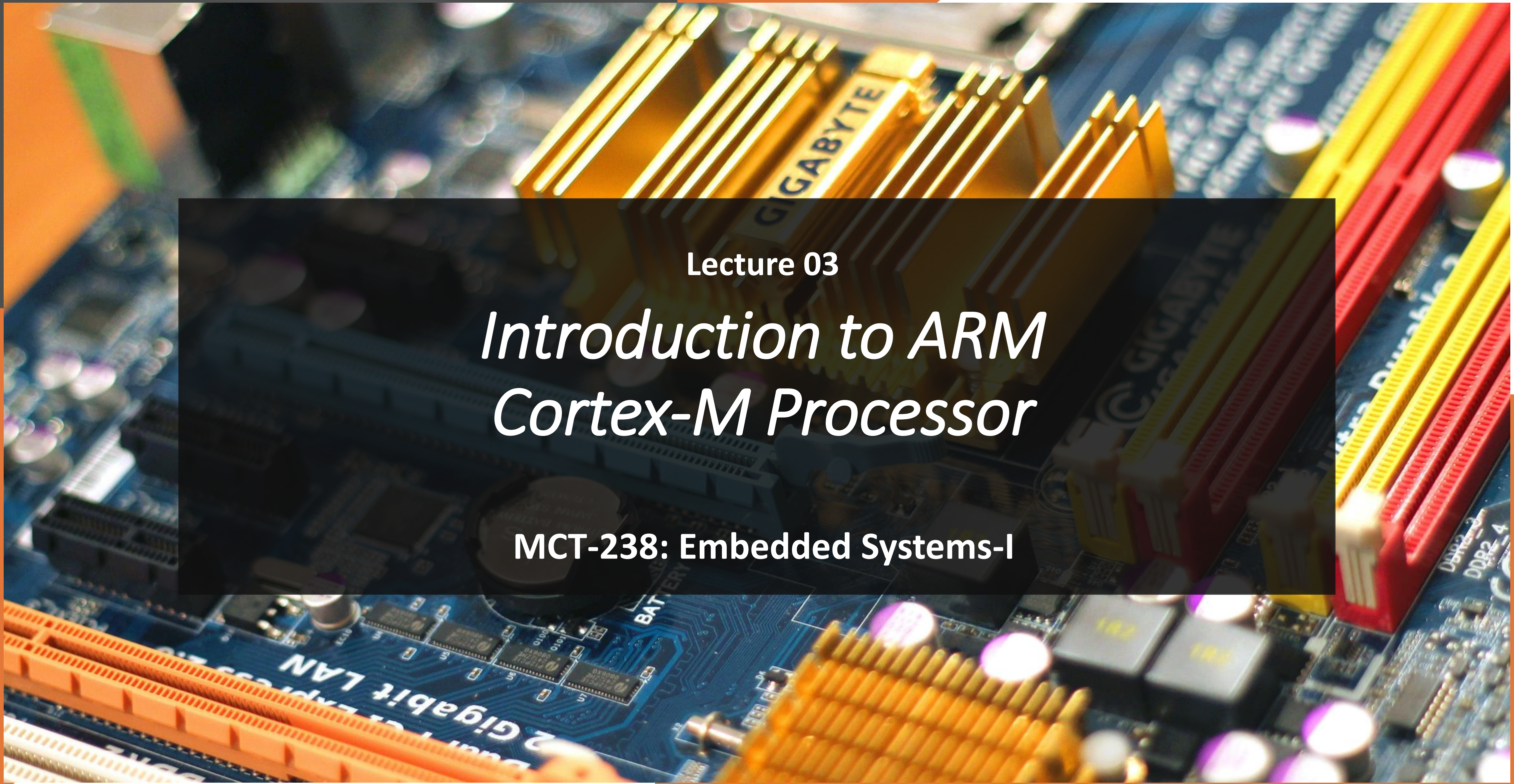
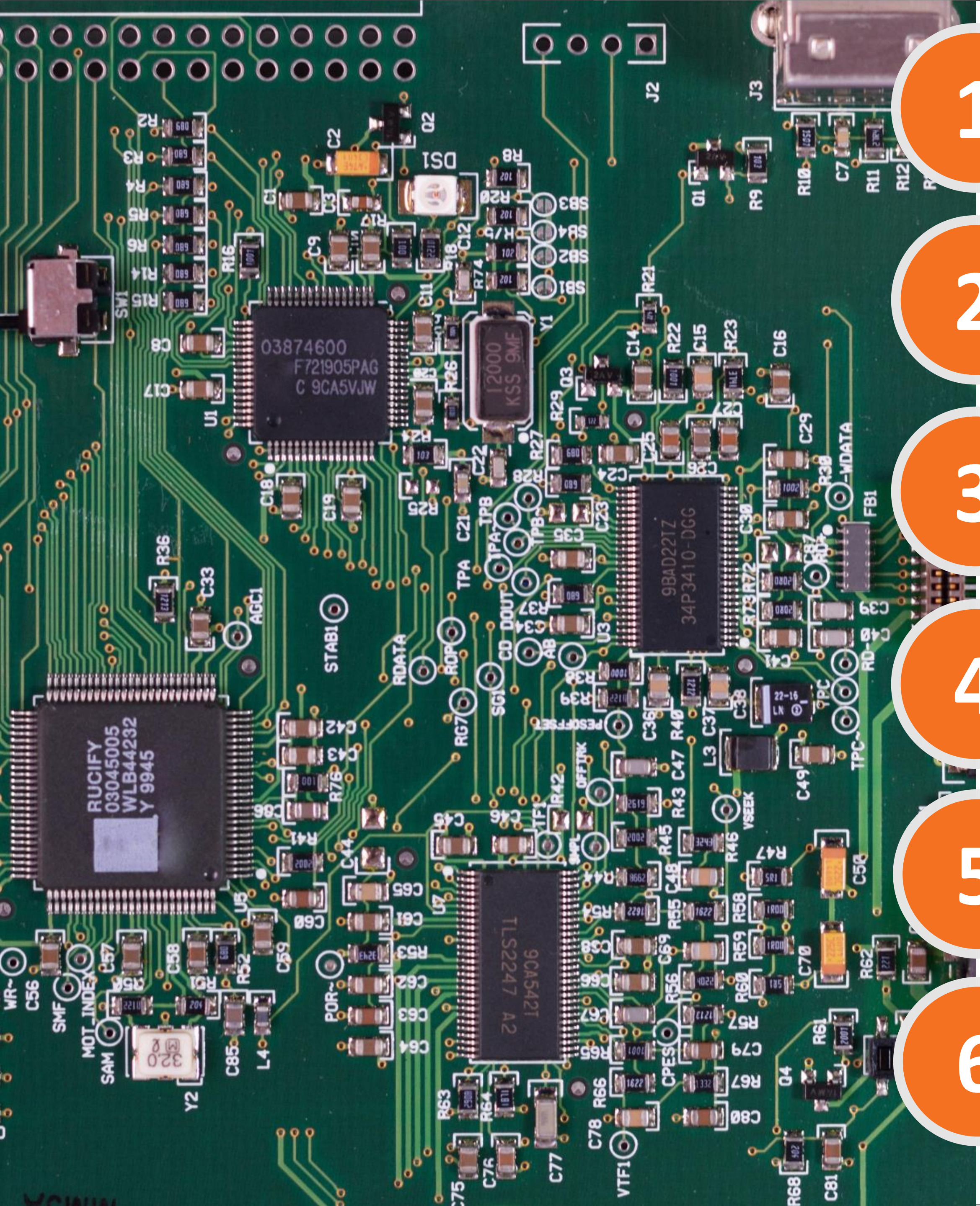


Lecture 03

Introduction to ARM Cortex-M Processor

MCT-238: Embedded Systems-I



1 ARM CORTEX-M ARCHITECTURE

2 REGISTERS ON ARM CORTEX-M PROCESSOR

3 PROCESSOR OPERATING MODES

4 ARM CORTEX-M MEMORY

5 INTERRUPTS AND PROCESSOR RESET SEQUENCE

6 PIPELINED ARCHITECTURE

ARM CORTEX-M ARCHITECTURE

Introduction to ARM Cortex-M Processor

Introduction to ARM Series

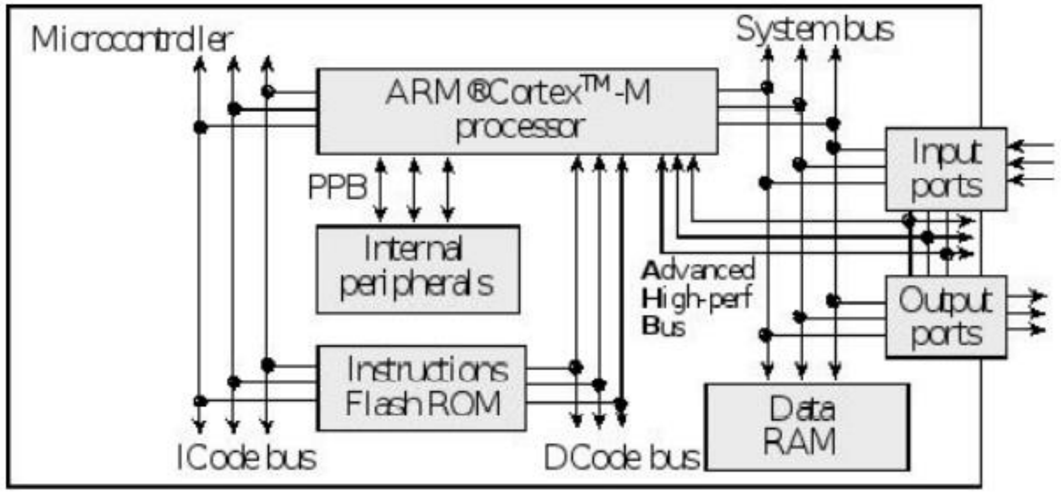
ARM CORTEX-M ARCHITECTURE

- ARM (**A**dvanced **R**ISC **M**achine) Limited is the company that designs 32-bit and 64-bit the core CPU components and licenses the intellectual property to partner organizations, which then build ARM-based chips according to their own requirements
- ARM processors uses a much simpler instruction set approach to maximize performance as compared to Intel which uses hardware approach for it
- Because of RISC architecture, ARM processors require fewer transistors, resulting in a smaller die size for the integrated circuitry which reduces hardware complexity and lower power consumption
- Processors based on ARMv7 is architecture have been divided into the following three profiles
    - **Cortex-M** profile: Processors for microcontroller based embedded systems
    - **Cortex-R** profile: Processors for real-time applications is the main motive of this profile
    - **Cortex-A** profile: Processors for high performance applications mainly covering the cellular market

Introduction to ARM Cortex-M Processor

ARM CORTEX-M ARCHITECTURE

- **Harvard Architecture** – separate data & instruction buses
- **Instruction Set** combines high performance 32-bit and high code density 16-bit & 8-bit instructions
- Instructions are fetched from flash ROM using the **ICode Bus**
- Data are exchanged with memory and I/O via the **System Bus** interface
- **Advanced High Performance Bus** for high-speed devices like USB
- There are many sophisticated debugging features utilizing the **DCode bus**.



Introduction to ARM Cortex-M Processor

ARM CORTEX-M ARCHITECTURE

Nested Vectored Interrupt Controller (NVIC)

- NVIC manages interrupts, which are hardware-triggered software functions
- Some internal peripherals, like the NVIC communicate directly with the processor via the **Private Peripheral Bus** (PPB)
- The tight integration of the processor and interrupt controller provides fast execution of interrupt service routines (ISRs), dramatically reducing the interrupt latency

Byte Addressable Memory & I/O

- Even though data and instructions are fetched 32-bits at a time, each 8-bit byte has a unique address. This means memory and I/O ports are byte addressable
- The processor can read or write 8-bit, 16- bit, or 32-bit data
- Exactly how many bits are affected depends on the instruction, which we will see later

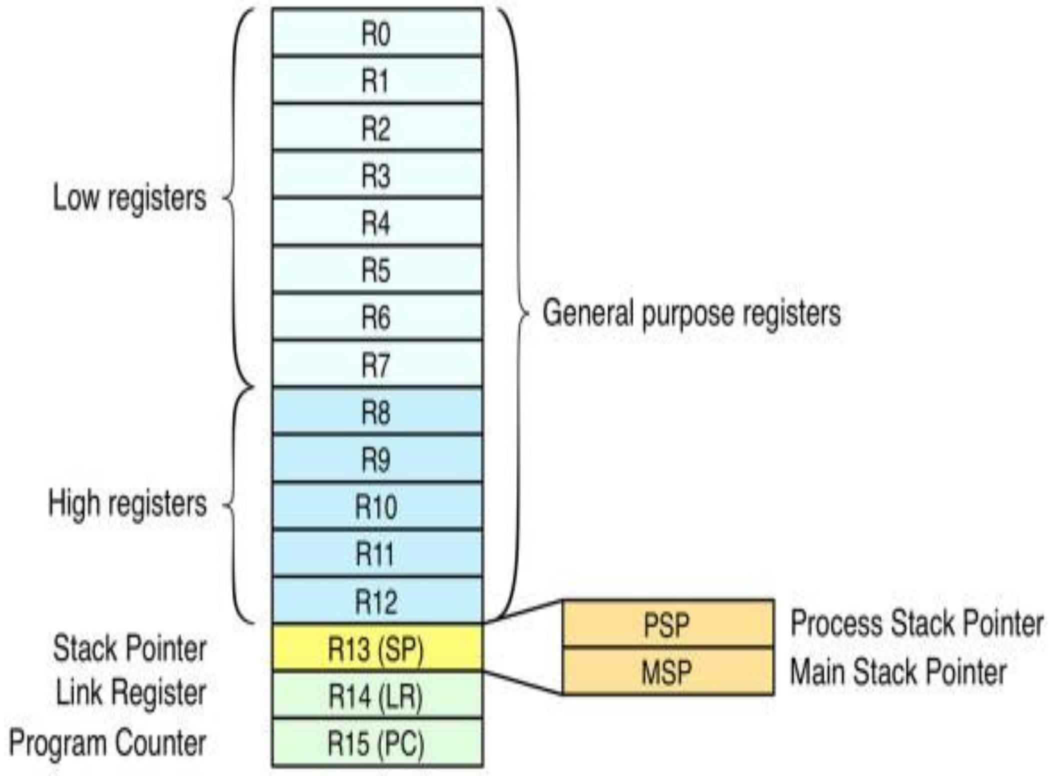
REGISTERS ON ARM CORTEX-M PROCESSOR

General Purpose Registers, Stack Pointer, Link Register, Program Counter, AAPCS, Program Status Registers, Priority/Fault Mask Register, Base Priority Mask Register

General Purpose Registers

REGISTERS ON ARM CORTEX-M PROCESSOR

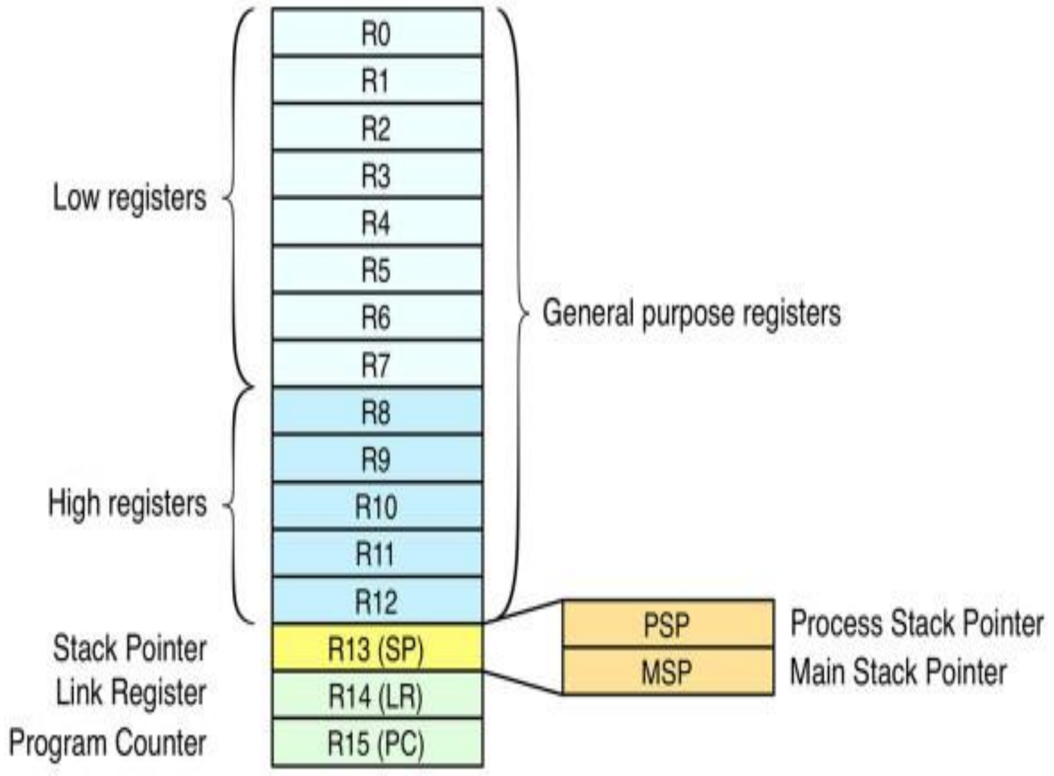
- **Registers** are high-speed storage inside the processor
- The ARM Cortex-M4 architecture provides 16 read/write, 32-bit registers, numbered from **R0** to **R15**
- **R0** to **R12** are general purpose registers and contain either data or addresses
- **R13** to **R15** have special hardware significance
- Registers **R0-R7** (Low Registers) are accessible by all instructions that specify a general-purpose register
- While registers **R8-R12** (High Registers) are accessible by only 32-bit instructions that specify a general-purpose register



Stack Pointer, SP

REGISTERS ON ARM CORTEX-M PROCESSOR

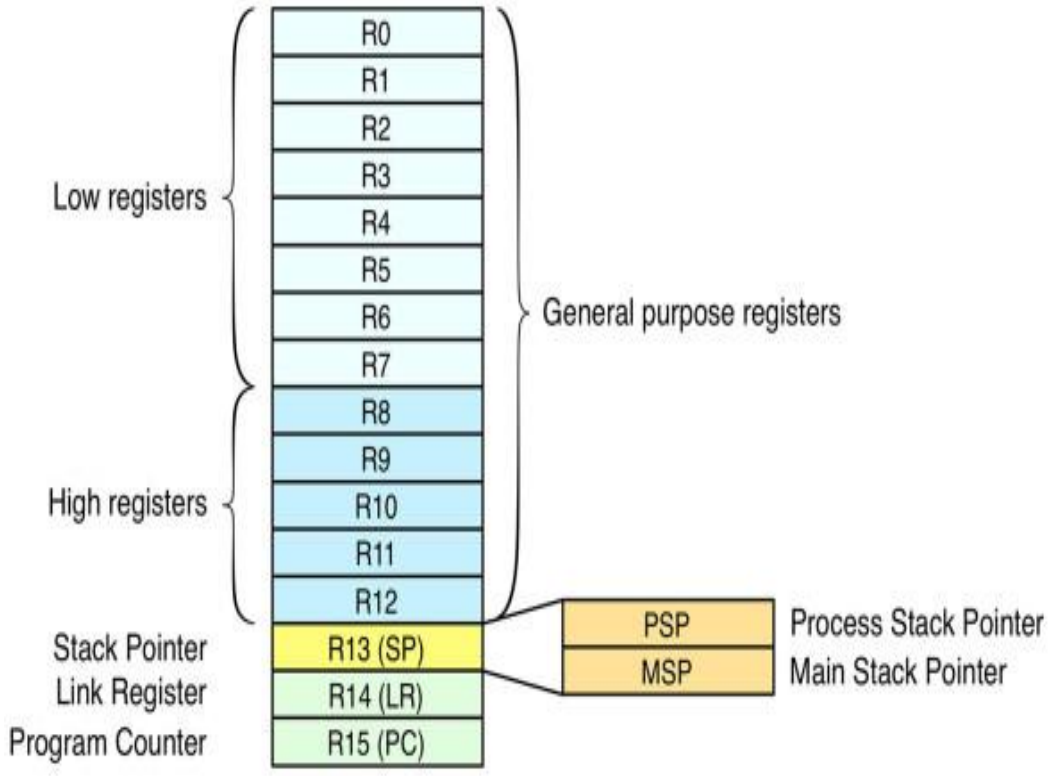
- Register **R13** (also called the **stack pointer**, **SP**) points to the top element of the stack
- It is used automatically in the **PUSH** and **POP** instructions to manage storage and recovery of registers in the stack
- CPU uses this register, therefor we can’t control it directly
- Because the **SP** ignores any writes to bits [1:0], this makes it aligned to a word (4 byte) boundary automatically
- **SP** is a banked register with two copies, namely Main Stack Pointer (MSP) and Process Stack Pointer (PSP).
- Only one copy of the stack pointer (R13) is visible and active at a given time
- Handler mode always uses MSP, but you can configure Thread mode to use either MSP or PSP



Link Register, LR

REGISTERS ON ARM CORTEX-M PROCESSOR

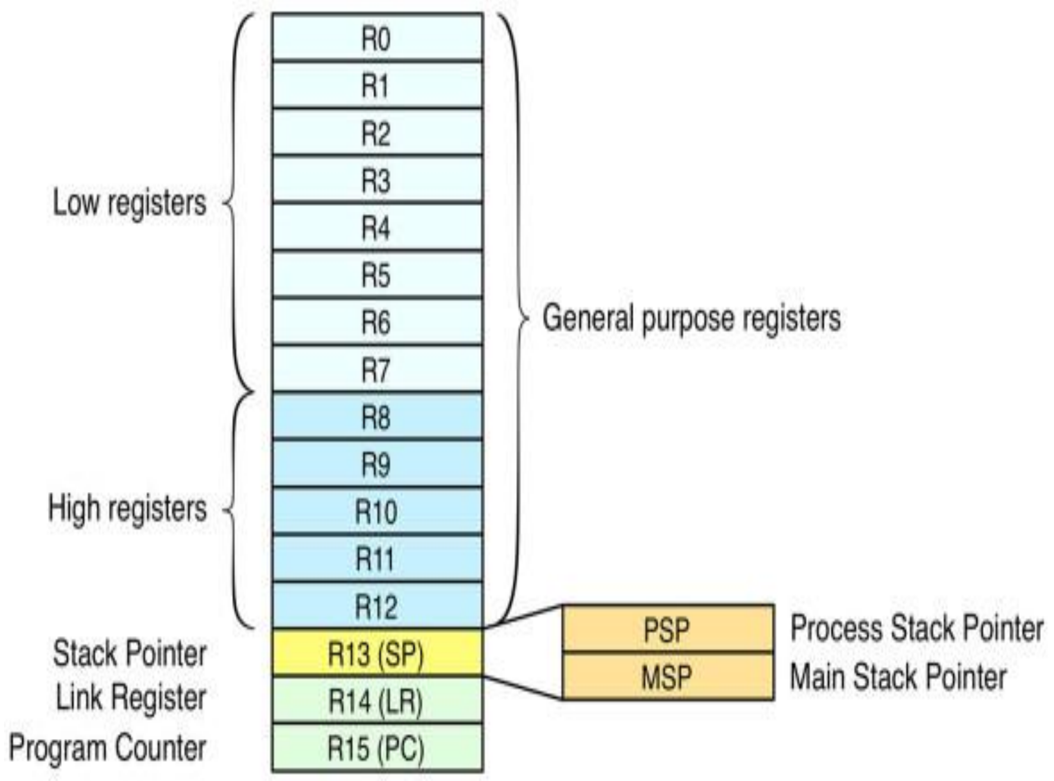
- Register **R14** (also called the **link register**, **LR**) is used to store the return location for functions or subroutine from **PC** when a Branch and Link (**BL**) or Branch and Link with Exchange (**BLX**) instruction is executed
- The **LR** is also used in a special way during exceptions, such as interrupts
- Return address means the first instruction to be executed after returning from the current function, subroutine, and exceptions
- When the **LR** is not used for holding a return address, it can be treated as a general-purpose register



Program Counter, PC

REGISTERS ON ARM CORTEX-M PROCESSOR

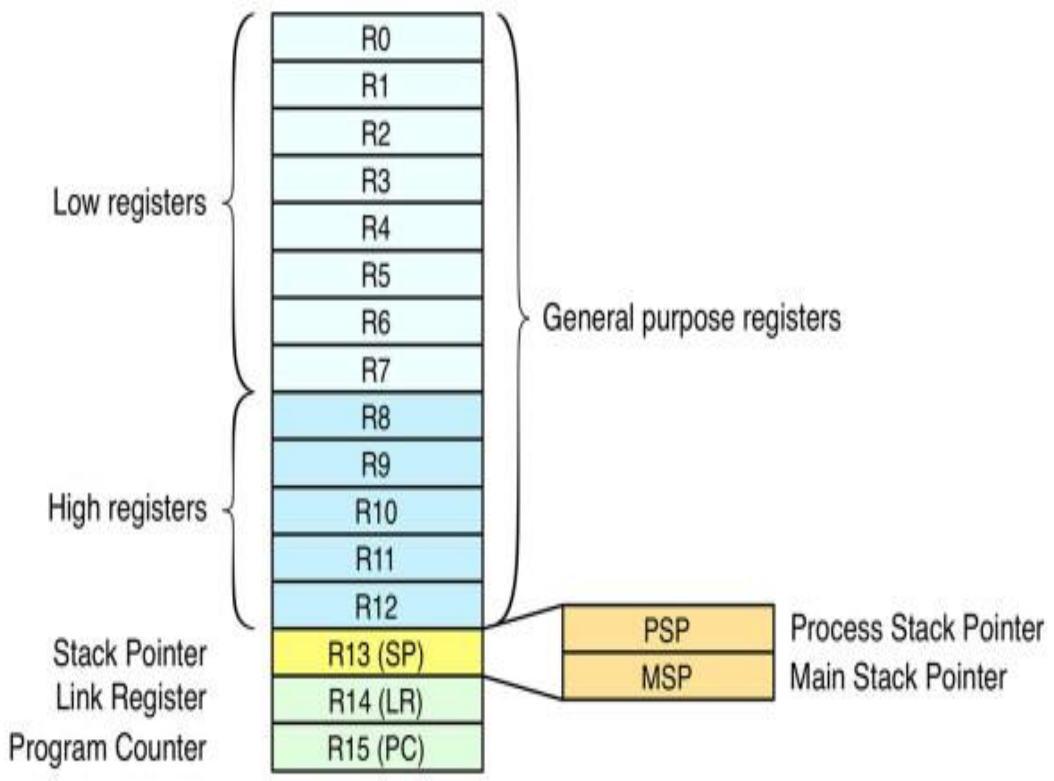
- Register **R15** (also called the **program counter**, **PC**) points to the next instruction to be fetched from memory.
- The processor fetches an instruction using the **PC** and then increments the **PC**
- On reset, the processor loads the **PC** with the value of the reset vector, which is at address 0x00000004
- Bit 0 of this register is always 0, which ensures that the instructions are always aligned to either word or halfword boundaries in the code memory
- The **PC** register can be accessed in either privileged or unprivileged mode



ARM Architecture Procedure Call Standard, AAPCS

REGISTERS ON ARM CORTEX-M PROCESSOR

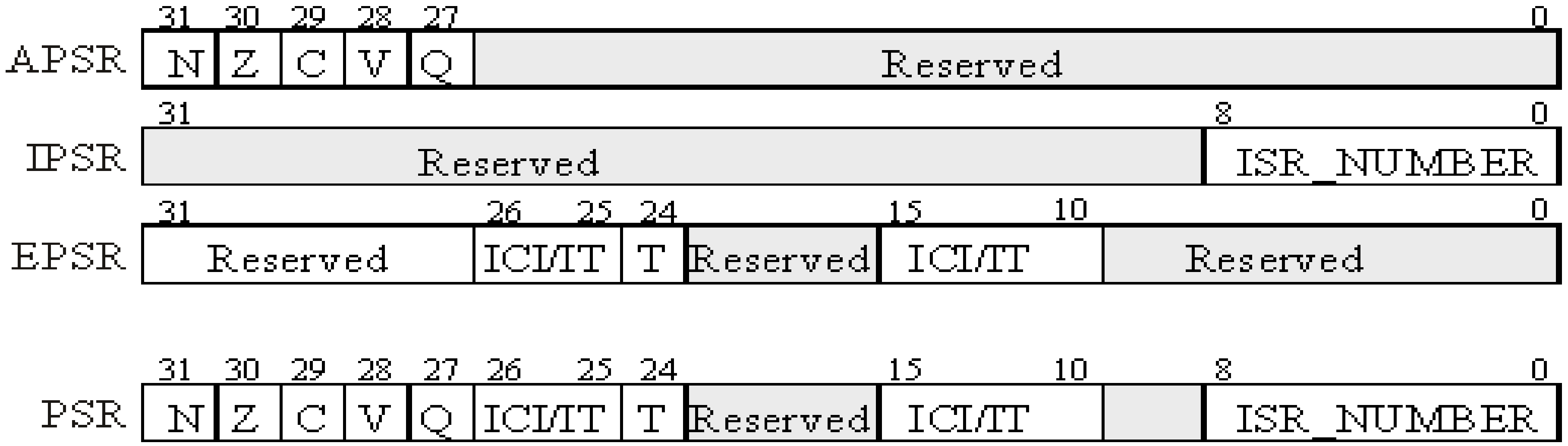
- **AAPCS** is a part of the ARM **Application Binary Interface** (**ABI**)
- It uses registers **R0**, **R1**, **R2**, and **R3** to pass input parameters into a C function
- Functions must preserve the values of registers **R4–R11**
- Also according to **AAPCS** we place the return parameter in Register **R0**
- **AAPCS** requires we push and pop an even number of registers to maintain an 8-byte alignment on the stack
- In this class, the **SP** will always be the main stack pointer (**MSP**), not the Process Stack Pointer (**PSP**)



Program Status Registers

REGISTERS ON ARM CORTEX-M PROCESSOR

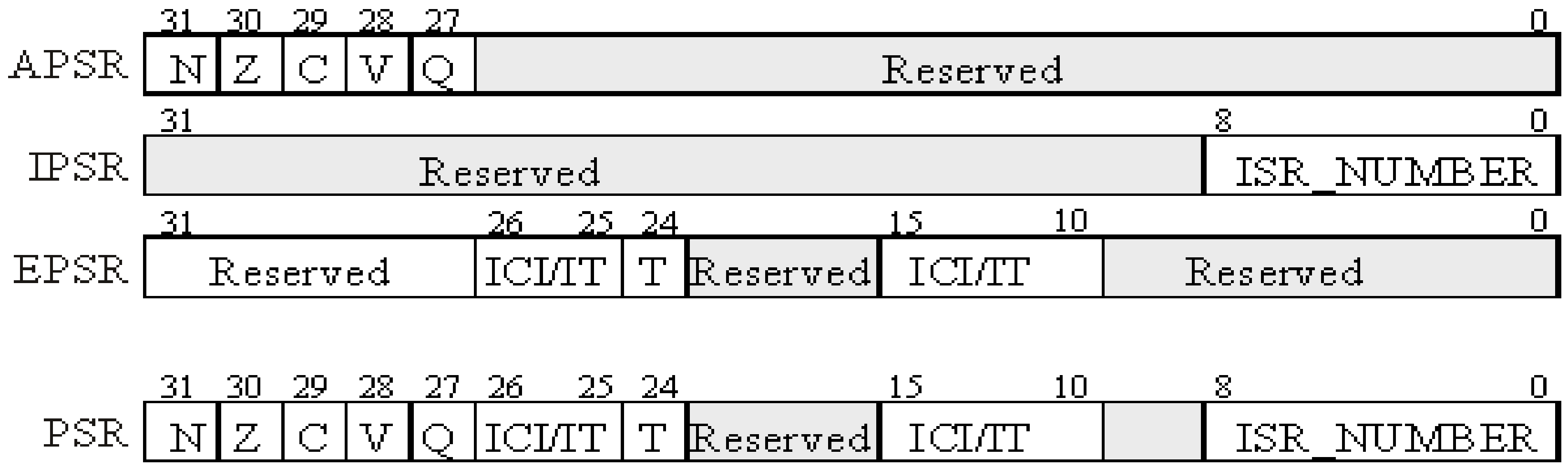
- There are three status registers
  - **Application Program Status Register** (**APSR**)
  - **Interrupt Program Status Register** (**IPSR**)
  - **Execution Program Status Register** (**EPSR**)
- These registers have special functions and can be accessed only by special instructions and cannot be used for normal data processing
- Can be accessed individually or in combination as the **Program Status Registers** (**PSR**)



Program Status Register

REGISTERS ON ARM CORTEX-M PROCESSOR

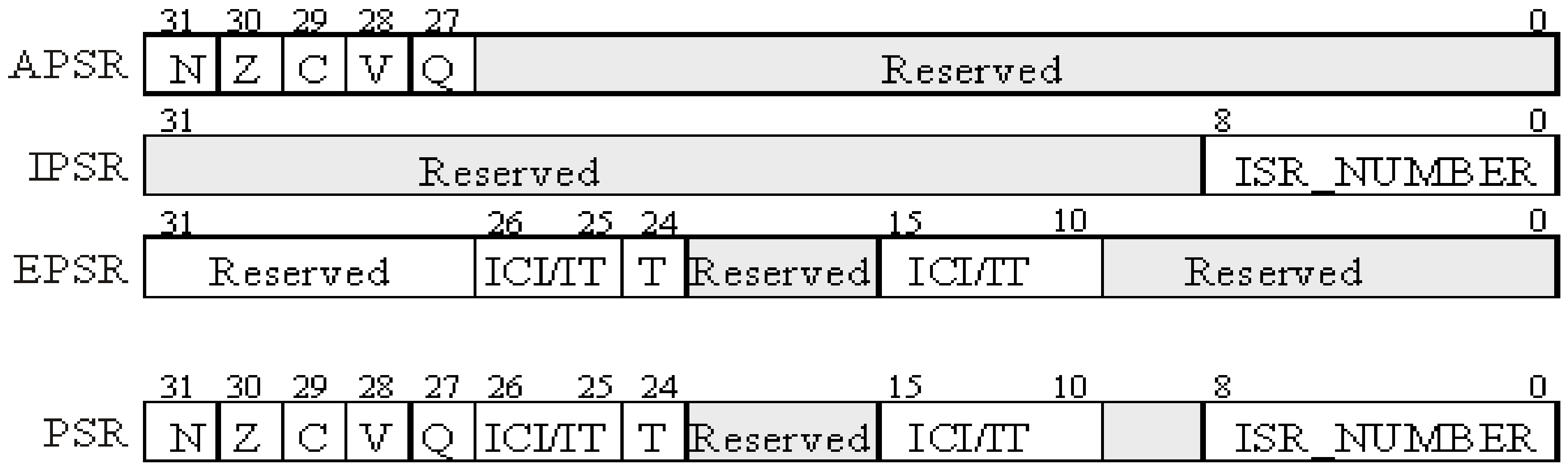
- The **N**, **Z**, **V**, **C**, and **Q** bits give information about the result of a previous **ALU** operation
  - **N** bit is set after an arithmetical or logical operation signifying whether the result is negative
  - **Z** bit is set if the result is zero
  - **C** bit means carry and is set on an unsigned overflow
  - **V** bit signifies signed overflow
  - **Q** bit indicates that "saturation" has occurred



Program Status Register

REGISTERS ON ARM CORTEX-M PROCESSOR

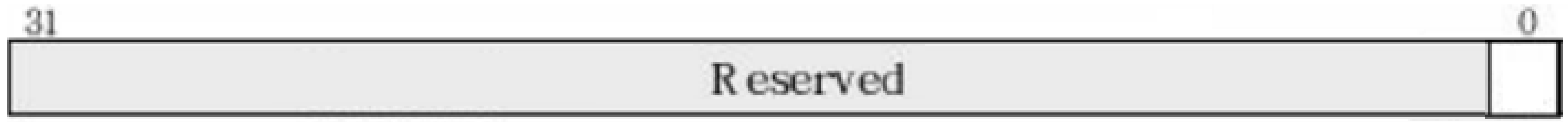
- The **ISR_NUMBER** indicates which interrupt if any the processor is handling
- The **T** bit will always be 1, indicating the ARM ® Cortex™-M processor is executing Thumb ® instructions
- If-Then (**IT**) instruction or the Interruptible-Continuable Instruction (**ICI**) field for an interrupted load multiple or store multiple instruction



Priority Mask Register (PRIMASK)

REGISTERS ON ARM CORTEX-M PROCESSOR

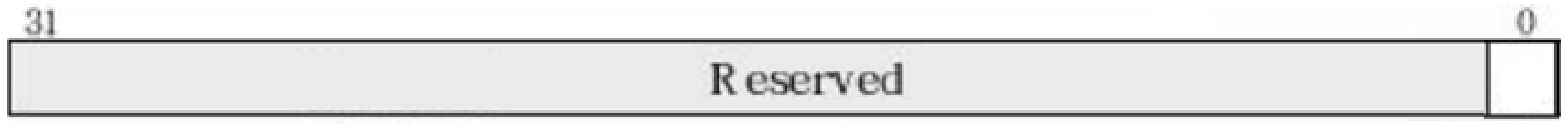
- Bit **0** of the special register **PRIMASK** is the interrupt mask bit.
  - If this bit is 1, interrupts and exceptions with programmable priority are not allowed.
  - If the bit is 0, then interrupts are allowed.
- Reset, non-maskable interrupt (NMI), and hard fault are the only exceptions with fixed priority that are not masked by PRIMASK
- One of the common usages of PRIMASK is to disable all of the interrupts when executing a critical code section that should not be interrupted once its execution starts



Fault Mask Register (FAULTMASK)

REGISTERS ON ARM CORTEX-M PROCESSOR

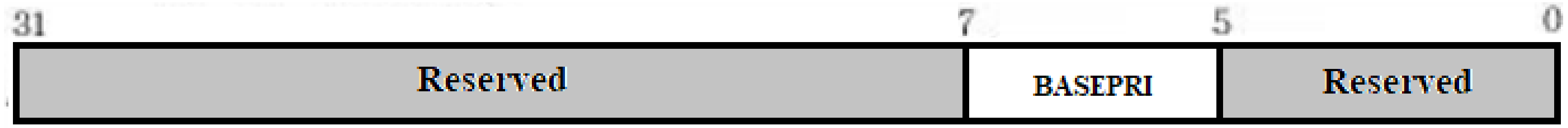
- Bit **0** of the special register **FAULTMASK** is the fault mask bit.
  - If this bit is 1, interrupts and exceptions are not allowed except Reset and NMI
  - If the bit is 0, then interrupts and faults are allowed.
- The interrupt service routine corresponding to FAULTMASK can efficiently avoid any further triggering of fault interrupts.
- For instance, FAULTMASK can be used to suppress any bus faults.
- In contrast to PRIMASK, the FAULTMASK is cleared automatically when returning from an exception.



Base Priority Mask Register (BASEPRI)

REGISTERS ON ARM CORTEX-M PROCESSOR

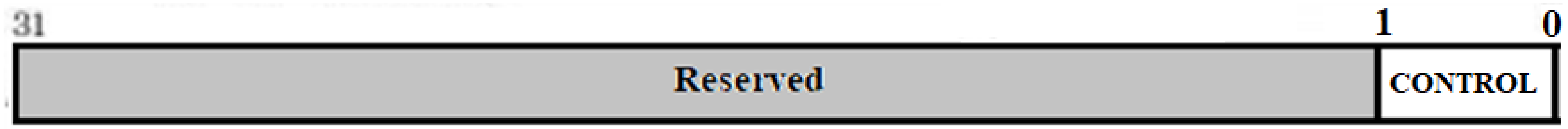
- It used for disabling interrupts, with flexibility, temporarily when dealing with time critical applications
- When BASEPRI is set to a nonzero value, it blocks all the interrupts of either the same or lower priority, while it allows the processor to accept the interrupts of higher priority for processing.
- For example, if **BASEPRI** bit 5-7 equals 3, then requests with level 0, 1, and 2 can interrupt, while requests at levels 3 and higher will be postponed.
- A lower number means a higher priority interrupt. The details of interrupt processing will be presented later on.
- When BASEPRI is set to 0, it is disabled.



Control Register (CONTROL)

REGISTERS ON ARM CORTEX-M PROCESSOR

- Bit0 and Bit1 of this register is used as following
- **Bit0**: Privilege level selection
  - 0 sets thread mode as privileged mode
  - 1 makes thread mode switch to unprivileged (user) level
- **Bit1**: Selection of stack pointer register
  - 0 (default setting) main stack is used through main stack pointer (MSP) register
  - 1 process stack is used through process stack pointer (PSP) register



Levels of Operations

REGISTERS ON ARM CORTEX-M PROCESSOR

There are two levels of operation associated with the processor

- **Privileged Level**:
  - Permission to modify the contents in system control space (SCS), which is a part of the memory region for configuration registers and debugging components, is allowed
  - The software can use all the instructions and has access to all resources
- **Unprivileged (User) Level**:
  - Permission to modify the contents in SCS is blocked
  - Has limited access to the MSR and MRS instructions, and cannot use the CPS instruction
  - Might have restricted access to memory or peripherals like system timer, NVIC, or system control block
- Only privileged software can write to the CONTROL register to change the privilege level for software execution in Thread mode.
- Unprivileged software can use the SVC instruction to make a *supervisor call* to transfer control to privileged software.

PROCESSOR OPERATING MODES

Thread Mode vs Handler Mode

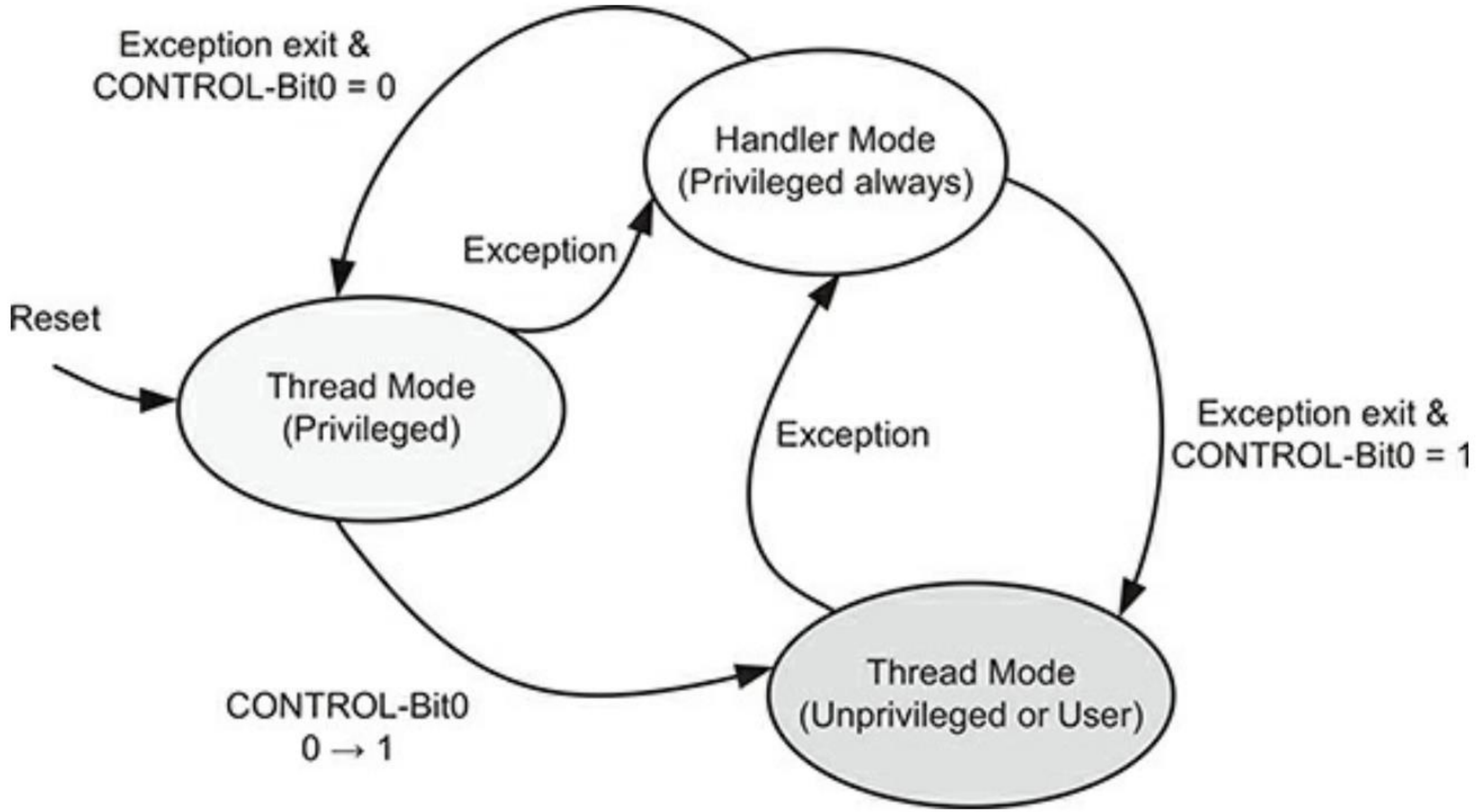
Thread Mode vs Handler Mode

PROCESSOR OPERATING MODES

- There are two running modes for Cortex-M Processor:
  - **Thread Mode:** The processor enters Thread mode (also known as **foreground**) on reset or as a result of an exception return. Code execution in thread mode can have either privileged or unprivileged (user) access levels.
  - **Handler Mode:** The processor enters Handler mode (also known as **background**) as a result of an exception/ISR. All code is privileged in handler mode.
- The processor knows whether it is running in the thread mode or in the handler mode
- Switching between thread and handler modes occurs automatically.
- The processor begins in thread mode, signified by ISR_NUMBER=0.
- Whenever it is servicing an interrupt it switches to handler mode, signified by setting ISR_NUMBER to specify which interrupt is being processed.

Possible Transitions for different Privilege Levels

PROCESSOR OPERATING MODES

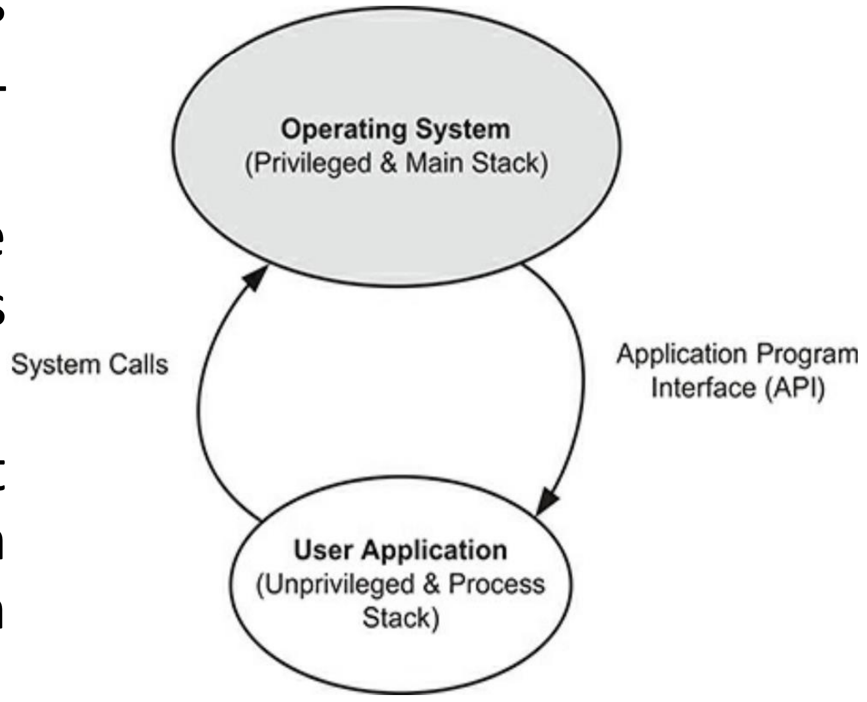


Thread Mode vs Handler Mode

PROCESSOR OPERATING MODES

RTOS based User Application Development

- A user application running at privilege level can access critical SCS memory region as well as the CONTROL register enabling it to modify their content
- In case of third-party application, running at privilege level, crash might lead to entire system crash due to its ability to access critical region
- However, a third-party application, running at unprivileged level, crash will only lead to application crash and rest of the system and other applications will remain functioning normally
- **Therefore, the operating system is granted privileged access, while the applications run at unprivileged level**


Operating System
(Privileged & Main Stack)
System Calls
Application Program Interface (API)
User Application
(Unprivileged & Process Stack)

Bit0 and Bit1 of CONTROL Register

PROCESSOR OPERATING MODES

Bit0	Bit1	Description
0	0	For simple applications: Entire program runs at privileged level and only the Main Stack Pointer (MSP) is used by both the main program as well as interrupt handlers.
0	1	User application runs on top of RTOS in privileged thread mode. The user application uses Process Stack Pointer (PSP), while the MSP is used by the operating system kernel and exception or interrupt handlers.
1	0	The user application is running with unprivileged access level and still uses MSP. It is highly unlikely that this scenario will be used by the user applications, where the user application and the operating system share the stack memory.
1	1	User application runs on top of RTOS in unprivileged thread mode. The user application uses Process Stack Pointer (PSP), while the MSP is used by the operating system kernel and exception or interrupt handlers.

ARM CORTEX-M MEMORY

Memory & I/O Modules, Memory Map, Ways of Storing Multi-byte Data

Memory & I/O Modules

ARM CORTEX-M MEMORY

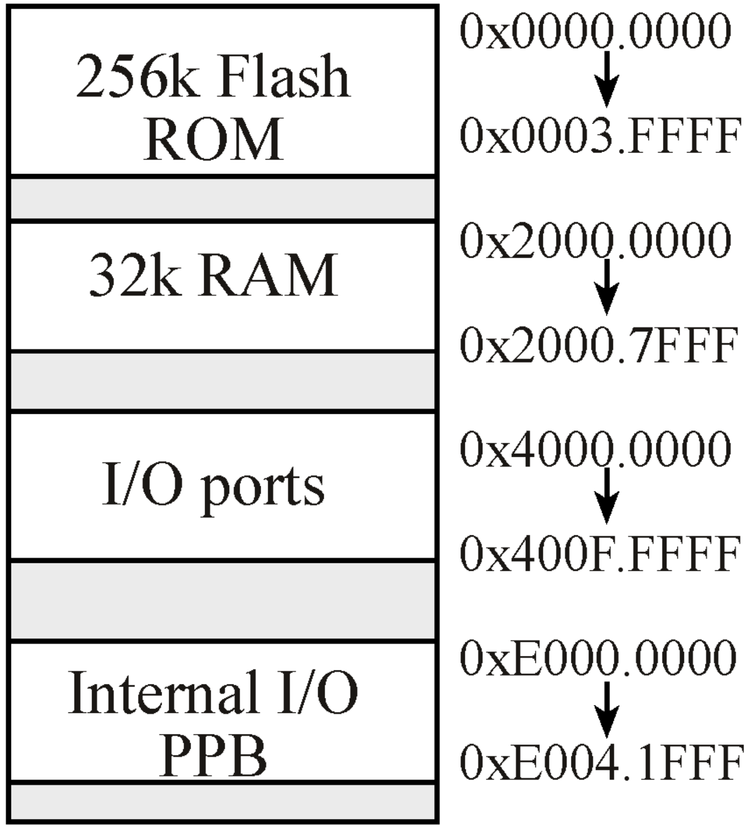
- Microcontrollers within the same family differ by the amount of memory and by the types of I/O modules
- All of these have SysTick, RTC, timers, UART, I2C, SSI, and ADC

Part Number	RAM	Flash	I/O	I/O Module
TM4C1231C3PM	32	12	43	floating point, CAN, DMA
TM4C1233H6PM	32	256	43	floating point, CAN, DMA, USB
TM4C123GH6PM	32	256	43	floating point, CAN, DMA, USB, PWM
TM4C123GH6ZRB	32	256	120	floating point, CAN, DMA, USB, PWM
TM4C1294NCPDT	256	1024	90	floating point, CAN, DMA, USB, PWM, Ethernet

Memory Map

ARM CORTEX-M MEMORY

- Although specific for the TM4C123, all ARM ® Cortex™-M microcontrollers have similar memory maps
- In general,
  - Flash ROM begins at address 0x0000.0000,
  - RAM begins at 0x2000.0000,
  - the peripheral I/O space is from 0x4000.0000 to 0x5FFF.FFFF,
  - and I/O modules on the private peripheral bus (PPB) exist from 0xE000.0000 to 0xE00F.FFFF
- In particular, the only differences in the memory map for the various 180 members of the LM3S/TM4C family are the ending addresses of the flash and RAM

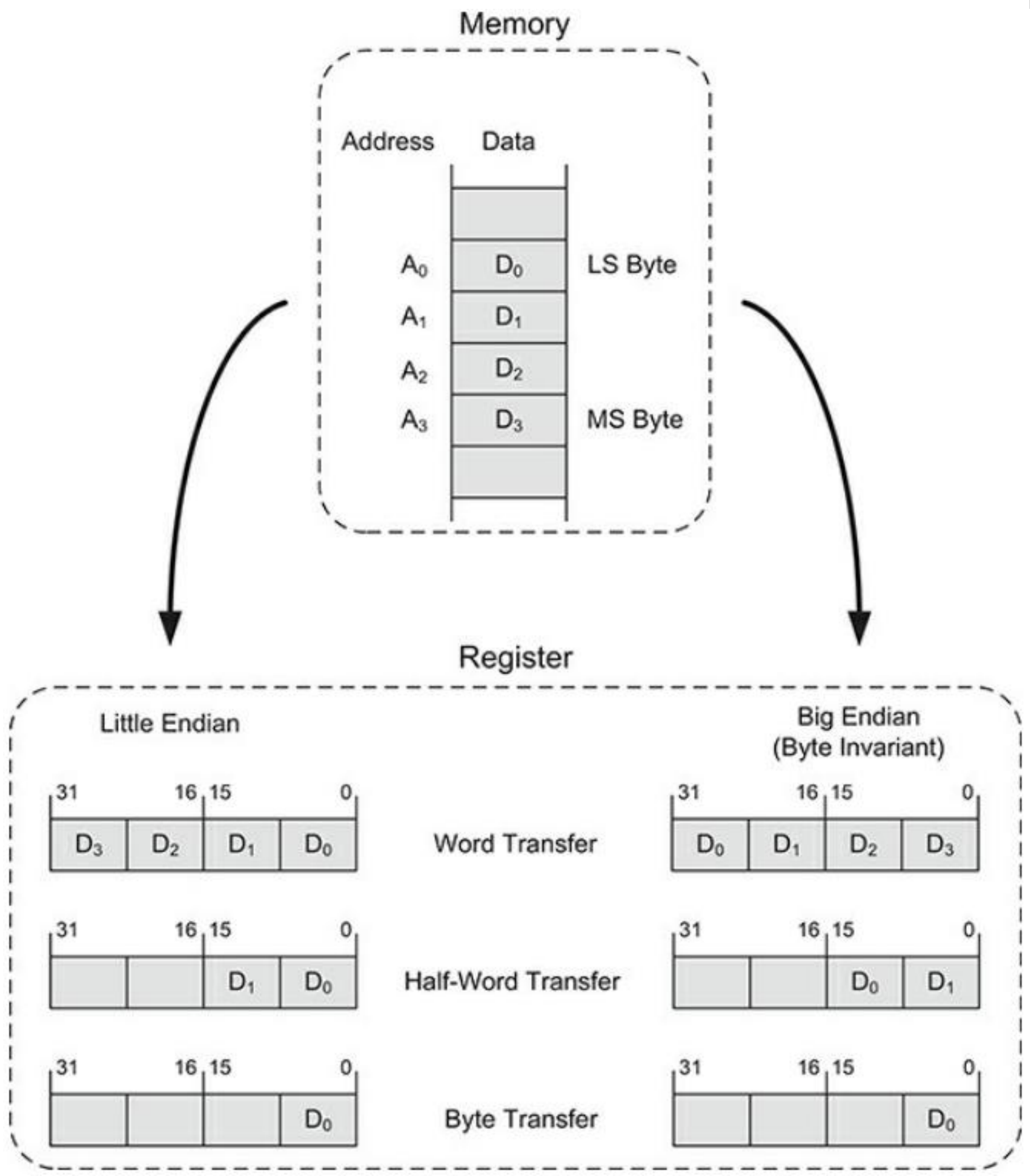


Memory Map of TM4C123GH6PM

Memory Endianness

ARM CORTEX-M MEMORY

- Since the memory systems on most computers are byte addressable (a unique address for each byte), therefor there are two possible ways to store multi-byte of data
- **Little-endian** is an order in which the "little end" (least significant value in the sequence) is stored first
- **Big-endian** is an order in which the "big end" (most significant value in the sequence) is stored first, at the lowest storage address
- These two approaches only makes sense if you capture the multi-byte data as one non-divisible piece of information



Ways of Storing Multi-byte Data

ARM CORTEX-M MEMORY

- **Freescale** microcomputers implement the big-endian approach
- **Intel** microcomputers implement the little-endian approach
- **Cortex-M** microcontrollers use the little-endian format
- Instruction fetches on the **ARM** are always little endian
- Many **ARM processors** are bi-endian, because they can be configured to efficiently handle both big- and little-endian data
- An error will occur when data is stored in Big Endian by one computer and read in Little Endian format on another

INTERRUPTS AND PROCESSOR RESET SEQUENCE

ARM Cortex-M Processor

What is meant by an Interrupt?

INTERRUPTS AND PROCESSOR RESET SEQUENCE

- Normally, a processor executes the program in a predefined sequence.
- However, due to the interaction with real world physical phenomenon, we might be interested in knowing if an event of interest has occurred.
- One possible solution, called **Polling**, is to keep on checking an appropriate indicator for the occurrence of the event. This solution is inefficient because we use processing resources in checking the event even if it has not occurred.
- An alternative, more efficient, solution is based on **interrupts**, where the processor is informed by the dedicated interrupt related hardware, configured appropriately, about the occurrence of an event. It causes deviation from the normal sequence of program execution

What is meant by an Interrupt?

INTERRUPTS AND PROCESSOR RESET SEQUENCE

- So, the interrupt can be thought of as an indicator about the occurrence of an event, which can be generated by either hardware or software.
- In case of an event or exception, it is natural to think that a response is required from the processor.
- The processor response generally involves a sequence of operations to be performed and is implemented as a dedicated response function and is called **service routine**.
- The sequence of operations involved from the occurrence of an event to the resumption of normal program execution again has the following main steps.
    1. When an event occurs, an interrupt request is generated to the processor.
    2. The processor suspends current task execution in response to the interrupt and starts executing an interrupt service routine to generate the response to the interrupt
    3. Once the response to the event is completed, processor resumes execution of the suspended task.

Reset Interrupt

INTERRUPTS AND PROCESSOR RESET SEQUENCE

- When a microcontroller is powered or if it reset by pressing the reset button, an interrupt called **reset interrupt** is generated.
- Every processor has an associated sequence of operations, which it performs to reset event.
- The first operation performed by the Cortex-M processor after a reset or power-on involves reading of two words from instruction memory.
- **Address 0x00000000**: This address in the code memory (most likely a Flash memory) contains the starting value that is loaded to the main stack pointer, MSP (R13).
- **Address 0x00000004**: Reset vector is contained at this address and the program counter is loaded with this value to jump to the reset interrupt service routine. The main function is called from within reset interrupt service routine

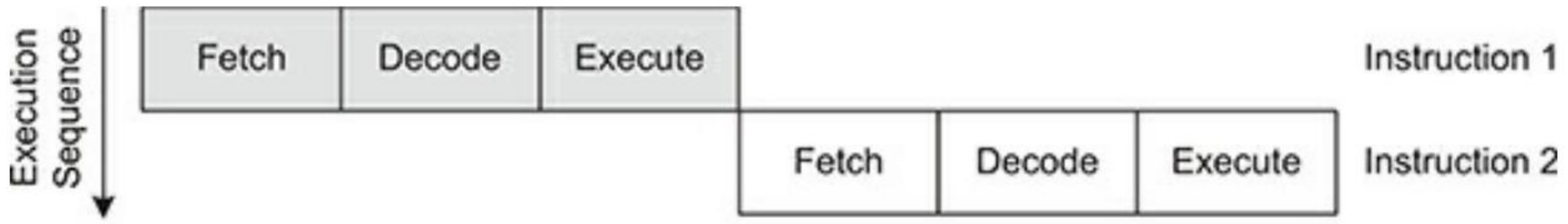
PIPELINED ARCHITECTURE

ARM Cortex-M Processor

Instruction Processing

PIPELINED ARCHITECTURE

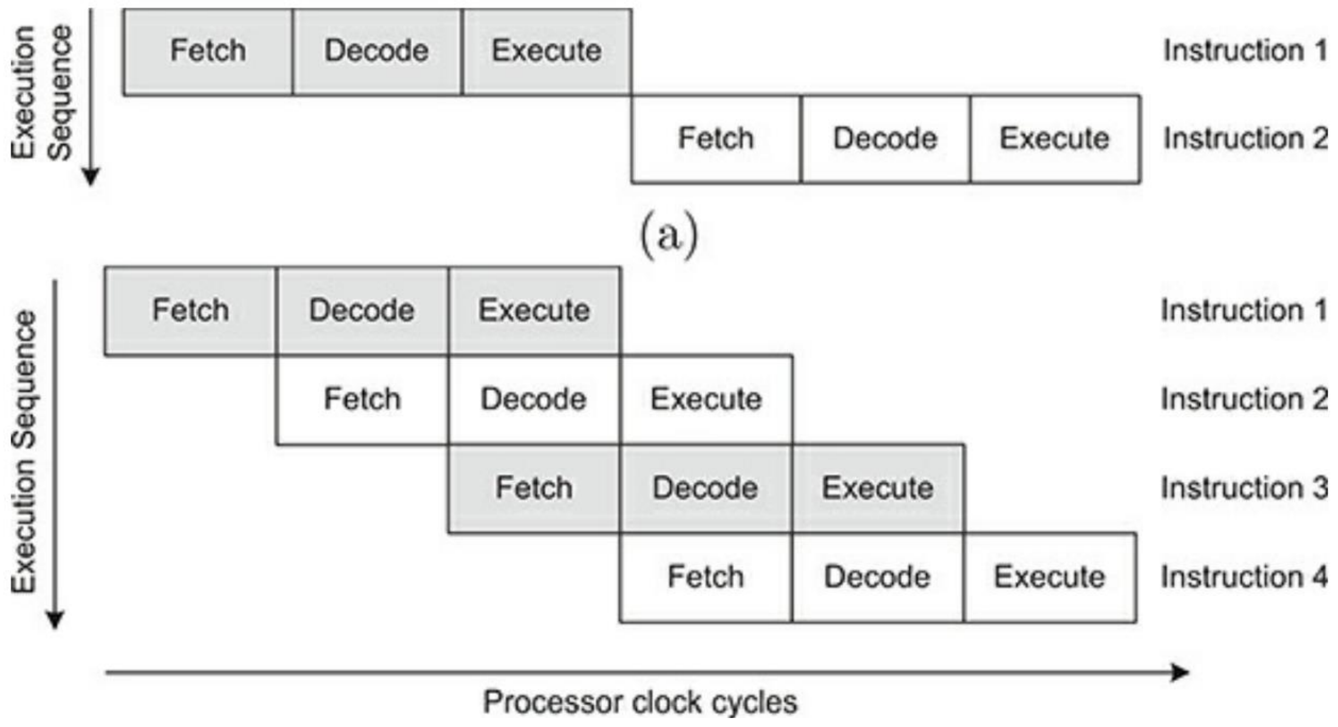
- The three basic steps involved in processing of an instruction are fetch, decode, and execute.
- Each of these step must take at least one clock cycle to complete
- If a microprocessor follows these steps in sequential fashion, the it will require six clock cycles to complete the execution of two instructions
- We can observe that while the microprocessor is decoding the first instruction the buses remain idle



Instruction Processing

PIPELINED ARCHITECTURE

- Using this pipeline architecture, we only need 4 clock cycles to execute two instructions
- When the result of first instruction is written back to memory, as a result of instruction execution, the third instruction is fetched at the same time.
- It is possible due the fact that we have two separate buses for accessing code and data (i.e., Harvard Architecture)



THANK YOU

Any Questions???