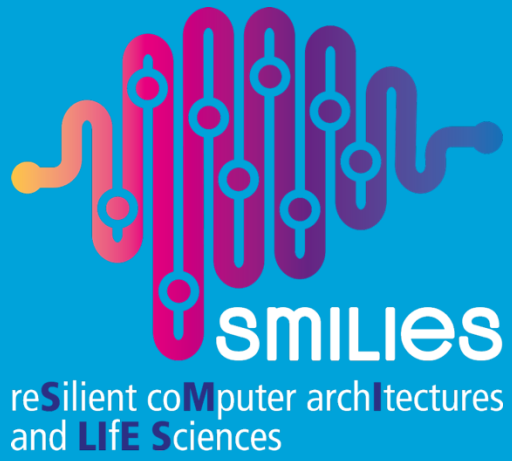




Politecnico di Torino

Department of Control and Computer Engineering


# INTRODUCTION TO ARM PROCESSOR

STEFANO DI CARLO

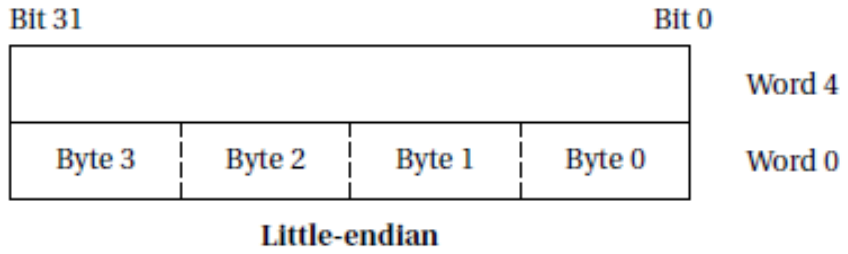
# FLOW OF TOPICS

- ARM Architecture
- ARM programmer’s model
- ARM Development tools
- Memory Hierarchy
- ARM Assembly Language Programming
- Simple Examples
- Architectural Support for Operating systems

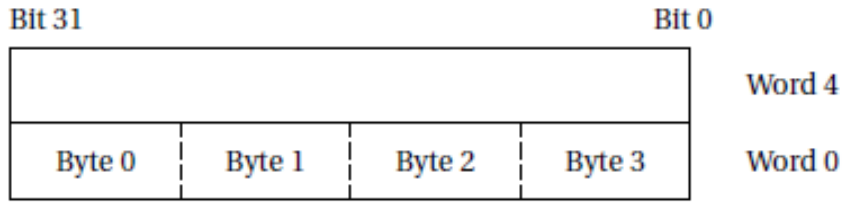
# ARM ARCHITECTURE

# ARM ARCHITECTURE

- Follows RISC (Reduced Instruction Set Computer) Architecture
- Both in Von Neumann and Harvard Architecture
- Both Little endian and Big endian
  - 32-bit processor
  - 32-bit address line
- Features Used from RISC design
  - A Load/Store Architecture
  - Fixed length 32-bit instructions
  - 3 operands instruction formats
- Features rejected from RISC design
  - Register windows: several set of registers used for different function calls
  - Delayed branches: a technique to reduce pipeline stalls caused by branch instructions. The instruction immediately following a branch instruction (the branch delay slot) is always executed, regardless of whether the branch is taken or not.
  - Single cycle execution of all instructions



Little-endian



Big-endian

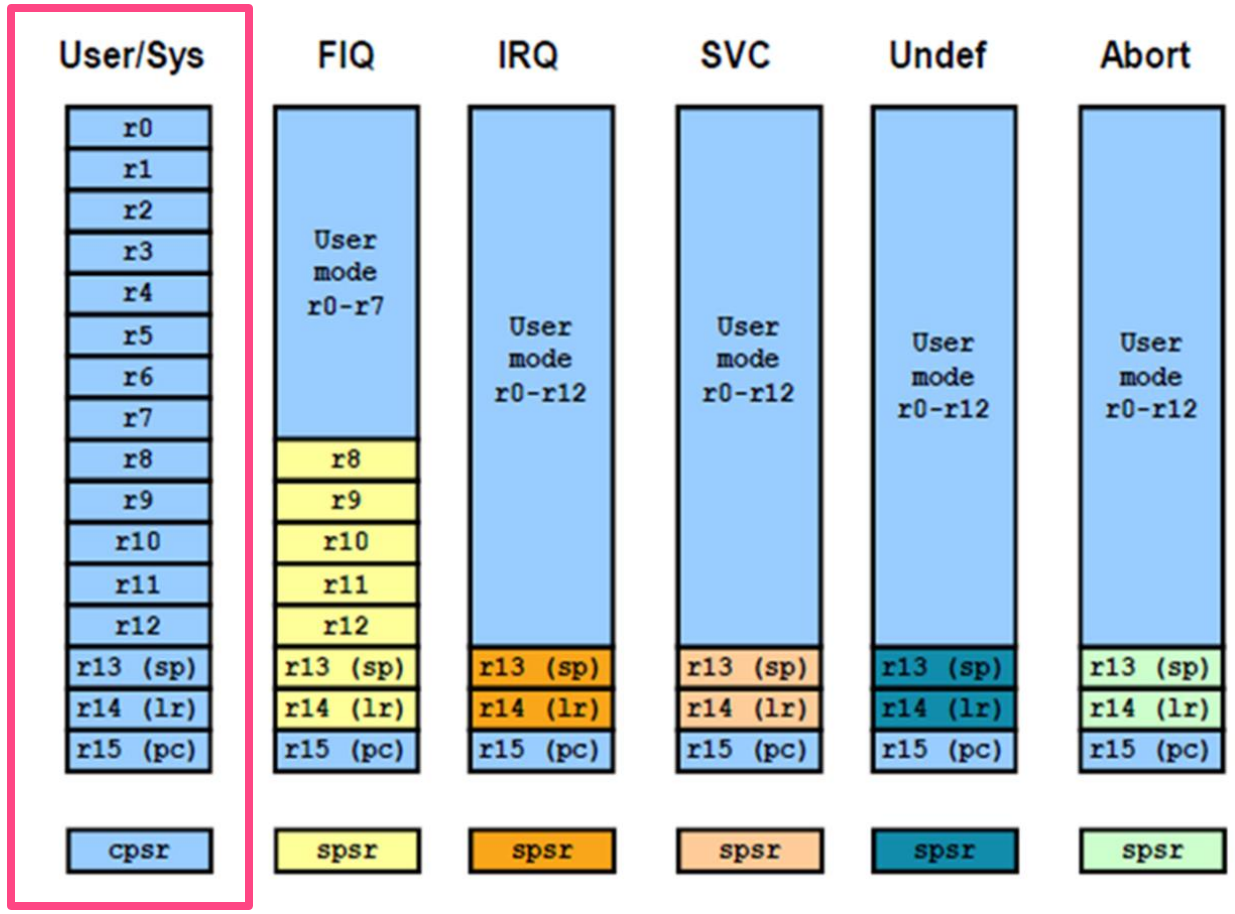
# Programming Model

# PROGRAMMING MODEL

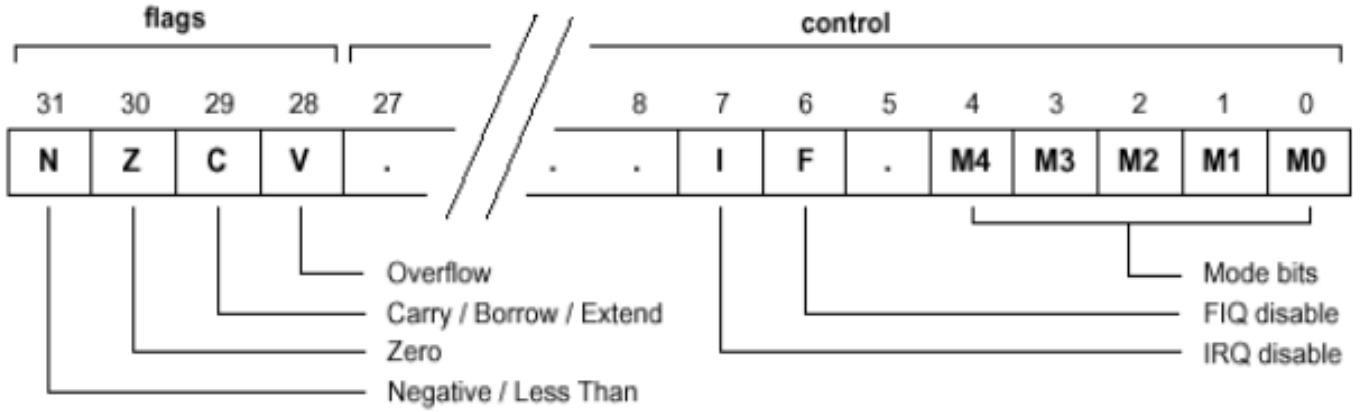
- ARM’s Register
- CPSR Register
- Memory System
- Load Store Architecture
- ARM Instruction Set
- I/O system
- ARM exceptions

# ARM REGISTERS (USER MODE)

- **General purpose registers (r0 to r12):** 32bit general purpose registers.
- **SP - Stack Pointer (r13):** typically used as the stack pointer, pointing to the top of the current stack in memory.
- **LR - Link Register (r14)**: holding the return address for function calls. When a function is called, the address of the instruction following the call is stored in R14, so the program can return to that point after the function execution completes.
- **PC - Program Counter (r15):** holds the address of the next instruction to be executed. It automatically increments with each instruction fetch and can also be manipulated for branching and jumping operations.
- **Status register (CPSR)**

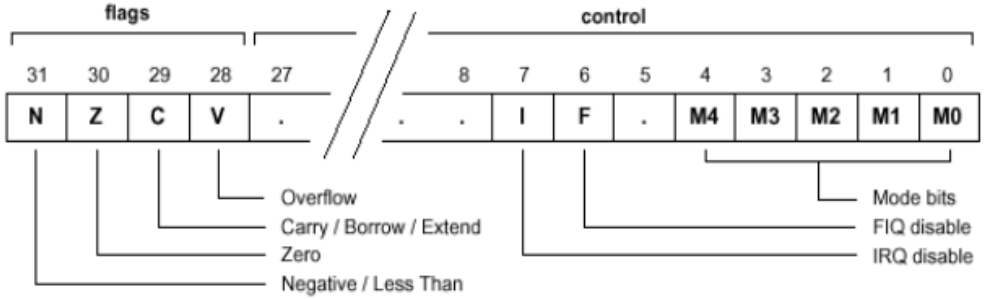


# CURRENT PROGRAM STATUS REGISTERS




- Condition code flags
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation Carried out
  - V = ALU operation oVerflowed
- Mode bits
  - Specify the processor mode
- Interrupt Disable bits.
  - I = 1: Disables the Interrupt ReQuest (IRQ).
  - F = 1: Disables the Fast Interrupt request (FIQ).
- T Bit
  - Architecture xT only
  - T = 0: Processor in ARM state
  - T = 1: Processor in Thumb state

# CURRENT PROGRAM STATUS REGISTERS




b10000 = User mode

b10001 = FIQ mode

b10010 = IRQ mode

b10011 = Supervisor mode

b10111 = Abort mode

b11011 = Undefined mode

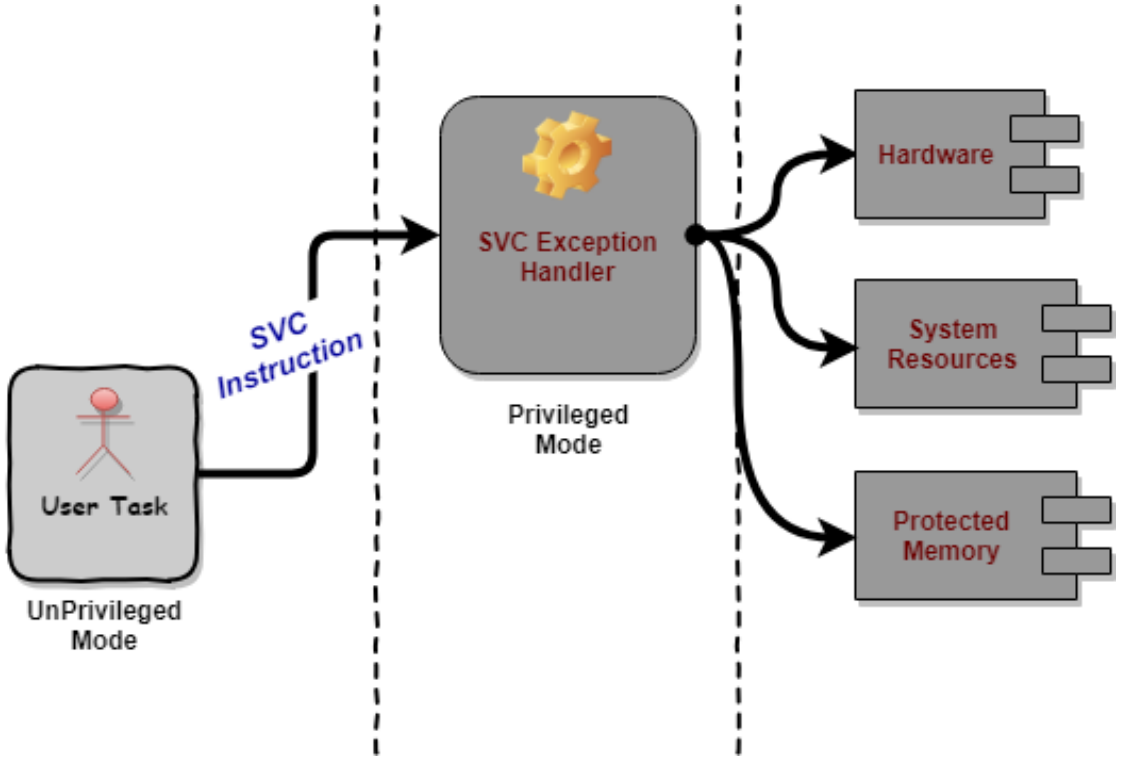
b11111 = System mode.

- **User mode(usr)** — normal program execution mode
- **Fast Interrupt mode (fiq)** — supports a high-speed data transfer or channel process.
- **Interrupt mode (irq)** — psed for general-purpose interrupt handling.
- **Supervisor mode (svc)** — protected mode for the operating system.
- **Abort mode (abt)** — implements virtual memory and/or memory protection
- **System mode (sys)** — A privileged user mode for the operating system. (runs OS tasks)
- **Undefined mode (und)** — supports a software emulation of hardware coprocessors

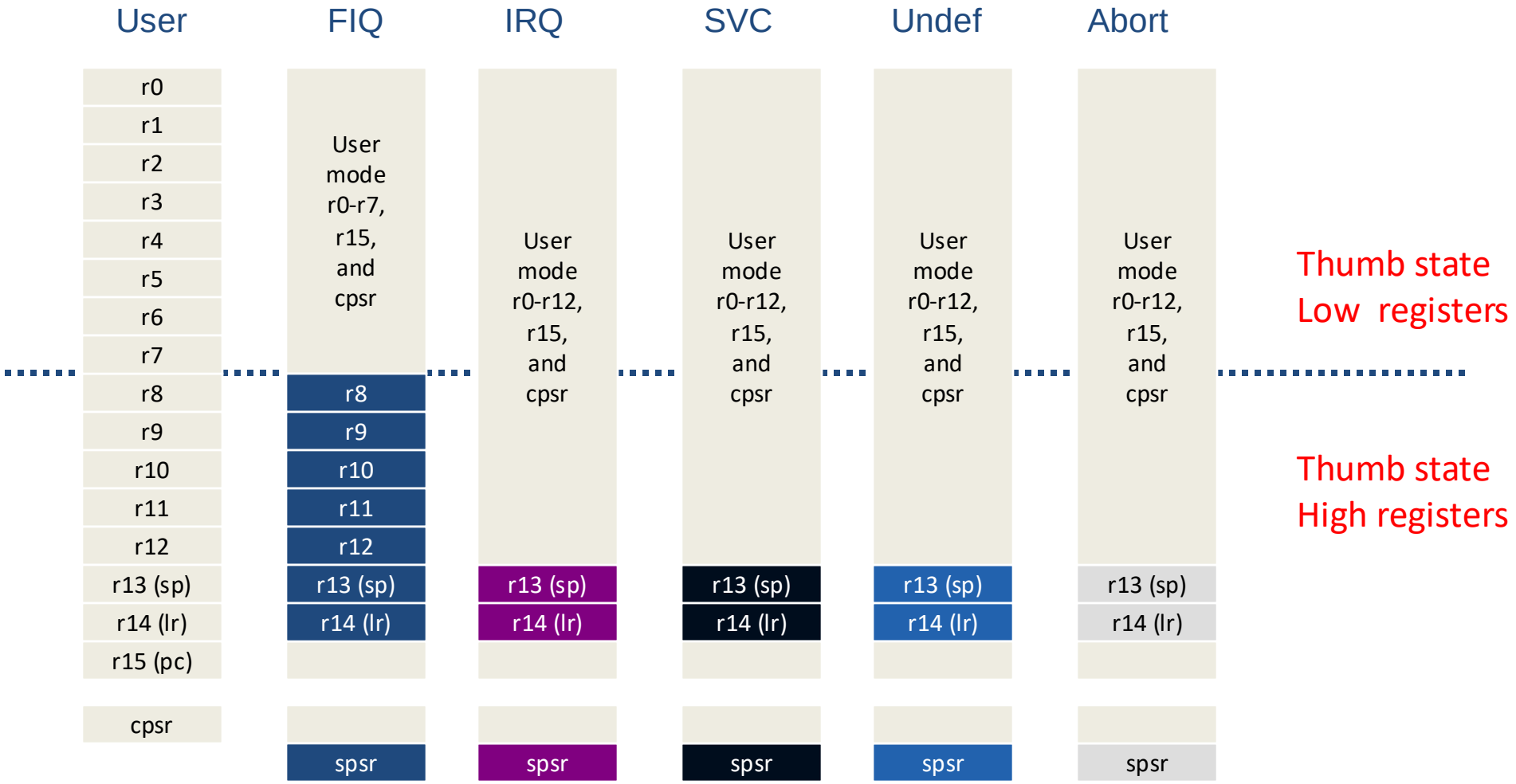
Except user mode, all are known as privileged mode.

# SUPERVISOR MODE

- A protection mechanism ensures that user code cannot gain supervisor privileges without appropriate checks.
- System-level functions can only be accessed through specified supervisor calls.
- These functions generally include any accesses to hardware peripheral registers, and to widely used operations, i.e., character input and output.
- The svc mode can be entered when an SVC instruction is executed.



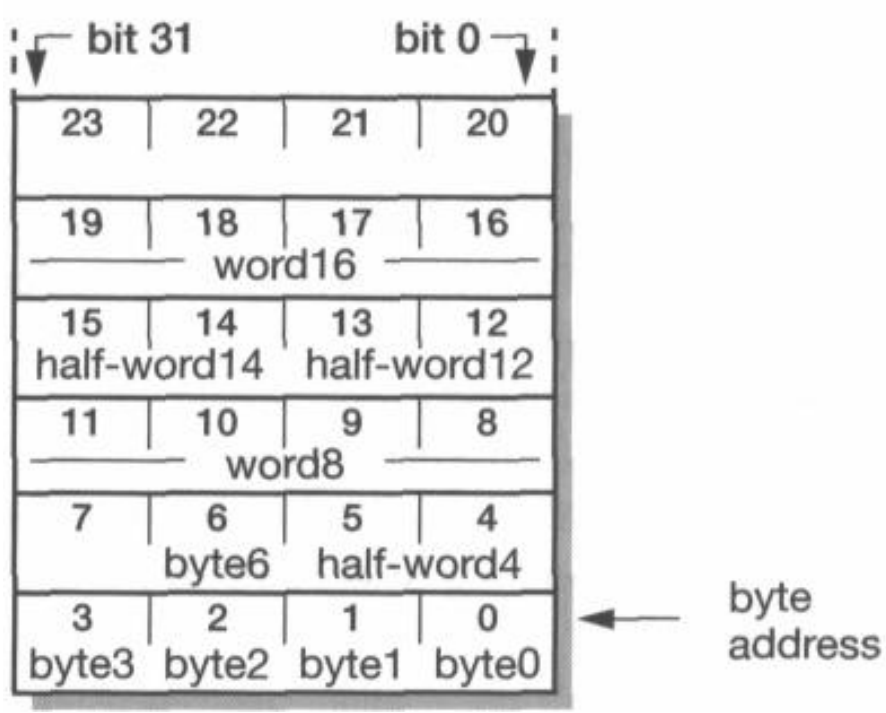
# REGISTER ORGANIZATION SUMMARY




Note: System mode uses the User mode register set

# MEMORY SYSTEM

- In addition to the processor register, ARM system has memory state
- Memory may be viewed as a linear array of bytes numbered form 0 up to $2^{32} - 1$
- Data items may be 8-bit bytes, 16-bit half words or 32-bit words
  - Words are always aligned on 4-byte boundaries
  - Half words are aligned on even byte boundaries
  - Byte may occupy any of these locations



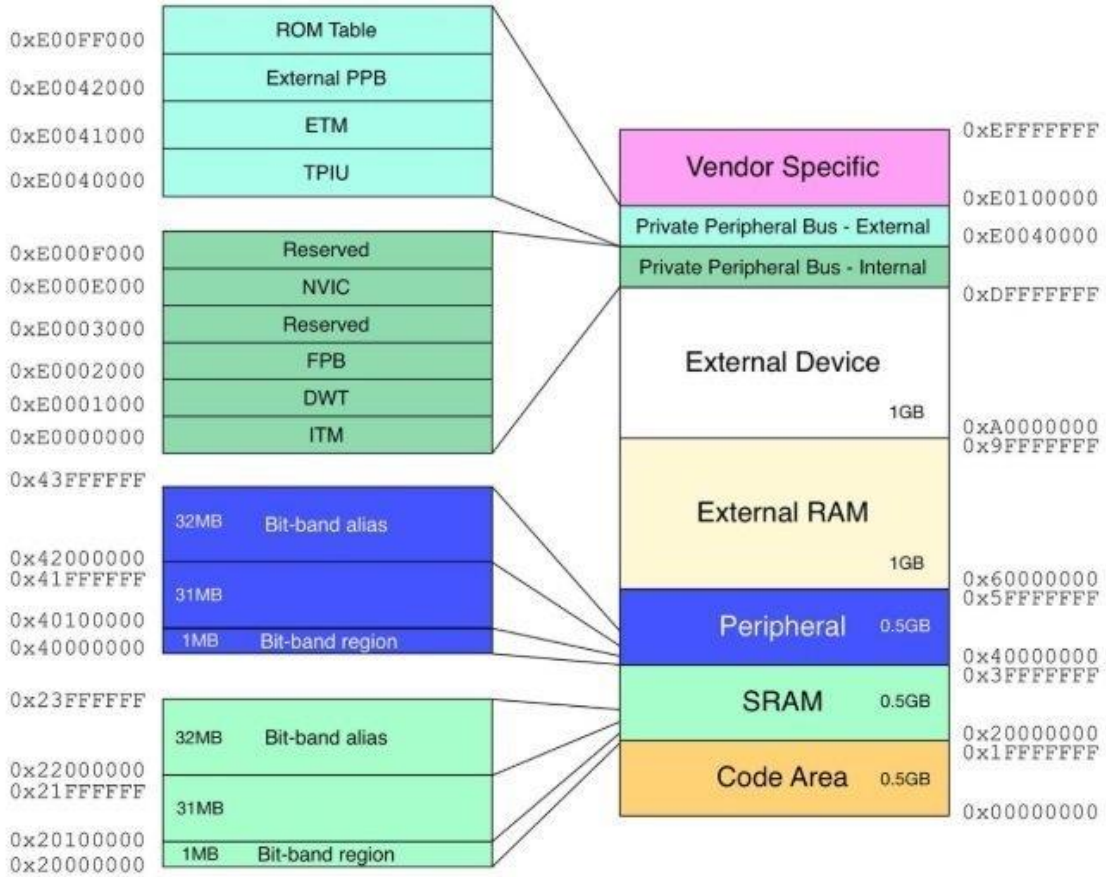
# LOAD STORE ARCHITECTURE

- ARM does not support 'memory-to-memory' operations.
- All ARM instructions fall into one of the following three categories:
  - **Data processing instructions** — use and change only register values. For example, an instruction can add two registers and place the result in a register.
  - **Data transfer instructions** — copy memory values into registers (load instructions) or copy register values into memory (store instructions).
  - **Control flow instructions** —cause execution to switch to a different address, either permanently (branch instructions) or saving a return address to resume the original sequence (branch and link instructions) or trapping into system code (supervisor calls).

# I/O SYSTEM

- The ARM handles I/O (input/output) peripherals (such as disk controllers, network interfaces, and so on) as memory-mapped devices with interrupt support.
- The internal registers in these devices appear as addressable locations within the ARM's memory map and may be read and written using the same (load-store) instructions as any other memory locations.

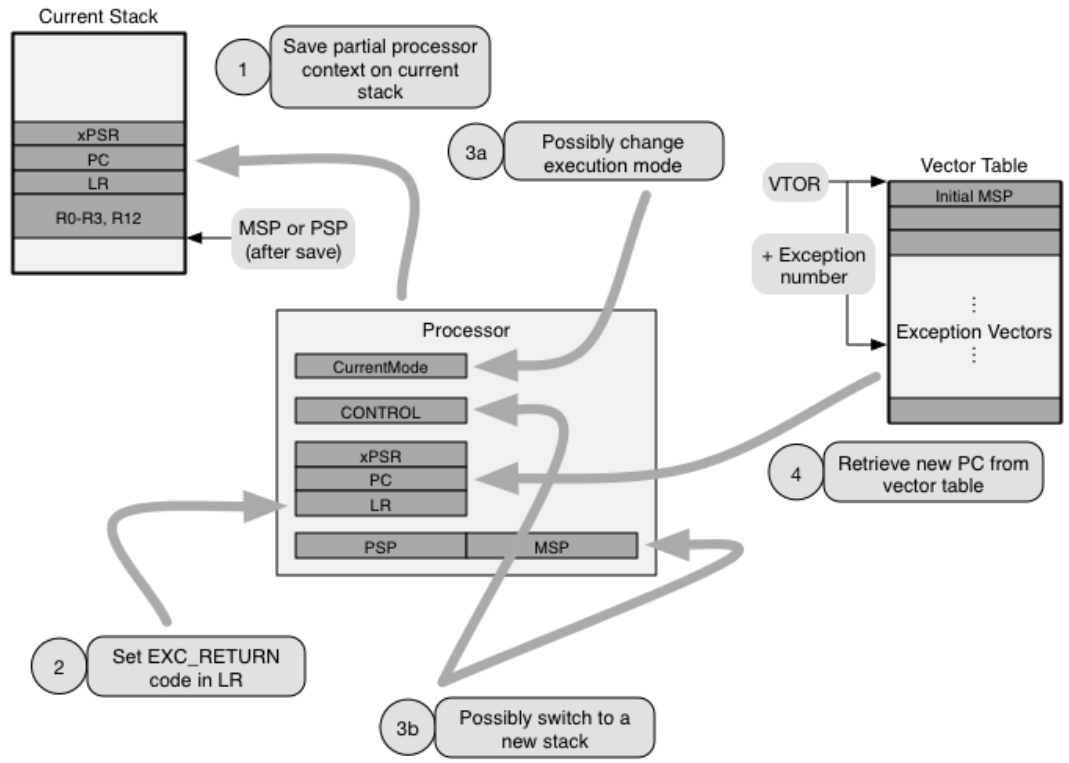
TM4C123GH6PM ARM Cortex M4 microcontroller




Taken form https://stackoverflow.com/questions/74227194/how-does-memory-mapped-i-o-mmio-work-on-arm-architectures

# ARM EXCEPTIONS

- The ARM architecture supports a range of interrupts, traps and supervisor calls, all grouped under the general heading of **exceptions**.
  - **Step 1:** The processor saves the current execution context (registers, program counter, and processor status) onto the stack. The stack pointer is set to point to the current stack, which is typically the main stack pointer (MSP) or process stack pointer (PSP), depending on the configuration.
  - **Step 2:** Set the return code in the Link Registrer
  - **Step 3:** Change the execution mode (if needed) and set a new stack (if needed)
  - **Step 4:** The processor uses the vector table to identify the corresponding exception handler address for the triggered exception. The vector table is typically located at a fixed memory address.
- Once the exception is handled, the processor executes the **BX LR** instruction (or **BX** with the link register) to return to the main program. Before returning, the processor restores the saved context from the stack.



# ARM INSTRUCTION SET

- The most notable features of the ARM instruction set are:
  - 32 bits wide
  - Load-store architecture
  - 3-address data processing instructions
  - conditional execution of every instruction
  - The inclusion of very powerful load and store multiple register instructions
  - The ability to perform a general shift operations and a general ALU operation in a single instruction that executes in a single clock cycle
  - Open instruction set extensions through the coprocessor instruction set, including adding new registers and data types to the programmer's model
  - A very dense 16-bit compressed representation of the instruction set in the **Thumb architecture**

# ARM ASSEMBLY LANGUAGE PROGRAMMING

- Data processing Instructions
- Data Transfer Instructions
- Control flow Instruction

- [https://iitd-plos.github.io/col718/ref/arm-instructionset.pdf]

# DATA PROCESSING INSTRUCTIONS

- Simple register operands
- Register movement operations
- Comparison operations
- Immediate Operands
- Shifted register operands
- Multiplies

# SIMPLE REGISTER OPERANDS

```
; Simple math operations
ADD r0, r1, r2  ; r0 := r1 + r2
ADC r0, r1, r2  ; r0:= r1 + r2 + C
SUB r0, r1, r2  ; r0 := r1 - r2
SBC r0, r1, r2  ; r0 := r1 - r2 + C -1
RSB r0, r1, r2  ; r0 := r2 - r1
RSC r0, r1, r2  ; r0 := r2 - r1 + C - 1
; Simple logical operations
AND r0, r1, r2          ; r0 := r1 and r2
ORR r0, r1, r2          ; r0 := r1 or r2
EOR r0, r1, r2          ; r0 := r1 xor r2
BIC r0, r1, r2          ; r0 := r1 and nor r2 (Bit Clear)
```

# REGISTER MOVEMENT OPERATIONS

```
MOV r0, r2          ; r0 := r2
MVN r0, r2          ; r0 := not r2
```

# COMPARISON OPERATIONS

```
CMP r1, r2          ; Compare and set flags
TST r1, r2          ; As AND but result is not written
CMN r1, r2          ; As ADD, but result is not written
TEQ r1, r2          ; As EOR, but result is not written
```

# IMMEDIATE OPERANDS

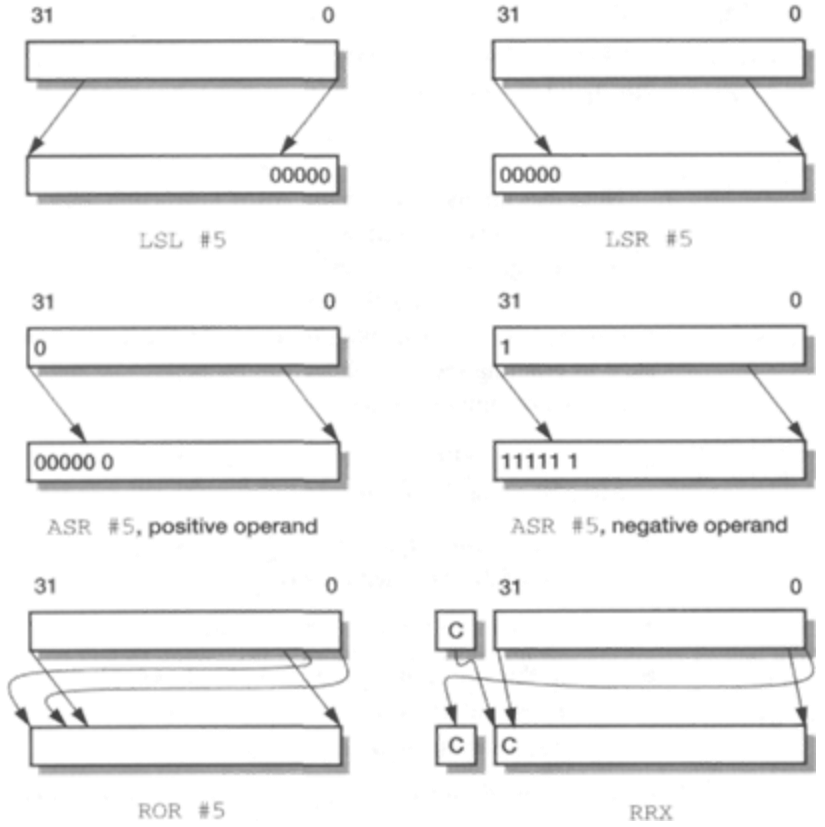
```
ADD    r3, r3, #1      ; r3 = r2 + #1
AND    r8, r7, #FF     ; r8 = r7 + #FF
```

# SHIFTED REGISTER OPERANDS

```
<OPERATION>     r3, r3, #amount
```

- **Logical Shift Left (LSL)** — Shifts bits to the left and fills the vacated bits with zeros
- **Logical Shift Right (LSR)** — Shifts bits to the right and fills the vacated bits with zeros
- **Arithmetic Shift Right (ASR)** — Shifts bits to the right and fills the vacated bits with the sign bit (most significant bit).
- **Rotate Right (ROR)** — Rotates bits to the right. The bits shifted out from the right end are wrapped around to the left end.
- **Rotate Right with Extend (RRX)** — Performs a right rotation with the least significant bit (LSB) being loaded into the carry flag. The most significant bit (MSB) is filled with the carry.

- ARM Instructions ARM instructions can utilize these shift operations directly in their operands.




```
ADD R0, R1, R2, LSL #2  ; R2 is shifted left by 2 before being added to R1
SUB R3, R4, R5, LSR #1  ; R5 is shifted right by 1 before being subtracted from R4
MOV R6, R7, ASR #3      ; R7 is arithmetically shifted right by 3 and moved to R6
```

# MULTIPLICATIONS

```
MUL    r4, r3, r2          ; r4 := r3 * r2
MLA    r4, r3, r2, r1      ; r4 := r3 * r2 + r1


;Immediate second operands are not  supported.
;The result register must not be the same as the first source
register.
```

# DATA TRANSFER INSTRUCTIONS

- Basic addressing
- Multiple register data transfer
- Swap

# DATA TRANSFER INSTRUCTIONS

| | |
|---|---|
| LDR | Load |
| STR | Store |
| LDRH | Load half-word |
| STRH | Store half-word |
| LDRSH | Load half-word signed |
| LDRB | Load byte |
| STRB | Store byte |
| ADR | Set register to address |

# BASIC ADDRESSING

```
; Register Indirect Addressing Mode
LDR   r0, [r1]        ; r0 := mem32[r1]
STR   r0, [r1]        ; mem32[r1] = r0
; Initializing an address pointer
ADR   r1, TABLE       ; r1 stores the Address of TABLE
; Base Plus Offset
LDR   r0,   [r1,#4]   ; r0= mem32[r1+4]
```

# MULTIPLE REGISTER DATA TRANSFER

```
STMIA r0!, {r1,r2}          ; mem32[r0] := r1
                            ; mem32[r0+4] := r2
                            ; r0 := r0 + 8
STMIA r0,  {r1-r3}          ; mem32[r0] := r1
                            ; mem32[r0+4] := r2
                            ; mem32[r0+8] := r3
LDMIA r0!, {r1,r2}          ; r1 := mem32[r0]
                            ; r2 := mem32[r0+4]
                            ; r0 := r0 + 8
LDMIA r0,  {r1-r3}          ; r1 := mem32[r0]
                            ; r2 := mem32[r0+4]
                            ; r2 := mem32[r0+8]
```

# SWAP

```
SWP r0, r1, [r2]  ;Load R0 with the word addressed by R1 and store
                   r1 at r2

SWPB r2, r3, [r4] ;Load R2 with the byte addresse by r4 and store
                  bits 0 to 7 of R3 at R4

SWPEQ r0, r0, [r1] ;Conditionally swap (if Z flag is set) the
                    contents of the word addressed by R1 with r0
```

# CONDITIONAL BRANCHES

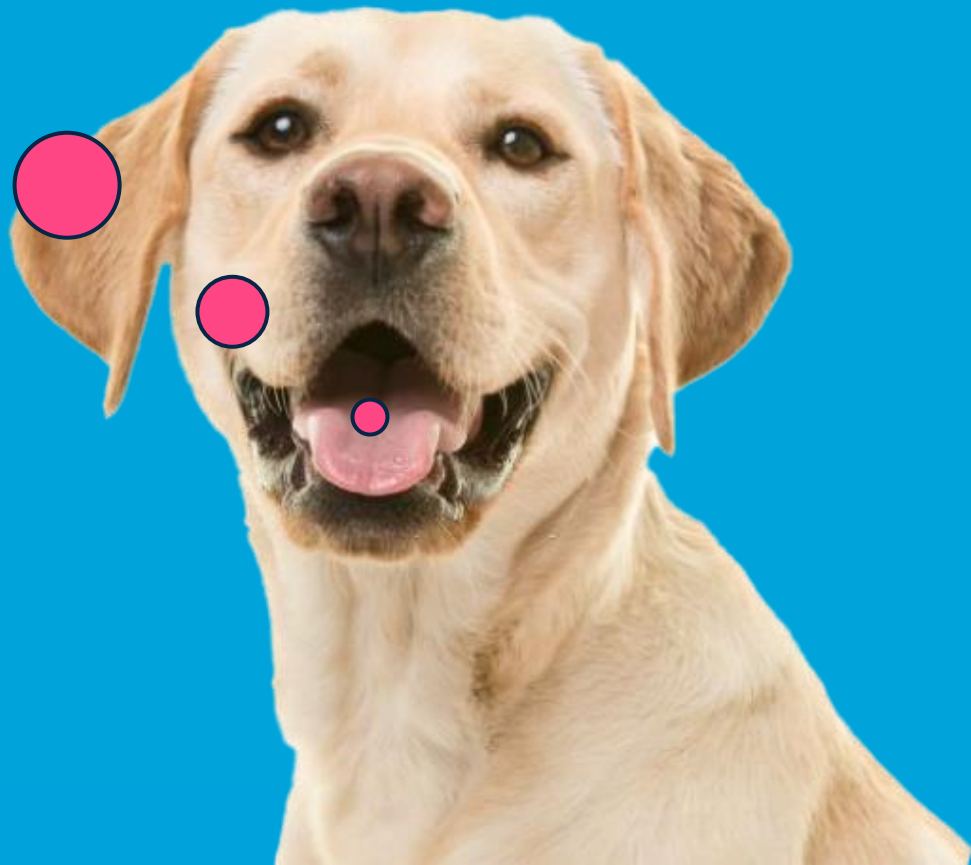
| Branch | | Interpretation | Normal uses | |
|---|---|---|---|---|
| B BAL | | Unconditional<br>Always | Always take this branch<br>Always take this branch | |
| BEQ | Z=1 | Equal | Comparison equal or zero result | |
| BNE | Z=0 | Not equal | Comparison not equal or non-zero result | |
| BPL | N=0 | Plus | Result positive or zero | |
| BMI | N=1 | Minus | Result minus or negative | |
| BCC<br>BLO | C=0 | Carry clear<br>Lower | Arithmetic operation did not give carry-out<br>Unsigned comparison gave lower | |
| BCS<br>BHS | C=1 | Carry set Higher<br>or same | Arithmetic operation gave carry-out<br>Unsigned comparison gave higher or same | |
| BVC | V=0 | Overflow clear | Signed integer operation; no overflow occurred | |
| BVS | V=1 | Overflow set | Signed integer operation; overflow occurred | |
| BGT | | Greater than | Signed integer comparison gave greater than | Z=0 & N=V |
| BGE | N=V | Greater or equal | Signed integer comparison gave greater or equal | |
| BLT | N!=V | Less than | Signed integer comparison gave less than | |
| BLE | | Less or equal | Signed integer comparison gave less than or equal | Z=1 & N!=V |
| BHI | | Higher | Unsigned comparison gave higher | C=1 & Z=0 |
| BLS | | Lower or same | Unsigned comparison gave lower or same | C=0 & Z=1 |

SMILIES
reSilient coMputer archItectures and LIfE Sciences

Politecnico di Torino
Department of Control and Computer Engineering

DAUIN

QUESTIONS?



# THANK YOU!