



ARM Instruction Set Introduction

Main features of the ARM Instruction Set

- Every instruction can be conditionally executed.
- A load/store architecture
 - Data processing instructions act only on registers
 - Three operand format
 - Combined ALU and shifter for high speed bit manipulation
 - Specific memory access instructions with powerful auto-indexing addressing modes.
 - 32 bit and 8 bit data types
 - Flexible multiple register load and store instructions
- Instruction set extension via coprocessors.

Cortex-M3 Register Set

- 18 registers - 32-bit wide
- The following data types are supported:
 - byte: 8 bits
 - halfword: 16 bits
 - word: 32 bits

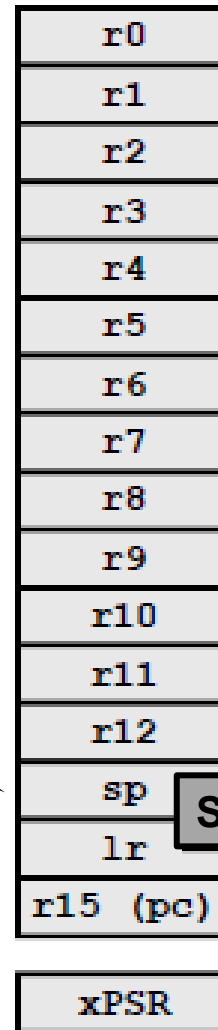
di cui
3 inutilizzabili

R13 used as stack pointer

R14 used as link register

R15 used as program counter

Main



A replica of the stack pointer register (PSP) is available to be used in thread mode

Process

Sp_proc

Sp_main

The Program Counter (R15)

- When the processor is executing in ARM state:
 - All instructions are 32 bits in length
 - All instructions must be word aligned
 - Therefore the PC value is stored in bits [31:2] with bits [1:0] equal to zero (as instruction cannot be halfword or byte aligned).
- Differently from other processors like the 80x86 family, the ARM permits to directly write in the PC, which is mapped over register R15.
- Example

The Link Register (R14)

- R14 is used as the subroutine link register (LR) and stores the return address when Branch with Link operations are performed, calculated from the PC.
- Thus to return from a linked branch

- `MOV r15, r14`

or

- `MOV pc, lr`

(11230000)

→ Si possono anche usare questi altri



Così si sposta il valore del link register nel PC

The Stack Pointer (R13)

- R13 is used as the **Stack Pointer** and it is **autonomously updated**
 - At boot time its value is retrieved from the vector table
 - During program execution if a stack oriented instruction is executed.

Initial value
hardware loaded
from IVT

Exception Type	Index	Vector Address
(Top of Stack)	0	0x00000000
Reset	1	0x00000004
NMI	2	0x00000008
Hard fault	3	0x0000000C
Memory management fault	4	0x00000010
Bus fault	5	0x00000014
Usage fault	6	0x00000018
SVcall	11	0x0000002C
Debug monitor	12	0x00000030
PendSV	14	0x00000038
SysTick	15	0x0000003C
Interrupts	≥16	≥0x00000040

Processor operating modes and levels

- **Two operating modes:**
 - thread mode: on reset or after an exception
 - handler mode: when an exception occurs
- **Two access levels:**
 - user level: limited access to resources
 - privileged level: access to all resources
- **Handler mode is always privileged.**

System Registers

- **BASEPRI:** Base Priority Mask register
- **PRIMASK:** PRIMASK is a 1-bit-wide interrupt mask register. When set, it blocks all interrupts apart from the non-maskable interrupt (NMI) and the hard fault exception. The PRIMASK prevents activation of all exceptions with configurable priority.
- **FAULTMASK:** FAULTMASK prevents activation of all exceptions except for the **Non-Maskable Interrupt (NMI)**.
- **CONTROL:** The CONTROL register controls the stack used and the privilege level for software execution when the processor is in thread mode and, if implemented, indicates whether the FPU state is active.

CONTROL Register

- This register uses the following bits:
- **CONTROL[2]** [only Cortex-M4 and Cortex-M7]
 - =0 FPU not active
 - =1 FPU active
- **CONTROL[1]**
 - =0 In handler mode - MSP is selected. No alternate stack possible for handler mode.
 - =0 In thread mode - Default stack pointer MSP is used.
 - =1 In thread mode - Alternate stack pointer PSP is used.
- **CONTROL[0]** [not Cortex-M0]
 - =0 In thread mode and privileged state.
 - =1 In thread mode and user state.

The Program Status Registers (PSR)

31					25					20					15					10					5					0				
N	Z	C	V	Q						GE																								
					IT		T								ICI/IT																			
																		ISRNUM																

Logical Instruction

Arithmetic Instruction

Flag

Negative
(N='1')

No meaning

Bit 31 of the result has been set
Indicates a negative number in
signed operations

Zero
(Z='1')

Result is all zeroes

Result of operation was zero

Carry
(C='1')

After Shift operation
'1' was left in carry flag

Result was greater than 32 bits

oVerflow
(V='1')

No meaning

Result was greater than 31 bits
Indicates a possible corruption of
the sign bit in signed
numbers

Condition Flags

	Logical Instruction	Arithmetic Instruction
<u>Flag</u>		
Negative (N='1')	No meaning	Bit 31 of the result has been set Indicates a negative number in signed operations
Zero (Z='1')	Result is all zeroes	Result of operation was zero
Carry (C='1')	After Shift operation '1' was left in carry flag	Result was greater than 32 bits
oVerflow (V='1')	No meaning	Result was greater than 31 bits Indicates a possible corruption of the sign bit in signed numbers

Conditional Execution

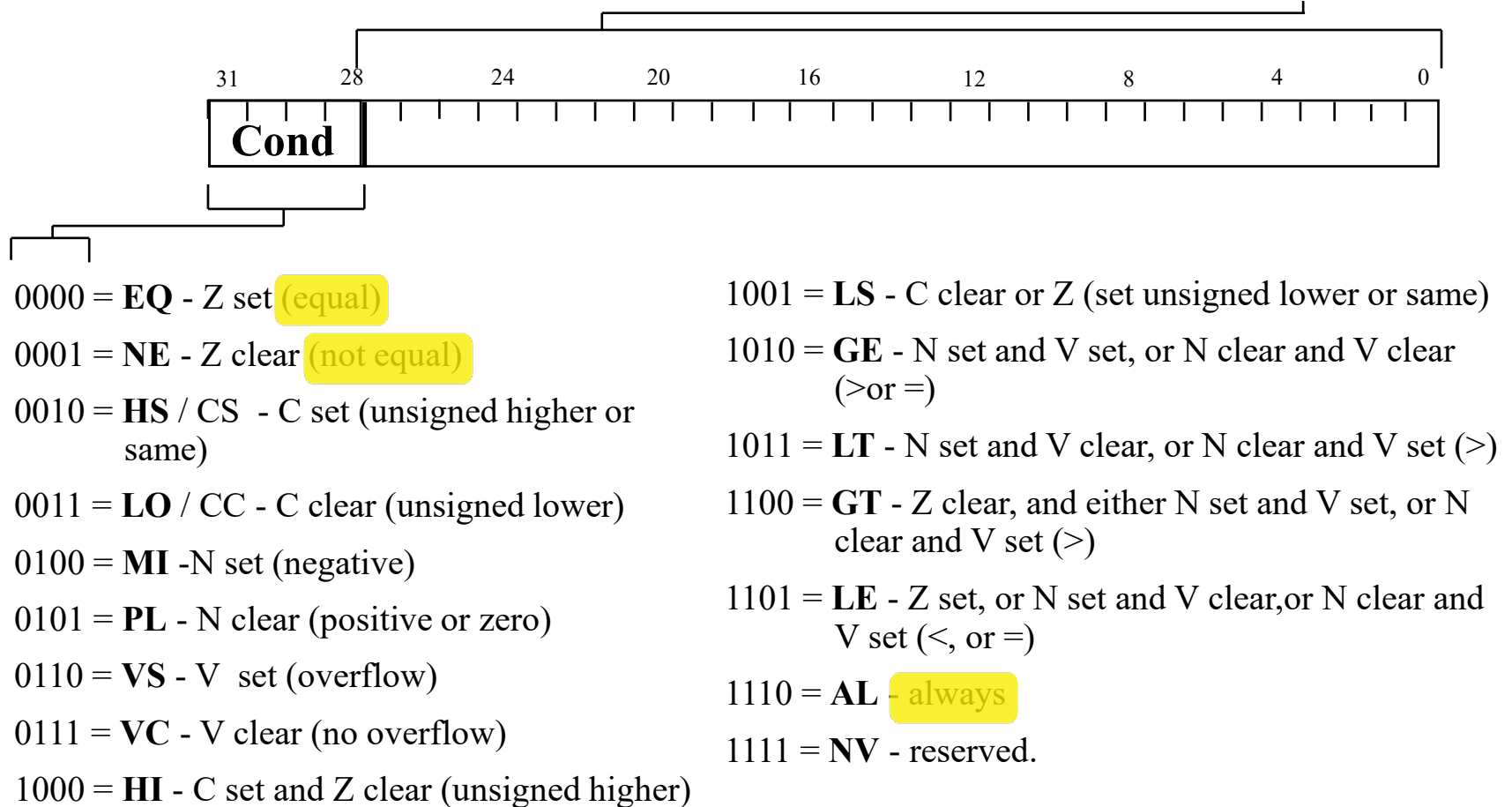
- Most instruction sets only allow branches to be executed conditionally.
- However by reusing the condition evaluation hardware, ARM effectively increases number of instructions.
 - All instructions contain a condition field which determines whether the CPU will execute them.
 - Non-executed instructions soak up 1 cycle.
 - Still have to complete cycle so as to allow fetching and decoding of following instructions.

Conditional Execution (II)


- This removes the need for many branches, which stall the pipeline (3 cycles to refill).
 - Allows very dense in-line code, without branches.
 - The Time penalty of not executing several conditional instructions is frequently less than overhead of the branch or subroutine call that would otherwise be needed.

The Condition Field

Opcode and operands



Using and updating the Condition Field

- To execute an instruction conditionally, simply postfix it with the appropriate condition:
 - For example an add instruction takes the form:
 - `ADD r0,r1,r2 ; r0 = r1 + r2 (ADDAL)`
 - To execute this only if the zero flag is set:
 - `ADDEQ r0,r1,r2 ; If zero flag set then...`
 `; ... r0 = r1 + r2`
- By default, data processing operations do not affect the condition flags (apart from the comparisons where this is the only effect).

Using and updating the Condition Field

- To cause the condition flags to be updated, the S bit of the instruction needs to be set by postfixing the instruction (and any condition code) with an “S”.

```
ADDS r0,r1,r2    ; r0 = r1 + r2  
; ... and set flags
```

An example

- If $R4 - R3 == 0$ then $R0 = R1$
- Else $R0 = R2$

SUBS	R4, R4, R3
MOVEQ	R0, R1
MOVNE	R0, R2

move if equal
move if not equal

↓
chi le use all'esame +2 punti