Marcello TRAIOLA
traiola@lirmm.fr
LIRMM - Campus St.Priest, Bat.4
161, rue Ada - 34095 Montpellier

You will find course materials in the Course page of the website:
http://www.lirmm.fr/~traiola/courses
Course Planning

1. Introduction to Advanced RISC Machines
   1.1 ARM Cortex-M4 processor

2. Assembler language: basics

3. Assembler language: functions

4. STM32 peripherals
Von Neuman Architecture

Three main parts:
- Central Processing Unit (CPU)
- Memory
- Peripherals
Von Neuman Architecture

Three main parts:
- Central Processing Unit (CPU)
- Memory
- Peripherals
**Instruction set architecture (ISA):** is the set of processor design techniques used to implement the instruction work flow on hardware. In more practical words, ISA tells you which commands the processor understands, how many registers it has and basically how it processes your program instructions.

We can consider the ISA as an interface between the software and the hardware of a computer system.
RISC VS CISC

- **CISC**: A complex instruction set computer, is a computer where single instructions can execute several low-level operations (such as a load from memory, an arithmetic operation, and a memory store) or are capable of multi-step operations or addressing modes within single instructions.

- **RISC**: A reduced instruction set computer is a computer which only use simple instructions that can be divide into multiple instructions which perform low-level operation within single clock cycle.
Let us take as example the multiplication of two numbers:
\[ A = A \times B; \rightarrow \text{C statement} \]

- **CISC solution (assembly):**
  ```
  MULT A, B
  ```

- **Advantages:**
  - little work for the compiler;
  - few lines of code;
  - little RAM to store instructions.

- **RISC solution (assembly):**
  ```
  LOAD R1, A
  LOAD R2, B
  PROD R3, A, B
  STORE R3, A
  ```

- **Advantages:**
  - one clock cycle per instruction;
  - approximately the same execution time as the CISC;
  - reduced instructions \(\rightarrow\) less transistors;
  - pipeline oriented.
Advanced RISC Machines

Advanced RISC Machines (ARM)

- **Cortex®-M processors**: MCU + DSP
  - Smallest footprint / lowest power

- **Cortex®-R processors**: RTOS
  - Highest performance / real-time

- **Cortex®-A processors**: Rich OS
  - Highest performance
Advanced RISC Machines

ARM Cortex-M

ARM Cortex-M Product Line

Consistent 32 bit processor architecture across all applications

Lowest cost
Low power

Outstanding energy efficiency

Digital Signal Control Processor with DSP Accelerated SIMD Floating point

'8/16-bit' Traditional application space

'16/32-bit' Traditional application space
Case Study

STM32 Cortex-M4
Processor Modes:
- **Thread mode**: Used to execute application software (default mode)
- **Handler mode**: Used to handle exceptions (returns to Thread mode when it has finished exception processing)

Privilege Levels:
- **Unprivileged**: Some Limitations (not all the registers/instructions are accessible)
- **Privileged**: all instructions and all resources are accessible.

<table>
<thead>
<tr>
<th>Processor mode</th>
<th>Used to execute</th>
<th>Privilege level for software execution</th>
<th>Stack used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
<td>Applications</td>
<td>Privileged or unprivileged(^1)</td>
<td>Main stack or process stack(^1)</td>
</tr>
<tr>
<td>Handler</td>
<td>Exception handlers</td>
<td>Always privileged</td>
<td>Main stack</td>
</tr>
</tbody>
</table>
Registers Bank

- **Low registers**
  - R0
  - R1
  - R2
  - R3
  - R4
  - R5
  - R6
  - R7
  - R8
  - R9

- **Stack Pointer**
  - SP (R13)

- **High registers**
  - R10
  - R11
  - R12

- **General-purpose registers**
  - LR (R14)
  - PC (R15)

- **Special registers**
  - PSR
  - PRIMASK
  - FAULTMASK
  - BASEPRI
  - CONTROL

- **Exception mask registers**
  - CONTROL register
Registers Bank

- General-purpose registers (R0-R12):
  - 32-bit general-purpose registers for data operations
  - Only low level registers can use immediate values

- Stack pointer, SP or R13:
  - The stack is typically used to store temporary values. An assembler function has to store on the stack the contents of any registers it is going to use. Just before finishing, the function has to restore the register values from the stack.

- Link Register, LR or R14
  - It stores the return address for subroutines, function calls, and exceptions.

- Program counter, PC or R15
  - It contains the current program address

- Control Register: controls the privilege level for software execution when the processor is in Thread mode
Registers Bank

Program status register

It contains the current state of the condition flags from previous instruction executions.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
</table>
| Bit 31 | N: Negative or less than flag:  
0: Operation result was positive, zero, greater than, or equal  
1: Operation result was negative or less than. |
| Bit 30 | Z: Zero flag:  
0: Operation result was not zero  
1: Operation result was zero. |
| Bit 29 | C: Carry or borrow flag:  
0: Add operation did not result in a carry bit or subtract operation resulted in a borrow bit  
1: Add operation resulted in a carry bit or subtract operation did not result in a borrow bit. |
| Bit 28 | V: Overflow flag:  
0: Operation did not result in an overflow  
1: Operation resulted in an overflow. |
| Bit 27 | Q: DSP overflow and saturation flag: Sticky saturation flag.  
0: Indicates that saturation has not occurred since reset or since the bit was last cleared to zero  
1: Indicates when an SSAT or USAT instruction results in saturation, or indicates a DSP overflow.  
This bit is cleared to zero by software using an MRS instruction. |
| Bits 26:20 | Reserved. |
| Bits 19:16 | GE[3:0]: Greater than or Equal flags. |
| Bits 15:0 | Reserved. |
The processor has a fixed memory map that provides up to 4 GB of addressable memory.

- The 4GB are divided into sub-regions:
  - Each region is given for a particular usage
  - Easy for portability

- Data Type:
  - 32-bit words
  - 16-bit halfwords
  - 8-bit bytes
- **Code**: program instructions;
- **SRAM**: to store data (example global variables);
- **Peripheral**: on-chip peripheral like AHB, APB peripherals
- **External RAM**: mainly used for DDR or FLASH external memory
- **External device**: example SD card or USB port
- **Private Peripheral**: for interacting with internal peripherals (example interrupt vector table)
```c
int main () {
    int a=0;
    int b=0;
    for ( a = 1; a < 4; a = a + 1 ){
        b+=a;
    }
    return 0;
}
```

What does the program do?

Once compiled, what happens?