

EMBEDDED SYSTEMS -

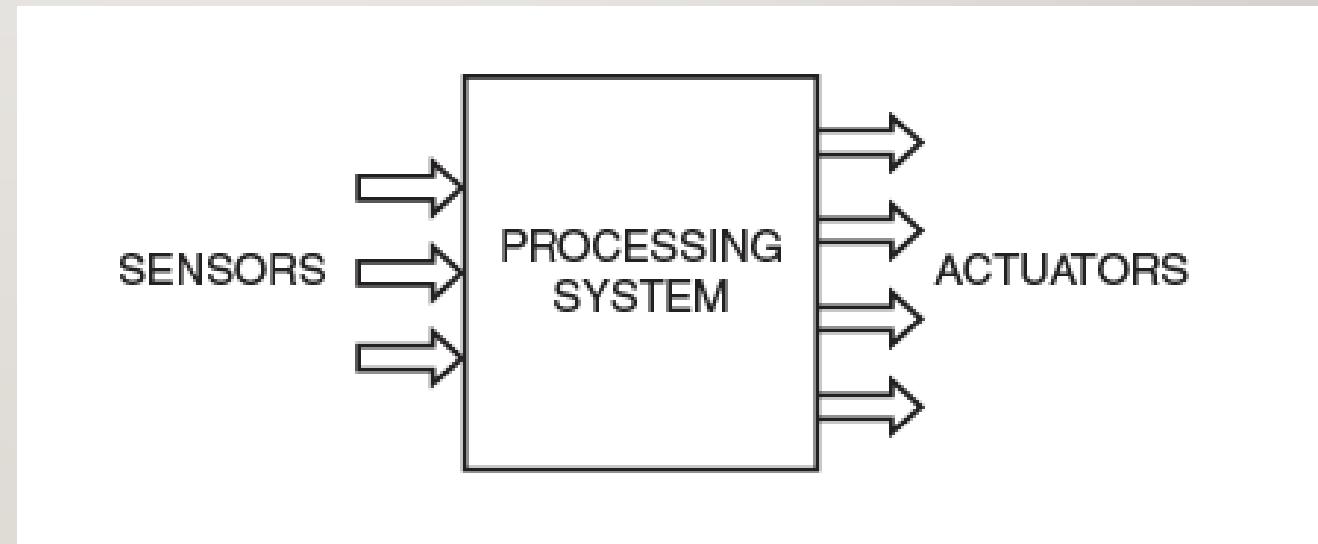
AN INTRODUCTION

I. INTRODUCTION

- Today World is becoming more and more technology enabled and so the word “Embedded Systems” becoming more familiar and widely used.
- Embedded systems are the one that uses electronics along with sensors and actuators.
- So, the world of embedded systems encompasses almost all the technological marvels we see today.
- We can even say, any high-end machinery that has electronics control is considered to be part of the world embedded systems, just as a small hand-held electronic device.

DEFINITION OF EMBEDDED SYSTEMS

Embedded system as any electronic-based system which takes in information through sensors, and produces an output actuation.



MODEL OF AN EMBEDDED SYSTEM

CHARACTERISTICS OF EMBEDDED SYSTEMS

1. They have a set of specialized functions. Each such system is meant for a particular task or set of tasks. This task set is not alterable later.
2. Any electronic device of now has a processing unit which takes in sensor data, process it and then produces signals, which does some actuation. The “processing” is usually done by software.
3. Thus, essentially an embedded system has an electronics hardware with software embedded into it. This makes the processing unit to be a microprocessor.
4. To get the sensor data, input peripherals are needed, and for output actuation, output peripherals are needed. Adding such peripherals to a microprocessor, makes the processing unit to become a Microcontroller unit (MCU). We also use as SoC (System on Chip).

CHARACTERISTICS OF EMBEDDED SYSTEMS CONTINUED ...

5. Many of the are part of a bigger systems- for example, washing machines, refrigerators, automobiles etc. are all controlled by embedded electronics.
6. Many of them have to operate and produce results within a stipulated time. Thus, we say that some of them need to do time critical computations.
7. Application wise- the domain is very fast. Some of them are placed in inaccessible areas, such as forests, some in extreme climatic conditions like case of military and space applications. Some will be very simple, as Temperature sensors, pedometers etc..

II. DESIGN OF AN EMBEDDED SYSTEM

Having the view of embedded system, we understood that the design may be constrained by the following factors:

- 1) The computing capability of the MCU used.
- 2) The power dissipation allowed
- 3) The size of the device
- 4) The memory (RAM and ROM) capacity of the MCU.



-
- In view of design constraints involved, the smart phone is considered to be an embedded system as it is designed under the limited computing capacity of its processor
 - Very low power dissipation allowed, small size and limited size of memory.
 - But, personal computer is relatively lesser constraints and is not considered as a part of embedded systems.
 - It can be considered to be a general purpose system which can have a very powerful general purpose processor can afford to use much more power and does not need to have as much constraints with respect to its size or memory capacity.

III. APPLICATIONS

Lets have some few examples for Embedded Systems

1. **Mobile Phones & Tablets:** The range of smartphones and tablets available in the market is very large. The design of these devices satisfy all the criteria for being labelled as embedded systems
2. **Consumer electronics and Household Appliances:** This is also a very important category. Cameras, Music Players, DVD players, remote controls, washing machines, refrigerators and many more.

APPLICATIONS CONTD....

3. **Automotive Controls:** This is the largest application field for embedded systems. It has electronic controls for engine, fuel system, windows, door, etc.. More advanced automobiles have navigation, parking assistance, cruise control, driverless cars.
4. Other fields applications are banking (ATMs, Currency Counters) Aviation and Military, Toys and Robotics.

IV. EMBEDDED PROCESSORS

- Got some general idea about the embedded systems.
- Now look into the important component of Embedded processor.
- When peripherals are integrated into the same package, the processing unit is usually called as **MCU (Microcontroller Control Unit)**.
- Many MCU's are available that are classified based on their data bus widths.
- Some of the MCU names will be discussed based on their usage of specific applications and few are mentioned here.

MCUs named are:

- * **8 Bit** : 8051, PIC and the AVR series are a few of the MCUs with 8-bit data bus width. They are popular in the academic field and for applications which do not need high-level computations.
- * **16 Bit** : PIC and AVR series have 16-Bit chips as well. A very low power 16-Bits MCU is the MSP 430 which has become very popular in medical and other low power devices.
- * **32 and 64 Bit**: ARM is the most popular of the 32-bit MCUs, though there are many 32 PIC versions as well. Newest version of ARM for advanced applications are all 64 bit. Intel has developed 32 and 64 bit Atom and Quark processors that have started incorporating embedded systems.

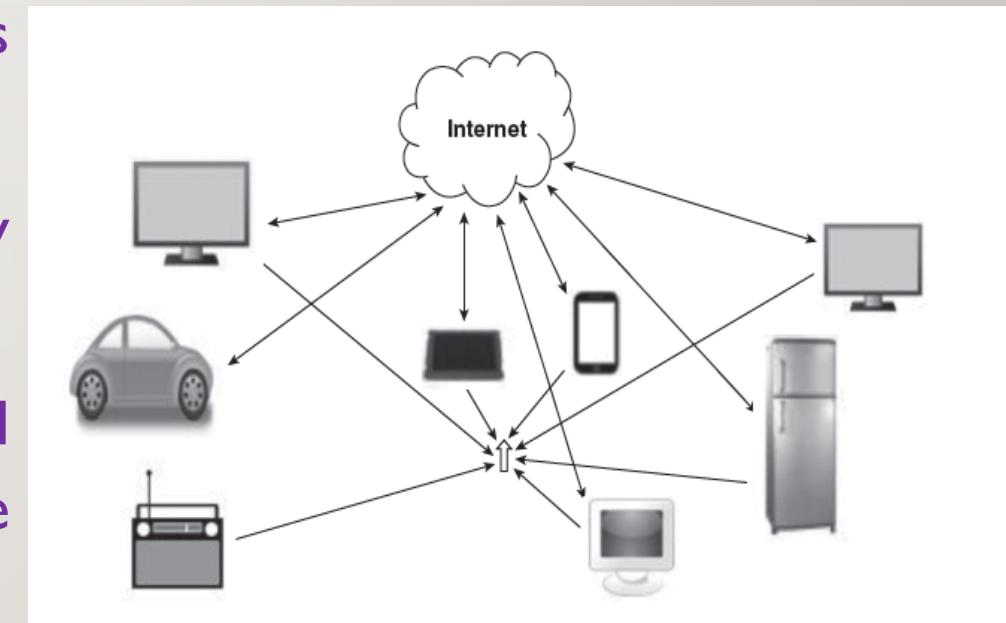
V. OPERATING SYSTEM (OS)

- We all know that our phones has OS.
- One question comes to mind is, What is the role of OS here ?
- An OS is the software which acts as the manager of the system.
- As phone does many tasks, managing the task set efficiently by assigning priorities where necessary and avoiding conflicts is the role of the manager and OS does this efficiently.
- Such OS are embedded OSes and they have to be small enough to reside inside the small ROM space available in the device.

-
- Some embedded applications are time critical, such as ABS (Anti-Braking System) in cars.
 - Even soft drink vending machine works on deadline, which the application has to produce the correct result.
 - If not, we say that the system failure has occurred.
 - In order to manage that, it needs RTOS (Real Time Operating System) which imposes and ensures the timing criterion.
 - An RTOS uses deadline-based task scheduling algorithm.
 - So, most of the embedded systems around us is managed and controlled by RTOS.

VI. CONNECTIVITY

- We live in a connected world where we use various protocols for communications.
- Next is already on its way is the idea of connectivity between devices without human intervention.
- The expectation is all devices will be connected through the internet and that many functions will be able to function efficiently without human intervention.



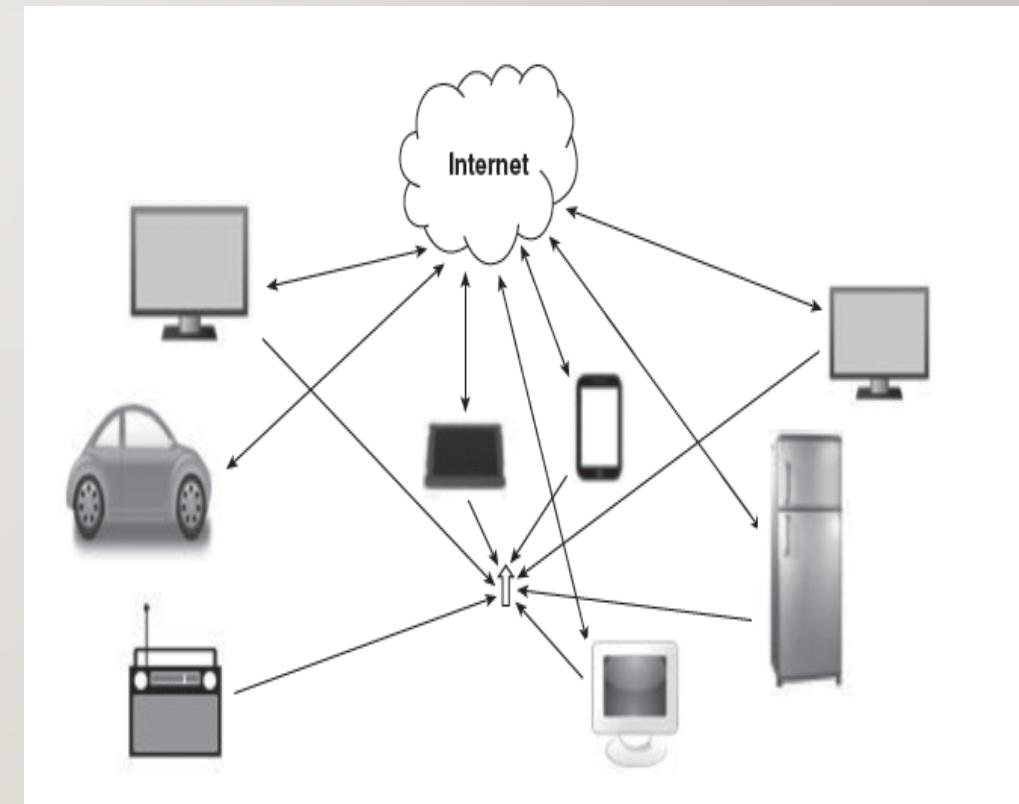
This expectation is the motivation behind the idea named Internet of Things (IoT)

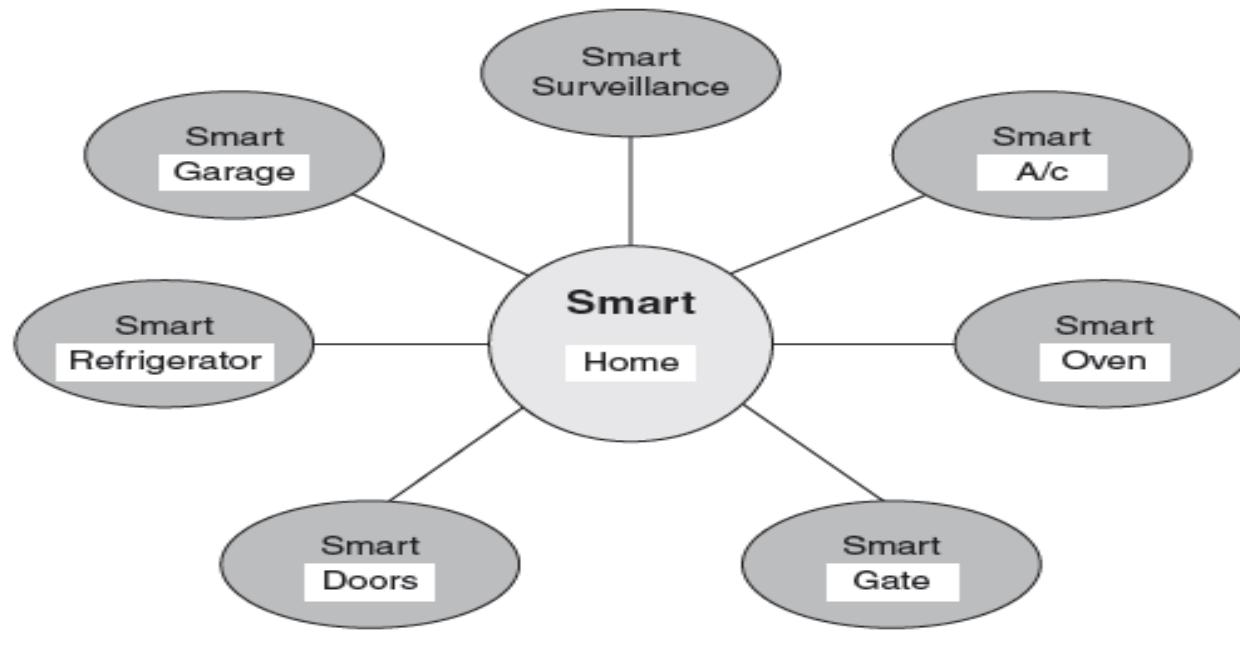
INTERNET OF THINGS (IOT)

- IoT is the scheme of things where devices are connected to the internet and sensor data may be uploaded to the web.
- Later these data can be used by the system to make actuations, ideally, without human intervention.
- Smart homes, smart cities, smart cars, smart factories etc. are some of the examples IoT
- All the IoT based devices need to internet connectivity and so Wi-Fi and/or Ethernet protocols will become necessity.

THE MOST IMPORTANT FEATURES OF IOT-BASED SYSTEMS MAY BE LISTED AS:

- Sensing & Monitoring
- Decision making based on input data from many sensors
- Network connectivity
- Data storage and computation in the 'cloud'





- Shows components of a 'smart home' in which many appliances and facilities may be automated by the IoT concept.
- Besides Internet protocols, other short range protocols like: Bluetooth classic, BLE (Bluetooth Low Energy), Zigbee, NFC (Near Field Communication) etc. are also accessories in the IoT revolution.
- To make the IoT dream a reality 'Cloud Computing' also needs to be more prevalent.

CLOUD & ITS REALIZATION

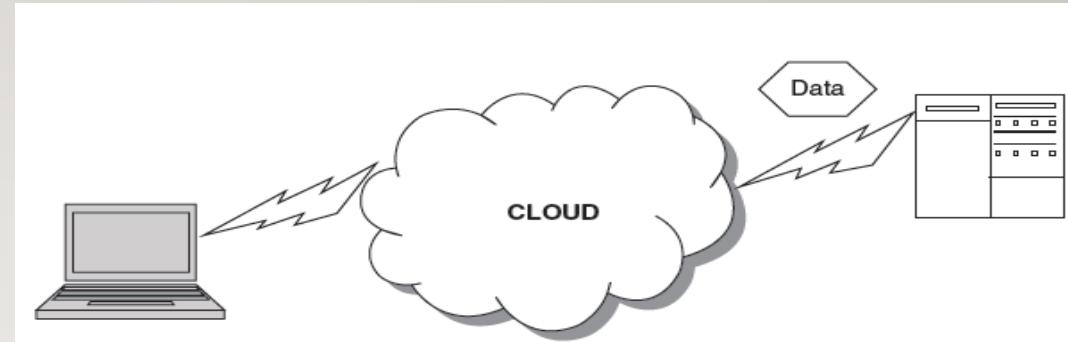


Figure 1.5 Visualization of cloud and its storage

- During the talk of IoT, we could see that an immense amount of data is generated from the sensors that are deployed.
- Where could all this data be stored ?
- Where could computation and analytics using this data be done ?
- For all this, its none other than 'Cloud'.
- In 1990's it started as a metaphor for internet and later internet services began to be represented by the word cloud.

-
- The cloud is a ‘storage and computational ‘ support provided by an array of networked computers.
 - If our local system does not have enough storage space, we save it in ‘cloud’
 - Likewise, if our local computing device does not have enough computational capability, we can seek the service of the ‘Cloud Computing Platform’.
 - Thus providing computational platform.
 - So cloud provides both Computational & Storage space support.

CONCLUSION

- Hence, we see that world of embedded systems is vast and it is making its entry into every aspect of our lives.
- The future of this field is very promising and exiting.

UNIT- 1

Chapter- 2

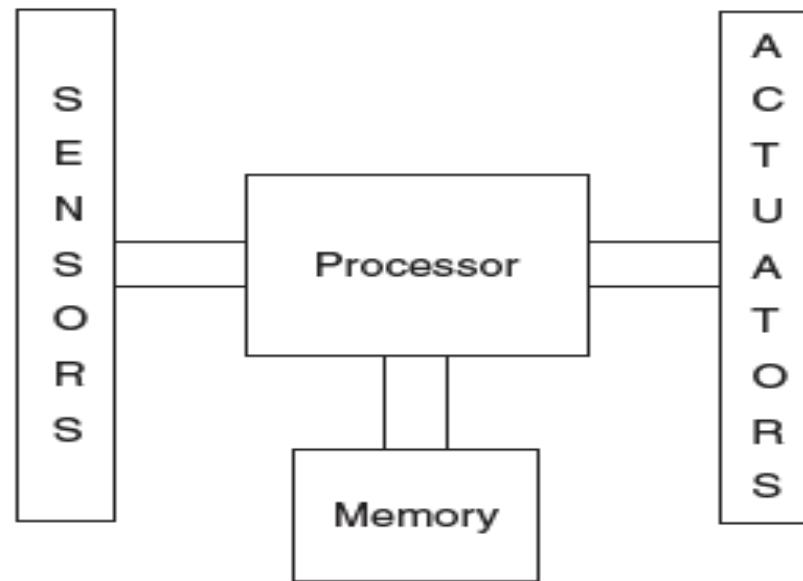
INTERNAL COMPONENTS OF A SYSTEM-ON-CHIP (SoC)

INTRODUCTION

- Previously we learnt about the very brief on the concepts of Embedded World.
- Now lets understand on the finer detail of all that we have perceived from outside.
- More focus will be towards the typical microcontroller units and its Internal units.
- Also look at the new trends that are takin place because having systems which are connected to the internet are becoming a necessity.

2.1 GENERAL MICROCONTROLLER UNIT

- Here, more focus is on hardware aspects of an embedded system.
- Observe the block diagram :

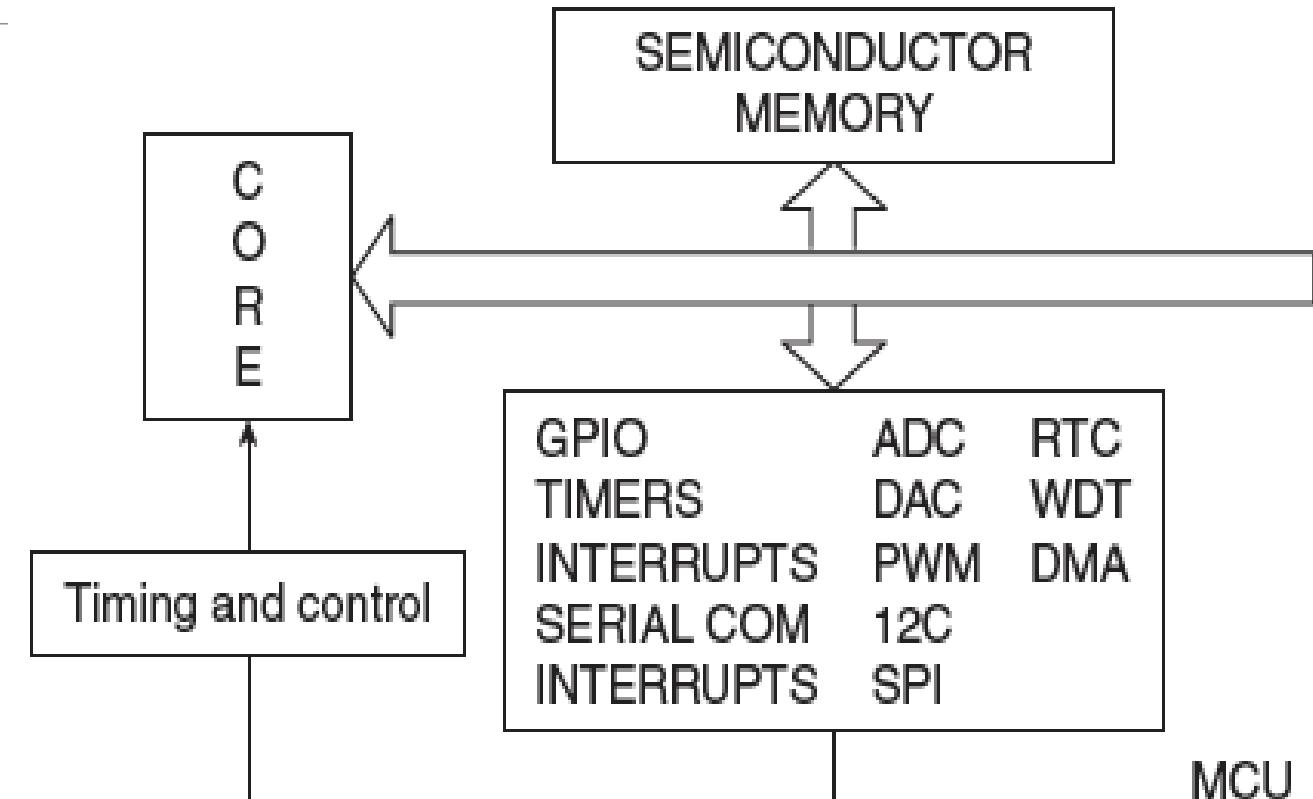


Model of a simple embedded system

- Embedded system has a computing unit to which sensors and actuators are connected.

- Here more focus on MCU which is a ‘Single Chip Computer’ and is a heart of embedded system.
- What is an MCU?
- MCU is a computing engine, that is , a CPU, to which a peripheral controllers and memory are added.
- It’s also known as ‘System on Chip’.
- Peripherals: input and output devices which are needed for system.
- Eg: Keyboard as Input and LCD Screen as Output peripheral.
- As these are outside to MCU.
- Some are ‘Controllers’ that are inside the chip.
- Peripheral controllers are dedicated hardware units, that provides control signals for the MCU to communicate with actual peripheral devices.

- So, keyboard and LCD are easily connected and can reside on MCU.
 - Some of the other functional units that also reside on MCU are: A to D convertor, UARTs, Timers etc.
 - We say all are peripherals.
-
- Conceptual Internal diagram of an MCU chip.
 - CORE: CPU or computing engine
 - Memory and a set of peripherals interconnected by a system bus.
 - These peripherals are found in most commonly in all MCU's.
 - Advanced peripherals are found in more advanced system and all work on a basis of time synchronized manner, due to timers and control units.

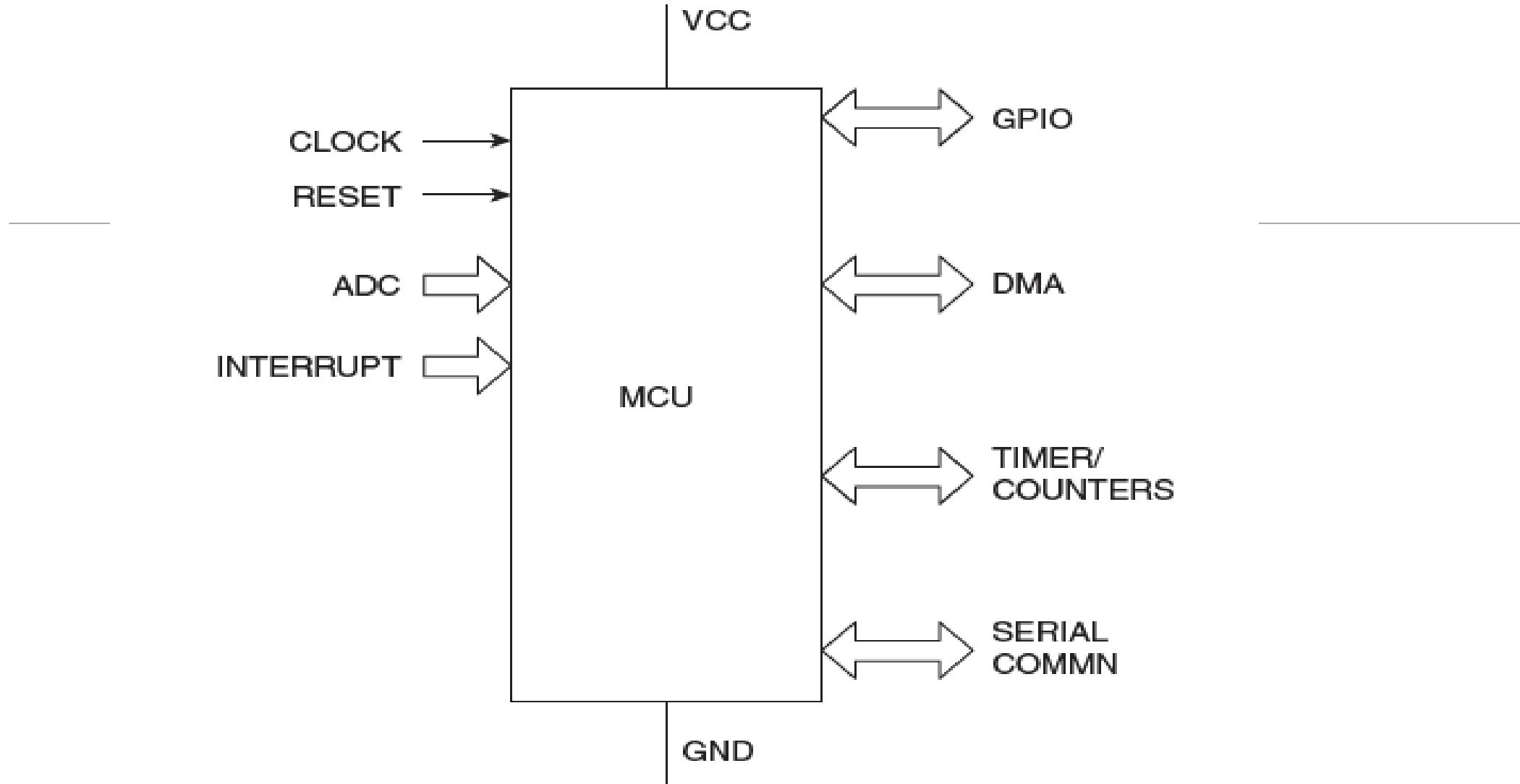


Internal Components of a Typical MCU

- Internal bus shown here is mainly consists of basic address, data and control bus.
- If the data bus has 16 bits, it means that 16 bits can be handled at a go by the CPU and the word length of the general purpose register in the core is also 16.
- The width of the internal address bus fixes an upper limit on the capacity of the internal memory that can be implemented.
- If it is of 16 bits, the maximum memory capacity is 64 KB
- If this bus is 32-Bits wide, the maximum addressable memory is 4 GB
- **Taking only the reference of MCU, understand the details of these internal units.**

2.2 PIN DIAGRAM

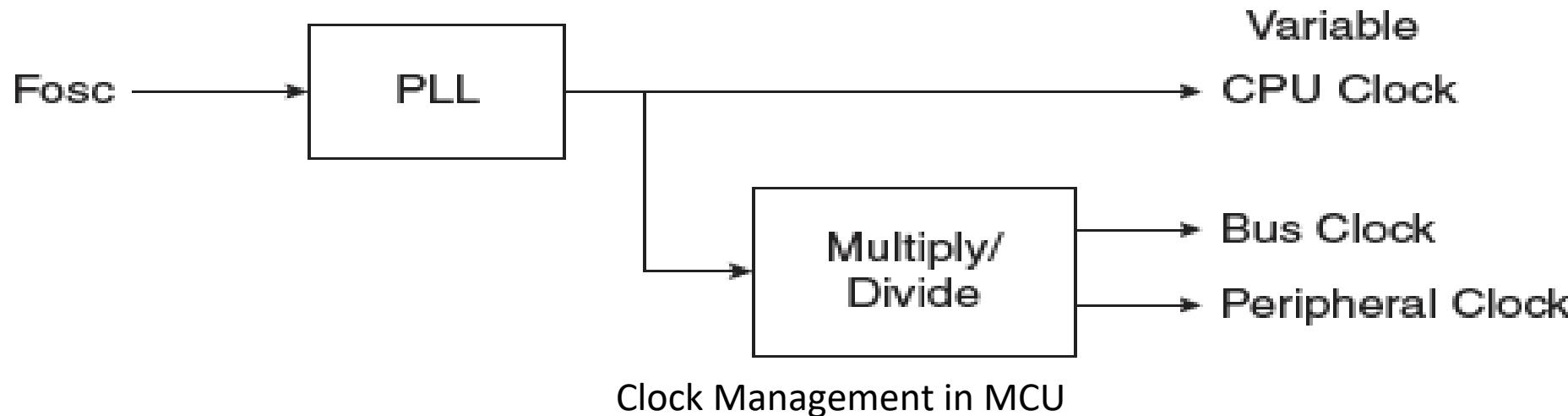
- Number of pins of an MCU depends on the number of peripherals it has.
- All peripherals has pins to send and receive signals to/from external devices.
- These pins are called I/O pins.
- In most of the MCU's , it has more than one function for a pin, such that the exact function can be configured as per the one.



Functional PIN Diagram of a MCU showing set of pins for each of the peripherals

2.2.1 Clock

- Know that MCU is SoC
- Every system is synchronized with a clock, designated as the system clock.
- The frequency of the clock in this system is to measure how fast the system can perform its functions.
- With MCU, CORE, peripherals and the BUS which interconnect them.
- All will have the same clock OR each of the has a separate clock, so it seems to be complex.
- Means it can have system clock, core clock, bus clock, peripheral clock- each of the operating in different frequencies.



Clock Management in MCU

- Referring to above picture, gives an idea about clock management for a typical system
- Basically, a crystal is used for clock systems.
- Most modern MCU, have the options of using either an external frequencies or the output of an internal oscillator as a basic clock.
- MCU has a circuitry, that supplies different frequencies for different part of the system.
- As frequency greater than 100 MHz is needed, a phase locked loop (PLL) is used as it can generate different frequencies.
- PLL can also change the CPU clock frequency dynamically for power management.

What are the orders of clock frequencies generally used in embedded systems?

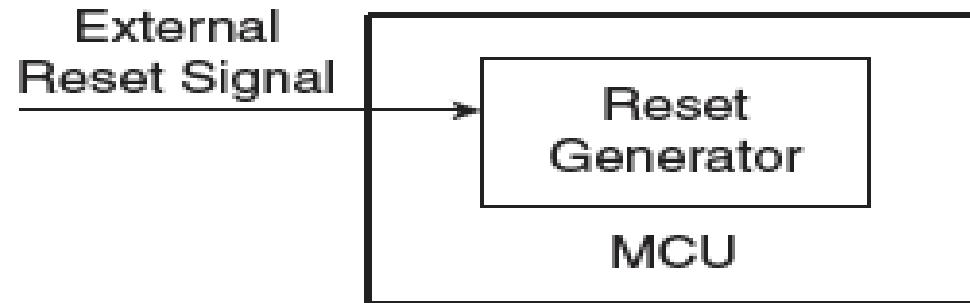
Embedded systems do not use very high frequencies of the order of 3GHz that we see in the latest PCs. Each embedded device has its specific functions and many applications require low frequencies. Thus, core clock frequencies can be as low as 10MHz and peripherals may operate at much lower rates. High-end microcontrollers have frequencies in the range of 60-80 MHz, on the very high end side, there are mobile phones and tablets which use frequencies of around 2 GHz.

2.2.2 Reset

- Word Reset means “Restart”.
- Systems are expected to start from the known state.
- As MCU is reset, internal registers are cleared and the first instruction to be executed is taken from a specific address.
- Hence, program counter should be loaded with that address.

Question: How is Reset Done?

- Any MCU chip, there is a reset pin at which an external circuit may be connected.



- Above figure, shows that there is an internal circuitry also, named as “Reset Generator”.
- That takes care of the reset function.
- There are 3 different kinds of reset options.

2.2.2.1 Power-on-reset

- As power is first supplied to circuit, system must start from a known state.
- This is the state the MCU is in, every time it is powered on.
- The same state must also be reached if a hard reset is done by a push button switch any time during the operation of the circuit.
- Though an internal reset generator, signal from external circuit is mandatorily needed at the reset pin.
- This signal should appear at the reset pin whenever the power supply is switched on.
- This signal is termed a pulse that should remain at the reset pin.
- Resetting follow the circuit similar to Schmitt Trigger.

2.2.2.2 Brown out reset

- When the power supply goes below a certain pre-fixed value
- “ Brown out” will occur
- Especially battery operated devices, its quite common high.
- When brownout occurs, the MCU starts behaving abnormally and gives wrong results.
- Leads to cause of catastrophic situations.
- To avoid this, all MCU's have the capability of reset, as it reaches brownout threshold.
- For any MCU's not having internal brownout capability, external circuit can be used to detect brownout and cause a reset to occur.

2.2.2.3 Watch dog reset

- Most of the embedded systems are unsupervised and can go into infinite loops due to noise signals.
- To overcome this, in MCU there should be a ‘Watch Dog’ which initiates a ‘reset’ on break out of the infinite loop.
- This type of reset is also called a soft reset or a warm boot.

2.3.1 Register of the Core

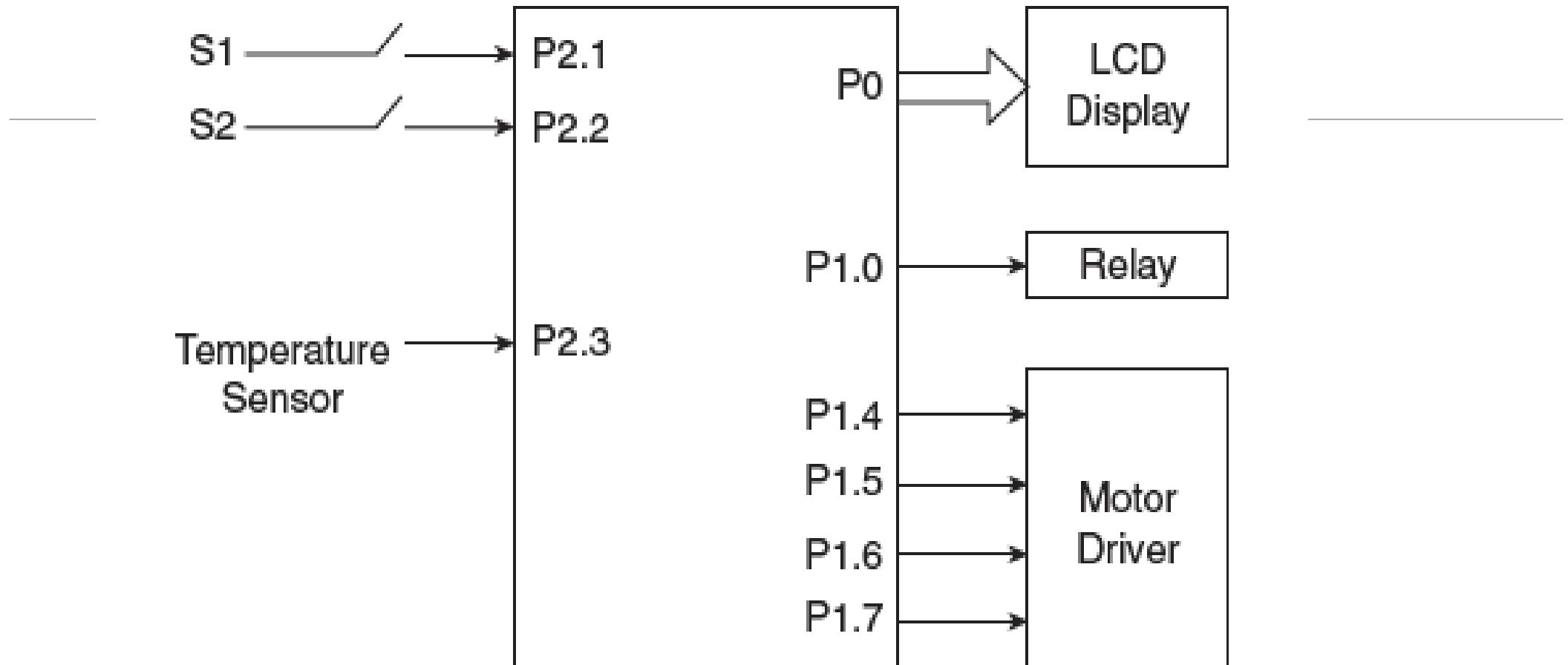
- Microcontroller has a CPU which we usually refer to core.
- As it is termed the Heart of the computing system.
- Core has registers, that are used for computations.
- These registers are termed as “General Purpose Registers” OR “Scratchpad Registers”.
- These registers provide temporary storage for operands in computations.
- As core in an MCU is full fledged CPU, it has several registers, including general purpose registers, program counters, stack registers, flags etc.
- This can be seen when ALP , shows the access to all its intricate behavioural aspects and registers.

2.3.2 Registers of Peripherals

- For any MCU, peripherals are inside the chip. Denoted as 'On-Chip' peripherals.
- Each such peripheral is a dedicated hardware.
- Means, while the processor is executing programs, many peripherals can work in parallel.
- For eg: take the case of a timer.
- The processor, through its instructions, only needs to tell a timer to start counting.
- Timer does not need any more intervention until it reaches the desired count.
- At that point, it will signal the processor by setting a flag or sending an 'Interrupt' and the processor then take any action.
- A number of peripherals run parallel, like ADC, DAC, Serial IO etc.

2.4.1 General Purpose I/O

- MCU has a computing engine for processing
- As it needs to communicate with external world, it needs input/output pins.
- Input pins are used to take in for sensor data for processing
- Output pins for sending actuation signals to output devices.
- Those pins are called GPIO or “General Purpose Input Output pins”
- Pins can also be taken as groups or ports.
- Thus, an 8-bit MCU has 8-bit ports, with additional capability.

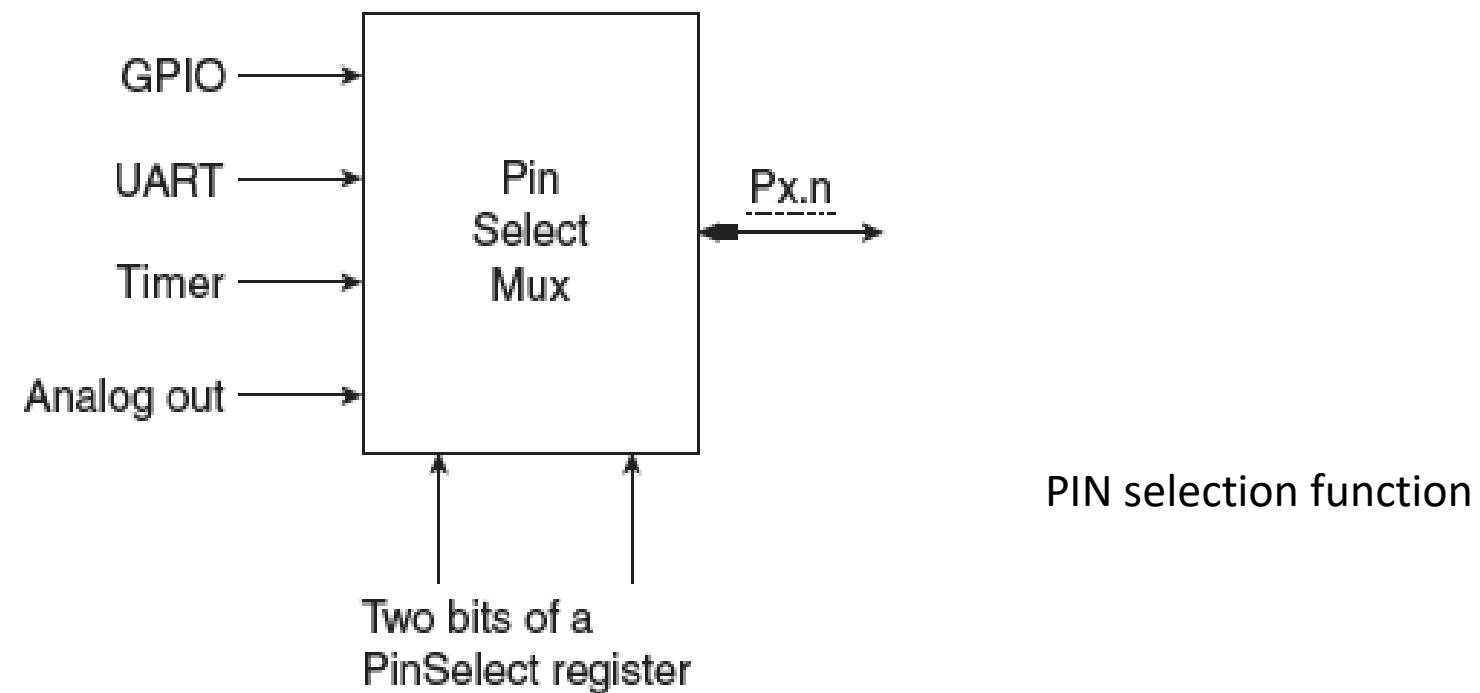


Using GPIO pins as Input & Output

Referring to previous diagram

- It shows a case of many GPIO pins used for various I/O functions.
- An 8-bit port is configured as output and is utilized for sending 8-bit data to an LCD display.
- one-bit pin is used for a relay
- 4-bit output pins for sending commands to a motor driver IC.
- Two-bit input pins are controlled by switches (S1 & S2)
- Another one takes sensor data from a temperature sensor.
- This illustrates that, it is possible to use GPIO pins as we need.

-
- In most MCUs, however, each pin has more than one function.



-
- Lets see a typical pin, Pn.x
 - It is shown to have perform four functions, but only one of which may be chosen at a time.
 - This choice selection happens on the basis of multiplexing called multiplexer.
 - The select bit of the multiplexer are realized by two bits in a ‘Pin Select’ register.
 - First option of pin is using as GPIO, where it can be used either input or output pin
 - It works on the basis of set/reset.

2.5.1 Data Transfer Modes

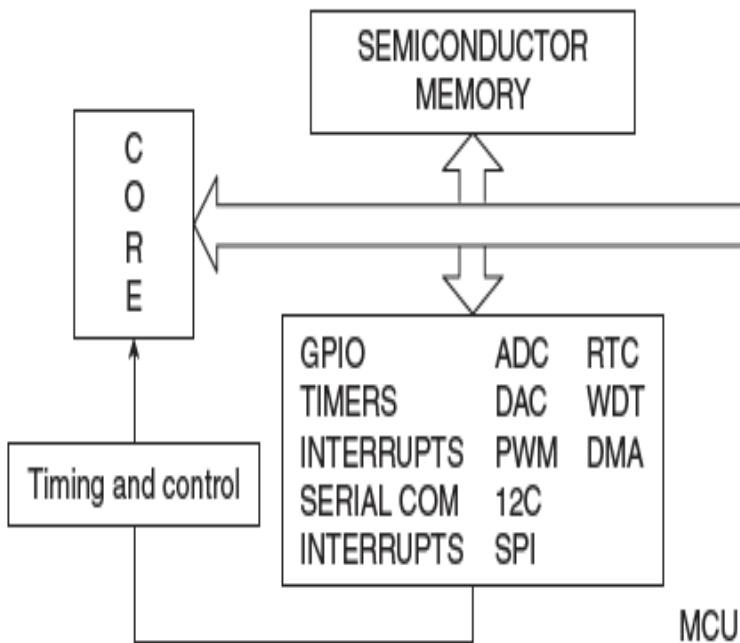
- Earlier diagram have shown us that sensor connected for input device and actuators as output devices in an embedded system.
- Lets see the case of a keyboard.
- How does it communicates with the MCU?
- To keep MCU 'in a loop' waiting for a key press.
- So MCU has to continuously monitor this action on one of its input pins.
- This method of accepting an input data is called 'Polling'.
- MCU will be unable to do anything else if it is continuously polling some pin, in this fashion.
- Because of inefficiency involved, polling for data is rarely used in practical systems.
- So we use interrupts.

2.5.2 Interrupts

- Word ‘Interrupt’ is very familiar to us, as always get interrupted while we are busy working something important.
- Similar in case of systems also.
- Interrupt is a method, that allows MCU to do any activity until it gets a signal from the keyboard that is been pressed.
- This signal from the keyboard is called an ‘Interrupt’.
- After receiving these signals, the MCU stops what it is doing and takes the action.
- If the processor decides to attend new task, it temporarily abandon the current task and takes up the new task, after completion, it resumes back.

-
- The processor executes a stream of instructions and the Program Counter (PC) is the register that sequences it.
 - At any point, PC points to the next instruction to be executed.
 - When an interrupt arrives, the process completes the current instruction and saves the context of the current task.
 - Here, important is , it has to save the address of the next instruction of this sequence

2.6.1 Timers and Counters



- Timer/Counter is one of the part of peripherals of MCU.
- This is a dedicated hardware for timing applications.
- Reference clock for this hardware is the peripheral clock.
- Which is a low frequency clock derived from the core clock.
- It acts as a 'Interval Timer' or as an event counter.
- Interval counter creates a delay and by using this delay, a waveform can be generated at any output pin.

Following are the steps involved in the working of an interval timer:

- 1) A timer count register is loaded with a number and the timer is started.
 - 2) The count increases/ decreases until it reaches the maximum/minimum count.
 - 3) At this point of time, the count register is cleared and a flag is set or an interrupt is triggered to signal the event of timer overflow.
- The time elapsed from the starting point, is the required delay.
 - This whole sequence can be repeated many times after re-loading the timer register.
 - If an output pin is toggled every time the timer register overflows, we get a symmetric square wave at the pin.

2.6.2 Event Counter

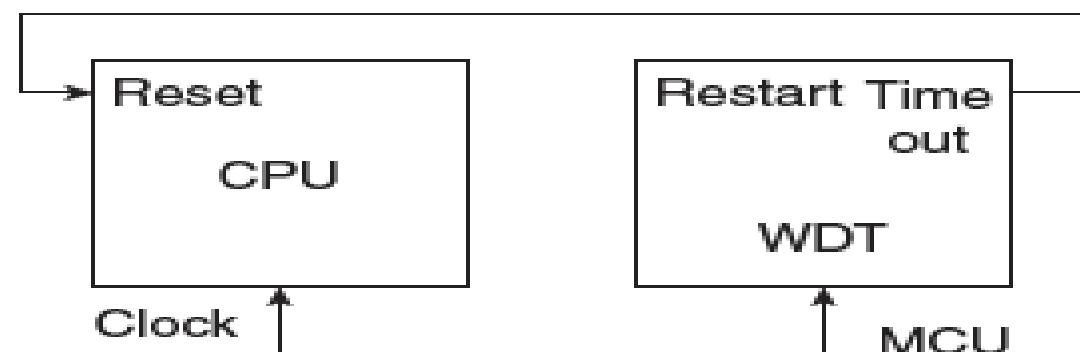
- The same hardware can be used for to count external events.
- Lets say we need to measure the frequency of pulse train.
- This frequency can be fed into the counter input pin.
- Instead of peripheral clock being the reference, this unknown frequency acts as the reference clock.
- Every time a leading or trailing edge is sensed at the counter pin, the timer count register count increases by one.

2.6.3 Watch Dog Timers

- This is an on-chip peripheral in advanced MCU's
- Low-end MCU's may not have it, and can be added if needed, as chips are available.
- Embedded systems have to sort out their problems on their own, as they don't allow user intervention.
- A WDT is provided to help a system in the event of an unresolvable anomaly in its working.
- This timer monitors the working of the MCU to make sure it does not go into endless loops.
- The point to note is that, when unexpected events do occur, the only way out of it is to restart the system and start all over again.
- What WDT does is to check whether such a RESET is necessary.

How does the WDT do this?

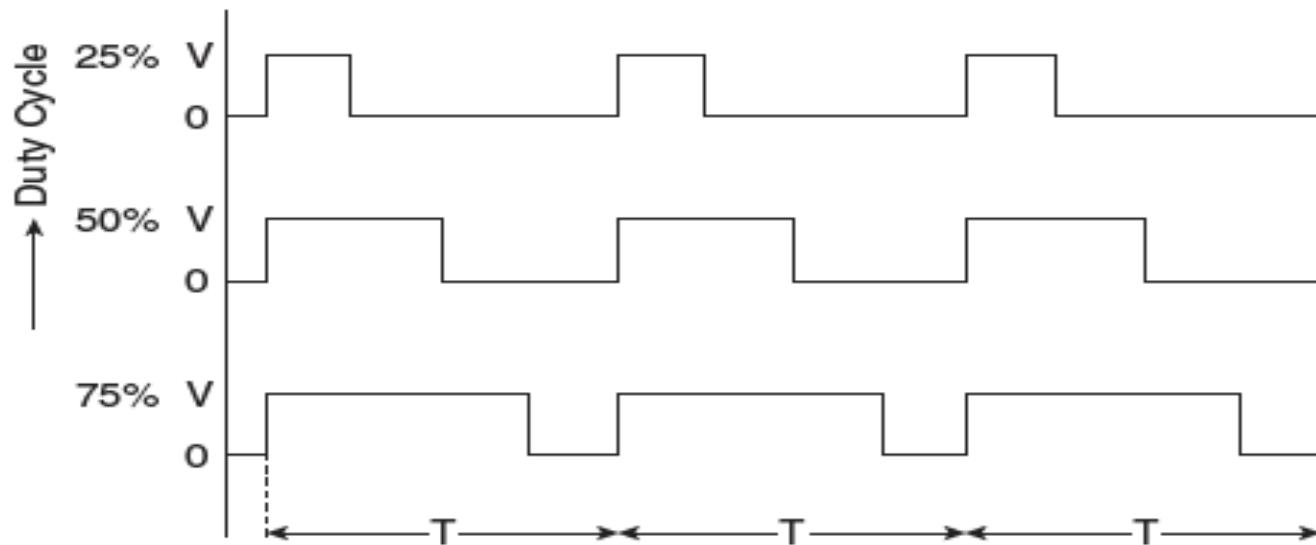
- It is like an ordinary timer
- Normally, it starts counting and before it reaches its terminal count, the internal mechanism restarts it.
- If the MCU stuck in the infinite loop, the 'Internal Mechanism' of restarting the WDT is blocked.
- So, the WDT counts down to zero and then it resets the whole MCU, so that the system recovers from its erroneous state.
- Count is decided in a WDT after considering how much time is to be allowed for the system to recover before it is forcibly reset



2.6.4 Real Time Clock

- A real time clock inside the MCU is for providing calendar functions- the precise time in hours, minutes, seconds and also date, required for various applications.
- All embedded systems need real time information (Human Time)
- Even though a timer can be programmed to count time, it is best if there is a dedicated hardware which can be programmed.
- using its own registers to count 'real time' and present it easily for display and other purposes.
- Such a clock can also provide the timing for operating system, as high-end embedded systems use.
- It should also have a battery backup, if the clock is to be kept running all the time.

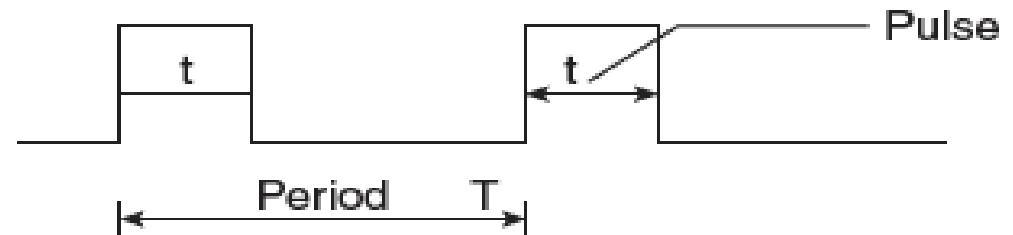
-
- Asymmetric waveforms can have varying duty cycles.



- When we are able to change the duty cycle of a square wave, we say that it is “Pulse Width Modulated”.

2.7 Pulse Width Modulator

- Considering its functionality, PWM (Pulse Width Modulator) module is timer.
- As it has specific and important applications, it is a separate module in MCUs.
- Lets see what an PWN waveform is:



- A symmetric wave has half of it period in the high or '1' state and the other half in the low or '0' state.
- So its duty cycle is 50% where the duty cycle is defined as follows:

Duty Cycle in % = (Time in the high state/Period) x 100

$$= (t/T) \times 100$$

$$= 50\% \text{ if } T = 2t$$

UNIT- 2

Chapter- 2

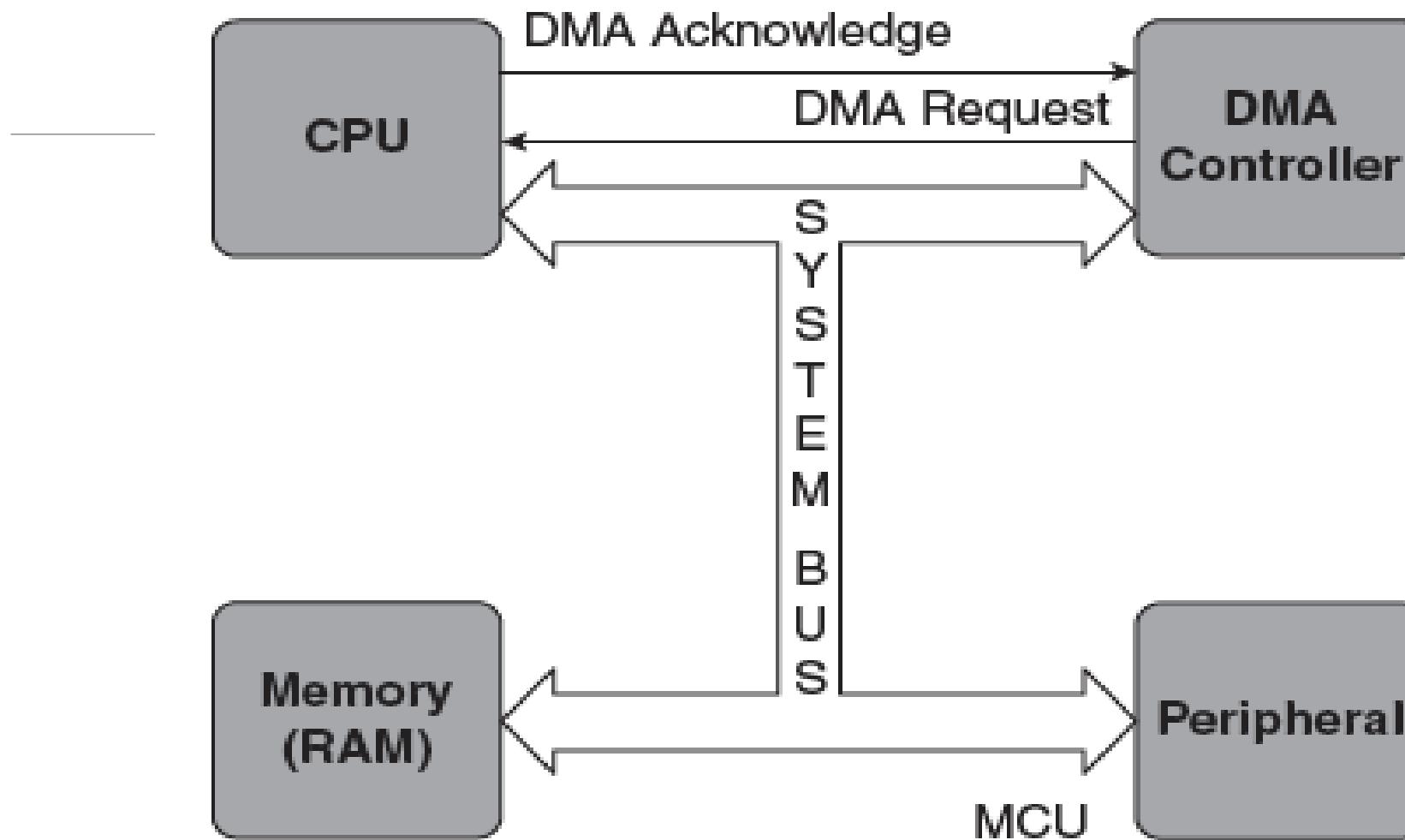
INTERNAL COMPONENTS OF A SYSTEM-ON-CHIP (SoC)

2.9 Serial Communication

- Data from the MCU can be sent or received in serial form through specific protocol.
- One important peripheral is the UART (Universal Asynchronous Receiver Transmitter)
- All MCU's have a UART module inside.
- A dedicated hardware for serial communication is based on RS232C protocol.
- The pins TxD and RxD are the serial transmit and receive pins.
- Many external modules like GSM and GPS use the UART pins for interfacing an MCU.

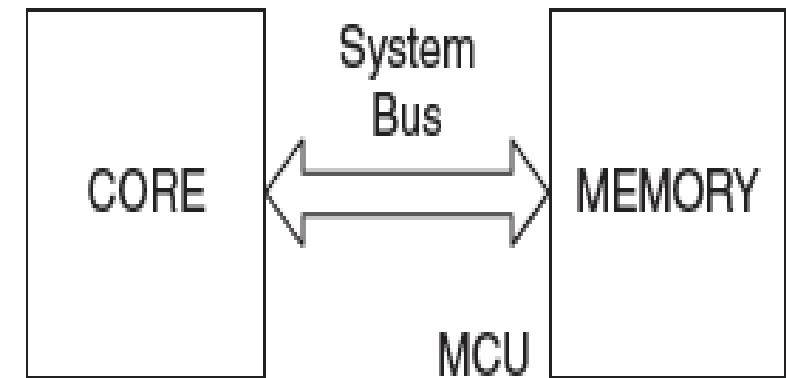
2.10 Direct Memory Access

- Basically, in a system when a data is to be moved between memory and external devices, it is channelled through the CPU registers. (For Normal Case)
- But, if a lot of data is involved, it is a waste of time and effort to use the CPU as an mediator.
- DMA is the scenario when the transfer of data is directly done between and I/O device and memory.
- This implies, that the connection to the CPU is blocked when DMA is being done.
- Many MCUs have the DMA controllers inside it and this unit is responsible for initiating and controlling DMA operations.
- When there is a need for DMA, this unit gives a 'DMA request' to the CPU, which responds by a 'DMA Acknowledge' signal.



2.12 Semiconductor Memory

- We have covered all the peripherals of typical MCU
- Lets see the other component of an MCU, Memory.



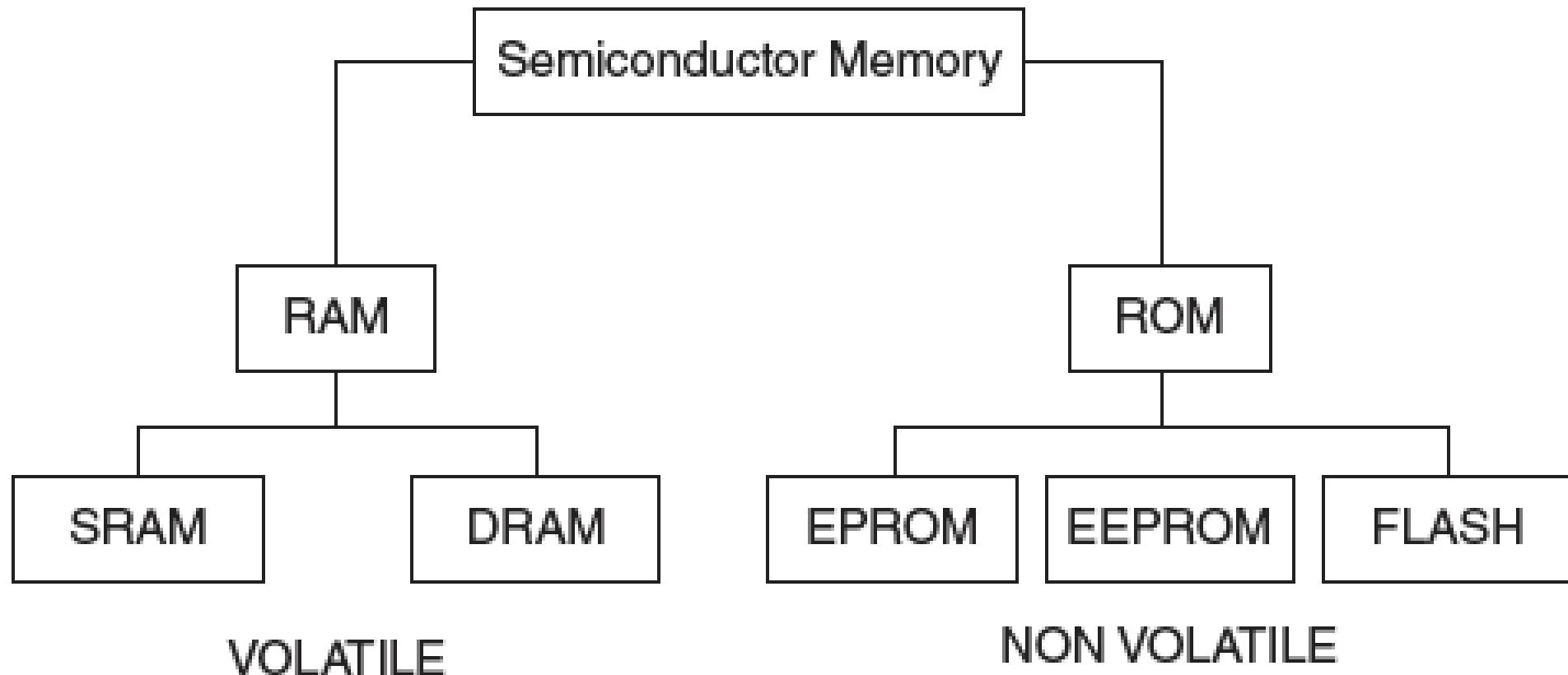
- Fig shows that inside MCU chip, the core (CPU) communicates with memory.
- The code and data need to be stored in memory and it is the prime activity in any computing system.
- Accessing data from memory by the core is termed as Reading/Writing.
- When data is written into the memory is called as 'Store'
- Whereas the data reading from it is called 'Load'.

-
- These operations take a certain amount of time depending on the type of technology used for memory.
 - The value of the 'access times' tells that a memory is either fast or slow, and this depends on the technology used.
 - So, in this section, let us discuss the different types of memory devices which are used in embedded applications.
 - Inside the memory hardware, each address stores one byte of data. It means that memory is generally 'byte oriented'
 - So if a 32-bit word is stored in memory, it occupies four bytes space and four addresses are used for it.

- In embedded systems, memory space is a resource which is to be used very carefully, because it is limited.
- For eg: let us consider an MCU in which 32 bits are allowed for the address bus.

- So there can 2^{32} bytes or 4 GB of memory at a maximum.
- This maximum capacity may not be fully realized inside the chip, because for many applications it may not be necessary and also because it takes chip area and increases the price of the chip.
- This memory is realized by many types of memory components.
- There will be RAM (Random Access memory) as well as ROM (Read Only memory).
- ROM is used to store code
- RAM is used as an intermediate area for computations.
- In MCUs, memory is usually inside the chip, but if more memory is needed, it is possible to have off-chip memory as well.

Different types of semiconductor memories which are useful for embedded systems



2.12.1 Random access memory (RAM)

Following are the distinct characteristics of RAM:

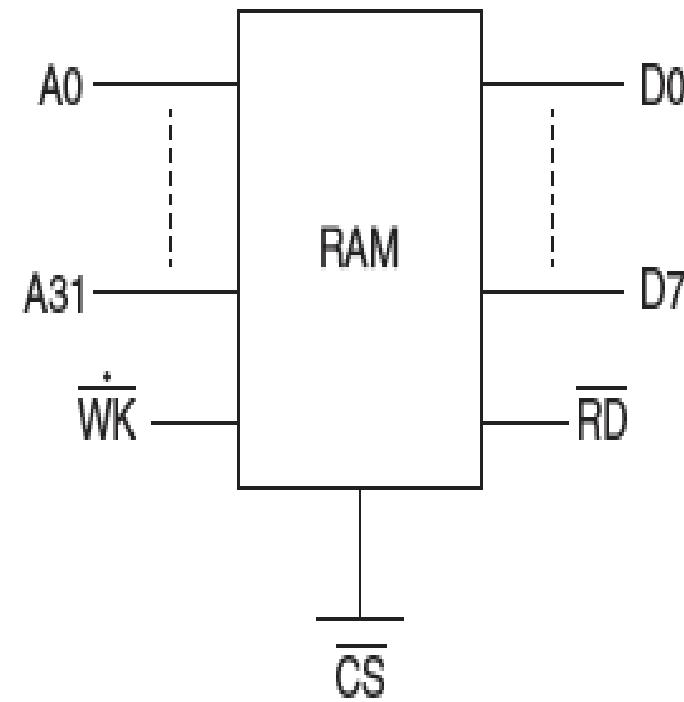
1. It is volatile, that is the contents are lost when power is switched off.
2. It can be read from and written to by program instructions, and so called a Read/Write memory.
3. The memory access time is the same irrespective of the location of the data we want to access — that is why it is called 'random access' in contrast to serial access in magnetic memory.

2.12.1.1 RAM technology

- RAM is realized by different technologies.
- The fastest type is SRAM which stands for Static RAM.
- In this type, storage is in the form of a voltage.
- So the content remains stable or static, as long as power supply is maintained.

2.12.1.2 SRAM chip

- SRAM chip with a 32-bit address bus and an 8-bit data bus and the control signals for reading and writing.
- There is also a *CS* (Chip Select) signal which ensures that this chip is accessed only after it is selected.

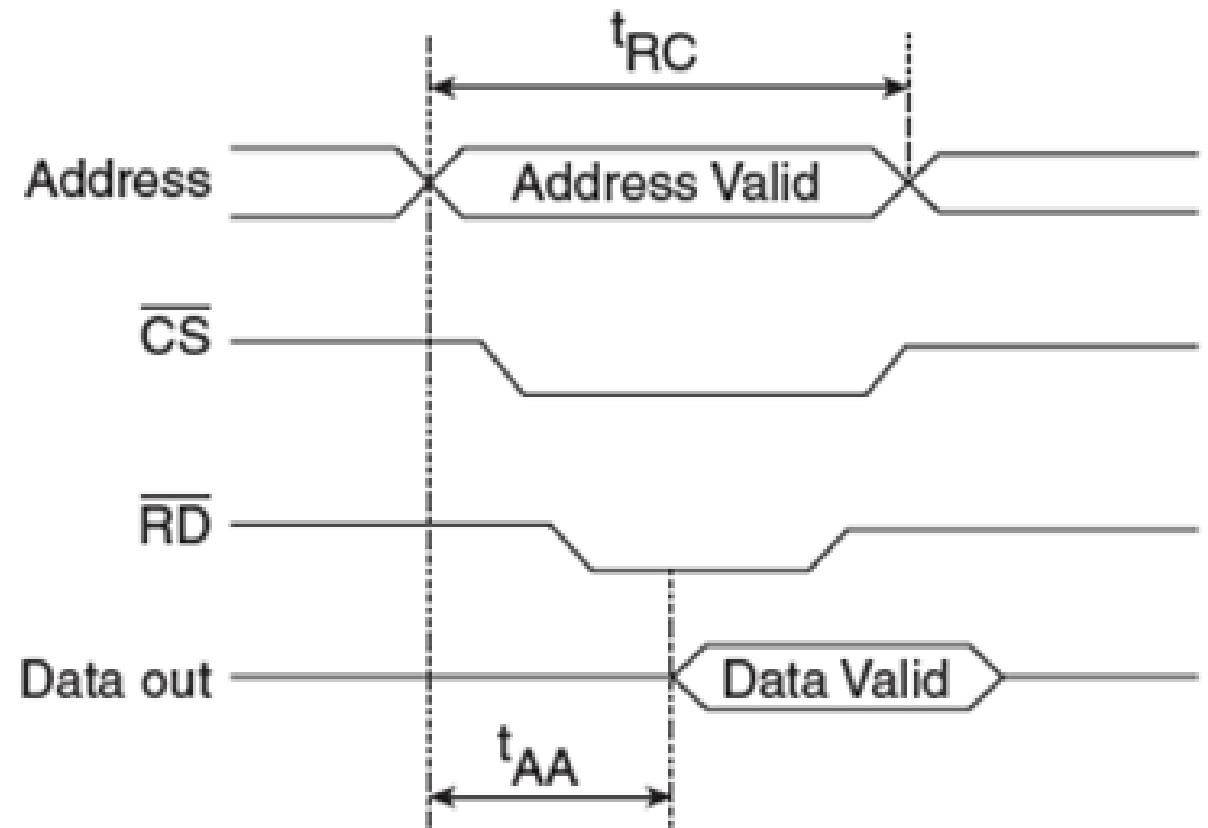


Memory read cycle

- The reading cycle of one byte of data from this SRAM chip has the following steps:
- The address of the location whose content is to be read, is placed on the address bus.
- The chip is selected by making **CS** low.
- The **RD** signal is activated.
- The data appears on the data bus after a minimum time.

Asynchronous read

- t_{AA} , the **read access time**. This is defined as the time taken for the data to appear on the data bus, after the address is placed on the address bus.
- t_{RC} the **read cycle** is the minimum time that must elapse before the next read operation can be started.



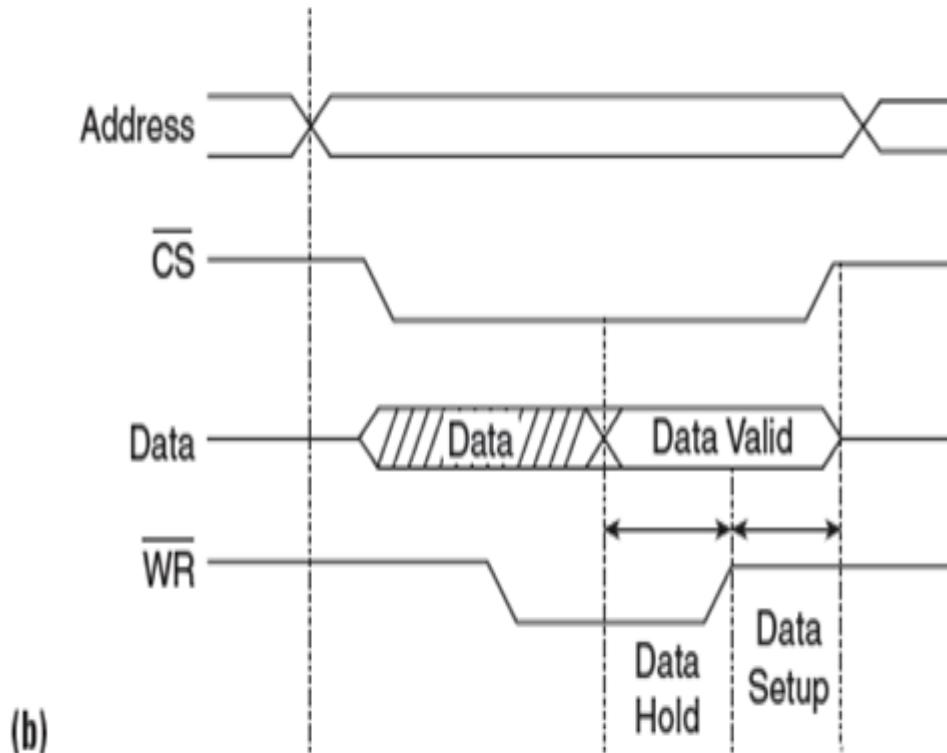
Memory write cycle

Writing has similar steps as reading:

1. On the address bus, the relevant address is placed.
2. The *CS* signal is activated.
3. The data for writing is placed on the data bus.
4. The *WR* signal is activated.
5. The data is considered valid and is written in the addressed location.

Asynchronous Write

- In these diagrams, we find that that the operations of reading and writing *are not* timed by any clock, and hence are 'asynchronous'.
- This is a bygone concept.
- Today, SRAMs are timed by the clock signals of the system they are part of, and are actually 'Synchronous SRAMs'.



2.12.2 Dynamic RAM (DRAM)

- Next we take a look at DRAM which is used as primary (main) memory PCs and also as RAM in many embedded boards. It is named 'dynamic Ram'
- Because data is stored as voltage in a capacitor which leaks away with time thus is dynamic or changing with time.
- It consists of a single field effect transistor (FET) and a capacitor, which is the storage element.
- The amount of charge in the capacitor decides the cell logic of either a '1' or '0'.
- A capacitor, as we know, does not retain the charge it contains unless it is replenished continuously.
- This action of recharging the capacitor is referred to as 'refreshing' and is done at regular intervals.

Read cycle of DRAM

The steps in the Read cycle are summarized below. Refer to the timing diagram

1. The row address is placed on the rows
2. The row address strobe RAS signal is then activated.
3. The address latch inside the chip saves the row address.
4. Next, the column address is placed on the same address.
5. The column address strobe CAS signal is then activated.
6. The address latch inside the chip saves the column address also.
7. The CAS pin also serves as the output enable.
8. With this, the data in the selected address is available at the output buffers of the chip, and it is transferred to the data bus of the processor.
9. CAS and RAS return to their previous state to complete the read cycle.

SDRAM

- This is 'Synchronous' DRAM —
- DRAM whose operations are synchronized by a clock.
- This is the DRAM which is actually used now Technologically it is the same as DRAM but because of being synchronous with the system clock and controlled by a 'finite state machine',
- many attributes pertaining to memory operations can be finely tuned.

DRAM

- This is an SDRAM which has 'Double Data Rate'.
- The 'double data rate occurs because it transfers data at both the rising and falling edges of the clock.
- DDR-2 and DDR-3 are just faster versions of DDR SDRAMs and use special techniques for speed.

2.12.3 Read only memory (ROM)

- This is 'Read Only Memory' which means that once written to, the contents remain unchanged until we deliberately change it by special means of erasing and re-writing into it.
- ROM is used for keeping information 'firmly'.
- The only operation we can do on ROM is reading its contents.
- We use the word 'firmware' when we refer to the contents of ROM.

Where does ROM find application in embedded systems?

- In embedded systems, once the application code is tested and found to be working, it is stored in ROM.
- Thus, ROM is the memory area where code is stored.
- In advanced embedded systems, operating systems are also stored in ROM.
- ROM is also used as a storage for data.
- Think of the data we store in our USB sticks, memory cards of mobile phones, cameras etc.
- All these are instances of ROM storage.

2.12.3.1 *EPROM*

- This stands for Electrically Programmable ROM (E-PROM)
- Which data can be burned or erased, by using a device called PROM programmer.
- To erase the contents, it is exposed to ultraviolet light.
- All this needs the chip to be removed from its circuit.

2.12.3.2 *OTP ROM*

If the same information is to be burned into many ROM chips and these chips are part of a consumer product, the need for 're-programming' is absent.

In such a case, a set of ROMs are taken to a factory and programmed *en masse*.

No facility is given to re-program them and such ROMs are called OTP (One Time Programmable) ROMs.

2.12.3.3 EEPROM

This is 'Electrically Erasable programmable ROM'.

The difference in this is that erasing and writing data can be done while the chip is in a circuit, but 'higher than normal voltages' are needed for erasing and re-writing.

EEPROMs are constructed like EPROMS, but allow the erasing of individual bytes or the entire memory without UV light.

The problem with EEPROM is that only one byte at a time can be erased and re-written, so the whole process is too slow for large amounts of data.

So, EEPROMs are used in systems which need to store small amounts of information which may be required to be rarely changed, like calibration tables, configuration information etc.

Many MCUs (PIC, for example) have a small amount of EEPROM inside them.

2.12.3.4 *Flash ROM*

This is the type of ROM which is erasable and re-writable with normal circuit voltages and also at high speeds.

Dr Fujio Masuko of Toshiba is credited to be the inventor of Flash memory.

We all are used to Flash devices.

SD card

SD stands for 'Secure Digital'.

These are flash memory cards with security features added to it.

The security features include cryptographic protection for copyrighted data.

SD cards are widely used for storage in many of Our hand-held devices, and are available in various storage and physical sizes.

2.12.4 Cache

This is a memory concept meant for speeding up memory accesses. Caches are always used in general purpose computing systems, but in the embedded domain, only high-end MCUs use it.

The cache is a type of memory which is placed close to the processing engine, and is fast in response and low in volume.

It is used to store a copy important data and code which needs to be accessed at a rapid rate.

2.13 Designing Low Power Systems

- In our current world, everyone harps (Talks) on the need for 'low power design'.
- The requirement of high performance (which means computation at high speed) has been modified to 'performance per watt'
- which means that only if the power dissipation is low, does high performance become acceptable.

How do we quantify the power requirements of any system or component?

- The term 'Thermal Design Power' (TDP) is used to compare different systems in terms of power.
- TDP is defined as the maximum amount of power (in watts), the cooling system must be able to dissipate.
- It does not mean the maximum wattage, but is meant to give an idea of how much power would be necessary to run applications for which a system is designed for.
- There is a TDP value for a processor, as well as TDP values for systems.

What is the TDP of systems we see around us?

- A smart phone has a typical value of 4-5 W
- A tablet may have up to around 12 W
- Low power embedded systems are expected to maintain a value below 4 W.
- TDP rating is very important, obviously.
- While starting a design for which power specifications are stringent, it is important to verify and make sure that the processor has low TDP (with Displays, Sensors etc.. Should belong to 'LPD' category).
- For this to happen, there are 2 stages:

Design & Power management usage

2.13.1 Design stage

1. Processor Design:

- ❑ In the IC design itself, many techniques are being used.
- ❑ Use of extremely low power supply voltages
- ❑ Adopting techniques for low static and leakage currents,
- ❑ insertion of sleep transistors and many other methods to lower the TDP of the processor are adopted.

2.13.1 Design stage

2. System Design:

- ❑ Here the designer needs to choose low power I/O devices, and of low complexity hardware.
- ❑ The clock frequency used is a matter of importance as high clock speeds directly translate to higher power.
- ❑ Unless there is a real requirement for high speed, refraining from using high clock frequencies is a rule to be adhered to.
- ❑ Higher bus widths also contribute to higher power.
- ❑ It is up to the hardware designer to take care of these matters, while the software designer should write optimum code which uses the hardware efficiently.

2.13.2 Power management

- Many embedded systems are in inactive states until they are called upon for some work.
- Think of mobile phones and many other handheld devices.
- All these devices are in either the idle state or deep sleep states.
- These sleep states have been implemented in the processor design itself, but it requires a protocol to ensure that it works as required.
- In this context, there is a power management strategy named ACPI (Advanced Configuration and Power Interface) which has been formalized for computing devices — PCs as well as embedded devices.
- This protocol is embedded into the Operating System such that it becomes operational during the time the device is being used.
- Voltages and frequencies are dynamically controlled and modulated to get performance in tune with the requirements of low power.

What does ACPI do?

It defines a number of states from active to sleep and off states, for various levels of performance and requirements for PCs, phones, tablets etc.

2.14 BUS ARCHITECTURE

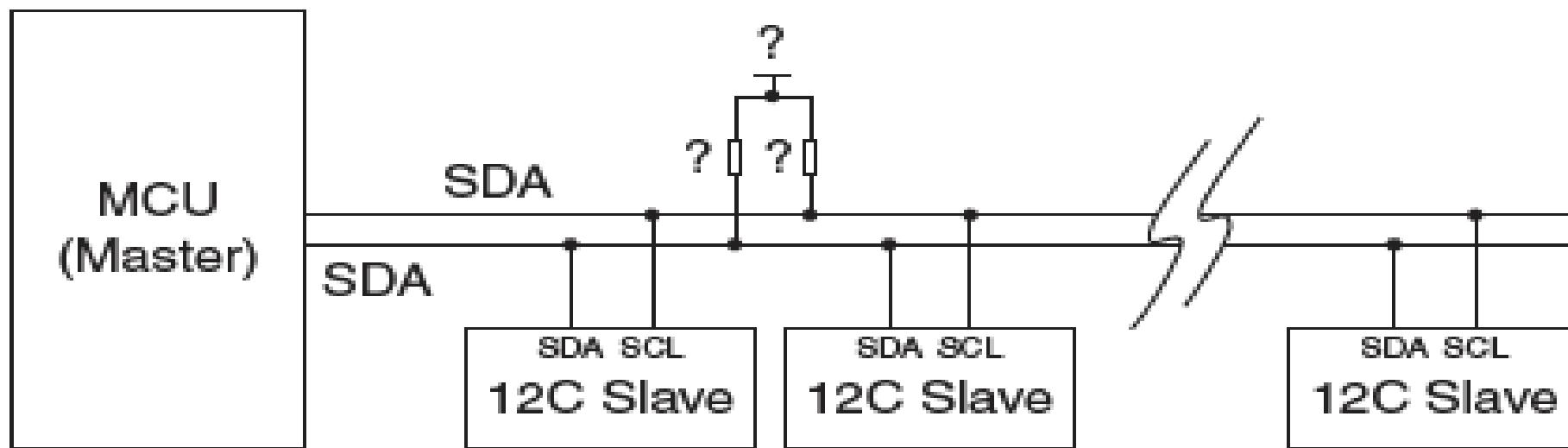
- A bus is a group of wires that transfers information within a computer or between computers.
- The information may be an address, data or some control signals.
- When information travels over a bus, it uses and follows a protocol.
- A protocol is a set of rules associated with the transfer of information over a specific bus.
- A bus may be classified as internal or external.

-
- External buses are those that transfer the information between a system and components outside the system. For example, the USB (Universal Serial Bus) is an external bus.
 - Buses that communicate between components on the same board are referred to as on-board buses.
 - In this section, we will discuss two popular on-board buses — the controllers for such buses are present in all MCUs, and that makes it necessary for a brief discussion on two of them.

2.14.1 Inter integrated circuit bus (I2C)

- ❑ This is a two-wire bus, developed by Philips and is generally used for communication between ICs on the board.
- ❑ For example, an MCU may communicate with other ICs like a temperature monitor, EEPROM, ADC etc, if they also have I2C controllers inside.
- ❑ Figure shows the conceptual view of an I2C bus.
- ❑ It is a serial, synchronous, byte-oriented bus, which uses only two wires (hence, the name 'Two wire Protocol').
- ❑ The operation is timed by a clock, data is sent/received serially and after each byte is received, there is an indication to that effect.

The I2C Bus with One Master and Many Slaves



Let us examine the finer details of the bus. Figure 2.26 shows that there are only two wires for the bus, namely the SCL (serial clock) and serial data (SDA). Each wire has a resistor connected to the supply voltage. This is a 'wired AND connection' for the bus that indicates an open collector or open drain connection depending on whether a bipolar transistor or FET is used.

2.14.1.1 Open drain/collector connection

- ❑ Open drain/Open collector refers to a type of output which can either pull the bus down to (ground, in most cases), or "release" the bus and let it be pulled up by a pull-up resistor.
- ❑ When the bus is released by the master or a slave, the pull-up resistor (R) on the line is responsible for pulling the bus voltage up to the power supply voltage.
- ❑ The necessity of such a "wired AND" connection is that if any device on the line wants to communicate
- ❑ It will pull the line down and no other device can use it, until this device releases the bus. This is an important concept to realize when dealing with I2C devices

2.14.1.2 Steps in I2C protocol

Let us list out the steps in the I2C protocol

1. START and STOP signals: The master sends the START signal by pulling the SDA line low, while the SCL line is high. At the end of data transfer the master sends the STOP signal, which is a HIGH state of the SDA signal, while the SCL line is high.
2. Data Transfer: This starts after the START signal. The master sends an address (MSB first). This is the address (7 or 10 bit) of the device to which it intends to communicate.
3. The next is R/W signal. If the master wants to get a data from a slave, it is READ operation, otherwise it is WRITE.

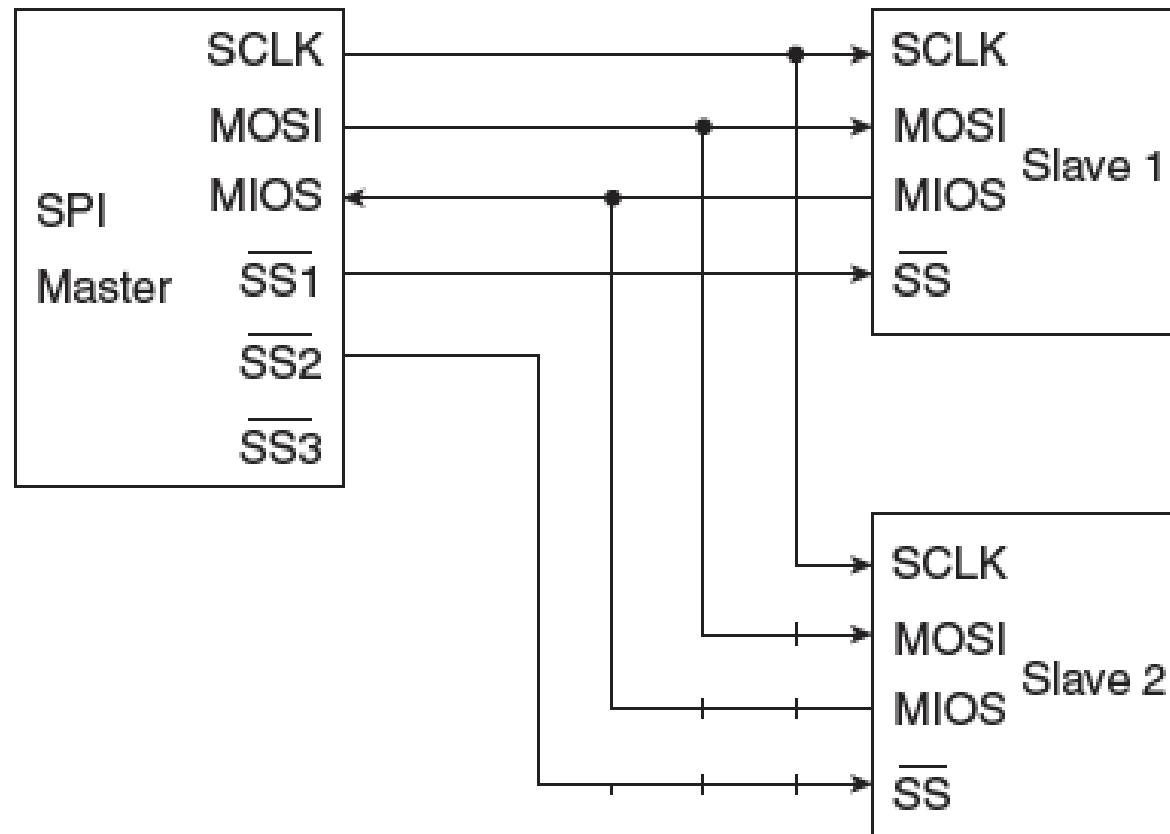
-
4. The slave whose address has been received will respond with an ACK (Acknowledge)) signal.
 5. After this, the actual data transfer occurs. Data is sent one byte (MSB first) at a time. After each byte is sent, the receiver returns an acknowledge signal ACK.
 6. Each byte of data (including the address byte) is followed by one ACK bit from the receiver. The ACK bit allows the receiver to communicate to the transmitter that the byte was successfully received and another byte may be sent. After the data transfer is fully over, and the final acknowledge signal is received, the master sends the STOP signal. Before the receiver can send an ACK, the transmitter must release the SDA line. To send an ACK bit, the receiver shall pull down the SDA line.

2.14.2 Serial Peripheral Interface

- ❑ The Serial Peripheral Interface (SPI) is another on-board protocol. It has three signals and is also a master slave protocol. An SPI bus has only one master.
- ❑ It can have more than one slave.
- ❑ There is a slave Select signal in the master using which a slave can be selected.
- ❑ The data transmission between the master and one slave occurs through two signals:

MOSI — Master Out Slave In

MISO — Master In Slave Out



Chapter-3

Embedded systems-

The software

PRASHANTH KAMBLI, Dept. of ISE, RIT

The concepts covered in this chapter are:

- The difference between big endian and little endian data formats
- Why data alignment is needed?
- The difference between memory mapped I/O and peripheral I/O
- RISC processors and the Load—Store architecture
- How stacks are realized ?
- The different flags available in processors
- The meaning of the term 'Instruction Set Architecture'
- What constitutes an IDE ?
- The components of software debuggers

INTRODUCTION

- We have discussed the hardware aspects of general embedded systems.
- Here, we will discuss additional aspects of processors and embedded systems, which are more related more to software.

3.1 ENDIAN-NESS

- ❑ It is to be understood that when data is stored in memory, each address corresponds to a storage of only one byte
- ❑ we say that memory is 'byte oriented'.
- ❑ When a word of 16 bits is stored, it occupies two byte spaces in memory, and it is stored in ADDRESS, and ADDRESS+1.
- ❑ Now it is regarding the order in which the data bytes are to be stored.

-
- Does the lower byte get stored in the lower address or is it the other way round?
 - In fact, there are two ways of addressing the matter, and thus two formats have come up.
 - This applies to 16-bit, 32-bit and 64-bit data.

1. Little Endian

- In this format, the lower byte of the multi-byte word is always stored in the lower address.
- Consider the case of the 32-bit word Ox12345678.
- Let us store it in address 0x40000000.
- How this 32-bit word is stored in memory, in little endian format.
- This format is used in **Intel architecture**.

Address	Byte stored
0x40000000	78
0x40000001	56
0x40000002	34
0x40000003	12

Figure 3.1 The little endian format for a 32-bit number

2. Big Endian

- Here the lowest byte is stored in the highest address.
- Motorola processors follow this scheme.

Address	Byte stored
0x40000000	12
0x40000001	34
0x40000002	56
0x40000003	78

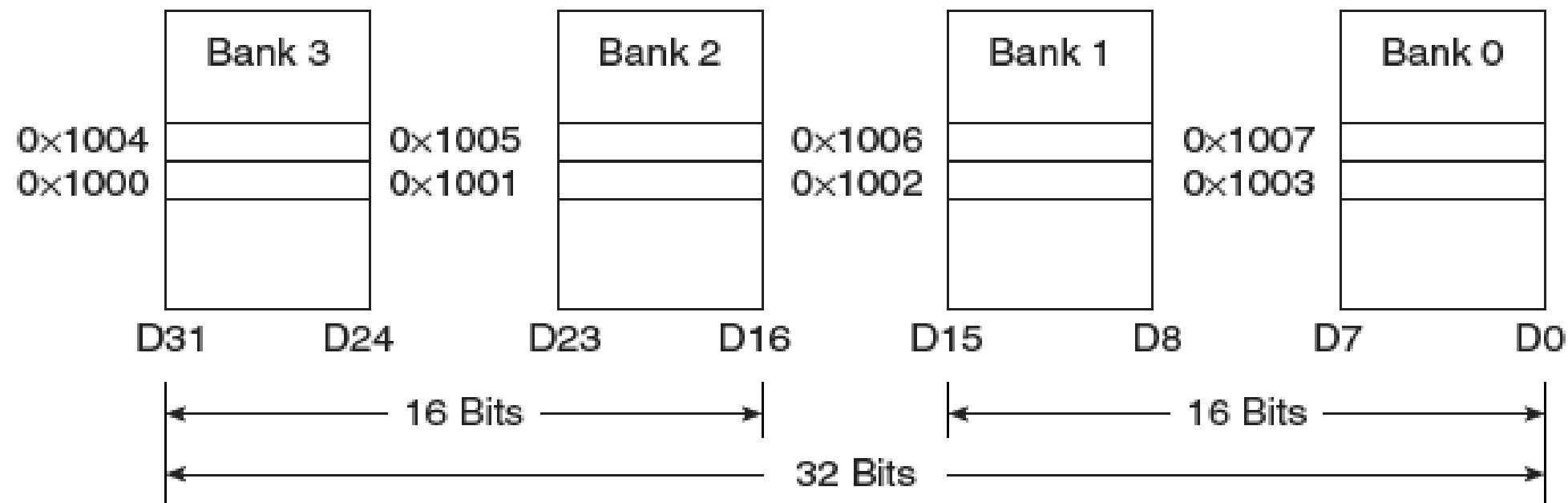
Figure 3.2 The big endian format for a 32-bit number

- ARM processors can use any of the two formats.
- There is a bit in a control register of the processor, using which, one of the formats may be chosen.
- In either case, when a 32-bit number is to be read from memory, only one address *needs* to be specified **in** the program — that is, the starting address.
- In the examples shown, the address 0x40000000 will be specified in the program, but the required four locations will be accessed to get the full 32-bit data.
- This applies for the case of 'writing' also.
- In this book we follow Little Endian

3.2 DATA ALIGNMENT AND MEMORY BANKS

- As ARM processor has 32 as its word length, it means that it can access 32 bits in one cycle, has 32-bit registers and can do 32-bit computations.
- But it still may need to handle data as 8 bit and 16 bits.
- It is understood that data stored in one address of memory is one byte.
- For four bytes, four addresses are needed.
- In memory, these four bytes are taken from four memory banks to get a 32-bit word.

Memory Banks for a 32-Bit Processor



- Each bank contributes one byte to a 32-bit word.
- For getting a 32-bit data, four banks have to be accessed simultaneously.
- For getting a 16-bit data, two banks have to be read together.
- Storing or Loading of 4 bytes in memory can be done in one cycle
- Because the processor has a 32-bit data bus.
- When 32-bit data is stored in memory, four addresses are needed.
- But we need to specify only one address in our instruction.
- For 32-bit data if we specify an address Ox 1000, this address and the next three addresses are accessed automatically with complete 32-bit data is obtained in one cycle.
- However, if the address specified is Ox 1001, two cycles of data transfer are needed.
- In one cycle, the addresses Ox1001, Ox1002 and Ox1003 are accessed. Only 3 bytes of the data are transferred. In the next cycle, the address Ox 1004 is accessed.

- This discrepancy is because of 'misalignment'.
- It is obvious that in the first case, the data bytes stored in corresponding positions of each memory banks are accessed.
- We say that the address Ox 1000 is an aligned address for 32-bit data.
- For 16-bit data to have aligned access, either Ox 1000 or Ox 1002 must be specified. Using Ox1001 causes misalignment and an extra cycle will be needed for access.
- In short, the condition for aligned data access is for 32-bit data, the address should be divisible by 4 (the lowest two bits of the address must be 0).
- For 16-bit data, the address should be divisible by 2 (the LSB must be zero).
- The compilers for ARM make sure that data alignment conditions are followed.

3.3 PERIPHERAL I/O AND MEMORY MAPPED I/O

- For any processor, there is an associated address space.
- If the address bus is of 20 bits, the address space is 2^{20} bytes, which is 1 MB.
- If the address bus is 32 bits, the address space is 2^{32} bytes, which is 4 GB.
- This address space may be mapped to memory.
- But the processor has I/O devices also and they need addresses.
- There are two ways in which the I/O address space is mapped.

3.3.1 Peripheral or I/O mapped I/O

- In this, the address spaces of memory and I/O are disjoint.
- For example, for 8086, the complete 1 MB of address is mapped to memory alone.
- For I/O, there is another address space.
- In this type of mapping, I/O access and memory access have separate instructions.
- I/O is accessed using special IN and OUT instructions. Memory is accessed by MOV instructions.
- This method is also called 'Isolated I/O'.

3.3.2 Memory mapped I/O

- Take the case of ARM7. It has 4 GB of address space.
- This address space is shared by memory and I/O.
- The instructions to access memory and I/O are the same.
- This is the method currently used by most advanced MCUs.

3.4 LOAD STORE ARCHITECTURE

- There are two faculties of thought in processor design.
- They are the RISC and CISC philosophies, which means 'Reduced Instruction Set Computer' and 'Complex Instruction Set Computer', respectively.
- The hardware used for realizing instructions is simple and so only simple and basic operations are available as instructions in RISC.
- The complex operations are expected to be realized by writing code using the simple instructions available.

-
- In CISC, there are complex instructions which have dedicated hardware for each of them.
 - So RISC may be slower and simpler
 - While CISC will be faster, but CISC will dissipate more power because of its relatively complex hardware.
 - The term 'Load Store Architecture' is commonly used in the case of RISC processors.

RISC processors have the following characteristics

1. All instructions are executed in one cycle.
2. All instructions have the same length.
3. Data processing instructions use only registers.
4. The only instructions that access memory are the Load and Store instructions.

Comparison with CISC processor

- In CISC Processor (8086)
- Instruction ADD MEM, AX
- AX means that data from an address names MEM is to be added to the register AX.
- The result is to be placed in MEM
- This instruction is a data processing instructions, but within it, it access memory for getting a source operand.

- In RISC Processor, no case of data processing to access memory.
- To do an ADD operation, the steps are:
 1. Get first operand from memory to a register. This is done through Load Instructions.
 2. Get second operand from memory to another register. This is done through another Load Instructions.
 3. Add the content of the two registers.
 4. The result is saved in memory. This is done through Store operation.
- In case of RISC processors, only the Load and Store instructions access memory.
- Thus, RISC is called a 'Load Store Architecture'.

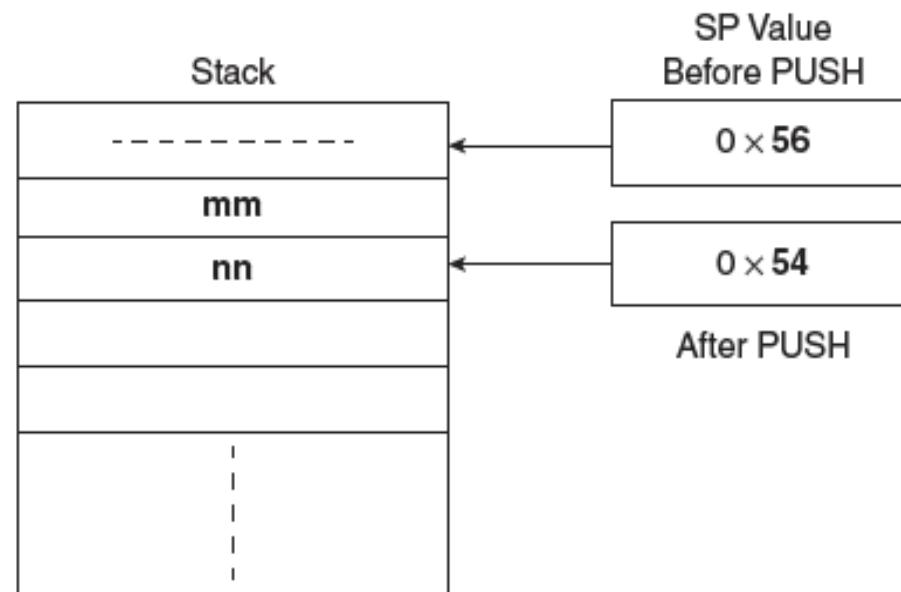
3.5 STACK

- All processors have a stack associated with it.
- A stack is not a hardware element.
- It is a data structure created in memory (RAM).
- It is some space defined in memory to store data temporarily.
- The special feature of the stack structure is its LIFO (Last In First Out) characteristic.
- There are only two operations for a stack data structure — PUSH and POP.

-
- As the stack is a data structure, it may be defined in different ways.
 - Thus, we may have a descending stack or an ascending stack.
 - Defining a stack amounts to just defining a stack pointer for it.
 - If it is an ascending stack, it 'grows upward' when data is 'pushed' in.
 - For a descending stack, the stack grows downwards, and goes to decreasing addresses as data is pushed on it.

3.5.1 Descending stack

- Let us have a look at the PUSH operation of a descending stack

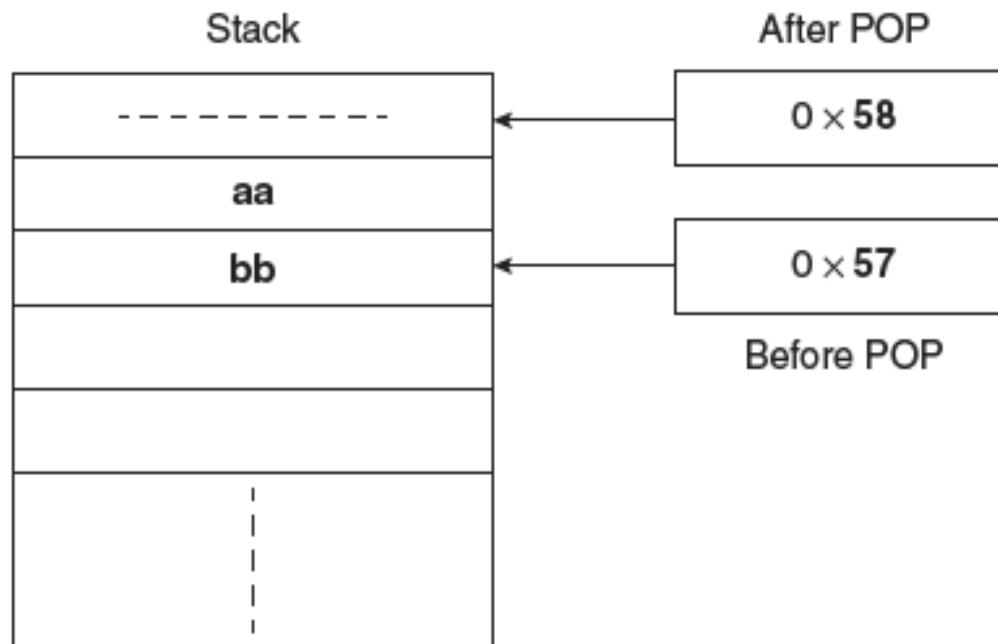


- To define this stack, we first load the number 0x58 in the stack pointer register SP.
- Now let us transfer two numbers mm and nn from two registers into the stack using the push operation.

The steps are as follows:

1. The SP value is 0x56. It is decremented and becomes 0x55. The number mm is stored in that address.
2. The SP value is decremented again to become 0x54. The number nn is saved in this address. This is the second PUSH operation.
3. The SP value is now 0x54.

The **POP** operation for a descending stack has the following steps and is the reverse of the **PUSH** operation



-
1. Before the POP instruction, the SP has a value 0X57.
 2. The data bb is copied to a register. SP is incremented.
 3. The SP value is now 0x58.

For an ascending stack, the operations of PUSH and POP are just the reverse.

3.6 FLAGS

- All processors have status flags to indicate something about the result of an operation.
- Each flag is a flip flop which is either set or reset.
- The one or zero state of flags may be needed before a conditional execution.
- The flag bits are generally seen in some status register of the processor.
- In 8051, flags bits are available in a register called the Processor Status Word.
- In ARM, there is a register called CPSR (Current Program Status Register), which holds the flag bits.

3.6.1 Carry Flag (C)

- Carry Flag (C) gets set if there is a carry out from the most significant bit during calculations.
- When an 8-bit addition causes the result to be greater than 8 bits, there is a carry out from the MSB (D7) that causes the flag to be set.
- For 16-bit operations, the carry will be from D15, and CF will be set.
- For 32-bit processors, it is set when the result is greater than 32 bits.
- The C flag will also be set in the case of a 'Borrow' during subtraction.

3.6.2 Zero Flag (Z)

- When the results of an arithmetic or logic operation is zero, the zero flag gets set
- if we keep on decrementing the contents of a register, it will finally become zero.
- At this instant, the zero flag gets set, that is, $Z=1$.
- Also when two numbers are compared.
- Comparison is achieved by subtraction.
- If the numbers compared are equal, the zero flag is set ($Z=1$) including equality of the operands.

3.6.3 Negative Flag (N)

- After an arithmetic or logic operations, if the result contains a negative number.
- The N flag is set.
- It contains the MSB of the result,
- Which can be interpreted as the sign bit in signed arithmetic operations.

3.6.4 Overflow Flag (V)

This flag is set under one of the following conditions

1. There is an overflow into the MSB from the bit of lower significance, but no carry out from the MSB.
2. There is a carry out from the MSB, but no carry into the MSB.

-
- To understand the overflow flag, let use an example. We declare that the numbers we use are 'signed'. We add +5 and +4 using four-bit word length.

The addition is 0101

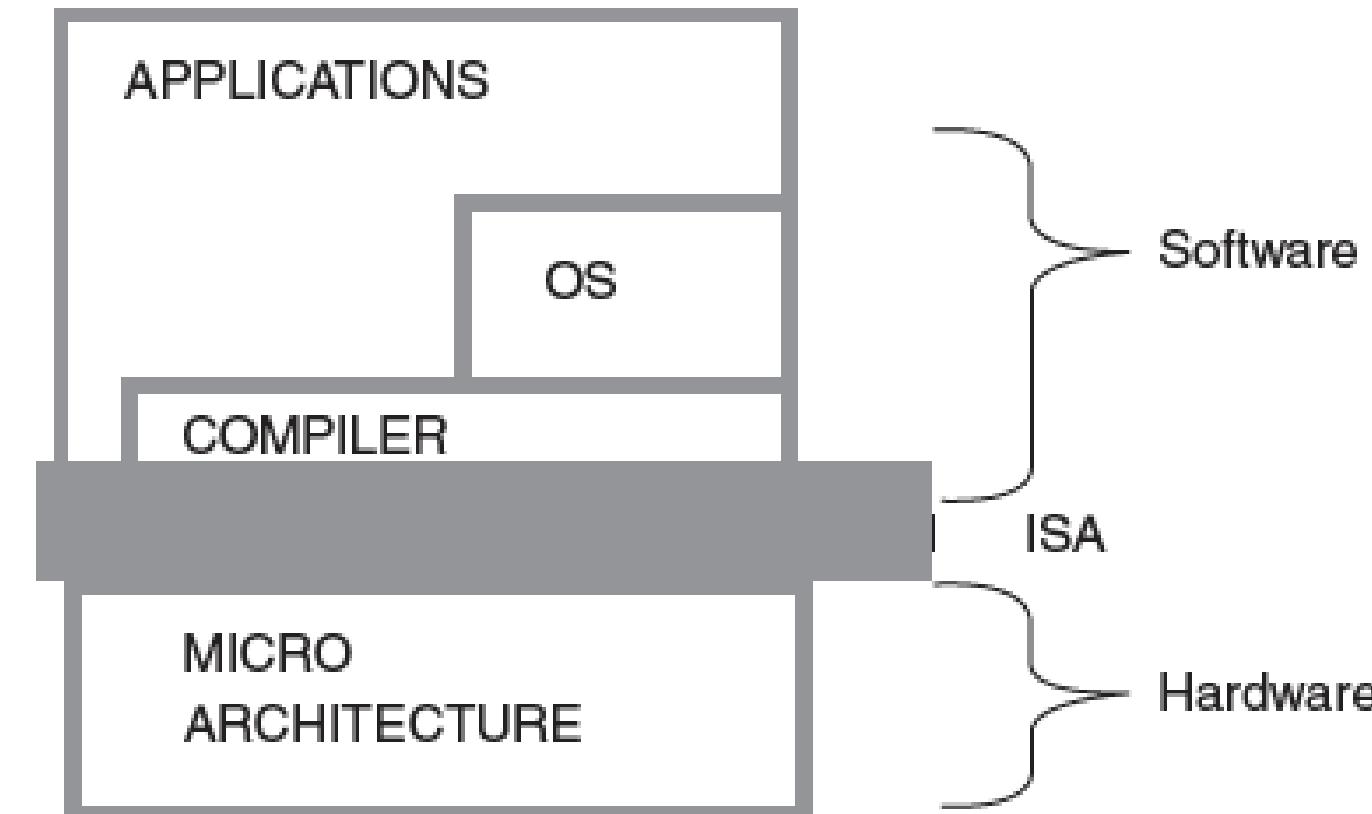
$$\begin{array}{r} + 0100 \\ \hline 1001 \end{array}$$

The sum shows the MSB to be '1'. There has been an overflow into the MSB.

The V flag will be set.

3.7 INSTRUCTION SET ARCHITECTURE

- Instruction Set Architecture (ISA) is a word that is frequently encountered in Computer architecture.
- The ISA is the processor, viewed in terms of its 'instruction set and register set'.
- ISA defines the 'hardware software interface'.
- when a compiler designer or an assembly language programmer looks at the processor, he is concerned only with the instruction set and the registers that he can use.
- Below the ISA, there is the CPU.



- The word 'microarchitecture' is the 'implementation' of hardware to get the required ISA.
- We see that above the ISA, we have the applications and OS implemented using the specific ISA.
- The compiler is the software that understands and uses the ISA for software implementations.
- The x86 ISA is used in the processors of our PCs.
- Most of our mobile phones use the ARM processor.
- Thus, there is the ISA that is different from these the ARM ISA.
- It means that the register set and instruction set of types of computing cores are different.

3.8 INTEGRATED DEVELOPMENT ENVIRONMENT

- An integrated development environment (IDE) is a programming environment
- It has been made into an application program, and has the following components
 - a. Code editor
 - b. Cross compiler
 - c. Debugger
 - d. Graphical user interface (GUI) builder
- An IDE is specific for a particular family of MCUs.

Why do we need an IDE?

- For a specific processor 8051, we need an IDE for developing and testing the programs before we put them in ROM.
- The IDE of 8051, will have in it all components.
- It has ISA, the registers, instructions, the cross assembler/complier and a simulator.
- The user writes his program and it can be tested in the IDE.

Components of IDE

1. Code Editor :

This editor allows code to be written, changed and saved as files in folders called 'Projects'.

During the course of the use of the IDE, the project folder will be found to contain many different types of files associated with that project.

Components of IDE

2. Complier :

A compiler is a program that translates one program Source Code into object code.

Compiler is used for programs that translate source from a high-level language to a lower level language (Assemble language or Machine code).

For a machine to ‘run’ the object code, the final conversion should be into the machine code.

Components of IDE

3. Builder :

Once the code has been written and saved as a file in a project, it needs to ‘Build the project’.

A builder includes the compiler and assembler

It also has the Linker

In an IDE, the processes of ‘Compile, Assemble and Link’ are together called ‘Build’

3.9 DEBUGGING

- During the course of embedded system design and development, it is usually necessary to 'debug' the code.
- This is because many as we visualize it during the time of writing the code.
- For short codes, a logical re-thinking may be sufficient to find the mistake and correct the code.
- But there are codes, for which we need help in locating the logical error and correcting it.
- This is the role of a debugger.

IDE has debugging facilities also

Simulator:

One part of the IDE that is very useful is the simulator.

This is a software based on the architectural model of the MCU.

So it mimics the working of the MCU by having its registers, memory and peripherals

Generally all the functional errors in an application can be detected by running it on the simulator.

Simulators run at much slower rates than the actual processor, and so the timing issues related to programs may not be detected.

A simulator is a great tool for a basic level of code debugging.

1. *Single* Stepping:

This is a very useful activity when using a debugger.

The code can be run, one line at a time and execution stopped after each line.

The results of code execution can be verified in the registers, memory etc.

A simulator is a great tool for a basic level of code debugging.

2. Breakpoint:

When single stepping *seems* too cumbersome, it is useful to set breakpoints.

One can set a breakpoint after a few lines and then do the same activities as single stepping.

A breakpoint is a location, where the *processor* stops its execution and gives program control to the debugger.

At this location, the program counter is equal to the address at which the breakpoint has been set.

A simulator is a great tool for a basic level of code debugging.

3. Watch point:

A watch point is similar to a breakpoint, but it is the address or value of a data access that is monitored.

This means that it is not an instruction being executed from a specific address, that is monitored.

A *register* or a memory address is specified to identify a location that is to have its contents tested.

Watch points are also known as data breakpoints, implying that they are data dependent.

Execution of the application stops when the address being monitored is accessed by the application.

END Of Unit II

Unit -3

Chapter-4

THE ARCHITECTURE OF ARM 7

INTRODUCTION

- In the current world of embedded systems
- ARM processor has made its mark in almost all high-end devices
- It might be mobile phone, automotive hardware, factories or aircrafts
- The first popular one in this series is ARM7 which was used extensively in Apple IPods, PDAs (Personal Digital Assistant) and many other applications.

- Even though the new Cortex series of ARM has superseded ARM7 the architectural difference between them is rather slight.
- It is in this context that we learn the ARM7 architecture first, before moving on to the Cortex series.

HISTORY OF ARM

- It was a company named Acorn RISC machines
- design of the first ARM processor and it was based on a Berkley university design
- ARM1 was launched in 1985, and it was found that its RISC core had capabilities comparable to the CISC processors of that time
- was made with less number of transistors and generated much less power
- ARM2 to ARM6 were designed but never really made a big impact.
- ARM7 was the first commercially successful ARM processor.
- ARM7 was followed by ARM 9 and ARM 11 which were higher versions.

4.1.1 THE BUSINESS MODEL OF ARM

- Right from the beginning, ARM's business model was different from that of Intel, the semiconductor giant.
- Intel is the company that manufactures the processors of personal computers and servers.
- Intel designs and fabricates its chips and sells them as Intel chips.
- ARM, on the other hand only does the design of its computing core.
- It sells these designs, designated as IP (Intellectual Property) to licensees who can do adequate modifications or additions to the design, and then get it fabricated in global foundries.

- The designs sold by ARM may be in various forms — it can be a hardware description code or a transistor layout.
- Thus, ARM designs the 'processor' part or the computing engine, which is also called a 'core'.
- Licensees may add peripherals to it and make it into a microcontroller or 'SoC'.

4.1.2 ARM AS A RISC PROCESSOR

- This powerful processor is designated as a RISC processor.
- RISC stands for Reduced Instruction Set Computer.
- A RISC architecture has only simple instructions realized in hardware.
- Complex operations are made possible by writing programs using these simple instructions.
-

- To elaborate, think of the 'divide' operation.
- RISC processors do not have an instruction for division
- A program that does repeated subtraction is a method of getting division done.

- The advantage is that the absence of a dedicated division unit reduces the complexity of the processor, and also leads to less power dissipation.
- The disadvantage is that division in RISC is not as fast as in CISC (Complex Instruction Set Computer).
- ARM design started as a pure RISC architecture.
- But as its importance and market intrusion improved, it was realized that some complex instructions are also needed.
-

- Digital Signal Processing is one area in which complex computations are unavoidable, for which dedicated hardware is an absolute must.
- As a result of this necessity, ARM processors had to add a few complex instructions.
- So ARM is now a RISC processor with CISC instructions also added to it.

SOME OF THE FEATURES ADDED TO LATER GENERATIONS, FOR MAKING IT A TRULY COMPUTATIONALLY INTENSIVE PROCESSOR ARE

1. SIMD (Single Instruction Multiple Data) instructions
2. Floating Point Support
3. Specialized DSP (Digital Signal Processing) instructions

THE CHIEF FEATURES OF RISC ARE LISTED BELOW. THIS HOLDS FOR MOST OF THE INSTRUCTIONS

1. Instructions have the same size of 32 bits.
2. Each instruction takes one cycle to complete.
3. It is a 'load store' architecture.

The end result of the RISC approach made the processor to be a low power dissipating one and that became the reason why it easily conquered the growing embedded market, where hand held and battery operated devices are the major constituents.

4.1.3 ADDITIONAL FEATURES

- As ARM was accepted in the embedded market, more features were added to it.
- ARM design was modular in structure so it was easy to add new features.
- The flexibility on account of the business model of ARM, made the addition of new and special features optional.
- Knowing these features will let us understand the early naming conventions of the different ARM chips.

4.1.3.1 THUMB SET

- The ARM ISA has only 32-bit instructions
- This means that each and every instruction has to be of 32 bits.
- For simple applications, such a powerful set of instructions are not needed.
- Thus, a new set of instructions, named THUMB, which are of 16 bits length was added.
- This is helpful in reducing the amount of code memory needed.

- The amount of the code in unit area of memory, that is, the code density, is more, when THUMB is used.
- Thumb code takes 40% less space in comparison to regular 32-bit ARM code but is slightly less efficient.
- There is also the facility to mix ARM and THUMB code and this is called **ARM-THUMB interworking**.

4.1.3.2 CACHE

- All modern processors have caches, though some have only TCMs (Tightly Coupled Memories)
- ARM7 has 8 KB cache in which both instructions and data are allowed.

4.1.3.3 LONG MULTIPLIER

- Even though fast multiplication is a complex instruction, many ARM variants have such multipliers.

4.1.3.4 DEBUG UNIT

- Inside the chip, there is a dedicated unit which provides the necessary support for testing and debugging
- This unit is designed based on the specifications of JTAG (Joint Test Action Group).
- A 'boundary scan architecture' is defined which makes it easy to test the chip.

4.1.3.5 EMBEDDED ICE MACRO CELL

- All ARM processors need not have this feature.
- This is also a separate 'cell' to facilitate advanced debugging.

4.1.3.6 SYNTHESIZABLE

- When the ARM IP is sold to the licensee in the form of an HDL (Hardware Definition Language) code, the buyer can flash this code to an FPGA (Field Programmable Gate Array), add peripherals, make changes etc.
- Such ARM cores are said to be 'synthesizable'. Some FPGAs can also have this ARM design as a 'soft core' in it.

4.1.3.7 JAZELLE

- In the early days of ARM, it was common for some ARM processors to execute Java bytecode in hardware as a third execution state along with the existing ARM and Thumb mode.
- This is useful to increase the execution speed of Java ME games and applications.
- Such an extension is now obsolete.

4.1.3.8 ENHANCED DSP INSTRUCTIONS

- ARM to make useful in advanced signal processing applications
- It was necessary to add advanced signal processing instructions in the processor.
- Such processors would have DSP instructions
- which meant that specialized DSP hardware is available in them.

4.1.3.9 VECTOR FLOATING POINT UNIT

- Such a specialized hardware implies support for single and double precision arithmetic which enables floating point computations.

4.1.4 NAMING CONVENTIONS

- Naming convention is associated with the processor.
- The most popular ARM processor is ARM7 TDMI (Thumb Debug Multiplier In circuit Emulator)
- which means an ARM7 processor with Thumb extension (T), Debug interface (D), long multiplier (M) and an ICE macro cell.(I). ARM7TDMI-E means that DSP extensions are available, and ARM7TDMI-S means that it is in synthesizable form.
- These naming conventions are not used for the new ARM processors, but many ARM7 processor capabilities can be gauged by this convention.

Table 4.1 Early naming conventions for ARM

ARM {x}{y}{z}{T}{D}{M}{I}{E}{J}{F}{H}{S}	
X	Family (7, 8, 9, 10, 11, ...)
Y	Memory management/protection unit
Z	Cache
T	Thumb 16-bit decoder
D	JTAG debug
M	Fast Multiplier
I	Embedded ICE macrocell
E	DSP Enhanced instructions (assumes TDMI)
J	Jazelle
F	Vector Floating-point Unit
S	Svnthesizable Version

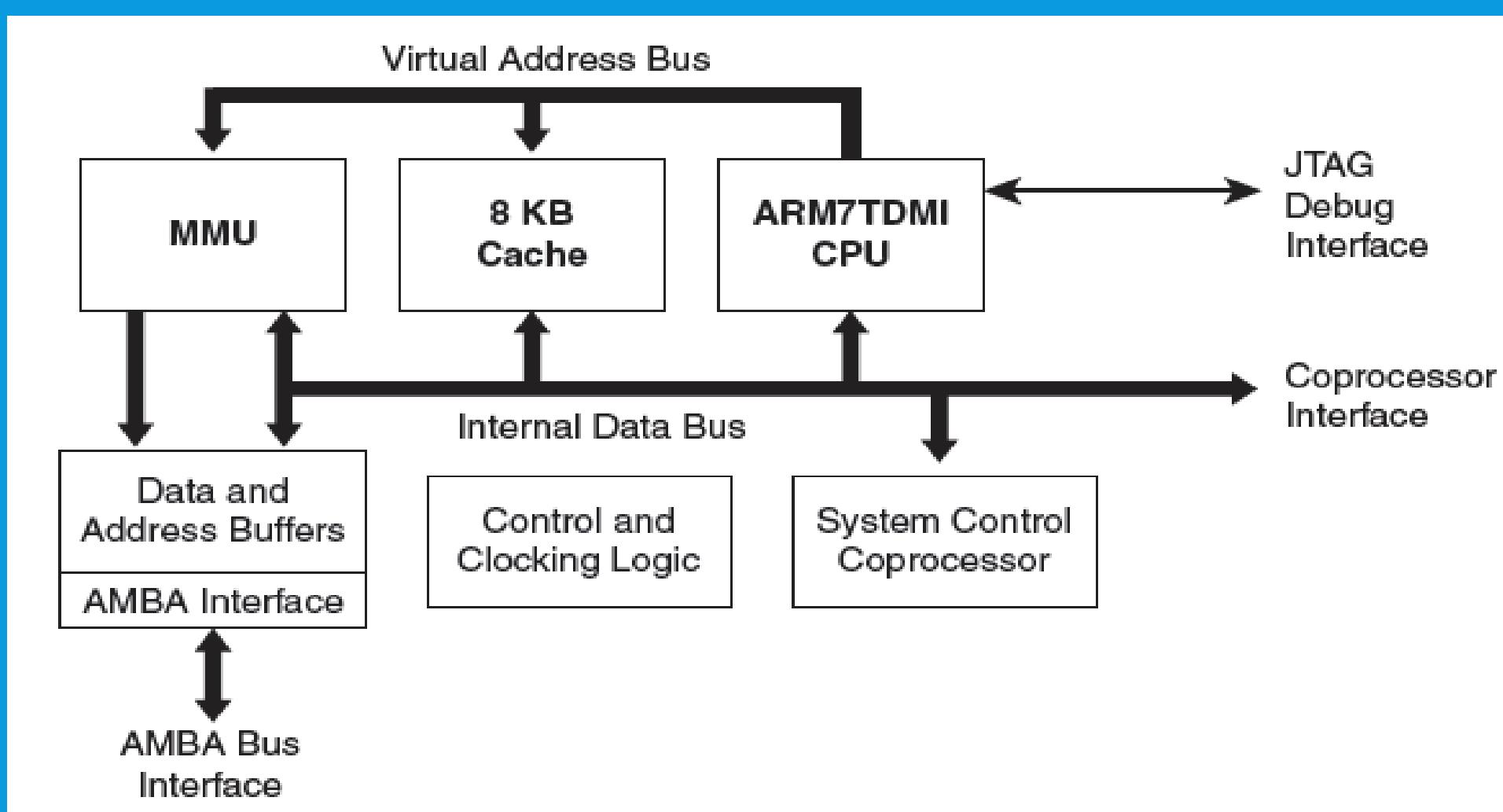
4.1.5 ARM ISA VARIANTS

- The first Instruction Set Architecture of ARM7 was called ARMv4.
- With a Thumb extension, it came to be called ARMv4T.
- The next ISA was ARMv5 and used for ARMS and it implied the Thumb set inherent in it.
- With DSP instructions added to it, the ISA became ARMv5E.
- The next ISA series was ARMv6 and ARMv7 and used for ARM 11 as well as for the Cortex series.
- The latest ISA for 64 bit ARM is ARMv8.

4.2 ARM7 ARCHITECTURE

- We have seen the basic aspects of the ARM architecture
- let us get into the details
- Typical ARM7 processor as shown in the block diagram
- Note that it is an ARM7TDMI CPU.
- All the functional blocks in this diagram will be explained.
- We start with the programmers model.

BLOCK DIAGRAM OF A TYPICAL ARM7 PROCESSOR



- MMU- Memory Management Unit
- 8KB Cache - TCMs (Tightly Coupled Memories)
- ARM₇TDMI CPU- The most popular ARM processor is ARM₇ TDMI (Thumb Debug Multiplier In circuit Emulator)
- Data and Address Buffers
- AMBA Interface- Advanced Microcontroller Bus Architecture
- Control and Clocking Logic
- System Control Co-Processor

4.2.1 PROGRAMMERS MODEL

- A programmers model is the view of the processor, in terms of its registers and its instruction set.
- Since the CPU is ARM7 TDMI, we know that it can operate in the ARM and THUMB states.
- The ARM state is one in which it executes 32-bit, word-aligned ARM instructions.
- The THUMB state is one in which it operates with 16-bit, half word-aligned THUMB Instructions.
- The data format in memory can be **big endian** or **little endian**
- This is selected by a bit named '**bigend**' bit in the Control Register.
- It supports byte (8-bit), halfword (16-bit) and word (32-bit) data types.
- Words must be aligned to 4-byte boundaries and half words to 2-byte boundaries.

4.2.2 MODES OF OPERATIONS

- There are seven modes of operation as follows:
 1. **User:** Unprivileged mode under which most tasks run.
 2. **FIQ (Fast Interrupt mode):** entered when a high priority (fast) interrupt is raised.
 3. **IRQ (Interrupt Request):** entered when a low priority (normal) interrupt is raised.
 4. **Supervisor:** entered on reset and when a Software Interrupt instruction is executed.
 5. **Abort:** entered when memory access violations occur.
 6. **Undef:** entered when an undefined instruction occurs.
 7. **System:** privileged mode of operation.

WHAT IS THE IMPORTANCE OF THESE MODES?

- All application programs run in the user mode.
- The other modes, which are called 'exception modes' are entered for processing interrupts or for accessing protected resources.
- Mode changing may be done under 'software control' (by bit settings in registers) or by external interrupts or exceptions due to specific conditions or errors.
- Following Table indicates the specific exception through which each of the exception modes are entered.

Table 4.2 Exception modes

Mode on entry	Exception
Supervisor	Reset
Undefined	Undefined instruction
Supervisor	Software interrupt
Abort	Abort (prefetch)
Abort	Abort (data)
Reserved	Reserved
IRQ	IRQ
FIQ	FIQ

4.2.3 REGISTER SET

- ARM7 has a total of 37 registers
 - 31 general-purpose 32-bit registers
 - Six status registers
- All these registers cannot be seen at the same time.
- Some are available only in certain modes and states.

ARM REGISTER SET

ARM State General Registers and Program Counter

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

ARM State Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

 = banked register

- The upper part of figure shows the general purpose registers and the Program counter.
- The same set of registers are used in the System and User modes.
- Certain registers are indicated as being 'banked'.
- To understand the idea of banking, let us take a look at the general registers in the FIQ mode. Note that here the registers R8 to R14 are notated as R8_fiq, ..., R14_fiq.
- When a mode switch occurs from the user mode to the FIQ mode, the registers R8 to R14 of the user mode are replaced with a new set notated as R8_fiq, R9_fiq, , R14_fiq.

- FIQ is a mode that is entered for a high priority 'fast interrupt request'. To enable fast response, no time is spent in 'saving the context' of the registers R8 to R14, there is no need to save the content of these registers and their content remains intact. The FIQ mode does not use these registers as it has an entirely new set of registers R8 to R14 to be used in this mode.
- In the other modes, only R13 and R14 are banked. Note that R13 and R14 are the 'Stack pointer register (SP)' and 'Link register (LR)' respectively of each mode.

BUT WHAT IS THE USE OF THE LINK REGISTER?

- The link register is one that is not seen in most other processors.
- Its incorporation into ARM is meant to speed up branching during procedure calls and interrupt processing.
- Generally, in most processors, when a call or interrupt occurs, the 'return address' is saved in the stack which is in memory.
- This involves memory access which causes a certain amount of delay.
- In ARM, this delay is avoided by having this link register to save the return address.
- Since all registers are inside the CPU, no memory access is necessary and thus an additional element of speed is obtained.

THE PROGRAM COUNTER REGISTER (PC)

- There is only one PC in any processor.
- The Program Counter is the register which sequences the instructions as they are being fetched and executed.
- In ARM, the value in the Program counter is the address of the current instruction being 'fetched' (rather than the address of the instruction being executed, as in other processors).

4.2.4 PROGRAM STATUS REGISTERS

- The lower part of figure shows
- One CPSR (Current Program Status Register)
- Banked SPSRs (Saved Program Status Register) for each mode.
- This is the register which has the status bits and control bits for the 'current' time
- Bits 8-27 are 'reserved' which means they are undefined for ARM7.
- The upper five bits store the flag bits and are generally used to understand the result of a data processing instruction.

- The lower eight bits of the CPSR are collectively known as the control bits.
- These change when an exception arises.
- If the processor is operating in a privileged mode, they can also be manipulated by the software.
- The I and F bits, when set, are used for disabling the IRQ and FIQ interrupt modes.
- The T bit indicates the state of operation. If $T = 1$, the processor is in the Thumb state, otherwise, it is in the ARM state.

- The MO—M₄ bits are used to indicate the mode of operation.
- Note that only seven modes are possible here, and any other number in the mode bits will cause an error from which an exit is possible only by a reset.

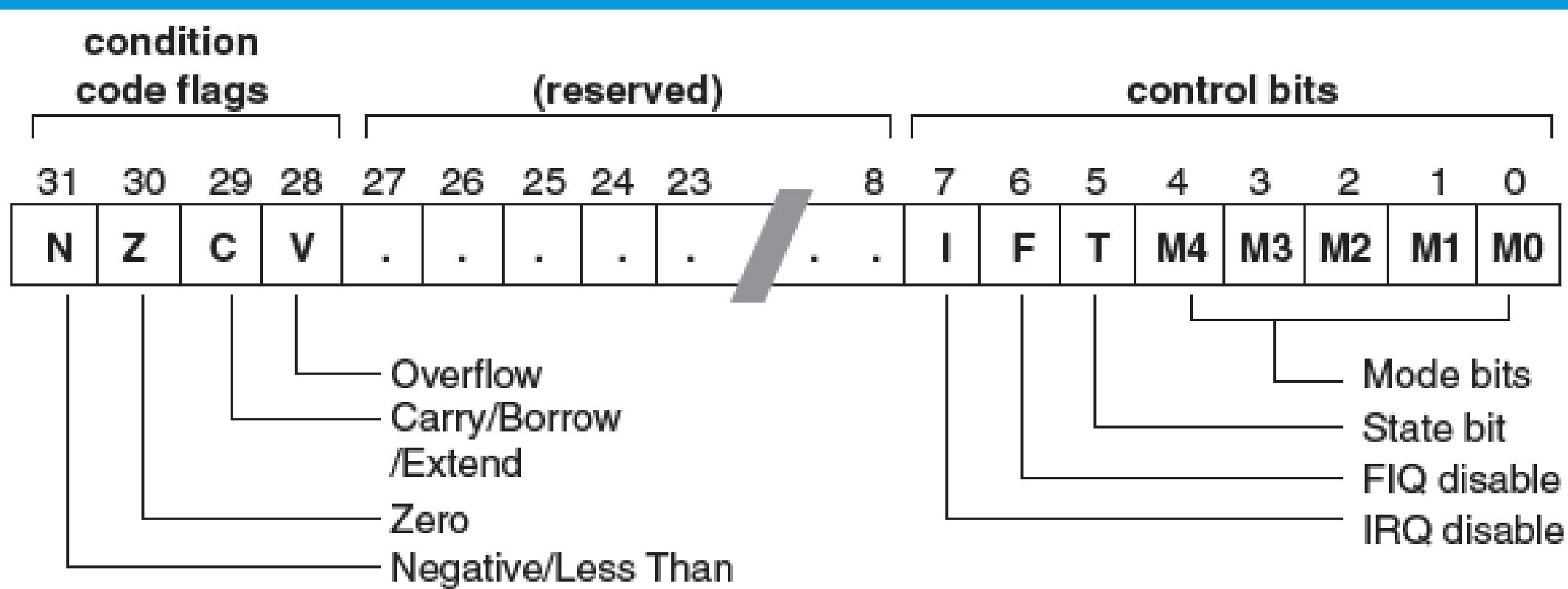


Figure 4.3 Bit configuration of CPSR (Reproduced with permission from ARM Limited. Copyright © ARM Limited)

4.2.5 SAVED PROGRAM STATUS REGISTERS

- There are five 'Saved Program Status Registers (SPSRs)', that is, one for each of the 'exception' modes of operation.
- These SPSRs are shown as banked registers
- When a mode switching occurs, the corresponding SPSR saves the current CPSR value into it.
- The system mode and user modes do not have SPSRs because they are not entered through exception mechanisms.

4.2.6 THE THUMB STATE REGISTERS

- Lets discuss the register set when the processor is in the Thumb state.
- This state involves 16-bit instructions
- It will still handle 32-bit data, but since low complexity applications are envisaged for this state of operation, the number of registers used in this state is less.
- The THUMB state register set is a subset of the ARM state set.
- The programmer has direct access to eight general purpose registers, R0—R7, the Program Counter (PC), a stack pointer register (SP), a link register (LR), and the CPSR.
- There are banked Stack Pointers, Link Registers and SPSRs for each exception mode.

THUMB State General Registers and Program Counter

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
SP	SP_fiq	SP_svc	SP_abt	SP_irq	SP_und
LR	LR_fiq	LR_svc	LR_abt	LR_irq	LR_und
PC	PC	PC	PC	PC	PC

THUMB State Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und	

 = banked register

Figure 4.4 The THUMB state registers (Reproduced with permission from ARM Limited. Copyright © ARM Limited)

THE RELATIONSHIP BETWEEN ARM AND THUMB STATE REGISTERS

- The THUMB state registers relate to the ARM state registers in the following way:
 1. THUMB state R0—R7 and ARM state R0—R7 are identical.
 2. THUMB state CPSR and SPSRs and ARM state CPSR and SPSRs are identical.
 3. THUMB state SP maps onto ARM state R13.
 4. THUMB state LR maps onto ARM state R14.
 5. The THUMB state Program Counter maps onto the ARM state Program Counter (R15).

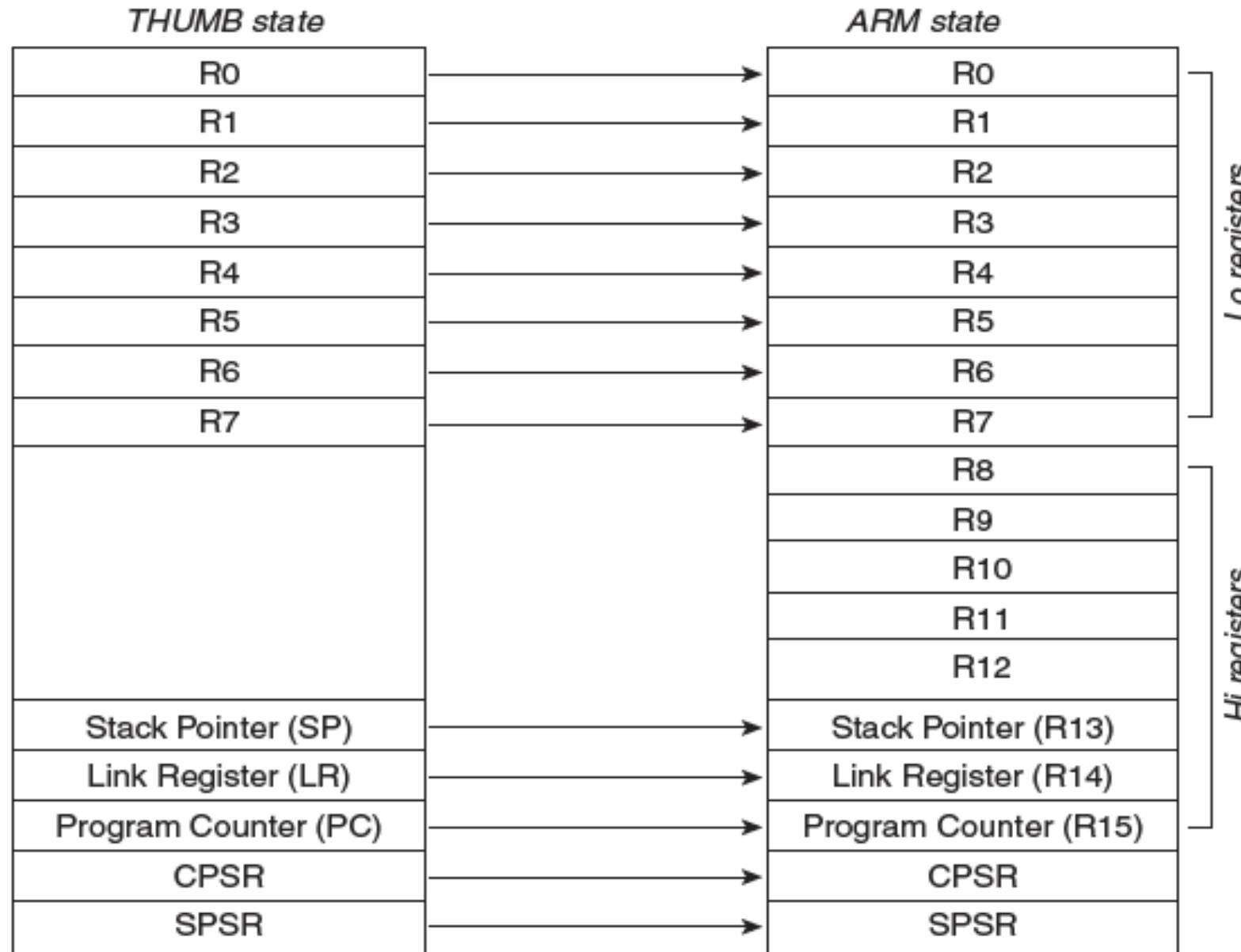


Figure 4.5 Relationship between the ARM and THUMB states (Reproduced with permission from ARM Limited. Copyright © ARM Limited)

4.3 INTERRUPTS AND EXCEPTIONS

- Exceptions/Interrupts are events that change the context of the processor
- They change the normal flow of the program
- The event may be triggered by software, a fault condition, a system call or an interrupt from a peripheral.
- The first thing to do when an exception occurs is to save the context of the processor
- Means that all working register contents must be saved
- The current content of the PC must also be saved

LOOK AT THE WAY THE PROCESSOR HANDLES AN EXCEPTION

1. The 'return address' which corresponds to the current PC value +8 is copied to the Link Register. The addition of 8 to PC is because the PC always contains the address of the instruction that is being fetched. Because of the three stage pipeline, the address of the instruction that is currently being executed is PC+8.
2. The content of CPSR is copied to the appropriate SPSR
3. The mode bits in the CPSR are changed to the mode of the exception mode to which switching has occurred.
4. The new PC value is the vector of the exception

Up on Completion of the exception handler (ISR)

The following steps must be taken to return to the context of the interrupted program.

1. The link register contents are used to get back the PC value.
2. The SPSR is copied back to the CPSR.
3. The interrupt disable flags, if they were set on entry, are cleared.

4.3.1 EXCEPTION/INTERRUPT VECTORS

- Each exception (**interrupt**) has a specific address to which control branches to.
- This address is called the 'vector' of the exception.
- Interrupts which have predefined and fixed vectors, are generally designated as 'vectored interrupts'. Table shows the vectors of the exceptions defined for ARM7.

Table 4.4 Exception vectors

Exception	Address
Reset	0x00000000
Undefined instruction	0x00000004
Software interrupt	0x00000008
Abort (prefetch)	0x0000000C
Abort (data)	0x00000010
Reserved	0x00000014
IRQ	0x00000018
FIO	0x0000001C

4.3.2 EXCEPTION/INTERRUPT HANDLERS

- What are the sources for interrupts?
- The current program running in the processor may be interrupted by an external device which requires service.
- A second case is when an **interrupt** occurs due to a software instruction. Another case is when an error occurs, like data abort or undefined instruction.
- In the first two cases, the new program **that comes** into play (called ISR or interrupt/ exception handler) performs the **service for the peripheral or acts upon** the interrupting instruction.

BUT WHAT EXACTLY HAPPENS WHEN AN EXCEPTION IS GENERATED ON AN ERROR?

- Here also control branches to the vector of the exception.
- The handler of the exception should contain the code which performs the necessary action to override the effect of the error.
- Thus, the system is prevented from crashing or going into a reset state before being able to save the current context.
- The ISRs of error-generated exceptions may be referred to as error handlers or fault handlers.
- These handlers allow the processor to exit gracefully from the error state without causing havoc to the system

4.3.2.1 RESET

- When the processor is switched on it is in the Reset condition.
- All the hardware is initialized with a 'reset pulse'.
- The mode that is entered on reset is the 'Supervisor' mode.
- The first instruction that is to be executed then is taken from the address Oxooooooo.
- In the course of the operation of the processors, reset may happen due to various reasons and then the system operation starts again from the reset vector.

4.3.2.2 UNDEFINED INSTRUCTION

- If an instruction is fetched and found to be a binary number which is not listed as one of the opcodes of the processor
- This exception is generated.
- The interrupt handler at the corresponding vector is to take appropriate action to exit gracefully from the error state.

4.3.2.3 SOFTWARE INTERRUPT

- The software interrupt instruction [SWI (now changed to SVC-supervisor call)] is used for entering Supervisor mode
- Usually to request a particular supervisor function.
- This is used by the OS to request access to protected resources.

4.3.2.4 ABORT

- An abort indicates that the current memory access cannot be completed.
- There are two types of abort:
 1. Prefetch abort occurs during an instruction prefetch.
 2. Data abort occurs during a data access (R/W), when data at an invalid address is attempted to be accessed.

4.3.2.5 PREFETCH ABORT

- It occurs when the processor prefetches code (could be a case of branching) which is at an invalid address.
- It may be that this address is in a protected memory area,
- Or this address is a peripheral address which has not been mapped to a peripheral.
- If a prefetch abort occurs, the prefetched instruction is marked as invalid, but the exception will not happen until the instruction reaches the execution unit of the pipeline.

4.3.2.6 RESERVED

Table 4.4 shows a vector for an exception that is not used for ARM7 but may be used for later processors.

4.3.2.7 IRQ

- The Interrupt Request (IRQ) exception is a normal interrupt caused by a LOW level on the IRQ pin.
- It is generally used by peripherals to get service.

4.3.2.8 FAST INTERRUPT REQUEST

- This is used for very important actions and is designed to be very fast.
- When this interrupt occurs, the processor goes to the FIQ mode, where a new set of registers are used, so as to avoid the delay for context saving.
- In practical situations, the applications which use the FIQ exception are coded in assembly to bring in an additional level of speed.

4.3.3 PRIORITY OF EXCEPTIONS

- When multiple exceptions arise at the same time, there is a priority prefixed to determine the order in which they are handled.
- The order is as follows:
 1. Reset (Highest priority)
 2. Data abort
 3. FIQ
 4. IRQ
 5. Prefetch abort
 6. Undefined Instruction, Software interrupt (Lowest priority)

4.4 ARM7 PIPELINE

- The idea of pipelining is well known in computer architecture.
- The simplest is a three-stage pipeline to overlap the basic operations of fetch, decode and execute.
- ARM 1 used a three-stage pipeline and this has been continued for ARM7.
- ARM9 uses a five-stage pipeline and ARM 10 uses a six-stage pipeline.

FIGURES SHOW THE THREE-STAGE PIPELINE OF ARM7.

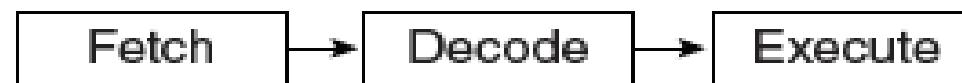


Figure 4.6 Three-stage pipeline of ARM 7

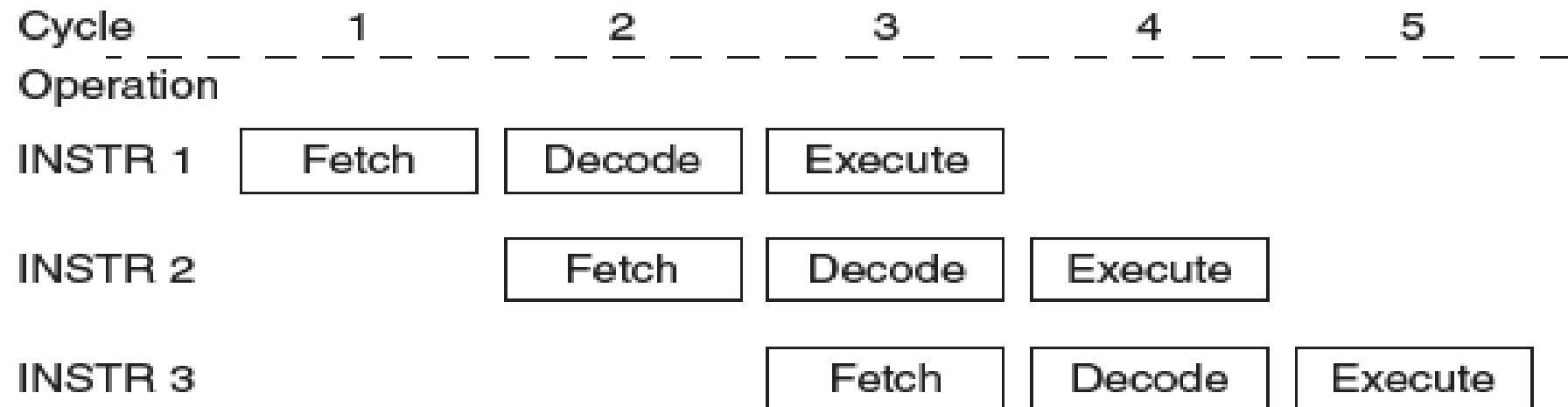


Figure 4.7 Pipeline filling in ARM 7

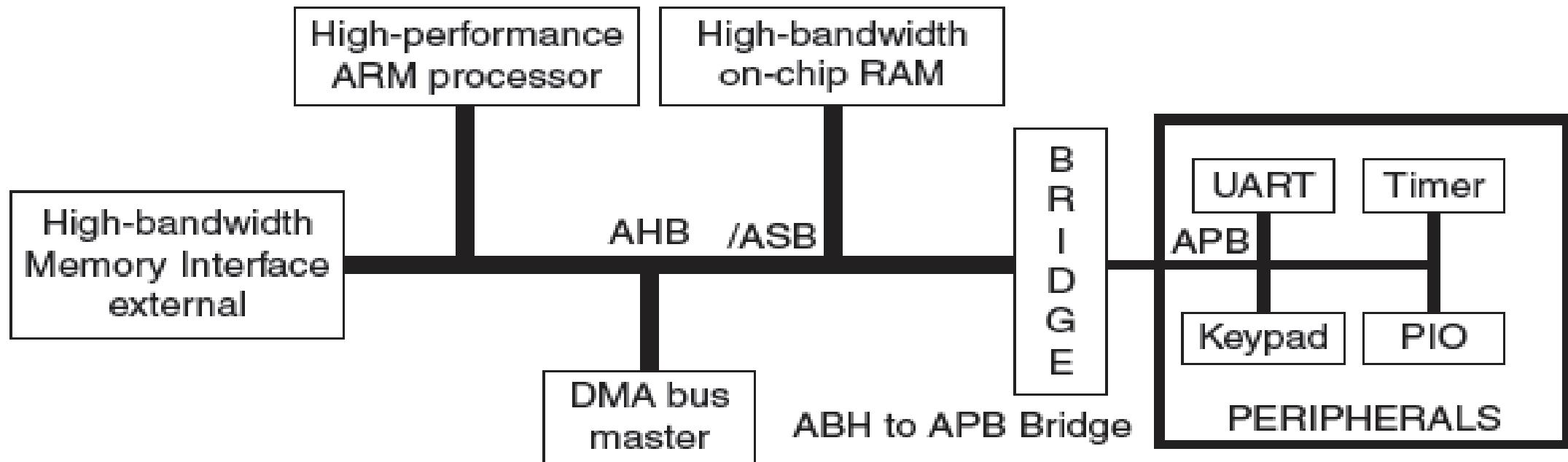
4.5 ADVANCED FEATURES

Now that we have covered the basic aspects of the ARM7 core, let us move on to some of its advanced features

4.5.1 AMBA

- AMBA is associated with ARM and it is a widely used interconnection standard for System on Chip (SoC) design.
- AMBA stands for Advanced Microcontroller Bus Architecture'
- In 1996 it was developed as a standard bus to facilitate the concept of modularity when adding on-chip functional blocks like memory and peripherals.
- The AMBA specification has become a well-accepted de-facto standard for the semiconductor industry
- used by 95% of ARM's partners and a number of IP providers.
- The AMBA interface is processor and technology independent

COMPONENTS OF AMBA BUS



AMBA Advanced High-performance Bus (AHB)

* High performance

AMBA Advanced Peripheral Bus (APB)

* Low power

Figure 4.8 Components of AMBA bus

AMBA SPECIFICATION

The AMBA specification defines three buses

1. Advanced System Bus (ASB) (which is now obsolete)
2. Advanced High-performance Bus (AHB)
3. Advanced Peripheral Bus (APB).

WHERE ARE THESE BUSES USED?

1. AHB is able to sustain the external memory bandwidth, on which the CPU, on-chip memory and other Direct Memory Access (DMA) devices reside. This part has to be a high-speed bus.
2. APB: This is the part which is used by the on-chip peripherals and operates at a lower speed.

- In 2003, ARM introduced the 3rd generation, AMBA 3,
- AXI to reach even higher performance interconnect and the Advanced Trace Bus (ATB) as part of the Core Sight on-chip debug and trace solution.
- This is used in Cortex processors, not on ARM7.

Figure shows that the high-performance modules are on the AHB/ ASB bus. There is a bridge which converts this standard to a lower speed peripheral bus. This will be discussed further in Chapter 6.

4.5.2 COPROCESSORS

- For ARM7 to ARM 11, additional and sometimes optional functions were obtained by the concept of coprocessors
- A coprocessor is an additional functional block which has its own instruction set
- When some function needs to be done, which is not supported by the general ARM core, a coprocessor is brought in to do it
- For example, the ARM core is not capable of high-end floating point arithmetic processing.

- When the need of such processing is needed, an arithmetic coprocessor is used, which operates in unison with the main core.
- Similar to this, there are coprocessors for the control of MMU, cache, DSP etc.
- A licensee of ARM, when designing a chip, has the option to add the coprocessor modules that is needed for his design.

- Each coprocessor has its own functional hardware with its instructions.
- Up to 16 coprocessors (CPO to C15) have been defined in the ARM cores.
- Some of the important architectures. All of them are not present in all core coprocessors of ARM7 are as follows:
 1. CP10: Vector Floating Point unit
 2. CP11: SIMD hardware and software named NEON
 3. CP14: Debug unit
 4. CP15: System control coprocessor for memory and cache management (including TCM)

4.5.3 MEMORY MANAGEMENT AND MEMORY PROTECTION

- Many ARM cores have only a memory protection unit (MPU).
- More advanced ARM chips have a memory management unit which includes protection features as well.
- Here the MMU of a typical ARM7 chip. ARM710T is one such processor.
- The MMU performs two primary functions:
 1. Translation of virtual addresses to physical addresses
 2. Provision of protection to memory by stipulating 'access permissions'

- The MMU has the following dedicated hardware to perform these functions:
 1. Translation Lookaside memory (TLB)
 2. Access control Logic
 3. Translation table walking logic
- ARM7 memory sizes are designated as either sections or pages.
- Sections are 1 MB blocks of memory. Pages may be small of size 4 KB, or large of size 64KB
- MMU also supports the concept of domains — which are areas of memory that can be defined to possess individual access rights.

4.5.3.1 USING THE TLB

- A TLB stores a certain number of translations of logical addresses to physical addresses.
- For the ARM7 MMU, 64 translated entries are saved in the TLB.
- If the required translation is available in the TLB, the 'access control logic' performs a permission check to verify if access is to be allowed.
- If access is permitted, the MMU outputs the appropriate physical address corresponding to the virtual address.
- If access is not permitted, the MMU signals the CPU to go to the abort mode through the abort exception.

Chapter 5

ASSEMBLY PROGRAMMING OF ARM7

INTRODUCTION

- Assembly Language Programming (ALP) is an efficient way of using the instruction set of a processor.
- Here coding is done in a symbolic language which is generally referred to as “mnemonics”.
- These mnemonic are directly converted into machine language and are executed by the processor.
- **Assembler:** Tool that does the translation from assembly language to machine language.
- Means its process is one-to-one process such that, each mnemonic gets translated to only a specific machine code.

5.1 EMBEDDED PROGRAM DEVELOPMENT

- ❑ ARM is an embedded processor and the code which is developed has finally to be 'burned' into its ROM.
- ❑ Code has to be developed, and thoroughly tested and verified to be correct and working.
- ❑ We need a host computer which is a PC, in which the development and testing of the code is done
- ❑ PC runs on an x86 processor, but the program that we are working on is for another processor's Instruction Set Architecture (ISA), that is, ARM.
- ❑ when the assembly process is complete, the binary code that is obtained is of a processor which is not the host computer (x86). So the conversion is termed as 'cross assembly'. If the source program is written in high level language, the translation is called 'cross compiling'.

5.1 EMBEDDED PROGRAM DEVELOPMENT (CONTD..)

- The host PC should have an IDE (Integrated Development Environment) which is a programming environment that has a code editor, cross compiler and assembler, debugger, simulator and graphical user interface (GUI).
- The IDE has knowledge of the registers, instruction set, memory and peripheral mapping of the embedded processor for which it is used.
- IDE of 8051 cannot be used for ARM.
- Our approach is to use assembly programming for the ARM core, and C programming with peripherals

POPULAR IDES IN USE FOR VARIOUS PROCESSORS.

Table 5.1 A short list of popular IDEs

IDE	MCUs for which they are used	Supplier
Keil RVDK	ARM, 8051	Keil Inc,
Eclipse	ARM	OPEN SOURCE
PSoC Creator	PSoC 3 and 5	CYPRESS SEMICONDUCTORS
PSoc Designer	PSoC 1	CYPRESS SEMICONDUCTORS
IAR	MSP 430, ARM, 8051	IAR SYSTEMS
Code Composer Studio	DSP Processors, MSP 430	TEXAS INSTRUMENTS
MPLAB	PIC	MICROCHIP TECHNOLOGY
AVR STUDIO	AVR MCU	ATMEL CORPORATION
CODEWARRIOR	ARM	FREESCALE SEMICONDUCTORS
VISUAL DSP++	Blackfin DSP processor	Analog Devices

NOTE TO REMEMBER

- ❑ When using the assembler, the results of the program are checked in registers and/or memory.
- ❑ When memory is being used, we may need to choose a specific processor, so that we can define the ROM and RAM spaces.
- ❑ We have chosen NXP's LPC 214x series, which has ROM starting at address 0x00000000 and RAM at 0x40000000.

5.1.1 FEATURES OF ARMV4T ISA

- The version of ISA used for ARM7 is v4T (the character 'T' means that Thumb instructions are also included).
- Feature are:
 1. In general, all Instructions are of size 32 bits for ARM and 16 bits for THUMB, though there are a few exceptions.
 2. There is a barrel shifter in the ALU, as shown in Fig. 5.1. This means that one of the operands may be shifted/rotated. This is useful for simplifying some types of operations.
 3. The use of the barrel shifter makes possible the combination of shift and ALU operations in a single instruction.
 4. All operands of data processing instructions are in registers.

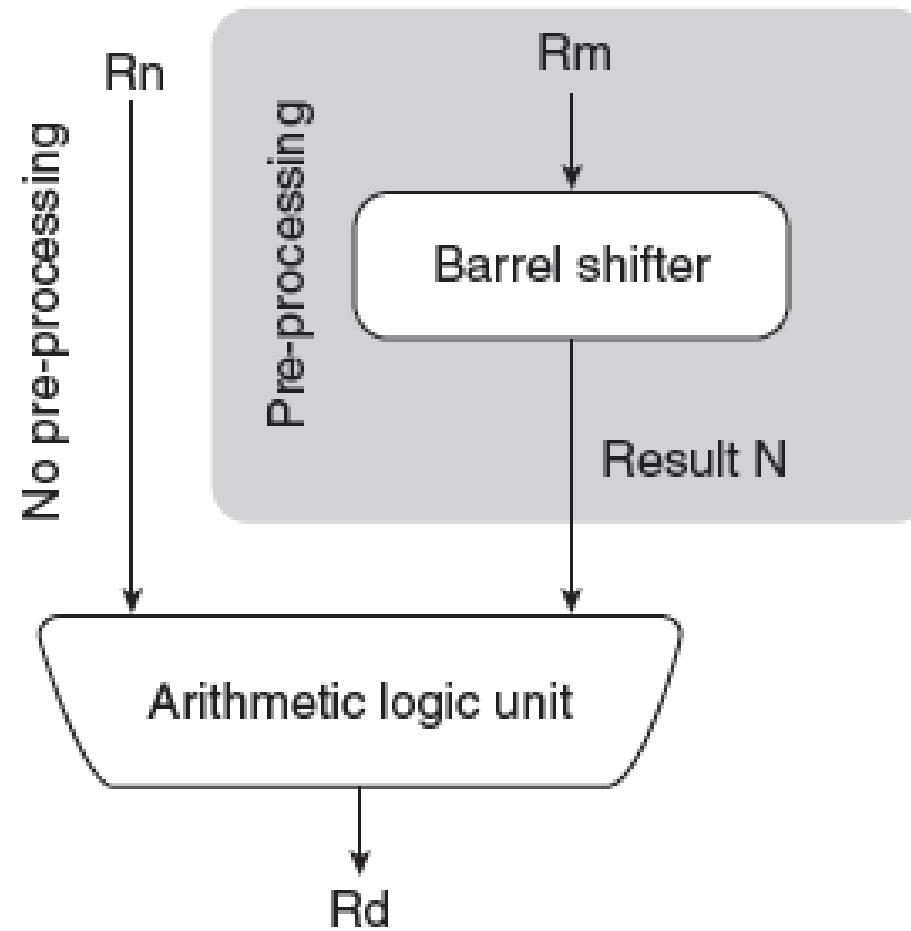


Figure 5.1 Barrel shifter in ALU

5. Only the load and store instructions access memory.
6. Most other processors use condition checking only with branch instructions. In contrast, here we find that many other instructions have conditions appended to it
7. There is a special technique for handling immediate instructions.
8. There are instructions to load/store data in multiple registers.
9. Arithmetic and logical instructions have a three operand format.

SOME POINTS TO NOTE BEFORE WE START ASSEMBLY PROGRAMMING

- The processor can opt to use the big endian or little endian format
- In our discussion here, we choose the little endian format.
- 8-bit (byte), 16-bit(half word) and 32-bit(word) data sizes are possible.
- Data alignment is necessary for efficiency.

5.1.2 ASSEMBLY LANGUAGE FORMAT



- A line in assembly language has the following format.

Label	Opcode	Operands	;comments
SAM	Add	R7,R6,R5	;add R6 and R5 & copy sum to R7

In the example line, SAM is the label, ADD is the opcode, and R7, R6, R5 are the operands.

What is written after the semicolon is a comment. It is good programming practice to include comments as part of programming.

5.2 ARM7 INSTRUCTION SET

- Lets start Assembly Language Programming by looking at the instruction set.
 - The instructions can be classified into the following different types:
 - I. Data processing instructions
 - II. Load store instructions — single register, multiple register
 - III. Branch instructions
 - IV. Status register access instructions
- We will discuss only the first three types.

5.2.1 DATA PROCESSING INSTRUCTIONS

- This set includes move, arithmetic, logical and compare instructions.
- Lets start with the concept of the barrel shifter which is part of the ALU.
- A barrel shifter is a unit which can shift and rotate operands by any amount.
- Figure shows that when there are two source operands in the ALU, one of them may be shifted/rotated any number of times
- Note that the maximum size of any register is 32 bits
- so the barrel shifter operations are not needed for shift/rotations beyond 32 times.

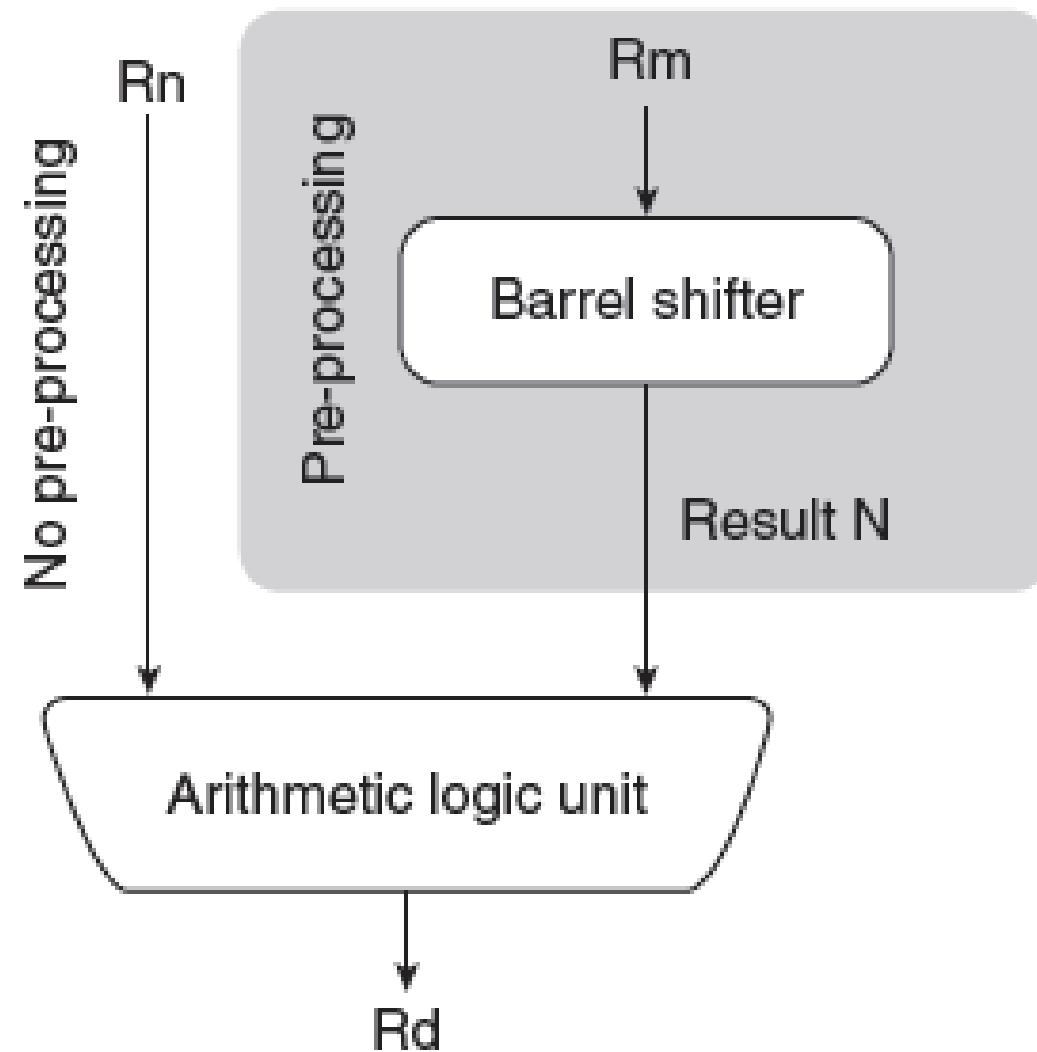


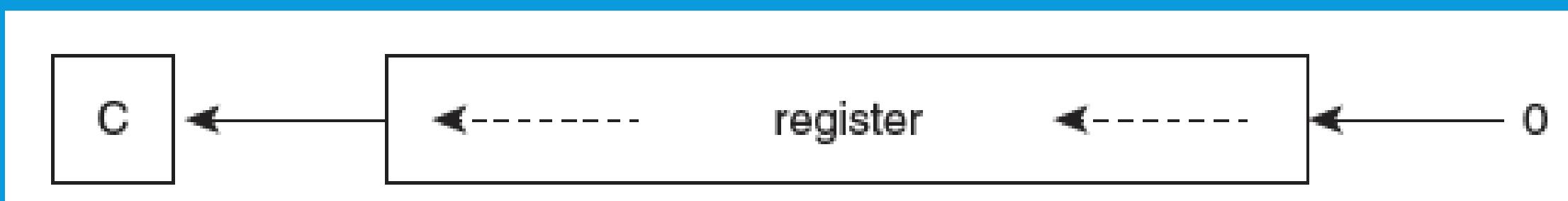
Figure 5.1 Barrel shifter in ALU

Table 5.2 List of shift and rotate operations

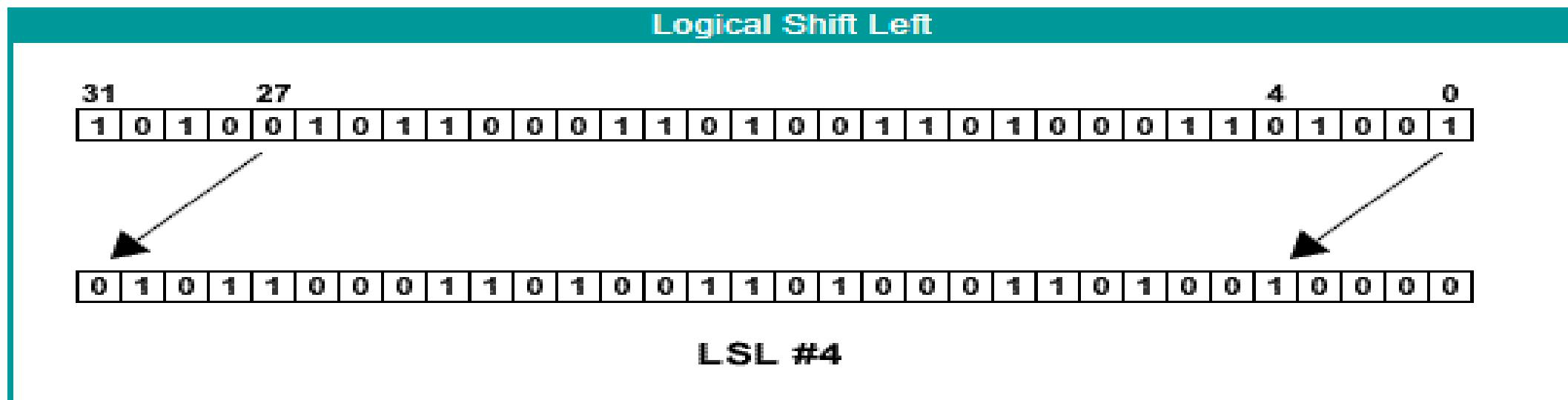
Mnemonic	Description
LSL	logical shift left
LSR	logical shift right
ASR	arithmetic shift right
ROR	rotate right
RRX	rotate right extended

5.2.1.1 LOGICAL SHIFT LEFT (LSL #n)

- When the shift amount is specified in the instruction, it is contained in a 5-bit field which may take any value from 0-31.
- A logical shift left (LSL) takes the contents of the specified register and shifts it left.
- A left shift by one bit position is equivalent to multiplication by 2.
- For example, the effect of LSL #4 causes a multiplication by 2^4 , that is, by 16.
- The least significant bits of the result are filled with zeros, and the high bits of the register are discarded.
- The most significant discarded bit goes into the Carry flag.



LSL is a logical shift left by 0 to 31 places. The vacated bits at the least significant end of the word are filled with zeros.



EXAMPLE

Consider the following ARM instruction with $r1 = 3$ and $r2 = 5$

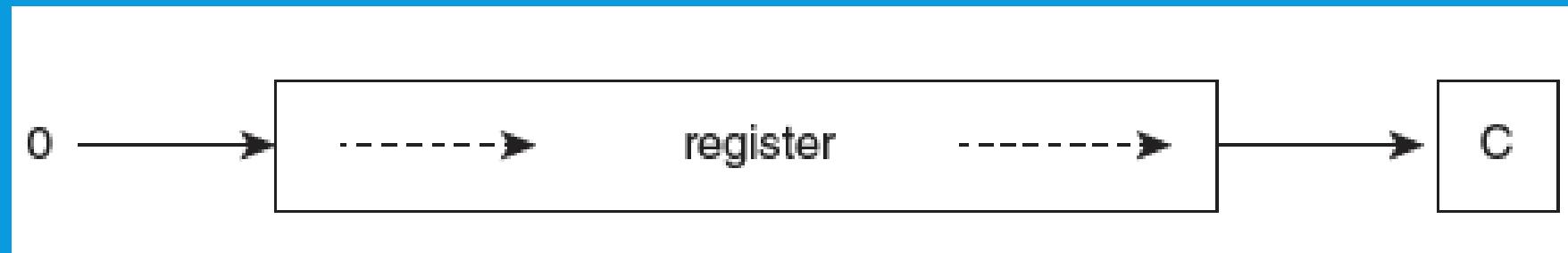
`ADD r0, r1, r2 ; r0 := r1 + r2 which is r0 := 3 + 5 = 8`

Now try the same instruction with a LSL operand, say LSL #3 (logical shift left 3 places which is equivalent to multiplying by 8 (2^3)):

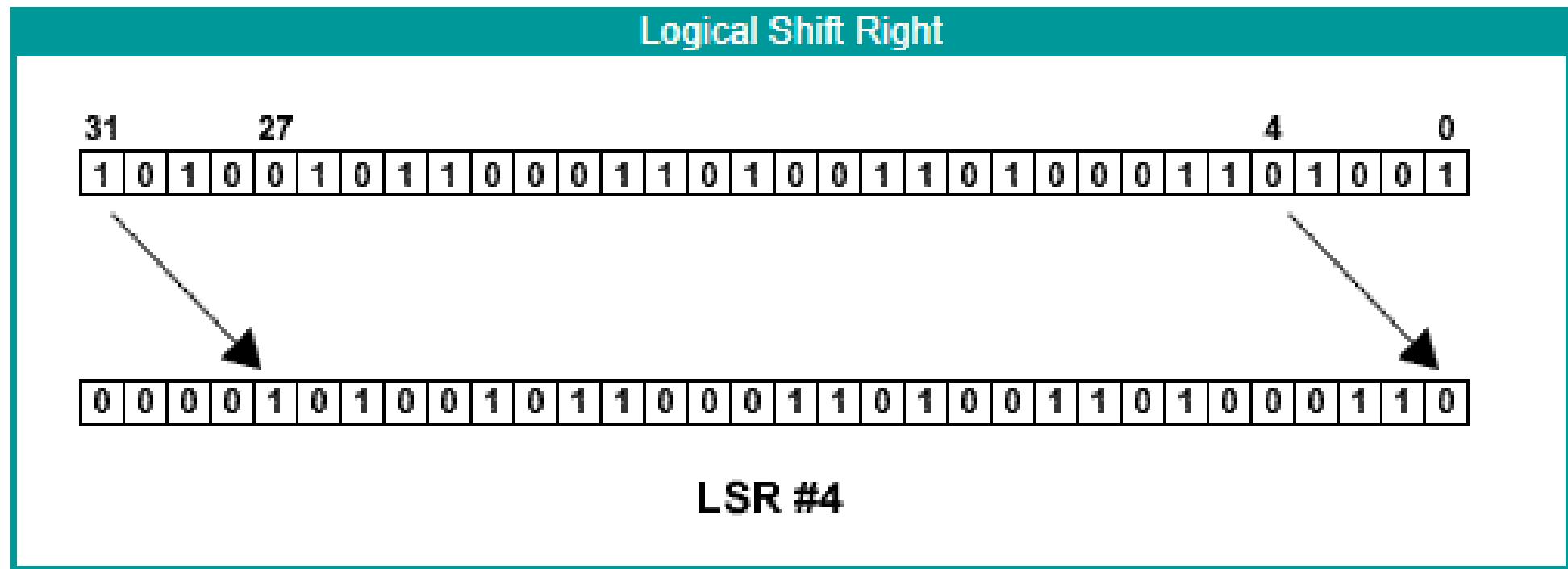
`ADD r0, r1, r2, LSL #3 ; r0 := r1 + (8 x r2) which is r0 := 3 + (8 x 5) = 43`

5.2.1.2 LOGICAL SHIFT RIGHT (LSR #n)

- This is similar to LSL but the shifting is towards right.
- Shifting right by each bit position is equivalent to a division by 2.
- Thus, LSR #5 causes the data in the specified register to be divided by 32.



LSR is a logical shift *right* by 0 to 32 places. The vacated bits at the most significant end of the word are filled with zeros.



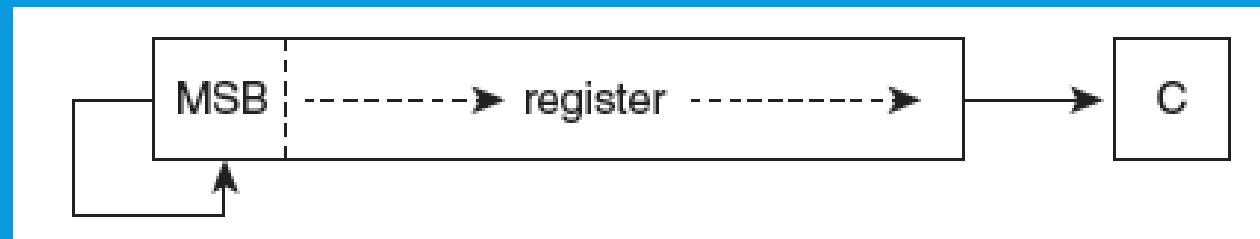
Consider a MOV instruction with r1 = 168 and r2 = 3:

MOV r0, r1, LSR r2	; shift the binary value of 168 3 places to the right
	; 168 = 0000 0000 0000 0000 0000 0000 1010 1000
	; shifted 3 places,
	; becomes 0000 0000 0000 0000 0000 0000 0001 0101
	; r0 := 21

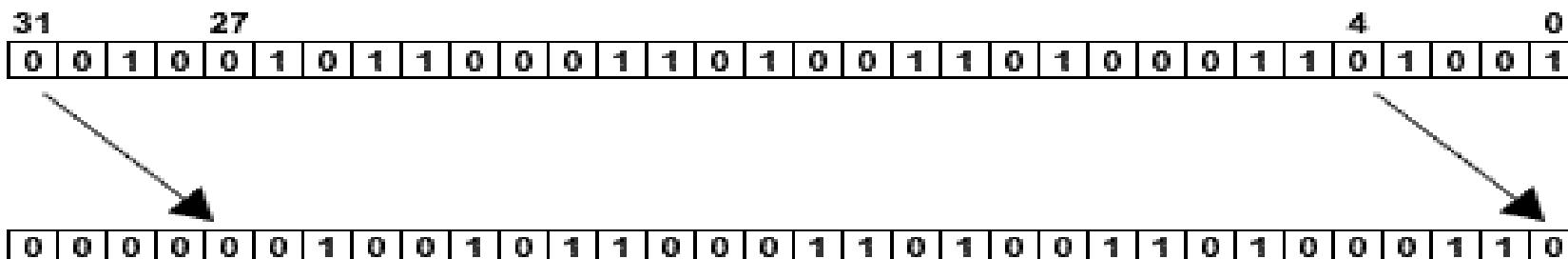
For positive numbers, LSR 3 is the same as dividing by 2^3 (8).

5.2.1.3 ARITHMETIC SHIFT RIGHT (ASR #n)

- An arithmetic shift right (ASR) is similar to logical shift right, except that the high bits are filled with the MSB of the register instead of zeros.
- This preserves the sign in 2's complement notation. Thus, sign extension is done on the data, along with it being divided by 2 for each shift.
- Thus, ASR #4 causes a division by 16

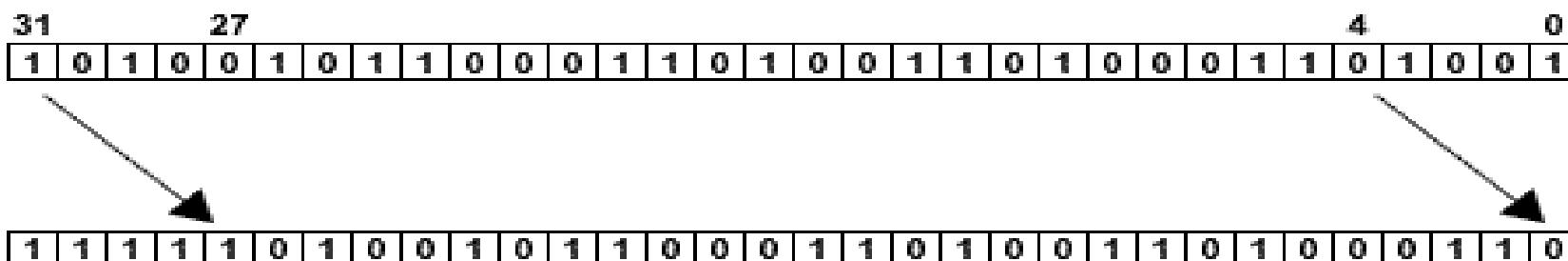


Arithmetic Shift Right Positive Value



Arithmetic Shift Right Positive Value

Arithmetic Shift Right Negative Value



ASR #4 negative value

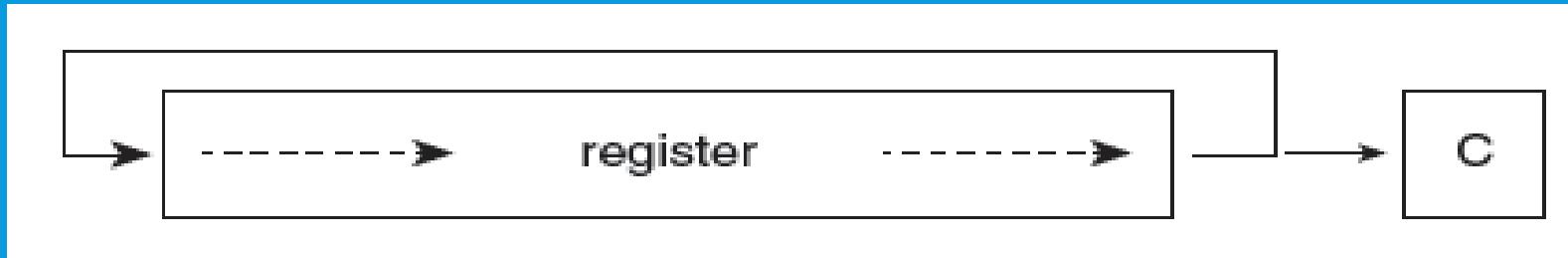
To Show The Difference Between Logical And Arithmetic Right Shifts, Consider A LSR and ASR Where The Value To Be Shifted Is 112 And The Shift Is 3 Places. To Keep The Mathematics Simple, We Will Use 8-bit Numbers.

- The bit pattern for 112 is 0111 0000.
- Performing LSR #3 transforms the bit pattern to be 0000 1110
- Result is 14 : 14 to base 10
- Performing ASR #3 again transforms the bit pattern to 0000 1110
- Now what happens if the number is -112?
- First create the 2's complement bit pattern for -112 by flipping the bit pattern of 112 and adding 1.
- 0111 0000 becomes $(1000\ 1111 + 1)$ becomes 1001 0000
- Now perform a LSR #3 operation on 1001 0000.

- This shifts the bits 3 places to the right and fills in the vacated bits with 0s:
- 1001 0000 becomes 0001 0010 which is +18 to Base 10
- Now instead perform an ASR #3 operation on 1001 0000.
- This shifts the bits 3 places to the right and fills the vacated bits with copies of the most significant bit, i.e. 1's.
- 1001 0000 becomes 1111 0010 which is -14 to base 10.

5.2.1.4 ROTATE RIGHT (ROR #n)

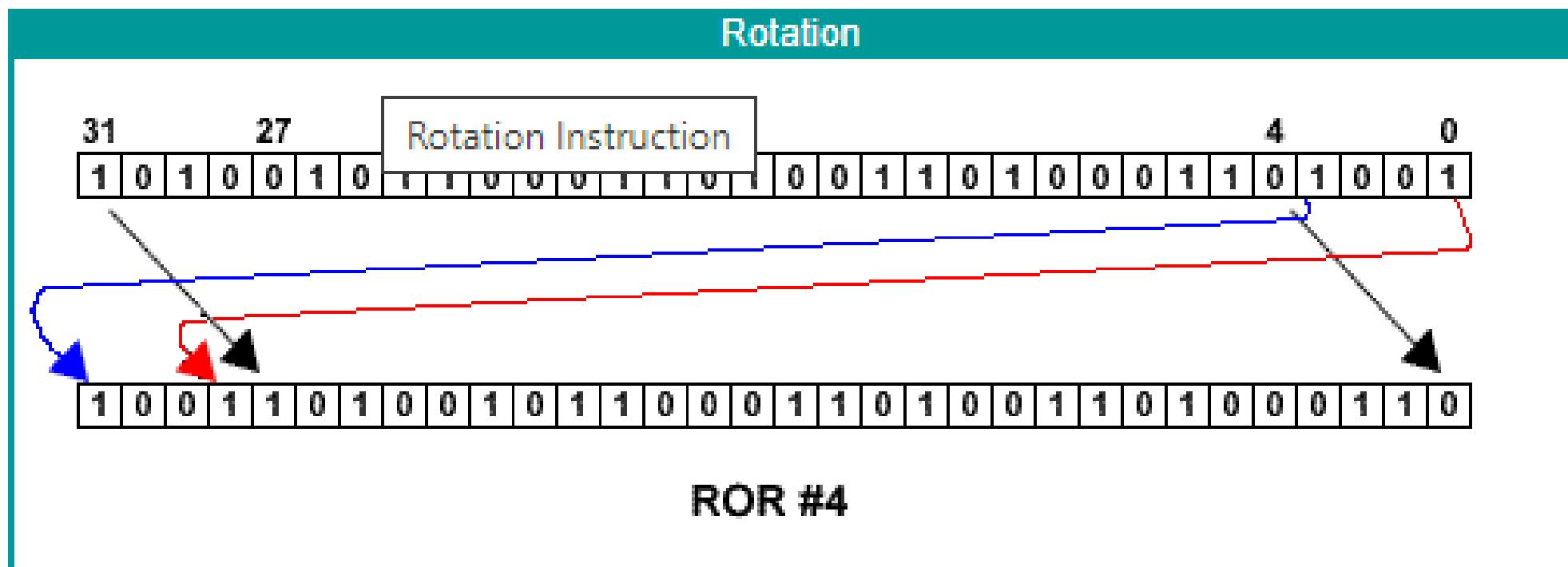
- Right shifting is done and the bits which overflow are reintroduced at the left end of the register.
- The last bit shifted out, is put in the carry flag as well.



- There is no 'rotate left' instruction, because left rotation by n times can be achieved by rotating to the right $(32 - n)$ times.
- For example, rotating 4 times to the left is achieved by rotating $32 - 4 = 28$ times to the right.

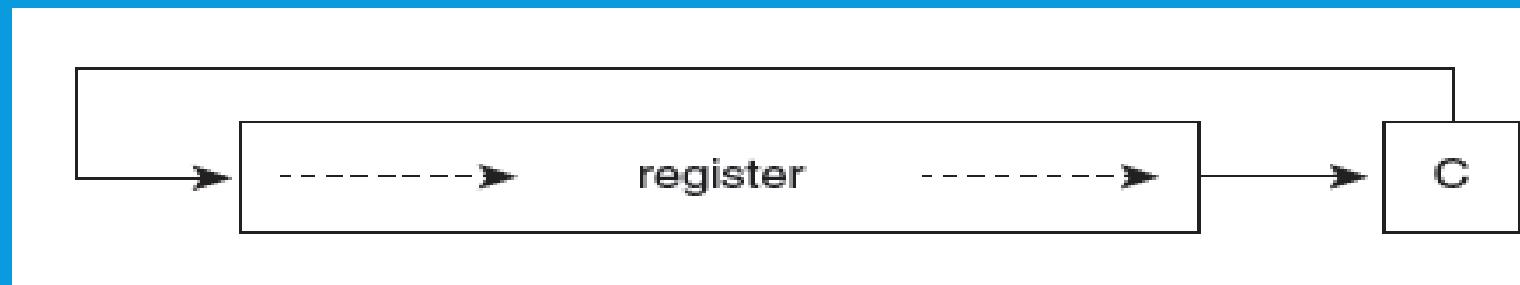
MOV r0, r0, ROR #3 ; rotate value in r0 3 places and put result in r0

MOV r2, r0, ROR r1 ; rotate r0 by value of places in r1 and put result in r2



5.2.1.5 ROTATE RIGHT EXTENDED (RRX #n)

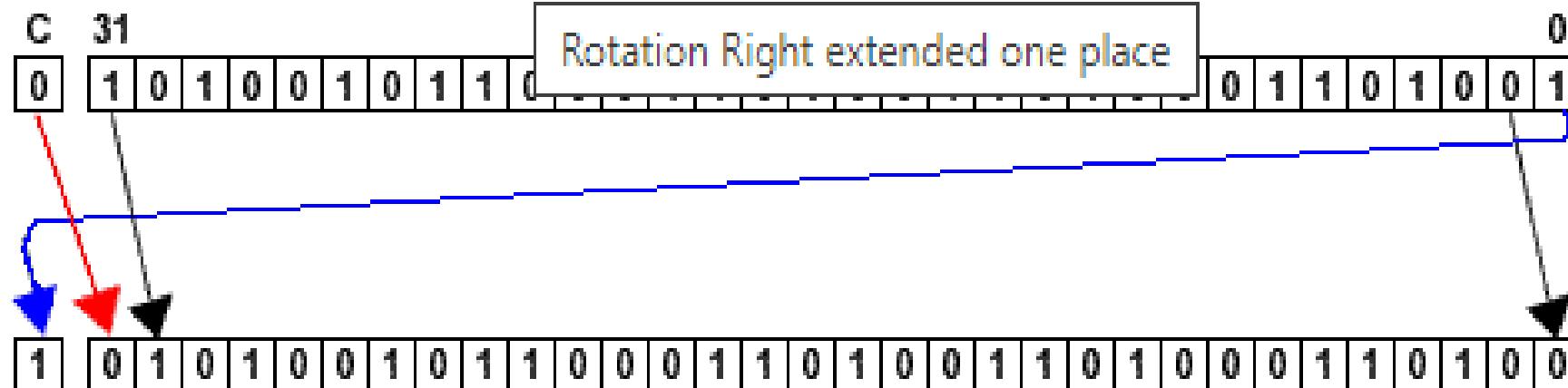
- RRX is a ROR operation with a crucial difference.
- It rotates the number to the right by one place but the original bit 31 is filled by the value of the Carry flag and the original bit 0 is moved into the Carry flag.



- This allows a 33-bit rotation by using both the register and the carry flag.

MOV r0, r0, RRX ; rotate right extended and put result back into r0

Rotation Right extended one place



RRX

PROBLEMS

Find the results of the following operations given the contents of
R1=0x0089EF32, R2=0x80456730, R3=0x8

1. LSL R1,#8
2. LSR R1,#12
3. ASR R2,R3
4. ROR R2,R3

ANSWERS

1. LSL R1,#8

R1=0000 0000 1000 1001 1110 1111 0011 0010

Left shifting logically 8 times gives

R1= 1000 1001 1110 1111 0011 0010 0000 0000

2. LSR R1,#12

R1=0000 0000 1000 1001 1110 1111 0011 0010

Logically shifting it right 12 times gives,

R1= 0000 0000 0000 0000 0000 1000 1001 1110

3. ASR R2,R3

R2=1000 0000 0100 0101 0110 0111 0011 0000

Arithmetic right shift will do sign extension (the MSB is '1' here)

R2 = 1111 1111 1000 0000 0100 0101 0110 0111

4. ROR R2,R3

R2=1000 0000 0100 0101 0110 0111 0011 0000

Rotating it right eight times will make it

R2=0011 0000 1000 0000 0100 0101 0110 0111

5.2.1.6 MOV Instruction (MOV, MVN)

- The most frequently used instruction in any processor is the one that copies from a source to a destination.
- In ARM, the 'MOV' instruction does the copying of data to a destination register.
- The simplest formats of the MOV instruction are as follows:

1. MOV dest, src	;Move (copy) the contents of the source to the destination
2. MVN dest, src	;Move (copy) the negated value of source to destination
MOV R1, R3	;copy the value in R3 to R1
MVN R5, R9	;copy the negated value of R9 to R5
MOV R4,#45	;copy the immediate 8 bit number 45 to R4

EXAMPLE 1

EXAMPLE 5.2

If $R1=0$ and $R2=0x896700FF$, what is the content of the destination registers after the execution of the following instructions?

1. **MOV R3,R1**
2. **MVN R4,R1**
3. **MVN R5,R2**
4. **MOV R6,#235**

SOLUTION

1. **R3=0**
2. **R4=0xFFFFFFFF**
3. **R5=0x7698FF00**
4. **R6=0x000000EB** (235 in 32 bit hex format)

EXAMPLE 2

EXAMPLE 5.3

Find the result in the destination register after the execution of the following instructions, if

R6 =0x FFEE1100 and **R2=8**.

1. **MOV R10,R6, LSR #2**
2. **MVN R9, R6, LSL #4**
3. **MOV R8, R6, ASR R2**

SOLUTION

1. **R10=0x3FFB8440** ;the content of R6 is right shifted twice and moved to R10
2. **R9= 0x011EEFFF** ;the content of R6 is left shifted 4 times, negated and; and moved to R9
3. **R8= 0xFFFFEE11** ;the content of R6 is right shifted arithmetically twice; and moved to R8

5.2.1.7 THE SUFFIX 'S'

- For any data processing instruction to have the capability to update the conditional flags, it must be suffixed with the character 'S'.
- If this suffix is not appended to the instruction, the updating of flags does not occur.
- For example, the MOV instruction may be appended as MOVS.
- Other instructions which we will see presently, may also be appended with the suffix S

IF R1=0, R2=0x896700FF,

What is the content of the destination registers after the execution of the following instructions? Which flags are updated for each case?

1. **MOVS R3,R1**
The destination register R3 contains 0x0. The Z flag is updated to 1, indicating that the destination register is zero.
2. **MVNS R4,R1**
The destination register R4 contains 0xFFFFFFFF. The N (Negative) flag is set to 1 indicating that the MSB of the destination is 1, that is, a negative number is in the destination register.
3. **MVNS R5,R2**
The destination register R5 contains 0x7698FF00. None of the conditional flags are set. N=0, as the MSB is not '1', because a positive number is in the destination register.

5.2.1.8 CONDITIONAL EXECUTION

- One of ARM's special features is that it can execute any instruction conditionally.
- The word conditionally implies that the state of the condition flags are checked before it is decided whether the instruction should be executed or not.
- If the required condition is not met, the instruction becomes a NOP (No Operation).

LIST OF CONDITION CODES and the FLAGS UPDATED CORRESPONDINGLY

Table 5.3 List of condition codes

Code	Suffix	Flags	Meaning
0000	EQ	Z set	equal
0001	NE	Z clear	not equal
0010	CS	C set	unsigned higher or same
0011	CC	C clear	unsigned lower
0100	MI	N set	negative
0101	PL	N clear	positive or zero
0110	VS	V set	overflow
0111	VC	V clear	no overflow
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear or Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal to V	less than
1100	GT	Z clear AND (N equals V)	greater than
1101	LE	Z set OR (N not equal to V)	less than or equal
1110	AL	(ignored)	always

Format of a Data Processing Instruction



- Figure shows the instruction format of a typical data processing instruction.
- Looking at Fig, we see that four bits are allocated to specify the condition.
- The instruction produces a result by performing a specified arithmetic or logical operation on one or two operands. The first operand is always a register (Rn).
- The second operand may be a shifted register (Rm) or a rotated 8 bit immediate value.
- Rd is the destination register.
- The flag bits in the CPSR may be updated if the instruction has been appended with 'S'.
- Otherwise, the flags are not affected.

How Do We Use Conditional Execution?

Let us take a few examples.

MOVEQ R2,R4

- The above instruction checks the flags associated with the condition `EQ:(equal). Referring to Table 5.3, we see that the Z flag should be set ($Z=1$), is the condition specified.
- After this instruction is fetched and decoded, the condition (whether Z flag is set) is checked before this instruction comes to the execution stage. If found to be true, the execution of copying the content of R4 to R2 is done. But if $Z=0$, the instruction simply becomes an NOP, that is, it does not get executed.

MOVHI R5,R6

- Before it is decided whether these instructions should be executed or not, the flags associated with the 'HI' condition is looked at. Table 5.3 shows that C=1 and Z=0 are the conditions.
- If a previous instruction had updated the flags and these conditions are met, the MOVHI instruction gets executed — otherwise it becomes a NOR

What Are The Pros And Cons Of Such Conditional Execution (Which Is Not Done Inmost Other ISAs)?

The plus points (pros) are:

- ❖ Most ISAs use conditions only with branch instructions. Here, conditions may be appended to any other instruction, and the condition evaluation hardware is shared for many instructions. The end result is that the effective number of instructions in the ISA becomes more than as seen directly. This is achieved without additional hardware.
- ❖ Codes become dense and branching is avoided to a great extent
- ❖ By avoiding branching (unless absolutely necessary), pipeline stalling is prevented

What Are The Pros And Cons Of Such Conditional Execution (Which Is Not Done Inmost Other ISAs)?

The minus points (cons) is:

- ❖ When an instruction becomes a NOP, one cycle time is wasted.
- ❖ This is because, even though execution of the instruction is not done, 'fetching and decoding' cannot be avoided.

5.2.1.9 ARITHMETIC AND LOGICAL INSTRUCTIONS

- The general format is
- $\langle \text{opcode} \rangle \{ \text{cond} \} \{ S \} \text{Rd}, \text{Rn} \langle \text{Op2} \rangle$ where, Rd is the destination register, and Rn and Op2 are the source operands.

5.2.1.9 ADDITION AND SUBTRACTION

lists of addition and subtraction instructions of this ISA

Table 5.4 List of ADD and SUBTRACT instructions

Operation	A typical instruction	Computation
Add	ADD R5, R0,R1	$R5=R0+R1$
Add with carry	ADC R5,R0,R1	$R5=R0+R1+C$
Subtract	SUB R5,R0,R1	$R5=R0-R1$
Subtract with carry	SBC R5,R0,R1	$R5=R0-R1-!C$
Reverse Subtract	RSB R5,R0,R1	$R5=R1-R0$
Reverse Subtract with carry	RSC R5,R0,R1	$R5=R1-R0-!C$

Take a look at this code snippet

```
ADDS R2,R3,R4  
MOVCS R10,R2  
MOV R9,R2
```

What is done here?

The first line adds the content of R3 and R4 and saves the sum in R2. Since the suffix 'S' has been appended to the ADD instruction, the flags are updated. The next line has a condition 'CS' for checking the content of the carry flag (Ref Table 5.3). If the addition has caused a 'carry', the second instruction is executed and the content of R2 is copied to R10. If the carry flag is not found to be set, the second instruction becomes a NOP and execution proceeds to the third instruction in which R2 is copied to R9. The end result is that R10 will have the sum of R3 and R4 only if the addition causes a 'carry'. Otherwise, only R9 will have this sum. Note that the third instruction is an unconditional one.

Table 5.5 List of logical instructions

Operation	A typical instruction	Computation
AND	AND R5, R0,R1	$R5=R0 \text{ AND } R1$
OR	ORR R5,R0,R1	$R5=R0 \text{ OR } R1$
EX-OR	EOR R5,R0,R1	$R5=R0 \text{ EX-OR } R1$
Bit Clear	BIC R5,R0,R1	$R5=R0 \text{ (AND NOT) } R1$

EXAMPLE

Find the output of the following instructions given that R2=0x0F, R1=0xF0

1. AND R3,R1,R2
2. ANDS R3,R1,R2
3. BIC R5,R1,R2

SOLUTION

1. **R3=0x00** The destination register has '0' as its content. But Z flag is not set.
2. **R3=0x00** Here the 'Z' flag is set, because the instruction has the 'S' suffix.
3. **R5= 0xF0** Here the operation is that, R1 does an AND operation with the complement of the second operand(R2). The result shows (as '1') which of the bits of the first operand (R1) are '0's.

5.2.1.10 COMPARE Instruction

- There are four instructions for comparison.
- Table shows the list and the operations performed for each of them.
- The important feature of these instructions is that they do not produce a result — they only update the flags.
- Thus, the 'S' suffix is not needed for them.
- The format for these instructions is
- <opcode>{cond}Rn, Operand2

Table 5.6 List and operations of compare instructions

Operation	A typical instruction	Computation
Compare	CMP R1,R2	R1-R2
Compare negated	CMN R1,R2	R1+R2
Test on Equal	TEQ R1,R2	R1 XOR R2
Bit test	TST R1,R2	R1 AND R2

Table 5.7 Flag setting after a compare instruction

If	C	Z	N
R1 > R2	1	0	0
R1 < R2	0	0	1
R1 = R2	1	1	0

PROBLEM

R1=0x0,R2=0xF

What is the result and relevance of the following two instructions?

1. **TST R1,R2**
2. **TEQ R2,#0x0F**

SOLUTION

1. Since the ANDing of the two registers produces a '0', the Z flag is set. Here the second operand R2 is said to be a 'mask' and it verifies if any bit of the first operand has at least one '1'. The fact that Z=1, makes it clear that R1 does not have a '1' in the lowest 4 bit positions.
If R1=01 and R2=0xFF, the Z flag is not set. Thus, it is confirmed that the R1 has at least one '1'.
2. Since the Ex-or-ing of the two registers produce a zero, the Z flag is set. TEQ does exclusive ORing which tests for equality. If both the operands are equal, the Z flag is set. It verifies if the value in a register is equal to a specified number which is the second operand.

PROBLEM

Now let us use some arithmetic with shifts and rotates (All results of shifts must remain within the 32 bit range of the register size).

1. **MOV R10,R10,LSL #2**
2. **MOV R2,R2,LSR #2**
3. **MOV R5,R5,ASR #2**
4. **ADD R5,R5,R5,LSL #4**
5. **RSB R3, R4, R4, LSL #4**
6. **SUB R1, R1, R1,LSL #3**

SOLUTION

1. $R10=R10$ left shifted by 2, which is $2^2R10 = 4R10$
2. $R2=R2$ divided by $2^2 = R2/4$
3. $R5=R5$ divided by 2^2 , but signed.
4. $R5=R5+R5 \times 2^4 = R5+16R5 = 17R5$
5. $R3 = 16R4 - R4 = 15R4$
6. $R1 = R1 - 8R1 = -7R1$

PROBLEM

How can 64 bit addition be done using the 32-bit ARM processor?

Let the 64 bit numbers be in two sets of registers say
R1, R0 (R0 holding the lower 32 bits)
R3, R2 (R2 holding the lower 32 bits)
The program lines are

ADDS R2,R2,R0
ADC R3,R3,R1

The 64-bit sum will be R3 and R2

If the result is expected to be more than 64 bits, the code is

ADDS R2,R2,R0
ADCS R3,R3,R1

The sum will then be in the Carry bit (MSB) and R3,R2. It will be a 65-bit number.

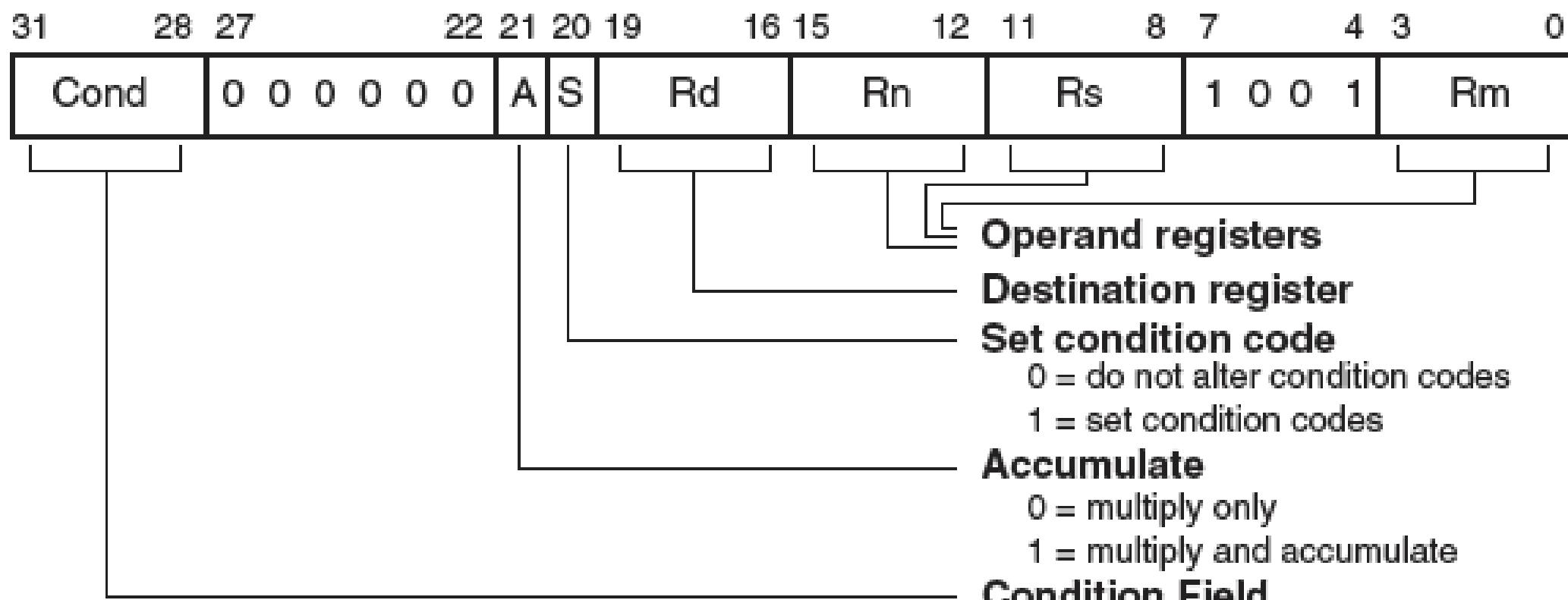
5.2.1.11 MULTIPLICATION

- There are two basic multiplication instructions.
- One is 'Multiply' and the other is 'Multiply and Accumulate'
- The word 'Accumulate' indicates 'Addition'
- Instruction format for the both is:

MUL{<cond>} {S} Rd, Rm, Rs ; $Rd = Rm * Rs$

MLA{<cond>} {S} Rd, Rm, Rs, Rn ; $Rd = (Rm * Rs) + Rn$

Figure 5.8 Instruction format of multiplication instructions



PROBLEM

Examples

MUL R1,R2,R3

;R1=R2xR3

MULS R1, R2, R3

;R1=R2xR3 and flags are updated

MULEQ R1,R2,R3

;R1=R2xR3 executed only if Z flag is set

MLA R4, R3, R2, R1

;R4 = R3xR2+R1

MLAEQS R1,R2,R3,R4

;R1:=R2*R3+R4 -executed only if Z=1 ;On execution, flags are updated

Restrictions on the use of the multiply instructions:

- The source and destination registers are 32 bits in length. If the product is longer than 32 bits, only the lower bits are preserved in the destination register.
- Immediate data cannot be used as a source operand
- The PC (R15) cannot be used as any of the registers
- Operands can be considered signed or unsigned. The user must be able to interpret correctly, the results of the multiplication.

Table 5.8 List of extended multiplication instructions

Mnemonic	Description	Purpose
UMULL{cond}{S} RdLo,RdHi,Rm,Rs	Unsigned Multiply Long	$32 \times 32 = 64$
UMLAL{cond}{S} RdLo,RdHi,Rm,Rs	Unsigned Multiply & Accumulate Long	$32 \times 32 + 64 = 64$
SMULL{cond}{S} RdLo,RdHi,Rm,Rs	Signed Multiply Long	$32 \times 32 = 64$
SMLAL{cond}{S} RdLo,RdHi,Rm,Rs	Signed Multiply & Accumulate Long	$32 \times 32 + 64 = 64$

5.2.1.12 DIVISION

- ❖ There is no explicit divide instruction in this ISA
- ❖ Division can be accomplished by repeated subtraction.
- ❖ Later examples we can see the options.

5.2.1.13 BRANCH Instruction

- Branch instructions are the ones that give the power of decision making to a computer.
- For ARM, there are no jump or call instructions as in the case of other processors.
- The branch instruction with the mnemonic 'B' suffices for all jumping operations and procedure calls.

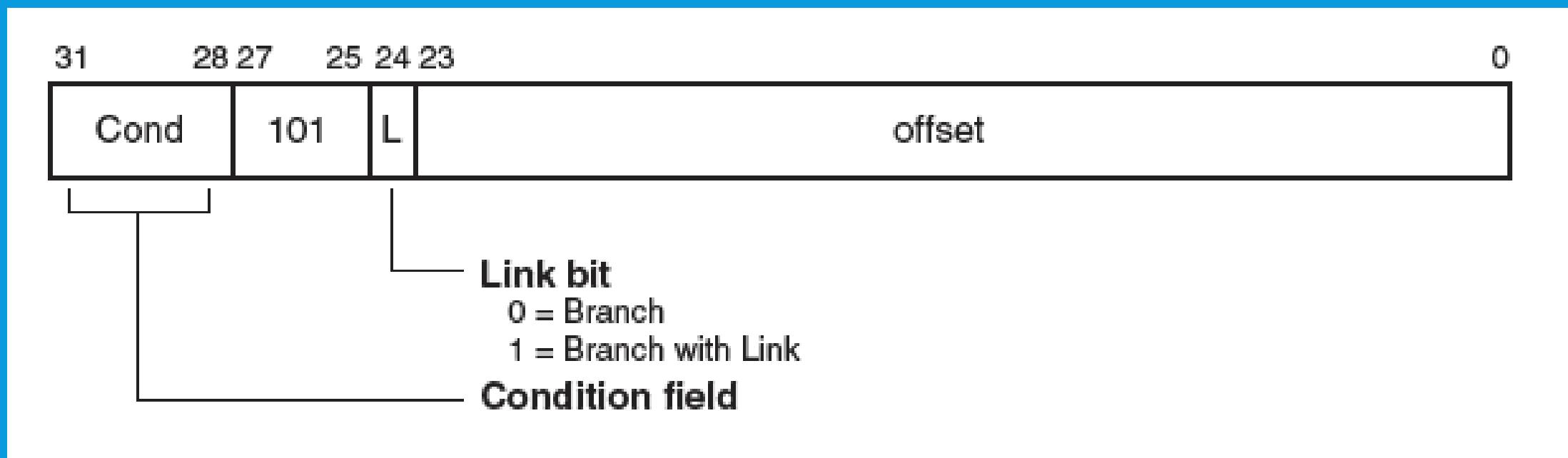
5.2.1.13.1 BRANCH (B)

- The general format of a branch instruction is 'B label'
- 'label' corresponds to the address of the target location.
- Condition codes may be appended to make it a conditional branch instruction.
- The assembler finds the target address with the use of the offset specified in the instruction.

Table 5.9 Branch instructions

Mnemonic	Instruction
B	Branch
BL	Branch and link
BX	Branch and Exchange
BLX	Branch Exchange with link

Figure 5.9 Format of branch instruction



5.2.1.13.2 BRANCH With LINK (BL)

- This instruction saves the current PC into the Link register (R14).
- Thus, BL facilitates subroutine/procedure calls.
- When this instruction is encountered, branching occurs to the target address and the 'return address' (the address to which control must come back after the procedure is ended), is saved in the link register.

Write an ALP (assembly language program) to add the first 5 natural numbers. The sum must be in a register.

AREA NAT, CODE, READONLY

ENTRY

MOV R0,#0 ;R0=0

MOV R1,#0 ;R1=0

BACKK ADD R0,R0,#1 ;Increment R0 by 1

ADD R1,R1,R0 ;Add the numbers. The sum is in R1

CMP R0,#5 ;check if R0=5

BNE BACKK ;repeat if R0 is not equal to 5

GO B GO ;continue branching to GO

END ;end of the assembly module

5.3 ASSEMBLY LANGUAGE PROGRAMMING

- Now we can start assembly programming using the Keil Microvision V assembler.
- The step-by-step method of using this IDE for Assembly Language

5.3.1 ASSEMBLER Directives

- For each assembler, there are 'directives'.
- They are specific for an assembler and are non-executable statements, which means that they don't cause any 'execution' by the processor.
- Directives are related solely to the assembler.
- We need a few directives of the Keil Assembler to get started in assembly programming.

5.3.1.1 AREA

- The “AREA” directive instructs the assembler to assemble a new code or data section”.
- The above is a statement given in the user manual of Keil Microvision.
- It implies that the assembler understands two types of information —code and data.
- Data is usually stored in R/W memory (RAM) while code is in Read only memory (ROM).
- The assembler requires the names of these areas of code or data.

- Typical statements that use the AREA directive are

AREA LINEAR, CODE, Read only

AREA NOTE, DATA, R/W

- It is not mandatory to use the words 'Read Only' and 'R/W' because by default, data is in R/W memory and code is in Read Only Memory.
- Code is always in ROM, though data is also allowed to be stored there.
- As a matter of fact, the R/W memory is volatile and is used only for intermediate storage of data in the course of instruction execution

5.3.1.2 ENTRY

- The ENTRY directive marks the first instruction to be executed within an application.
- Because an application cannot have more than one entry point, the ENTRY directive can appear in only one of the source modules.

5.3.1.3 END

- This directive is the last line in an assembly module and it indicates that the assembler need not read beyond that line.

5.3.1.4 DCB, DCW and DCD

- The ARM processor has a word length of 32 bits, but it can handle data of 8 bits (byte) and 16 bits (half word) also.
- The Keil assembler, however, uses the notations of byte, word and double word for 8 bits, 16 bits and 32 bits, respectively.
- The directives used are DCB for data byte, DCW for data word and DCD for data double word.
- The following lines show how these directives are used.
- These directives are used by the assembler to store data in ROM.

EXAMPLE

BAG DCB 0x32, 56, 0x7E

BAG DCW 0x2234, 0x ED37, 0x9067

BOG DCD 0x DE345678, 0x34009812

The first line says that three bytes are to be stored in contiguous locations with labels, RAG, RAG+1, and RAG+2. Recollect that each address always corresponds to one byte of data. The second line indicates that 16 bit data is being stored in locations BAG, BAG+2 and BAG+4. The third line states that 32 bit data is to be stored in addresses with labels BOG and BOG+4.

5.3.1.5 EQU

- The EQU directive is for equating a label to a constant.
- When the label is encountered by the assembler during the process of assembly, it is replaced by its actual value.
- Examples of the usage of this directive are
 - RAPID EQU 65
 - STRT_ADDR EQU 0x84000000

Write an ALP (assembly language program) to add the first 5 natural numbers. The sum must be in a register.

AREA NAT, CODE, READONLY

ENTRY

MOV R0,#0 ;R0=0

MOV R1,#0 ;R1=0

BACKK **ADD R0,R0,#1** ;Increment R0 by 1

ADD R1,R1,R0 ;Add the numbers. The sum is in R1

CMP R0,#5 ;check if R0=5

BNE BACKK ;repeat if R0 is not equal to 5

GO **B GO** ;continue branching to GO

END ;end of the assembly module

Write and test a program for adding the first ten odd numbers (starting from 1)

AREA ODD, CODE, READONLY
ENTRY

MOV R1,#1 ;R1=1

MOV R2,#9;R2=9, the counter

MOV R3,#1 ;R3=1

BACKK ADD R3,R3,#2 ;R3 has the odd numbers

ADD R1,R1,R3 ;R1 contains the final sum

**SUBS R2,R2,#1 ;R2 is the counter for 10
numbers**

BNE BACKK ;repeat the addition until R2=0

GO B GO
END

5.4 ACCESSING MEMORY

- We already know that ARM is a load-store architecture.
- This means that only the load and store instructions access memory directly.
- There are load/store instructions with single registers, as well as with multiple registers.

5.4.1 SINGLE REGISTER DATA TRANSFER

- The word 'load' indicates that data from memory is to be moved to a register. 'Store' means that data from a register is saved in memory
- The format for a load/store instruction is
 $LDR/STR \{<cond>\}<Rd>, <addressing\ mode>$
- The operation of load/ store has a register Rd in which the data is placed. The 'addressing mode' implies that an effective address is generated by address calculation.

Unit- 5

ARCHITECTURE OF ARM CORTEX-M

INTRODUCTION

- Earlier In Chapter 4 It Was Stated That arm7 Was Successfully Created
- Later ARM 7 Was Followed By ARM 9 And ARM 11
- Means More Features Were Added To It, But The Basic Architecture Was Similar To ARM7
- The Latest Series Of These Processors Are Designated As Cortex
- They Have Slightly Different Architecture As Compared

ARM PROCESSOR FAMILIES



- High Performance Processors Capable Of Full Operating System (OS) Support;
- Applications Include Smartphones, Digital TV, Smart Books, Home Gateways Etc.



Cortex- R Series (Real-time)

- High Performance For Realtime Applications;
- High Reliability
- Applications Include Automotive Braking System, Powertrains Etc.

➤ Cortex-M Series (Microcontroller)

➤ Cortex-M3, M4, M7, M8, M33

-- Applications Include Microcontrollers, Mixed Signal Devices, Smart Sensors, Automotive Body Electronics And Airbags; More Recently IoT

➤ Securcore Series

-- High Security Applications

➤ Previous Classic Processors: Include ARM7, ARM9, ARM11 Families

CORTEX – M PROCESSOR



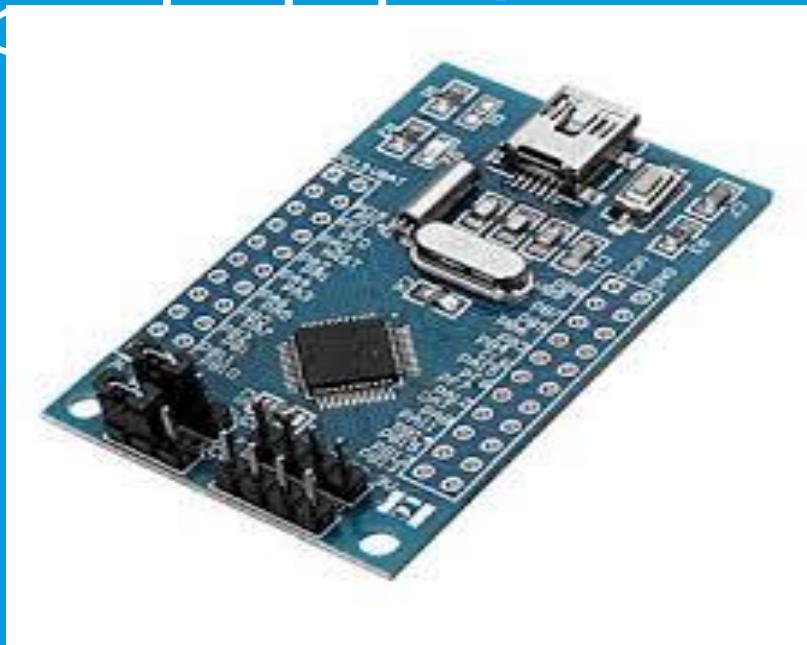
- The Cortex-m Family Has Been Designed For Embedded Applications With The Use Of Microcontrollers.
- Cortex-m Family Is A Low End Of The Cortex Family
- But It Posses A Very Good Computational Capabilities.
- The M Series Has Many Variants Which Are M0, M0+, M2, M3, M4 And M7.
- Each Variant Has A Special Feature And Depending On The Application, The User Can Select A Particular Variant.

- one of the point in focus is low power capability and that feature is taken into consideration at every stage in processor design.
- Lets see the brief reference to each of the members of the cortex-m family.

Cortex-M PROCESSORS FAMILY

➤ **Cortex- M0:**

size, on account of its very low gate count. It is meant to be used in very low power embedded products.



➤ The Arm Cortex-M0 processor is one of the smallest Arm processors available

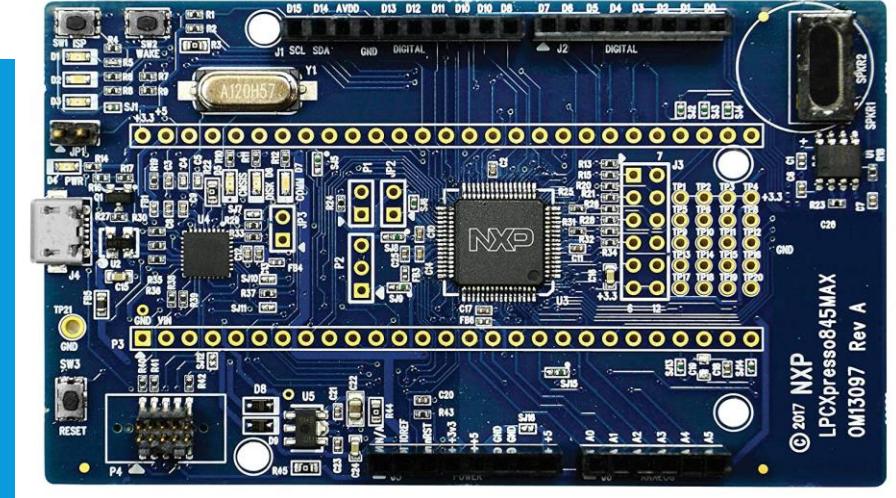


smaller die, low power and minimal code footprint, enabling developers to achieve 32-bit performance at an 8-bit price point, bypassing the step to 16-bit devices.

➤ The ultra-low gate count of the processor enables its deployment in analog and mixed signal devices.

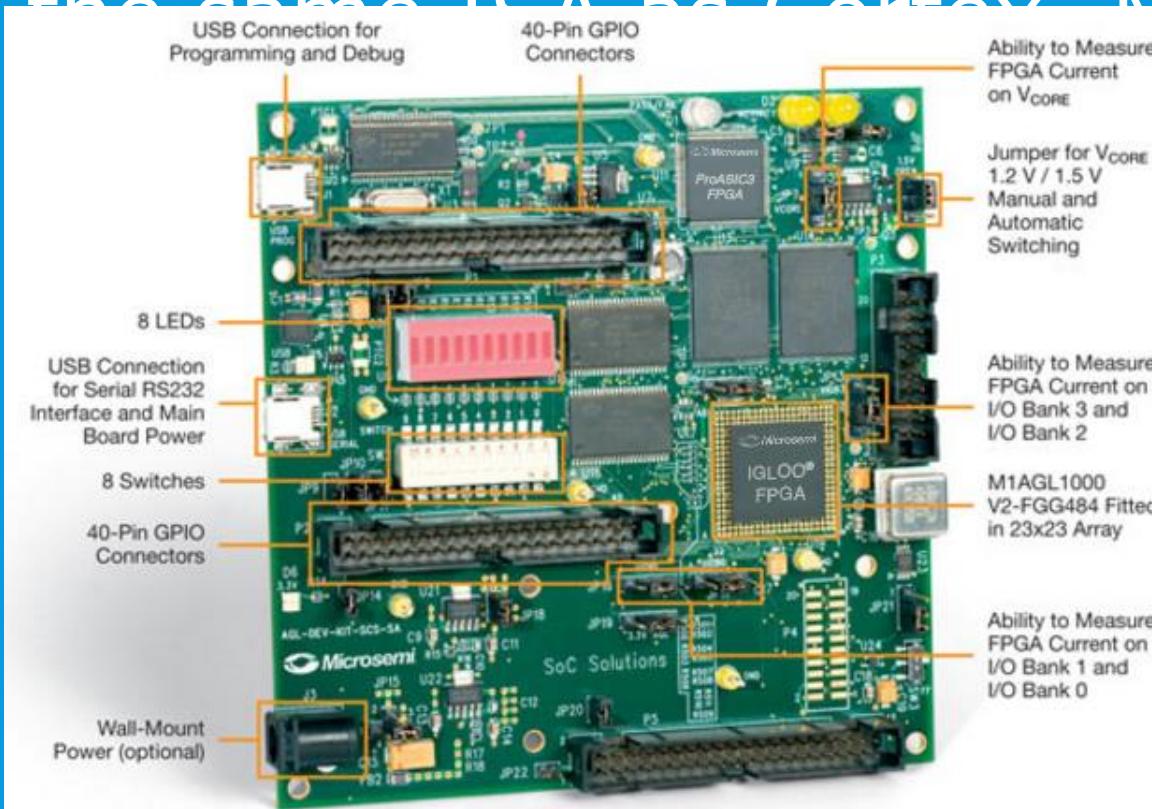
➤ Cortex- M0+:

- 32-bit, Low-Power Processor at an 8-bit Cost
- The Cortex-M0+ processor has the smallest footprint and lowest power requirements of all the Cortex-M



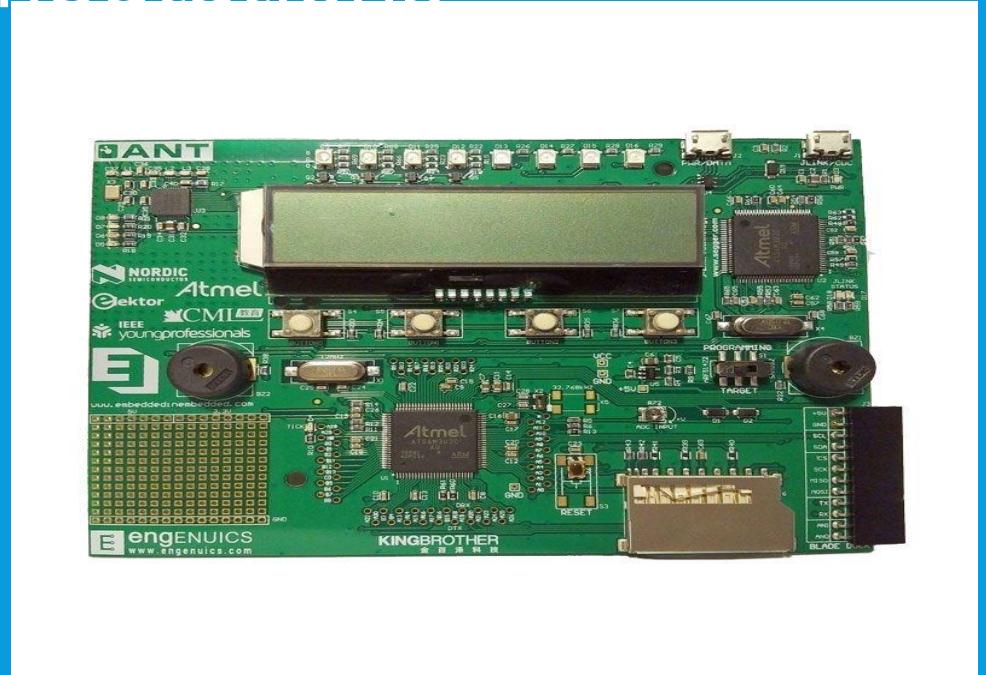
➤ Cortex- M1:

- Implementation utilizing the memory blocks on the FPGA
- It has the same ISA as Cortex-M0



➤ Cortex- M3:

- It has a hardware divider which is not normally available in ARM processors.
- It also has a Multiply and Accumulator unit, which helps in fast DSP computations

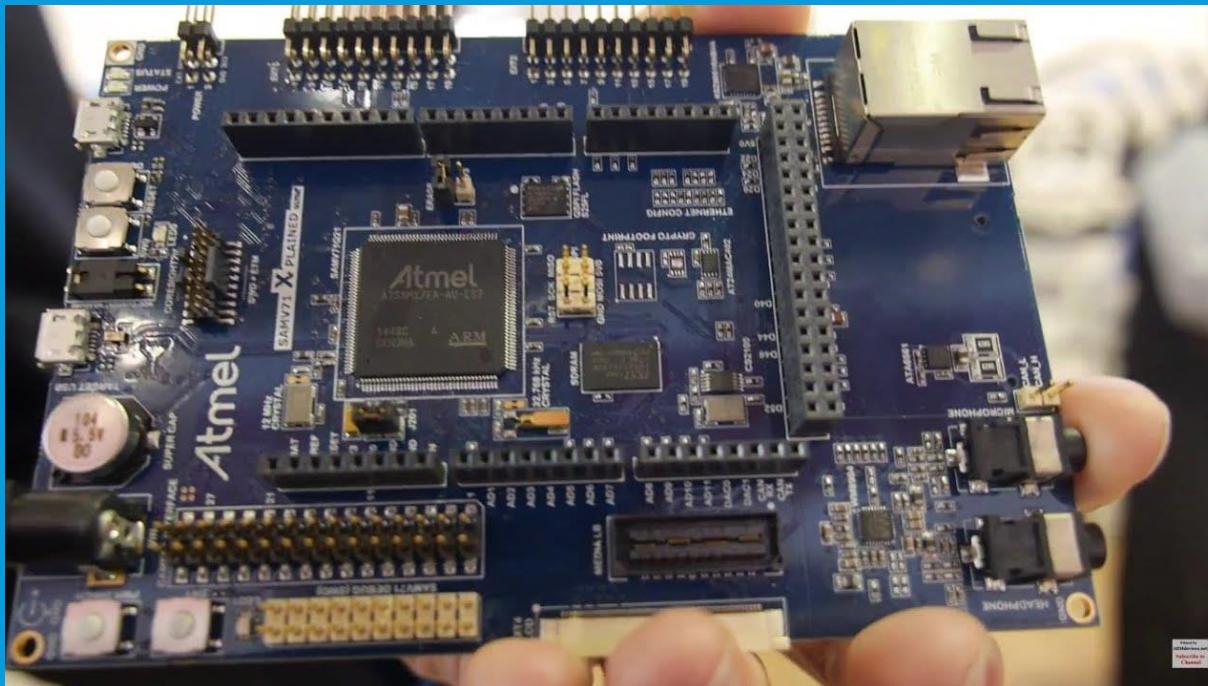


➤ Cortex- M4:



➤ Cortex- M7:

➤ It has advanced features like superscalar capability, longer pipeline, superior bus interfaces, rich DSP feature etc.



- Cortex M0, M0+ and M1 processors use the ARM v6-M ISA
- Cortex M3, M4 and M7 processors use the ARM v7-M ISA

CORTEX- MO

- ❑ Cortex-M0 was released in the year 2009.
- ❑ It was the smaller ARM processor with highly energy efficient
- ❑ It was good enough in performance for most embedded applications
- ❑ Based on its market review and performance, it was stated as 'Fastest ever licenced ARM processor'.

- ❑ It mainly focus on low power and small silicon area while keeping the performance high.
- ❑ For this 32-bit MCU, the power silicon area are almost the same as that of 8 or 16 bit MCU.
- ❑ Its performance is 2 to 10 times better than many 8 and 16 bit processors.

WAT ABOUT CORTEX-M0+?

- ❑ Cortex-M0+ was released three years later.
- ❑ Performance wise, the core is an upgraded version of Mo
- ❑ Uses the same software tools.
- ❑ But difference, Mo has 3-stage pipeline whereas Mo+ has only 2-stages.
- ❑ The smaller pipeline helps in more power savings.
- ❑ Cortex-M0+ is ideal for academic purposes, boards development and also quite inexpensive.

- The architectural features of all members of the Cortex-M series have many things in common
- So, need to understand thorough concept of Cortex-M0 is the first step.
- So we discuss here only on M0 architecture keeping in mind that the M0+ is almost same.
- This makes it easier to understand other members of series.

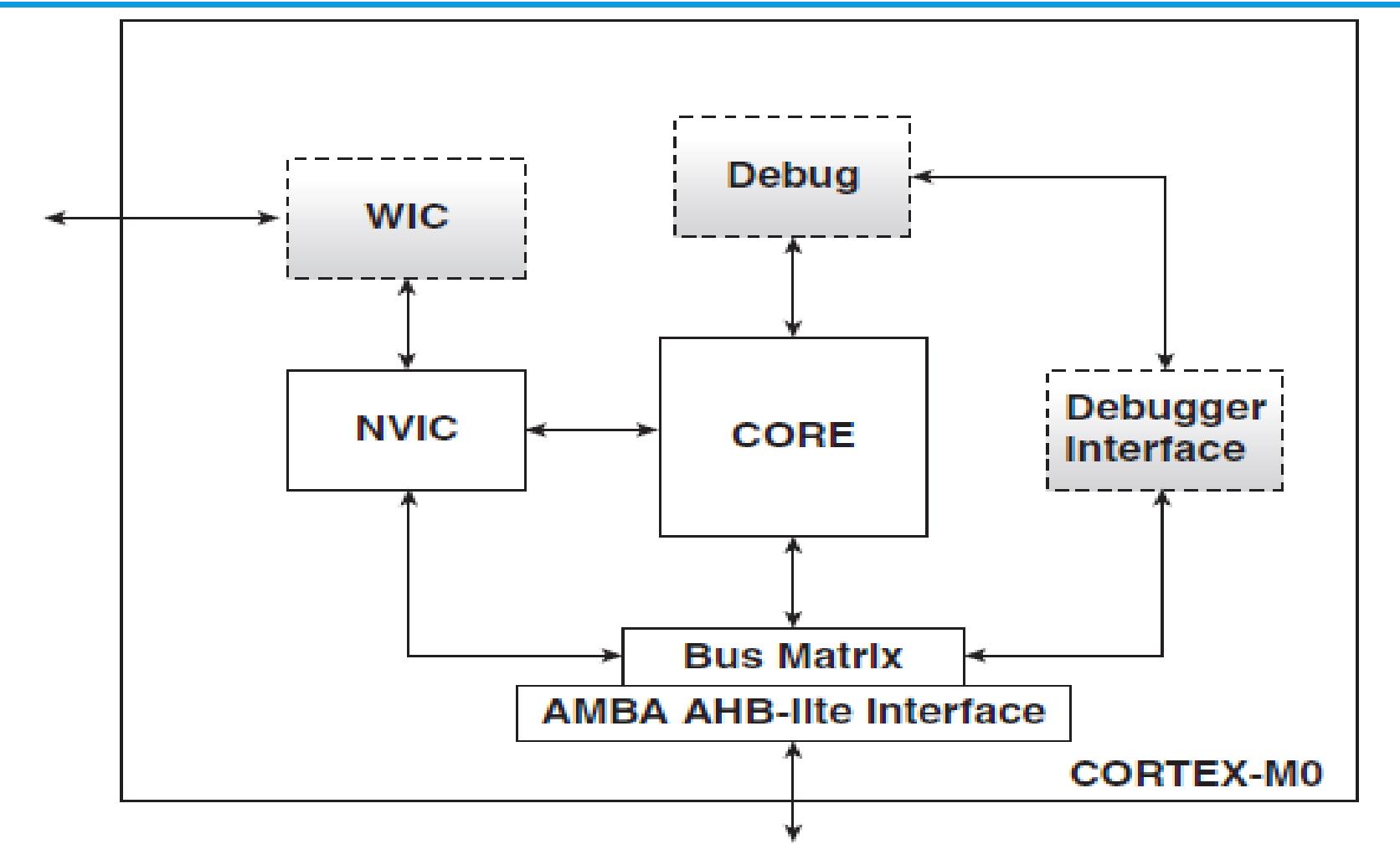
ADVANTAGES FOR A DESIGNER TO CHOOSE MO FOR HIS DESIGN

- Its architecture is simple and easy to learn and understand
- It has low gate count
- Its performance is quite good as far as comparison capability is concerned
- It operates at very low power levels and is energy efficient
- Its code density is high
- Its interrupt handling is deterministic
- It is upward compatible with the whole Cortex- M family
- It has an integrated Memory Protection Unit (Optional) and thus is capable of providing platform security.
- Its design is optimised for low power and low area

FEATURES OF CORTEX-M0

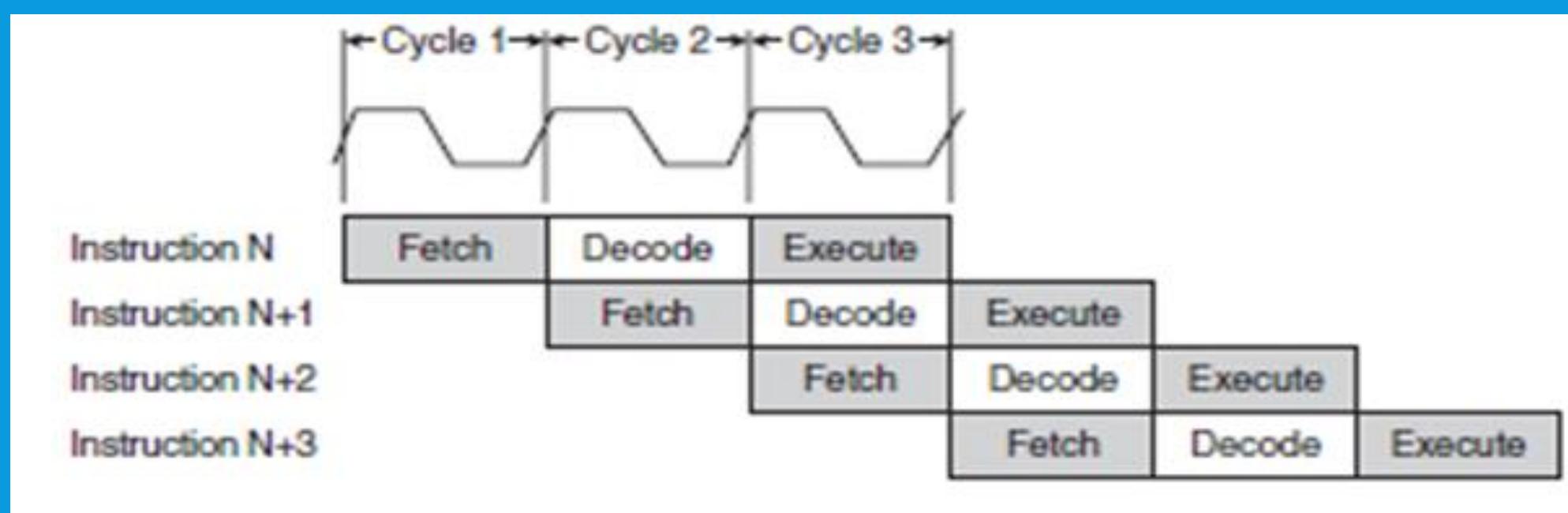
- ❑ It is a 32-Bit processor using ARM v6M ISA.
- ❑ Most of its instructions have single cycle execution capability
- ❑ It has only 56 base instructions, through some instructions have more than one form of usage
- ❑ It is a three-stage pipeline design
- ❑ It is a Von Neuman design

7.2.2 THE CORE



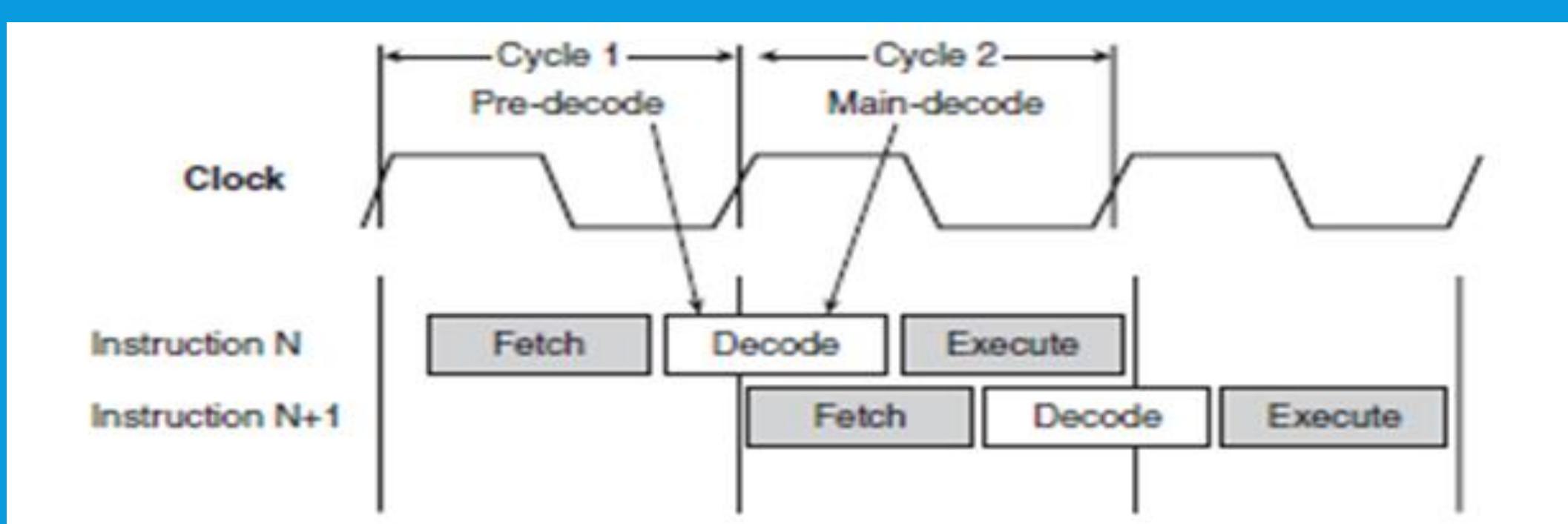
- ❑ Its Cortex-Mo processor
- ❑ Dotted components are optional
- ❑ Mandatory components are Nested Vectored Interrupt Controller (NVIC)
- ❑ The bus matrix that interfaces to the AMBA AHB-lite bus
- ❑ The processor core is the computing engine which contains the registers and ALU
- ❑ The instruction pipeline has 3 stages for Mo and 2 stages for Mo+

THREE-STAGE PIPELINE FOR CORTEX-M0



- ❑ The three stage pipeline of Mo is a regular fetch-decode-execute pipeline
- ❑ It takes 3 cycles to process an instruction

TWO-STAGE PIPELINE FOR CORTEX-M0+



- ❑ Mo+ has 2 stages
- ❑ fetch, decode & execute are completed in two cycles

7.2.3 NESTED VECTORED INTERRUPT CONTROLLER (NVIC)

- This is the hardware block that handles all the interrupts that comes to the processor.
- Mechanism is found, that handles multiple interrupts and accept one of them on priority resolution schemes
- Once the specified interrupt is acknowledged, the interrupting device can access its interrupt handler
- The NVIC has been designed for interrupt latency to be low and deterministic.

7.2.4 AMBA AHB-LITE

- ❑ AHB bus is a part of AMBA standard
- ❑ It is on-chip internal bus protocol for data transfers within the chip

7.2.5 DEBUG UNIT

- ❑ It has a breakpoint and watch point units
- ❑ User can halt execution at points he has chosen and do active debugging
- ❑ The debugger access port is the interface by which the processor is connected to the host system
- ❑ debugging may be done by the JTAG interface

7.2.6 WAKEUP INTERRUPT CONTROLLER

- ❑ It is optional unit which provides ultra low power sleep support
- ❑ As the processor is lowered down, most of its functional units including the NVIC are inactive
- ❑ In this state, if an interrupt arrives, it is handled by the wake up interrupt controller until the processor is brought back to active state.
- ❑ Thereafter, interrupt handling is done by NVIC

7.3 MODES AND STATES

- It has Various modes and States for operation
 - 1. Processor States
 - 2. Thread Mode
 - 3. Handler Mode

7.3.1 PROCESSOR STATES

- Processor has 2 states – simply implies the kind of activity it is involved in
- When it is executing a program, it is said to be in the **Thumb State**
- The other state is the **Debug state**

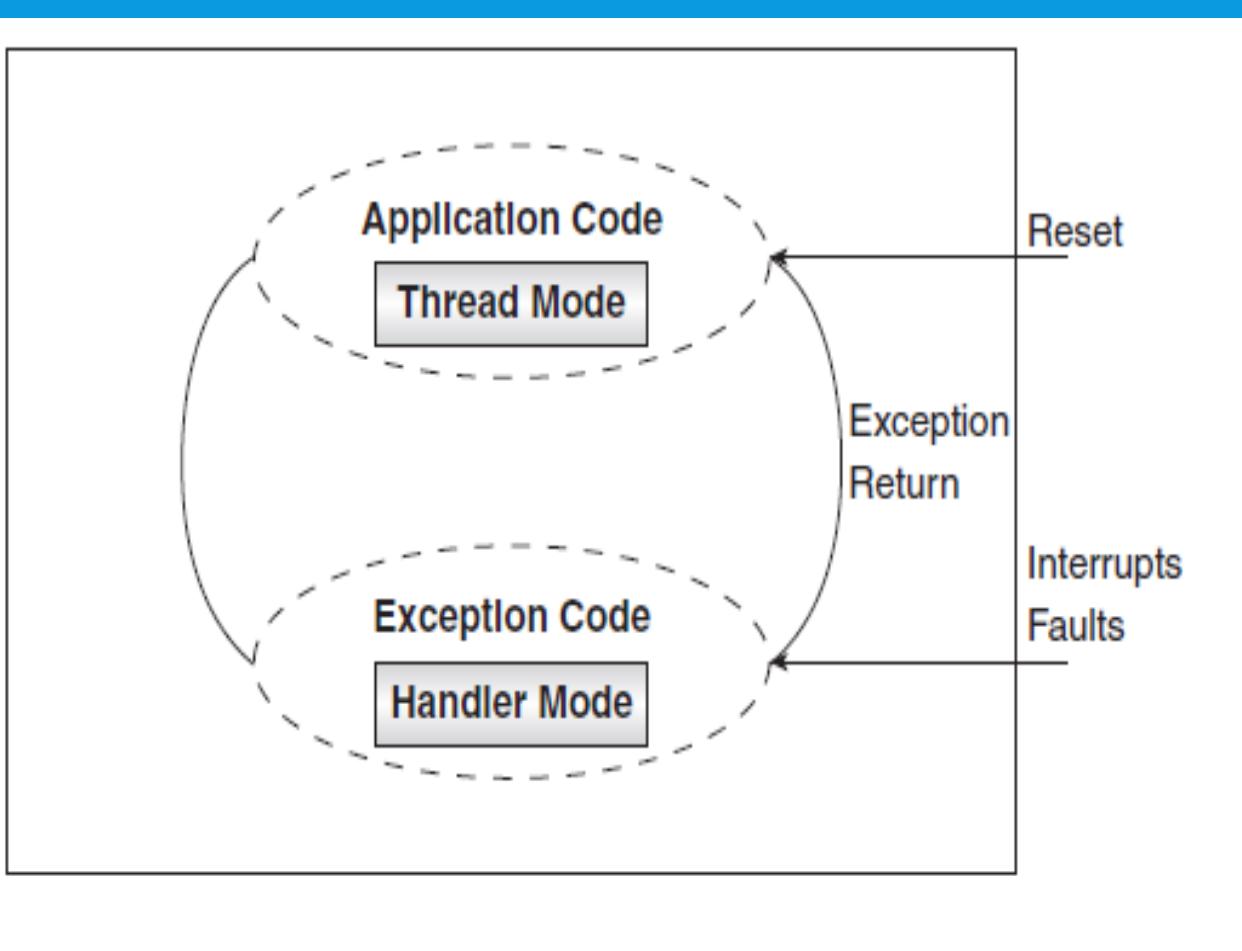
7.3.2 MODES OF OPERATION

When the processor is in the Thumb state, it can be either the Thread or the Handler mode.

7.3.2.1 THREAD MODE

- ❑ When the processor is powered on, it is in the Thread Mode
- ❑ This is the mode, as it is entered on reset and also on return from an interrupt
- ❑ There are two stacks defined for the processor:
 - 1. Main Stack**
 - 2. Process Stack**
- ❑ In the Thread mode, either of them may be used and a control register can be configured to decide whether the processor should use the main stack or the process stack.

7.3.2.2 HANDLER MODE



- This is the mode which is entered as a result of an interrupt
- figure shows the usage of these modes
- It is the **main stack**, that is used in this mode

Table 7.1 Difference between thread and handler modes

Processor mode	Use to execute	Stack used
Thread	Applications	Main Stack or Processor stack
Handler	Exception Handlers	Main Stack

If the processor has an OS running in it, then the thread mode will pertain to user tasks and the handler mode will be allocated for OS tasks as well as for interrupt handling

7.4 PROGRAMMING MODEL

- In an Assembly Language or a Complier design programming, the programmer sees the processor as,
- A set of Registers
- A set of Instructions

7.4.1 REGISTER SET

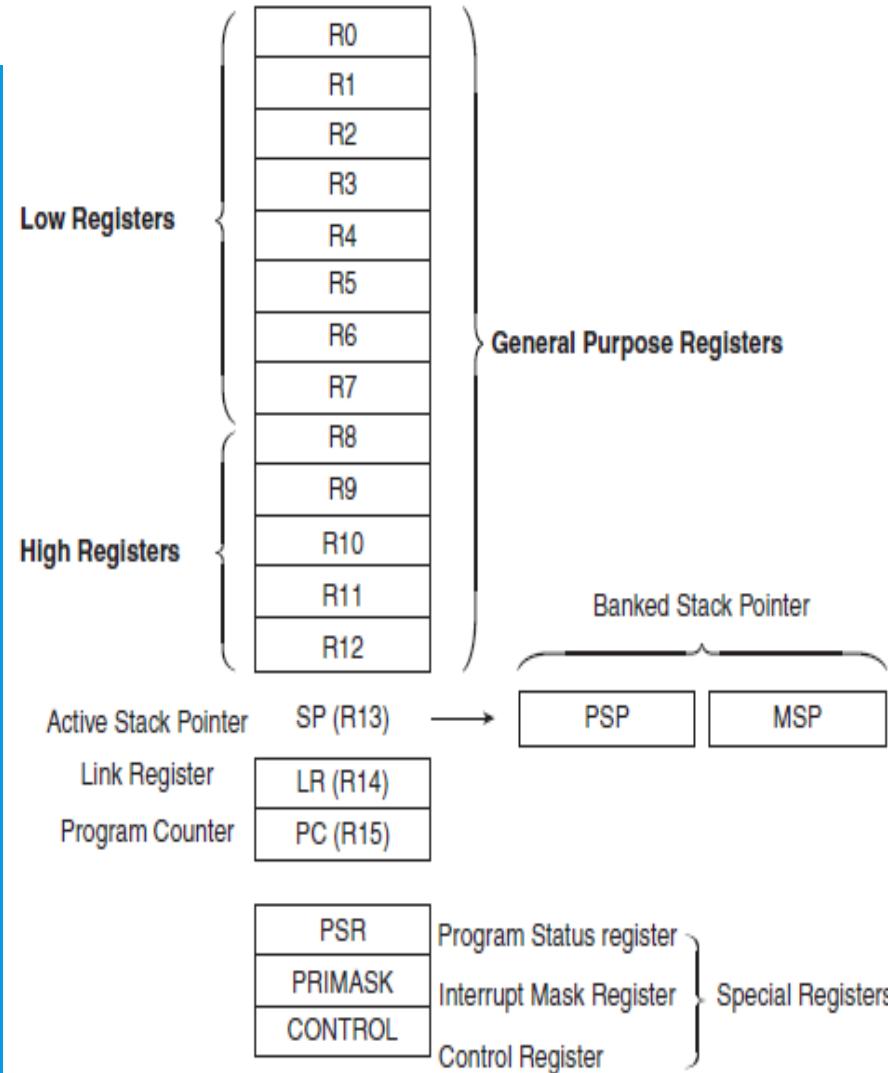


Figure 7.6 The register structure of Cortex-M0 (Reproduced with permission from ARM Limited. Copyright © ARM Limited)

- Figure shows the registers of the core
- General purpose registers are all 32 bit and are named R0 to R12
- They are used for temporary data storage
- For holding operands during computations
- Many instructions can use only registers R0 to R7 and are called low registers
- There are other instructions where all registers can use

7.4.2 SPECIAL REGISTERS

1. THE STACK POINTER (R13)

- ❖ Processor uses a descending stack
- ❖ Stack pointer which is also called R13 consists of 2 entities:
- ❖ MSP (Main Stack Pointer)
- ❖ PSP (Processor Stack Pointer)
- ❖ MSP is a default stack pointer, it is also the stack pointer used in handing interrupt handlers
- ❖ Few applications need only one stack pointer, so MSP will be used
- ❖ System with OS takes use of MSP for OS tasks and PSP for the use of applications

7.4.2 SPECIAL REGISTERS

2. THE LINK REGISTER (R14)

- ❖ As soon as function is called or an interrupt occurs, control branches to a new location
- ❖ Address to which to return to, is the current value of PC, is saved
- ❖ Register in which the return address is stored is the Link Register
- ❖ At the end of function, the contents of the Link Register can be copied back to PC
- ❖ Such a register increases speed because when functions are called, the return address is available in a processor register, rather to a stack.
- ❖ Advantage is because registers are within CPU, but memory is outside it and memory access is slower.

7.4.2 SPECIAL REGISTERS

3. THE PROGRAM COUNTER (R15)

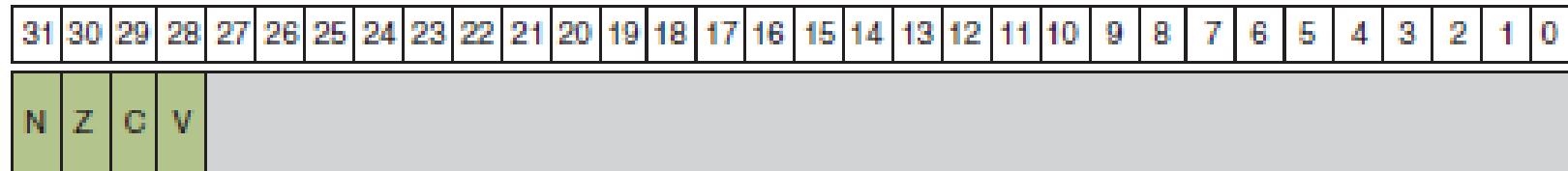
- ❖ This register sequences the flow of instruction execution
- ❖ It always contains the address of the next instruction to be executed
- ❖ It is a register which is a writable as well
- ❖ Means there are ways by which can cause branching by simply writing an address into the program counter

7.4.2 SPECIAL REGISTERS

4. THE PROGRAM STATUS REGISTER (PSR)

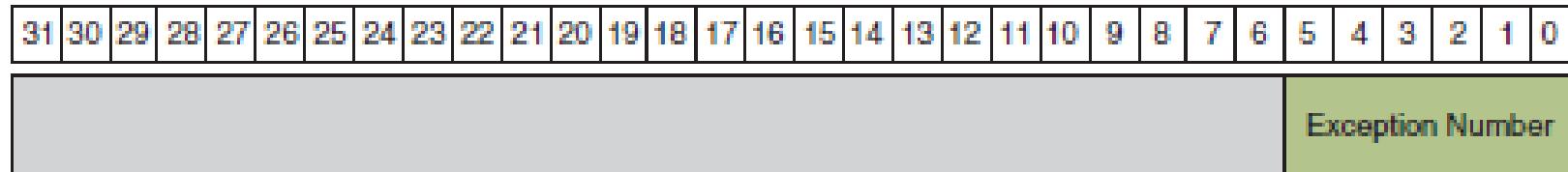
- ❖ This register holds various kinds of status information
- ❖ It is a 32-bit register which holds the status information in different bit fields, and combines three registers which are:
 1. Application Program Status Register (APSR)
 2. Interrupt Program Status Register (IPSR)
 3. Execution Program Status Register (EPSR)

- APSR – Application Program Status Register

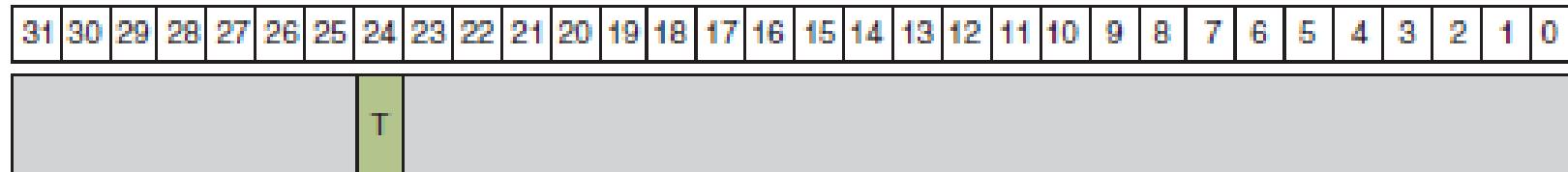


- Contains the Negative, Zero, Carry and Overflow flags from the ALU

- IPSR – Interrupt Program Status Register



- EPSR – Execution Program Status Register



- Thumb code is executed

- Composite register of APSR, IPSR and EPSR

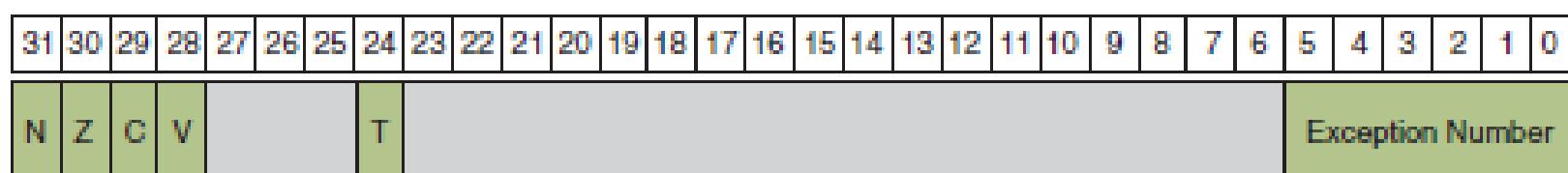


Figure 7.7 The three registers of the PSR

Figure 7.8 Combined PSR

- ❖ The status flag N, Z, C, V are in the upper four bits of APSR
- ❖ The lower six bits of the IPSR are for the exception/interrupt number
- ❖ bit filed 24 of the EPSR indicated the 'state' of the processor (Either in ARM or Thumb)
- ❖ Since the processor operates only in the Thumb mode, this bit is always 1
- ❖ clearing this bit causes an exception

Table 7.2 Three other combinations of status registers

Register	Type	Combination
IEPSR	Read Only	EPSR and IPSR
IAPSR	Read/Write	APSR and IPSR
EAPSR	Read/Write	APSR and EPSR

Above table indicates that there are three other combinations of these status registers, that is, with two of them taken together.

7.4.3 OTHER SPECIAL REGISTERS

PRI Masks

- ❖ Certain interrupts which are 'non-maskable' and others whose priorities are configurable
- ❖ This register is related to the 'masking' of interrupts, it is used to prevent the activation of all configurable interrupts
- ❖ This register has 32 bits, out of which only the LSB is used
- ❖ Setting the LSB to '1' makes inactive
- ❖ all Interrupts with configurable priority

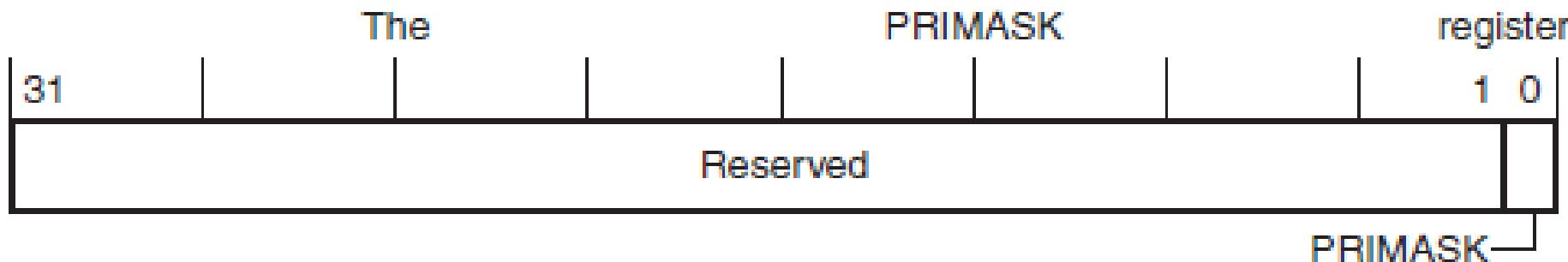


Table 2–7 PRIMASK register bit assignments

Bits	Name	Function
[31:1]	-	Reserved
[0]	PRIMASK	0 = no effect 1 = prevents the activation of all exceptions with configurable priority.

Figure 7.9 PRI MASK register and bit assignments (*Reproduced with permission from ARM Limited. Copyright © ARM Limited*)

Control Register

- ❖ This is another 32-bit register of which only one bit is meaningful
- ❖ Bit 1 selects the stack to be used when the processor is in the 'Thread' mode
- ❖ If the Bit=0, the main stack is specified to be active, otherwise the PSP is considered to be active

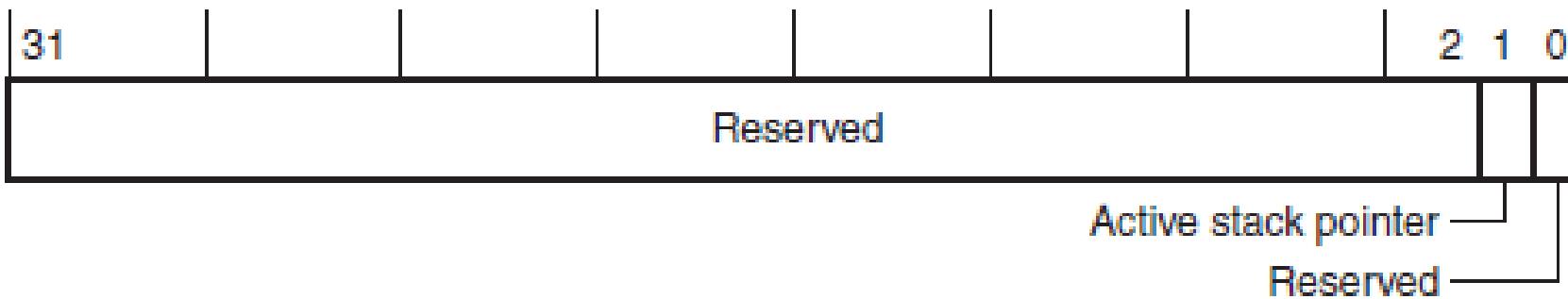


Table 2–8 CONTROL register bit assignments

Bits	Name	Function
[31:2]	-	Reserved
[1]	Active stack pointer	Defines the current stack: 0 = MSP is the current stack pointer 1 = PSP is the current stack pointer. In Handler mode this bit reads as zero and ignores writes.
[0]	-	Reserved.

Figure 7.10 The Control Register and its bit assignments (Reproduced with permission from ARM Limited. Copyright © ARM Limited)

7.5 MEMORY MODEL

- ❖ As address bus is of 32 bits, the addressing range of the processor is $2^{32} = 4 \text{ GB}$
- ❖ Processor uses memory mapped I/O
- ❖ 4 GB space is partitioned into space memory as well as for peripherals

Core Memory Mapped Register Space,
i.e. NVIC
(XN)

External Peripherals
(XN)

External Memory

External Peripherals
(XN)

Data Memory (code can also be placed here)

Executable region for program code and data
(vector table is fixed at address 0x0000 0000)

Device	511MB	0xE010 0000
Private Peripheral Bus		0xE000 0000
External Device	1GB	
External RAM	1GB	0xA000 0000
Peripheral	500MB	0x6000 0000
SRAM	500MB	0x4000 0000
Code	500MB	0x2000 0000
		0x0000 0000

XN – execute never

Figure 7.11 The memory map of Cortex-M0

MEMORY MODEL OF COTEX-M0

1. Code: This part of memory is of 512 MB

It is the lowest end of the map and is meant to store program code, even though data is allowed

This part is realised using ROM, usually FLASH ROM

Interrupt vector table is stored here.

MEMORY MODEL OF COTEX-M0

2. SRAM: This part of memory is of 512 MB

Can store data, though program code is allowed here

This is part where Stack and Heap memories are initialized to be present

Many designs may realise this memory part using SDRAM

In programs, this section is designated as Read/Write memory

MEMORY MODEL OF COTEX-M0

3. Peripheral: Memory addresses in this region are allocated to peripheral address.

Though part of it can be realised by memory device to store data.

It can be used only for 'non-executable' information
This attribute is referred to as 'EXECUTE NEVER' (XN)

MEMORY MODEL OF COTEX-M0

4. External RAM: This is 1 Gb of space which can be mapped to external RAM

It can store data as well as executable program code and can be realised by different kinds of semiconductor memories

MEMORY MODEL OF COTEX-M0

5. Device Memory: This is 1 Gb of space

It is a non-executable region and is used for device addresses

MEMORY MODEL OF COTEX-M0

6. Private Peripheral Bus: This is a non-executable region used for addresses of the registers of the NVIC, System Tier, and System Control Block

Only word accesses can be used in this region

MEMORY MODEL OF COTEX-M0

7. Device Memory: This is the uppermost 512 MB constitute another non-executable (XN) device memory.

Chip designers are allowed to use it in the way they want, or just leave it as 'reserved'

7.5.1 TYPES AND ATTRIBUTES OF MEMORY

- ❖ Have seen the different sections of the memory space allocated for different applications
- ❖ Access to different areas of memory may be 'Shareable' if there is more than one processor/bus master in the system
- ❖ The order in which access is allowed to occur for region of memory, is defined by certain memory attributes
- ❖ they are:

Normal, Device and Strongly-ordered

1. NORMAL

- ❖ Memory used for program execution and data storage generally occurs within the 'Normal' designation.
- ❖ In such kind, the processor has the freedom to do 'Out of Order' accessing and even speculative reads
- ❖ Means that memory access need not be in the order in which a program is coded
- ❖ Examples: Preprogrammed flash
ROM, SDRAM, SRAM and DDR memory

2. DEVICE

- ❖ In the case of I/O's rules of access is stricter
- ❖ Out of order is not permitted

3. STRONGLY ORDERED

- ❖ This attribute is required where it is necessary to ensure strict ordering of the access relative to what occurred in program order before the access and after it
- ❖ This means that the processor preserves transaction order relative to all other transactions

Strongly-ordered memory always assumes the resource to be shareable

7.5.2 OUT-OF-ORDER ACCESS

- ❖ As the program is coded, compiled and made ready to run, we assume that execution is in the order of the program lines.
- ❖ Program may change the order of execution
- ❖ It may also give higher efficiency and speed
- ❖ Similarly, the order in which the memory is accessed may also be changed.

Reasons behind it is listed as:

1. The processor can reorder same memory
2. Memory or devices in the memory map may have different wait states
3. Some memory accesses are buffered or speculative