00 00
0x00000020	00 00 00 00
0x00000024	00 00 00 00
0x00000028	00 00 00 00
0x0000002C	00 00 00 00
0x00000030	00 00 00 00
0x00000034	00 00 00 00
0x00000038	00 00 00 00

Proj1.s

```

1 AREA RESET, DATA, READONLY
2 EXPORT __Vectors
3 __Vectors
4 DCD 0x10001000
5 DCD Reset_Handler
6 ALIGN
7 AREA mycode, CODE, READONLY
8 ENTRY
9 EXPORT Reset_Handler
10 Reset_Handler
11 LDR R0, SRC
12 LDR R1, =0x23456780
13 LDR R2, =SRC
14 LDR R3, [R2]
15 STOP B STOP
16 SRC DCD 0x12345678
17 END

```

Register	Value
<b>Core</b>	
R0	0x12345678
R1	0x23456780
<b>R2</b>	<b>0x00000014</b>
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
<b>R15 (PC)</b>	<b>0x0000000E</b>
+ xPSR	0x01000000
+ Banked	
+ System	
- Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP

Address	Value
0x00000000	00 10 00 10
0x00000004	09 00 00 00
0x00000008	02 48 03 49
0x0000000C	03 4A 13 68
0x00000010	FE E7 00 00
0x00000014	78 56 34 12
0x00000018	80 67 45 23
0x0000001C	14 00 00 00
0x00000020	00 00 00 00
0x00000024	00 00 00 00
0x00000028	00 00 00 00
0x0000002C	00 00 00 00
0x00000030	00 00 00 00
0x00000034	00 00 00 00
0x00000038	00 00 00 00

Proj1.s

```

1      AREA RESET, DATA, READONLY
2      EXPORT __Vectors
3 __Vectors
4      DCD 0x10001000
5      DCD Reset_Handler
6      ALIGN
7      AREA mycode, CODE, READONLY
8      ENTRY
9      EXPORT Reset_Handler
10 Reset_Handler
11      LDR R0, SRC
12      LDR R1, =0x23456780
13      LDR R2, =SRC
14      LDR R3, [R2]
15 STOP B STOP
16 SRC DCD 0x12345678
17 END

```

Register	Value
<b>Core</b>	
R0	0x12345678
R1	0x23456780
R2	0x00000014
<b>R3</b>	<b>0x12345678</b>
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFF
<b>R15 (PC)</b>	<b>0x00000010</b>
+ xPSR	
+ Banked	0x01000000
+ System	
+ Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP

Project    Registers

Address	Value
0x00000000:	00 10 00 10
0x00000004:	09 00 00 00
0x00000008:	02 48 03 49
0x0000000C:	03 4A 13 68
0x00000010:	FE E7 00 00
0x00000014:	78 56 34 12
0x00000018:	80 67 45 23
0x0000001C:	14 00 00 00
0x00000020:	00 00 00 00
0x00000024:	00 00 00 00
0x00000028:	00 00 00 00
0x0000002C:	00 00 00 00
0x00000030:	00 00 00 00
0x00000034:	00 00 00 00
0x00000038:	00 00 00 00

Proj1.s

```

2      EXPORT _Vectors
3  _Vectors
4      DCD 0x10001000
5      DCD Reset_Handler
6      ALIGN
7      AREA mycode, CODE, READONLY
8      ENTRY
9      EXPORT Reset_Handler
10 Reset_Handler
11      LDR R0, =SRC
12      LDR R1, [R0]
13      LDR R2, =DST
14      STR R1, [R2]
15 STOP B STOP
16 SRC DCD 0x12345678
17 AREA mydata, DATA, READWRITE
18 DST DCD 0
19 END
20

```

**Registers**

Register	Value
<b>Core</b>	
R0	0x00000014
R1	0x12345678
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x0000000C
xPSR	0x01000000
<b>Banked</b>	
<b>System</b>	
<b>Internal</b>	
Mode	Thread
Privilege	Privileged
Stack	MSP

**Memory 1**

Address:	0
0x00000000:	00 10 00 10
0x00000004:	09 00 00 00
0x00000008:	03 48 01 68
0x0000000C:	03 4A 11 60
0x00000010:	FE E7 00 00
0x00000014:	78 56 34 12
0x00000018:	14 00 00 00
0x0000001C:	00 00 00 10
0x00000020:	00 00 00 00
0x00000024:	00 00 00 00
0x00000028:	00 00 00 00
0x0000002C:	00 00 00 00
0x00000030:	00 00 00 00
0x00000034:	00 00 00 00
0x00000038:	00 00 00 00

**Memory 2**

Address:	0x10000000
0x10000000:	00 00 00 00
0x10000004:	00 00 00 00
0x10000008:	00 00 00 00
0x1000000C:	00 00 00 00
0x10000010:	00 00 00 00
0x10000014:	00 00 00 00
0x10000018:	00 00 00 00
0x1000001C:	00 00 00 00
0x10000020:	00 00 00 00
0x10000024:	00 00 00 00
0x10000028:	00 00 00 00
0x1000002C:	00 00 00 00
0x10000030:	00 00 00 00
0x10000034:	00 00 00 00
0x10000038:	00 00 00 00

```

Proj1.s

2 EXPORT __Vectors
3 __Vectors
4 DCD 0x10001000
5 DCD Reset_Handler
6 ALIGN
7 AREA mycode, CODE, READONLY
8 ENTRY
9 EXPORT Reset_Handler
10 Reset_Handler
11 LDR R0, =SRC
12 LDR R1, [R0]
13 LDR R2, =DST
14 STR R1, [R2]
15 STOP B STOP
16 SRC DCD 0x12345678
17 AREA mydata, DATA, READWRITE
18 DST DCD 0
19 END

```

Register	Value
Core	
R0	0x00000014
R1	0x12345678
<b>R2</b>	<b>0x10000000</b>
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
<b>R15 (PC)</b>	<b>0x0000000E</b>
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP

Address:	0
0x00000000:	00 10 00 10
0x00000004:	09 00 00 00
0x00000008:	03 48 01 68
0x0000000C:	03 4A 11 60
0x00000010:	FE E7 00 00
0x00000014:	<b>78 56 34 12</b>
0x00000018:	<b>14 00 00 00</b>
0x0000001C:	<b>00 00 00 10</b>
0x00000020:	00 00 00 00
0x00000024:	00 00 00 00
0x00000028:	00 00 00 00
0x0000002C:	00 00 00 00
0x00000030:	00 00 00 00
0x00000034:	00 00 00 00
0x00000038:	00 00 00 00

Address:	0x10000000
0x10000000:	00 00 00 00
0x10000004:	00 00 00 00
0x10000008:	00 00 00 00
0x1000000C:	00 00 00 00
0x10000010:	00 00 00 00
0x10000014:	00 00 00 00
0x10000018:	00 00 00 00
0x1000001C:	00 00 00 00
0x10000020:	00 00 00 00
0x10000024:	00 00 00 00
0x10000028:	00 00 00 00
0x1000002C:	00 00 00 00
0x10000030:	00 00 00 00
0x10000034:	00 00 00 00
0x10000038:	00 00 00 00

Proj1.s

```
2 EXPORT __Vectors
3 _Vectors
4 DCD 0x10001000
5 DCD Reset_Handler
6 ALIGN
7 AREA mycode, CODE, READONLY
8 ENTRY
9 EXPORT Reset_Handler
10 Reset_Handler
11 LDR R0, =SRC
12 LDR R1, [R0]
13 LDR R2, =DST
14 STR R1, [R2]
15 STOP B STOP
16 SRC DCD 0x12345678
17 AREA mydata, DATA, READWRITE
18 DST DCD 0
19 END
```

Registers

Register	Value
Core	
R0	0x00000014
R1	0x12345678
R2	0x10000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFF
R15 (PC)	0x00000010
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP

Memory 1

Address	Value
0x00000000	00 10 00 10
0x00000004	09 00 00 00
0x00000008	03 48 01 68
0x0000000C	03 4A 11 60
0x00000010	FE E7 00 00
0x00000014	78 56 34 12
0x00000018	14 00 00 00
0x0000001C	00 00 00 10
0x00000020	00 00 00 00
0x00000024	00 00 00 00
0x00000028	00 00 00 00
0x0000002C	00 00 00 00
0x00000030	00 00 00 00
0x00000034	00 00 00 00
0x00000038	00 00 00 00

Memory 2

Address	Value
0x10000000	78 56 34 12
0x10000004	00 00 00 00
0x10000008	00 00 00 00
0x1000000C	00 00 00 00
0x10000010	00 00 00 00
0x10000014	00 00 00 00
0x10000018	00 00 00 00
0x1000001C	00 00 00 00
0x10000020	00 00 00 00
0x10000024	00 00 00 00
0x10000028	00 00 00 00
0x1000002C	00 00 00 00
0x10000030	00 00 00 00
0x10000034	00 00 00 00
0x10000038	00 00 00 00

Proj1.s

```

1 AREA RESET, DATA, READONLY
2 EXPORT _Vectors
3 Vectors
4 DCD 0x10001000
5 DCD Reset_Handler
6 ALIGN
7 AREA mycode, CODE, READONLY
8 ENTRY
9 EXPORT Reset_Handler
10 Reset_Handler
11 LDR R0, =SRC
12 LDR R1, [R0]
13 STOP B STOP
14 SRC DCD 0x12345678, 0x23456781, 0x3445
15 AREA mydata, DATA, READWRITE
16 DST DCD 0
17 END
18

```

**Registers**

Register	Value
R0	0x00000010
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFF
R15 (PC)	0x0000000A
xPSR	0x01000000

**Memory 1**

Address	Value
0x00000000	00 10 00 10
0x00000004	09 00 00 00
0x00000008	04 48 01 68
0x0000000C	FE E7 00 00
0x00000010	78 56 34 12
0x00000014	81 67 45 23
0x00000018	45 34 00 00
0x0000001C	10 00 00 00
0x00000020	00 00 00 00
0x00000024	00 00 00 00
0x00000028	00 00 00 00
0x0000002C	00 00 00 00
0x00000030	00 00 00 00
0x00000034	00 00 00 00
0x00000038	00 00 00 00

**Memory 2**

Address	Value
0x10000000	00 00 00 00
0x10000004	00 00 00 00
0x10000008	00 00 00 00
0x1000000C	00 00 00 00
0x10000010	00 00 00 00
0x10000014	00 00 00 00
0x10000018	00 00 00 00
0x1000001C	00 00 00 00
0x10000020	00 00 00 00
0x10000024	00 00 00 00
0x10000028	00 00 00 00
0x1000002C	00 00 00 00
0x10000030	00 00 00 00
0x10000034	00 00 00 00
0x10000038	00 00 00 00

```

1 AREA RESET, DATA, READONLY
2 EXPORT __Vectors
3 Vectors
4 DCD 0x10001000
5 DCD Reset_Handler
6 ALIGN
7 AREA mycode, CODE, READONLY
8 ENTRY
9 EXPORT Reset_Handler
10 Reset_Handler
11 LDR R0, =SRC
12 LDR R1, [R0]
13 STOP B STOP
14 SRC DCD 0x12345678, 0x23456781, 0x345
15 AREA mydata, DATA, READWRITE
16 DST DCD 0
17 END
18

```

**Registers**

Register	Value
<b>Core</b>	
R0	0x00000010
R1	0x12345678
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x0000000C
xPSR	0x01000000
<b>Banked</b>	
<b>System</b>	
<b>Internal</b>	
Mode	Thread
Privilege	Privileged
Start	MSP

**Memory 1**

Address: 0

0x00000000:	00	10	00	10
0x00000004:	09	00	00	00
0x00000008:	04	48	01	68
0x0000000C:	FE	E7	00	00
0x00000010:	78	56	34	12
0x00000014:	81	67	45	23
0x00000018:	45	34	00	00
0x0000001C:	10	00	00	00
0x00000020:	00	00	00	00
0x00000024:	00	00	00	00
0x00000028:	00	00	00	00
0x0000002C:	00	00	00	00
0x00000030:	00	00	00	00
0x00000034:	00	00	00	00
0x00000038:	00	00	00	00

**Memory 2**

Address: 0x10000000

0x10000000:	00	00	00	00
0x10000004:	00	00	00	00
0x10000008:	00	00	00	00
0x1000000C:	00	00	00	00
0x10000010:	00	00	00	00
0x10000014:	00	00	00	00
0x10000018:	00	00	00	00
0x1000001C:	00	00	00	00
0x10000020:	00	00	00	00
0x10000024:	00	00	00	00
0x10000028:	00	00	00	00
0x1000002C:	00	00	00	00
0x10000030:	00	00	00	00
0x10000034:	00	00	00	00
0x10000038:	00	00	00	00

# Addressing Modes

The way in which an operand is specified in an instruction is called Addressing Mode

1. Register - MOV R0, R1
2. Immediate – MOV R0, #1
3. Register Indirect- LDR R0, [R1]  
STR R0, [R1]
4. Indexed

Indexed Addressing Mode	Syntax	Pointing Location in Memory	Rm Value After Execution
<b>Preindex</b>	LDR Rd, [Rm,#k]	Rm+#k	Rm
<b>Preindex with WB*</b>	LDR Rd, [Rm,#k]!	Rm + #k	Rm + #k
<b>Postindex</b>	LDR Rd, [Rm],#k	Rm	Rm + #k

\*WB means Writeback

\*\*Rd and Rm are any of registers and #k is a signed 12-bit immediate value between -4095 and +4095

# Preindex Mode

Proj1.s

```
1 AREA RESET, DATA, READONLY
2 EXPORT __Vectors
3 Vectors
4 DCD 0x10001000
5 DCD Reset_Handler
6 ALIGN
7 AREA mycode, CODE, READONLY
8 ENTRY
9 EXPORT Reset_Handler
10 Reset_Handler
11 LDR R0, =SRC
12 LDR R1, [R0,#4]
13 STOP B STOP
14 SRC DCD 0x12345678, 0x23456781, 0x3445
15 AREA mydata, DATA, READWRITE
16 DST DCD 0
17 END
18
```

Registers

Register	Value
R0	0x00000010
R1	0x23456781
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x0000000C
+ xPSR	0x01000000
+ Banked	
+ System	
- Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP

Memory 1

Address	Value
0x00000000	00 10 00 10
0x00000004	09 00 00 00
0x00000008	04 48 41 68
0x0000000C	FE E7 00 00
0x00000010	70 56 34 12
0x00000014	81 67 45 23
0x00000018	45 34 00 00
0x0000001C	10 00 00 00
0x00000020	00 00 00 00
0x00000024	00 00 00 00
0x00000028	00 00 00 00
0x0000002C	00 00 00 00
0x00000030	00 00 00 00
0x00000034	00 00 00 00
0x00000038	00 00 00 00

Memory 2

Address	Value
0x10000000	00 00 00 00
0x10000004	00 00 00 00
0x10000008	00 00 00 00
0x1000000C	00 00 00 00
0x10000010	00 00 00 00
0x10000014	00 00 00 00
0x10000018	00 00 00 00
0x1000001C	00 00 00 00
0x10000020	00 00 00 00
0x10000024	00 00 00 00
0x10000028	00 00 00 00
0x1000002C	00 00 00 00
0x10000030	00 00 00 00
0x10000034	00 00 00 00
0x10000038	00 00 00 00

Project Registers

# Addressing Modes

The way in which an operand is specified in an instruction is called Addressing Mode

1. Register - MOV R0, R1
2. Immediate – MOV R0, #1
3. Register Indirect- LDR R0, [R1]  
STR R0, [R1]
4. Indexed

Indexed Addressing Mode	Syntax	Pointing Location in Memory	Rm Value After Execution
<b>Preindex</b>	LDR Rd, [Rm,#k]	Rm+#k	Rm
<b>Preindex with WB*</b>	LDR Rd, [Rm,#k]!	Rm + #k	Rm + #k
<b>Postindex</b>	LDR Rd, [Rm],#k	Rm	Rm + #k

\*WB means Writeback

\*\*Rd and Rm are any of registers and #k is a signed 12-bit immediate value between -4095 and +4095

# Postindex Mode

Proj1.s

```
1 AREA RESET, DATA, READONLY
2 EXPORT __Vectors
3 Vectors
4 DCD 0x10001000
5 DCD Reset_Handler
6 ALIGN
7 AREA mycode, CODE, READONLY
8 ENTRY
9 EXPORT Reset_Handler
10 Reset_Handler
11 LDR R0, =SRC
12 LDR R1, [R0], #4
13 STOP B STOP
14 SRC DCD 0x12345678, 0x23456781, 0x3445
15 AREA mydata, DATA, READWRITE
16 DST DCD 0
17 END
18
```

Registers

Register	Value
R0	0x00000014
R1	0x12345678
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x0000000E
xPSR	0x01000000

Memory 1

Address	Value
0x00000000	00 10 00 10
0x00000004	09 00 00 00
0x00000008	04 48 50 F8
0x0000000C	04 1B FE E7
0x00000010	78 56 34 12
0x00000014	81 67 45 23
0x00000018	45 34 00 00
0x0000001C	10 00 00 00
0x00000020	00 00 00 00
0x00000024	00 00 00 00
0x00000028	00 00 00 00
0x0000002C	00 00 00 00
0x00000030	00 00 00 00
0x00000034	00 00 00 00
0x00000038	00 00 00 00

Memory 2

Address	Value
0x10000000	00 00 00 00
0x10000004	00 00 00 00
0x10000008	00 00 00 00
0x1000000C	00 00 00 00
0x10000010	00 00 00 00
0x10000014	00 00 00 00
0x10000018	00 00 00 00
0x1000001C	00 00 00 00
0x10000020	00 00 00 00
0x10000024	00 00 00 00
0x10000028	00 00 00 00
0x1000002C	00 00 00 00
0x10000030	00 00 00 00
0x10000034	00 00 00 00
0x10000038	00 00 00 00

Project Registers

Proj1.s

```

1 AREA RESET, DATA, READONLY
2 EXPORT __Vectors
3 _Vectors
4 DCD 0x10001000
5 DCD Reset_Handler
6 ALIGN
7 AREA mycode, CODE, READONLY
8 ENTRY
9 EXPORT Reset_Handler
10 Reset_Handler
11 LDR R0, =SRC
12 LDR R1, [R0], #4
13 LDR R2, [R0], #4
14 STOP B STOP
15 SRC DCD 0x12345678, 0x23456781, 0x3445
16 AREA mydata, DATA, READWRITE
17 DST DCD 0
18 END

```

Registers

Register	Value
<b>Core</b>	
R0	0x00000018
R1	0x12345678
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
<b>R15 (PC)</b>	<b>0x0000000E</b>
xPSR	0x01000000
<b>Banked</b>	
<b>System</b>	
<b>Internal</b>	
Mode	Thread
Privilege	Privileged
Stack	MSP

Memory 1

Address	Value
0x00000000	00 10 00 10
0x00000004	09 00 00 00
0x00000008	05 48 50 F8
0x0000000C	04 1B 50 F8
0x00000010	04 2B FE E7
0x00000014	78 56 34 12
0x00000018	81 67 45 23
0x0000001C	45 34 00 00
0x00000020	14 00 00 00
0x00000024	00 00 00 00
0x00000028	00 00 00 00
0x0000002C	00 00 00 00
0x00000030	00 00 00 00
0x00000034	00 00 00 00
0x00000038	00 00 00 00

Memory 2

Address	Value
0x10000000	00 00 00 00
0x10000004	00 00 00 00
0x10000008	00 00 00 00
0x1000000C	00 00 00 00
0x10000010	00 00 00 00
0x10000014	00 00 00 00
0x10000018	00 00 00 00
0x1000001C	00 00 00 00
0x10000020	00 00 00 00
0x10000024	00 00 00 00
0x10000028	00 00 00 00
0x1000002C	00 00 00 00
0x10000030	00 00 00 00
0x10000034	00 00 00 00
0x10000038	00 00 00 00

## Proj1.s

```

1 AREA RESET, DATA, READONLY
2 EXPORT __Vectors
3 __Vectors
4 DCD 0x10001000
5 DCD Reset_Handler
6 ALIGN
7 AREA mycode, CODE, READONLY
8 ENTRY
9 EXPORT Reset_Handler
10 Reset_Handler
11 LDR R0, =SRC
12 LDR R1, [R0], #4
13 LDR R2, [R0], #4
14 STOP B STOP
15 SRC DCD 0x12345678, 0x23456781, 0x3445
16 AREA mydata, DATA, READWRITE
17 DST DCD 0
18 END

```

Register	Value
<b>Core</b>	
R0	0x0000001C
R1	0x12345678
R2	0x23456781
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x00000012
+ xPSR	0x01000000
<b>Banked</b>	
<b>System</b>	
<b>Internal</b>	
Mode	Thread
Privilege	Privileged
Stack	MSP

Address:	0
0x00000000:	00 10 00 10
0x00000004:	09 00 00 00
0x00000008:	05 48 50 F8
0x0000000C:	04 1B 50 F8
0x00000010:	04 2B FE E7
0x00000014:	78 56 34 12
0x00000018:	81 67 45 23
0x0000001C:	45 34 00 00
0x00000020:	14 00 00 00
0x00000024:	00 00 00 00
0x00000028:	00 00 00 00
0x0000002C:	00 00 00 00
0x00000030:	00 00 00 00
0x00000034:	00 00 00 00
0x00000038:	00 00 00 00

Address:	0x10000000
0x10000000:	00 00 00 00
0x10000004:	00 00 00 00
0x10000008:	00 00 00 00
0x1000000C:	00 00 00 00
0x10000010:	00 00 00 00
0x10000014:	00 00 00 00
0x10000018:	00 00 00 00
0x1000001C:	00 00 00 00
0x10000020:	00 00 00 00
0x10000024:	00 00 00 00
0x10000028:	00 00 00 00
0x1000002C:	00 00 00 00
0x10000030:	00 00 00 00
0x10000034:	00 00 00 00
0x10000038:	00 00 00 00

# Addressing Modes

The way in which an operand is specified in an instruction is called Addressing Mode

1. Register - MOV R0, R1
2. Immediate – MOV R0, #1
3. Register Indirect- LDR R0, [R1]  
STR R0, [R1]
4. Indexed

Indexed Addressing Mode	Syntax	Pointing Location in Memory	Rm Value After Execution
<b>Preindex</b>	LDR Rd, [Rm,#k]	Rm+#k	Rm
<b>Preindex with WB*</b>	LDR Rd, [Rm,#k]!	Rm + #k	Rm + #k
<b>Postindex</b>	LDR Rd, [Rm],#k	Rm	Rm + #k

\*WB means Writeback

\*\*Rd and Rm are any of registers and #k is a signed 12-bit immediate value between -4095 and +4095

# Preindex with WB Mode

```
Proj1.s
1 AREA RESET, DATA, READONLY
2 EXPORT __Vectors
3 __Vectors
4 DCD 0x10001000
5 DCD Reset_Handler
6 ALIGN
7 AREA mycode, CODE, READONLY
8 ENTRY
9 EXPORT Reset_Handler
10 Reset_Handler
11 LDR R0, =SRC
12 LDR R1, [R0,#4]!
13 STOP B STOP
14 SRC DCD 0x12345678, 0x23456781, 0x3445
15 AREA mydata, DATA, READWRITE
16 DST DCD 0
17 END
18
```

Registers

Register	Value	Memory 1	Memory 2
R0	0x00000010	0x00000000: 00 10 00 10	0x10000000: 00 00 00 00
R1	0x00000000	0x00000004: 09 00 00 00	0x10000004: 00 00 00 00
R2	0x00000000	0x00000008: 04 48 50 F8	0x10000008: 00 00 00 00
R3	0x00000000	0x0000000C: 04 1F FE E7	0x1000000C: 00 00 00 00
R4	0x00000000	0x00000010: 78 56 34 12	0x10000010: 00 00 00 00
R5	0x00000000	0x00000014: 81 67 45 23	0x10000014: 00 00 00 00
R6	0x00000000	0x00000018: 45 34 00 00	0x10000018: 00 00 00 00
R7	0x00000000	0x0000001C: 10 00 00 00	0x1000001C: 00 00 00 00
R8	0x00000000	0x00000020: 00 00 00 00	0x10000020: 00 00 00 00
R9	0x00000000	0x00000024: 00 00 00 00	0x10000024: 00 00 00 00
R10	0x00000000	0x00000028: 00 00 00 00	0x10000028: 00 00 00 00
R11	0x00000000	0x0000002C: 00 00 00 00	0x1000002C: 00 00 00 00
R12	0x00000000	0x00000030: 00 00 00 00	0x10000030: 00 00 00 00
R13 (SP)	0x10001000	0x00000034: 00 00 00 00	0x10000034: 00 00 00 00
R14 (LR)	0xFFFFFFF	0x00000038: 00 00 00 00	0x10000038: 00 00 00 00
R15 (PC)	0x0000000A		
xPSR	0x01000000		

Memory 1 Address: 0

Memory 2 Address: 0x10000000

Core  
Banked  
System  
Internal  
Mode Thread  
Privilege Privileged  
Starv MSP

Project Registers

Proj1.s

```

1 AREA RESET, DATA, READONLY
2 EXPORT __Vectors
3 Vectors
4 DCD 0x10001000
5 DCD Reset_Handler
6 ALIGN
7 AREA mycode, CODE, READONLY
8 ENTRY
9 EXPORT Reset_Handler
10 Reset_Handler
11 LDR R0, =SRC
12 LDR R1, [R0, #4]!
13 STOP B STOP
14 SRC DCD 0x12345678, 0x23456781, 0x3445
15 AREA mydata, DATA, READWRITE
16 DST DCD 0
17 END

```

**Registers**

Register	Value
R0	0x00000000
R1	0x23456781
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x0000000E
xPSR	0x01000000

**Memory 1**

Address	Value
0x00000000	00 10 00 10
0x00000004	09 00 00 00
0x00000008	04 48 50 F8
0x0000000C	04 1F FE E7
0x00000010	78 56 34 12
0x00000014	81 67 45 23
0x00000018	45 34 00 00
0x0000001C	10 00 00 00
0x00000020	00 00 00 00
0x00000024	00 00 00 00
0x00000028	00 00 00 00
0x0000002C	00 00 00 00
0x00000030	00 00 00 00
0x00000034	00 00 00 00
0x00000038	00 00 00 00

**Memory 2**

Address	Value
0x10000000	00 00 00 00
0x10000004	00 00 00 00
0x10000008	00 00 00 00
0x1000000C	00 00 00 00
0x10000010	00 00 00 00
0x10000014	00 00 00 00
0x10000018	00 00 00 00
0x1000001C	00 00 00 00
0x10000020	00 00 00 00
0x10000024	00 00 00 00
0x10000028	00 00 00 00
0x1000002C	00 00 00 00
0x10000030	00 00 00 00
0x10000034	00 00 00 00
0x10000038	00 00 00 00

**Project | Registers**

# Load/Store Byte/Half Word

---

LDRB – Load Byte

LDRH – Load Halfword

LDRSB – Load Signed byte

LDRSH – Load Signed Halfword

STRH – Store Halfword

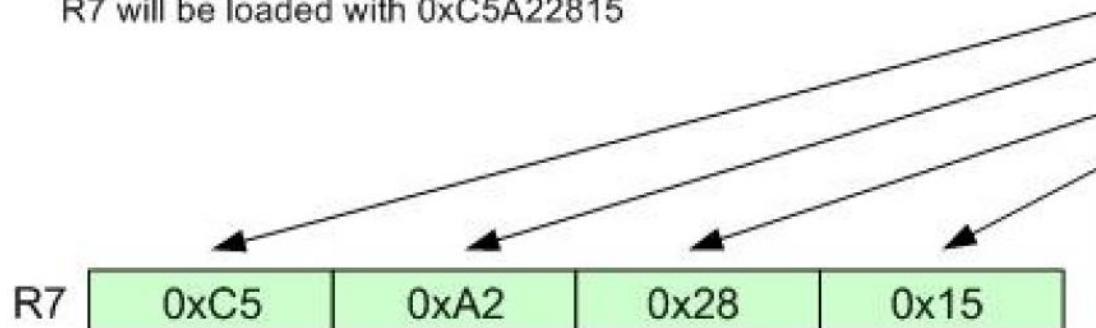
Note : For all these instructions – Indirect/Indexed  
addressing modes are applicable)

Assume that R5=0x40000200, and locations 0x40000200 through 0x40000203 contain 0x15, 0x28, 0xA2 and 0xC5, respectively.

After running the following instruction:

**LDR R7, [R5]**

R7 will be loaded with 0xC5A22815

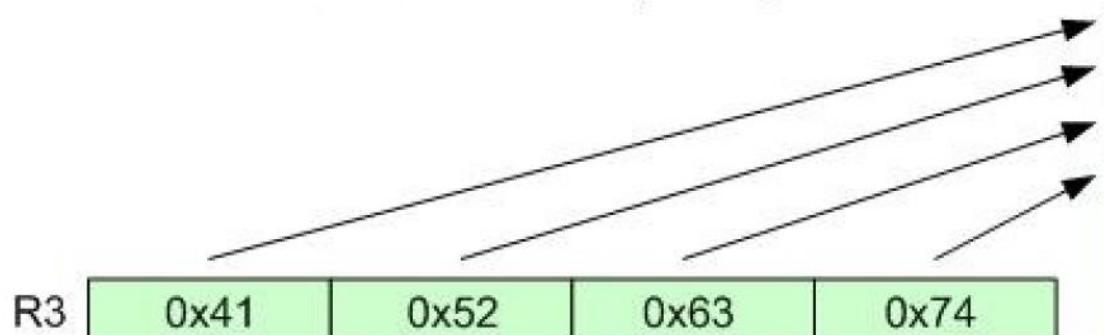


0xC5	0x4000 0203
0xA2	0x4000 0202
0x28	0x4000 0201
0x15	0x4000 0200

Assume that R6=0x40000200, and R3 = 0x41526374. After running the following instruction:

**STR R3, [R6]**

locations 0x40000200 through 0x40000203 will be loaded with 0x74, 0x63, 0x52, and 0x41, respectively.



0x41	0x4000 0203
0x52	0x4000 0202
0x63	0x4000 0201
0x74	0x4000 0200

<b>Data Size</b>	<b>Bits</b>	<b>Decimal</b>	<b>Hexadecimal</b>	<b>Load instruction used</b>
<b>Byte</b>	8	0 – 255	0 - 0xFF	STRB
<b>Half-word</b>	16	0 – 65535	0 - 0xFFFF	STRH
<b>Word</b>	32	0 – $2^{32}-1$	0 - 0xFFFFFFFF	STR

# MOV instruction

---

MOV Rd, Rn

MOV Rd, #0x12

MOVW Rd, #0x1234 (Move Word)

MVN Rd, Rn (Move Negative)

MVN Rd, #0x12

MSR Special\_Function\_Reg, Rn (Move SFR from Register)

MRS Rn, Special\_Function\_Reg (Move Register from SFR)

# MOV instruction

MOV Rd, Rn

MOV Rd, #0x12

MOVW Rd, #0x1234 (Move Word. i.e to the Lower 16 bit)

MOVT Rd, #0x1234 (Move Top. i.e. to higher 16 bit)

MVN Rd, Rn (Move Negative – 1's compliment)

MVN Rd, #0x12

MSR Special\_Function\_Reg, Rn (Move SFR from Register)

MRS Rn, Special\_Function\_Reg (Move Register from SFR)

The screenshot shows a debugger interface with two main windows: 'Registers' and 'Disassembly'.

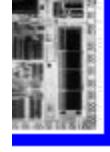
**Registers Window:** Shows the state of various registers. The 'Core' section includes R0 (0x00000000), R1 (0x00000012), R2 (0x00001234), R3 (0x12340000), R4 (0xFFFFFFFED), R5 (0xFFFFFFFEC), R6 (0x00000000), R7 (0x00000000), R8 (0x00000000), R9 (0x00000000), R10 (0x00000000), R11 (0x00000000), R12 (0x00000000), R13 (SP) (0x10001000), R14 (LR) (0xFFFFFFFF), R15 (PC) (0x0000001C). The 'xPSR' section shows its value as 0x01000000. The 'Banked' section is collapsed.

**Disassembly Window:** Shows assembly code for a program named 'Reset\_Handler'. The code includes DCD instructions for memory, an ALIGN directive, an ENTRY point, and several MOV, MVN, MSR, and MRS instructions. The PC is at address 0x0000001C.

Register	Value
R0	0x00000000
R1	0x00000012
R2	0x00001234
R3	0x12340000
R4	0xFFFFFFFED
R5	0xFFFFFFFEC
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x10001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x0000001C
xPSR	0x01000000

Disassembly:

```
17:      MSR xPSR, R5
P1_2.s    P2_1.s    P2_3.s
4        DCD 0x10001000
5        DCD Reset_Handler
6        ALIGN
7        AREA mycode, CODE, READONLY
8        ENTRY
9        EXPORT Reset_Handler
10
11 Reset_Handler
12        MOV R1, #0x12
13        MOVW R2, #0x1234
14        MOVT R3, #0x1234
15        MVN R4, R1
16        MVN R5, #0x13
17        MSR xPSR, R5
18        MRS R6, xPSR
19        STOP
20        B STOP
21        END
```



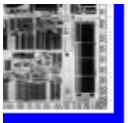
# ADD instruction

ADD Rd, Rn, opr2 ;  $Rd \leftarrow Rn + opr2$  (addition)

ADC Rd, Rn, opr2;  $Rd \leftarrow Rn + opr2 + C$

Flags not affected.

Use S suffix to update flags: ADDS, ADCS



## SUBTRACT

SUB Rd, Rn, opr2 ; Rd  $\leftarrow$  Rn-opr2 (**Subtract**)

SBC Rd, Rn, opr2; Rd  $\leftarrow$  Rn -opr2 – (1- C) (**Subtract with Carry**)

RSB Rd, Rn, opr2 ; Rd  $\leftarrow$  opr2-Rn (**Reverse Subtract**)

RSC Rd, Rn, opr2 ; Rd  $\leftarrow$  opr2-Rn – (1-C) (**Reverse Subtract with Carry**)

Flags not affected.

Use S suffix to update flags : SUBS, SBCS, RSBS, RSCS



**Figure 2- 11: CPSR (Current Program Status Register)**

**Example 2-5**

Show the status of the C and Z flags after the addition of

- a) 0x0000009C and 0xFFFFF64 in the following instruction:

```
;assume R1 = 0x0000009C and R2 = 0xFFFFF64
```

```
ADDS R2,R1,R2 ;add R1 to R2 and place the result in R2
```

- b) 0x0000009C and 0xFFFFF69 in the following instruction:

```
;assume R1 = 0x0000009C and R2 = 0xFFFFF69
```

```
ADDS R2,R1,R2 ;add R1 to R2 and place the result in R2
```

**Example 2-6**

Show the status of the Z flag during the execution of the following program:

```
MOV R2,#4 ;R2 = 4
```

```
MOV R3,#2 ;R3 = 2
```

```
MOV R4,#4 ;R4 = 4
```

```
SUBS R5,R2,R3 ;R5 = R2 - R3 (R5 = 4 - 2 = 2)
```

```
SUBS R5,R2,R4 ;R5 = R2 - R4 (R5 = 4 - 4 = 0)
```

**FLAG REGISTER (CPSR)**

Look at the following case for 8-bit data size:

+ 96    0110 0000

**Check The Status of the Flag**

+ 70    +0100 0110

+166    1010 0110

**Check The Status of the Flag**

Examine the following case:

- 128    1000 0000

+7    0000 0111

Observe the results of the following:

+ —2    +1111 1110

+ +18    +0001 0010

-2    1111 1110

+ -5    1111 1011

Observe the results in the following 16-bit hex numbers:

+6E2F 0110 1110 0010 1111

+ +13D4 0001 0011 1101 0100

[Check The Status of the Flag](#)

Observe the results in the following 16-bit hex numbers:

+542F 0101 0100 0010 1111

+ +12E0 +0001 0010 1110 0000

[Check The Status of the Flag](#)



# Embedded System and Design

---

**DR MANOJ TOLANI (PHD-IIIT ALLAHABAD)**  
**ASSISTANT PROFESSOR (DEPARTMENT OF ICT)**

**Core**

R0	0x00000000
R1	0x00000024
R2	0x00000001
<b>R3</b>	<b>0x00000006</b>
R4	0x00000000
R5	0x00000000
R6	0x10000000
R7	0x00000000
PC	0x00000000

**Load Block of six 32 bit numbers from code memory to data memory****SRC**

0x00000024:	0000001A
0x00000028:	0000001B
0x0000002C:	0000001C
0x00000030:	0000001D
0x00000034:	0000001E
0x00000038:	0000001F

**DST**

0x10000000:	0000001A
0x10000004:	0000001B
0x10000008:	0000001C
0x1000000C:	0000001D
0x10000010:	0000001E
0x10000014:	0000001F
0x10000018:	00000000
0x1000001C:	00000000

**Core**

R0	0x00000000
R1	0x00000024
R2	0x00000001
<b>R3</b>	<b>0x00000006</b>
R4	0x00000000
R5	0x00000000
R6	0x10000000
R7	0x00000000
PC	0x00000000

SRC

0x00000024: 0000001A  
0x00000028: 0000001B  
0x0000002C: 0000001C  
0x00000030: 0000001D  
0x00000034: 0000001E  
0x00000038: 0000001F

DST

0x10000000: 0000001A
0x10000004: 0000001B
0x10000008: 0000001C
0x1000000C: 0000001D
0x10000010: 0000001E
0x10000014: 0000001F
0x10000018: 00000000
0x1000001C: 00000000

**Load Block of six 32 bit numbers from one memory to another**

Reset\_Handler

```
LDR R0, =SRC
LDR R1, =DST
MOV R3, #N
LOOP
    LDR R2, [R0], #4
    STR R2, [R1], #4
    SUBS R3, #1
    BNE LOOP
STOP B STOP
SRC DCD 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F
N EQU 6
AREA mydata,DATA,READWRITE
DST DCD 0,0,0,0,0,0
END
```

Address:	Value
0x10000000	0000000A
0x10000004	0000000B
0x10000008	0000000C
0x1000000C	0000000D
0x10000010	0000000E
0x10000014	0000000F
0x10000018	00000000
0x1000001C	00000000
0x10000020	00000000
0x10000024	00000000
0x10000028	00000000
0x1000002C	00000000
0x10000030	00000000
0x10000034	00000000
0x10000038	00000000
0x1000003C	00000000
0x10000040	0000000A
0x10000044	0000000B
0x10000048	0000000C
0x1000004C	0000000D
0x10000050	0000000E
0x10000054	0000000F
0x10000058	00000000
0x1000005C	00000000
0x10000060	00000000
0x10000064	00000000

Load Block of 6 32 bit numbers from code memory to data memory and data again from one data memory location to another data memory location

a. When the source and destination data memory blocks are non-overlapping

Address:	0x10000000
0x10000000:	0000001A
0x10000004:	0000001B
0x10000008:	0000001C
0x1000000C:	0000001D
0x10000010:	0000001E
0x10000014:	0000001F
0x10000018:	00000000
0x1000001C:	00000000
0x10000020:	00000000
0x10000024:	00000000
0x10000028:	00000000
0x1000002C:	00000000
0x10000030:	00000000
0x10000034:	00000000
0x10000038:	00000000
0x1000003C:	00000000
0x10000040:	0000001A
0x10000044:	0000001B
0x10000048:	0000001C
0x1000004C:	0000001D
0x10000050:	0000001E
0x10000054:	0000001F
0x10000058:	00000000
0x1000005C:	00000000
0x10000060:	00000000
0x10000064:	00000000

Load Block of 6 32 bit numbers from code memory to data memory and data again from one data memory location to another data memory location

**a. When the source and destination data memory blocks are non-overlapping**

```

Reset_Handler
    LDR R0, =SRC
    LDR R1, =DST
    MOV R3, #N
LOOP
    LDR R2, [R0], #4
    STR R2, [R1], #4
    SUBS R3, #1
    BNE LOOP
    MOV R3, #N
    LDR R0, =DST
    LDR R1, =0x10000040
LOOP1
    LDR R2, [R0], #4
    STR R2, [R1], #4
    SUBS R3, #1
    BNE LOOP1
STOP B STOP
SRC DCD 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F
N EQU 6
AREA mydata,DATA,READWRITE
DST DCD 0,0,0,0,0,0
END

```

Address: 0x10000000

0x10000000 : 0000001A

0x10000004 : 0000001B

0x10000008 : 0000001C

0x1000000C : 0000001D

0x10000010 : 0000001E 0000001A

0x10000014 : 0000001F 0000001B

0x10000018 : 0000001C

0x1000001C : 0000001D

0x10000020 : 0000001E

0x10000024 : 0000001F

0x10000028 : Load Block of 6 32 bit numbers from code memory to data memory and data again from one data memory location to another data memory location

0x1000002C : b. When the source and destination data memory blocks are overlapping

0x10000030 :

Shift

Overlapping

Address:	Value
0x10000000	00000001A
0x10000004	00000001B
0x10000008	00000001C
0x1000000C	00000001D
0x10000010	00000001E 00000001A
0x10000014	00000001F 00000001B
0x10000018	00000001C
0x1000001C	00000001D
0x10000020	00000001E
0x10000024	00000001F
0x10000028	
0x1000002C	
0x10000030	
0x10000034	
0x10000038	
0x1000003C	
0x10000040	
0x10000044	
0x10000048	
0x1000004C	
0x10000050	
0x10000054	
0x10000058	
0x1000005C	
0x10000060	
0x10000064	

## Reset\_Handler

```
LDR R0, =SRC  
LDR R1, =DST  
MOV R3, #N  
LOOP  
    LDR R2, [R0], #4  
    STR R2, [R1], #4  
    SUBS R3, #1  
    BNE LOOP  
    MOV R3, #N  
    LDR R0, =DST+(N-1)*4  
    LDR R1, =DST+(N-1)*4+SH*4  
LOOP1  
    LDR R2, [R0], #-4  
    STR R2, [R1], #-4  
    SUBS R3, #1  
    BNE LOOP1  
STOP B STOP  
SRC DCD 0xA, 0xB, 0xC, 0xD, 0xE, 0xF  
N EQU 6  
SH EQU 4  
        AREA mydata,DATA,READWRITE  
DST DCD 0,0,0,0,0,0  
END
```

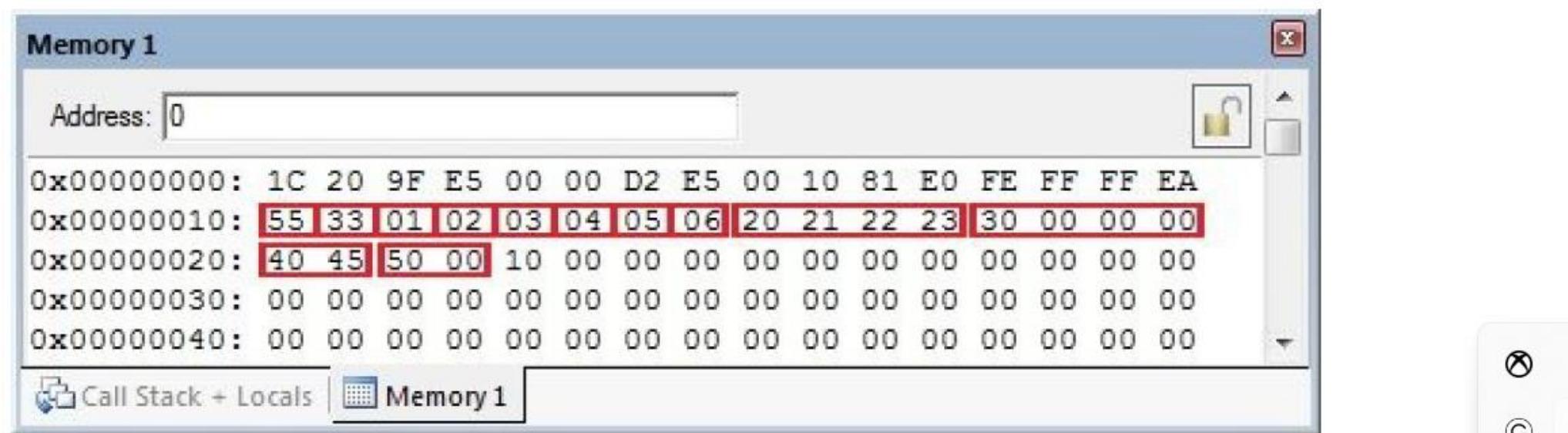
## OUR\_FIXED\_DATA

DCB 0x55,0x33,1,2,3,4,5,6

DCD 0x23222120,0x30

DCW 0x4540,0x50

END



# BRANCH INSTRUCTIONS

---

## B                  Branch (unconditional jump)

*Flags:* Unchanged.

*Format:*      B target            jump to target address

## Bxx                Branch Conditional

*Flags:* Unaffected.

*Format:*      Bxx target ;jump to target upon  
condition

Instruction	Flags Affected
<b>BCS</b>	Branch if C = 1
<b>BCC</b>	Branch if C = 0
<b>BEQ</b>	Branch if Z = 1
<b>BNE</b>	Branch if Z = 0
<b>BMI</b>	Branch if N = 1
<b>BPL</b>	Branch if N = 0
<b>BVS</b>	Branch if V = 1
<b>BVC</b>	Branch if V = 0

# BRANCH INSTRUCTIONS

---

Based on multiple flags

"B condition" where the condition refers to the comparison of unsigned numbers. After a compare (CMP Rn,Op2) instruction is executed, C and Z indicate the result of the comparison, as follows:

	C	Z
<b>Rn &gt; Op2</b>	1	0
<b>Rn = Op2</b>	1	1
<b>Rn &lt; Op2</b>	0	0

# BRANCH INSTRUCTIONS

---

Instruction	Condition
<b>BHI</b> Branch if Higher	jump if C=1 and Z=0
<b>BEQ</b> Branch if Equal	jump if C=1 and Z=1
<b>BLS</b> Branch if Lower or same	jump if C=0 or Z=1
<b>BLO</b> Brach if lower	C=0 and Z=0
<b>BHS</b> Brach if Higher or same	C=1 or Z=1

For unsigned comparison

# BRANCH INSTRUCTIONS

---

Instruction	Condition
<b>BCS</b> Branch if Carry Set	jump if C=1
<b>BCC</b> Branch if Carry Clear	jump if C=0
<b>BEQ</b> Branch if Equal	jump if Z=1
<b>BNE</b> Branch if Not Equal	jump if Z=0
<b>BMI</b> Branch if Minus/Negative	jump if N=1
<b>BPL</b> Branch if Plus/Positive	jump if N=0
<b>BVS</b> Branch if Overflow	jump if V=1
<b>BVC</b> Branch if No overflow	jump if V=0

Based on single flag

# BRANCH INSTRUCTIONS

---

Instruction	
<b>BGE</b>	Branch Greater or Equal
<b>BLT</b>	Branch Less than
<b>BGT</b>	Branch Greater than
<b>BLE</b>	Branch Less or Equal
<b>BEQ</b>	Branch if Equal

$V=N$  or  $Z=1$

$V$  is inverse  $N$  and  $Z=0$

$V=N$  and  $Z=0$

$V$  is inverse  $N$  or  $Z=1$

jump if  $Z = 1$

<b>Rn &gt; Op2</b>	$V=N$ or $Z=0$
<b>Rn = Op2</b>	$Z=1$
<b>Rn &lt; Op2</b>	$V$ inverse of $N$

For the signed comparison

### *CASE 1: Addition of individual word data*

#### **Program 3-1**

**Write a program to calculate the total sum of five words of data. Each data value represents the mass of a planet in integer. The decimal data are as follow: 1000000000, 2000000000, 3000000000, 4000000000, and 4100000000.**

## *CASE 1: Addition of individual word data*

### Program 3-1

**Write a program to calculate the total sum of five words of data. Each data value represents the mass of a planet in integer. The decimal data are as follow: 1000000000, 2000000000, 3000000000, 4000000000, and 4100000000.**

### Reset\_Handler

```
LDR R1, =SRC  
LDR R2, =DST  
MOV R4, #5  
MOV R8, #0  
MOV R9, #0
```

### LOOP

```
LDR R3, [R1], #4  
ADDS R8, R3  
ADC R9, #0  
SUBS R4, #1  
BNE LOOP
```

```
STR R8, [R2], #4  
STR R9, [R2]
```

### STOP

```
B STOP
```

```
SRC DCD 1000000000, 2000000000, 3000000000,  
4000000000, 4100000000
```

```
AREA dataset, DATA, READWRITE
```

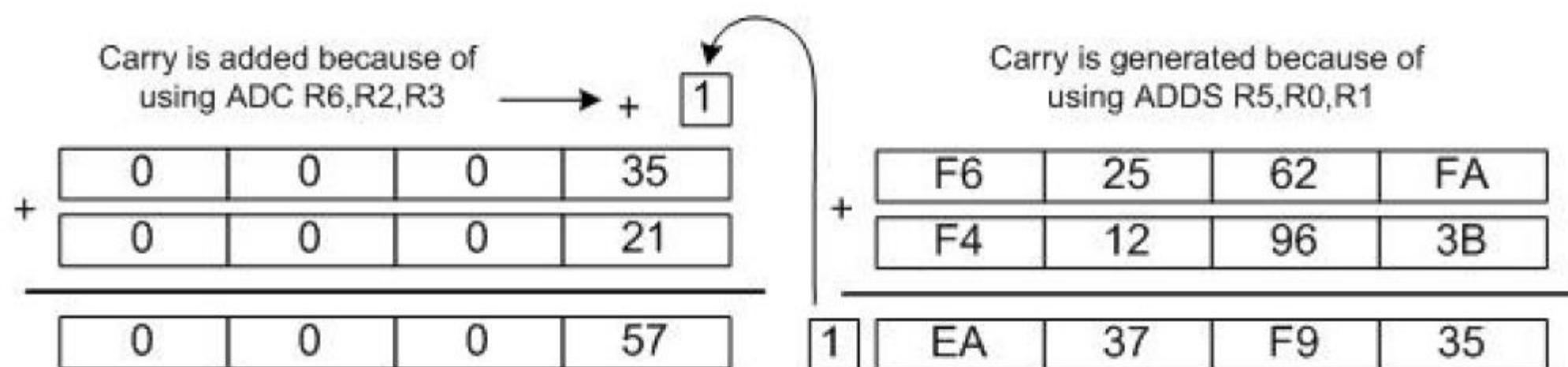
```
DST DCD 0x0, 0x0
```

```
END
```

## CASE 2: Addition of multiword numbers

Write a program which adds 0x35F62562FA to 0x21F412963B:

0x35F62562FA to 0x21F412963B:



## CASE 2: Addition of multiword numbers

Analyze the following program which adds 0x35F62562FA to 0x21F412963B:

```
LDR    R0,=0xF62562FA      ;R0 = 0xF62562FA
LDR    R1,=0xF412963B      ;R1 = 0xF412963B
MOV    R2,#0x35            ;R2 = 0x35
MOV    R3,#0x21            ;R3 = 0x21
ADDS   R5,R1,R0            ;R5 = 0xF62562FA + 0xF412963B
;now C = 1
|ADC    R6,R2,R3          ;R6 = R2 + R3 + C
;  = 0x35 + 21 + 1 = 0x57
```

Show the steps involved for the following cases:

a)

```
MOV R2,#0x4F ;R2 = 0x4F  
MOV R3,#0x39 ;R3 = 0x39  
SUBS R4,R2,R3 ;R4 = R2 - R3
```

b)

```
MOV R2,#0x4F ;R2 = 0x4F  
SUBS R4,R2,#0x05 ;R4 = R2 - 0x05
```

Analyze the following instructions:

LDR	R2,=0x88888888	;R2 = 0x88888888
LDR	R3,=0x33333333	;R3 = 0x33333333
SUBS	R4,R2,R3	;R4 = R2 – R3

Analyze the following instructions:

MOV	R1,#0x4C	;R1 = 0x4C
MOV	R2,#0x6E	;R2 = 0x6E
SUBS	R0,R1,R2	;R0 = R1 – R2

---

Analyze the following program which subtracts 0x21F62562FA from 0x35F412963B:

```
LDR    R0,=0xF62562FA      ;R0 = 0xF62562FA,  
; notice the syntax for LDR  
LDR    R1,=0xF412963B      ;R1 = 0xF412963B  
MOV    R2,#0x21             ;R2 = 0x21  
MOV    R3,#0x35             ;R3 = 0x35  
SUBS   R5,R1,R0             ;R5 = R1 - R0  
;  = 0xF412963B - 0xF62562FA, and C = 0  
SBC    R6,R3,R2             ;R6 = R3 - R2 - 1 + C  
;  = 0x35 - 0x21 - 1 + 0 = 0x13
```



# Embedded System and Design

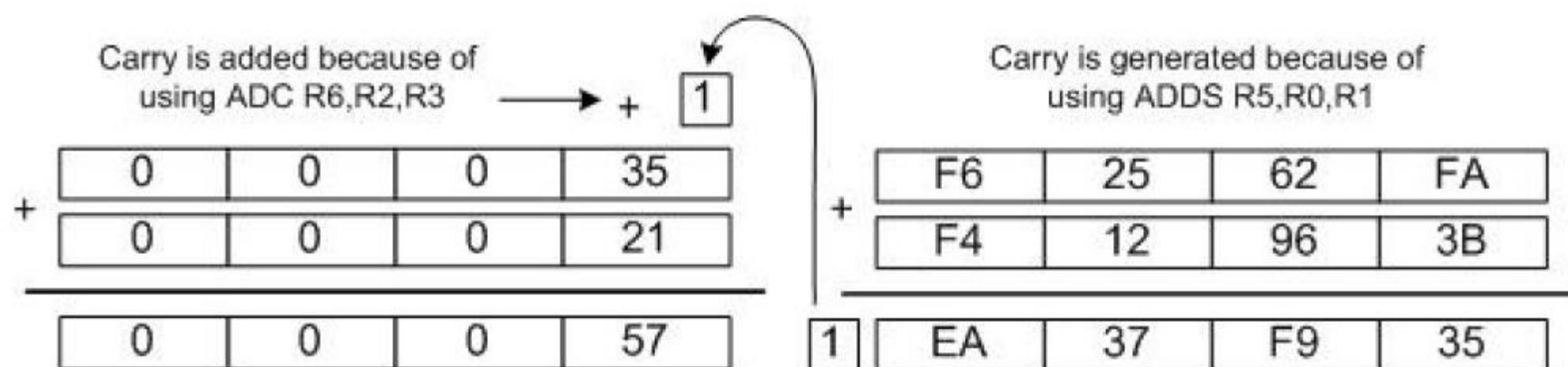
---

**DR MANOJ TOLANI (PHD-IIIT ALLAHABAD)**  
**ASSISTANT PROFESSOR (DEPARTMENT OF ICT)**

## CASE 2: Addition of multiword numbers

Write a program which adds 0x35F62562FA to 0x21F412963B:

0x35F62562FA to 0x21F412963B:



## **CASE 2: Addition of multiword numbers**

Analyze the following program which adds 0x35F62562FA to 0x21F412963B:

## CASE 2: Addition of multiword numbers

Analyze the following program which adds 0x35F62562FA to 0x21F412963B:

```
LDR    R0,=0xF62562FA      ;R0 = 0xF62562FA
LDR    R1,=0xF412963B      ;R1 = 0xF412963B
MOV    R2,#0x35            ;R2 = 0x35
MOV    R3,#0x21            ;R3 = 0x21
ADDS   R5,R1,R0            ;R5 = 0xF62562FA + 0xF412963B
;now C = 1
|ADC    R6,R2,R3          ;R6 = R2 + R3 + C
;  = 0x35 + 21 + 1 = 0x57
```

Show the steps involved for the following cases:

a)

```
MOV R2,#0x4F ;R2 = 0x4F  
MOV R3,#0x39 ;R3 = 0x39  
SUBS R4,R2,R3 ;R4 = R2 - R3
```

b)

```
MOV R2,#0x4F ;R2 = 0x4F  
SUBS R4,R2,#0x05 ;R4 = R2 - 0x05
```

Analyze the following instructions:

LDR	R2,=0x88888888	;R2 = 0x88888888
LDR	R3,=0x33333333	;R3 = 0x33333333
SUBS	R4,R2,R3	;R4 = R2 – R3

Analyze the following instructions:

MOV	R1,#0x4C	;R1 = 0x4C
MOV	R2,#0x6E	;R2 = 0x6E
SUBS	R0,R1,R2	;R0 = R1 – R2

---

Analyze the following program which subtracts 0x21F62562FA from 0x35F412963B:

---

Analyze the following program which subtracts 0x21F62562FA from 0x35F412963B:

```
LDR    R0,=0xF62562FA      ;R0 = 0xF62562FA,  
; notice the syntax for LDR  
LDR    R1,=0xF412963B      ;R1 = 0xF412963B  
MOV    R2,#0x21             ;R2 = 0x21  
MOV    R3,#0x35             ;R3 = 0x35  
SUBS   R5,R1,R0             ;R5 = R1 - R0  
;  = 0xF412963B - 0xF62562FA, and C = 0  
SBC    R6,R3,R2             ;R6 = R3 - R2 - 1 + C  
;  = 0x35 - 0x21 - 1 + 0 = 0x13
```

# MULTIPLICATION

---

MUL Rd, Rn, Rm ;  $Rd = Rn \times Rm$  (**Multiply**)

MLA Rd, Rs1, Rs2, Rs3 ;  $Rd = (Rs1 \times Rs2) + Rs3$  (**Multiply and Accumulate**)

MLS Rd, Rm, Rs, Rn ;  $Rd = Rn - (Rs \times Rm)$  (**Multiply and Subtract**)

UMULL RdLo, RdHi, Rn, Rm ;  $RdHi:RdLo = Rm \times Rn$  (**Unsigned Multiply Long**)

SMULL Rdlo, Rdhi, Rn, Rm ;  $Rdhi:Rdlo = Rm \times Rn$  (**Signed Multiply Long**)

Flags not affected.

# LOGICAL INSTRUCTIONS

---

AND Rd, Rn, Op2

;Rd = Rn ANDed Op2

ORR Rd, Rn, Op2

;Rd = Rn ORed with Op2

ORN Rd, Rn, Op2

;Rd = Rn ORed with 1's comp of Op2

EOR Rd, Rn, Op2

;Rd = Rn XORed Op2

Flags not affected.

Use S suffix to update flags

TEQ Rn, Op2

;performs Rn Ex-OR Op2

TST Rn, Op2

;performs Rn AND Op2

Flags N,Z affected



# Embedded System and Design

---

**DR MANOJ TOLANI (PHD-IIIT ALLAHABAD)**  
**ASSISTANT PROFESSOR (DEPARTMENT OF ICT)**

# Load and Store Multiple

---

## LDM Load Multiple registers

*Flags:* Unaffected.

*Format:* LDM Rn, {Rx,Ry,...}

## STM

## Store Multiple

*Flags:* Unaffected.

*Format:*

STM Rn, {Rx,Ry,...}

# Load and Store Multiple

LDR R1, =0x10000000  
LDM R1, {R2,R4,R6}

0x10000000	98
0x10000001	76
0x10000002	54
0x10000003	32
0x10000004	FF
0x10000005	FF
0x10000006	FF
0x10000007	FE
0x10000008	89
0x10000009	AB
0x1000000A	CD
0x1000000B	EF
0x1000000C	
0x1000000D	
0x1000000E	
0x1000000F	

# Load and Store Multiple

LDR R1, =0x10000000  
LDM R1, {R2,R4,R6}

R2 = 0x32547698  
R4 = 0xFFFFFFF  
R6 = 0xEFCDAB89  
R1= 0x10000000

-----

In case of !  
LDM R1!, {R2,R4,R6}  
R1=0x1000000C

0x10000000	98
0x10000001	76
0x10000002	54
0x10000003	32
0x10000004	FF
0x10000005	FF
0x10000006	FF
0x10000007	FE
0x10000008	89
0x10000009	AB
0x1000000A	CD
0x1000000B	EF
0x1000000C	
0x1000000D	
0x1000000E	
0x1000000F	

# Load and Store Multiple

```
LDR R1, =0x10000000  
LDR R2, = 0x32547698  
LDR R4, =0xFFFFFFF  
LDR R6, = 0xEFCDAB89  
STM R1, {R2,R4,R6}
```

0x10000000	
0x10000001	
0x10000002	
0x10000003	
0x10000004	
0x10000005	
0x10000006	
0x10000007	
0x10000008	
0x10000009	
0x1000000A	
0x1000000B	
0x1000000C	
0x1000000D	
0x1000000E	
0x1000000F	

# Load and Store Multiple

LDR R1, =0x10000000  
LDR R2, = 0x32547698  
LDR R4, =0xFFFFFFF  
LDR R6, = 0xEFCDAB89  
STM R1, {R2,R4,R6}

R1= 0x10000000

In case of !

STM R1!, {R2,R4,R6}  
R1=0x1000000C

0x10000000	98
0x10000001	76
0x10000002	54
0x10000003	32
0x10000004	FF
0x10000005	FF
0x10000006	FF
0x10000007	FE
0x10000008	89
0x10000009	AB
0x1000000A	CD
0x1000000B	EF
0x1000000C	
0x1000000D	
0x1000000E	
0x1000000F	

# Load and Store Multiple

---

**LDMDB**                    Load Multiple registers and  
**Decrement Before each access**  
*Flags:* Unaffected.

*Format:*      LDMDB Rn,{Rx,Ry,...}

**STMDB**                    Store Multiple register and  
**Decrement Before**  
*Flags:* Unaffected.

*Format:*      STMDB Rn,{Rx,Ry,...}

# Load and Store Multiple

LDR R1, =0x1000000C  
LDMDB R1, {R2,R4,R6}

0x10000000	98
0x10000001	76
0x10000002	54
0x10000003	32
0x10000004	FF
0x10000005	FF
0x10000006	FF
0x10000007	FE
0x10000008	89
0x10000009	AB
0x1000000A	CD
0x1000000B	EF
0x1000000C	
0x1000000D	
0x1000000E	
0x1000000F	

# Load and Store Multiple

LDR R1, =0x1000000C  
LDMDB R1, {R2,R4,R6}

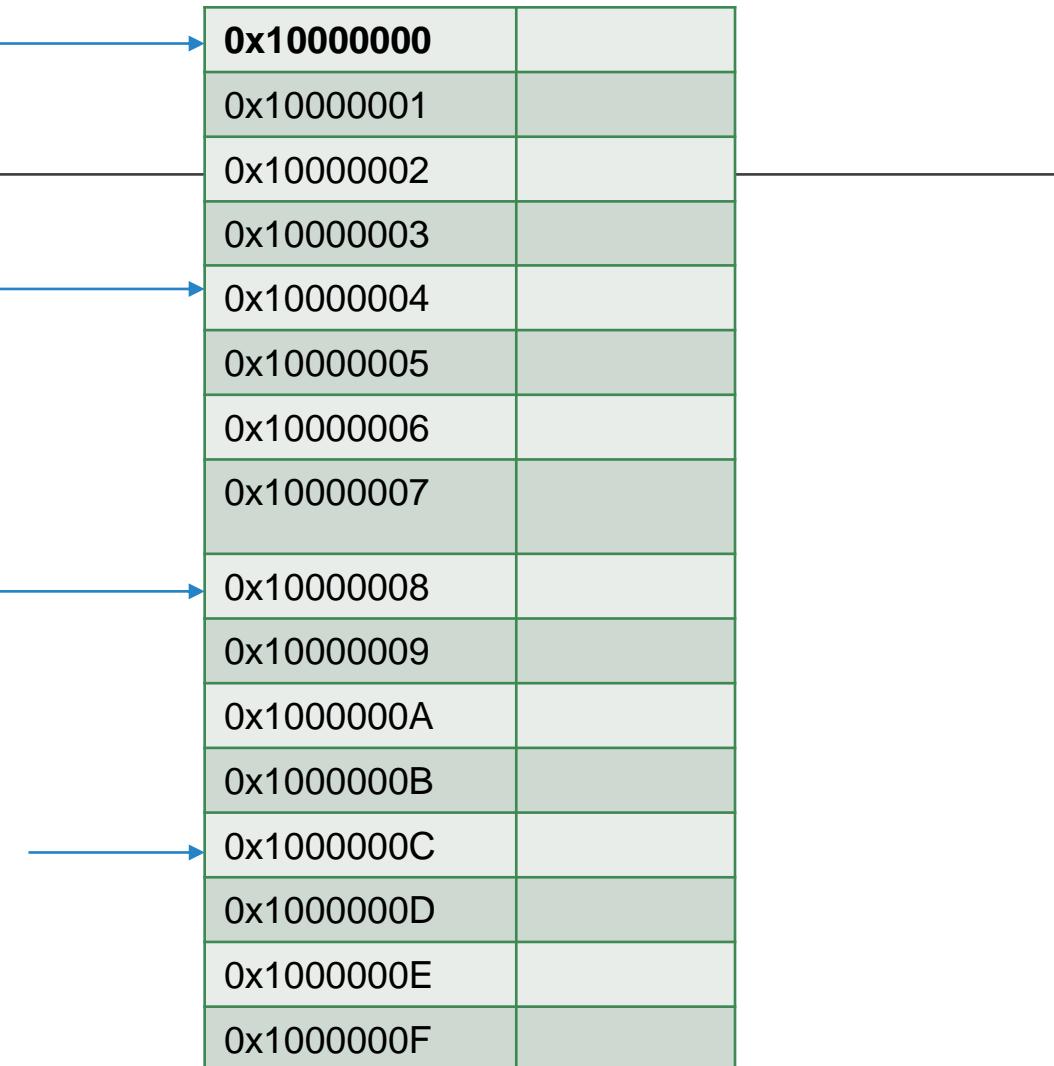
R2 = 0x32547698  
R4 = 0xFFFFFFF  
R6 = 0xEFCDAB89  
R1= 0x1000000C

-----  
In case of !  
LDMDB R1!, {R2,R4,R6}  
R1=0x10000000

0x10000000	98
0x10000001	76
0x10000002	54
0x10000003	32
0x10000004	FF
0x10000005	FF
0x10000006	FF
0x10000007	FE
0x10000008	89
0x10000009	AB
0x1000000A	CD
0x1000000B	EF
0x1000000C	
0x1000000D	
0x1000000E	
0x1000000F	

# Load and Store Multiple

```
LDR R1, =0x1000000C  
LDR R2, = 0x32547698  
LDR R4, =0xFFFFFFF  
LDR R6, = 0xEFCDAB89  
STMDB R1, {R2,R4,R6}
```



# Load and Store Multiple

```
LDR R1, =0x1000000C  
LDR R2, = 0x32547698  
LDR R4, =0xFFFFFFF  
LDR R6, = 0xEFCDAB89  
STMDB R1, {R2,R4,R6}
```

R1= 0x1000000C

-----

In case of !  
STMDB R1!, {R2,R4,R6}  
R1=0x10000000

0x10000000	98
0x10000001	76
0x10000002	54
0x10000003	32
0x10000004	FF
0x10000005	FF
0x10000006	FF
0x10000007	FE
0x10000008	89
0x10000009	AB
0x1000000A	CD
0x1000000B	EF
0x1000000C	
0x1000000D	
0x1000000E	
0x1000000F	

# GCD

---

```
While (a != b)
{
    If (a > b)
        a = a-b;
    Else if (b > a)
        b = b-a;
}
```

# GCD

## Using SUBHI and SUBLO

```
LDR R0, =NUM1
LDR R1, =NUM2
LDR R0,[R0]
LDR R1,[R1]
UP   CMP R0, R1
BEQ EXIT
SUBHI R0,R1
SUBLO R1,R0
B UP
EXIT LDR R2,=GCD
STR R0, [R2]
```

## Without using SUBHI and SUBLO

```
Reset_Handler
    LDR R0, =NUM1
    LDR R1, =NUM2
    LDR R0, [R0]
    LDR R1, [R1]
    UP   CMP R0, R1
    BEQ EXIT
    BHI HIGH
    BLO LOW
    HIGH SUB R0, R1
    B UP
    LOW SUB R1, R0
    B UP
    EXIT
    LDR R2,=GCD
    STR R0, [R2]
    STOP B STOP
    NUM1 DCD 35
    NUM2 DCD 14
    AREA database, DATA, READWRITE
    GCD      DCD 0
    END
```

# BCD to Hex

If BCD value 0xXYZW

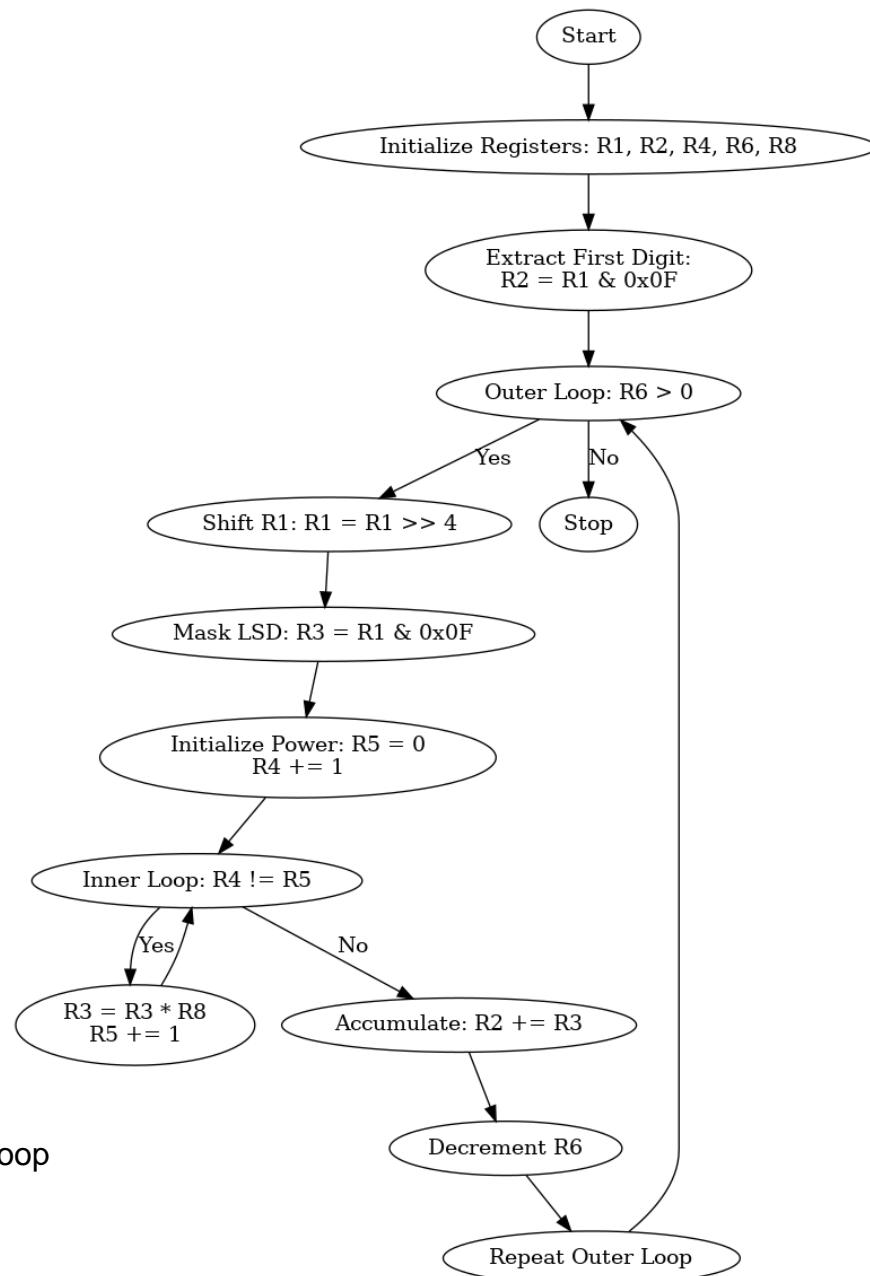
Its Hexadecimal Value will be

$$\text{Hexadecimal} = X \times (0xA)^3 + Y \times (0xA)^2 + Z \times (0xA)^1 + W \times (0xA)^0$$

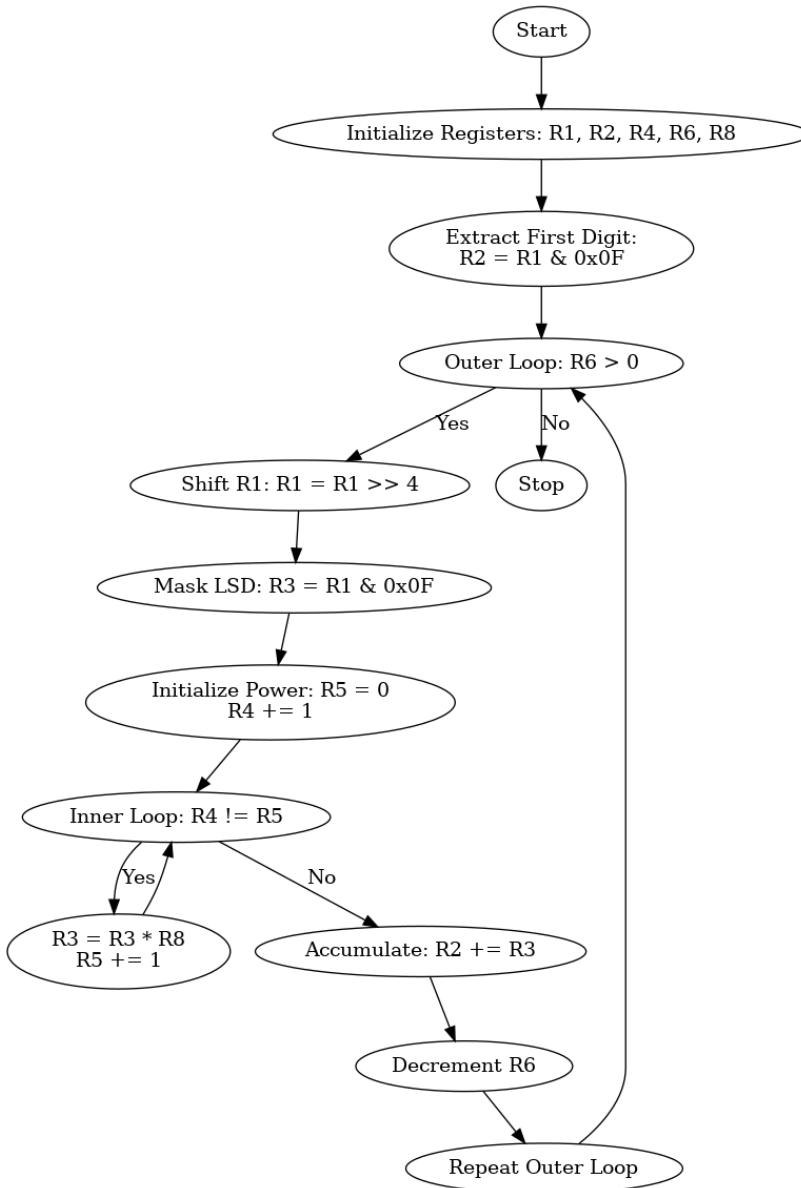
Assume BCD Value =0x378

$$\text{Hexadecimal} = 3 \times (0xA)^2 + 7 \times (0xA)^1 + 8 \times (0xA)^0$$

```
R1 = bcd_value          # Input BCD value
R2                         # Accumulated result (Initially loaded with the first digit using AND Masking)
R4 = 0x0                  # Initialize Power
R6 = 2                     # 3 digits, loop executes 2 times because one digit is accumulated in R2 before the loop
R8 = 0xA                  # Base value for conversion (Decimal)
```



# BCD to Hex



```

Reset_Handler
  LDR R1, =0x378
  AND R2, R1, #0x0F ; R2 accumulates final result
  MOV R3, #0 ; Temporary register for LSD storage
  MOV R4, #0 ; Power Count Value
  MOV R6, #2 ; 3 Digits so 3-1 times execution
  MOV R8, #0xA ; Base Value

LOOP
  ADD R2, R3
  LSR R1, #4
  AND R3, R1, #0x0F
  ADD R4, #1
  MOV R5, #0 ; Power Initialization

LOOP1
  MUL R3, R8
  ADD R5, #1
  TEQ R4, R5
  BNE LOOP1
  SUBS R6, #1
  BNE LOOP

  ADD R2, R3
  LDR R7, =DST
  STR R2, [R7]

STOP
  B STOP
  AREA dataseg, DATA, READWRITE
DST DCD 0x0
END
  
```

$0xF7EAB \div 0xA = Q_1 = 0x18CAA, R_1 = 0x7$   
 $0x18CAA \div 0xA = Q_2 = 0x27AA, R_2 = 0x6$   
 $0x27AA \div 0xA = Q_3 = 0x3F7, R_3 = 0x4$   
 $0x3F7 \div 0xA = Q_4 = 0x65, R_4 = 0x5$   
 $0x65 \div 0xA = Q_5 = 0xA, R_5 = 0x1$   
 $0xA \div 0xA = Q_6 = 0x1, R_6 = 0x0$

Final BCD: 1 0 1 5 4 6 7

## HEX to BCD Conversion

- Continuous Division by base value 0xA

Write an ALP to Convert Hex No. 0xF7EAB to BCD

Loop	Dividend (R1)	Base 0xA (R2)	Quotient (R0=R1/R2)	Remainder (R4)	Place Value Shift (R6)	Shifted Remainder	Accumulated Result (R5)
First Iteration	0XF7EAB	0xA	0X18CAA	0x7	0 (0x1)	0x7	0x7
Second	0X18CAA		0X27AA	0x6	4 (0x10)	0x60	0x67
Third	0X27AA		0X3F7	0x4	8 (0x100)	0x400	0x467
Fourth	0X3F7		0X65	0x5	12 (0x1000)	0x5000	0x5467
Fifth	0X65		0XA	0x1	16 (0x10000)	0x10000	0x15467
Sixth	0XA		0X1	0x0	20 (0x100000)	0x0	0x15467
	Exit Loop		Final Place Value	0x1000000	24 (0x1000000)	-	0x1015467
Final BCD		-	-	-	-	-	0x1015467

## HEX to BCD Conversion

```
Reset_Handler
    LDR R1, =0xF7EAB
    MOV R2, #0x0A
    MOV R5, #0x0
    MOV R6, #0

LOOP    UDIV R0, R1, R2 ; R0=Quotient
        MUL R3, R0, R2
        SUB R4, R1, R3 ; R4=Remainder
        LSL R4, R6 ; Multiplication of 0x1, 0x10 ... based on place value
        ADD R5, R4 ; Accumulation of Final Value ... final Result
        ADD R6, #4 ; This is for changing place value from LSB to MSB side
        MOV R1, R0 ; Making Quotient to Dividend
        CMP R0, R2 ; Comparing Quotient to base value to Exit from the loop
        BHS LOOP ; Branch if Value Higher or Same to Base value

        LSL R0, R6 ; Final Mumtiplication of Place Value
        ADD R5, R0 ; Final Accumulation of result

        LDR R7, =BCD
        STR R5, [R7]

STOP B STOP
        AREA daseseg, DATA, READWRITE
BCD      DCD 0
END
```

## BCD and ASCII Conversion

### ASCII to unpacked BCD conversion

each ASCII number is ANDed with  
“0000 1111” (0x0F)

For Hexadecimal

41 A

42 B

43 C

44 D

45 E

46 F

Key	ASCII	Binary(hex)	BCD (unpacked)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	45	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

## ASCII to packed BCD conversion

Key	ASCII	Unpacked BCD	Packed BCD
2	32	00000010	
7	37	00000111	00100111 (0x27)

## Packed BCD to ASCII conversion

Packed BCD	Unpacked BCD	ASCII
0x29	0x02 & 0x09	0x32 & 0x39
0010 1001	0000 0010 & 0000 1001	011 0010 & 011 1001

# Compare Instructions

### CMP Compare

**Flags:** Affected: V, N, Z, C.

**Format:** CMP Rn,Op2 ;sets flags as if "Rn-Op2"

## **CMN Compare Negative**

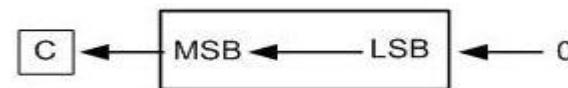
**Flags:** Affected: V, N, Z,C.

**Format:** CMN Rn,Op2 ;sets flags as if "Rn + Op2"

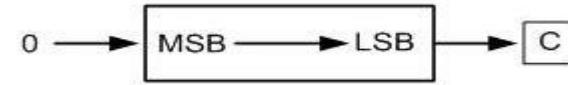
# SHIFT AND ROTATE

---

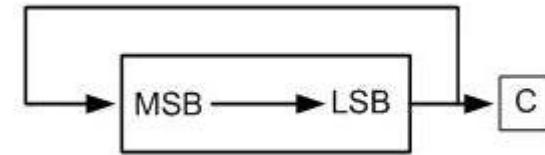
LSL Rd, Rn, Op2



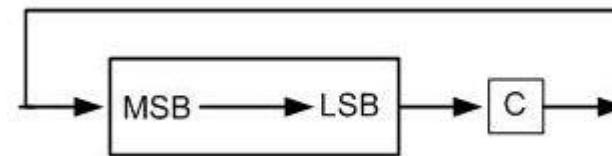
LSR Rd, Rn, Op2



ASR Rd, Rn, Op2



ROR Rd, Rn, Op2



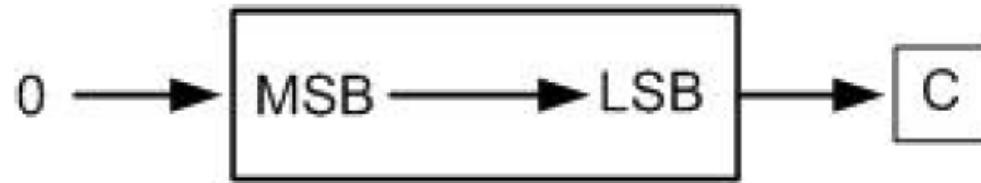
RRX Rd, Rm ;Rd = rotate Rm right 1 bit position

Flags not affected.

Use S suffix to update flags

## Rotate and Barrel Shifter

### LSR



Show the result of LSR in the following:

MOV R0,#0x9A ;R0 = 0x9A

MOVS R1,R0,LSR #3 ;shift R0 to right 3 times

*Example 1:*

LDR R2,=0x00001000

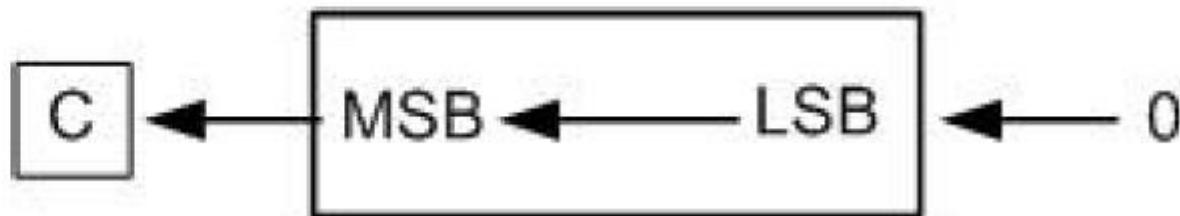
LSR R0,R2,#8 ;R0=R2 is shifted right 8 times

Show the results of LSR in the following:

LDR R0,=0x88 ;R0=0x88

MOVS R1,R0,LSR #3 ;shift R0 right three times (R1 = 0x11)

*LSL*



Show the effects of LSL in the following:

LDR R1,=0x0F000006

MOVS R2,R1,LSL #8

Show the results of LSL in the following:

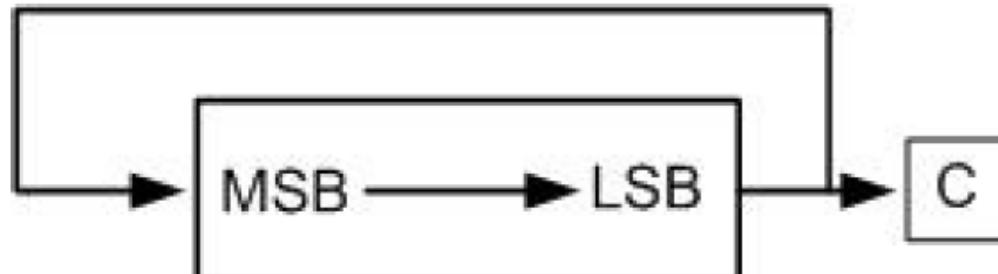
TIMES EQU 0x5

LDR R1,#0x7 ;R1=0x7

MOV R2,#TIMES ;R2=0x05

MOV R1,R1,LSL R2 ;shift R1 left R2 number of times

## ROR (rotate right)



MOV R1,#0x36

;R1 = 0000 0000 0000 0000 0000 0000 0011 0110

MOVS R1,R1,ROR #1

MOVS R1,R1,ROR #1

MOVS R1,R1,ROR #1

MOV R1,#0x36

;R1 = 0000 0000 0000 0000 0000 0000 0011 0110

MOVS R1,R1,ROR #1

;R1 = 0000 0000 0000 0000 0000 0000 0001 1011 C=0

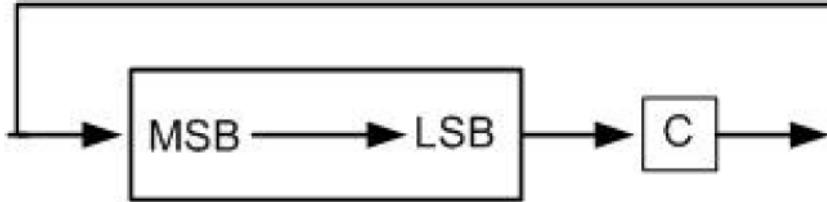
MOVS R1,R1,ROR #1

;R1 = 1000 0000 0000 0000 0000 0000 0000 1101 C=1

MOVS R1,R1,ROR #1

;R1 = 1100 0000 0000 0000 0000 0000 0000 0110 C=1

## RRX rotate right through carry



;assume C=0

```
MOV R2,#0x26
```

;R2 = 0000 0000 0000 0000 0000 0000 0010 0110

```
MOVS R2,R2,RRX
```

;R2 = 0000 0000 0000 0000 0000 0000 0001 0011 C=0

```
MOVS R2,R2,RRX
```

;R2 = 0000 0000 0000 0000 0000 0000 0000 1001 C=1

```
MOVS R2,R2,RRX
```

;R2 = 1000 0000 0000 0000 0000 0000 0100 C=1

Not affect Carry (Rotation Instruction)

MOV R0,#0xAA,#2

MOV R1,#0x20,#28

MOV R4,#0x99,#6

MOV R2,#0x55,#24

MVN R0,#0xAA,#2

MVN R1,#0x20,#28

MVN R4,#0x99,#6

MVN R2,#0x55,#24

1. Find the contents of R3 after executing the following code:

MOV R0,#0x04

MOV R3,R0,LSR #2

2. Find the contents of R4 after executing the following code:

LDR R1,=0xA0F2

MOV R2,#0x3

MOV R4,R1,LSR R2

3. Find the contents of R3 after executing the following code:

LDR R1,=0xA0F2

MOV R2,#0x3

MOV R3,R1,LSL R2

---

4. Find the contents of R5 after executing the following code:

SUBS R0,R0,R0

MOV R0,#0xAA

MOV R5,R0,ROR #4

5. Find the contents of R0 after executing the following code:

LDR R2,=0xA0F2

MOV R1,0x4

MOV R0,R2,RRX R1

6. Give the result in R1 for the following:

MVN R1,#0x01, #2

7. Give the result in R2 for the following:

MVN R2,#0x02, #28

---

4. Find the contents of R5 after executing the following code:

SUBS R0,R0,R0

MOV R0,#0xAA

MOV R5,R0,ROR #4

5. Find the contents of R0 after executing the following code:

LDR R2,=0xA0F2

MOV R1,0x4

MOV R0,R2,RRX R1

6. Give the result in R1 for the following:

MVN R1,#0x01, #2

7. Give the result in R2 for the following:

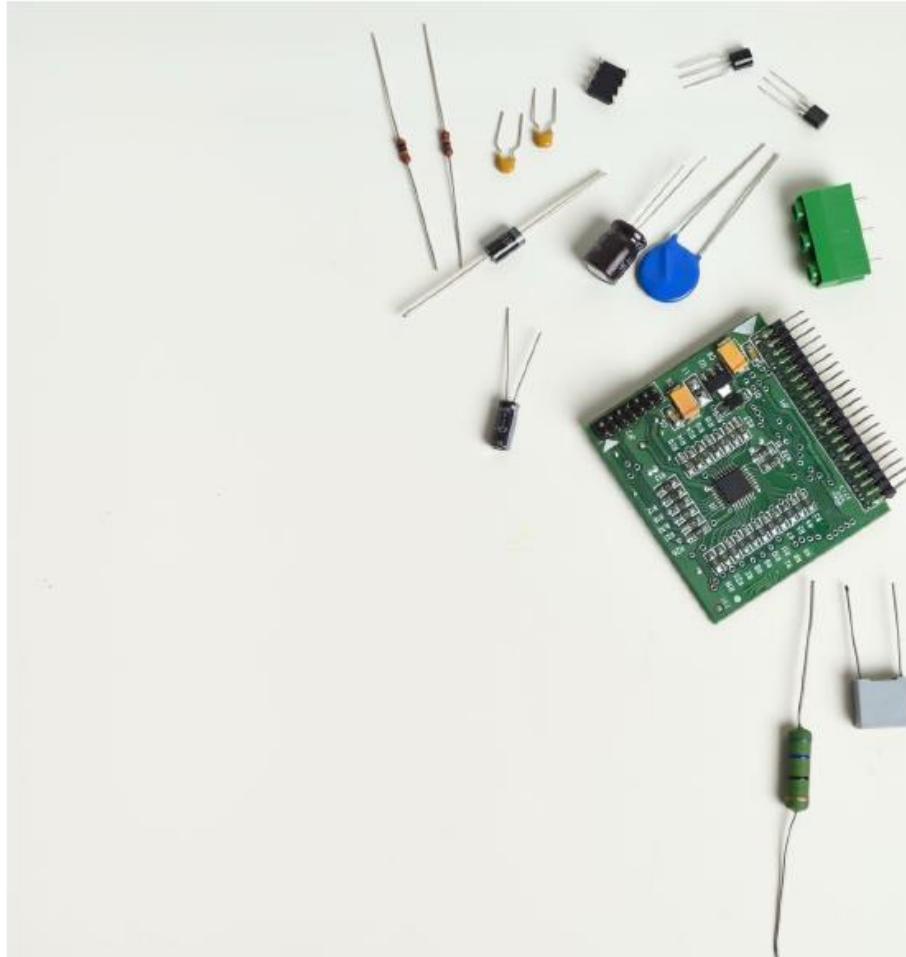
MVN R2,#0x02, #28



# Embedded System and Design

---

**DR MANOJ TOLANI (PHD-IIIT ALLAHABAD)**  
**ASSISTANT PROFESSOR (DEPARTMENT OF ICT)**



# MY INTRODUCTION

- First, let me introduce myself. My name is **Manoj Tolani**. I joined MIT, Manipal (MAHE) in **April 2023**.
- I am currently **Assistant Professor in Information and Communication Technology Department**.
- I received an **MTech degree (Specialization: Digital Systems)** from the **Madan Mohan Malviya Engineering College, Gorakhpur** (Now MMMUT) in 2012.
- After MTech, I joined **PSIT Engineering College** as an **Assistant Professor in 2012**. I worked for **3.5 Years at PSIT**.
- I attempted **GATE Exam in 2015**, and my **AIR was 1214**. I also qualified for UGC-NET also in 2015.
- In **2016**, I joined **IIIT Allahabad** as a research scholar for a PhD Degree.
- I **completed my Ph.D. in Feb 2021**. I worked **6 months at IIIT-Allahabad as Teaching-cum-Research Associate**.
- After that, I joined **Atria Institute of Technology, Bangalore** as an Assistant Professor. I worked for **2 Years at AIT**, and after that, I joined this college (**MIT, Manipal**) in **April 2023**.
- My research interest includes **Wireless Sensor Networks, IoT, Photonics, and Machine Learning**.
- I have published **18+ research papers in SCI/Scopus Journals/Conferences** (IEEE, Elsevier, Springer etc.)

# BCD Addition

## BCD Addition of 0x9938 and 0x9939.

### Initial Setup:

Registers R0 and R1 are initialized with 0x9938 and 0x9939, also initialized as follows:

- R8: Shift amount register (Initially=0 Count increases in each loop 4, 8)
- R7: Carry register for intermediate carry bits.
- R6: Stores the final result.
- R4: Loop counter (set to 8 for processing 8 nibbles).
- R10: Holds the decimal threshold (10 or 0xA).

### First AND Operation (Extract Lowest Nibble):

Perform AND operation on R0 and R1 with 0xF to extract the least significant nibbles.

- R2 = R0 AND 0xF = 8
- R3 = R1 AND 0xF = 9

Shift R0, and R1 right side 4 bits for next Nibble

### Add the Nibbles (R2 + R3):

Add the extracted nibbles and include the carry from R7.

- R5 = R2 + R3 = 8 + 9 = 17 (0x11)
- Carry = 0 (Initially carry R7= 0).

Reset Carry R7 = 0

N1 → R0  
N2 → R1  
Count → R4=#8  
R6, R7, R8 =0, R10=0xA

Mask Target Digits (R2, R3)  
Add Numbers  
Add Carry if Any (R7)  
Reset Carry

Compare Result if >0x0A

Subtract  
Result-0x0A  
Accumulate Carry

Position the Result to Correct Decimal  
Position  
Rotate for Next Target

No

Yes

## Check for Carry

Compare R5 with R10 (10):

- If  $R5 \geq R10$ , go to the DOWN section.
- Otherwise, continue to the [UP section](#).

In this case,  $R5 = 17$ , so the program branches to DOWN.

## Handle Carry (DOWN Section)

Subtract 10 from R5 and increment the carry register R7.

- $R5 = R5 - 10 = 7$
- $R7 = 1$  (Carry for Next Nibble)

Go to the [UP section](#).

## Store Partial Result (UP Section):

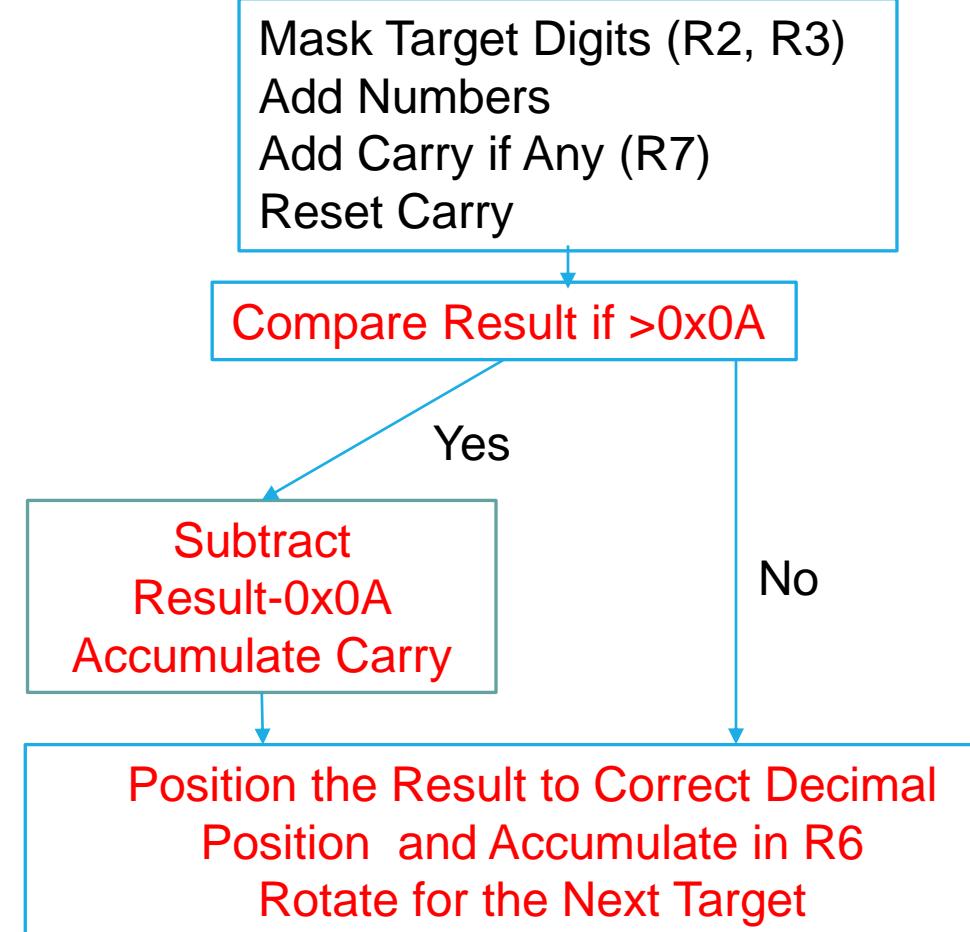
Left-shift R5 by R8 and add to the result register R6.

- $R6 = R6 + (R5 \ll R8)$
- Increment R8 by 4 (to process the next nibble).
- Decrement R4 (loop counter).

After processing the first nibble:

- $R6 = 0 + (7 \ll 0) = 7$
- $R8 = 4$
- $R4 = 7$

N1 → R0  
N2 → R1  
Count → R4=#8  
R6, R7, R8 =0



```
Reset_Handler
    LDR R0, =0x9938
    LDR R1, =0x9939
    MOV R8, #0
    MOV R7, #0
    MOV R6, #0
    MOV R4, #8
    MOV R10, #0xA

LOOP    AND R2, R0, #0xF
        AND R3, R1, #0xF
        LSR R0, #4
        LSR R1, #4
        ADD R5, R2, R3
        ADD R5, R7
        MOV R7, #0
        CMP R5, R10
        BHS DOWN

UP      LSL R5, R8
        ADD R6, R5
        ADD R8, #4
        SUBS R4, #1
        BNE LOOP
        B EXIT

        DOWN     SUB R5, #0xA
                  ADD R7, #1
                  B UP

        EXIT    LDR R9, =DST
                  STR R6, [R9]

        STOP    B STOP
                  AREA data1, DATA, READWRITE
DST    DCD 0
        END
```

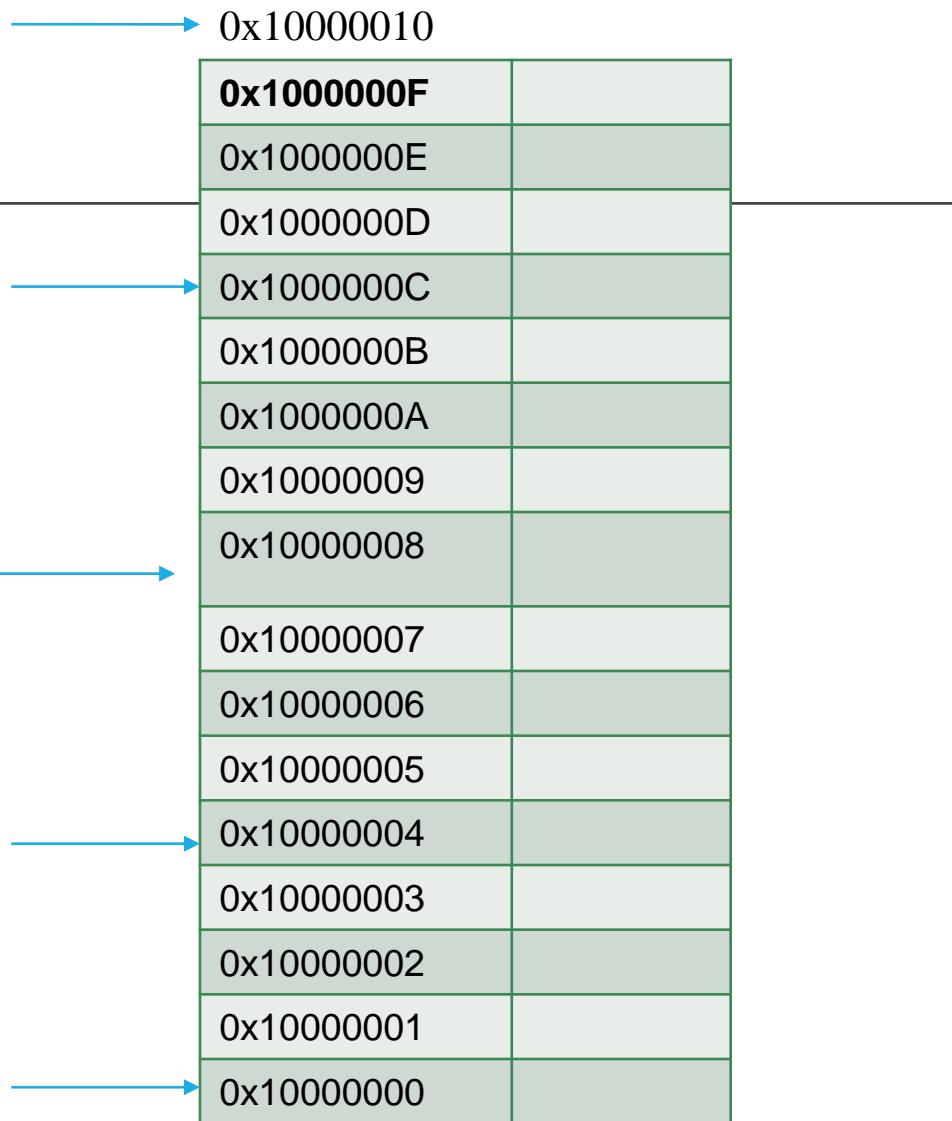
# PUSH and POP

## Example

PUSH {R1,R3-R5}; same as PUSH {R1,R3,R4,R5}

- Example Program

```
LDR R13, =0x10000010  
LDR R1, =0x12345678  
LDR R8, =0xABCDDEF12  
LDR R3, =0x56AB9678  
LDR R5, =0x12CD3412  
PUSH {R1,R5,R8, R3}
```



# PUSH and POP

## Example

PUSH {R1,R3-R5}; same as PUSH {R1,R3,R4,R5}

- Example Program

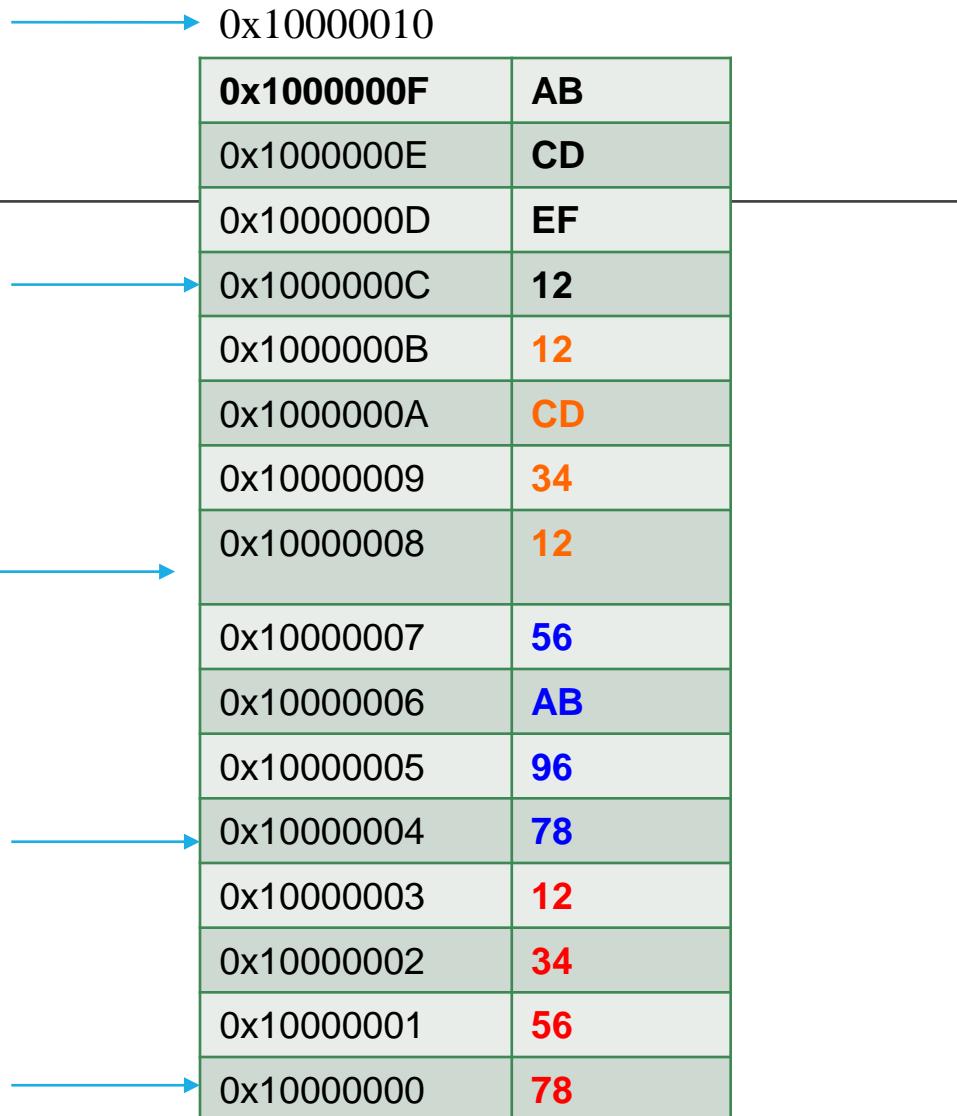
```
LDR R13, =0x10000010  
LDR R1, =0x12345678  
LDR R8, =0xABCDE12  
LDR R3, =0x56AB9678  
LDR R5, =0x12CD3412  
PUSH {R1,R5,R8, R3}
```

0x100000F	AB
0x100000E	CD
0x100000D	EF
0x100000C	12
0x100000B	12
0x100000A	CD
0x1000009	34
0x1000008	12
0x1000007	56
0x1000006	AB
0x1000005	96
0x1000004	78
0x1000003	12
0x1000002	34
0x1000001	56
0x1000000	78

# PUSH and POP

Example Program

```
LDR R13, =0x10000000  
POP {R2,R6,R7,R4}
```



# PUSH and POP

Example Program

```
LDR R13, =0x10000000  
POP {R2,R6,R7,R4}
```

After execution

```
R7=0xABCDEF12  
R4=0x56AB9678  
R2=0x12345678  
R6=0x1234CD12
```

0x100000F	AB
0x100000E	CD
0x100000D	EF
0x100000C	12
0x100000B	12
0x100000A	CD
0x1000009	34
0x1000008	12
0x1000007	56
0x1000006	AB
0x1000005	96
0x1000004	78
0x1000003	12
0x1000002	34
0x1000001	56
0x1000000	78

# Fully Ascending Stack

```
LDR R13, =0x10000000
LDR R2, = 0x32547698
LDR R4, =0xFFFFFFF
LDR R6, = 0x32547698
STM R13!, {R2,R4,R6}
R13= 0x1000000C
MOV R2, #0
MOV R4, #0
MOV R6, #0
LDMDB R13!, {R2,R4,R6}
R13=0x10000000
```

0x10000000	98
0x10000001	76
0x10000002	54
0x10000003	32
0x10000004	FF
0x10000005	FF
0x10000006	FF
0x10000007	FE
0x10000008	89
0x10000009	AB
0x1000000A	CD
0x1000000B	EF
0x1000000C	
0x1000000D	
0x1000000E	
0x1000000F	

SP increments for PUSH  
SP decrements for POP

# Fully Descending Stack

```
LDR R13, =0x1000000C  
LDR R2, = 0x32547698  
LDR R4, =0xFFFFFFF  
LDR R6, = 0x32547698  
STMDB R13!, {R2,R4,R6}  
R13= 0x10000000  
MOV R2, #0  
MOV R4, #0  
MOV R6, #0  
LDM R13!, {R2,R4,R6}  
R13=0x1000000C
```

SP decrements for PUSH  
SP increments for POP

0x10000000	98
0x10000001	76
0x10000002	54
0x10000003	32
0x10000004	FF
0x10000005	FF
0x10000006	FF
0x10000007	FE
0x10000008	89
0x10000009	AB
0x1000000A	CD
0x1000000B	EF
0x1000000C	
0x1000000D	
0x1000000E	
0x1000000F	

# Function Call & Return

**BL Branch with Link (this is Call instruction)**

*Flags:* Unchanged.

*Format:* BL Subroutine\_Addr ;transfer

**BX Branch Indirect (BX LR is used for  
Return)**

*Flags:* Unchanged.

*Format:* BX Rm ;BX LR is used for Return  
from a subroutine

# Function Call & Return

```
BL HEX_BCD ; call HEX_BCD  
MOV R0,R1
```

-----

-----

HERE B HERE

```
HEX_BCD MOV R4,R5
```

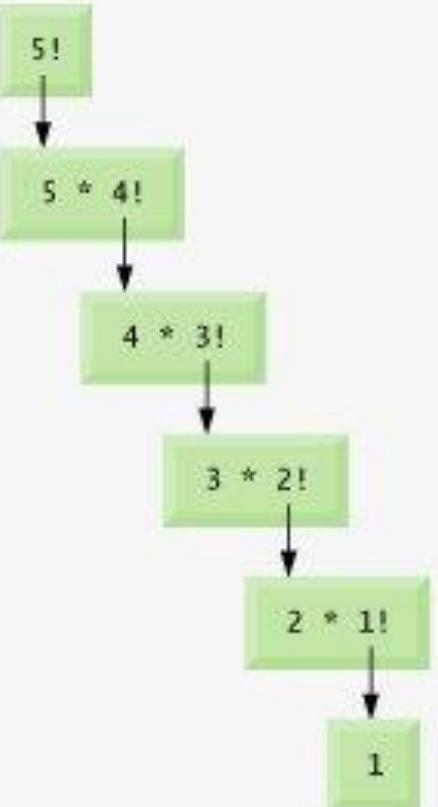
-----

-----

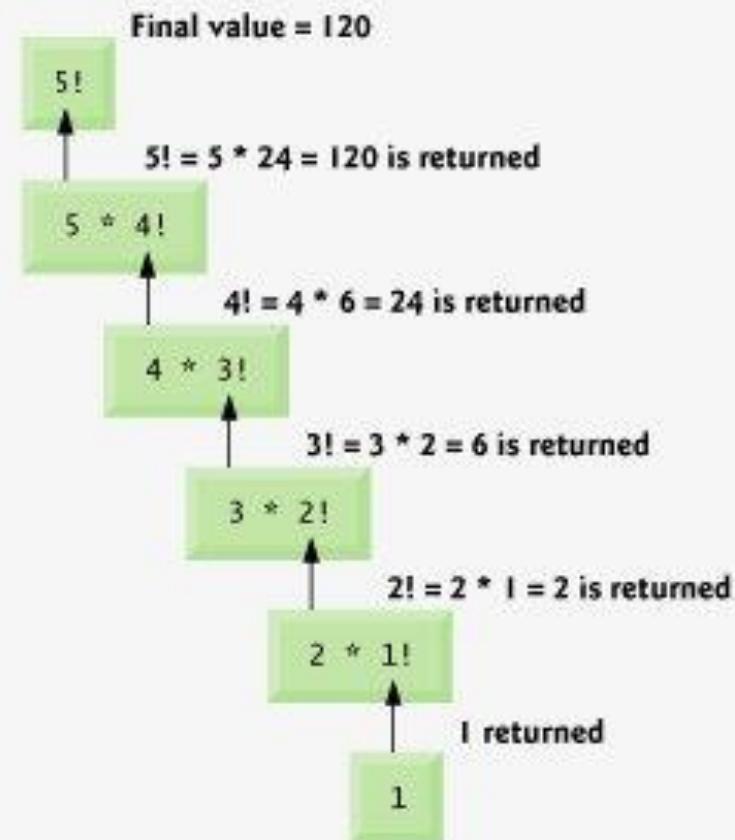
BX LR ; Return

```
    R0      0x00000000
    R1      0x00000020
    R2      0x00000020
    R3      0x00000000
    R4      0x00000011
    R5      0x00000000
    R6      0x00000000
    R7      0x00000000
    R8      0x00000000
    R9      0x00000000
    R10     0x00000000
    R11     0x00000000
    R12     0x00000000
    R13 ... 0x01001000
    R14 ... 0x00000011
    R15 ... 0x00000012
    xPSR   0x01000000
    .
.
.
Reset_Handler
    MOV R2, #32
    BL HEX_BCD
    MOV R1, R2
HERE B HERE
HEX_BCD MOV R4, LR
    MOV R5, R1
    BX LR
end
```

# Factorial using Recursion



(a) Sequence of recursive calls.



(b) Values returned from each recursive call.

# Factorial using Recursion

Reset\_Handler

	LDR R1,NUM		POP {LR}
00000013	LDR R13,=0X10001000	00000029	POP {R1}
	BL FACT1		MUL R2,R1,R2
	LDR R1,=FACT		BX LR
	STR R2,[R1]	EXIT	MOV R2,#1
STOP	B STOP		BX LR
FACT1	CMP R1,#1	NUM	DCD 0X07
	BEQ EXIT		
	PUSH {R1}		AREA DATA1, DATA, READWRITE
	PUSH {LR}		
	SUB R1, #1	FACT	DCD 0
	BL FACT1		END

0x10000FCC:	00000000
0x10000FD0:	00000029
0x10000FD4:	00000002
0x10000FD8:	00000029
0x10000FDC:	00000003
0x10000FE0:	00000029
0x10000FE4:	00000004
0x10000FE8:	00000029
0x10000FEC:	00000005
0x10000FF0:	00000029
0x10000FF4:	00000006
0x10000FF8:	00000013
0x10000FFC:	00000007
0x10001000:	00000000
0x10001004:	00000000

## Convert NEG to POSITIVE in array

---

```
LDR R0, =ARRAY
MOV R1,#10
UP LDR R2, [R0], #4
CMP R2, #0
RSBLT R2, #0
STRLT R2, [R0, #-4]
SUB R1, #0
TEQ R1, #0
BNE UP
```

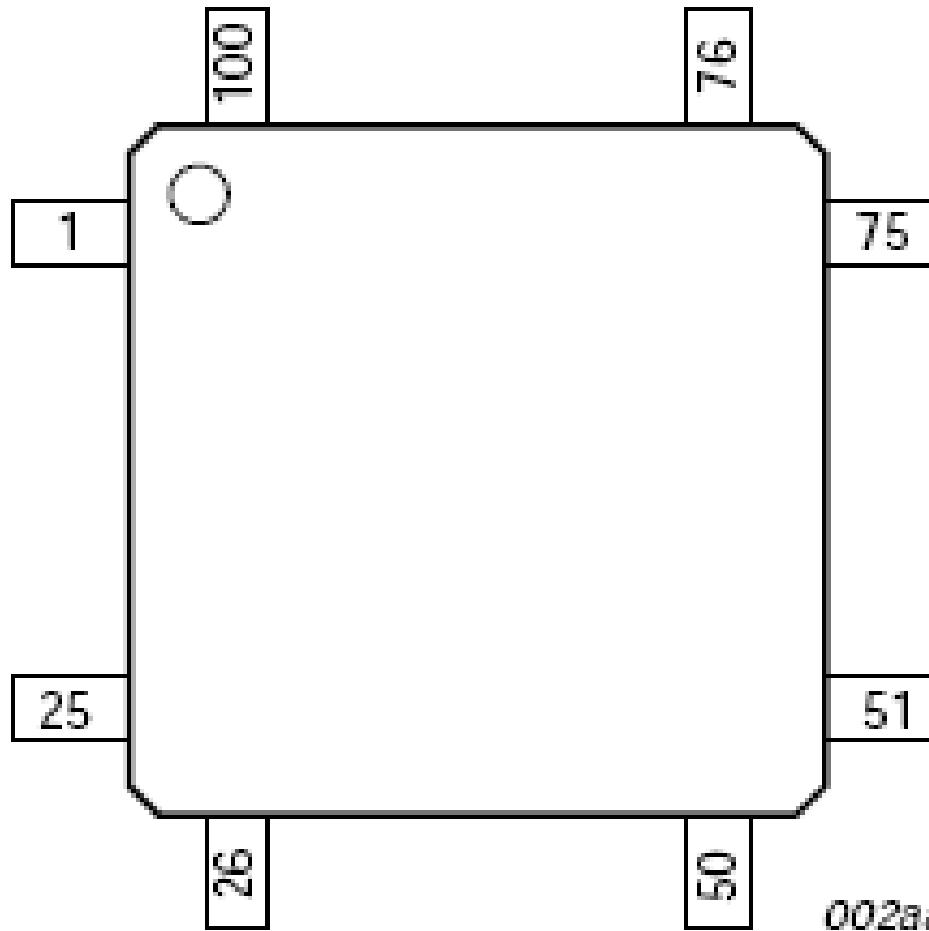


# Embedded System and Design

---

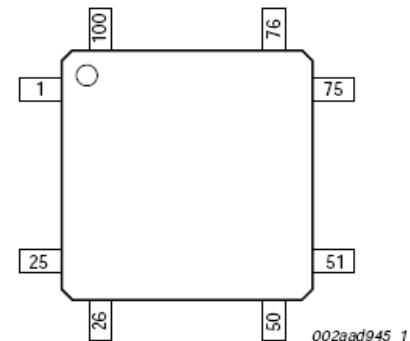
**DR MANOJ TOLANI (PHD-IIIT ALLAHABAD)**  
**ASSISTANT PROFESSOR (DEPARTMENT OF ICT)**

# LPC 1768



002aad945\_1

# LPC 1768 PIN CONFIGURATION



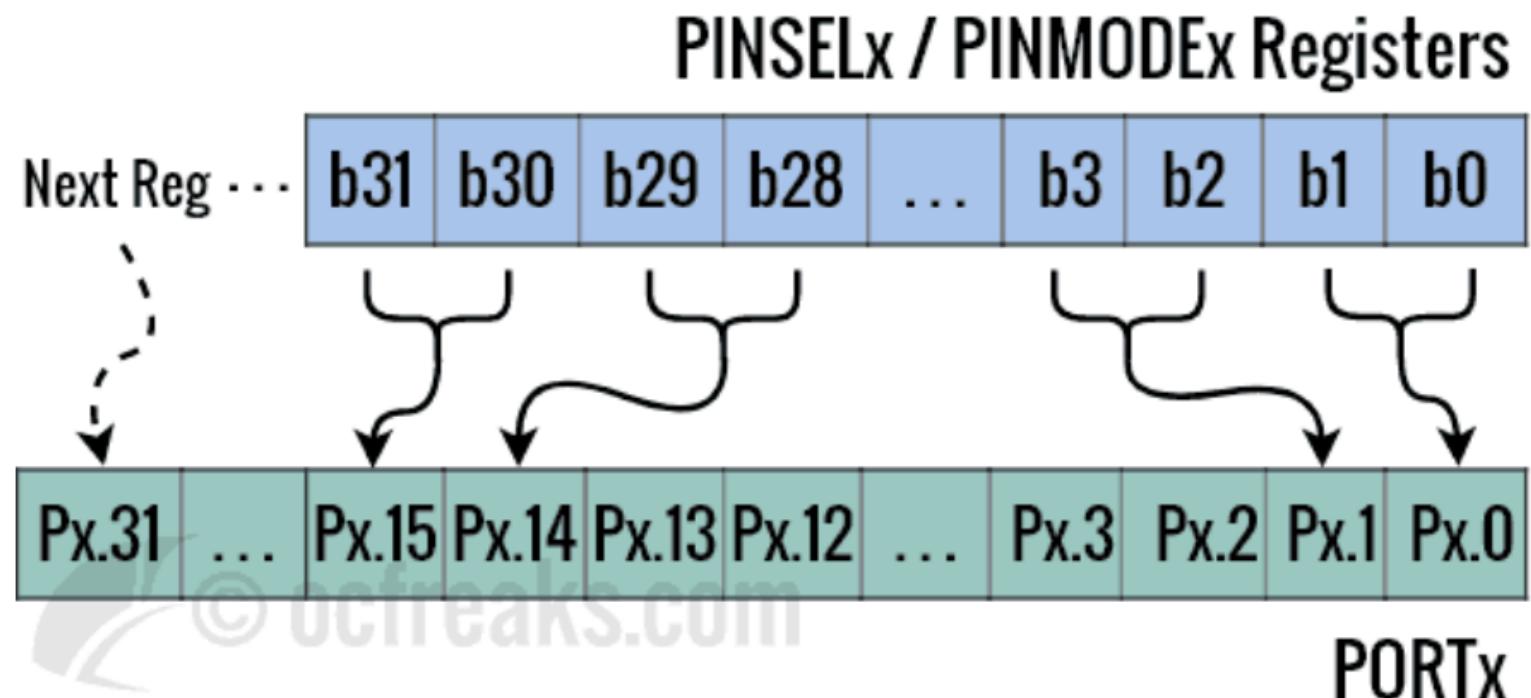
## PX.Y – Port X Pin Y

P0[0] to P0[31]	I/O	<b>Port 0:</b> Port 0 is a 32-bit I/O port with individual direction controls for each bit. The operation of port 0 pins depends upon the pin function selected via the pin connect block. Pins 12, 13, 14, and 31 of this port are not available.
P1[0] to P1[31]	I/O	<b>Port 1:</b> Port 1 is a 32-bit I/O port with individual direction controls for each bit. The operation of port 1 pins depends upon the pin function selected via the pin connect block. Pins 2, 3, 5, 6, 7, 11, 12, and 13 of this port are not available.
P2[0] to P2[31]	I/O	<b>Port 2:</b> Port 2 is a 32-bit I/O port with individual direction controls for each bit. The operation of port 2 pins depends upon the pin function selected via the pin connect block. Pins 14 through 31 of this port are not available.
P3[0] to P3[31]	I/O	<b>Port 3:</b> Port 3 is a 32-bit I/O port with individual direction controls for each bit. The operation of port 3 pins depends upon the pin function selected via the pin connect block. Pins 0 through 24, and 27 through 31 of this port are not available.
P4[0] to P4[31]	I/O	<b>Port 4:</b> Port 4 is a 32-bit I/O port with individual direction controls for each bit. The operation of port 4 pins depends upon the pin function selected via the pin connect block. Pins 0 through 27, 30, and 31 of this port are not available.

- In Port 0 Pins 12, 13, 14 & 31 are not available.
- In Port 1 Pins 2, 3, 7, 6, 5, 11, 12, & 13 are not available.
- In Port 2 only pins 0 to 13 are available and rest are reserved.
- In Port 3 only pins 25,26 are available and rest are reserved.
- Finally in Port 4 only 28,29 are available and rest are reserved.

# PIN CONNECT BLOCK

Most of the PINS of LPC176x MCU are **Multiplexed** i.e. these pins can be configured to provide up to 4 different **functions**. By default, after Power-On or Reset : all pins of all ports are set as GPIO so we can directly use them when learning GPIO usage. The different functions that any particular pin provides can be selected by setting appropriate value in the PINSEL register for the corresponding pin. Each pin on any port has 2 corresponding bits in PINSEL register. The first 16 pins (0-15) on a given port will have a corresponding 32 bit PINSEL register and the rest 16 bits will have another register. For example bits 0 & 1 in PINSEL0 are used to select function for Pin 1 of Port 0, bits 2 & 3 in PINSEL0 are used to select function for PIN 2 of port 0 and so on. The same is applicable for PINMODE register which will go through in last section of this article. Have a look at the diagram given below.



# PIN CONNECT BLOCK

---

## PINSEL0 to Function

### PINSEL9 Values

00	Primary (default) function, typically GPIO port
01	First alternate function
10	Second alternate function
11	Third alternate function

Register	Controls
PINSEL0	P0[15:0]
PINSEL1	P0 [31:16]
PINSEL2	P1 [15:0] (Ethernet)
PINSEL3	P1 [31:16]
PINSEL4	P2 [15:0]
PINSEL5	P2 [31:16]
PINSEL6	P3 [15:0]
PINSEL7	P3 [31:16]
PINSEL8	P4 [15:0]
PINSEL9	P4 [31:16]



PINSEL0

P0.15

## PIN CONNECT BLOCK

P0.1 P0.0

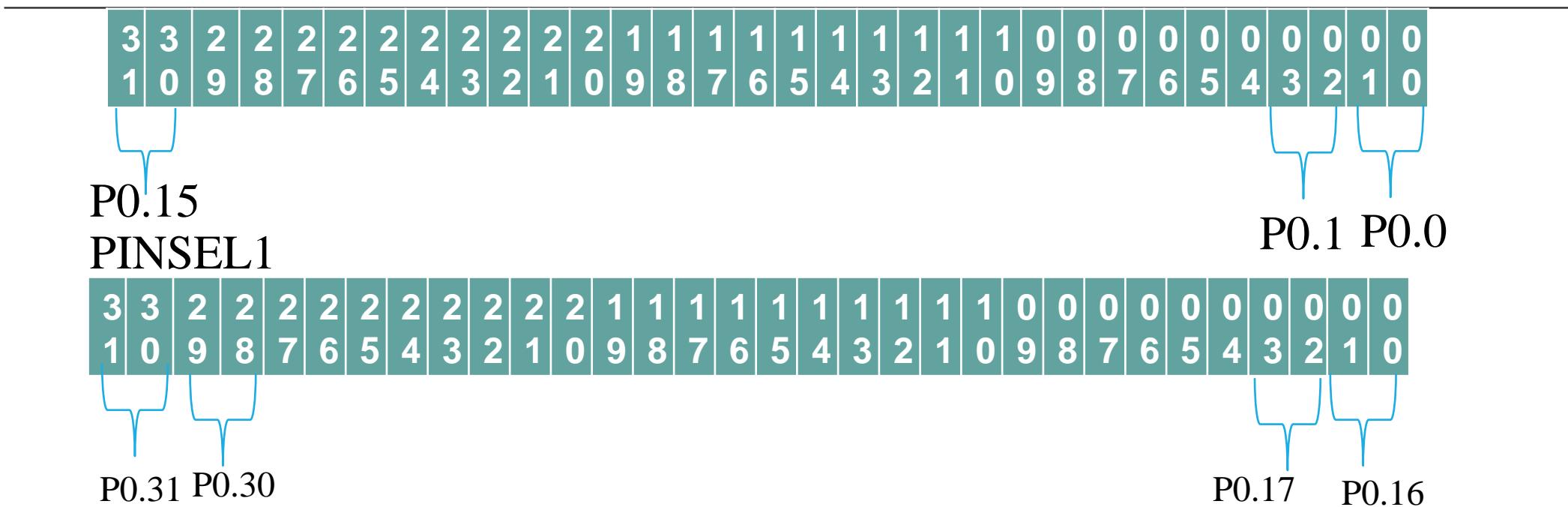
PINSEL0	Pin name	Function when 00	Function when 01	Function when 10	Function when 11
1:0	P0.0	GPIO Port 0.0	RD1	TXD3	SDA1
3:2	P0.1	GPIO Port 0.1	TD1	RXD3	SCL1
5:4	P0.2	GPIO Port 0.2	TXD0	AD0.7	Reserved
7:6	P0.3	GPIO Port 0.3	RXD0	AD0.6	Reserved
9:8	P0.4 <sup>[1]</sup>	GPIO Port 0.4	I2SRX_CLK	RD2	CAP2.0
11:10	P0.5 <sup>[1]</sup>	GPIO Port 0.5	I2SRX_WS	TD2	CAP2.1
13:12	P0.6	GPIO Port 0.6	I2SRX_SDA	SSEL1	MAT2.0
15:14	P0.7	GPIO Port 0.7	I2STX_CLK	SCK1	MAT2.1
17:16	P0.8	GPIO Port 0.8	I2STX_WS	MISO1	MAT2.2
19:18	P0.9	GPIO Port 0.9	I2STX_SDA	MOSI1	MAT2.3
21:20	P0.10	GPIO Port 0.10	TXD2	SDA2	MAT3.0
23:22	P0.11	GPIO Port 0.11	RXD2	SCL2	MAT3.1
29:24	-	Reserved	Reserved	Reserved	Reserved
31:30	P0.15	GPIO Port 0.15	TXD1	SCK0	SCK

# PIN CONNECT BLOCK

<b>PINSEL1</b>	<b>Pin name</b>	<b>Function when 00</b>	<b>Function when 01</b>	<b>Function when 10</b>	<b>Function when 11</b>
1:0	P0.16	GPIO Port 0.16	RXD1	SSEL0	SSEL
3:2	P0.17	GPIO Port 0.17	CTS1	MISO0	MISO
5:4	P0.18	GPIO Port 0.18	DCD1	MOSI0	MOSI
7:6	P0.19 <sup>[1]</sup>	GPIO Port 0.19	DSR1	Reserved	SDA1
9:8	P0.20 <sup>[1]</sup>	GPIO Port 0.20	DTR1	Reserved	SCL1
11:10	P0.21 <sup>[1]</sup>	GPIO Port 0.21	RI1	Reserved	RD1
13:12	P0.22	GPIO Port 0.22	RTS1	Reserved	TD1
15:14	P0.23 <sup>[1]</sup>	GPIO Port 0.23	AD0.0	I2SRX_CLK	CAP3.0
17:16	P0.24 <sup>[1]</sup>	GPIO Port 0.24	AD0.1	I2SRX_WS	CAP3.1
19:18	P0.25	GPIO Port 0.25	AD0.2	I2SRX_SDA	TXD3
21:20	P0.26	GPIO Port 0.26	AD0.3	AOUT	RXD3
23:22	P0.27 <sup>[1][2]</sup>	GPIO Port 0.27	SDA0	USB_SDA	Reserved
25:24	P0.28 <sup>[1][2]</sup>	GPIO Port 0.28	SCL0	USB_SCL	Reserved
27:26	P0.29	GPIO Port 0.29	USB_D+	Reserved	Reserved
29:28	P0.30	GPIO Port 0.30	USB_D-	Reserved	Reserved
31:30	-	Reserved	Reserved	Reserved	Reserved

# PIN CONNECT BLOCK

# PINSEL0



**Configure P0.1 with Function-01 and P0.15 with Function-02, Value to be loaded to PINSEL0:**

**Configure P0.17 with Function-03 and P0.30 with Function-01, Value to be loaded to PINSEL1:**

**Configure P0.1 with Function-01 and P0.15 with Function-02, Value to be loaded to PINSEL0:**

**10000000000000000000000000000000100 = 0x80000004 = (2<<30) | (1<<2)= (2<<(15-0)\*2 | (1<<(1-0)\*2)**

**Configure P0.17 with Function-03 and P0.30 with Function-01, Value to be loaded to PINSEL1:**

**000100000000000000000000000000001100 = 0x1000000C = (1<<28) | (3<<2)**

**= (1<< (30-16)\*2) | (3<<(17-16)\*2)**

# FAST I/O

- The GPIO ports are controlled by five memory mapped registers
  - **FIODIR**: Fast GPIO Port Direction control register
  - **FIOMASK**: Fast Mask register for port
  - **FIOPIN**: Fast Port Pin value register
  - **FIOSET**: Fast Port Output Set register
  - **FIOCLR**: Fast Port Output Clear register

**1) FIODIR** : This is the GPIO direction control register. Setting a bit to 0 in this register will configure the corresponding pin to be used as an Input while setting it to 1 will configure it as Output.

**2) FIOMASK** : This gives masking mechanism for any pin i.e. it is used for Pin access control. Setting a bit to 0 means that the corresponding pin will be affected by changes to other registers like FIOPIN, FIOSET, FIOCLR. Writing a 1 means that the corresponding pin won't be affected by other registers.

**3) FIOPIN** : This register can be used to Read or Write values directly to the pins. Regardless of the direction set for the particular pins it gives the current state of the GPIO pin when read.

**4) FIOSET** : It is used to drive an 'output' configured pin to Logic 1 i.e HIGH. Writing Zero does NOT have any effect and hence it cannot be used to drive a pin to Logic 0 i.e LOW. For driving pins LOW FIOCLR is used which is explained below.

**5) FIOCLR** : It is used to drive an 'output' configured pin to Logic 0 i.e LOW. Writing Zero does NOT have any effect and hence it cannot be used to drive a pin to Logic 1.

# GPIO (General Purpose Input/Output)

FIODIR	Fast GPIO Port Direction control register. This register individually controls the direction of each port pin.					
FIOPIN	Fast Port Pin value register using FIOMASK. The current state of digital port pins can be read from this register, regardless of pin direction or alternate function selection (as long as pins are not configured as an input to ADC). The value read is masked by ANDing with inverted FIOMASK. Writing to this register places corresponding values in all bits enabled by zeros in FIOMASK.  <b>Important:</b> if an FIOPIN register is read, its bit(s) masked with 1 in the FIOMASK register will be read as 0 regardless of the physical pin state.	<hr/> <table><tr><td>0</td><td>Controlled pin is input.</td></tr><tr><td>1</td><td>Controlled pin is output.</td></tr></table> <hr/>	0	Controlled pin is input.	1	Controlled pin is output.
0	Controlled pin is input.					
1	Controlled pin is output.					
FIOSET	Fast Port Output Set register using FIOMASK. This register controls the state of output pins. Writing 1s produces highs at the corresponding port pins. Writing 0s has no effect. Reading this register returns the current contents of the port output register. Only bits enabled by 0 in FIOMASK can be altered.	<hr/> <table><tr><td>0</td><td>Controlled pin output is unchanged.</td></tr><tr><td>1</td><td>Controlled pin output is set to HIGH.</td></tr></table> <hr/>	0	Controlled pin output is unchanged.	1	Controlled pin output is set to HIGH.
0	Controlled pin output is unchanged.					
1	Controlled pin output is set to HIGH.					
FIOCLR	Fast Port Output Clear register using FIOMASK. This register controls the state of output pins. Writing 1s produces lows at the corresponding port pins. Writing 0s has no effect. Only bits enabled by 0 in FIOMASK can be altered.	<hr/> <table><tr><td>0</td><td>Controlled pin output is unchanged.</td></tr><tr><td>1</td><td>Controlled pin output is set to LOW.</td></tr></table> <hr/>	0	Controlled pin output is unchanged.	1	Controlled pin output is set to LOW.
0	Controlled pin output is unchanged.					
1	Controlled pin output is set to LOW.					
FIOMASK	Fast Mask register for port. Writes, sets, clears, and reads to port (done via writes to FIOPIN, FIOSET, and FIOCLR, and reads of FIOPIN) alter or return only the bits enabled by zeros in this register.	<table><tr><td>0</td><td>Controlled pin is affected by writes to the port's FIOxSET, FIOxCLR, and FIOxPIN register(s). Current state of the pin can be read from the FIOxPIN register.</td></tr><tr><td>1</td><td>Controlled pin is not affected by writes into the port's FIOxSET, FIOxCLR and FIOxPIN register(s). When the FIOxPIN register is read, this bit will not be updated with the state of the physical pin.</td></tr></table>	0	Controlled pin is affected by writes to the port's FIOxSET, FIOxCLR, and FIOxPIN register(s). Current state of the pin can be read from the FIOxPIN register.	1	Controlled pin is not affected by writes into the port's FIOxSET, FIOxCLR and FIOxPIN register(s). When the FIOxPIN register is read, this bit will not be updated with the state of the physical pin.
0	Controlled pin is affected by writes to the port's FIOxSET, FIOxCLR, and FIOxPIN register(s). Current state of the pin can be read from the FIOxPIN register.					
1	Controlled pin is not affected by writes into the port's FIOxSET, FIOxCLR and FIOxPIN register(s). When the FIOxPIN register is read, this bit will not be updated with the state of the physical pin.					

# GPIO Port Direction Register FIOxDIR

---

- This word accessible register is used to control the direction of the pins
- There are 5 direction control register each of size is 32 bit

Register Name	Port	Access	Reset Value	Address
FIO0DIR	0	Read/Write	0x0000 0000	0x2009 C000
FIO1DIR	1	Read/Write	0x0000 0000	0x2009 C020
FIO2DIR <small>↳</small>	2	Read/Write	0x0000 0000	0x2009 C040
FIO3DIR	3	Read/Write	0x0000 0000	0x2009 C060
FIO4DIR	4	Read/Write	0x0000 0000	0x2009 C080

# GPIO (General Purpose Input/Output)

FIOxDIR

3 3 2 2 2 2 2 2 2 2 2 1 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0

FIOxDIRH

FIOxDIRL

FIOxDIR3

FIOxDIR2

FIOxDIR1

FIOxDIRO

Same concept applicable to FIOxSET, FIOxCLR, FIOxPIN, FIOxMASK (Ex:  
For Port-1 , we can have FIO1SET, FIO1SETH, FIO1SETL, FIO1SET3,  
FIO1SET2, FIO1SET1, FIO1SET0.....

# GPIO (General Purpose Input/Output)

Pin CNA	PIN LPC1768	Description
1	81	P0.4/I2SRX_CLK/RD2/CAP2.0
2	80	P0.5/I2SRX_WS/TD2/CAP2.1
3	79	P0.6/I2SRX_SDA/SSEL1/MAT2.0
4	78	P0.7/I2STX_CLK/SCK1//MAT2.1
5	77	P0.8/I2STX_WS/MISO1/MAT2.2
6	76	P0.9/I2STX_SDA/MOSI1/MAT2.3
7	48	P0.10/TXD2/SDA2/MAT3.0
8	49	P0.11/RXD2/SCL2/MAT3.1
9	-	No connection
10	-	Ground

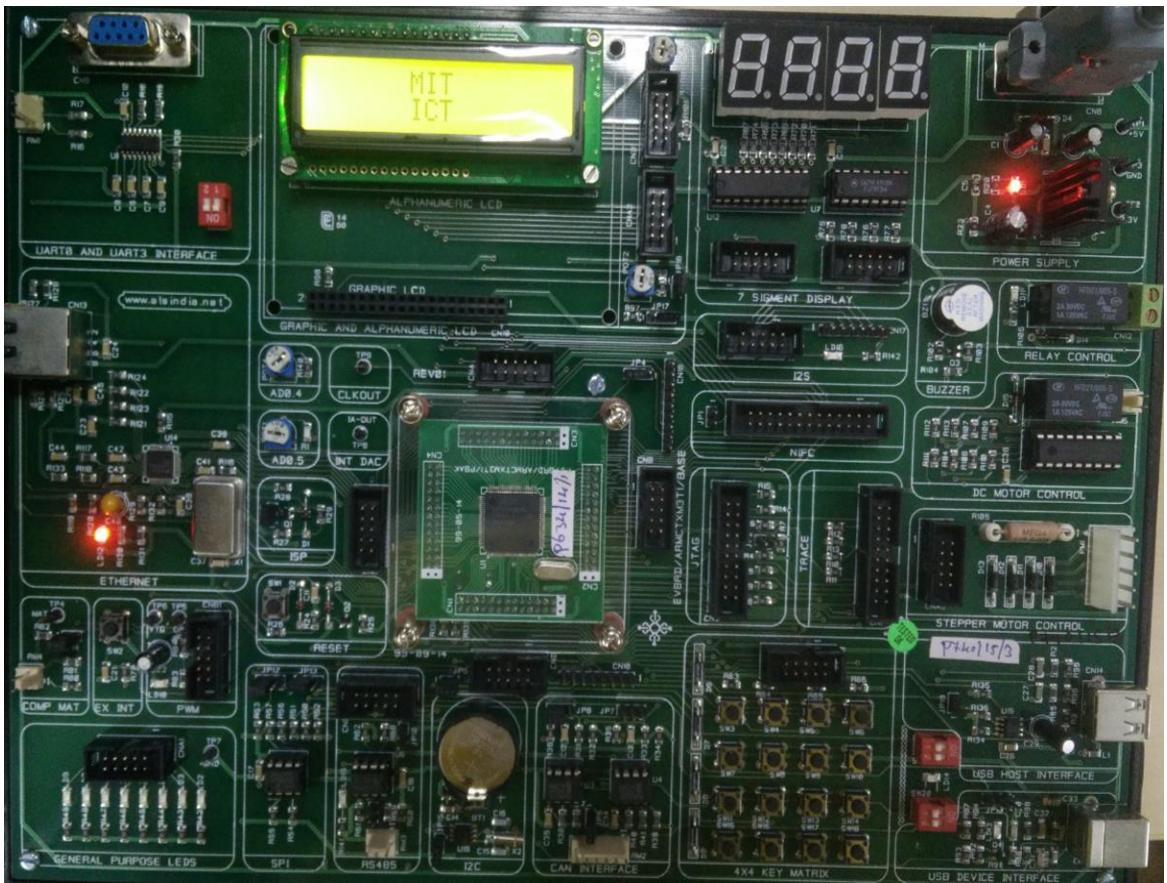
Pin CNB	Pin LPC1768	Description
1	37	P1.23/MCI1/PWM1.4/MISO0
2	38	P1.24/MCI2/PWM1.5/MOSI0
3	39	P1.25/MCOA1/MAT1.1
4	40	P1.26/MCOB1/PWM1.6/CAP0.0
5	53	P2.10/EINT0/NMI
6	52	P2.11/EINT1/I2STX_CLK
7	51	P2.12/EINT2/I2STX_WS
8	50	P2.13/EINT3/I2STX_SDA
9	-	No connection
10	-	Ground

Pin CNC	Pin LPC1768	Description
1	62	P0.15/TXD1/SCK0/SCK
2	63	P0.16/RXD1/SSEL0/SSEL
3	61	P0.17/CTS1/MISO0/MISO
4	60	P0.18/DCD1/MOSI0/MOSI
5	59	P0.19/DSR1/SDA1
6	58	P0.20/DTR1/SCL1
7	57	P0.21/RI1/RD1
8	56	P0.22/RTS1/TD1
9	50	P2.13/I2STX_SDA
10	-	Ground

Pin CND	Pin LPC1768	Description
1	9	P0.23/AD0.0/I2SRX_CLK/CAP3.0
2	8	P0.24/AD0.1/I2SRX_WS/CAP3.1
3	7	P0.25/AD0.2/I2SRX_SDA/TXD3
4	6	P0.26/AD0.3/AOUT/RXD3
5	25	P0.27/SDA0/USB/SDA
6	24	P0.28/SCL0/USB_SCL
7	75	P2.0/PWM1.1/TXD1
8	74	P2.1/PWM1.2/RXD1
9	-	No connection
10	-	Ground

# GPIO (General Purpose Input/Output)

---





# Embedded System and Design

---

**DR MANOJ TOLANI (PHD-IIIT ALLAHABAD)**  
**ASSISTANT PROFESSOR (DEPARTMENT OF ICT)**

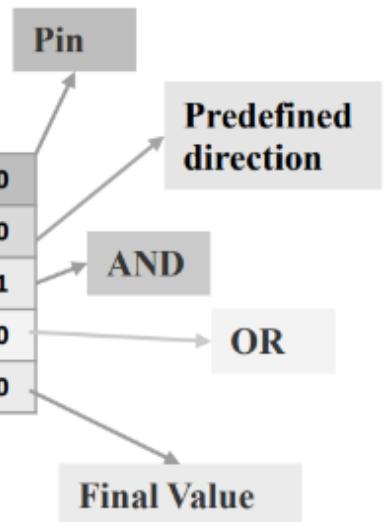
## GPIO Port direction control register

- Let us see an example: We want to set P0.4 (Port 0, Pin4) as input

FIO0DIR Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	1	0	1	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	1	1	1	0	1	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	1	1	1	0	1	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	1	1	0	0	1	0	0

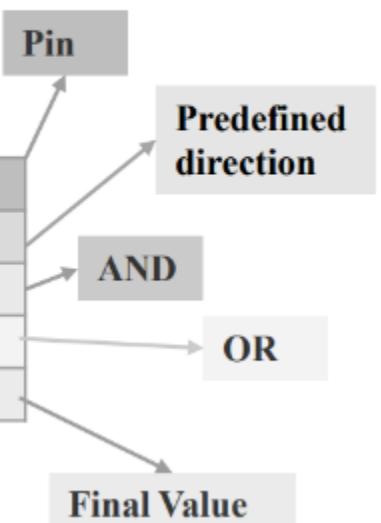
**FIO0DIR &= FFFF FFEF (AND)**



## Set P0.5 as Output

FIO0DIR Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	1	0	1	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	1	0	1	0	1	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	1	1	1	0	1	0	0	0	0	0	1	0	0	0	1	0	0	1	1	1	0	1	0	1	0	0	0



**FIO0DIR |= 0000 0020 (OR)**

## **GPIO Programming & Examples**

Lets say we want to set PIN 3 on Port 0 as output. It can be done in following ways:

## GPIO Programming & Examples

Lets say we want to set PIN 3 on Port 0 as output. It can be done in following ways:

```
CASE 1. LPC_GPIO0->FIODIR = (1<<3); // (binary using left shift - direct assign:  
other pins set to 0)
```

```
CASE 2. LPC_GPIO0->FIODIR |= 0x00000008; // or 0x8; (hexadecimal - OR and assign:  
other pins not affected)
```

```
CASE 3. LPC_GPIO0->FIODIR |= (1<<3); // (binary using left shift - OR and assign:  
other pins not affected)
```

### Example #1)

Consider that we want to configure Pin 4 of Port 0 i.e P0.4 as Output and want to drive it High(Logic 1). This can be done as :

## Example #1)

Consider that we want to configure Pin 4 of Port 0 i.e P0.4 as Output and want to drive it High(Logic 1). This can be done as :

```
LPC_GPIO0->FIODIR |= (1<<4); // Config P0.4 as Output  
LPC_GPIO0->FIOSET |= (1<<4); // Make output High for P0.4
```

Example #2)

Making output configured Pin 17 High of Port 0 i.e P0.17 and then Low can be done as follows:

## Example #2)

Making output configured Pin 17 High of Port 0 i.e P0.17 and then Low can be done as follows:

```
LPC_GPIO0->FIODIR |= (1<<17); // P0.17 is Output pin  
LPC_GPIO0->FIOSET |= (1<<17); // Output for P0.17 becomes High  
LPC_GPIO0->FIOCLR |= (1<<17); // Output for P0.17 becomes Low
```

Example #3)

Configuring P0.5 and P0.11 as Ouput and Setting them High:

### Example #3)

Configuring P0.5 and P0.11 as Ouput and Setting them High:

```
LPC_GPIO0->FIODIR |= (1<<5) | (1<<11); // Config P0.5 and P0.11 as Ouput  
LPC_GPIO0->FIOSET |= (1<<5) | (1<<11); // Make ouput High for P0.5 and P0.11
```

Example #4)

Configuring 1st 8 Pins of Port 0 (P0.0 to P0.7) as Output and Setting them High:

#### Example #4)

Configuring 1st 8 Pins of Port 0 (P0.0 to P0.7) as Ouput and Setting them High:

```
LPC_GPIO0->FIODIR |= 0xFF; // Config P0.0 to P0.7 as Ouput  
LPC_GPIO0->FIOSET |= 0xFF; // Make output High for P0.0 to P0.7
```

## GPIO port FIOxPIN

- Example :
  - Produce **HIGH** logic on pin 0, 1, 2, 3 of port 0
  - Produce **LOW** logic on pin 4, 5, 6, 7 of port 0

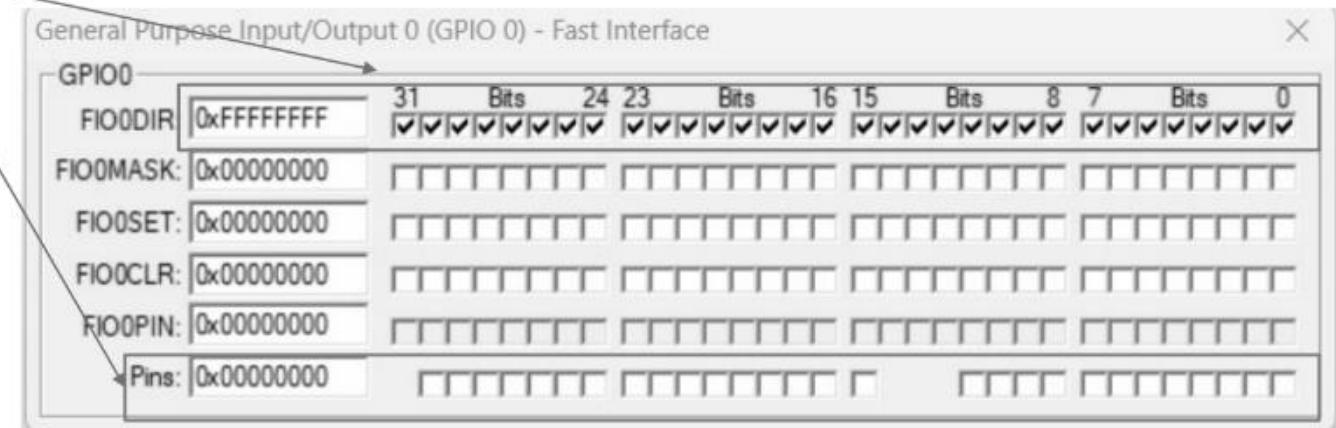
FIOOPIN																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	0	0	0	0	1	1	1	1	1	1

- OR operation for logic level **HIGH**
- AND operation for logic level **LOW**
- $\text{FIOOPIN} |= 0x0000000F;$
- $\text{FIOOPIN} \&= 0xFFFFF0F;$

## Analysis of Mask Operation if Port pins are configured as Output (No Predefined Pin Output Values)

```
#include <LPC17xx.h>
int main(void)
{
    LPC_GPIO0->FIODIR= 0xFFFFFFFF; //Configured Output Pins
```

```
LPC_GPIO0->FIOMASK = 0x0000FFFF;  
LPC_GPIO0->FIOPIN = 0x2BCDEF12;
```



```
#include <LPC17xx.h>
int main(void)
{
    LPC_GPIO0->FIODIR= 0xFFFFFFFF; //Configured Output Pins

    LPC_GPIO0->FIOMASK = 0x0000FFFF;
    LPC_GPIO0->FIOPIN = 0x2BCDEF12;
}
```

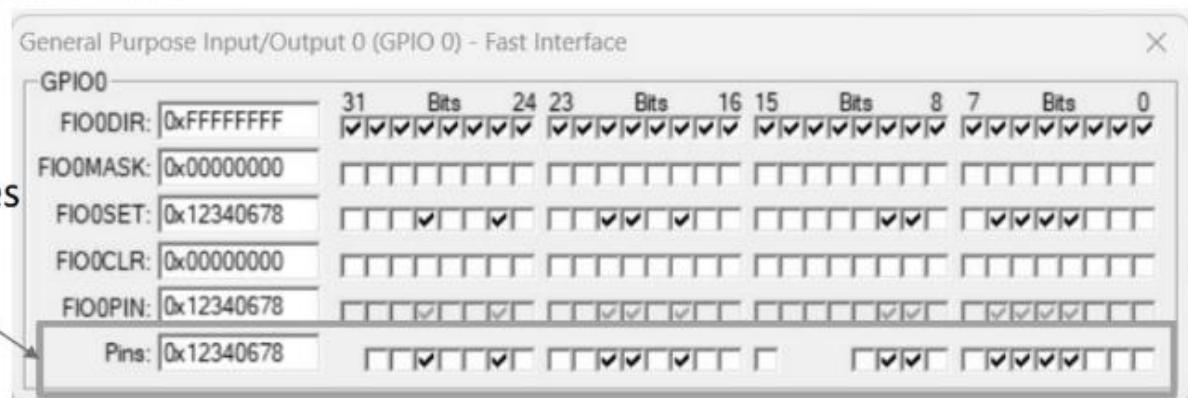


## Analysis of Mask Operation if Port pins are configured as Output (With Predefined Pin Output Values)

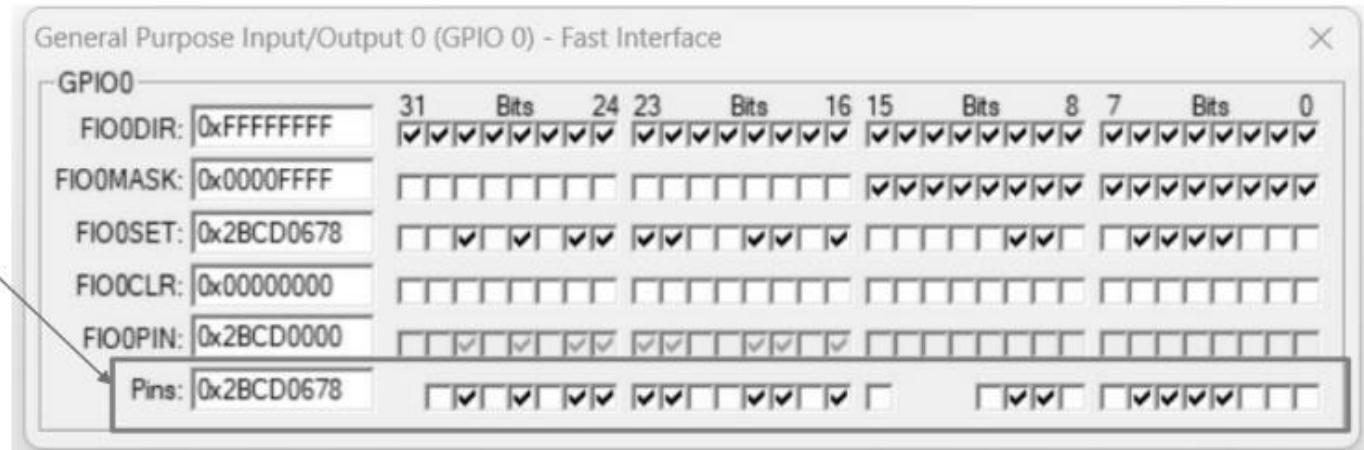
```
#include <LPC17xx.h>
int main(void)
{
    LPC_GPIO0->FIODIR= 0xFFFFFFFF; //Configured Output Pins

    LPC_GPIO0->FIOPIN = 0x12340678;
    LPC_GPIO0->FIOMASK = 0x0000FFFF;
    LPC_GPIO0->FIOPIN = 0x2BCDEF12;
}
```

Initial Pin Values



After Execution Pin Values



# GPIO (General Purpose Input/Output)

**Ex: Send 0xA5 to P0.15-P0.8 without affecting values on the remaining pins.  
This can be accomplished in several ways**

---

```
FIO0MASK = 0xFFFF00FF ;  
FIO0PIN  = 0x0000A500;
```

**Solution 2:** using 16-bit (half-word) accessible fast GPIO registers

```
FIO0MASKL = 0x00FF;  
FIO0PINL  = 0xA500;
```

**Solution 3:** using 8-bit (byte) accessible fast GPIO registers

```
FIO0PIN1  = 0xA5;
```

**Configure P0.1 with Function-01 and P0.15 with Function-02, Value to be loaded to PINSEL**

**Answer: PINSEL0=1<<2 | 2<<30**

**Configure P0.17 with Function-03 and P0.30 with Function-01, Value to be loaded to PINSEL**

**Answer: PINSEL1=3<<2 | 1<<28**

**Configure P2.3 with Function-01 and P2.14 with Function-02, Value to be loaded to PINSEL**

**Answer: PINSEL4=1<<6 | 2<<28**

**Configure P2.17 with Function-03 and P2.5 with Function-01, Value to be loaded to PINSEL**

**Answer:**  
**PINSEL5=3<<2**  
**PINSEL4=1<<10**

**Send 0xA5 to P0.15-P0.8 without affecting values on the remaining pins. Use FIOMASK and FIOPIN**

**Answer:**  
**FIOMASK=0xFFFF00FF;**  
**FIOPIN=0x0000A500;**

**Initial pins instruction 0x78560743. Send 0xA5 to P0.23-P0.16 without affecting values on the remaining pins (Instructions). Use FIOMASK and FIOPIN and find the value of the pins instruction after execution**

**Answer:**  
**FIOMASK=0xFF00FFFF;**  
**FIOPIN=0x00A50000;**

**Pin o/p after instruction 0x78A50743**

LPC\_GPIO0->FIOPIN |= 3F<<20



LPC\_GPIO0->FIOPIN |= 2E<<4



Write an embedded C program to turn ON and OFF LEDs connected to P0.11 – P0.4

```
#include <LPC17xx.h>
```

# Variable Initialization

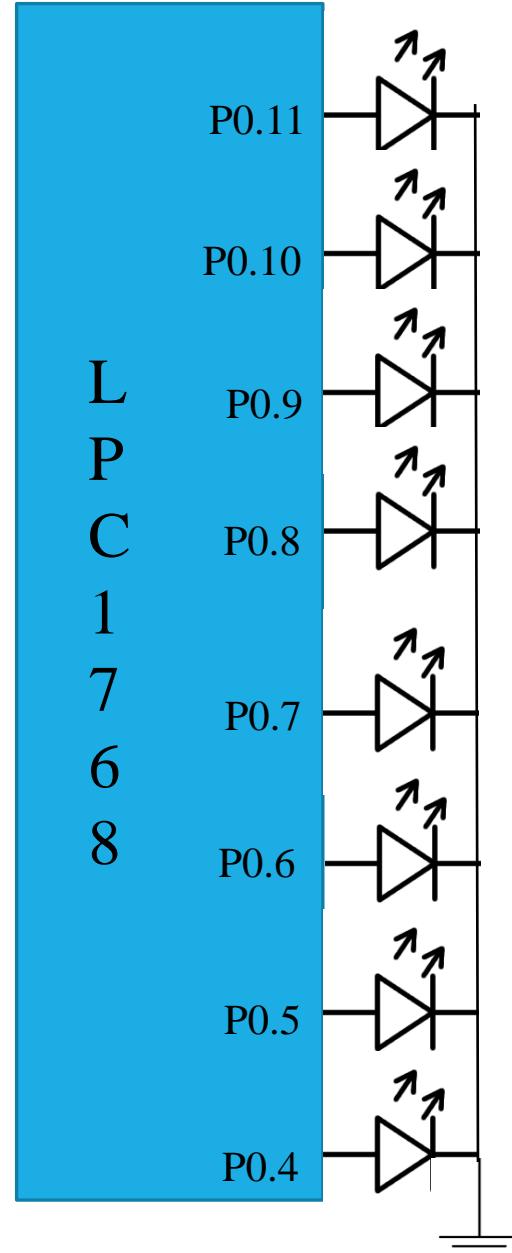
```
int main(void)  
{
```

## i/o Pin selection and Pin Direction

```
while(1)  
{
```

# Pin Operation

}



# GPIO (General Purpose Input/Output)

Write an embedded C program to turn ON and OFF LEDs connected to P0.11 – P0.4

```
#include <LPC17xx.h>
unsigned int j;
unsigned long LED = 0x00000FF0;

int main(void)
{
    SystemInit();
    SystemCoreClockUpdate();

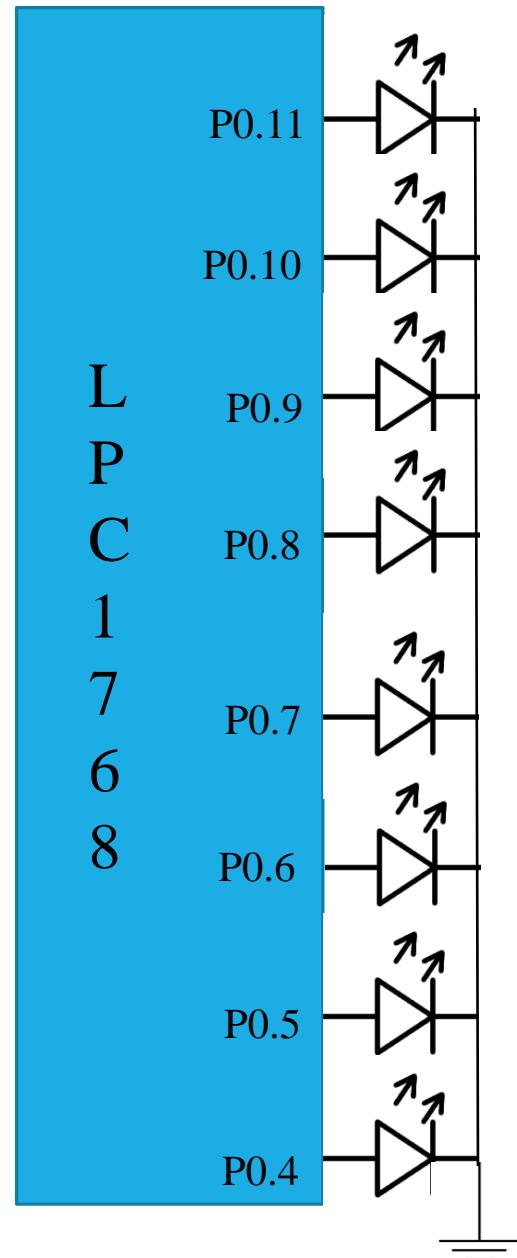
    LPC_PINCON->PINSEL0 = 0x00000000; // P0.15-P0.0 GPIO
    LPC_GPIO0->FIODIR = 0x00000FF0; // P0.11-P0.4 as output

    while(1)
    {
        [LPC_GPIO0->FIOSET = LED; // SET P0.11-P0.4
        for(j=0;j<10000;j++); // Delay

        [LPC_GPIO0->FIOCLR = LED; // CLEAR P0.11-P0.4
        for(j=0;j<10000;j++); //Delay

    }
}

LPC_GPIO0->FIOPIN= ~(LPC_GPIO0->FIOPIN & 0x00000FF0);
for(j=0;j<10000;j++); //Delay
```



# GPIO (General Purpose Input/Output)

Write an embedded C program to turn ON and OFF LEDs connected to P0.11 – P0.4

```
#include <LPC17xx.h>
unsigned int j;
unsigned int LED = 0x0FF0;

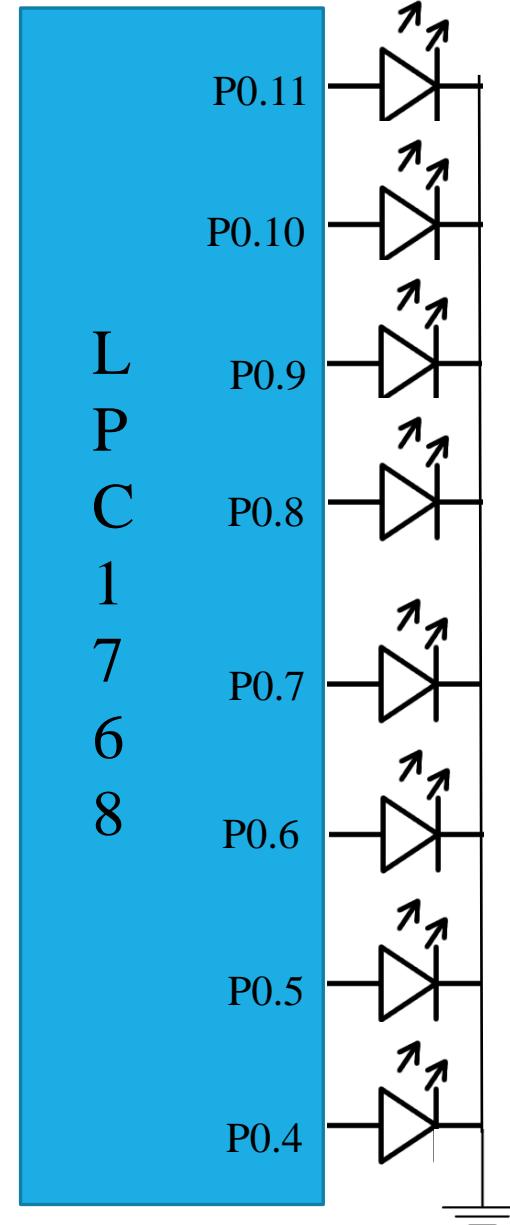
int main(void)
{
    SystemInit();
    SystemCoreClockUpdate();

    LPC_PINCON->PINSEL0 = 0x00000000; // P0.15-P0.0 GPIO
    LPC_GPIO0->FIODIRL = 0x0FF0; // P0.11-P0.4 as output

    while(1)
    {
        LPC_GPIO0->FIOSETL = LED; // SET P0.11-P0.4
        for(j=0;j<10000;j++); // Delay

        LPC_GPIO0->FIOLRL = LED; // CLEAR P0.11-P0.4
        for(j=0;j<10000;j++); // Delay
    }

    LPC_GPIO0->FIOPINL= ~(LPC_GPIO0->FIOPINL & 0x0FF0);
    for(j=0;j<10000;j++); //Delay
}
```



LED = 0x00000010

0000 0000 0000 0000 0000 0000 0001 0000

LED<<=1

OR

LED=LED<<1

After Execution

LED = 0x00000020

0000 0000 0000 0000 0000 0000 0010 0000

# GPIO (General Purpose Input/Output)

---

8 bit Johnson Counter on LEDs

```
#include <LPC17xx.h>                                00000000
unsigned int i,j;                                     00000001
unsigned long LED = 0x00000010;                        00000011
                                                       00000111
                                                       00001111
                                                       00011111
                                                       00111111
                                                       01111111
                                                       11111111
                                                       11111110
                                                       11111100
                                                       11111000
                                                       11110000
                                                       11100000
                                                       11000000
                                                       10000000
                                                       00000000
int main(void)
{
    SystemInit()
    SystemCoreClockUpdate();

    LPC_PINCON->PINSEL0 = 0
        ;Configure Port0 pins P0.4-P0.11 ;as GPIO
    LPC_GPIO0->FIODIR = 0x00000FF0;
        ;Configure P0.4-P0.11 as output
```

# GPIO (General Purpose Input/Output)

---

```
while(1)
{
    LED = 0x00000010; Initial value on LED
    for(i=1;i<9;i++)          //ON the LED's serially
    {
        LPC_GPIO0->FIOSET = LED;

        for(j=0;j<10000;j++);
        LED <<= 1;
    }

    LED = 0x00000010;

    for(i=1;i<9;i++)          //OFF the LED's serially
    {
        LPC_GPIO0->FIOCLR = LED
        for(j=0;j<10000;j++);
        LED <<= 1;
    }

}
```



# Embedded System and Design

---

**DR MANOJ TOLANI (PHD-IIIT ALLAHABAD)**  
**ASSISTANT PROFESSOR (DEPARTMENT OF ICT)**

# GPIO (General Purpose Input/Output)

Write an embedded C program to turn ON LEDs connected to P0.11 – P0.4 when key connected to P2.12 pressed, else turn OFF.

```
#include <LPC17xx.h>
unsigned int j;
unsigned long LED = 0x00000FF0;

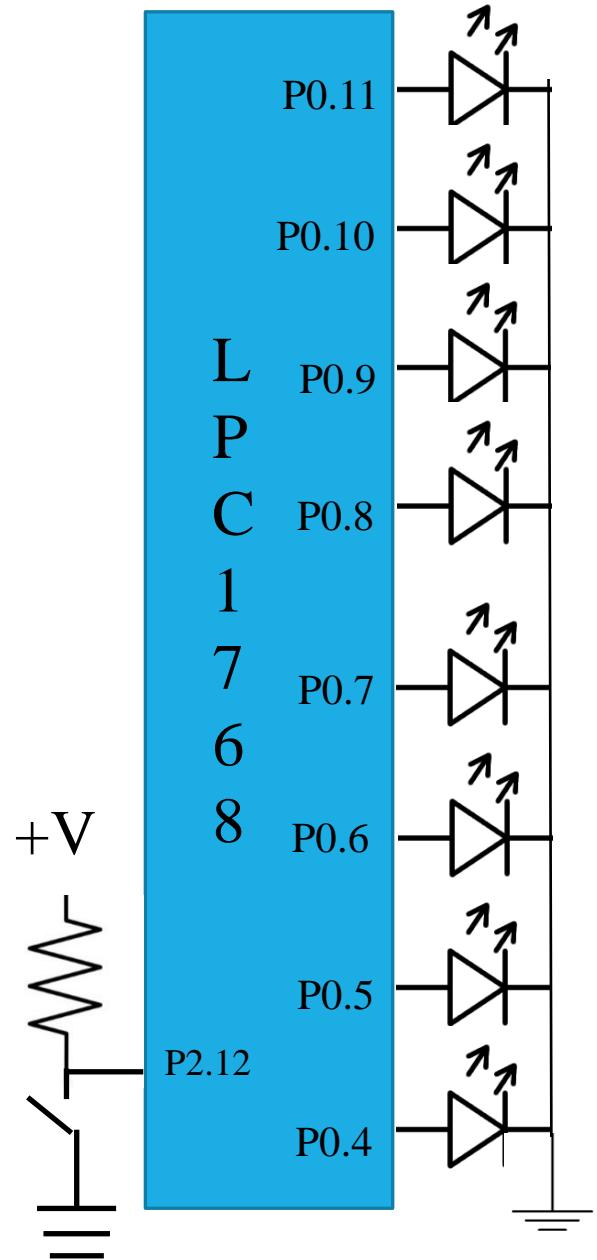
int main(void)
{
    SystemInit();
    SystemCoreClockUpdate();

    LPC_PINCON->PINSEL0 = 0x00000000; // P0.15-P0.0 GPIO
    LPC_GPIO0->FIODIR = 0x0000FF0; // P0.11-P0.4 as output

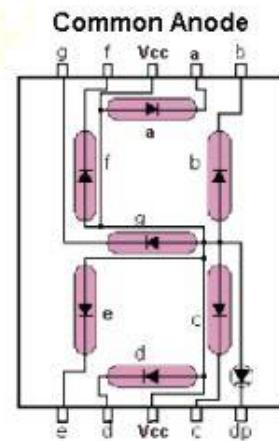
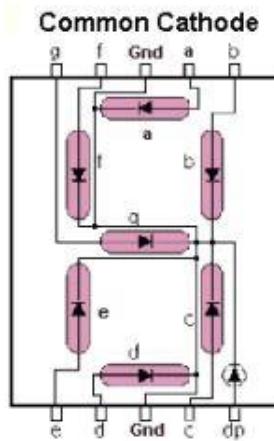
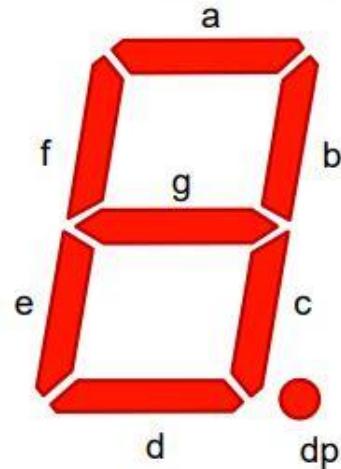
    while(1)
    {
        if ( !(LPC_GPIO2->FIOPIN & 1<<12))

            LPC_GPIO0->FIOSET = LED; // SET P0.11-P0.4
        else

            LPC_GPIO0->FIOCLR = LED; // CLEAR P0.11-P0.4
    }
}
```



- Common Cathode (all LED cathodes are connected)
  - Common Anode (all LED anodes are connected)



# Seven Segment Display

Decimal Digits 0-9

0 1 2 3 4 5 6 7 8 9

Select Alpha Characters

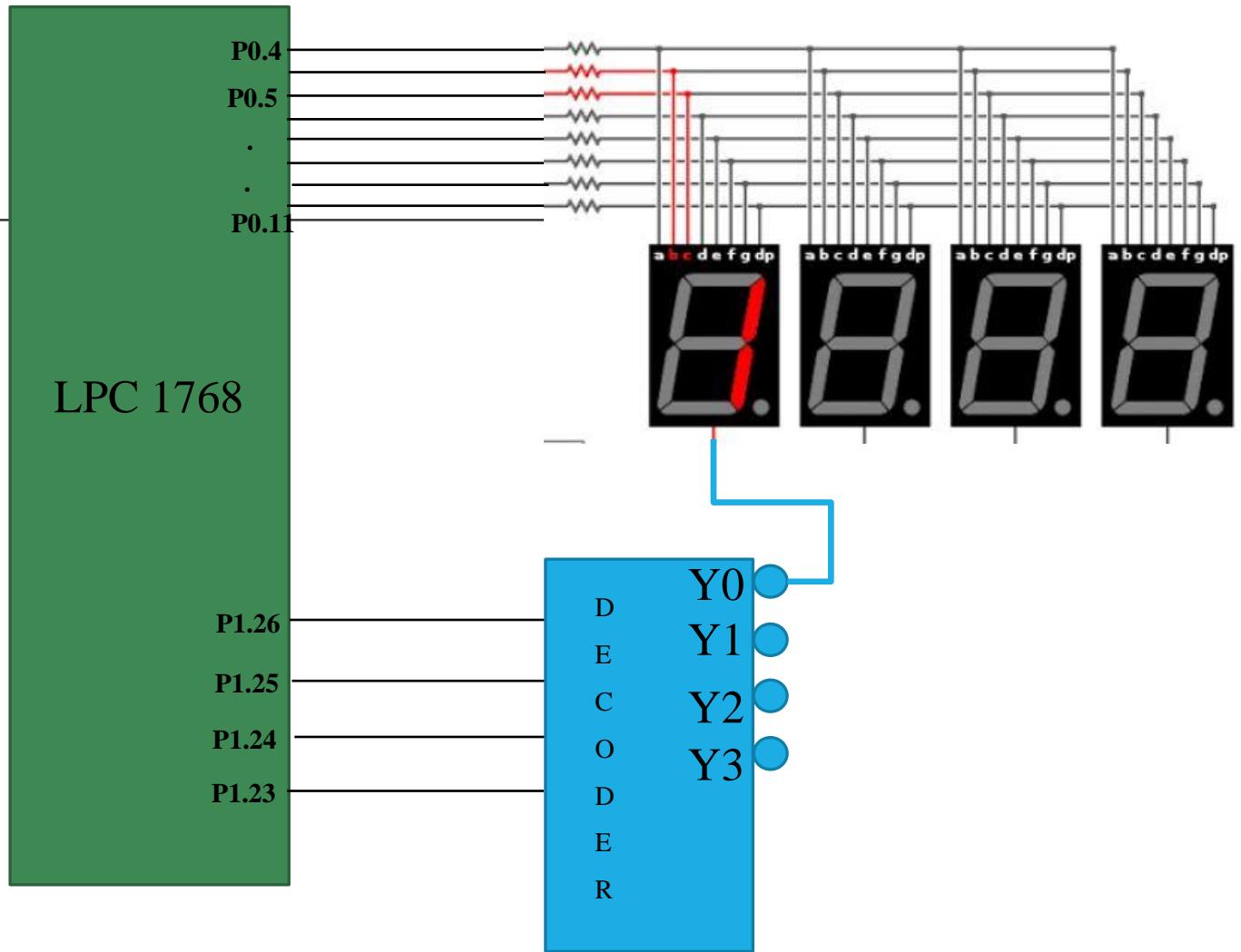
A B C D E F H I J  
L N O P Q S U V Z

# Seven Segment Display

---

<b>Number</b>	<b>h g f e d c b a</b>	<b>Hex Code</b>
0	0 0111111	3F
1	0 0000110	06
2	1011011	5B
3	1001111	4F
4	1100110	66
5	1101101	6D
6	1111101	7D
7	0000111	07
8	1111111	7F
9	1001111	4F

# Multiplexed Seven Segment Display



# Seven Segment Display Interfacing

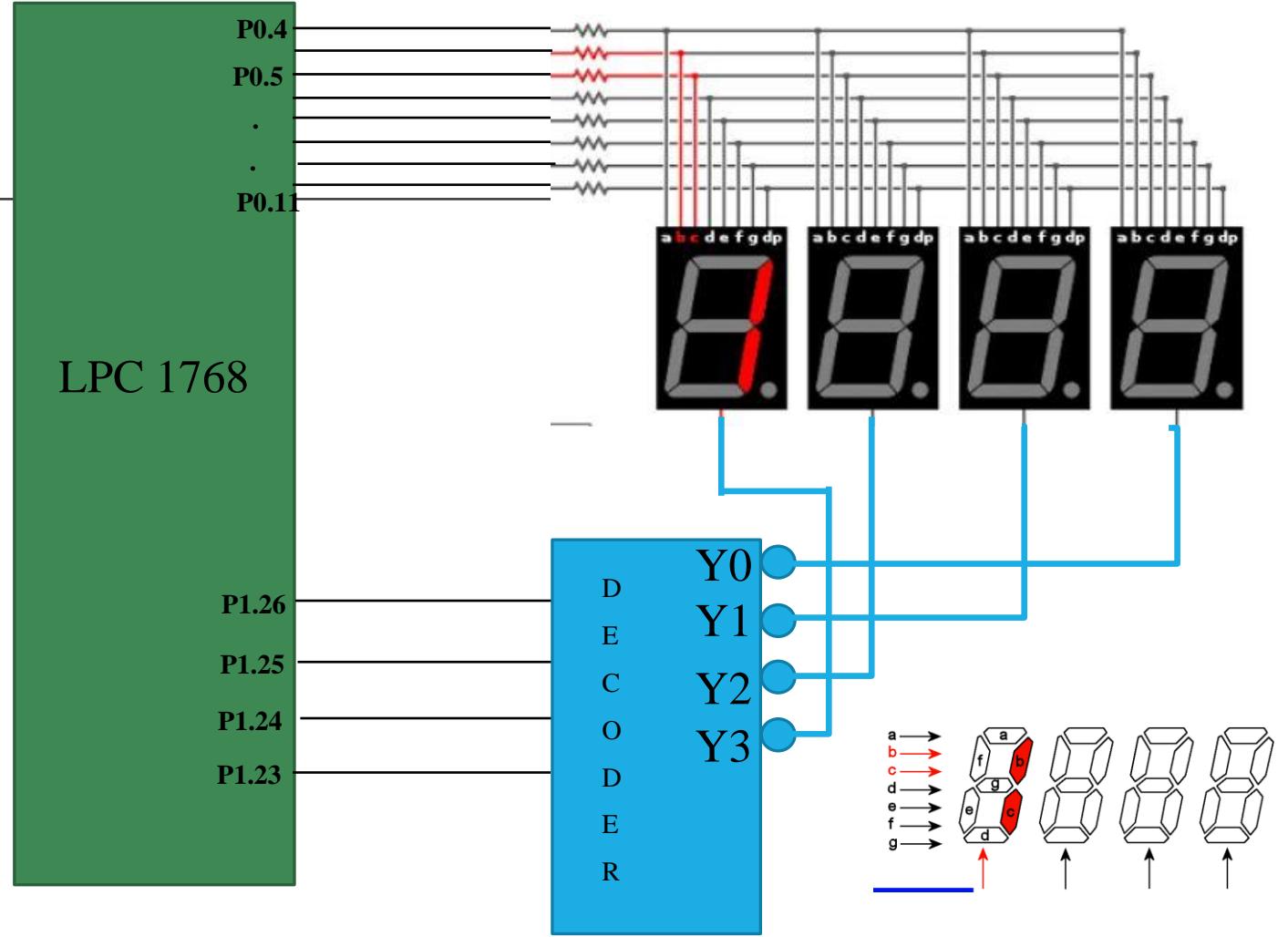
```
#include<lpc17xx.h>
unsigned char seven_seg[10]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
unsigned int i,j;
void delay(void);
int main(void)
{
    SystemInit();
    SystemCoreClockUpdate();

    LPC_PINCON->PINSEL0 = 0          ; //P0.4 to P0.11 GPIO data lines
    LPC_GPIO0->FIODIR = 0x00000FF0;    //P0.4 to P0.11 output
    LPC_GPIO1->FIODIR = 0x07800000; //P1.23 to P1.26 output

    while(1)
    {
        for(i=0; i<10; i++)
        {
            LPC_GPIO1->FIOPIN = 0 << 23;
            LPC_GPIO0->FIOPIN = seven_seg[i] << 4;

            delay();
        }
    }
}
void delay(void)
{
    for(j=0;j<10000000;j++);
}
```

# Multiplexed Seven Segment Display

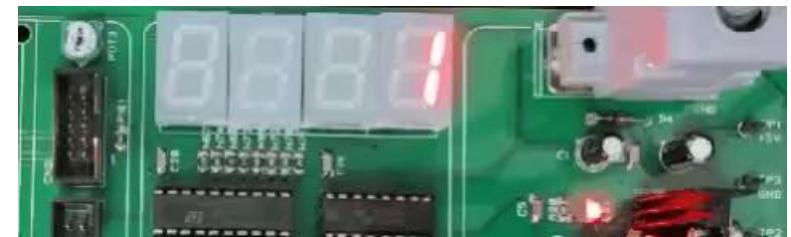


```

#include<lpc17xx.h>
unsigned char seven_seg[10]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
unsigned char new_seg[4]={0,1,2,3};
unsigned int i,j;
void delay(void);
int main(void)
{
    SystemInit();
    SystemCoreClockUpdate();

    LPC_PINCON->PINSEL0 = 0      ; //P0.4 to P0.11 GPIO data lines
    LPC_GPIO0->FIODIR = 0x00000FF0;          //P0.4 to P0.11 output
    LPC_GPIO1->FIODIR = 0x07800000; //P1.23 to P1.26 output
    while(1)
    {
        for(i=0; i<4; i++)
        {
            LPC_GPIO1->FIOPIN = new_seg[i] << 23;
            LPC_GPIO0->FIOPIN = seven_seg[i+1] << 4;
            delay();
        }
    }
}
void delay(void)
{
    for(j=0;j<10000000;j++);
}

```



# 4-digit BCD UP-Counter

```
#include <LPC17xx.h>
#include <stdio.h>

// Define Constants for 7-Segment Display Digit Selection
#define FIRST_SEG (0<<23)
#define SECOND_SEG (1<<23)
#define THIRD_SEG (2<<23)
#define FOURTH_SEG (3<<23)

unsigned int dig_count = 0; // Current digit count
unsigned int digit_value[4] = {0, 0, 0, 0}; // Stores the four-digit counter values
unsigned int select_segment[4] = {FIRST_SEG, SECOND_SEG, THIRD_SEG, FOURTH_SEG};

// Seven Segment Display Mapping
unsigned char seven_seg[10] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F};

unsigned long int i = 0;
unsigned int count = 0, flag = 0;
#define N 2000 // Delay threshold for counting speed
```

```
// Function Prototypes
void Display(void);
void delay(void);

int main(void)
{
    SystemInit();
    SystemCoreClockUpdate();

    // Configure GPIO Pins
    LPC_PINCON->PINSEL0 = 0; // P0.4 to P0.11 as GPIO (Data lines)
    LPC_PINCON->PINSEL3 = 0; // P1.23 to P1.26 as GPIO (Digit selection lines)

    LPC_GPIO0->FIODIR = 0x00000FF0; // P0.4 to P0.11 as output for 7-segment control
    LPC_GPIO1->FIODIR = 0x07800000; // P1.23 to P1.26 as output for digit selection

    while (1)
    {
        delay(); // Control counting speed
        dig_count += 1;

        if (dig_count == 4) // Reset after 4 digits
            dig_count = 0;

        Display(); // Update display
    }

    return 0;
}
```

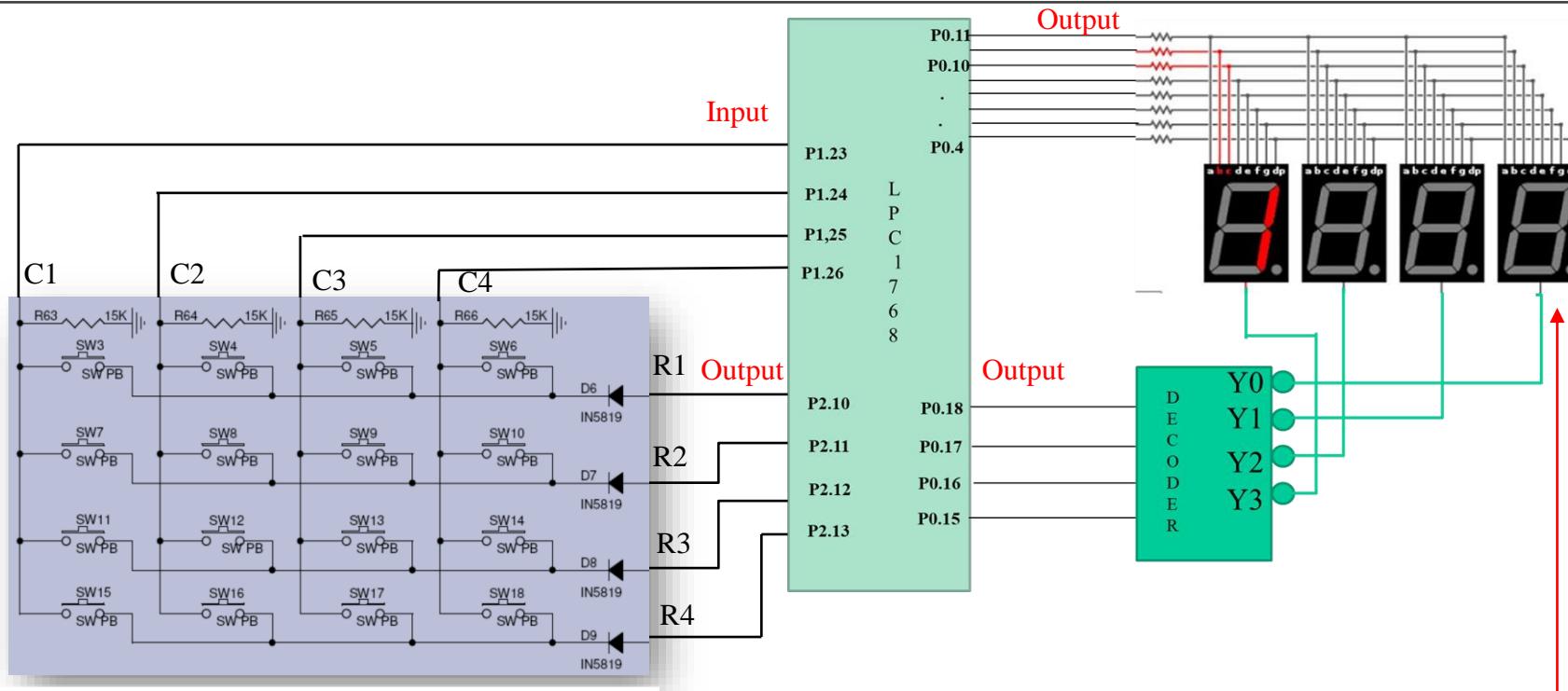
```
// Function to Display Numbers on Seven Segment Display
void Display(void)
{
    LPC_GPIO1->FIOPIN = select_segment[dig_count]; // Select active digit
    LPC_GPIO0->FIOPIN = seven_seg[digit_value[dig_count]] << 4; // Set segment data
    for (i = 0; i < 500; i++); // Small delay (optional)
    LPC_GPIO0->FIOCLR = 0x00000FF0; // Clear display before next update (optional)
}
```

```
// Function to Handle Delay and Counting Logic
void delay(void)
{
    for (i = 0; i < 500; i++); // Small software delay

    if (count == N) // When counter reaches the threshold
    {
        flag = 0xFF; // Set flag to increment counter
        count = 0; // Reset count
    }
    else
        count += 1;
}
```

```
// If flag is set, update digit values
if (flag == 0xFF)
{
    flag = 0;
    digit_value[0] += 1; // Increment first digit
    if (digit_value[0] == 10) // Carry-over logic for multi-digit counter
    {
        digit_value[0] = 0;
        digit_value[1] += 1;
        if (digit_value[1] == 10)
        {
            digit_value[1] = 0;
            digit_value[2] += 1;
            if (digit_value[2] == 10)
            {
                digit_value[2] = 0;
                digit_value[3] += 1;
                if (digit_value[3] == 10) // Reset after 9999
                {
                    digit_value[3] = 0;
                }
            }
        }
    }
}
```

# Matrix Keyboard Interfacing



Display keycode on  
Digit1 (0 to F)

# Matrix Keyboard Interfacing

---

```
#include <LPC17xx.h>

#define FIRST_SEG 0xFFFF87FFF

void scan(void);

unsigned char col, row, flag;
unsigned long int i, var1, temp, temp3, temp2;
unsigned char SEVEN_CODE[4][4] = {
    {0x3F, 0x06, 0x5B, 0x4F},
    {0x66, 0x6D, 0x7D, 0x07},
    {0x7F, 0x6F, 0x77, 0x7C},
    {0x58, 0x5E, 0x79, 0x71}
};
```

# Cont...

```
int main(void) {
    SystemInit();
    SystemCoreClockUpdate();

    LPC_PINCON->PINSEL0 = 0; // P0.4 to P0.11 GPIO data lines
    LPC_GPIO0->FIODIR = 0xFFFFFFFF; // Port 0 output
    LPC_PINCON->PINSEL3 = 0; // P1.23 to P1.26 made GPIO
    LPC_PINCON->PINSEL4 = 0; // P2.10 to P2.13 made GPIO
    LPC_GPIO2->FIODIR = 0x00003C00; // Output P2.10 to P2.13 (rows)
    LPC_GPIO1->FIODIR = 0; // Input P1.23 to P1.26 (cols)
```

# Cont...

```
while (1) {
    for (row = 0; row < 4; row++) {
        if (row == 0) temp = 1 << 10;
        else if (row == 1) temp = 1 << 11;
        else if (row == 2) temp = 1 << 12;
        else if (row == 3) temp = 1 << 13;

        LPC_GPIO2->FIOPIN = temp;
        flag = 0;
        scan();
        if (flag == 1) {
            temp2 = SEVEN_CODE[row][col];
            LPC_GPIO0->FIOMASK = 0xFFFF87FFF;
            LPC_GPIO0->FIOPIN = FIRST_SEG;
            temp2 = temp2 << 4;
            LPC_GPIO0->FIOMASK = 0xFFFFF00F; // Taking Data Lines for 7-Segment Display
            LPC_GPIO0->FIOPIN = temp2;
            break;
        }
    }
}
```

# Cont...

```
void scan(void) {  
    unsigned long temp3;  
    temp3 = LPC_GPIO1->FIOPIN;  
    temp3 &= 0x07800000;  
    if (temp3 != 0x00000000) {  
        flag = 1;  
        if (temp3 == 1 << 23) col = 0;  
        else if (temp3 == 1 << 24) col = 1;  
        else if (temp3 == 1 << 25) col = 2;  
        else if (temp3 == 1 << 26) col = 3;  
    }  
}
```

## Practice Problems

```
LPC_GPIO0->FIODIR = 0xFFFFFFFF;  
LPC_GPIO0->FIOPIN = 0x3456889A;          ..Pin Status:  
LPC_GPIO0->FIOMASK = 0xFFFFFFF00;  
LPC_GPIO0->FIOSET = 0x23FFFF23;          ..Pin Status:  
LPC_GPIO0->FIOMASK1 = 0xF0;  
LPC_GPIO0->FIOCLR1 = 0x38;                ..Pin Status:  
LPC_GPIO0->FIOMASKH = 0xF00F;  
LPC_GPIO0->FIOSETH = 0xABCD;              ..Pin Status:
```

```
LPC_GPIO0->FIODIR = 0xFFFFFFFF;  
LPC_GPIO0->FIOPIN = 0x3456889A;  
LPC_GPIO0->FIOMASK = 0xFFFFFFF00;  
LPC_GPIO0->FIOSET = 0x23FFFF23;  
LPC_GPIO0->FIOMASK1 = 0xF0;  
LPC_GPIO0->FIOCLR1 = 0x38;  
LPC_GPIO0->FIOMASKH = 0xF00F;  
LPC_GPIO0->FIOSETH = 0xABCD;
```

Final Pin Status: 0x3FD680BB

```
LPC_GPIO0->FIODIR = 0xFFFFFFFF;  
LPC_GPIO0->FIOPIN = 0x3456889A; ..Pin Status:  
LPC_GPIO0->FIOMASK = 0x00FFFFFF;  
LPC_GPIO0->FIOSET = 0x23FFFF23; ..Pin Status:  
LPC_GPIO0->FIOMASK2 = 0xF0; ..Pin Status:  
LPC_GPIO0->FIOSET2 = 0x38; ..Pin Status:  
LPC_GPIO0->FIOMASKL = 0xF00F;  
LPC_GPIO0->FIOCLRL = 0xABCD; ..Pin Status:
```

```
LPC_GPIO0->FIODIR = 0xFFFFFFFF;  
LPC_GPIO0->FIOPIN = 0x3456889A;  
LPC_GPIO0->FIOMASK = 0x00FFFFFF;  
LPC_GPIO0->FIOSET = 0x23FFFF23;  
LPC_GPIO0->FIOMASK2 = 0xF0;  
LPC_GPIO0->FIOSET2 = 0x38;  
LPC_GPIO0->FIOMASKL = 0xF00F;  
LPC_GPIO0->FIOCLRL = 0xABCD;
```

Final Pin Status : 0x375E801A

```
LPC_GPIO0->FIODIR = 0xFFFFFFFF;  
LPC_GPIO0->FIOPIN = 0x3456889A; ..Pin Status:  
LPC_GPIO0->FIOMASK = 0xFF00FFFF;  
LPC_GPIO0->FIOSET = 0x2356FF23; ..Pin Status:  
LPC_GPIO0->FIOMASK3 = 0x0F; ..Pin Status:  
LPC_GPIO0->FIOSET3 = 0xAB; ..Pin Status:  
LPC_GPIO0->FIOMASKH = 0xFF0F;  
LPC_GPIO0->FIOCLRH = 0xABCD; ..Pin Status:
```

```
LPC_GPIO0->FIODIR = 0xFFFFFFFF;  
LPC_GPIO0->FIOPIN = 0x3456889A;  
LPC_GPIO0->FIOMASK = 0xFF00FFFF;  
LPC_GPIO0->FIOSET = 0x2356FF23;  
LPC_GPIO0->FIOMASK3 = 0x0F;  
LPC_GPIO0->FIOSET3 = 0xAB;  
LPC_GPIO0->FIOMASKH = 0xFF0F;  
LPC_GPIO0->FIOCLRH = 0xABCD;
```

Final Pin Status: 0x741F889A

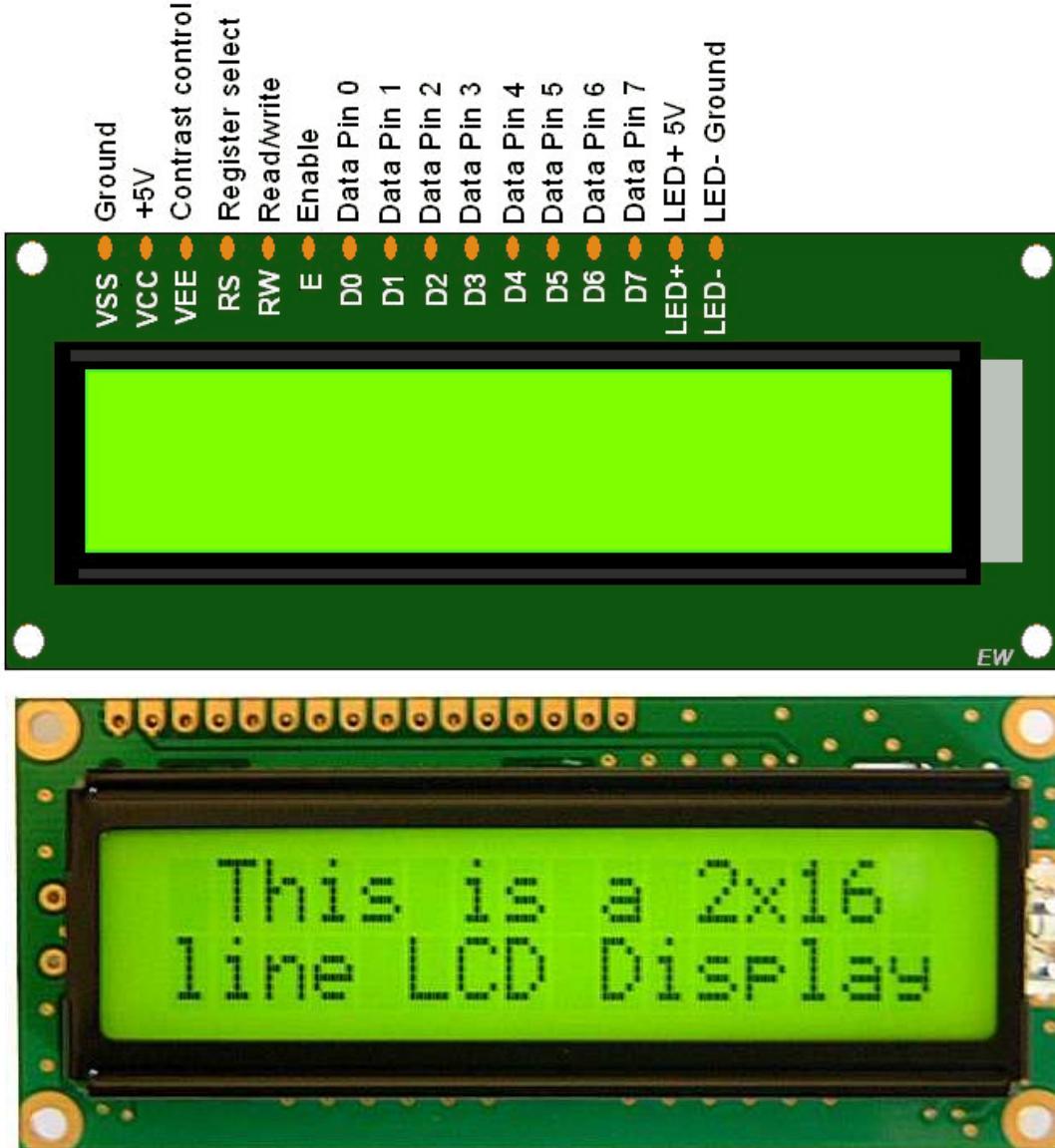


# Embedded System and Design

---

**DR MANOJ TOLANI (PHD-IIIT ALLAHABAD)**  
**ASSISTANT PROFESSOR (DEPARTMENT OF ICT)**

# Liquid Crystal Display (LCD) Interfacing



16x2 Liquid Crystal Display which will display the 32 characters at a time in two rows (16 characters in one row).

- **Pin1 (Ground):** This is a GND pin of display.
- **Pin2 (VCC):** This is the voltage supply pin of the display.
- **Pin3 (VEE):** This pin is used to adjust the contrast by connecting to a potentiometer.
- **Pin4 (Register Select):** This pin used to select command or data register. When RS is 0, Command Register is selected and when RS=1, Data Register is selected
- **Pin5 (Read/Write):** This pin is used to select read or write operation (0 = Write Operation, and 1 = Read Operation).
- **Pin 6 (Enable):** LOW to HIGH transition at this pin to perform Read/Write operation
- **Pins 7-14 (Data Pins):** These pins are used to send data/command to the display. In 8-bit LCD mode all the pins D7 to D0 are used. In 4-bit LCD mode only D7-D4 pins are used.
- **Pin15 (+ve pin of the LED):** This pin is connected to +5V
- **Pin 16 (-ve pin of the LED):** This pin is connected to GND.

# Liquid Crystal Display (LCD) Interfacing

Instruction	Code										Description	Execution Time (max) (when $f_{op}$ or $f_{osc}$ is 270 kHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 s
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 s
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 s
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 s

I/D=1: Increment Mode; I/D=0: Decrement Mode

S=1: Shift

S/C=1: Display Shift; S/C=0: Cursor Shift

R/L=1: Right Shift; R/L=0: Left Shift

DL=1: 8D DL=0: 4D

N=1: 2R N=0: 1R

F=1: 5x10 Style; F=0: 5x7 Style

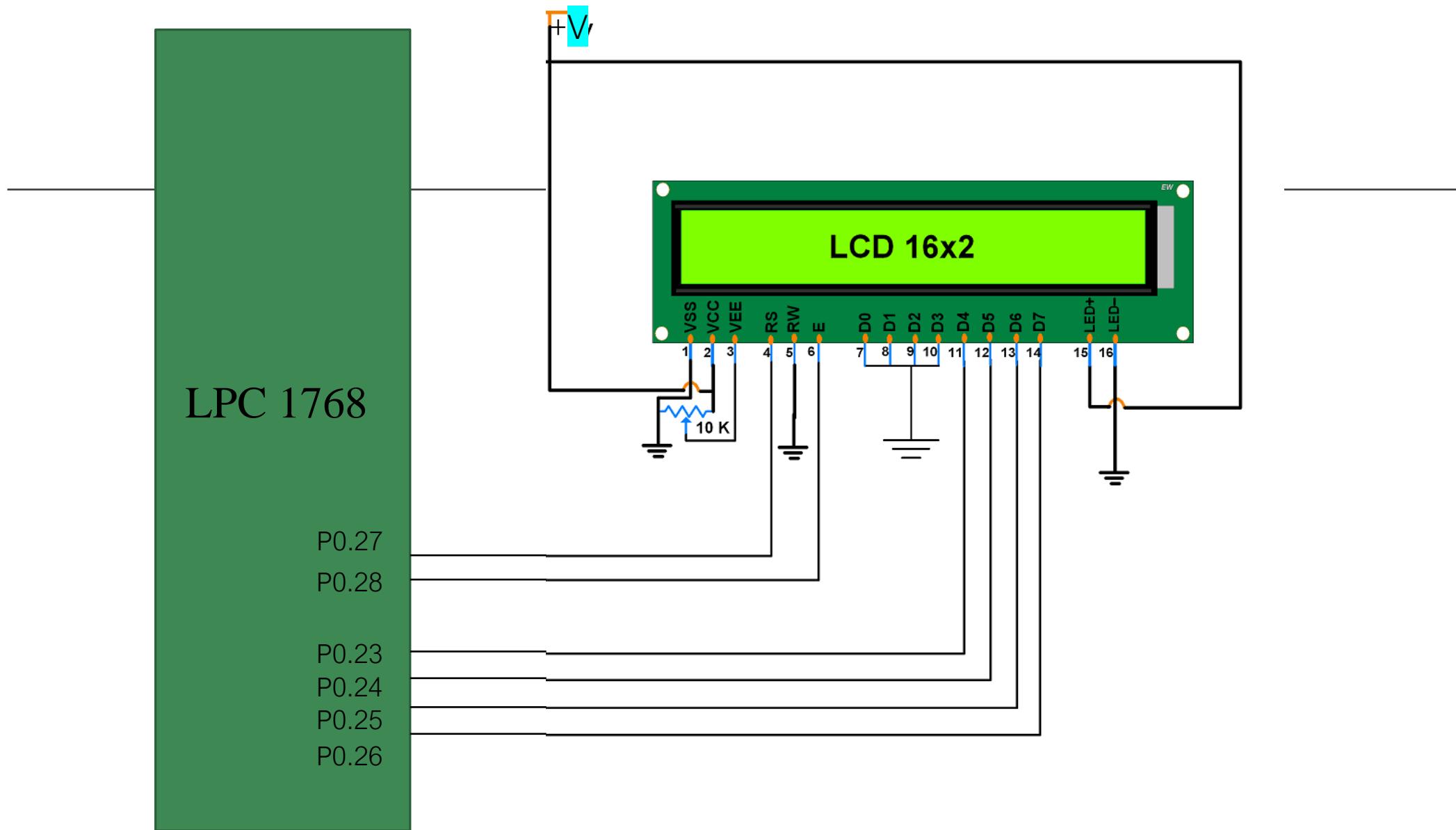
BF=1: Execute Internal Function;

BF=0: Command Received

## LCD Command Codes

Code (Hex)	Command to LCD Instruction Register
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	Display on, cursor blinking
10	Shift cursor position to left OFF
14	Shift cursor position to right ON
18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to beginning to 1st line
C0	Force cursor to beginning to 2nd line
38	2 lines and 5x7 matrix

# Liquid Crystal Display (LCD) Interfacing



# Liquid Crystal Display (LCD) Interfacing

```
#include <lpc17xx.h>
#define RS_CTRL 0x08000000 //P0.27
#define EN_CTRL 0x10000000 //P0.28
#define DT_CTRL 0x07800000 //P0.23 to P0.26 data lines

unsigned long int temp1=0, temp2=0,i,j ;
unsigned char flag1 =0, flag2 =0;
unsigned char msg[] = {"WELCOME "};
void lcd_write(void);
void port_write(void);
void delay_lcd(unsigned int);
unsigned long int init_command[] = {0x30,0x30,0x30,0x20,0x28,0x0c,0x06,0x01,0x80};
int main(void)
{
    SystemInit();
    SystemCoreClockUpdate();
    LPC_GPIO0->FIODIR = DT_CTRL | RS_CTRL | EN_CTRL; //Config output
    flag1 =0;//Command
    for (i=0; i<9;i++)
    {
        temp1 = init_command[i];
        lcd_write(); //send Init commands to LCD
    }
    flag1 =1;//Data
    i=0;
    while (msg[i++] != '\0')
    {
        temp1 = msg[i];
        lcd_write(); //Send data bytes
    }
    while(1);
}
```

3 times send 8-bit mode command, followed by 4-bit mode

0x28	// Function Set: 4-bit, 2 lines, 5x8 dots font
0x0C	// Display ON, Cursor OFF
0x06	// Entry Mode Set: Auto Increment Cursor
0x01	// Clear Display
0x80	// Set DDRAM Address to 0x00 (Cursor Home)

# Liquid Crystal Display (LCD) Interfacing (Contd...)

```
void lcd_write(void)
{
    flag2 = (flag1 == 1) ? 0 :((temp1 == 0x30) || (temp1 == 0x20)) ? 1 : 0;//If command is 0x30 (Working in 8-bit mode initially), send '3' on D7-D4 (D3-D0 already grounded)
    temp2 = temp1 & 0xf0;;
    temp2 = temp2 << 19;//data lines from 23 to 26. Shift left (26-8+1) times so that higher digit is sent on P0.26 to P0.23
    port_write(); // Output the higher digit on P0.26-P0.23
    if (!flag2) // Other than command 0x30, send the lower 4-bit also
    {
        temp2 = temp1 & 0x0f; //26-4+1
        temp2 = temp2 << 23;
        port_write(); // Output the lower digit on P0.26-P0.23
    }
}
void port_write(void)
{
    LPC_GPIO0->FIOPIN = temp2;
    if (flag1 == 0)
        LPC_GPIO0->FIOCLR = RS_CTRL; // Select command register
    else
        LPC_GPIO0->FIOSET = RS_CTRL; //Select data register

    LPC_GPIO0->FIOSET = EN_CTRL; //Apply edge on Enable
    delay_lcd(25);
    LPC_GPIO0->FIOCLR = EN_CTRL;
    delay_lcd(5000);

}
void delay_lcd(unsigned int r1)
{
    unsigned int r;
    for(r=0;r<r1;r++);
    return;
}
```

Objective: First Line show Message 'Counter', Second Line Count Values from 0-9 (Display on LCD)

```
unsigned char msg[] = {"COUNTER"};
unsigned char counter = '0'; // Start counting from ASCII '0'
```

```
int main(void)
{
    SystemInit();
    SystemCoreClockUpdate();
    LPC_GPIO0->FIODIR = DT_CTRL | RS_CTRL | EN_CTRL; // Config output

    flag1 = 0; // Command Mode
    for (i=0; i<9; i++)
    {
        temp1 = init_command[i];
        lcd_write(); // Send Init commands to LCD
    }

    flag1 = 1; // Data Mode

    i = 0;
    while (msg[i] != '\0')
    {
        temp1 = msg[i];
        lcd_write(); // Send data bytes
    }
}
```

```
while (1)
{
    temp1 = 0xC0; // Move cursor back to first position
    flag1 = 0; // Command mode
    lcd_write();

    flag1 = 1; // Back to Data mode
    temp1 = counter; // Load counter value (ASCII digit)
    lcd_write(); // Display on LCD

    delay_lcd(5000000); // Delay to make count visible

    counter++; // Increment counter
    if (counter > '9') // Reset after 9
    {
        counter = '0';

    }
}
```

## Objective: First Line show Message 'Counter', Second Line Count Values from 00-99 (Display on LCD)

```
unsigned char msg[] = {"COUNTER"};
unsigned char counter = '0'; // Start counting from ASCII '0'
unsigned char counter1 = '0'; // Start counting from ASCII '0'
```

```
int main(void)
{
    SystemInit();
    SystemCoreClockUpdate();
    LPC_GPIO0->FIODIR = DT_CTRL | RS_CTRL | EN_CTRL; // Config output

    flag1 = 0; // Command Mode
    for (i=0; i<9; i++)
    {
        temp1 = init_command[i];
        lcd_write(); // Send Init commands to LCD
    }

    flag1 = 1; // Data Mode

    i =0;
    while (msg[i++] != '\0')
    {
        temp1 = msg[i-1];
        lcd_write(); //Send data bytes
    }
}
```

```
while (1)
{
    temp1 = 0xC1; // Move cursor back to first position
    flag1 = 0; // Command mode
    lcd_write();

    flag1 = 1; // Back to Data mode
    temp1 = counter; // Load counter value (ASCII digit)
    lcd_write(); // Display on LCD

    temp1 = 0xC0; // Move cursor back to first position
    flag1 = 0; // Command mode
    lcd_write();

    flag1 = 1; // Back to Data mode
    temp1 = counter1; // Load counter value (ASCII digit)
    lcd_write(); // Display on LCD

    delay_lcd(5000000); // Delay to make count visible

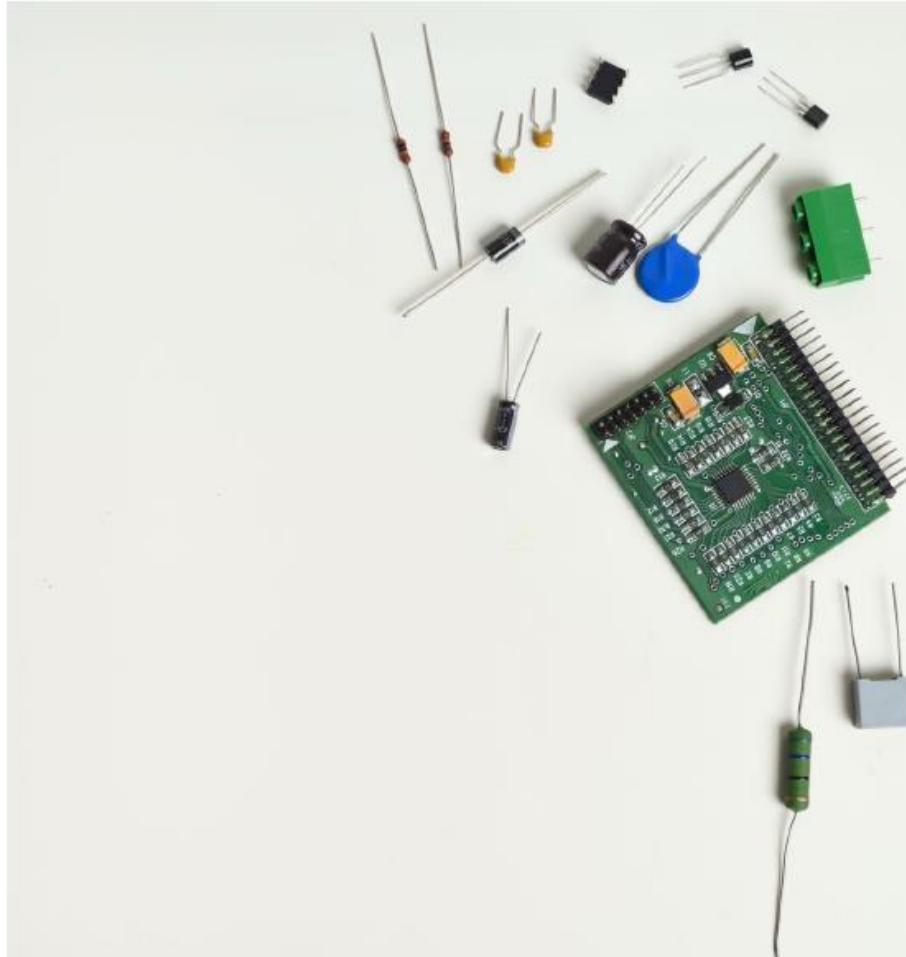
    counter++; // Increment counter
    if (counter > '9') // Reset after 9
    {
        counter = '0';
        counter1++; // Increment counter
        if (counter1 > '9') // Reset after 9
        {
            counter1 = '0';
        }
    }
}
```



# Embedded System and Design

---

**DR MANOJ TOLANI (PHD-IIIT ALLAHABAD)**  
**ASSISTANT PROFESSOR (DEPARTMENT OF ICT)**



# MY INTRODUCTION

- First, let me introduce myself. My name is **Manoj Tolani**. I joined MIT, Manipal (MAHE) in **April 2023**.
- I am currently **Assistant Professor in Information and Communication Technology Department**.
- I received an **MTech degree (Specialization: Digital Systems)** from the **Madan Mohan Malviya Engineering College, Gorakhpur** (Now MMMUT) in 2012.
- After MTech, I joined **PSIT Engineering College** as an **Assistant Professor in 2012**. I worked for **3.5 Years at PSIT**.
- I attempted **GATE Exam in 2015**, and my **AIR was 1214**. I also qualified for UGC-NET also in 2015.
- In **2016**, I joined **IIIT Allahabad** as a research scholar for a PhD Degree.
- I **completed my Ph.D. in Feb 2021**. I worked **6 months at IIIT-Allahabad as Teaching-cum-Research Associate**.
- After that, I joined **Atria Institute of Technology, Bangalore** as an Assistant Professor. I worked for **2 Years at AIT**, and after that, I joined this college (**MIT, Manipal**) in **April 2023**.
- My research interest includes **Wireless Sensor Networks, IoT, Photonics, and Machine Learning**.
- I have published **18+ research papers in SCI/Scopus Journals/Conferences** (IEEE, Elsevier, Springer etc.)

## KEYBOARD INTERFACING with LCD

LCD: P0.23 to P0.26 data lines  
LCD: RS 0.27, EN 0.28

Keyboard: Output P2.10 to P2.13 (rows)  
Keyboard: Input P1.23 to P1.26 (cols)

```
#include <lpc17xx.h>

#define RS_CTRL 0x08000000 // P0.27
#define EN_CTRL 0x10000000 // P0.28
#define DT_CTRL 0x07800000 // P0.23 to P0.26 data lines

unsigned long int temp=0, temp11=0, temp12=0, i;
unsigned char flag=0, flag1 =0, flag2 =0;
unsigned char col, row;
unsigned char msg[] = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};

void lcd_write(void);
void port_write(void);
void scan(void);
void delay_lcd(unsigned int);

unsigned long int init_command[] = {0x30,0x30,0x30,0x20,0x28,0x0c,0x06,0x01,0x80};
```

```
int main(void)
{
    SystemInit();
    SystemCoreClockUpdate();
    LPC_GPIO0->FIODIR = DT_CTRL | RS_CTRL | EN_CTRL; // Config output

    LPC_GPIO2->FIODIR = 0x00003C00; // Output P2.10 to P2.13 (rows)
    LPC_GPIO1->FIODIR = 0; // Input P1.23 to P1.26 (cols)

    flag1 = 0; // Command Mode
    for (i=0; i<9; i++)
    {
        temp11 = init_command[i];
        lcd_write(); // Send Init commands to LCD
    }

    while (1) {
        flag1 = 0; // Command Mode
        temp11 = 0x80;
        lcd_write();
    }
}
```

```
flag1=1;
for (row = 0; row < 4; row++) {
    if (row == 0) temp = 1 << 10;
    else if (row == 1) temp = 1 << 11;
    else if (row == 2) temp = 1 << 12;
    else if (row == 3) temp = 1 << 13;

    LPC_GPIO2->FIOPIN = temp;
    flag = 0;
    scan();

    if (flag == 1) {
        temp11 = msg[row*4+col];
        lcd_write();
        break;
    }
}}
```

```
void scan(void) {
    unsigned long temp3;
    temp3 = LPC_GPIO1->FIOPIN;
    temp3 &= 0x07800000;
    if (temp3 != 0x00000000) {
        flag = 1;
        if (temp3 == 1 << 23) col = 0;
        else if (temp3 == 1 << 24) col = 1;
        else if (temp3 == 1 << 25) col = 2;
        else if (temp3 == 1 << 26) col = 3;
    }
}
```

```
void lcd_write(void)
{
    flag2 = (flag1 == 1) ? 0 : ((temp11 == 0x30) || (temp11 == 0x20)) ? 1 : 0;

    temp12 = temp11 & 0xf0;
    temp12 = temp12 << 19; // Send higher nibble
    port_write();

    if (!flag2)
    {
        temp12 = temp11 & 0x0f;
        temp12 = temp12 << 23; // Send lower nibble
        port_write();
    }
}
```

```
void port_write(void)
{
    LPC_GPIO0->FIOPIN = temp12;

    if (flag1 == 0)
        LPC_GPIO0->FIOCLR = RS_CTRL; // Command mode
    else
        LPC_GPIO0->FIOSET = RS_CTRL; // Data mode

    LPC_GPIO0->FIOSET = EN_CTRL; // Apply enable pulse
    delay_lcd(25);
    LPC_GPIO0->FIOCLR = EN_CTRL;
    delay_lcd(50000);
}

void delay_lcd(unsigned int r1)
{
    unsigned int r;
    for (r = 0; r < r1; r++);
}
```



# Embedded System and Design

---

**DR MANOJ TOLANI (PHD-IIIT ALLAHABAD)**  
**ASSISTANT PROFESSOR (DEPARTMENT OF ICT)**

# Timer/Counter Programming

---

**Timer** - Interval Timer to generate the intended delay. The Timer is designed to count cycles of the peripheral clock (PCLK)

**Counter** - Counting internal events. or an externally-supplied clock

There are four 32-bit Timers in LPC1768:

Timer0, Timer1, Timer2 and Timer3.

# Timer/Counter Programming

---

**PC** – Prescale Counter Register: It is a 32-bit register. The value in PC is incremented on every PCLK cycle and when its value matches the value in PR, the TC is incremented and the value in PC is RESET on the next PCLK cycle.

**PR** – Prescale Register: It is a 32-bit register and specifies the maximum value for the Prescale Counter (PC).

**TC** – Timer Counter Register: It is a 32-bit register and is incremented at every PR+1 cycles of PCLK.

**MR0 – MR3 – Match Registers:** Contains user loaded values to be compared with TC. When value in Match Register matches with TC, appropriate action can be performed.

## Calculating Timer Prescaler

First, we have to select the peripheral clock for the Timer. This is done with the help of PCLKSEL0 and PCLKSEL1 registers. The following table shows the PCLKSEL Register bits for each timer.

Timer0	PCLKSEL0 [3:2]
Timer1	PCLKSEL0 [5:4]
Timer2	PCLKSEL1 [13:12]
Timer3	PCLKSEL1 [15:14]

The value in these PCLKSEL bits will determine the final PCLK of the Timers.

00	PCLK = CCLK / 4 (Default)
01	PCLK = CCLK
10	PCLK = CCLK / 2
11	PCLK = CCLK / 8

After the PCLK is finalized, we can calculate the Prescaler Value.

The time period of one clock cycle is given by:

$$T_{PCLK} = \frac{1}{PCLK}$$

The minimum time duration (or the resolution of the Timer) produced by a Timer for a given PCLK and PR values is given by:

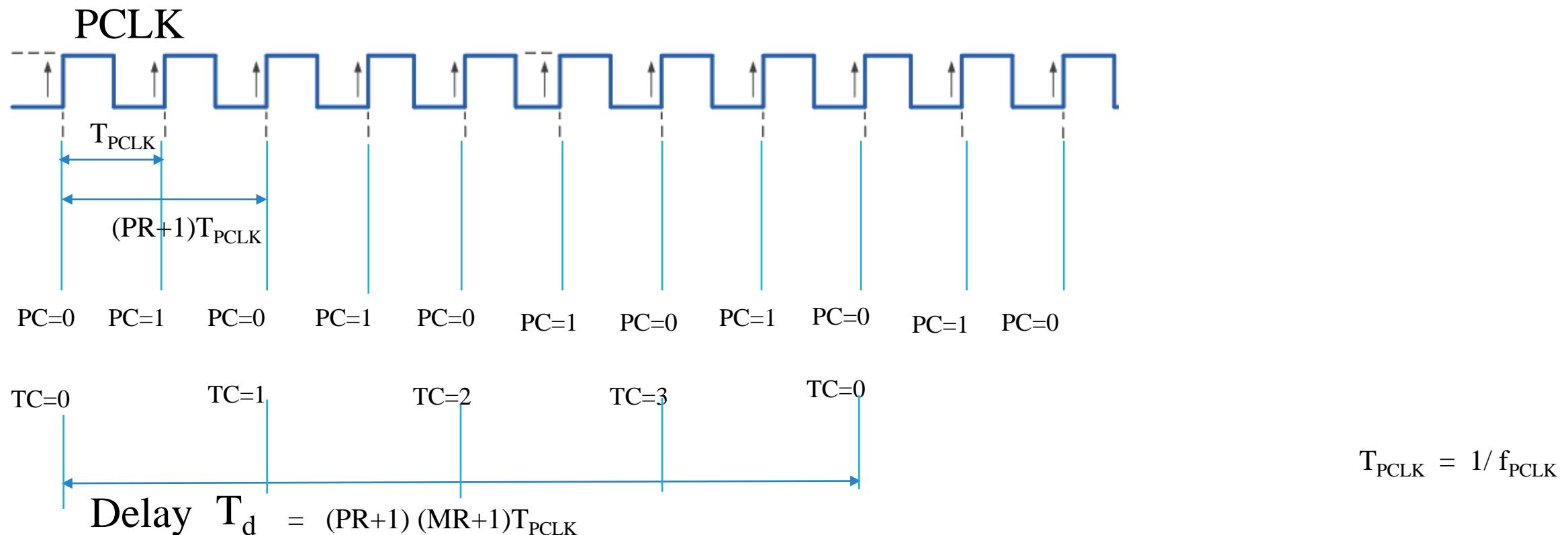
$$T_{RES} = \frac{PR+1}{PCLK}$$

From this, we can calculate the PR value for various timer resolutions. For example, if we want 1 millisecond resolution on a timer with 25 MHz PCLK, then

$$PR = (25 * 10^6 * 1 * 10^{-3}) - 1 = 24999$$

# Timer/Counter Programming

PR=1, MR0= 3



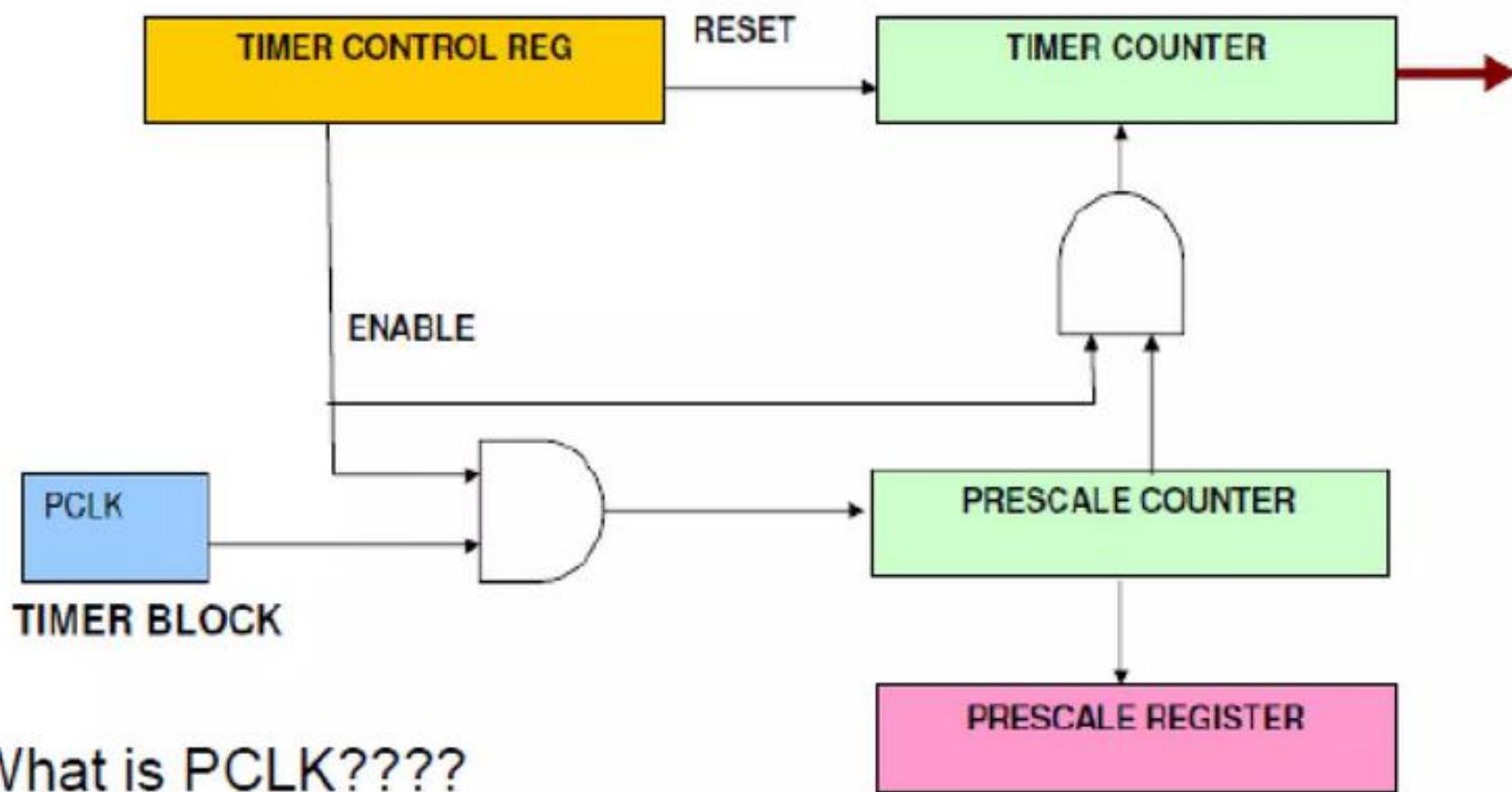
Ex: For frequency 3MHz, to get 1 second delay: PR= 999, MR0= 2999 PR= 2999, MR0= 999 etc.

# Timer/Counter Programming

---

TCR – Timer Control register: Used to control Timer Counter functions i.e. enable, disable and reset.

Bit 0	Counter Enable	When 1, TC and PC are enabled for counting. When 0, TC and PC are disabled.
Bit 1	Counter Reset	When 1, TC and PC are reset on next positive edge of PCLK. The counters remain reset until this bit is returned to 0.



What is PCLK????

# Timer/Counter Programming

## Match Registers (MR0 - MR3)

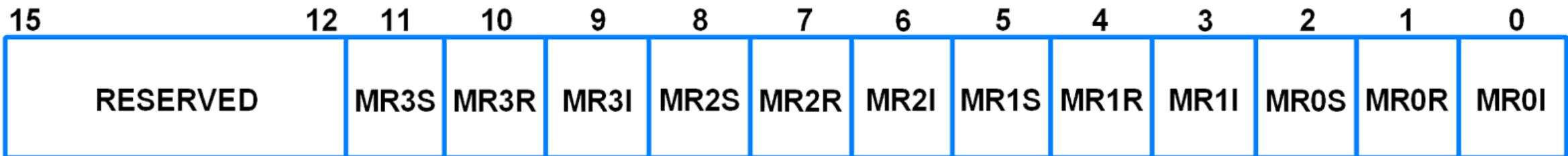
---

- The Match register values are continuously compared to the Timer Counter value.
- When the two values are equal, actions can be triggered automatically.
- The action possibilities are to generate an interrupt, reset the Timer Counter, or stop the timer. Actions are controlled by the settings in the MCR register.

**MCR – Match Control Register:** Used to control the operation on the TC when one of the Match Register matches with TC.

Bits [2:0] are used for MR0, Bits [5:3] are used for MR1, Bits [8:6] are used for MR2 and Bits [11:9] are used for MR3. Remaining bits are reserved. Let us see the actions for one match register MR0.

Bit 0	MR0I	When 1, an interrupt is generated when MR0 matches with TC.
Bit 1	MR0R	When 1, reset TC if MR0 matches with TC.
Bit 2	MR0S	When 1, stop TC and PR and set TCR[0] to 0 if MR0 matches with TC.



# Match Control Register (MCR)

The Match Control Register is used to control what operations are performed when one of the Match Registers matches the Timer Counter.

Bit	Symbol	Value	Description
0	MR0I	1	Interrupt on MR0: an interrupt is generated when MR0 matches the value in the TC.
		0	This interrupt is disabled
1	MR0R	1	Reset on MR0: the TC will be reset if MR0 matches it.
		0	Feature disabled.
2	MR0S	1	Stop on MR0: the TC and PC will be stopped and TCR[0] will be set to 0 if MR0 matches the TC.
		0	Feature disabled.
3	MR1I	1	Interrupt on MR1: an interrupt is generated when MR1 matches the value in the TC.
		0	This interrupt is disabled
4	MR1R	1	Reset on MR1: the TC will be reset if MR1 matches it.
		0	Feature disabled.
5	MR1S	1	Stop on MR1: the TC and PC will be stopped and TCR[0] will be set to 0 if MR1 matches the TC.
		0	Feature disabled.

# Timer/Counter Programming

---

MR2I	1	Interrupt on MR2: an interrupt is generated when MR2 matches the value in the TC.
	0	This interrupt is disabled
MR2R	1	Reset on MR2: the TC will be reset if MR2 matches it.
	0	Feature disabled.
MR2S	1	Stop on MR2: the TC and PC will be stopped and TCR[0] will be set to 0 if MR2 matches the TC.
	0	Feature disabled.
MR3I	1	Interrupt on MR3: an interrupt is generated when MR3 matches the value in the TC.
	0	This interrupt is disabled
MR3R	1	Reset on MR3: the TC will be reset if MR3 matches it.
	0	Feature disabled.
MR3S	1	Stop on MR3: the TC and PC will be stopped and TCR[0] will be set to 0 if MR3 matches the TC.
	0	Feature disabled.
-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.

**EMR - External Match Register:** Controls the External Match Pins MAT0.0, MAT0.1, MAT1.0, etc. Bit 0 controls the MATx.0 Pin of the Timerx and the value of this bit is determined by the bits [5:4].

**External Match Register (EMR):** It provides Match Outputs. Also, the External Match Register provides both control and status of the external match pins.

Bit	Symbol	Description
0	EM0	External Match 0. When a match occurs between the TC and MR0, this bit can either toggle, go low, go high, or do nothing, depending on bits 5:4 of this register. This bit can be driven onto a MATn.0 pin, in a positive-logic manner (0 = low, 1 = high).
1	EM1	External Match 1. When a match occurs between the TC and MR1, this bit can either toggle, go low, go high, or do nothing, depending on bits 7:6 of this register. This bit can be driven onto a MATn.1 pin, in a positive-logic manner (0 = low, 1 = high).
2	EM2	External Match 2. When a match occurs between the TC and MR2, this bit can either toggle, go low, go high, or do nothing, depending on bits 9:8 of this register. This bit can be driven onto a MATn.2 pin, in a positive-logic manner (0 = low, 1 = high).
3	EM3	External Match 3. When a match occurs between the TC and MR3, this bit can either toggle, go low, go high, or do nothing, depending on bits 11:10 of this register. This bit can be driven onto a MATn.3 pin, in a positive-logic manner (0 = low, 1 = high).
5:4	EMC0	External Match Control 0. Determines the functionality of External Match 0. <a href="#">Table 433</a> shows the encoding of these bits.
7:6	EMC1	External Match Control 1. Determines the functionality of External Match 1. <a href="#">Table 433</a> shows the encoding of these bits.
9:8	EMC2	External Match Control 2. Determines the functionality of External Match 2. <a href="#">Table 433</a> shows the encoding of these bits.
11:10	EMC3	External Match Control 3. Determines the functionality of External Match 3. <a href="#">Table 433</a> shows the encoding of these bits.

External Match Outputs	External Match Bit in EMR	Control Bits in EMR	Value in Control Bits	Action on External Match Pins / Bits
MATx.0	0	5:4	00	Do nothing
			01	Clear the Bit
			10	Set the Bit
			11	Toggle the Bit

Similarly,

- MATx.1, External Match Bit is 1 and Control bits are [7:6].
- MATx.2, External Match Bit is 2 and Control bits are [9:8].
- MATx.3, External Match Bit is 3 and Control bits are [11:10].

The Timer in LPC1768 MCU has two sets of pins associated with it. One set of pins are Capture Input Pins while the other set of pins are External Match Output Pins.

The naming convention for the pins names of the timer peripheral is as follows:

CAPx.y means Capture Input y of Timer x.

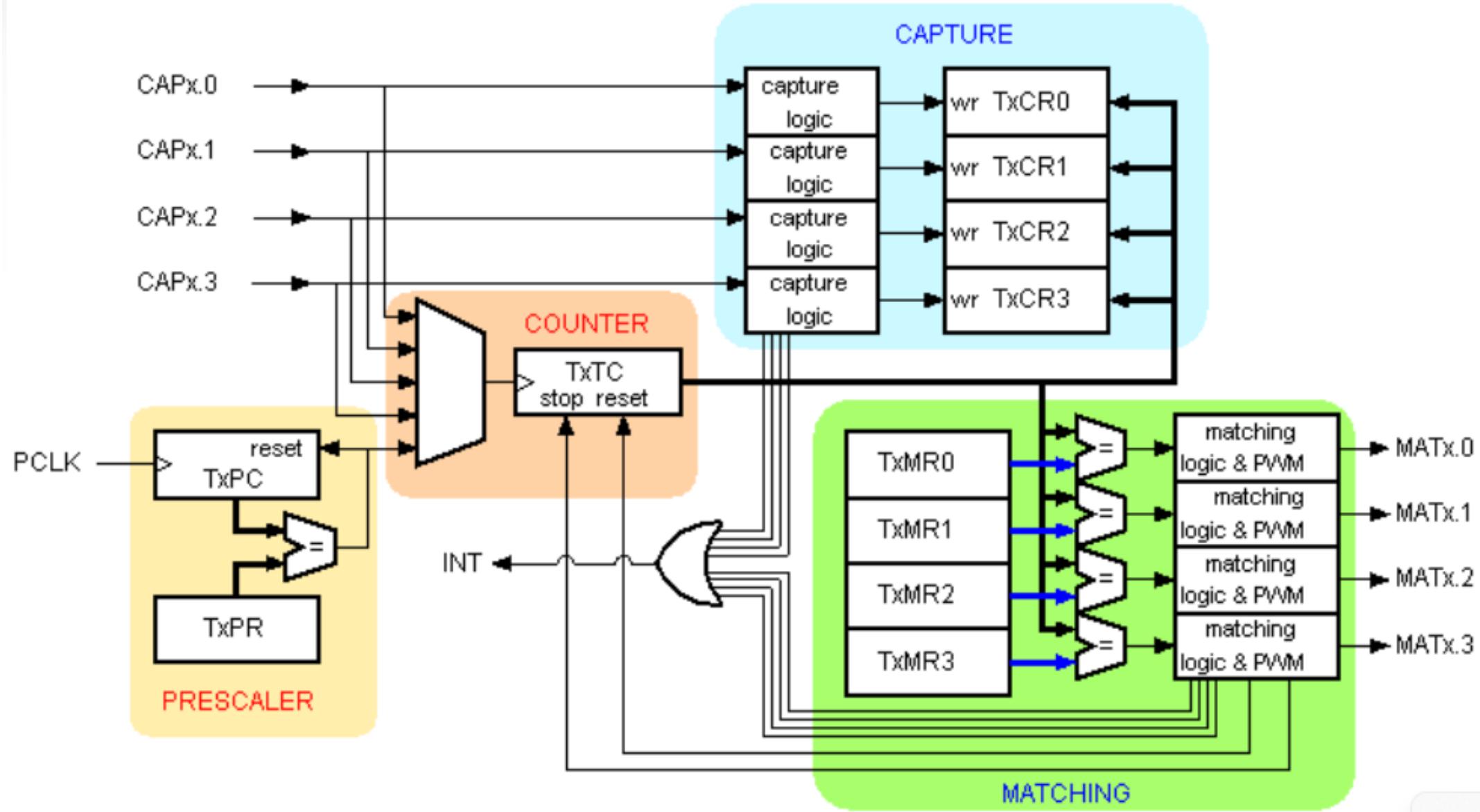
CAP0.0	P1.26
CAP0.1	P1.27
CAP1.0	P1.18 / P1.28 / P2.6
CAP1.1	P1.19 / P1.29
CAP2.0	P0.4
CAP2.1	P0.5
CAP3.0	P0.23
CAP3.1	P0.24
MAT0.0	P1.28 / P3.25
MAT0.1	P1.29 / P3.26
MAT1.0	P1.22
MAT1.1	P1.25
MAT2.0	P0.6 / P4.28
MAT2.1	P0.7 / P4.29
MAT2.2	P0.8
MAT2.3	P0.9
MAT3.0	P0.10
MAT3.1	P0.11

# Timer/Counter Programming

---

CTCR ( Count Control Register) is used to select between Timer and Counter mode, and in Counter mode to select the pin and edge(s) for counting.

Bit	Symbol	Value	Description
1:0	Counter/ Timer Mode		This field selects which rising PCLK edges can increment the Timer's Prescale Counter (PC), or clear the PC and increment the Timer Counter (TC).
		00	Timer Mode: the TC is incremented when the Prescale Counter matches the Prescale Register. The Prescale Counter is incremented on every rising PCLK edge.
		01	Counter Mode: TC is incremented on rising edges on the CAP input selected by bits 3:2.
		10	Counter Mode: TC is incremented on falling edges on the CAP input selected by bits 3:2.
		11	Counter Mode: TC is incremented on both edges on the CAP input selected by bits 3:2.
3:2	Count Input Select		When bits 1:0 in this register are not 00, these bits select which CAP pin is sampled for clocking.
		00	CAPn.0 for TIMERn
		01	CAPn.1 for TIMERn
		10	Reserved
		11	Reserved



1. **Toggle LED connected to P0.2 every second i.e. generate square wave with period 2 seconds**
2. **Generate square wave of period 2 seconds with 75% duty cycle on P0.2**

# Timer/Counter Programming

Toggle LED connected to P0.2 every second i.e. generate square wave with period 2 seconds

```
#include<stdio.h>
#include<LPC17xx.h>
void delay(void)
{
    LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
    LPC_TIM0->EMR = 0X20;//Set match bit upon match
    LPC_TIM0->PR = 1000; /
    LPC_TIM0->MR0 = 3000;      //for 1 second
    LPC_TIM0->MCR = 0x00000004; // stop PC and TC on MR0
    LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    while ( !(LPC_TIM0->EMR & 0x01)); // wait until match

    return;
}

int main(void)
{
    LPC_GPIO0->FIODIR=0x00000004;

    while(1)
    {

        LPC_GPIO0->FIOSET=0x4;
        delay();
        LPC_GPIO0->FIOPIN=~(LPC_GPIO0->FIOPIN & 0x00000004);
        Delay();

        LPC_GPIO0->FIOCLR=0x4;
        delay();
    }
}
```

# Timer/Counter Programming

```
#include<stdio.h>
#include<LPC17xx.h>
void delay(void)
{
    LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
    LPC_TIM0->EMR = 0X20; //Set EM0 upon match
    LPC_TIM0->PR = 2999;
    LPC_TIM0->MCR = 0x00000004; // stop PC and TC on MRO
    LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    while ( !(LPC_TIM0->EMR & 0x01));// Wait until EM0 is set
    return;
}
int main(void)
{
    LPC_GPIO0->FIODIR=0x00000004;
        while(1)
    {
        LPC_GPIO0->FIOPIN=0x00000004;
        LPC_TIM0->MRO = 1500; //For 1.5 seconds
        delay();
        LPC_GPIO0->FIOPIN=0x00000000;
        LPC_TIM0->MRO = 500; //For 0.5 seconds
        delay();
    }
}
```

**Generate square wave of period 2 seconds with 75% duty cycle on P0.2**

1. Toggle MAT 0.0 (after every 1 second) output line by taking EM0 on the output pin.

27		P3.25	MAT0.0	PWM1.2	
28		VDD			
29	USBD+	P0.29	USB_D+		
30	USBD-	P0.30	USB_D-		
31		VSS			
32	LED1	P1.18	USB_UP	PWM1.1	CAP1.0
33		P1.19	MC0A	USB_PPWR	CAP1.1
34	LED2	P1.20	MCFB0	PWM1.2	SCK0
35	LED3	P1.21	MCABORT	PWM1.3	SSEL0
36		P1.22	MC0B	USB_PWRD	MAT1.0
37	LED4	P1.23	MCFB1	PWM1.4	MISO0
38		P1.24	MCFB2	PWM1.5	MOSI0
39		P1.25	MC1A	MAT1.1	
40		P1.26	MC1B	PWM1.6	CAP0.0
41		VSS			
42		VDD			
43		P1.27	CLKOUT	USB_OVRCR	CAP0.1
44		P1.28	MC2A	PCAP1.0	MAT0.0
45		P1.29	MC2B	PCAP1.1	MAT0.1
46	p9	P0.0	CAN_RX1	TXD3	SDA1
47	p10	P0.1	CAN_TX1	RXD3	SCL1
48	p28	P0.10	TXD2	SDA2	MAT3.0
49	p27	P0.11	RXD2	SCL2	MAT3.1
50		P2.13	EINT3N	I2STX_SDA	

# Timer/Counter Programming

**Toggle MAT 0.0 output line (after every 1 second) by taking EM0 on the output pin.**

```
#include<stdio.h>
#include<LPC17xx.h>
void delay(void)
{
    LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
    LPC_TIM0->CTCR = 0x00000000;
    LPC_TIM0->EMR = 0X30;//Toggle bit upon match
    LPC_TIM0->PR = 1000; //
    LPC_TIM0->MR0 = 3000;      //
    LPC_TIM0->MCR = 0x00000002; // Reset TC
    LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    return;
}
int main(void)
{
    LPC_PINCON->PINSEL3 |= (3<<24);//Get EM0 on MAT0.0 (P1.28) line
    delay();
    while(1);
}
```

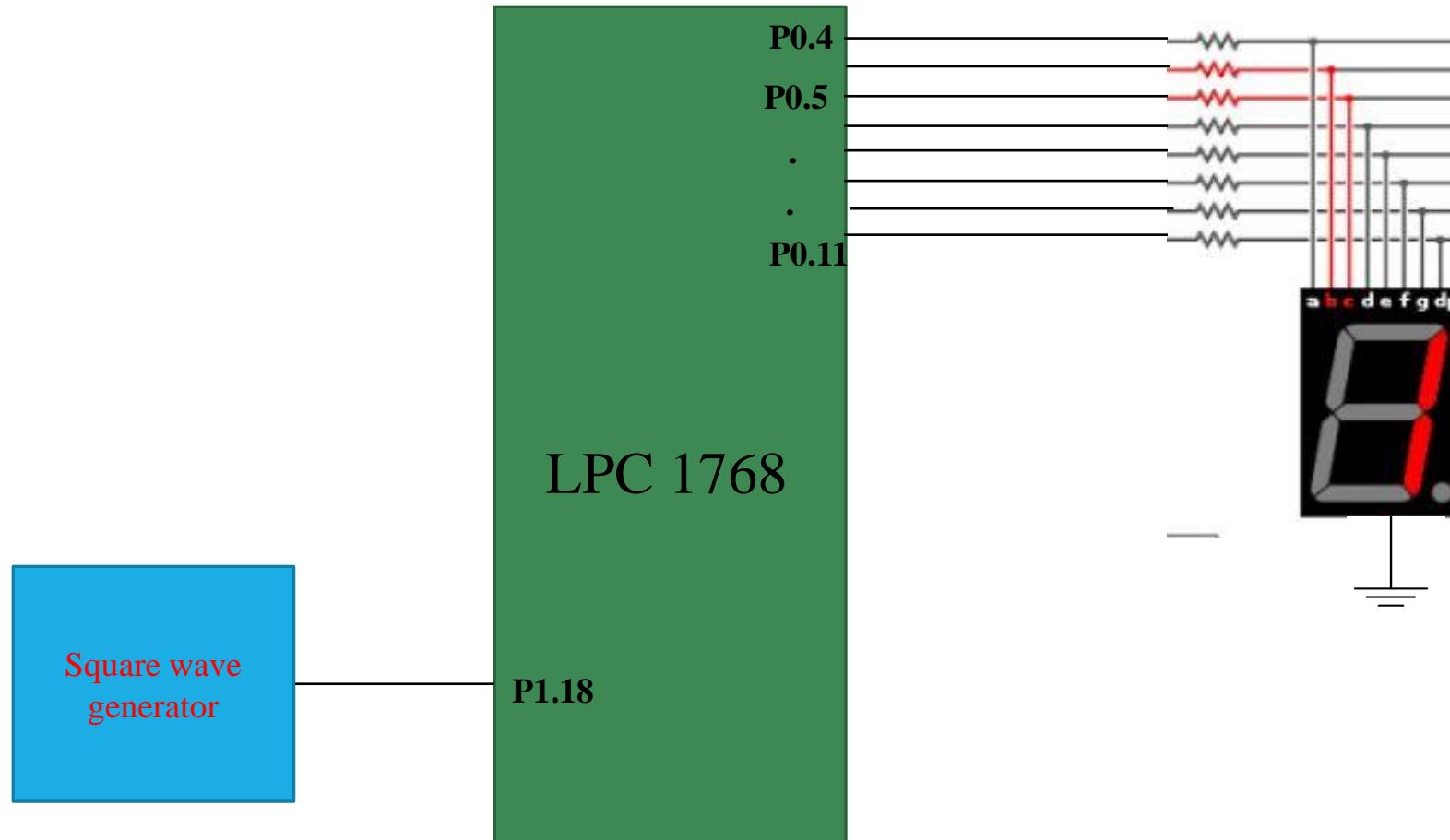
# Timer/Counter Programming

MAT 1.1(P1.25) toggles whenever count reaches 3. CAP 1.0 (P1.18) is counter clock. i.e Divide the frequency of the square waveform input at P1.18 by a factor of 8 on P1.25

```
#include<stdio.h>
#include<LPC17xx.h>
void init_timer1(void)
{
    LPC_PINCON->PINSEL3 |=(3<<18 | 3<<4); // MAT 1.1(P1.25) and CAP 1.0 (P1.18)
    LPC_TIM1->TCR=2;                      //Reset Counter1
    LPC_TIM1->CTCR = 0x2;                  // Counter at -ve edge of CAP1.0
    LPC_TIM1->MR1=0x03;                   //To count 4 clock pulses
    LPC_TIM1->MCR=0x10;                  //Clear TC upon Match1
    LPC_TIM1->EMR=0xC0;                  //Toggle EM1 upon Match
    LPC_TIM1->TCR=1;                     //Start Counter1
}
int main(void)
{
    init_timer1();
    while(1);
}
```

# Timer/Counter Programming

Assume that output of a square wave generator (Frequency < 10Hz) is connected to P1.18 (CAP1.0, Function-3), write a program to display the frequency of this square waveform on the seven segment connected to P011-P0.4.



# Timer/Counter Programming

```
#include<stdio.h>
#include<LPC17xx.h>
unsigned char seven_seg[10]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
void delay(void)
{
    LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
    LPC_TIM0->EMR = 0X20;//Set match bit upon match
    LPC_TIM0->PR = 3000; //for 1 ms
    LPC_TIM0->MRO = 1000; //for 1 second
    LPC_TIM0->MCR = 0x00000004; // stop PC and TC on MRO
    LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    while ( !(LPC_TIM0->EMR & 0x01)); // wait until match
}
```

# Timer/Counter Programming

```
int main(void)
{
    LPC_PINCON->PINSEL0 = 0      //P0.4 to P0.11 GPIO data lines
    LPC_GPIO0->FIODIR = 0x00000FF0;      //P0.4 to P0.11 output
    LPC_GPIO1->FIODIR = 0x07800000; //P1.23 to P1.26 output
    LPC_PINCON->PINSEL3 = (3<<4); // cap 1.0 (P1.18)
    LPC_TIM1->CTCR = 0x01; // Counter at +ve edge of CAP1.0

    while(1)
    {
        LPC_TIM1->TCR=2;//Reset Counter1
        LPC_TIM1->TCR=1;//Start Counter1
        Delay(); // wait for 1 second
        LPC_GPIO1->FIOPIN = 0<< 23;
        LPC_GPIO0->FIOPIN = seven_seg[LPC_TIM1->TC ] << 4; Counter1 on the seven segment
    }
}
```

## Analog To Digital Converter (ADC)

- Analog to Digital Conversion is used when we want to interface an external analog signal or when interfacing analog sensors, like for example a temperature sensor.
- The ADC block in LPC1768 Microcontroller is based on Successive Approximation Register(SAR) conversion method.
- LPC1768 ADC Module uses 12-bit SAR.
- The Measurement range is from VREFN to VREFP, or commonly from 0V to 3.3 V.
- Maximum of 8 multiplexed inputs can be used for ADC.

## Analog To Digital Converter (ADC)

Analog voltage and Digital value of the ADC are related as follows:

$$V_A = (V_{REFP} / 2^N) \text{ (Decimal Equivalent of Digital value)}$$

$V_A$  is the Input analog voltage

$V_{REFP}$  Is the Reference voltage of ADC

N is the number of bits

$(V_{REFP} / 2^N)$  is Resolution; It is a constant for given value of N and  $V_{REFP}$

Digital output is directly proportional to Analog input voltage

For 3.3 V, N=12, Resolution is 0.805 mV. i.e 0.805 mV change at the input creates  $\pm 1$  change at the digital output.

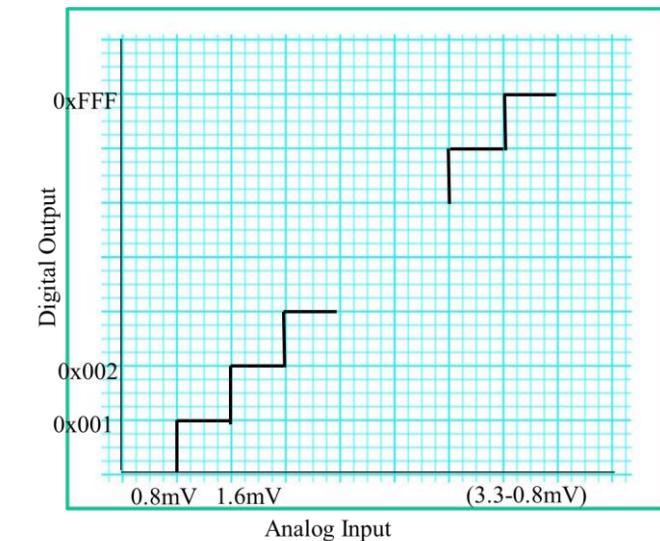
When analog voltage is 0 mV output decimal value is 0

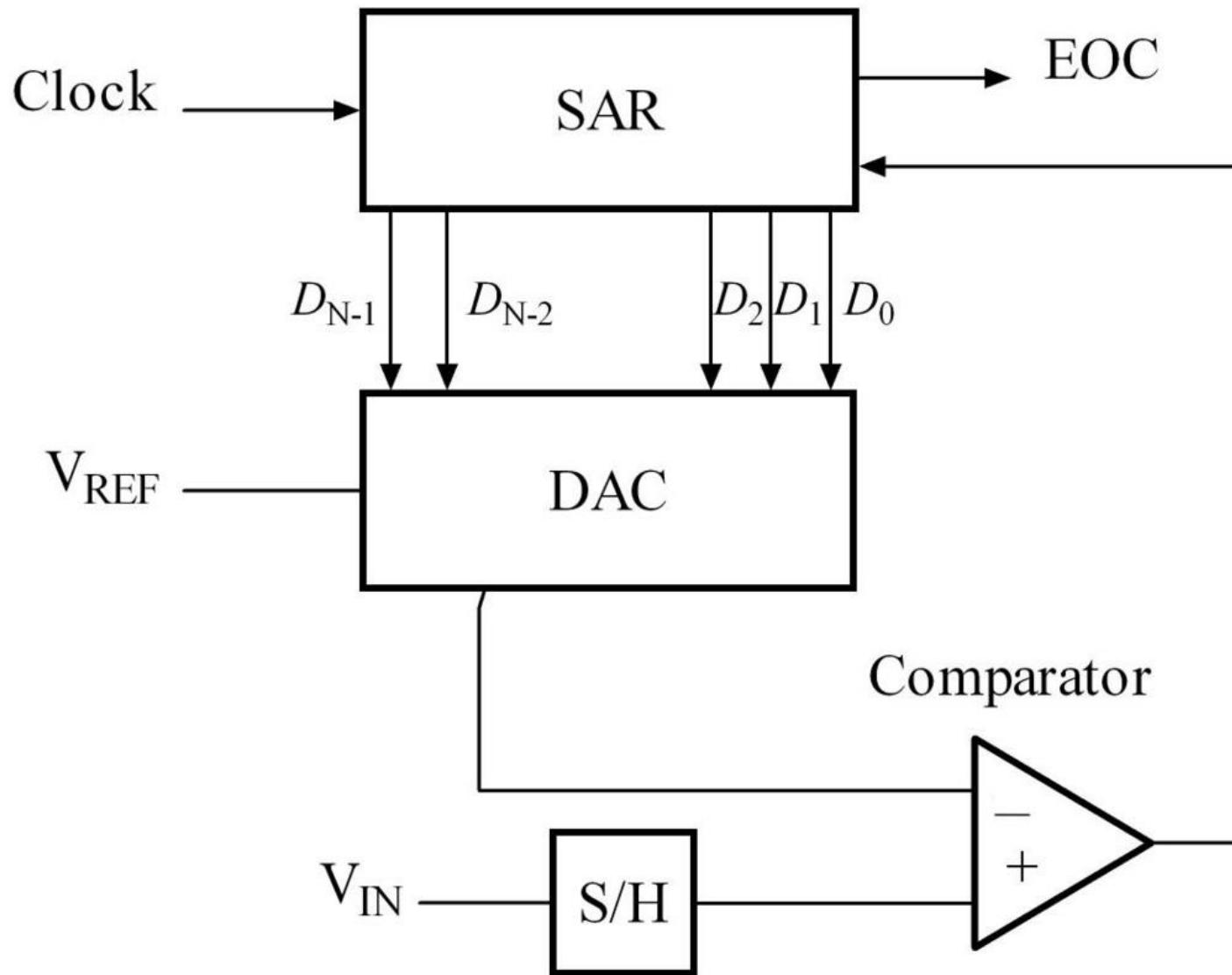
When analog voltage is 0.805 mV output decimal value is 1

When analog voltage is  $(0.805 \times 2 = 1.6)$  mV output decimal value is 2

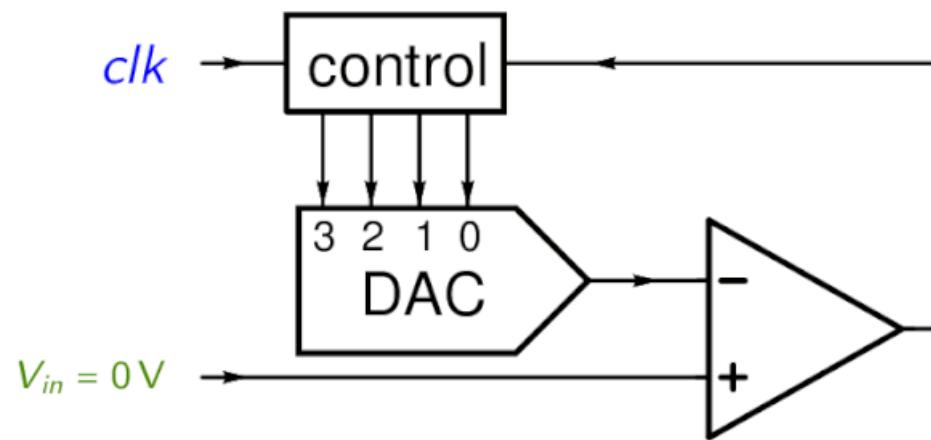
When analog voltage is  $(0.805 \times 3 = 2.4)$  mV output decimal value is 3

When analog voltage is  $(0.805 \times 4095 = 3.3 - 0.805)$  mV output decimal value is 4095





## Successive Approximation – example of a 4-bit ADC



Resolution:	
$5\text{ V} \times 1/2$	2.5000 V
$5\text{ V} \times 1/4$	1.2500 V
$5\text{ V} \times 1/8$	0.6250 V
$5\text{ V} \times 1/16$	0.3125 V
...	
$5\text{ V} \times 1/1024$	0.0049 V

Register	Description
ADCR	A/D COnrol Register: Used for Configuring the ADC
ADGDR	A/D Global Data Register: This register contains the ADC's DONE bit and the result of the most recent A/D conversion
ADINTEN	A/D Interrupt Enable Register
ADDR0 - ADDR7	A/D Channel Data Register: Contains the recent ADC value for respective channel
ADSTAT	A/D Status Register: Contains DONE & OVERRUN flag for all the ADC channels

## ADC Register Configuration

Now lets see how to configure the individual registers for ADC conversion.

ADCR									
31:28	27	26:24	23:22	21	20:17	16	15:8	7:0	
Reserved	EDGE	START	Reserved	PDN	Reserved	BURST	CLCKDIV	SEL	

## ADGDR ( ADC Global Data Register )

ADGDR					
31	27	26:24	23:16	15:4	3:0
DONE	OVERRUN	CHN	Reserved	RESULT	Reserved

### Bit 15:4 - RESULT

This field contains the 12bit A/D conversion value for the selected channel in **ADCR.SEL**

The vale for this register should be read oncve the conversion is completed ie DONE bit is set.

### Bit 26:24 - CHN : Channel

These bits contain the channel number for which the A/D conversion is done and the converted value is available in RESULT bits(e.g. 000 identifies channel 0, 011 channel 3...).

### Bit 27 - OVERRUN

This bit is set during the BURST mode where the previous conversion data is overwritten by the new A/D conversion value.

### Bit 31 - DONE

This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read and when the ADCR is written. If the ADCR is written while a conversion is still in progress, this bit is set and a new conversion is started.

- 4) **ADDR0 to ADDR7** – A/D Data registers : This register contains the result of the most recent conversion completed on the corresponding channel [0 to 7]. Its structure is same as ADGDR except Bits[26:24] are not used/reserved.
- 5) **ADSTAT** – A/D Status register : This register contains DONE and OVERRUN flags for all of the A/D channels along with A/D interrupt flag.
1. **Bits[7:0] – DONE[7 to 0]**: Here xth bit mirrors DONEx status flag from the result register for A/D channel x.
  2. **Bits[15:8] – OVERRUN[7 to 0]**: Even here the xth bit mirrors OVERRUNx status flag from the result register for A/D channel x
  3. **Bit 16 – ADINT**: This bit represents the A/D interrupt flag. It is 1 when any of the individual A/D channel DONE flags is asserted and enabled to contribute to the A/D interrupt via the ADINTEN(given below) register.
  4. Other bits are reserved.

# Analog To Digital Converter (ADC)

Pin	Type	Description
AD0.7 to AD0.0	Input	<b>Analog Inputs.</b> The ADC cell can measure the voltage on any of these input signals. Digital signals are disconnected from the ADC input pins when the ADC function is selected on that pin in the Pin Select register.

## ADCR - A/D Control Register

Bit	Symbol	Value	Description	
7:0	SEL		Selects which of the AD0.7:0 pins is (are) to be sampled and converted. For AD0, bit 0 selects Pin AD0.0, and bit 7 selects pin AD0.7. In software-controlled mode, only one of these bits should be 1. In hardware scan mode, any value containing 1 to 8 ones is allowed. All zeroes is equivalent to 0x01.	AD0.0-P0.23 FN 01 AD0.1-P0.24 FN 01 AD0.2-P0.25 FN 01 AD0.3-P0.26 FN 01 AD0.4-P0.30 FN 03 AD0.5-P0.31 FN 03 AD0.6-P0.3 FN 02 AD0.7-P0.2 FN 02
15:8	CLKDIV		The APB clock (PCLK_ADC0) is divided by (this value plus one) to produce the clock for the A/D converter, which should be less than or equal to 13 MHz. Typically, software should program the smallest value in this field that yields a clock of 13 MHz or slightly less, but in certain cases (such as a high-impedance analog source) a slower clock may be desirable.	
16	BURST	1	The AD converter does repeated conversions at up to 200 kHz, scanning (if necessary) through the pins selected by bits set to ones in the SEL field. The first conversion after the start corresponds to the least-significant 1 in the SEL field, then higher numbered 1-bits (pins) if applicable. Repeated conversions can be terminated by clearing this bit, but the conversion that's in progress when this bit is cleared will be completed.  <b>Remark:</b> START bits must be 000 when BURST = 1 or conversions will not start. If BURST is set to 1, the ADGINTEN bit in the AD0INTEN register ( <a href="#">Table 534</a> ) must be set to 0.	
0		0	Conversions are software controlled and require 65 clocks.	
20:17	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	

# Analog To Digital Converter (ADC)

## ADCR - A/D Control Register

21	PDN	1	The A/D converter is operational.
		0	The A/D converter is in power-down mode.
23:22	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	
26:24	START	When the BURST bit is 0, these bits control whether and when an A/D conversion is started:	
		000	No start (this value should be used when clearing PDN to 0).
		001	Start conversion now.
		010	Start conversion when the edge selected by bit 27 occurs on the P2.10 / EINT0 / NMI pin.
		011	Start conversion when the edge selected by bit 27 occurs on the P1.27 / CLKOUT / USB_OVRCRn / CAP0.1 pin.
		100	Start conversion when the edge selected by bit 27 occurs on MAT0.1. Note that this does not require that the MAT0.1 function appear on a device pin.
		101	Start conversion when the edge selected by bit 27 occurs on MAT0.3. Note that it is not possible to cause the MAT0.3 function to appear on a device pin.
		110	Start conversion when the edge selected by bit 27 occurs on MAT1.0. Note that this does not require that the MAT1.0 function appear on a device pin.
		111	Start conversion when the edge selected by bit 27 occurs on MAT1.1. Note that this does not require that the MAT1.1 function appear on a device pin.
27	EDGE	This bit is significant only when the START field contains 010-111. In these cases:	
		1	Start conversion on a falling edge on the selected CAP/MAT signal.
		0	Start conversion on a rising edge on the selected CAP/MAT signal.
31:28	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	

# Analog To Digital Converter (ADC)

## A/D Global Data Register (ADGDR) :

The A/D Global Data Register holds the result of the most recent A/D conversion that has completed, and also includes copies of the status flags that go with that conversion. Results of ADC conversion can be read in one of two ways. One is to use the A/D Global Data Register to read all data from the ADC. Another is to use the A/D Channel Data Registers.

Bit	Symbol	Description
3:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.
15:4	RESULT	When DONE is 1, this field contains a binary fraction representing the voltage on the AD0[n] pin selected by the SEL field, as it falls within the range of V <sub>REFP</sub> to V <sub>REFN</sub> . Zero in the field indicates that the voltage on the input pin was less than, equal to, or close to that on V <sub>REFN</sub> , while 0xFFFF indicates that the voltage on the input was close to, equal to, or greater than that on V <sub>REFP</sub> .
23:16	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.
26:24	CHN	These bits contain the channel from which the RESULT bits were converted (e.g. 000 identifies channel 0, 001 channel 1...).
29:27	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.
30	OVERRUN	This bit is 1 in burst mode if the results of one or more conversions was (were) lost and overwritten before the conversion that produced the result in the RESULT bits. This bit is cleared by reading this register.
31	DONE	This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read.

# Analog To Digital Converter (ADC)

## A/D Data Registers (ADDR0 to ADDR7)

The A/D Data Registers hold the result of the last conversion for each A/D channel, when an A/D conversion is complete. They also include the flags that indicate when a conversion has been completed and when a conversion overrun has occurred.

---

Bit	Symbol	Description
3:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.
15:4	RESULT	When DONE is 1, this field contains a binary fraction representing the voltage on the AD0[n] pin, as it falls within the range of $V_{REFP}$ to $V_{REFN}$ . Zero in the field indicates that the voltage on the input pin was less than, equal to, or close to that on $V_{REFN}$ , while 0xFFFF indicates that the voltage on the input was close to, equal to, or greater than that on $V_{REFP}$ .
29:16	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.
30	OVERRUN	This bit is 1 in burst mode if the results of one or more conversions was (were) lost and overwritten before the conversion that produced the result in the RESULT bits. This bit is cleared by reading this register.
31	DONE	This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read.

# Analog To Digital Converter (ADC)

**A/D Interrupt Enable register (ADINTEN) :** This register allows control over which A/D channels generate an interrupt when a conversion is complete.

Bit	Symbol	Value	Description
0	ADINTEN0	0	Completion of a conversion on ADC channel 0 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 0 will generate an interrupt.
1	ADINTEN1	0	Completion of a conversion on ADC channel 1 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 1 will generate an interrupt.
2	ADINTEN2	0	Completion of a conversion on ADC channel 2 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 2 will generate an interrupt.
3	ADINTEN3	0	Completion of a conversion on ADC channel 3 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 3 will generate an interrupt.
4	ADINTEN4	0	Completion of a conversion on ADC channel 4 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 4 will generate an interrupt.
5	ADINTEN5	0	Completion of a conversion on ADC channel 5 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 5 will generate an interrupt.
6	ADINTEN6	0	Completion of a conversion on ADC channel 6 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 6 will generate an interrupt.
7	ADINTEN7	0	Completion of a conversion on ADC channel 7 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 7 will generate an interrupt.
8	ADGINTEN	0	Only the individual ADC channels enabled by ADINTEN7:0 will generate interrupts.  <b>Remark:</b> This bit must be set to 0 in burst mode (BURST = 1 in the AD0CR register).
		1	Only the global DONE flag in ADDR is enabled to generate an interrupt.
31:17 -			Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.

## A/D Status register (ADSTAT) :

The A/D Status register allows checking the status of all A/D channels simultaneously. The DONE and OVERRUN flags appearing in the ADDRn register for each A/D channel are mirrored in ADSTAT. The interrupt flag (the logical OR of all DONE flags) is also found in ADSTAT.

---

Bit	Symbol	Description
0	DONE0	This bit mirrors the DONE status flag from the result register for A/D channel 0.
1	DONE1	This bit mirrors the DONE status flag from the result register for A/D channel 1.
2	DONE2	This bit mirrors the DONE status flag from the result register for A/D channel 2.
3	DONE3	This bit mirrors the DONE status flag from the result register for A/D channel 3.
4	DONE4	This bit mirrors the DONE status flag from the result register for A/D channel 4.
5	DONE5	This bit mirrors the DONE status flag from the result register for A/D channel 5.
6	DONE6	This bit mirrors the DONE status flag from the result register for A/D channel 6.
7	DONE7	This bit mirrors the DONE status flag from the result register for A/D channel 7.
8	OVERRUN0	This bit mirrors the OVERRUN status flag from the result register for A/D channel 0.
9	OVERRUN1	This bit mirrors the OVERRUN status flag from the result register for A/D channel 1.
10	OVERRUN2	This bit mirrors the OVERRUN status flag from the result register for A/D channel 2.
11	OVERRUN3	This bit mirrors the OVERRUN status flag from the result register for A/D channel 3.
12	OVERRUN4	This bit mirrors the OVERRUN status flag from the result register for A/D channel 4.
13	OVERRUN5	This bit mirrors the OVERRUN status flag from the result register for A/D channel 5.
14	OVERRUN6	This bit mirrors the OVERRUN status flag from the result register for A/D channel 6.
15	OVERRUN7	This bit mirrors the OVERRUN status flag from the result register for A/D channel 7.
16	ADINT	This bit is the A/D interrupt flag. It is one when any of the individual A/D channel Done flags is asserted and enabled to contribute to the A/D interrupt via the ADINTEN register.

# Steps for Configuring ADC

Below are the steps for configuring the LPC1768 ADC.

1. Configure the GPIO pin for ADC function using PINSEL register.
2. Enable the Clock to ADC module.
3. Deselect all the channels and Power on the internal ADC module by setting ADCR.PDN bit.
4. Select the Particular channel for A/D conversion by setting the corresponding bits in ADCR.SEL
5. Set the ADCR.START bit for starting the A/D conversion for selected channel.
6. Wait for the conversion to complete, ADGR.DONE bit will be set once conversion is over.
7. Read the 12-bit A/D value from ADGR.RESULT.
8. Use it for further processing or just display on LCD.

## ADC software mode for 2-channel concurrent conversion

```
#include<LPC17xx.h>
#include<stdio.h>
int main(void)
{
    unsigned long temp4, temp5;
    unsigned int i;

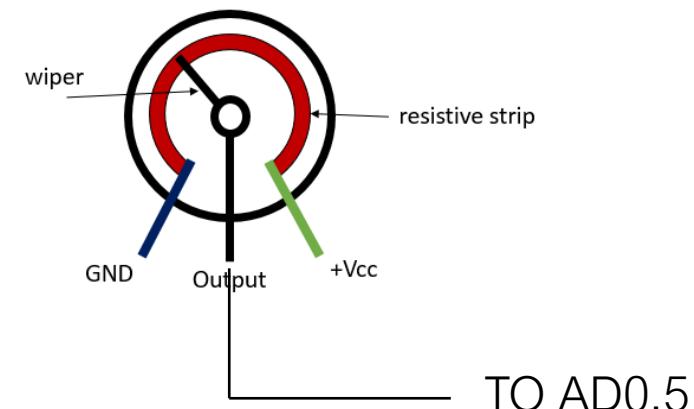
    SystemInit();
    SystemCoreClockUpdate();
    LPC_PINCON->PINSEL3 = (3<<28) | (3<<30);          //P1.30 as AD0.4 and P1.31 as AD0.5
    LPC_ADC->ADINTEN = 0;
    while(1)
    {
        LPC_ADC->ADCR = (1<<4)|(1<<21)|(1<<24);//           //ADC0.4, start conversion and operational
        for(i=0;i<2000;i++);
        while(((temp4=LPC_ADC->ADDR4) & (1<<31)) == 0); //wait till 'done' bit is 1, indicates conversion complete
        temp4 = LPC_ADC->ADDR4;
        temp4 >>= 4;
        temp4 &= 0x00000FFF;                                //12 bit ADC

        LPC_ADC->ADCR = (1<<5)|(1<<21)|(1<<24);//           //ADC0.5, start conversion and operational
        for(i=0;i<2000;i++);                            //delay for conversion
        while(((temp5=LPC_ADC->ADDR5) & (1<<31)) == 0); //wait till 'done' bit is 1, indicates conversion complete
        temp5 = LPC_ADC->ADDR5;
        temp5 >>= 4;
        temp5 &= 0x00000FFF;                                //12 bit ADC
    //Now you can use temp4 and temp5 for further processing based on your requirement
    }
}
```

## Input Analog voltage and display its digital equivalent on LCD

```
include<LPC17xx.h>
#include<stdio.h>
#define Ref_Vtg          3.300
#define Full_Scale 0xFFFF//12 bit ADC
unsigned long int init_command[] = {0x30,0x30,0x30,0x20,0x28,0x0c,0x06,0x01,0x80};
int main(void)
{
    flag1 =0;//Command
    unsigned long adc_temp;
    unsigned int i;
    float in_vtg;
    unsigned char vtg[7], dval[7];
    unsigned char Msg3[] = {"ANALOG IP:"};
    unsigned char Msg4[] = {"ADC OUTPUT:"};
    SystemInit();
    SystemCoreClockUpdate();
    lcd_init();//Initialize LCD
    LPC_PINCON->PINSEL3 |= 3<<30;           //P1.31 as AD0.5
    LPC_SC->PCONP |= (1<<12);//enable the peripheral ADC
    for (i=0; i<9;i++)
    {
        temp1 = init_command[i];
        lcd_write(); //send Init commands to LCD  }
    flag1=1;//Data
    i=0;
    while (Msg3[i++] != '\0')
    {
        temp1 = Msg3[i];
        lcd_write();//Send data bytes}
```

ANALOG INPUT 1.1V  
ADC OUTPUT 555



```

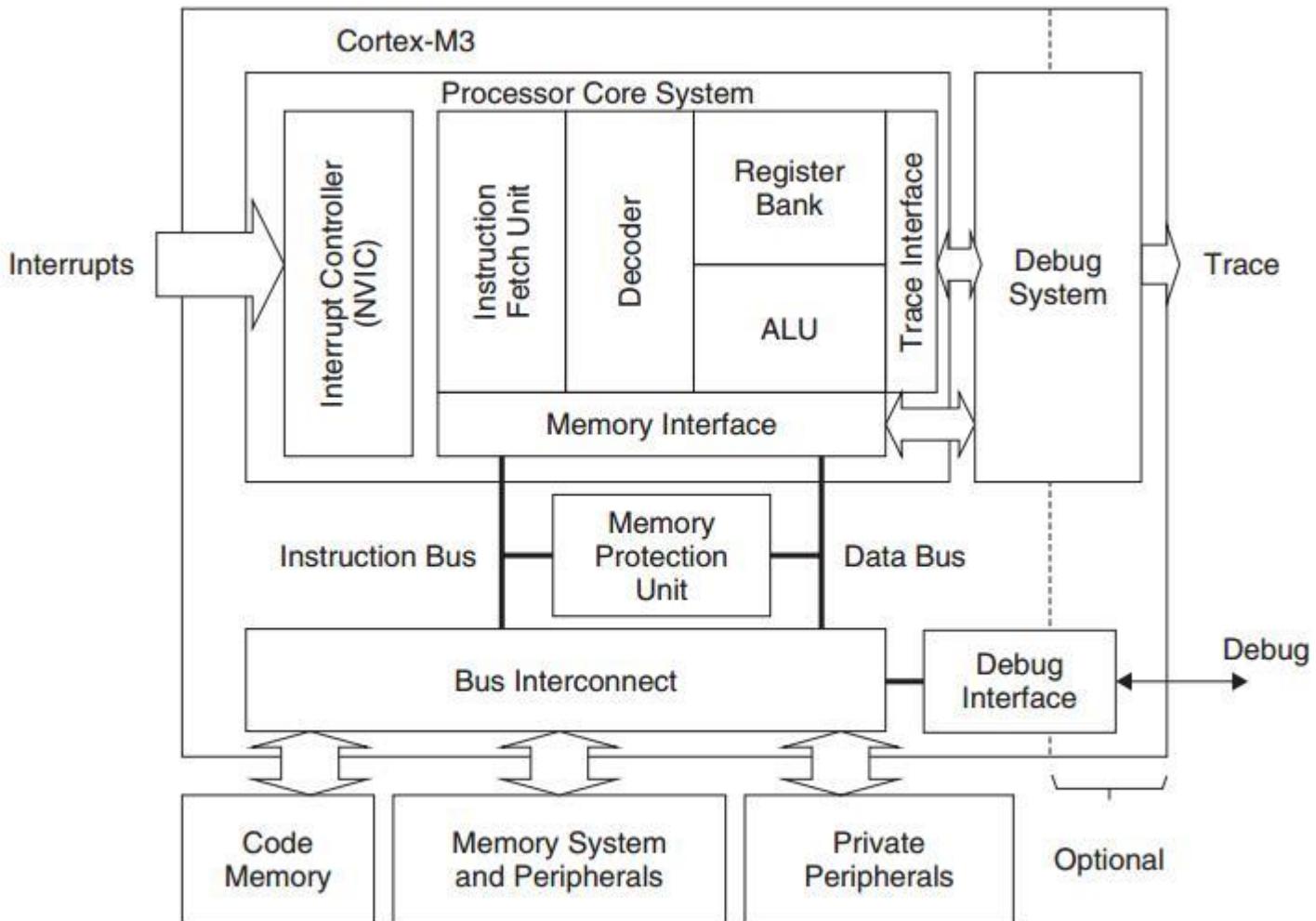
flag1=0; //Command
    temp1 = 0xC0;//Cursor at beginning of second line
    lcd_write();
    flag1=1;
    i =0;
    while (Msg4[i++] != '\0')
    {
        temp1 = Msg4[i];
        lcd_write();//Send data bytes
    }
while(1)
{
    LPC_ADC->ADCR = (1<<5)|(1<<21)|(1<<24);//ADC0.5, start conversion and operational
    while(((adc_temp=LPC_ADC->ADGDR) & (1<<31)) == 0);
    adc_temp = LPC_ADC->ADGDR;
    adc_temp >>= 4;
    adc_temp &= 0x00000FFF;           //12 bit ADC
    in_vtg = (((float)adc_temp * (float)Ref_Vtg))/((float)Full_Scale);           //calculating input analog voltage
    sprintf(vtg,"%3.2fV",in_vtg);           //convert the readings into string to display on LCD
    sprintf(dval,"%x",adc_temp);
    flag1=0;;
    temp1 = 0x8A;
    lcd_write();
    flag1=1;
    i =0;
    while (vtg[i++] != '\0')
    {
        temp1 = vtg[i];
        lcd_write();//Send data bytes
    }
}

```

---

```
flag1=0;
temp1 = 0xCB;
lcd_write();
flag1=1;
i =0;
while (dval[i++] != '\0')
{
    temp1 = dval[i];
    lcd_write();//Send data bytes
}
for(i=0;i<7;i++)
vtg[i] = dval[i] = 0;
}
}
```

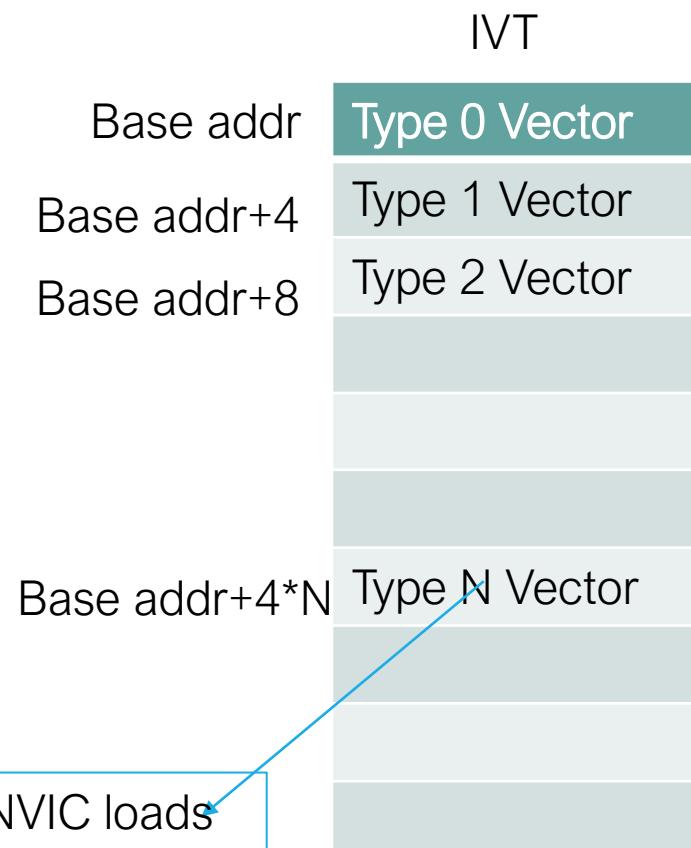
# Nested Vectored Interrupt Controller(NVIC)



## NestedVectored Interrupt Controller(NVIC)

- Controls system exceptions and peripheral interrupts
- In the LPC176x, the NVIC supports 35 vectored interrupts
- Each peripheral device may have one or more interrupt lines to the Vectored Interrupt Controller.
- Interrupt numbers relate to where entries are stored in the Interrupt vector table.
- Interrupt Vector – Is the address of Interrupt Service Subroutine (ISR)
- Interrupt vector of Interrupt Type N is stored at an offset  $N*4$  from the base address of Interrupt Vector Table(IVT)
- If the peripheral device is enabled to generate Interrupt when some event

When Interrupt occurs, NVIC loads vector to PC and executes the ISR to provide the service to peripheral



## Nested Vectored Interrupt Controller(NVIC)

---

- If the peripheral device is enabled to generate Interrupt when some event occurs, the INTR request is sent to NVIC
- If NVIC is enabled to service the INTR request from the peripheral, it services the INTR request by executing ISR pertaining to the peripheral.( i.e Save the return address, Get the Interrupt Vector from IVT and load that address to PC. Upon completion of ISR execution resumes the calling function )

## Nested Vectored Interrupt Controller(NVIC)

---

- In case of Timer, there are 6 INTR enable flags (4 in MCR and 2 in CCR. i.e 4 Match events and 2 Capture events can generate the Interrupt when the event occurs)
- When the event occurs the corresponding bit is set automatically in the IR register. This indicates the NVIC about the event
- If the NVIC is enabled to service the Timer Interrupt, it executes the corresponding ISR and gives the desired service to the Timer.
- In the ISR, clear the corresponding bit, by writing back 1.

# Timer/Counter Interrupt Programming

## Interrupt Register (IR)

---

The Interrupt Register consists of 4 bits for the match interrupts and 2 bits for the capture interrupts. If an interrupt is generated then the corresponding bit in the IR will be high. Otherwise, the bit will be low.

Writing a logic one to the corresponding IR bit will reset the interrupt. Writing a zero has no effect.

Bit	Symbol	Description
0	MR0 Interrupt	Interrupt flag for match channel 0.
1	MR1 Interrupt	Interrupt flag for match channel 1.
2	MR2 Interrupt	Interrupt flag for match channel 2.
3	MR3 Interrupt	Interrupt flag for match channel 3.
4	CR0 Interrupt	Interrupt flag for capture channel 0 event.
5	CR1 Interrupt	Interrupt flag for capture channel 1 event.

# Timer/Counter Interrupt Programming

Toggle LED connected to p0.2 every second while displaying the status of switch connected to P1.0 on the LED connected to P2.0

```
#include<stdio.h>
#include<LPC17xx.h>
unsigned int ticks=0,x;
void TIMER0_IRQHandler(void)
{
LPC_TIM0->IR = 1;
    ticks++;
    if(ticks==1000)
    {
        ticks=0;
        LPC_GPIO0->FIOPIN=~(LPC_GPIO0->FIOPIN & 0x00000004);
    }
}
void init_timer0(void)
{
    LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
    LPC_TIM0->CTCR =0x00;//Timer
    LPC_TIM0->MR0 = 2999; // For 1ms
    LPC_TIM0->EMR = 0X00;//Do nothing for EM0
    LPC_TIM0->PR = 0;
    LPC_TIM0->MCR = 0x00000003; //Reset TC upon Match-0 and generate INTR
    LPC_TIM0->TCR = 0x00000001; // Timer0 Enable

    return;
```

# Timer/Counter Interrupt Programming

---

```
int main(void)
{
    LPC_GPIO0->FIODIR=0x00000004;
    LPC_GPIO2->FIODIR=0x00000001;
    init_timer0();
    NVIC_EnableIRQ(TIMER0_IRQn);//timer 0 intr enabled in NVIC
    while(1)
    {
        LPC_GPIO2->FIOPIN=(LPC_GPIO1->FIOPIN & 0x01);
    }
}
```

# Timer/Counter Interrupt Programming

Toggle P0.2 whenever counter value reaches 3. I.e for every 4 edges using the counter interrupt.

```
#include<stdio.h>
#include<LPC17xx.h>
void TIMER0_IRQHandler(void)
{
    LPC_TIM0->IR = 1;
    LPC_GPIO0->FIOPIN=~(LPC_GPIO0->FIOPIN & 0x00000004);
}
void init_timer0(void)
{
    LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
    LPC_TIM0->CTCR = 0x05; // Counter at +ve edge of CAP0.1
    LPC_TIM0->MR0 = 3;
    LPC_TIM0->EMR = 0X00;
    LPC_TIM0->PR = 0;
    LPC_TIM0->MCR = 0x00000003;
    LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    return;
}

int main(void)
{
    LPC_GPIO0->FIODIR=0x00000004;
    LPC_PINCON->PINSEL3 |=((3<<22)|(3<<24));
    init_timer0();
    NVIC_EnableIRQ(TIMER0_IRQn);
    while(1);
}
```

43	P1.27	CLKOUT	USB OVRCR	CAP0.1
44	P1.28	MC2A	PCAP1.0	MATD.0

# Timer/Counter Interrupt Programming

Timer interrupt for rectangular waveform generation (1.5 second HIGH and 0.5 second LOW)

```
include<stdio.h>
#include<LPC17xx.h>
unsigned char flag=1;
void TIMERO_IRQHandler(void)
{
    LPC_TIM0->IR = 1;

    if(flag)
    {
        flag=0;
        LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
        LPC_GPIO0->FIOCLR=0x00000004;
        LPC_TIM0->MR0 = 500;
        LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    }
    else
    {
        flag=1;
        LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
        LPC_GPIO0->FIOSET=0x00000004;
        LPC_TIM0->MR0 = 1500;
        LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    }
}
```

# Timer/Counter Interrupt Programming

---

```
void init_timer0(void)
{
    LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
    LPC_TIM0->CTCR =0x00;
    LPC_TIM0->MR0 = 1500;
    LPC_TIM0->EMR = 0X00;
    LPC_TIM0->PR = 3000;
    LPC_TIM0->MCR = 0x00000005;
    LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    LPC_GPIO0->FIOSET=0x00000004;
    return;
}
int main(void)
{
    LPC_GPIO0->FIODIR=0x00000004;
    init_timer0();
    NVIC_EnableIRQ(TIMERO_IRQn);
    while(1);
}
```

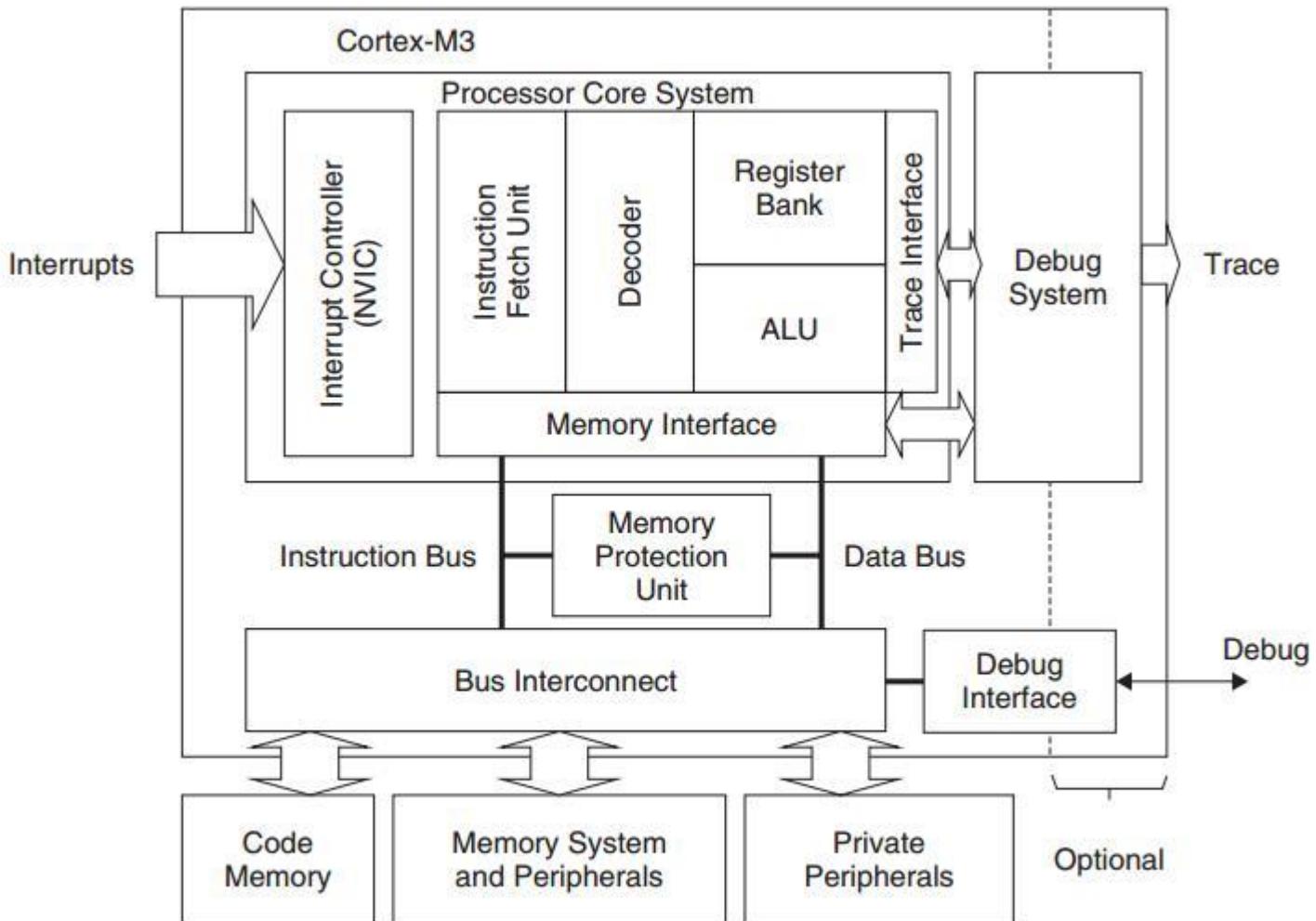


# Embedded System and Design

---

**DR MANOJ TOLANI (PHD-IIIT ALLAHABAD)**  
**ASSISTANT PROFESSOR (DEPARTMENT OF ICT)**

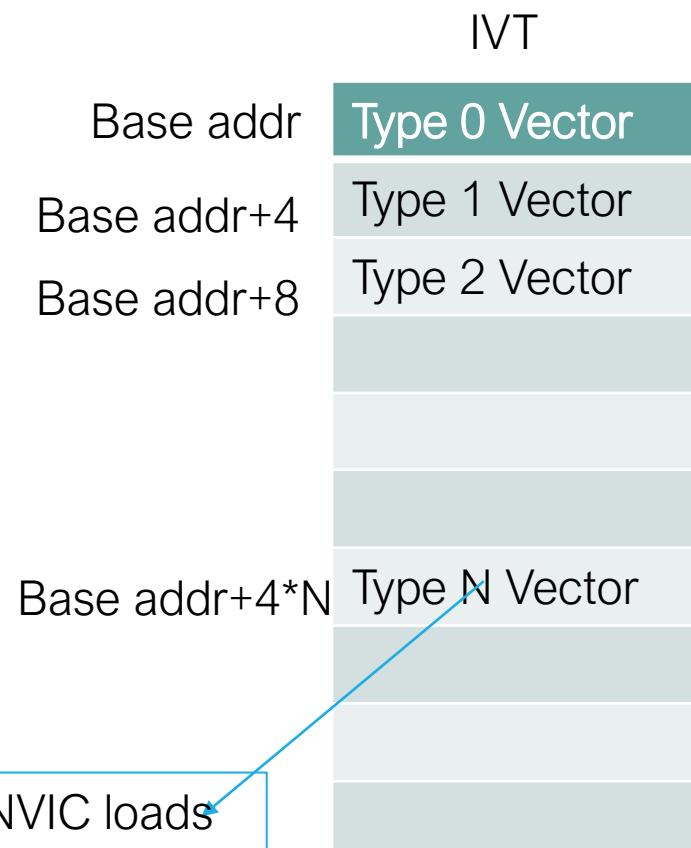
# Nested Vectored Interrupt Controller(NVIC)



## NestedVectored Interrupt Controller(NVIC)

- Controls system exceptions and peripheral interrupts
- In the LPC176x, the NVIC supports 35 vectored interrupts
- Each peripheral device may have one or more interrupt lines to the Vectored Interrupt Controller.
- Interrupt numbers relate to where entries are stored in the Interrupt vector table.
- Interrupt Vector – Is the address of Interrupt Service Subroutine (ISR)
- Interrupt vector of Interrupt Type N is stored at an offset  $N*4$  from the base address of Interrupt Vector Table(IVT)
- If the peripheral device is enabled to generate Interrupt when some event

When Interrupt occurs, NVIC loads vector to PC and executes the ISR to provide the service to peripheral



## Nested Vectored Interrupt Controller(NVIC)

---

- If the peripheral device is enabled to generate Interrupt when some event occurs, the INTR request is sent to NVIC
- If NVIC is enabled to service the INTR request from the peripheral, it services the INTR request by executing ISR pertaining to the peripheral.( i.e Save the return address, Get the Interrupt Vector from IVT and load that address to PC. Upon completion of ISR execution resumes the calling function )

## Nested Vectored Interrupt Controller(NVIC)

---

- In case of Timer, there are 6 INTR enable flags (4 in MCR and 2 in CCR. i.e 4 Match events and 2 Capture events can generate the Interrupt when the event occurs)
- When the event occurs the corresponding bit is set automatically in the IR register. This indicates the NVIC about the event
- If the NVIC is enabled to service the Timer Interrupt, it executes the corresponding ISR and gives the desired service to the Timer.
- In the ISR, clear the corresponding bit, by writing back 1.

# Timer/Counter Interrupt Programming

## Interrupt Register (IR)

---

The Interrupt Register consists of 4 bits for the match interrupts and 2 bits for the capture interrupts. If an interrupt is generated then the corresponding bit in the IR will be high. Otherwise, the bit will be low.

Writing a logic one to the corresponding IR bit will reset the interrupt. Writing a zero has no effect.

Bit	Symbol	Description
0	MR0 Interrupt	Interrupt flag for match channel 0.
1	MR1 Interrupt	Interrupt flag for match channel 1.
2	MR2 Interrupt	Interrupt flag for match channel 2.
3	MR3 Interrupt	Interrupt flag for match channel 3.
4	CR0 Interrupt	Interrupt flag for capture channel 0 event.
5	CR1 Interrupt	Interrupt flag for capture channel 1 event.

# Timer/Counter Interrupt Programming

Toggle LED connected to p0.2 every second while displaying the status of switch connected to P1.0 on the LED connected to P2.0

```
#include<stdio.h>
#include<LPC17xx.h>
unsigned int ticks=0,x;
void TIMER0_IRQHandler(void)
{
LPC_TIM0->IR = 1;
    ticks++;
    if(ticks==1000)
    {
        ticks=0;
        LPC_GPIO0->FIOPIN=~(LPC_GPIO0->FIOPIN & 0x00000004);
    }
}
void init_timer0(void)
{
    LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
    LPC_TIM0->CTCR =0x00;//Timer
    LPC_TIM0->MR0 = 2999; // For 1ms
    LPC_TIM0->EMR = 0X00;//Do nothing for EM0
    LPC_TIM0->PR = 0;
    LPC_TIM0->MCR = 0x00000003; //Reset TC upon Match-0 and generate INTR
    LPC_TIM0->TCR = 0x00000001; // Timer0 Enable

    return;
```

# Timer/Counter Interrupt Programming

---

```
int main(void)
{
    LPC_GPIO0->FIODIR=0x00000004;
    LPC_GPIO2->FIODIR=0x00000001;
    init_timer0();
    NVIC_EnableIRQ(TIMER0_IRQn);//timer 0 intr enabled in NVIC
    while(1)
    {
        LPC_GPIO2->FIOPIN=(LPC_GPIO1->FIOPIN & 0x01);
    }
}
```

# Timer/Counter Interrupt Programming

Toggle P0.2 whenever counter value reaches 3. Counter is connected to the capture pin CAP0.1. i.e for every 4 edges using the counter interrupt.

```
#include<stdio.h>
#include<LPC17xx.h>
void TIMER0_IRQHandler(void)
{
    LPC_TIM0->IR = 1;
    LPC_GPIO0->FIOPIN=~(LPC_GPIO0->FIOPIN & 0x00000004);
}
void init_timer0(void)
{
    LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
    LPC_TIM0->CTCR =0x05; // Counter at +ve edge of CAP0.1
    LPC_TIM0->MRO = 3;
    LPC_TIM0->EMR = 0X00;
    LPC_TIM0->PR = 0;
    LPC_TIM0->MCR = 0x00000003;
    LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    return;
}
int main(void)
{
    LPC_GPIO0->FIODIR=0x00000004;
    LPC_PINCON->PINSEL3 |=((3<<22)|(3<<24));
    init_timer0();
    NVIC_EnableIRQ(TIMER0_IRQn);
    while(1);
}
```

43	P1.27	CLKOUT	USB_OVRCR	CAP0.1
44	P1.28	MC2A	PCAP1.0	MAT0.0

# Timer/Counter Interrupt Programming

Timer interrupt for rectangular waveform generation (1.5 second HIGH and 0.5 second LOW)

```
include<stdio.h>
#include<LPC17xx.h>
unsigned char flag=1;
void TIMERO_IRQHandler(void)
{
    LPC_TIM0->IR = 1;

    if(flag)
    {
        flag=0;
        LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
        LPC_GPIO0->FIOCLR=0x00000004;
        LPC_TIM0->MR0 = 500;
        LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    }
    else
    {
        flag=1;
        LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
        LPC_GPIO0->FIOSET=0x00000004;
        LPC_TIM0->MR0 = 1500;
        LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    }
}
```

# Timer/Counter Interrupt Programming

---

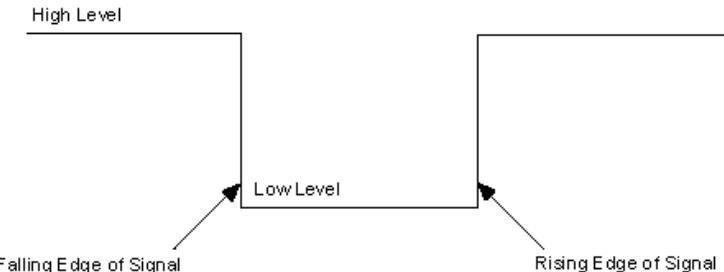
```
void init_timer0(void)
{
    LPC_TIM0->TCR = 0x00000002; // Timer0 Reset
    LPC_TIM0->CTCR =0x00;
    LPC_TIM0->MR0 = 1500;
    LPC_TIM0->EMR = 0X00;
    LPC_TIM0->PR = 3000;
    LPC_TIM0->MCR = 0x00000005;
    LPC_TIM0->TCR = 0x00000001; // Timer0 Enable
    LPC_GPIO0->FIOSET=0x00000004;
    return;
}
int main(void)
{
    LPC_GPIO0->FIODIR=0x00000004;
    init_timer0();
    NVIC_EnableIRQ(TIMERO_IRQn);
    while(1);
}
```

# External Hardware Interrupts

System Control Block of ARM has SFRs to handle External Hardware Interrupts.

- Level Triggered- Level 0 or Level 1 triggered
- Edge triggered – Rising Edge or Falling Edge

LPC1768 has four external interrupts EINT0-EINT3



Port Pin	PINSEL_FUNC_0	PINSEL_FUNC_1
P2.10	GPIO	EINT0
P2.11	GPIO	EINT1
P2_12	GPIO	EINT2
P2.13	GPIO	EINT3

# External Hardware Interrupts

---

## EINT Registers

Register	Description
PINSELx	To configure the pins as External Interrupts
EXTINT	External Interrupt Flag Register contains interrupt flags for EINT0,EINT1, EINT2 & EINT3.
EXTMODE	External Interrupt Mode register(Level/Edge Triggered)
EXTPOLAR	External Interrupt Polarity(Falling/Rising Edge, Active Low/High)

# External Hardware Interrupts

EXTINT				
31:4	3	2	1	0
RESERVED	EINT3	EINT2	EINT1	EINT0

**EINTx:** Bits will be set whenever the interrupt is detected on the particular interrupt pin.  
If the interrupts are enabled then the control goes to ISR.

**Writing one to specific bit will clear the corresponding interrupt.**

EXTMODE				
31:4	3	2	1	0
RESERVED	EXTMODE3	EXTMODE2	EXTMODE1	EXTMODE0

**EXTMODEx:** These bits are used to select whether the EINTx pin is level or edge Triggered

0: EINTx is Level Triggered.

1: EINTx is Edge Triggered.

# External Hardware Interrupts

EXTPOLAR				
31:4	3	2	1	0
RESERVED	EXTPOLAR3	EXTPOLAR2	EXTPOLAR1	EXTPOLAR0

**EXTPOLARx:** These bits are used to select polarity(LOW/HIGH, FALLING/RISING) of the EINTx interrupt depending on the EXTMODE register.

0: EINTx is Active Low or Falling Edge (depending on EXTMODEx).

1: EINTx is Active High or Rising Edge (depending on EXTMODEx).

EXTMODEx	EXTPOLARx	EINTx
0	0	Level 0
0	1	Level 1
1	0	Falling Edge
1	1	Rising Edge

# External Hardware Interrupts

---

## Steps to Configure External Hardware Interrupts

- Configure the pins as external interrupts in PINSELx register.
- Configure the EINTx as Edge/Level triggered in EXTMODE register.
- Select the polarity(Falling/Rising Edge, Active Low/High) of the interrupt in EXTPOLAR register.
- Finally enable the interrupts by calling `NVIC_EnableIRQ(EINTx_IRQHandler)`
- Clear the interrupt in EXTINT after entering ISR.

# External Hardware Interrupts

**Toggle LED connected to P1.23 at each negative edge of the input applied at P2.12 (EINT2, Function-01)**

---

# External Hardware Interrupts

**Toggle LED connected to P1.23 at each negative edge of the input applied at P2.12 (EINT2, Function-01)**

---

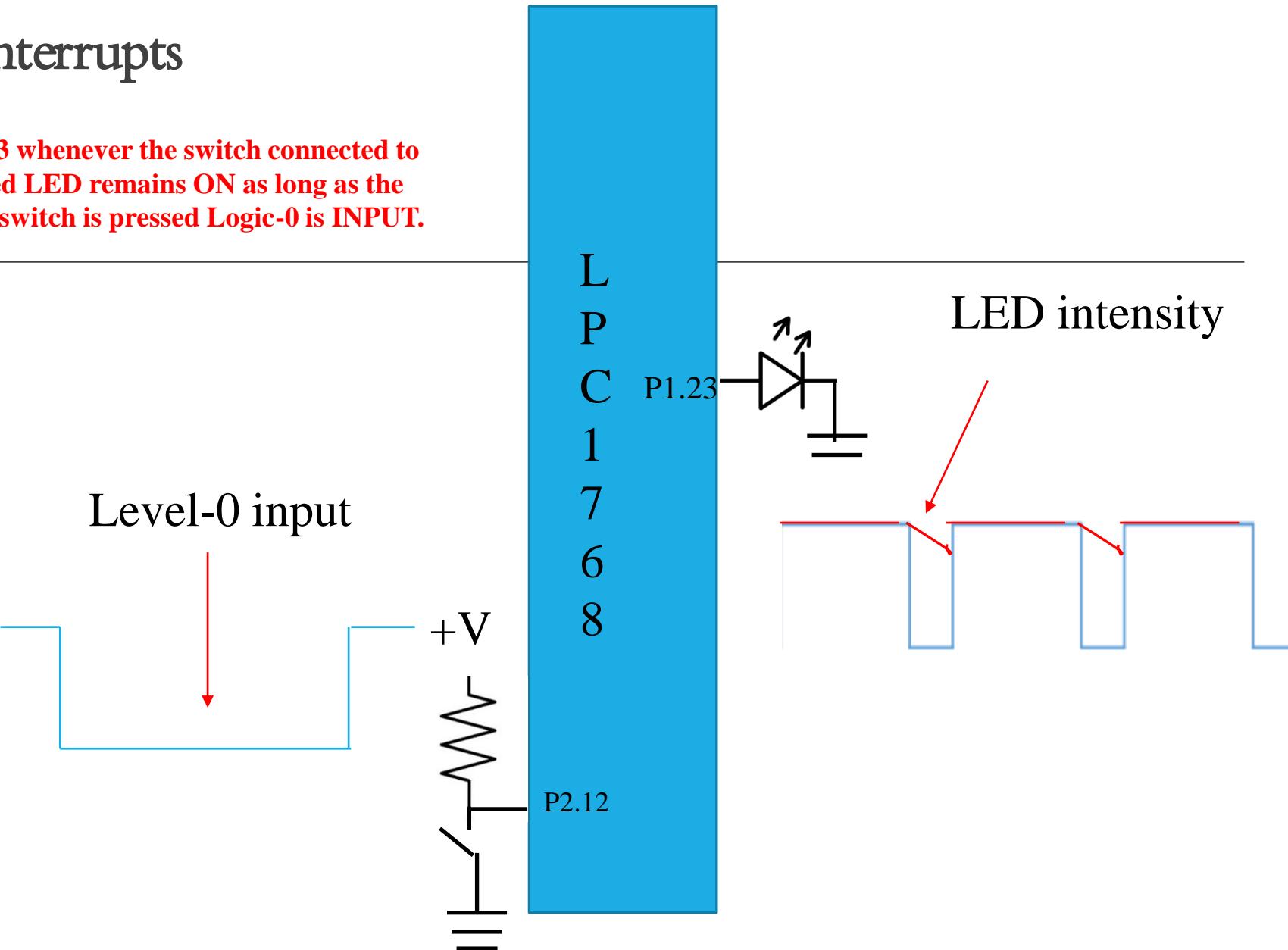
```
include<LPC17xx.h>
void EINT2_IRQHandler(void);
int main(void)
{
    SystemInit();
    SystemCoreClockUpdate();

    LPC_PINCON->PINSEL4 |= (1<<24);          //P2.12 as EINT2 i.e FUNCTION-01
    LPC_GPIO1->FIODIR = 0x00800000;           //P1.23 is assigned output
    LPC_SC->EXTMODE = 0x00000004;            //EINT2 is initiated as edge sensitive, 0 for level
    LPC_SC->EXTPOLAR = 0x00000000;           //EINT2 is falling edge sensitive, 1 for rising edge
    NVIC_EnableIRQ(EINT2_IRQn);
    while(1);
}

void EINT2_IRQHandler(void)
{
    LPC_SC->EXTINT = 0x00000004; //clears the interrupt
    LPC_GPIO1->FIOPIN = ~LPC_GPIO1->FIOPIN ;
}
```

# External Hardware Interrupts

Turn ON the LED connected to P1.23 whenever the switch connected to P2.12 (EINT2, Function-01) is pressed LED remains ON as long as the switch is pressed (Assume, when the switch is pressed Logic-0 is INPUT).



# External Hardware Interrupts

Turn ON the LED connected to P1.23 whenever the switch connected to P2.12 (EINT2, Function-01) is pressed LED remains ON as long as the switch is pressed (Assume, when the switch is pressed Logic-0 is INPUT).

---

```
#include<LPC17xx.h>
void EINT2_IRQHandler(void);
int main(void)
{
    LPC_PINCON->PINSEL4 |= (1<<24);           //P2.12 as EINT2 i.e FUNCTION-01
    LPC_GPIO1->FIODIR = 0x00800000;            //P1.23 is assigned output
    LPC_SC->EXTMODE = 0x00000000;              //EINT2 as level-0 sensitive
    LPC_SC->EXTPOLAR = 0x00000000;
    NVIC_EnableIRQ(EINT2_IRQn);
    while(1);

}
void EINT2_IRQHandler(void)
{
    LPC_SC->EXTINT = 0x00000004; //clear the interrupt
    LPC_GPIO1->FIOSET = 1<<23; //LED ON
    for (i=0;i<255;i++);
    LPC_GPIO1->FIOCLR = 1<<23; LED OFF
}
```

**Turn ON the LED connected to P1.23 whenever the switch connected to P2.12 (EINT2, Function-01) is pressed  
LED remains ON as long as the switch is pressed (Assume, when the switch is pressed Logic-0 is INPUT. In  
addition, generate a square wave of time period 4 second on pin P0.4.**

# Pulse Width Modulation (PWM)

PWM is one of the commonly used techniques to control the amount of power delivered through a pin. The PWM technique allows you to control the brightness of an LED, speed of a Motor, position of a Servo etc.

---

**The width of the Pulse i.e. the duration for which the pulse stays ON in a period is varied.  
Hence, it is called Pulse Width Modulation.**

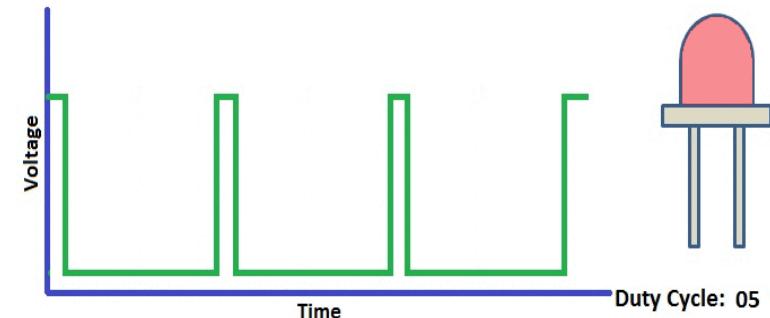
The Period of a PWM Cycle is the sum of duration for which the Pulse is HIGH and the duration for which the Pulse is LOW. This is usually represented by  $T_{ON}$  and  $T_{OFF}$ . So, Period of PWM =  $T_{ON} + T_{OFF}$ .

An important parameter of a PWM Signal is its Duty cycle. Duty cycle is the ratio of duration for which Pulse is HIGH to the total period of the PWM Signal.

$$\text{Duty cycle} = \frac{T_{ON}}{T_{ON} + T_{OFF}}$$

Duty cycle can also be represented as percentage.

$$\text{Duty cycle \%} = \frac{T_{ON}}{T_{ON} + T_{OFF}} * 100$$



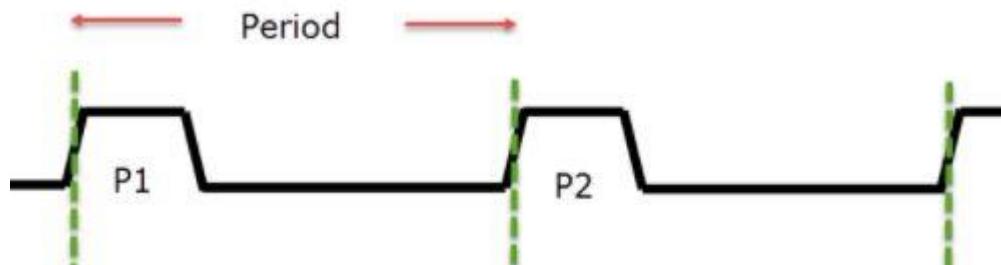
# Pulse Width Modulation (PWM)

Two types of PWM

:

- Single Edge PWM
  - Double Edge PWM
- 

In **Single Edge PWM**, the Pulse starts at the beginning of the period



In **Double Edge PWM**, both the edges can be modulated and hence, the pulse is placed anywhere in the period. Double Edge PWM is generally used in multi-phase motor control applications.



# Pulse Width Modulation (PWM)

Duty cycle of the PWM determines the average power of a PWM Signal and it is calculated using the following formula.

---

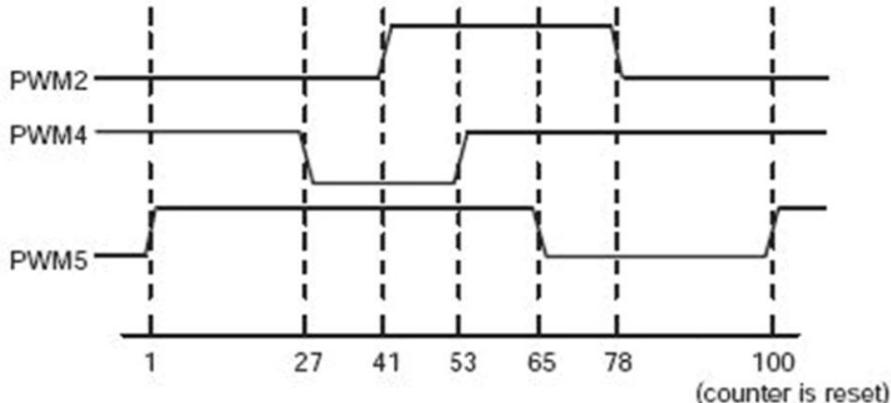
$$V_{AVG} = \text{Duty Cycle} * V_H$$

Where  $V_H$  is the maximum voltage level of the PWM Signal.

In LPC1768, there are 6 PWM outputs called **PWM1.1, PWM1.2, ...PWM1.6**

PWM1.1	P1.18 / P2.0
PWM1.2	P1.20 / P2.1 / P3.25
PWM1.3	P1.21 / P2.2 / P3.26
PWM1.4	P1.23 / P2.3
PWM1.5	P1.24 / P2.4
PWM1.6	P1.26 / P2.5

# Pulse Width Modulation (PWM)



The waveforms show one PWM cycle and demonstrate PWM outputs under the following conditions:

The timer is configured for PWM mode (counter resets to 1).

Match 0 is configured to reset the timer/counter when a match event occurs.

Control bits PWMSEL2 and PWMSEL4 are set.

The Match register values are as follows:

MR0 = 100 (PWM rate)

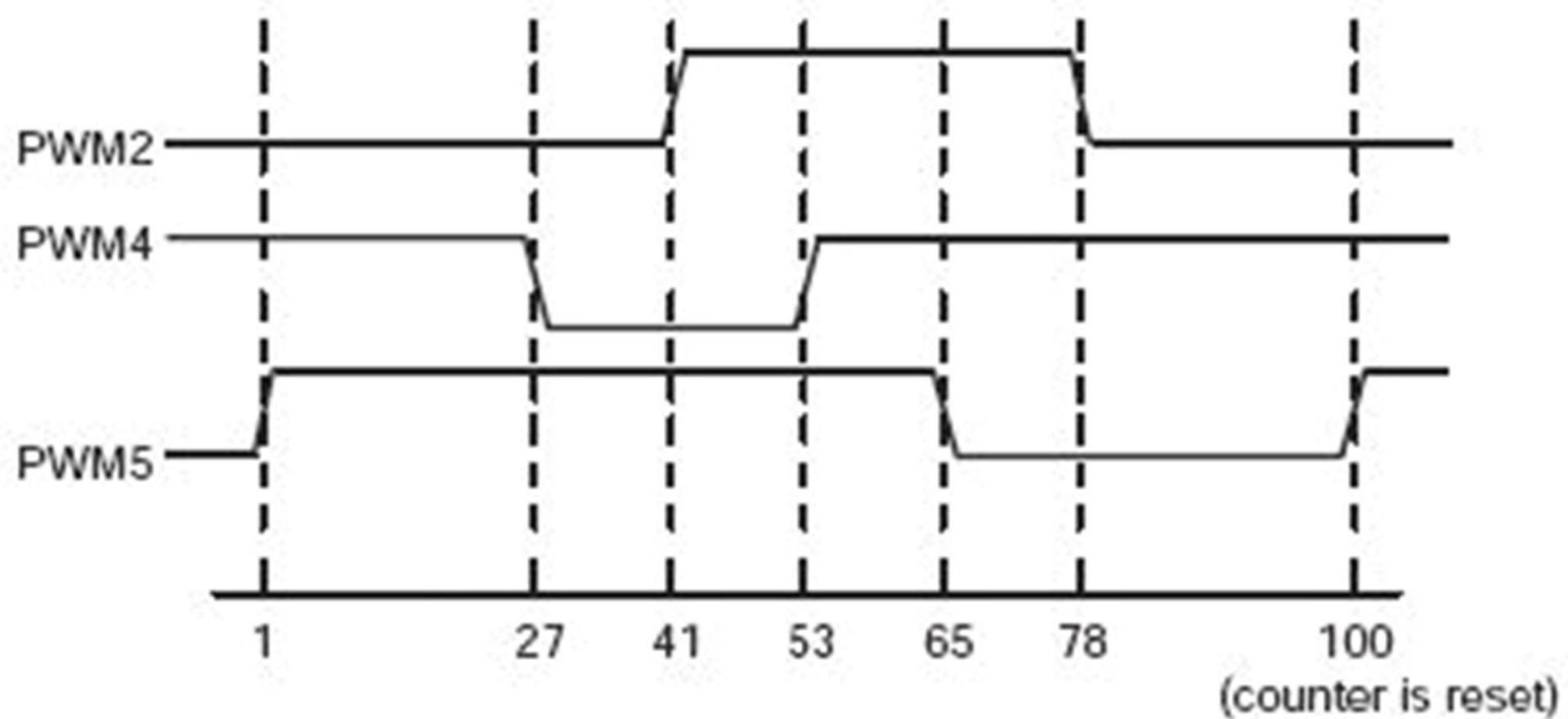
MR1 = 41, MR2 = 78 (PWM2 output)

MR3 = 53, MR4 = 27 (PWM4 output)

MR5 = 65 (PWM5 output)

Two match registers can be used to provide a single edge controlled PWM output. One match register (PWMMR0) controls the PWM cycle rate, by resetting the count upon match. The other match register controls the PWM edge position.

Three match registers can be used to provide a PWM output with both edges controlled. Again, the PWMMR0 match register controls the PWM cycle rate. The other match registers control the two PWM edge positions.



## Pulse Width Modulation (PWM)

---

The PWM is based on the standard Timer block and inherits all of its features, although only the PWM function is pinned out.

The Timer is designed to count cycles of the peripheral clock (PCLK) and optionally generate interrupts or perform other actions when specified timer values occur, based on seven match registers.

The PWM function is in addition to these features, and is based on match register events.

# Pulse Width Modulation (PWM)

---

## Set and reset inputs for PWM Flip-Flops

Single Edge

PWM Channel	Single Edge PWM (PWMSELn = 0)		Double Edge PWM (PWMSELn = 1)	
	Set by	Reset by	Set by	Reset by
1	Match 0	Match 1	Match 0 <sup>[1]</sup>	Match 1 <sup>[1]</sup>
2	Match 0	Match 2	Match 1	Match 2
3	Match 0	Match 3	Match 2 <sup>[2]</sup>	Match 3 <sup>[2]</sup>
4	Match 0	Match 4	Match 3	Match 4
5	Match 0	Match 5	Match 4 <sup>[2]</sup>	Match 5 <sup>[2]</sup>
6	Match 0	Match 6	Match 5	Match 6

Register	Description
IR	Interrupt Register: The IR can be read to identify which of eight possible interrupt sources are pending. Writing Logic-1 will clear the corresponding interrupt.
TCR	Timer Control Register: The TCR is used to control the Timer Counter functions(enable/disable/reset).
TC	Timer Counter: The 32-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR.
PR	Prescalar Register: This is used to specify the Prescalar value for incrementing the TC.
PC	Prescale Counter: The 32-bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, the TC is incremented.
MCR	Match Control Register: The MCR is used to control the resetting of TC and generating of interrupt whenever a Match occurs.
MR0	Match Register: This register hold the max cycle Time(Ton+Toff).
MR1-MR6	Match Registers: These registers holds the Match value(PWM Duty) for corresponding PWM channels(PWM1-PWM6).
PCR	PWM Control Register: PWM Control Register. Enables PWM outputs and selects PWM channel types as either single edge or double edge controlled.
LER	Load Enable Register: Enables use of new PWM values once the match occurs.

# Pulse Width Modulation (PWM)

## PWM Timer Control Register (PWMTCR)

The PWM Timer Control Register (PWMTCR) is used to control the operation of the PWM Timer Counter.

---

Bit 0	Counter Enable	When 1, PWM Timer Counter and Prescale Counter are enabled.
Bit 1	Counter Reset	When 1, PWM Timer Counter and Prescale Counter are reset on next positive edge of PCLK.
Bit 3	PWM Enable	When 1, PWM Mode is enabled. TC will reset to 1.

# Pulse Width Modulation (PWM)

## PWM Count Control Register (PWM1CTCR)

The Count Control Register (CTCR) is used to select between Timer and Counter mode

---

Bit	Symbol	Value	Description
1:0	Counter/ Timer Mode	00	Timer Mode: the TC is incremented when the Prescale Counter matches the Prescale Register.
		01	Counter Mode: the TC is incremented on rising edges of the PCAP input selected by bits 3:2.
		10	Counter Mode: the TC is incremented on falling edges of the PCAP input selected by bits 3:2.
		11	Counter Mode: the TC is incremented on both edges of the PCAP input selected by bits 3:2.
3:2	Count Input Select		When bits 1:0 of this register are not 00, these bits select which PCAP pin which carries the signal used to increment the TC.
		00	PCAP1.0
		01	PCAP1.1 (Other combinations are reserved)

# Pulse Width Modulation (PWM)

## PWM Match Control Register (PWM1MCR)

Bit	Symbol	Value	Description
0	PWMMR0I	1	Interrupt on PWMMR0: an interrupt is generated when PWMMR0 matches the value in the PWMTC.
		0	This interrupt is disabled.
1	PWMMR0R	1	Reset on PWMMR0: the PWMTC will be reset if PWMMR0 matches it.
		0	This feature is disabled.
2	PWMMR0S	1	Stop on PWMMR0: the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR0 matches the PWMTC.
		0	This feature is disabled
3	PWMMR1I	1	Interrupt on PWMMR1: an interrupt is generated when PWMMR1 matches the value in the PWMTC.
		0	This interrupt is disabled.
4	PWMMR1R	1	Reset on PWMMR1: the PWMTC will be reset if PWMMR1 matches it.
		0	This feature is disabled.
5	PWMMR1S	1	Stop on PWMMR1: the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR1 matches the PWMTC.
		0	This feature is disabled.
6	PWMMR2I	1	Interrupt on PWMMR2: an interrupt is generated when PWMMR2 matches the value in the PWMTC.
		0	This interrupt is disabled.
7	PWMMR2R	1	Reset on PWMMR2: the PWMTC will be reset if PWMMR2 matches it.
		0	This feature is disabled.

# Pulse Width Modulation (PWM)

Bit	Symbol	Value	Description
8	PWMMR2S	1	Stop on PWMMR2: the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR2 matches the PWMTC.
		0	This feature is disabled
9	PWMMR3I	1	Interrupt on PWMMR3: an interrupt is generated when PWMMR3 matches the value in the PWMTC.
		0	This interrupt is disabled.
10	PWMMR3R	1	Reset on PWMMR3: the PWMTC will be reset if PWMMR3 matches it.
		0	This feature is disabled
11	PWMMR3S	1	Stop on PWMMR3: the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR3 matches the PWMTC.
		0	This feature is disabled
12	PWMMR4I	1	Interrupt on PWMMR4: An interrupt is generated when PWMMR4 matches the value in the PWMTC.
		0	This interrupt is disabled.
13	PWMMR4R	1	Reset on PWMMR4: the PWMTC will be reset if PWMMR4 matches it.
		0	This feature is disabled.
14	PWMMR4S	1	Stop on PWMMR4: the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR4 matches the PWMTC.
		0	This feature is disabled
15	PWMMR5I	1	Interrupt on PWMMR5: An interrupt is generated when PWMMR5 matches the value in the PWMTC.
		0	This interrupt is disabled.
16	PWMMR5R	1	Reset on PWMMR5: the PWMTC will be reset if PWMMR5 matches it.
		0	This feature is disabled.
17	PWMMR5S	1	Stop on PWMMR5: the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR5 matches the PWMTC.
		0	This feature is disabled
18	PWMMR6I	1	Interrupt on PWMMR6: an interrupt is generated when PWMMR6 matches the value in the PWMTC.
		0	This interrupt is disabled.
19	PWMMR6R	1	Reset on PWMMR6: the PWMTC will be reset if PWMMR6 matches it.
		0	This feature is disabled.
20	PWMMR6S	1	Stop on PWMMR6: the PWMTC and PWMPC will be stopped and PWMTCR[0] will be set to 0 if PWMMR6 matches the PWMTC.

# Pulse Width Modulation (PWM)

## PWM Control Register (PWM1PCR)

The PWM Control Register is used to enable and select the type of each PWM channel.

Bit	Symbol	Value	Description	Bit	Symbol	Value	Description
1:0	Unused		Unused, always zero.	9	PWMENA1	1	The PWM1 output enabled.
2	PWMSEL2	1	Selects double edge controlled mode for the PWM2 output.			0	The PWM1 output disabled.
		0	Selects single edge controlled mode for PWM2.	10	PWMENA2	1	The PWM2 output enabled.
3	PWMSEL3	1	Selects double edge controlled mode for the PWM3 output.			0	The PWM2 output disabled.
		0	Selects single edge controlled mode for PWM3.	11	PWMENA3	1	The PWM3 output enabled.
4	PWMSEL4	1	Selects double edge controlled mode for the PWM4 output.			0	The PWM3 output disabled.
		0	Selects single edge controlled mode for PWM4.	12	PWMENA4	1	The PWM4 output enabled.
5	PWMSEL5	1	Selects double edge controlled mode for the PWM5 output.			0	The PWM4 output disabled.
		0	Selects single edge controlled mode for PWM5.	13	PWMENA5	1	The PWM5 output enabled.
6	PWMSEL6	1	Selects double edge controlled mode for the PWM6 output.			0	The PWM5 output disabled.
		0	Selects single edge controlled mode for PWM6.	14	PWMENA6	1	The PWM6 output enabled.
8:7	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.			0	The PWM6 output disabled.
				31:15	Unused		Unused, always zero.

# Pulse Width Modulation (PWM)

## PWM Latch Enable Register (PWM1LER)

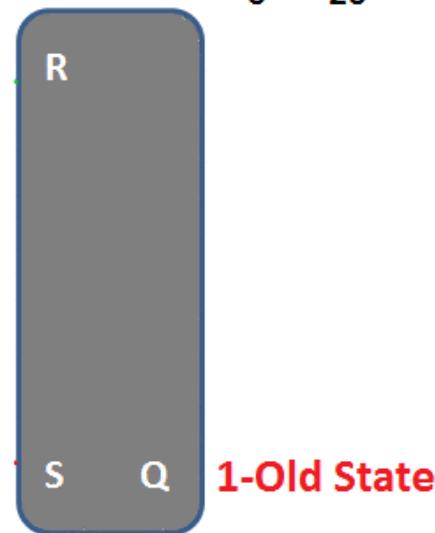
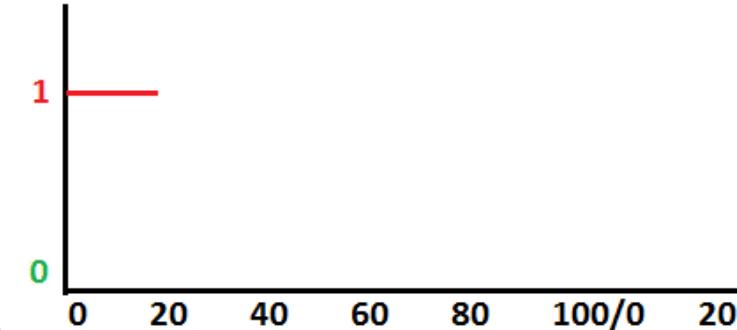
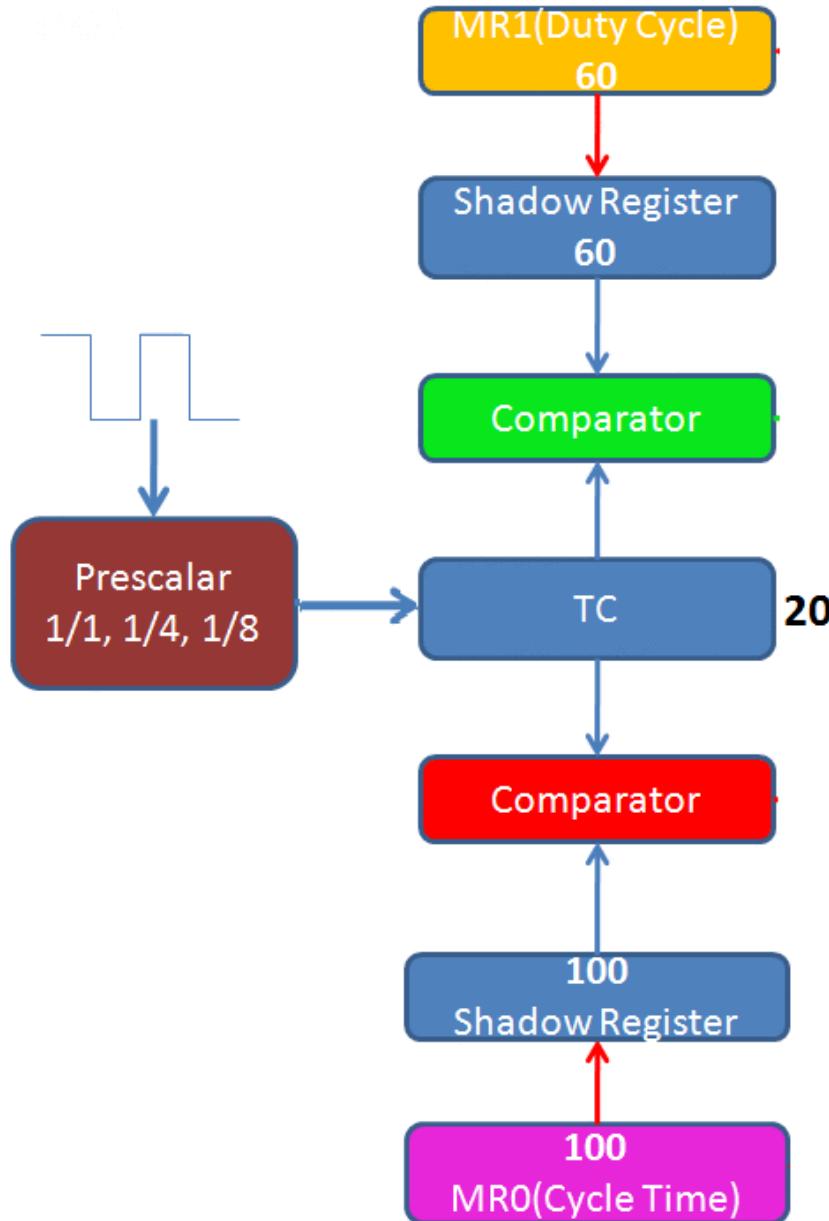
Bit	Symbol	Description
0	Enable PWM Match 0 Latch	Writing a one to this bit allows the last value written to the PWM Match 0 register to be become effective when the timer is next reset by a PWM Match event. See <a href="#">Section 24.6.4 "PWM Match Control Register (PWM1MCR - 0x4001 8014)".</a>
1	Enable PWM Match 1 Latch	Writing a one to this bit allows the last value written to the PWM Match 1 register to be become effective when the timer is next reset by a PWM Match event. See <a href="#">Section 24.6.4 "PWM Match Control Register (PWM1MCR - 0x4001 8014)".</a>
2	Enable PWM Match 2 Latch	Writing a one to this bit allows the last value written to the PWM Match 2 register to be become effective when the timer is next reset by a PWM Match event. See <a href="#">Section 24.6.4 "PWM Match Control Register (PWM1MCR - 0x4001 8014)".</a>
3	Enable PWM Match 3 Latch	Writing a one to this bit allows the last value written to the PWM Match 3 register to be become effective when the timer is next reset by a PWM Match event. See <a href="#">Section 24.6.4 "PWM Match Control Register (PWM1MCR - 0x4001 8014)".</a>
4	Enable PWM Match 4 Latch	Writing a one to this bit allows the last value written to the PWM Match 4 register to be become effective when the timer is next reset by a PWM Match event. See <a href="#">Section 24.6.4 "PWM Match Control Register (PWM1MCR - 0x4001 8014)".</a>
5	Enable PWM Match 5 Latch	Writing a one to this bit allows the last value written to the PWM Match 5 register to be become effective when the timer is next reset by a PWM Match event. See <a href="#">Section 24.6.4 "PWM Match Control Register (PWM1MCR - 0x4001 8014)".</a>
6	Enable PWM Match 6 Latch	Writing a one to this bit allows the last value written to the PWM Match 6 register to be become effective when the timer is next reset by a PWM Match event. See <a href="#">Section 24.6.4 "PWM Match Control Register (PWM1MCR - 0x4001 8014)".</a>

# Pulse Width Modulation (PWM)

## PWM Interrupt Register (PWM1IR)

The PWM Interrupt Register consists of 9 INTR Flags (7 Match, 2 Capture). If an interrupt is generated then the corresponding bit in the PWMIR will be high. Otherwise, the bit will be low. Writing a logic 1 to the corresponding IR bit will reset the interrupt.

Bit	Symbol	Description
0	PWMMR0 Interrupt	Interrupt flag for PWM match channel 0.
1	PWMMR1 Interrupt	Interrupt flag for PWM match channel 1.
2	PWMMR2 Interrupt	Interrupt flag for PWM match channel 2.
3	PWMMR3 Interrupt	Interrupt flag for PWM match channel 3.
4	PWMCAP0 Interrupt	Interrupt flag for capture input 0
5	PWMCAP1 Interrupt	Interrupt flag for capture input 1.
7:6	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.
8	PWMMR4 Interrupt	Interrupt flag for PWM match channel 4.
9	PWMMR5 Interrupt	Interrupt flag for PWM match channel 5.
10	PWMMR6 Interrupt	Interrupt flag for PWM match channel 6.
31:11	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.

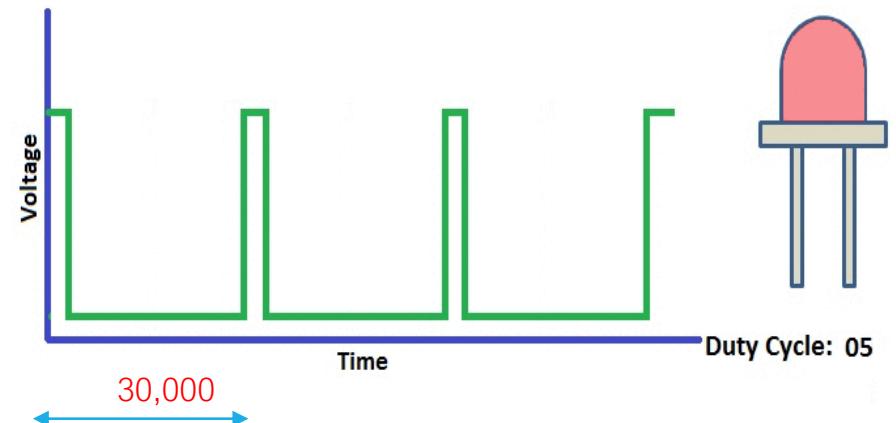


Slide: 1

# Pulse Width Modulation (PWM)

**PWM program to change the intensity of LED connected to (PWM1.4) P1.23 continuously**

```
#include <LPC17xx.H>
void pwm_init(void);
void PWM1_IRQHandler(void);
unsigned int a;
unsigned long int i;
unsigned char flag,flag1;
int main(void)
{
    SystemInit();
    SystemCoreClockUpdate();
    pwm_init();
    while(1);
}
void pwm_init(void)
{
    LPC_PINCON->PINSEL3 = 2<<14;           //pwm1.4 is selected for the
    LPC_PWM1->PR = 0x00000000;   //Count frequency : Fpclk
    LPC_PWM1->PCR = 0x00001000;   //select PWM1 single edge
    LPC_PWM1->MCR = 0x00000003;   //Reset and interrupt on PWMMR0
    LPC_PWM1->MR0 = 29999;      //setup match register 0 count -*+
    LPC_PWM1->MR4 = 50;        //setup match register MR4
    LPC_PWM1->LER = 0xFF;       //enable shadow copy register
    LPC_PWM1->TCR = 0x00000002; //RESET TC and PC
    LPC_PWM1->TCR = 0x00000009; //enable PWM and counter
    NVIC_EnableIRQ(PWM1_IRQHandler);
    return;
}
```



# Pulse Width Modulation (PWM)

```
void PWM1_IRQHandler(void)
{
    LPC_PWM1->IR = 0x01; //clear the interrupts
    if(flag == 0x0)
    {
        LPC_PWM1->MR4 += 50;
        LPC_PWM1->LER = 0xFF;
        if(LPC_PWM1->MR4 > 29900)
        {
            flag = 0x1;
            LPC_PWM1->LER = 0xFF; (Optional)
        }
    }
    else if(flag == 0x1)
    {
        LPC_PWM1->MR4 -= 50;
        LPC_PWM1->LER = 0xFF;
        if(LPC_PWM1->MR4 < 100)
        {
            flag = 0x0;
            LPC_PWM1->LER = 0xFF;
        }
    }
}
```

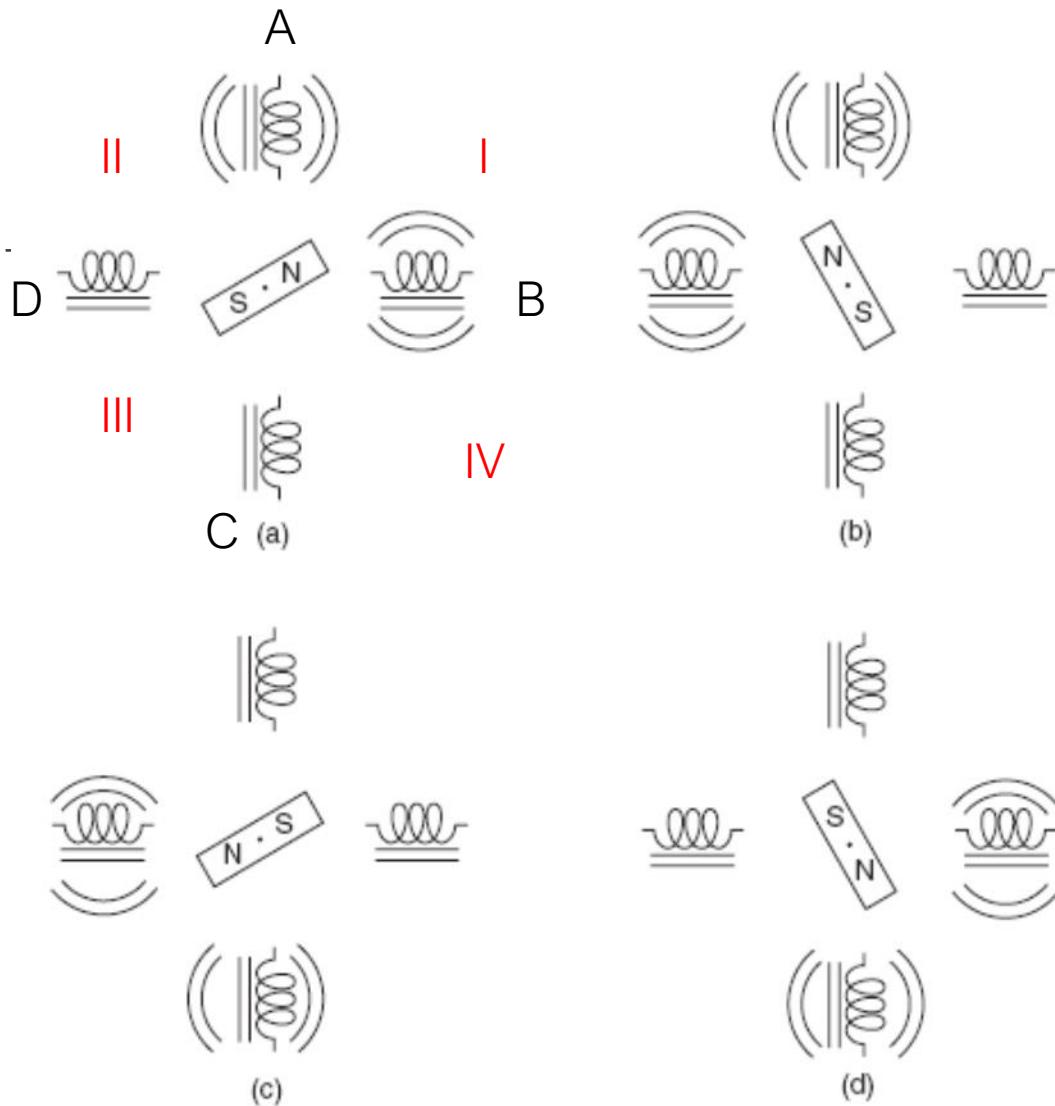


# Embedded System and Design

---

**DR MANOJ TOLANI (PHD-IIIT ALLAHABAD)**  
**ASSISTANT PROFESSOR (DEPARTMENT OF ICT)**

# Stepper Motor Interfacing

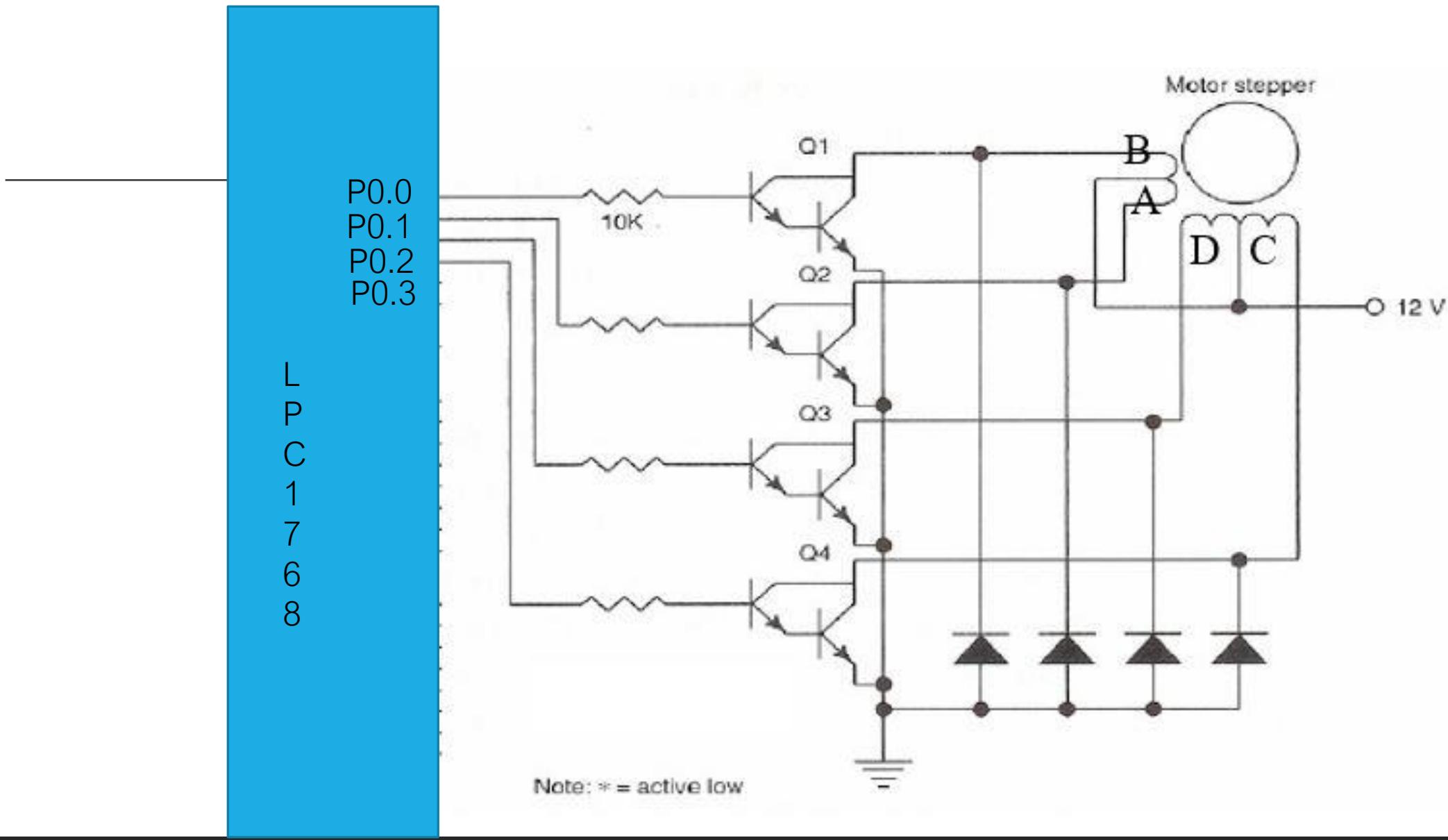


---

CDAB	
0 0 1 1	3
1 0 0 1	9
1 1 0 0	C
0 1 1 0	6

For Clockwise (Forward) : 3,9,C,6  
For Anticlockwise (Reverse): 3,6.C,9

# Stepper Motor Interfacing



# Stepper Motor Interfacing

Rotate the stepper motor 2.5 revolutions in the clockwise direction at 60 Revolutions Per Minute. Assume step angle = 6 degrees

```
#include <lpc17xx.h>
step_pos[] = { 3,9,C,6};
int main (void)
{
    unsigned int m,i;
    SystemInit();
    SystemCoreClockUpdate();
    LPC_GPIO0->FIODIR = 0x0F;// Output
    while ( 1 )
    {
        for(i=0; i < 150; i++)
        {
            LPC_GPIO0->FIOPIN = step_pos[i % 4];
            delay(); //Use Timer delay for 16.67 ms
        }
    }
}
```

Time for 1 revolution @60 rpm = 1 second  
Time for rotating one step = 1 second/60 steps = 16.67 ms  
(i.e  $360/6 = 60$  steps in 1 revolution)  
Number of steps needed =  $2.5 \times 60 = 150$

---

```
flag1=0;
temp1 = 0xCB;
lcd_write();
flag1=1;
i =0;
while (dval[i++] != '\0')
{
    temp1 = dval[i];
    lcd_write();//Send data bytes
}
for(i=0;i<7;i++)
vtg[i] = dval[i] = 0;
}
}
```

## Digital to Analog Converter (DAC)

DACs are used in audio equipment like Music Players to convert the digital data into analog audio signals. Similarly, there are video DACs, for converting digital video data into analog video signals to be displayed on a screen.

---

LPC1768 contains a 10-bit DAC peripheral based on Resistor String Architecture. It can produce buffered output and the maximum update rate is 1 MHz.

The output analog voltage of this 10-bit DAC can be calculated using the following formula.

$$V_{AOUT} = \frac{DACVALUE * (VREFP - VREFN)}{1024} + VREFN$$

Where,

$V_{AOUT}$  = Output Analog Voltage of DAC [Pin P0.26]

$V_{REFP}$  = Positive Reference Voltage of DAC

$V_{REFN}$  = Negative Reference Voltage of DAC

DACVALUE = 10-bit digital value, which must be converted to analog voltage.

## Digital to Analog Converter (DAC)

DACs are used in audio equipment like Music Players to convert the digital data into analog audio signals. Similarly, there are video DACs, for converting digital video data into analog video signals to be displayed on a screen.

---

LPC1768 contains a 10-bit DAC peripheral based on Resistor String Architecture. It can produce buffered output and the maximum update rate is 1 MHz.

The output analog voltage of this 10-bit DAC can be calculated using the following formula.

$$V_{AOUT} = \frac{DACVALUE * (VREFP - VREFN)}{1024} + VREFN$$

Where,

$V_{AOUT}$  = Output Analog Voltage of DAC [Pin P0.26]

$V_{REFP}$  = Positive Reference Voltage of DAC

$V_{REFN}$  = Negative Reference Voltage of DAC

DACVALUE = 10-bit digital value, which must be converted to analog voltage.

# Digital to Analog Converter (DAC)

**DACR – D/A Converter Register:** It contains the digital value that must be converted to analog value.

---

Bits [15:6]	VALUE	These bits contain the digital value (DACVALUE) that is to be converted to analog value ( $V_{AOUT}$ ) based on the previously mentioned formula.
Bit [16]	BIAS	When 0, settling time of DAC is 1 $\mu$ S (update rate is 1 MHz), max current is 700 $\mu$ A. When 1, settling time of DAC is 2.5 $\mu$ S (update rate is 400 kHz), max current is 350 $\mu$ A.

# Digital to Analog Converter (DAC)

## D/A Converter Control register (DACCCTRL)

Bit	Symbol	Value	Description
0	INT_DMA_REQ	0	This bit is cleared on any write to the DACR register.
		1	This bit is set by hardware when the timer times out.
1	DBLBUF_ENA	0	DACR double-buffering is disabled.
		1	When this bit and the CNT_ENA bit are both set, the double-buffering feature in the DACR register will be enabled. Writes to the DACR register are written to a pre-buffer and then transferred to the DACR on the next time-out of the counter.
2	CNT_ENA	0	Time-out counter operation is disabled.
		1	Time-out counter operation is enabled.
3	DMA_ENA	0	DMA access is disabled.
		1	DMA Burst Request Input 7 is enabled for the DAC (see <a href="#">Table 544</a> ).

## D/A Converter Counter Value register (DACCNTVAL)

Bit	Symbol	Description
15:0	VALUE	16-bit reload value for the DAC interrupt/DMA timer.

Double buffering is used in DACs when smooth, glitch-free analog output is required and precise timing control or reduced latency is necessary.

### Need of Double Buffering

- 1.Elimination of Glitches and Discontinuities**
- 2.Synchronization with Timing Requirements**
- 3.Reduced Latency and Improved Performance**
- 4.Flexibility and Responsiveness**

# Digital to Analog Converter (DAC)

## Double buffering

---

- Double-buffering is enabled only if both, the CNT\_ENA and the DBLBUF\_ENA bits are set in DACCTRL.
- In this case, any write to the DACR register will only load the pre-buffer
- The DACR will be loaded from the pre-buffer whenever the counter reaches zero
- At the same time the counter is reloaded with the COUNTVAL register value.
- Reading the DACR register will only return the contents of the DACR register itself, not the contents of the pre-buffer register.

# Digital to Analog Converter (DAC)

Generate a sawtooth waveform with peak-to-peak amplitude 3.3v at P0.26.

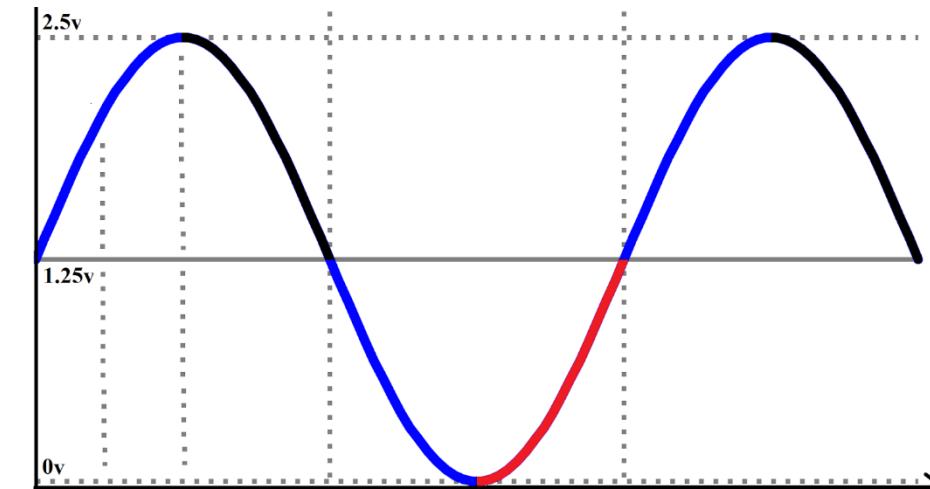
```
#include <lpc17xx.h>
void DAC_Init(void);
int main (void)
{
    unsigned int m,i;
    SystemInit();
    SystemCoreClockUpdate();
    LPC_PINCON->PINSEL1 = 2<<20;// Analog Output P0.26
    LPC_DAC->DACCNTVAL = 0x0050; // DAC Counter for Double Buffering
    LPC_DAC->DACCTRL = (0x1<<1)|(0x1<<2); //Double buffering
    while ( 1 )
    {
        LPC_DAC->DACR = (i << 6) ;
        i++;
        if ( i == 0x400 )           //Maximum value is 0x3FF in 10 bit DAC
        {
            i = 0;
        }
    }
}
```



# Digital to Analog Converter (DAC)

Generate a sinewave with peak to peak amplitude 2.5v at P0.26. (i.e.  $V_{out} = 1.25 + 1.25 \sin \theta$  )

```
#include <lpc17xx.h>
void DAC_Init(void);
sinetable[] = { 388, 582, 723, 776, 723, 582, 388, 194, 52, 0, 52, 194};
int main (void)
{
    unsigned int m,i;
    SystemInit();
    SystemCoreClockUpdate();
    LPC_PINCON->PINSEL1 = 2<<20;// Analog Input P0.26
    while ( 1 )
    {
        for(i=0; i < 12; i++) // Assuming samples separated by 30 degrees
        {
            LPC_DAC->DACR = (sinetable[i % 12]<< 6) ;
            delay(); // Call timer delay based on period. If period is 10 ms. Delay is (10ms/12)
        }
    }
}
```



# Digital to Analog Converter (DAC)

Generate a sinewave with peak to peak amplitude 2.5v at P0.26. (i.e.  $V_{out} = 1.25 + 1.25 \sin \theta$  )

$$V_0 = 1.25 + 1.25 \sin 0 = 1.25; \text{ Dig value} = 1024 \times V/3.3 = 388$$

$$V_{30} = 1.25 + 1.25 \sin 30 = 1.875; \text{ Dig value} = 582$$

$$V_{60} = 1.25 + 1.25 \sin 60 = 2.33; \text{ Dig value} = 723$$

$$V_{90} = 1.25 + 1.25 \sin 90 = 2.5; \text{ Dig value} = 776$$

$$V_{120} = 1.25 + 1.25 \sin 120 = 2.33; \text{ Dig value} = 723$$

$$V_{150} = 1.25 + 1.25 \sin 150 = 1.875; \text{ Dig value} = 582$$

$$V_{180} = 1.25 + 1.25 \sin 180 = 1.25; \text{ Dig value} = 388$$

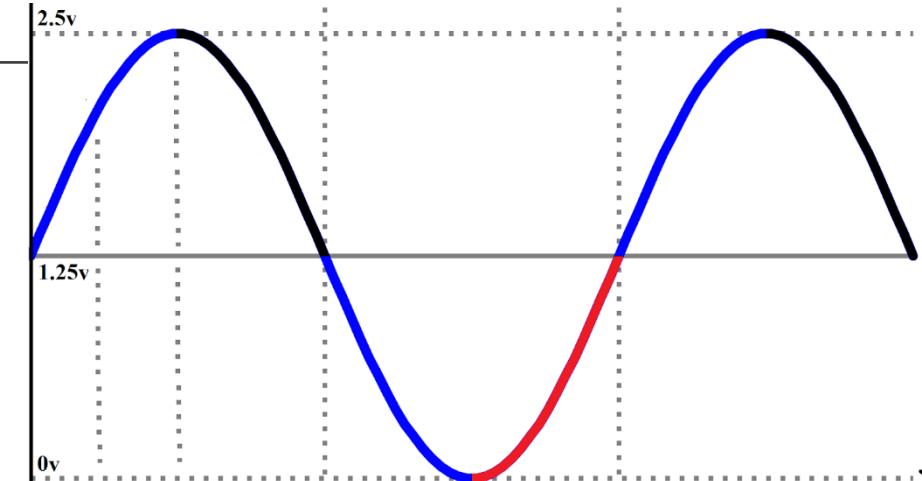
$$V_{210} = 1.25 + 1.25 \sin 210 = 0.626; \text{ Dig value} = 194$$

$$V_{240} = 1.25 + 1.25 \sin 240 = 0.167; \text{ Dig value} = 52$$

$$V_{270} = 1.25 + 1.25 \sin 270 = 0; \text{ Dig value} = 0$$

$$V_{310} = 1.25 + 1.25 \sin 310 = 0.167; \text{ Dig value} = 52$$

$$V_{330} = 1.25 + 1.25 \sin 330 = 0.626; \text{ Dig value} = 194$$



---

# ADC Burst Mode Interrupt with ADC

Register	Description
ADCR	A/D COnrol Register: Used for Configuring the ADC
ADGDR	A/D Global Data Register: This register contains the ADC's DONE bit and the result of the most recent A/D conversion
ADINTEN	A/D Interrupt Enable Register
ADDR0 - ADDR7	A/D Channel Data Register: Contains the recent ADC value for respective channel
ADSTAT	A/D Status Register: Contains DONE & OVERRUN flag for all the ADC channels

## ADC Register Configuration

Now lets see how to configure the individual registers for ADC conversion.

ADCR									
31:28	27	26:24	23:22	21	20:17	16	15:8	7:0	
Reserved	EDGE	START	Reserved	PDN	Reserved	BURST	CLCKDIV	SEL	

## ADGDR ( ADC Global Data Register )

ADGDR					
31	27	26:24	23:16	15:4	3:0
DONE	OVERRUN	CHN	Reserved	RESULT	Reserved

### Bit 15:4 - RESULT

This field contains the 12bit A/D conversion value for the selected channel in **ADCR.SEL**

The vale for this register should be read oncve the conversion is completed ie DONE bit is set.

### Bit 26:24 - CHN : Channel

These bits contain the channel number for which the A/D conversion is done and the converted value is available in RESULT bits(e.g. 000 identifies channel 0, 011 channel 3...).

### Bit 27 - OVERRUN

This bit is set during the BURST mode where the previous conversion data is overwritten by the new A/D conversion value.

### Bit 31 - DONE

This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read and when the ADCR is written. If the ADCR is written while a conversion is still in progress, this bit is set and a new conversion is started.

# Analog To Digital Converter (ADC)

**A/D Interrupt Enable register (ADINTEN) :** This register allows control over which A/D channels generate an interrupt when a conversion is complete.

Bit	Symbol	Value	Description
0	ADINTEN0	0	Completion of a conversion on ADC channel 0 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 0 will generate an interrupt.
1	ADINTEN1	0	Completion of a conversion on ADC channel 1 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 1 will generate an interrupt.
2	ADINTEN2	0	Completion of a conversion on ADC channel 2 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 2 will generate an interrupt.
3	ADINTEN3	0	Completion of a conversion on ADC channel 3 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 3 will generate an interrupt.
4	ADINTEN4	0	Completion of a conversion on ADC channel 4 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 4 will generate an interrupt.
5	ADINTEN5	0	Completion of a conversion on ADC channel 5 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 5 will generate an interrupt.
6	ADINTEN6	0	Completion of a conversion on ADC channel 6 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 6 will generate an interrupt.
7	ADINTEN7	0	Completion of a conversion on ADC channel 7 will not generate an interrupt.
		1	Completion of a conversion on ADC channel 7 will generate an interrupt.
8	ADGINTEN	0	Only the individual ADC channels enabled by ADINTEN7:0 will generate interrupts.  <b>Remark:</b> This bit must be set to 0 in burst mode (BURST = 1 in the AD0CR register).
		1	Only the global DONE flag in ADDR is enabled to generate an interrupt.
31:17 -			Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.

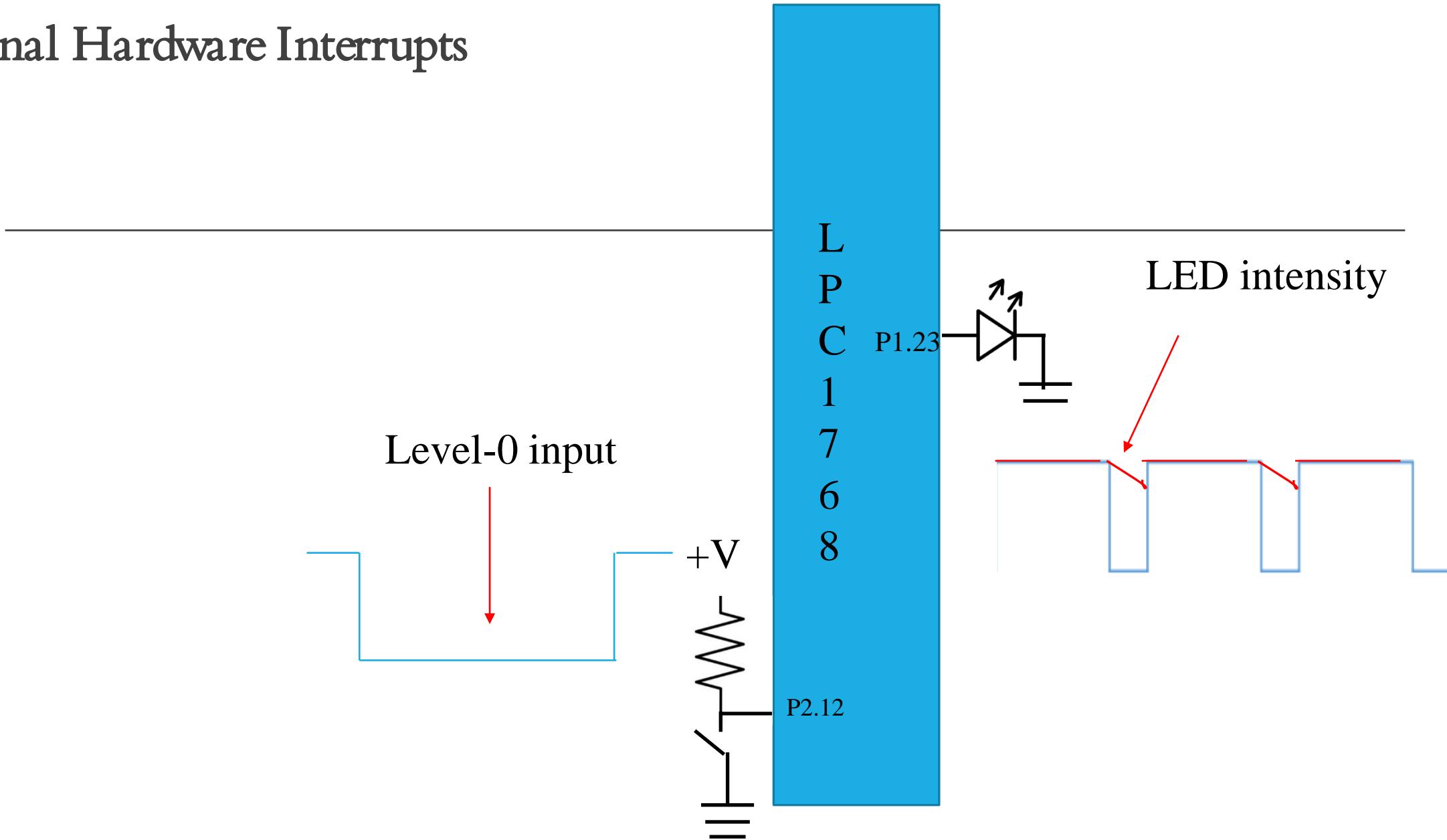
## ADC burst mode for 2-channel concurrent conversion

```
#include<LPC17xx.h>
#include<stdio.h>
int main(void)

{
    SystemInit();
    SystemCoreClockUpdate();
    LPC_PINCON->PINSEL3 =(3<<28)|(3<<30); //P1.30 as AD0.4 and P1.31 as AD0.5
    LPC_ADC->ADCR = (1<<4) | (1<<5)|(1<<16) | (1<<21); //Enable CH 4 and 5 for BURST mode with ADC power ON
    LPC_ADC->ADINTEN =(1<<4)|(1<<5); // Enable DONE for INTR
    NVIC_EnableIRQ(ADC_IRQn);
    while(1);
}

void ADC_IRQHandler(void)
{
    int channel,temp,result;
    channel=(LPC_ADC->ADGDR >>24) & 0x07;
    result= ( LPC_ADC->ADGDR >>4) & 0xFFFF;
    if(channel == 4)
    {
        temp4 = ( LPC_ADC->ADDR4 >>4) & 0xFFFF ; //Read to Clear Done flag
    }
    else if(channel == 5)
    {
        temp5 = ( LPC_ADC->ADDR5 >>4) & 0xFFFF ; //Read to Clear Done flag
    }
    //Now you can use temp4 and temp5 for further processing based on your requirement
}
```

# External Hardware Interrupts



# GPIO Interrupts

The pins of Port-0 and Port-2 can generate GPIO interrupts.

GPIO interrupts are mapped to EINT3 ISR

---

## GPIO overall Interrupt Status register (IOIntStatus)

Bit	Symbol	Value	Description
0	P0Int		Port 0 GPIO interrupt pending.
		0	There are no pending interrupts on Port 0.
		1	There is at least one pending interrupt on Port 0.
1	-	-	Reserved. The value read from a reserved bit is not defined.
2	P2Int		Port 2 GPIO interrupt pending.
		0	There are no pending interrupts on Port 2.
		1	There is at least one pending interrupt on Port 2.
31:2	-	-	Reserved. The value read from a reserved bit is not defined.

# GPIO Interrupts

## GPIO Interrupt Enable for port 0 Rising Edge (IO0IntEnR)

Bit	Symbol	Value	Description
0	P0.0ER	0	Enable rising edge interrupt for P0.0.
		1	Rising edge interrupt is disabled on P0.0.
1	P0.1ER		Enable rising edge interrupt for P0.1.
2	P0.2ER		Enable rising edge interrupt for P0.2.
3	P0.3ER		Enable rising edge interrupt for P0.3.
4	P0.4ER <sup>[1]</sup>		Enable rising edge interrupt for P0.4.
5	P0.5ER <sup>[1]</sup>		Enable rising edge interrupt for P0.5.
6	P0.6ER		Enable rising edge interrupt for P0.6.
7	P0.7ER		Enable rising edge interrupt for P0.7.
8	P0.8ER		Enable rising edge interrupt for P0.8.
9	P0.9ER		Enable rising edge interrupt for P0.9.
10	P0.10ER		Enable rising edge interrupt for P0.10.
11	P0.11ER		Enable rising edge interrupt for P0.11.
14:12	-		Reserved
15	P0.15ER		Enable rising edge interrupt for P0.15.
16	P0.16ER		Enable rising edge interrupt for P0.16.

Bit	Symbol	Value	Description
17	P0.17ER		Enable rising edge interrupt for P0.17.
18	P0.18ER		Enable rising edge interrupt for P0.18.
19	P0.19ER <sup>[1]</sup>		Enable rising edge interrupt for P0.19.
20	P0.20ER <sup>[1]</sup>		Enable rising edge interrupt for P0.20.
21	P0.21ER <sup>[1]</sup>		Enable rising edge interrupt for P0.21.
22	P0.22ER		Enable rising edge interrupt for P0.22.
23	P0.23ER <sup>[1]</sup>		Enable rising edge interrupt for P0.23.
24	P0.24ER <sup>[1]</sup>		Enable rising edge interrupt for P0.24.
25	P0.25ER		Enable rising edge interrupt for P0.25.
26	P0.26ER		Enable rising edge interrupt for P0.26.
27	P0.27ER <sup>[1]</sup>		Enable rising edge interrupt for P0.27.
28	P0.28ER <sup>[1]</sup>		Enable rising edge interrupt for P0.28.
29	P0.29ER		Enable rising edge interrupt for P0.29.
30	P0.30ER		Enable rising edge interrupt for P0.30.
31	-		Reserved.

Similarly ----- GPIO Interrupt Enable for port 2 (P2.0-P2.13) Rising Edge (IO2IntEnR)

# GPIO Interrupts

## GPIO Interrupt Enable for port 0 Falling Edge (IO0IntEnF)

Bit	Symbol	Value	Description
0	P0.0EF	0	Enable falling edge interrupt for P0.0.
		1	Falling edge interrupt is disabled on P0.0.
1	P0.1EF		Enable falling edge interrupt for P0.1.
2	P0.2EF		Enable falling edge interrupt for P0.2.
3	P0.3EF		Enable falling edge interrupt for P0.3.
4	P0.4EF <sup>[1]</sup>		Enable falling edge interrupt for P0.4.
5	P0.5EF <sup>[1]</sup>		Enable falling edge interrupt for P0.5.
6	P0.6EF		Enable falling edge interrupt for P0.6.
7	P0.7EF		Enable falling edge interrupt for P0.7.
8	P0.8EF		Enable falling edge interrupt for P0.8.
9	P0.9EF		Enable falling edge interrupt for P0.9.
10	P0.10EF		Enable falling edge interrupt for P0.10.
11	P0.11EF		Enable falling edge interrupt for P0.11.
14:12	-		Reserved.
15	P0.15EF		Enable falling edge interrupt for P0.15.
16	P0.16EF		Enable falling edge interrupt for P0.16.
17	P0.17EF		Enable falling edge interrupt for P0.17.
18	P0.18EF		Enable falling edge interrupt for P0.18.
19	P0.19EF <sup>[1]</sup>		Enable falling edge interrupt for P0.19.
20	P0.20EF <sup>[1]</sup>		Enable falling edge interrupt for P0.20.
21	P0.21EF <sup>[1]</sup>		Enable falling edge interrupt for P0.21.
22	P0.22EF		Enable falling edge interrupt for P0.22.
23	P0.23EF <sup>[1]</sup>		Enable falling edge interrupt for P0.23.
24	P0.24EF <sup>[1]</sup>		Enable falling edge interrupt for P0.24.
25	P0.25EF		Enable falling edge interrupt for P0.25.
26	P0.26EF		Enable falling edge interrupt for P0.26.
27	P0.27EF <sup>[1]</sup>		Enable falling edge interrupt for P0.27.
28	P0.28EF <sup>[1]</sup>		Enable falling edge interrupt for P0.28.
29	P0.29EF		Enable falling edge interrupt for P0.29.
30	P0.30EF		Enable falling edge interrupt for P0.30.
31	-		Reserved.

Similarly ----- GPIO Interrupt Enable for port 2 (P2.0-P2.13) Falling Edge (IO2IntEnF)

# GPIO Interrupts

## GPIO Interrupt Status for port 0 Rising Edge Interrupt (IO0IntStatR)

Bit	Symbol	Value	Description	Bit	Symbol	Description
0	P0.0REI		Status of Rising Edge Interrupt for P0.0	15	P0.15REI	Status of Rising Edge Interrupt for P0.15.
		0	A rising edge has not been detected on P0.0.	16	P0.16REI	Status of Rising Edge Interrupt for P0.16.
		1	Interrupt has been generated due to a rising edge on P0.0.	17	P0.17REI	Status of Rising Edge Interrupt for P0.17.
1	P0.1REI		Status of Rising Edge Interrupt for P0.1.	18	P0.18REI	Status of Rising Edge Interrupt for P0.18.
2	P0.2REI		Status of Rising Edge Interrupt for P0.2.	19	P0.19REI <sup>[1]</sup>	Status of Rising Edge Interrupt for P0.19.
3	P0.3REI		Status of Rising Edge Interrupt for P0.3.	20	P0.20REI <sup>[1]</sup>	Status of Rising Edge Interrupt for P0.20.
4	P0.4REI <sup>[1]</sup>		Status of Rising Edge Interrupt for P0.4.	21	P0.21REI <sup>[1]</sup>	Status of Rising Edge Interrupt for P0.21.
5	P0.5REI <sup>[1]</sup>		Status of Rising Edge Interrupt for P0.5.	22	P0.22REI	Status of Rising Edge Interrupt for P0.22.
6	P0.6REI		Status of Rising Edge Interrupt for P0.6.	23	P0.23REI <sup>[1]</sup>	Status of Rising Edge Interrupt for P0.23.
7	P0.7REI		Status of Rising Edge Interrupt for P0.7.	24	P0.24REI <sup>[1]</sup>	Status of Rising Edge Interrupt for P0.24.
8	P0.8REI		Status of Rising Edge Interrupt for P0.8.	25	P0.25REI	Status of Rising Edge Interrupt for P0.25.
9	P0.9REI		Status of Rising Edge Interrupt for P0.9.	26	P0.26REI	Status of Rising Edge Interrupt for P0.26.
10	P0.10REI		Status of Rising Edge Interrupt for P0.10.	27	P0.27REI <sup>[1]</sup>	Status of Rising Edge Interrupt for P0.27.
11	P0.11REI		Status of Rising Edge Interrupt for P0.11.	28	P0.28REI <sup>[1]</sup>	Status of Rising Edge Interrupt for P0.28.
14:12	-		Reserved.	29	P0.29REI	Status of Rising Edge Interrupt for P0.29.
				30	P0.30REI	Status of Rising Edge Interrupt for P0.30.
				31	-	Reserved.

Similarly ----- GPIO Interrupt Status for port 2 (P2.0-P2.13) Rising Edge Interrupt (IO2IntStatR)

# GPIO Interrupts

## GPIO Interrupt Status for port 0 Falling Edge Interrupt (IO0IntStatF)

Bit	Symbol	Value	Description
0	P0.0FEI		Status of Falling Edge Interrupt for P0.0
		0	A falling edge has not been detected on P0.0.
		1	Interrupt has been generated due to a falling edge on P0.0.
1	P0.1FEI		Status of Falling Edge Interrupt for P0.1.
2	P0.2FEI		Status of Falling Edge Interrupt for P0.2.
3	P0.3FEI		Status of Falling Edge Interrupt for P0.3.
4	P0.4FEI <sup>[1]</sup>		Status of Falling Edge Interrupt for P0.4.
5	P0.5FEI <sup>[1]</sup>		Status of Falling Edge Interrupt for P0.5.
6	P0.6FEI		Status of Falling Edge Interrupt for P0.6.
7	P0.7FEI		Status of Falling Edge Interrupt for P0.7.
8	P0.8FEI		Status of Falling Edge Interrupt for P0.8.
9	P0.9FEI		Status of Falling Edge Interrupt for P0.9.
10	P0.10FEI		Status of Falling Edge Interrupt for P0.10.
11	P0.11FEI		Status of Falling Edge Interrupt for P0.11.
14:12	-		Reserved.
15	P0.15FEI		Status of Falling Edge Interrupt for P0.15.
16	P0.16FEI		Status of Falling Edge Interrupt for P0.16.
17	P0.17FEI		Status of Falling Edge Interrupt for P0.17.
18	P0.18FEI		Status of Falling Edge Interrupt for P0.18.
19	P0.19FEI <sup>[1]</sup>		Status of Falling Edge Interrupt for P0.19.
20	P0.20FEI <sup>[1]</sup>		Status of Falling Edge Interrupt for P0.20.
21	P0.21FEI <sup>[1]</sup>		Status of Falling Edge Interrupt for P0.21.
22	P0.22FEI		Status of Falling Edge Interrupt for P0.22.
23	P0.23FEI <sup>[1]</sup>		Status of Falling Edge Interrupt for P0.23.
24	P0.24FEI <sup>[1]</sup>		Status of Falling Edge Interrupt for P0.24.
25	P0.25FEI		Status of Falling Edge Interrupt for P0.25.
26	P0.26FEI		Status of Falling Edge Interrupt for P0.26.
27	P0.27FEI <sup>[1]</sup>		Status of Falling Edge Interrupt for P0.27.
28	P0.28FEI <sup>[1]</sup>		Status of Falling Edge Interrupt for P0.28.
29	P0.29FEI		Status of Falling Edge Interrupt for P0.29.
30	P0.30FEI		Status of Falling Edge Interrupt for P0.30.
31	-		Reserved.

Similarly ----- GPIO Interrupt Status for port 2 (P2.0-P2.13) Falling Edge Interrupt (IO2IntStatF)

# GPIO Interrupts

## GPIO Interrupt Clear register for port 0 (IO0IntClr)

Bit	Symbol	Value	Description
0	P0.0CI	Clear	Clear GPIO port Interrupts for P0.0
		0	Corresponding bits in IOxIntStatR and IOxIntStatF are unchanged.
		1	Corresponding bits in IOxIntStatR and IOxStatF are cleared
1	P0.1CI		Clear GPIO port Interrupts for P0.1.
2	P0.2CI		Clear GPIO port Interrupts for P0.2.
3	P0.3CI		Clear GPIO port Interrupts for P0.3.
4	P0.4CI <sup>[1]</sup>		Clear GPIO port Interrupts for P0.4.
5	P0.5CI <sup>[1]</sup>		Clear GPIO port Interrupts for P0.5.
6	P0.6CI		Clear GPIO port Interrupts for P0.6.
7	P0.7CI		Clear GPIO port Interrupts for P0.7.
8	P0.8CI		Clear GPIO port Interrupts for P0.8.
9	P0.9CI		Clear GPIO port Interrupts for P0.9.
10	P0.10CI		Clear GPIO port Interrupts for P0.10.
11	P0.11CI		Clear GPIO port Interrupts for P0.11.
14:12	-		Reserved.

15	P0.15CI	Clear GPIO port Interrupts for P0.15.
16	P0.16CI	Clear GPIO port Interrupts for P0.16.
17	P0.17CI	Clear GPIO port Interrupts for P0.17.
18	P0.18CI	Clear GPIO port Interrupts for P0.18.
19	P0.19CI <sup>[1]</sup>	Clear GPIO port Interrupts for P0.19.
20	P0.20CI <sup>[1]</sup>	Clear GPIO port Interrupts for P0.20.
21	P0.21CI <sup>[1]</sup>	Clear GPIO port Interrupts for P0.21.
22	P0.22CI	Clear GPIO port Interrupts for P0.22.
23	P0.23CI <sup>[1]</sup>	Clear GPIO port Interrupts for P0.23.
24	P0.24CI <sup>[1]</sup>	Clear GPIO port Interrupts for P0.24.
25	P0.25CI	Clear GPIO port Interrupts for P0.25.
26	P0.26CI	Clear GPIO port Interrupts for P0.26.
27	P0.27CI <sup>[1]</sup>	Clear GPIO port Interrupts for P0.27.
28	P0.28CI <sup>[1]</sup>	Clear GPIO port Interrupts for P0.28.
29	P0.29CI	Clear GPIO port Interrupts for P0.29.
30	P0.30CI	Clear GPIO port Interrupts for P0.30.
31	-	Reserved.

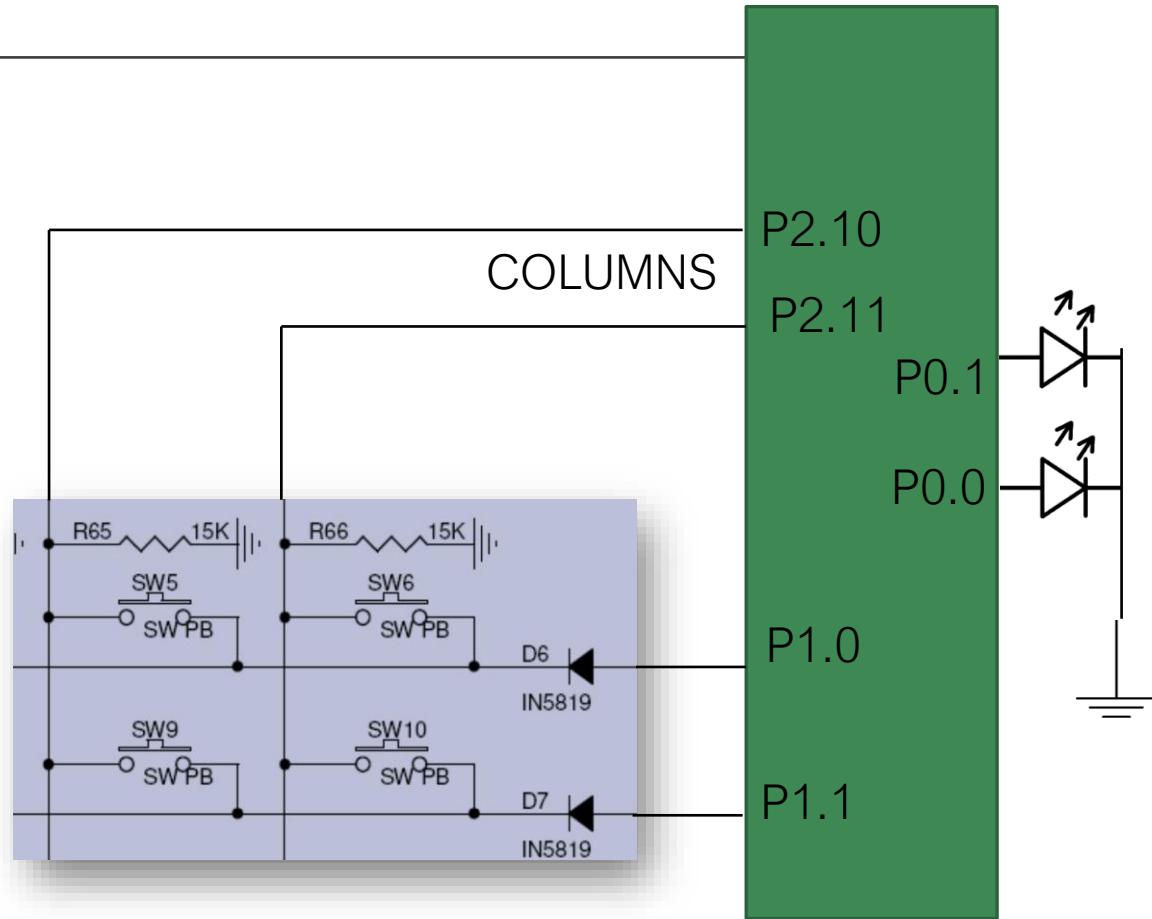
Similarly ----- GPIO Interrupt Clear Register for port 2 (P2.0-P2.13) (IO2IntClr)

Turn ON the LED connected to P1.23 whenever the +ve edge applied to P0.0 and Turn OFF whenever the +ve edge applied at P0.1

```
#include<LPC17xx.h>
void EINT3_IRQHandler(void);
int main(void)
{
    SystemInit();
    SystemCoreClockUpdate();
    LPC_GPIO1->FIODIR = 1<<23;           //P1.23 is assigned output
    LPC_GPIO1->FIOCLR = 1<<23;          //Initially LED is kept OFF
    LPC_GPIOINT->IO0IntEnR=0x00000003;    //P0.0 and P0.1 - Enable Rising Edge
    NVIC_EnableIRQ(EINT3_IRQn);           //Enable EINT3
    while(1);
}
void EINT3_IRQHandler(void)
{
    unsigned int x=LPC_GPIOINT->IO0IntStatR; // Get the status of Rising Edge Interrupts
    LPC_GPIOINT->IO0IntClr |=x; //Clear the Interrupt
    if (x==0x00000001) //If Rising Edge at P0.0
        LPC_GPIO1->FIOSET = 0x00800000;
    else //If Rising edge at P0.1
        LPC_GPIO1->FIOCLR = 0x00800000;
}
```

# GPIO Interrupts

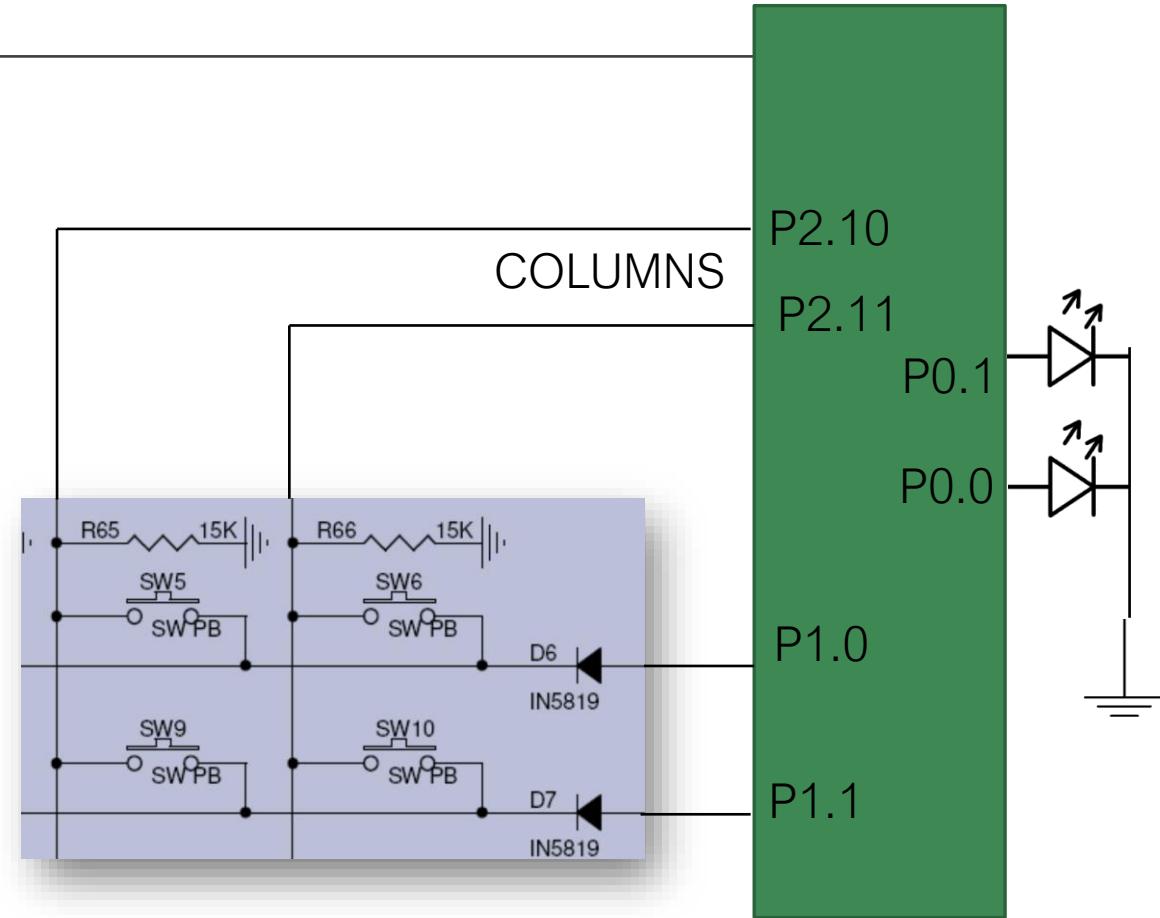
Assume that columns of a 2x2 matrix keyboard are connected to P2.10-P2.11 and rows are connected to P1.0-P1.1. Write an embedded C program using GPIO interrupt to display the keycode of the key pressed on LEDs connected to P0.0 to P0.1.



# GPIO Interrupts

Assume that columns of a 2x2 matrix keyboard are connected to P2.10-P2.11 and rows are connected to P1.0-P1.1. Write an embedded C program using GPIO interrupt to display the keycode of the key pressed on LEDs connected to P0.0 to P0.1.

```
#include <lpc17xx.h>
void EINT3_IRQHandler (void);
unsigned int row, col ;
int main(void)
{
    LPC_GPIO0->FIODIR =0x03;//p0.0 and p0.1 output
    LPC_GPIO1->FIODIR =0x03;// p1.0 to p1.1 output
    LPC_GPIO1->FIOSET =0x03;// Facilitate any key press
    detection
    LPC_GPIOINT->IO2IntEnR= 1<<10 | 1<<11; //Rising edge-
    P2.10,P2.11
    NVIC_EnableIRQ(EINT3_IRQn);//Enable GPIO INTR
    while(1);
}
```



```
void EINT3_IRQHandler (void)
{
    unsigned int temp3;
    temp3 = LPC_GPIOINT->IO2IntStatR; /
    if (temp3 == 1<<10)
        col = 0;
    else if (temp3 == 1<<11)
        col = 1;
    LPC_GPIOINT->IO2IntClr=1<<10 | 1<<11;//Clear Interrupt
    for (row=0;row <2;row++)
    {
        if (row==0)
            temp=0x01;
        else if (row==1)
            temp=0x02;
        LPC_GPIO1->FIOPIN=temp;
        if ( (LPC_GPIO2->FIOPIN & (1<<10) | (1<<11)) != 0) //Read the columns
        {
            LPC_GPIO0->FIOPIN=row *2+col; //Display the keycode
            LPC_GPIO1->FIOSET=0x03; // Facilitate any keypress detection
            break;
        }
    }
}
```

# Serial Communication

UART(Universal Asynchronous Receiver/Transceiver uses TxD(Transmit) Pin for sending Data and RxD(Receive) Pin to get data. UART sends & receives data in form of chunks or packets. These chunks or packets are also referred to as 'Frames'.

---



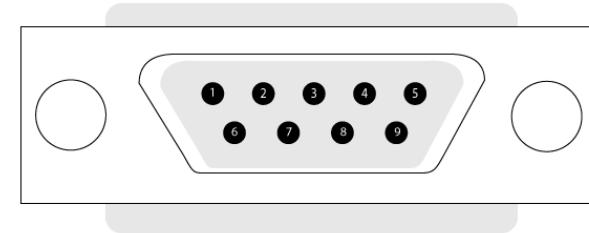
Pins:	TxD	RxD
UART0	P0.2	P0.3
UART1	P0.15/P2.0	P0.16/P2.1
UART2	P0.10/P2.8	P0.11/P2.9
UART3	P0.0/P0.25/P4.28	P0.1/P0.26/P4.29

# Serial Communication

The microcontroller has an inbuilt UART for carrying out serial communication. The serial communication is done in the asynchronous mode.

IBM introduced the **DB-9 RS-232** version of serial I/O standard, which is most widely used in PCs and several devices. In RS232, high and low bits are represented by flowing voltage ranges:

Bit	Voltage Range (in V)	
0	+3	+25
1	-25	-3



The range -3V to +3V is undefined.

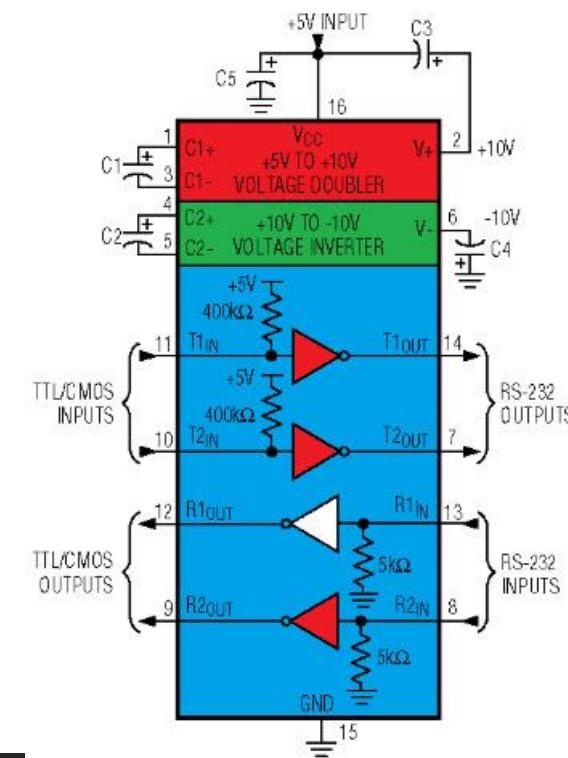
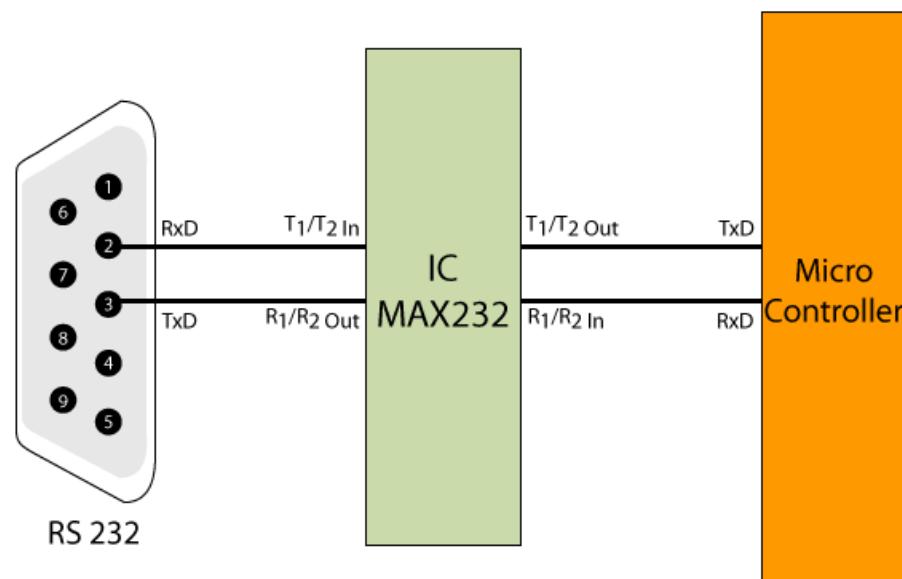
The TTL standards came a long time after the RS232 standard was set. Due to this reason RS232 voltage levels are not compatible with TTL logic.

Therefore, while connecting an RS232 to microcontroller system, a voltage converter is required. This converter converts the microcontroller output level to the RS232 voltage levels, and vice versa. IC **MAX232**, also known as line driver, is very commonly used for this purpose.

# Serial Communication

TxD pin of serial port connects to RxD pin of controller via MAX232. And similarly, RxD pin of serial port connects to the TxD pin of controller through MAX232.

MAX232 has two sets of **line drivers** for transferring and receiving data. The line drivers used for transmission are called T1 and T2, where as the line drivers for receiver are designated as R1 and R2. The connection of MAX232 with computer and the controller is shown in the circuit diagram.



# Serial Communication

## UARTn Transmit Holding Register (U0THR)

The UnTHR is the top byte of the UARTn TX FIFO. The top byte is the newest character in the TX FIFO and can be written via the bus interface. The Divisor Latch Access Bit (DLAB) in UnLCR must be zero in order to access the UnTHR.

Bit	Symbol	Description
7:0	THR	Writing to the UARTn Transmit Holding Register causes the data to be stored in the UARTn transmit FIFO. The byte will be sent when it reaches the bottom of the FIFO and the transmitter is available.
31:8	-	Reserved, user software should not write ones to reserved bits.

## UARTn Receiver Buffer Register (U0RBR)

The UnRBR is the top byte of the UARTn Rx FIFO. The top byte of the Rx FIFO contains the oldest character received and can be read via the bus interface. The Divisor Latch Access Bit (DLAB) in LCR must be zero in order to access the UnRBR.

Bit	Symbol	Description
7:0	RBR	The UARTn Receiver Buffer Register contains the oldest received byte in the UARTn Rx FIFO.
31:8	-	Reserved, the value read from a reserved bit is not defined.

# Serial Communication

## UARTn FIFO Control Register (U0FCR)

Bit	Symbol	Value	Description
0	FIFO Enable	0	UARTn FIFOs are disabled. Must not be used in the application.
		1	Active high enable for both UARTn Rx and TX FIFOs and UnFCR[7:1] access. This bit must be set for proper UART operation. Any transition on this bit will automatically clear the related UART FIFOs.
1	RX FIFO Reset	0	No impact on either of UARTn FIFOs.
		1	Writing a logic 1 to UnFCR[1] will clear all bytes in UARTn Rx FIFO, reset the pointer logic. This bit is self-clearing.
2	TX FIFO Reset	0	No impact on either of UARTn FIFOs.
		1	Writing a logic 1 to UnFCR[2] will clear all bytes in UARTn TX FIFO, reset the pointer logic. This bit is self-clearing.
3	DMA Mode Select		When the FIFO enable bit (bit 0 of this register) is set, this bit selects the DMA mode. See <a href="#">Section 14.4.6.1</a> .
5:4	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.
7:6	RX Trigger Level		These two bits determine how many receiver UARTn FIFO characters must be written before an interrupt or DMA request is activated.
		00	Trigger level 0 (1 character or 0x01)
		01	Trigger level 1 (4 characters or 0x04)
		10	Trigger level 2 (8 characters or 0x08)
		11	Trigger level 3 (14 characters or 0x0E)

# Serial Communication

## UARTn Interrupt Enable Register (U0IER)

Bit	Symbol	Value	Description
0	RBR Interrupt Enable	0	Enables the Receive Data Available interrupt for UARTn. It also controls the Character Receive Time-out interrupt.
		0	Disable the RDA interrupts.
		1	Enable the RDA interrupts.
1	THRE Interrupt Enable	0	Enables the THRE interrupt for UARTn. The status of this can be read from UnLSR[5].
		0	Disable the THRE interrupts.
		1	Enable the THRE interrupts.
2	RX Line Status Interrupt Enable	0	Enables the UARTn RX line status interrupts. The status of this interrupt can be read from UnLSR[4:1].
		0	Disable the RX line status interrupts.
		1	Enable the RX line status interrupts.
7:3	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.
8	ABEOIntEn	0	Enables the end of auto-baud interrupt.
		0	Disable end of auto-baud Interrupt.
		1	Enable end of auto-baud Interrupt.
9	ABTOIntEn	0	Enables the auto-baud time-out interrupt.
		0	Disable auto-baud time-out Interrupt.
		1	Enable auto-baud time-out Interrupt.

# Serial Communication

## UARTn Interrupt Identification Register (U0IIR)

Bit	Symbol	Value	Description
0	IntStatus		Interrupt status. Note that UnIIR[0] is active low. The pending interrupt can be determined by evaluating UnIIR[3:1].
		0	At least one interrupt is pending.
		1	No interrupt is pending.
3:1	IntId		Interrupt identification. UnIER[3:1] identifies an interrupt corresponding to the UARTn Rx or TX FIFO. All other combinations of UnIER[3:1] not listed below are reserved (000,100,101,111).
		011	1 - Receive Line Status (RLS).
		010	2a - Receive Data Available (RDA).
		110	2b - Character Time-out Indicator (CTI).
		001	3 - THRE Interrupt
5:4	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.
7:6	FIFO Enable		Copies of UnFCR[0].
8	ABEOInt		End of auto-baud interrupt. True if auto-baud has finished successfully and interrupt is enabled.
9	ABTOInt		Auto-baud time-out interrupt. True if auto-baud has timed out and interrupt is enabled.

# Serial Communication

## UARTn Divisor Latch

The UARTn Divisor Latch is part of the UARTn Baud Rate Generator and holds the value used, along with the Fractional Divider, to divide the PCLK in order to produce the baud rate clock.

---

The UnDLL and UnDLM registers together form a 16-bit divisor where UnDLL contains the lower 8 bits of the divisor and UnDLM contains the higher 8 bits of the divisor. A 0x0000 value is treated like a 0x0001 value as division by zero is not allowed.

The Divisor Latch Access Bit (DLAB) in UnLCR must be 1 in order to access the UARTn Divisor Latches.

### UARTn Divisor Latch LSB register (U0DLL)

7:0	DLLSB	The UARTn Divisor Latch LSB Register, along with the UnDLM register, determines the baud rate of the UARTn.
31:8	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.

### UARTn Divisor Latch MSB register (U0DLM)

7:0	DLMSB	The UARTn Divisor Latch MSB Register, along with the U0DLL register, determines the baud rate of the UARTn.
31:8	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.

# Serial Communication

## UARTn Fractional Divider Register (U0FDR)

Fractional Divider Register (U0/2/3FDR) controls the clock pre-scaler for the baud rate generation

Bit	Function	Value	Description
3:0	DIVADDVAL	0	Baud-rate generation pre-scaler divisor value. If this field is 0, fractional baud-rate generator will not impact the UARTn baudrate.
7:4	MULVAL	1	Baud-rate pre-scaler multiplier value. This field must be greater or equal 1 for UARTn to operate properly, regardless of whether the fractional baud-rate generator is used or not.
31:8	-	NA	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.

$$UARTn_{baudrate} = \frac{PCLK}{16 \times (256 \times UnDLM + UnDLL) \times \left( I + \frac{DivAddVal}{MulVal} \right)}$$

# Serial Communication

## Program to receive a character and send it back

```
#include<LPC17xx.h>
unsigned char rx0_flag=0, tx0_flag=0;
int main(void)
{
    UART0_Init();
    while(1)
    {
        while(rx0_flag == 0x00);
        rx0_flag = 0x00;
        LPC_UART0->THR = recv_data;
        while(tx0_flag == 0x00);
        tx0_flag = 0x00;
    }
}
void UART0_Init(void)
{
    LPC_PINCON->PINSEL0 |= 0x00000050; //P0.2, TxD0(Fn 01) and P0.3 , RxD0 (Fn 01)
    LPC_UART0->LCR = 0x00000083;      //enable divisor latch, parity disable, 1 stop bit, 8bit word length
    LPC_UART0->DLM = 0X00;
    LPC_UART0->DLL = 0x13;           //select baud rate 9600 bps
    LPC_UART0->LCR = 0X00000003;
    LPC_UART0->FCR = 0x07; //FIFO enable, Reset
    LPC_UART0->IER = 0X03;          //select Transmit and receive interrupt
    NVIC_EnableIRQ(UART0_IRQn);     //Assigning channel
}
```

# Serial Communication

## Program to receive a character and send it back

```
void UART0_IRQHandler(void)
{
    unsigned long Int_Stat;
    Int_Stat = LPC_UART0->IIR;          //reading the data from interrupt identification register
    Int_Stat = Int_Stat & 0x06;          //masking other than Transmit& Receive data indicator

    if((Int_Stat & 0x02)== 0x02) //transmit interrupt
        tx0_flag = 0xff;

    else if( (Int_Stat & 0x04) == 0x04) //receive data available
    {
        recv_data = LPC_UART0->RBR;
        rx0_flag = 0xff;
    }
}
```