ARM: Advanced RISC Machine

- 1990: fondata da Acorn computers, Apple Computer e VLSI technology
- Oggi detiene il 75% del mercato di processori a 32 bit per embedded
- Non produce CPU ma licenzia i suoi design
- Fornisce il supporto necessario per sviluppare un sistema completo
- Il risultato è una serie di processori multivendor compatibili sia come linguaggio che come toolchain

Partner ARM

ATAP Partners

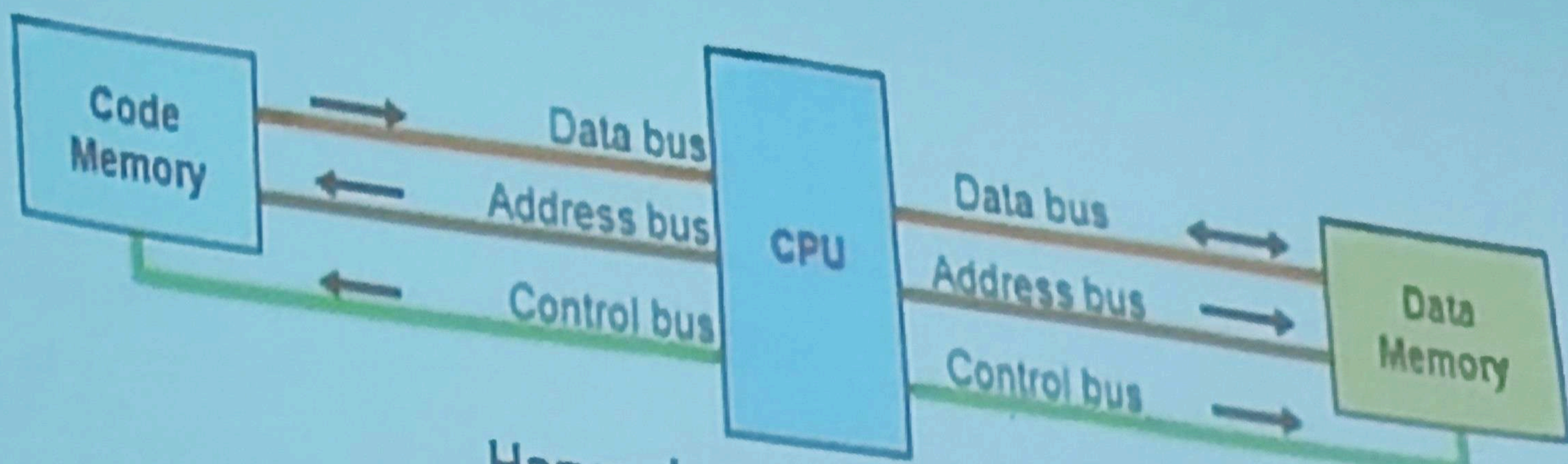
DNP
Infinite Technology Corporation
WinFINITY
BARCO SILEX
SOTA
STEPMIND
Think
Macrotech
SIEMENS
NSW
EPI
ASHLING
CoWare
SIDSA
ARCADIA
COMET SYSTEMS
TOGITECH
virtio
YOKOGWA
ADS
Tektronix
INNOVEDA
Computex
TOPPAN
HOYA
SCIWORX
FARADAY
MAZ
Axis
WindRiver
Sophia Systems
QUALCOMM
ZILOG
GOODRICH
OKI
MOTOROLA
FUJITSU
ERICSSON
ADMtek
NEC
TOSHIBA
Intersil
Verisity
Aptix
parthus
TALITY
MICRONAS
PHILIPS
AMI
Altant
SYNOPSYS
FLEXTRONICS
EPSON
UMC
ALCATEL
TIKOS
SANYO
Charmed
LAUTERBACH
cadence
TERMINAL SYSTEMS
realos
ARM
In Partnership
AKM
LINEO
agere
Panasonic
HYUNDAI
ZEEVO Intl
internichе
Microsoft
ACCESS
SHARP
YAMAHA
IBM
Virata
SONY
EMBLAZE
GEOWORKS
US Software
Triscend
INTERTRUST
KADAK
Tao Systems
Packet Video
JAVA
OSE
ERICSSON
Microsoft
AXE
symbian
Bluetooth
WindRiver
MICROWARE
Sun
FINOTEAM
Dolby
Symmetricom
CDS
LINUXWORKS
CMX
ASAHI CHEMICAL INDUSTRY CO.,LTD.

Tools Partners

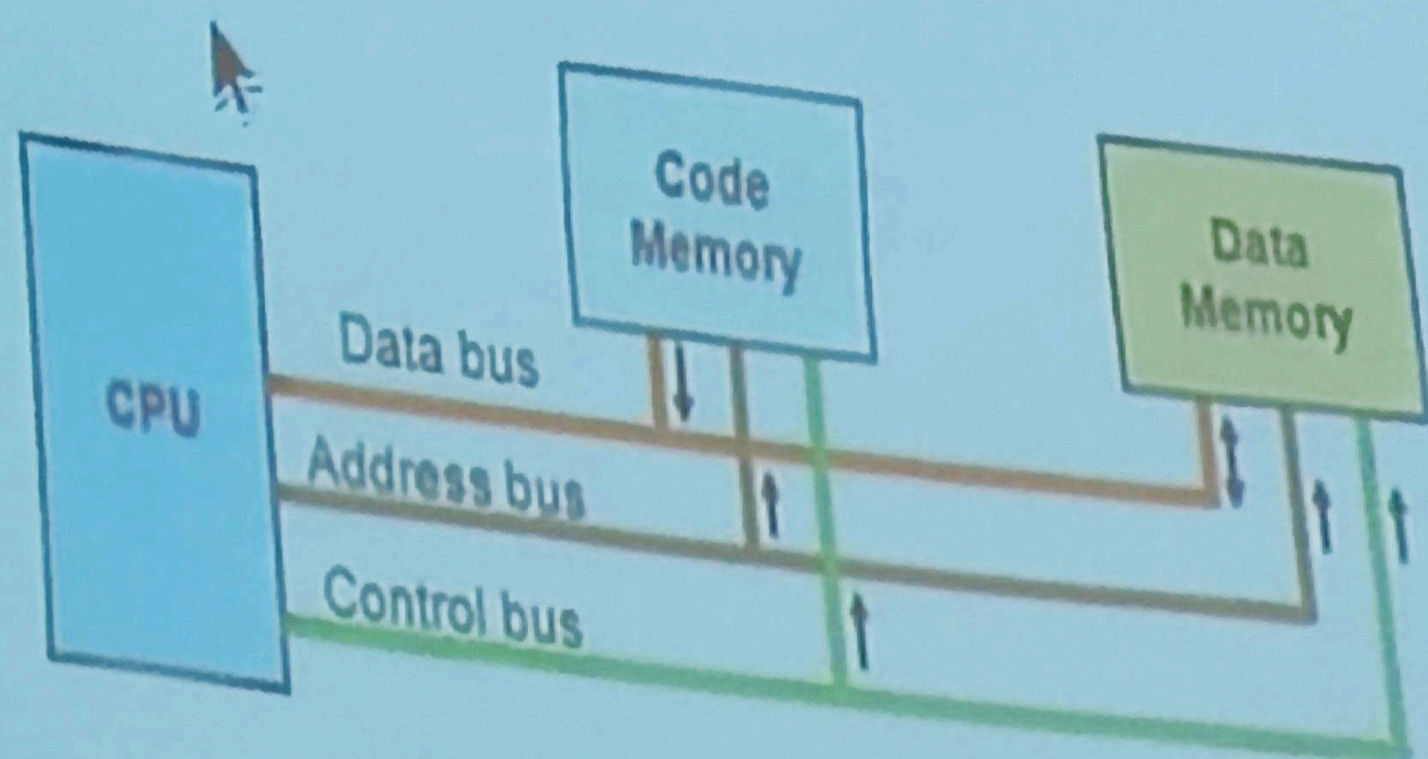
RTOS Partners

Software Partners

Architettura ARM

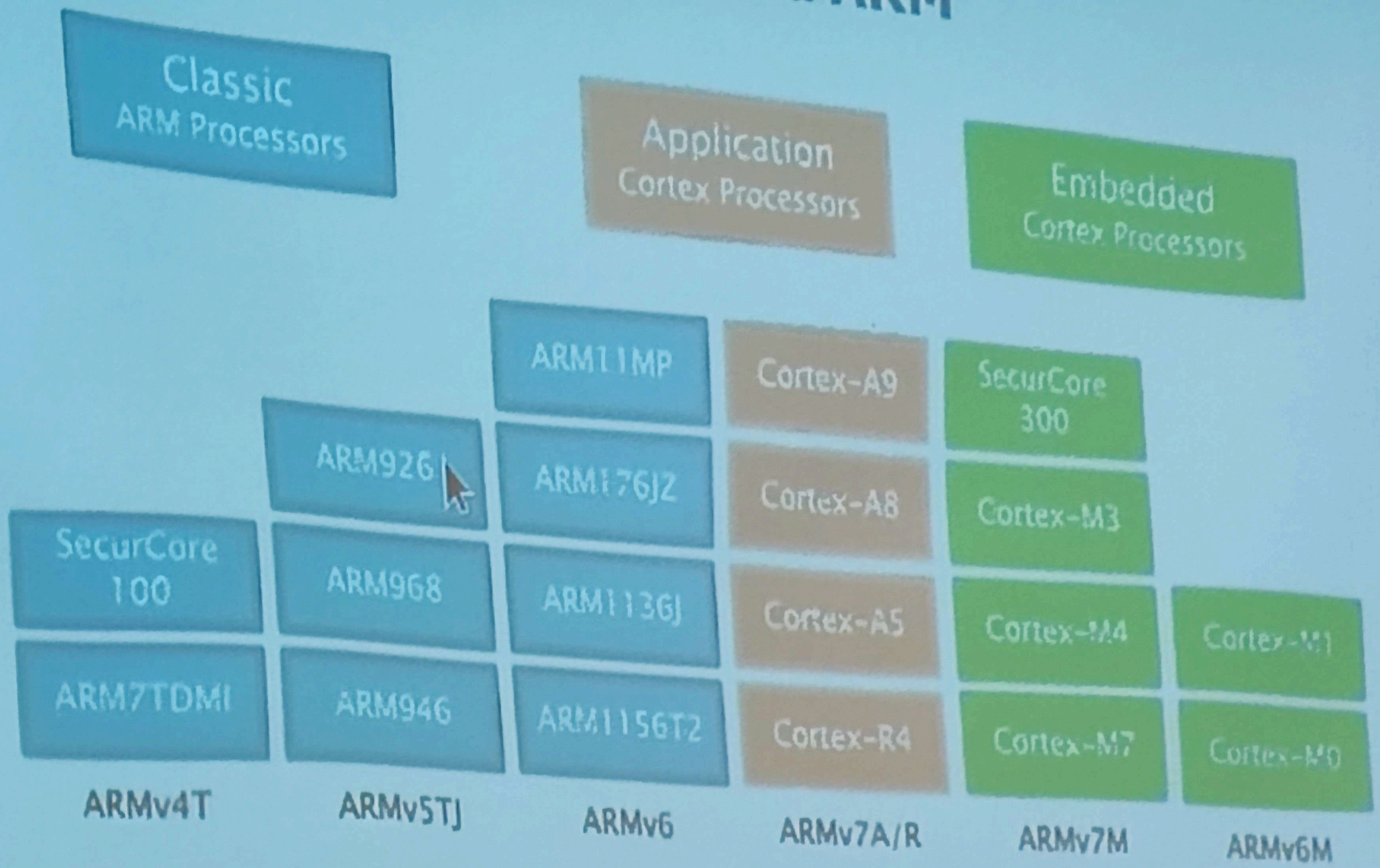


Harvard architecture



Von Neumann architecture

Architettura ARM



Architettura ARM

ARM CORE	FEATURE
ARM v1 (obsoleto)	Istruzioni a 26 bit, nessun moltiplicatore né coprocessore
ARM v2 (obsoleto)	Risultato a 32 bit, aggiunto coprocessore
ARM v3 (obsoleto)	Istruzioni a 32 bit
ARM v4 ARM v4T	Aggiunte istruzioni con segno, aggiunta Thumb mode
ARM v5TEJ	Aggiunto supporto per algoritmi DSP e Jazelle
ARM v6	Supporto per SIMD, aggiunto Thumb2. Aumentato supporto per virtualizzazione con TrustZone.
ARM v6M	Creato per alte performance a basso costo.
ARM v7	Ottimizzato THUMB2, migliorate operazioni floating point.

CORTEX-A	CORTEX-R	CORTEX-M
MMU e NEON (opzionale)	Profilo realtime. MPU.	Pensato per processare velocemente interrupt.

Instruction set ARM

- **Instruction set a 32 bit: ARM**
  - Architettura load/store in cui ogni istruzione è condizionale
- **Instruction set a 16 bit: Thumb**
  - Solo le istruzioni di branch sono condizionali e solo metà dei registri sono usati
- **Instruction set a 8 bit: Jazelle**
  - Permette esecuzione di Java byte code in hardware

Caratteristiche del linguaggio ARM

- ogni istruzione può essere preventivamente condizionata in base al registro di stato
- architettura load/store
- non esistono istruzioni di shift elemntari se non innestate all'interno di altre istruzioni di elaborazione (aritmetiche, logiche,...)
- le operazioni aritmetiche/logiche possono eseguire anche shift in una sola istruzione.
- non esiste una istruzione di divisione, questo compito è delegato al coprocesssore
- indirizzamento con autoincremento e autodecremento per ottimizzare i cicli nei programmi
- istruzioni di load/store multiple per massimizzare il throughtput dei dati

Terminologia ARM: le eccezioni

- Le eccezioni sono eventi asincroni che si veri cano a fronte di una interruzione:
  - hardware,
  - software (SoftWare Interrupt, chiamata anche SWI),
  - errore (es. istruzione non riconosciuta o un malfunzionamento durante la fase di prefetch delle istruzioni o nella lettura dei dati in memoria
  - reset

I processori ARM sono progettati per essere veloci e ridurre al minimo il tempo necessario al cambio di contesto necessario nella gestione delle eccezioni o nelle operazioni privilegiate.

Modalità di funzionamento

L'architettura ARM prevede le seguenti modalità di funzionamento:

- **User**: modalità **NON privilegiata** sotto cui girano la maggior parte dei task;
- **FIQ**: quando viene sollevato un interrupt ad alta priorità
- **IRQ**: quando viene sollevato un interrupt a bassa priorità
- **Supervisor**: al reset o a seguito di un'istruzione di software interrupt (SWI)
- **Undef**: usato per gestire violazioni all'accesso di memoria
- **System**: modalità privilegiata che usa gli stessi registri dell'user mode

Priorità delle eccezioni

Priorità	Livello	Eccezione	Causa
Max	1	Reset	Reset hardware o software
	2	Abort	Abort dati durante l'accesso in memoria
	3	FIQ	Interrupt ad altra priorità, *Fast Interrupt reQuest*
	4	IRQ	Interrupt a bassa priorità, *Interrupt ReQuest*
	5	Abort	Prefetch Abort durante la fase di fetch dell'istruzione
Min	6	Undefined	Istruzione non definita e software interrupt (SWI)

Architettura ARM

L'architettura ARM prevede la seguente organizzazione:

- 16 registri denominati da R0 a R15;
- Un registro di stato denominato Current Program Status Register (CPSR);
- 20 registri non accessibili nel modo utente/sistema.

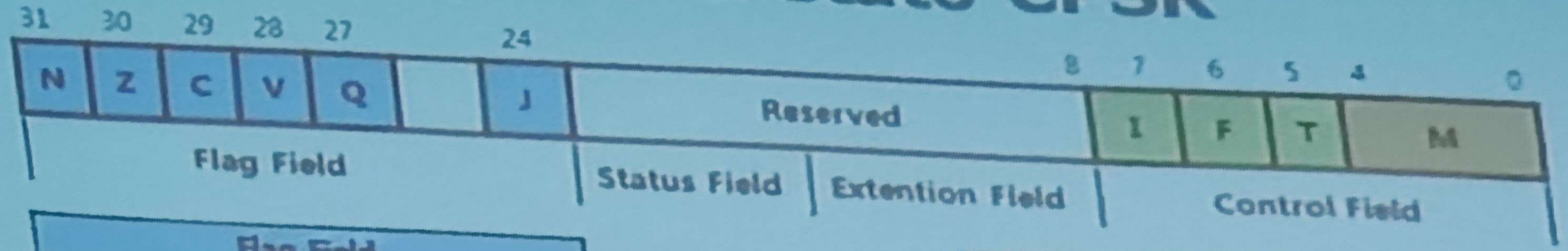
R0	A1
R1	A2
R2	A3
R3	A4
R4	V1
R5	V2
R6	V3
R7	V4 WR
R8	V5
R9	V6 SB
R10	V7 SL
R11	V8 FP
R12	IP
R13	SP
R14	LR
R15	PC

Registri

I principali registri dei processori ARM sono 16, nominati da R0 a R15, e si distinguono in base al loro utilizzo:

- R0 - R12 sono i registri di uso generale;
- R13 (o SP) è generalmente il registro dello **Stack Pointer**, cioè l'indirizzo di memoria dello stack. Non è però obbligatorio il suo utilizzo come Stack pointer; viene usato solo per convenzione;
- R14 (o LR) è il registro del **Link Register**, nel quale viene salvato l'indirizzo di ritorno nel momento in cui viene chiamata una procedura;
- R15 (o PC) è il **Program Counter**, il registro contiene l'indirizzo di memoria della prossima istruzione da eseguire.

Registro di stato CPSR

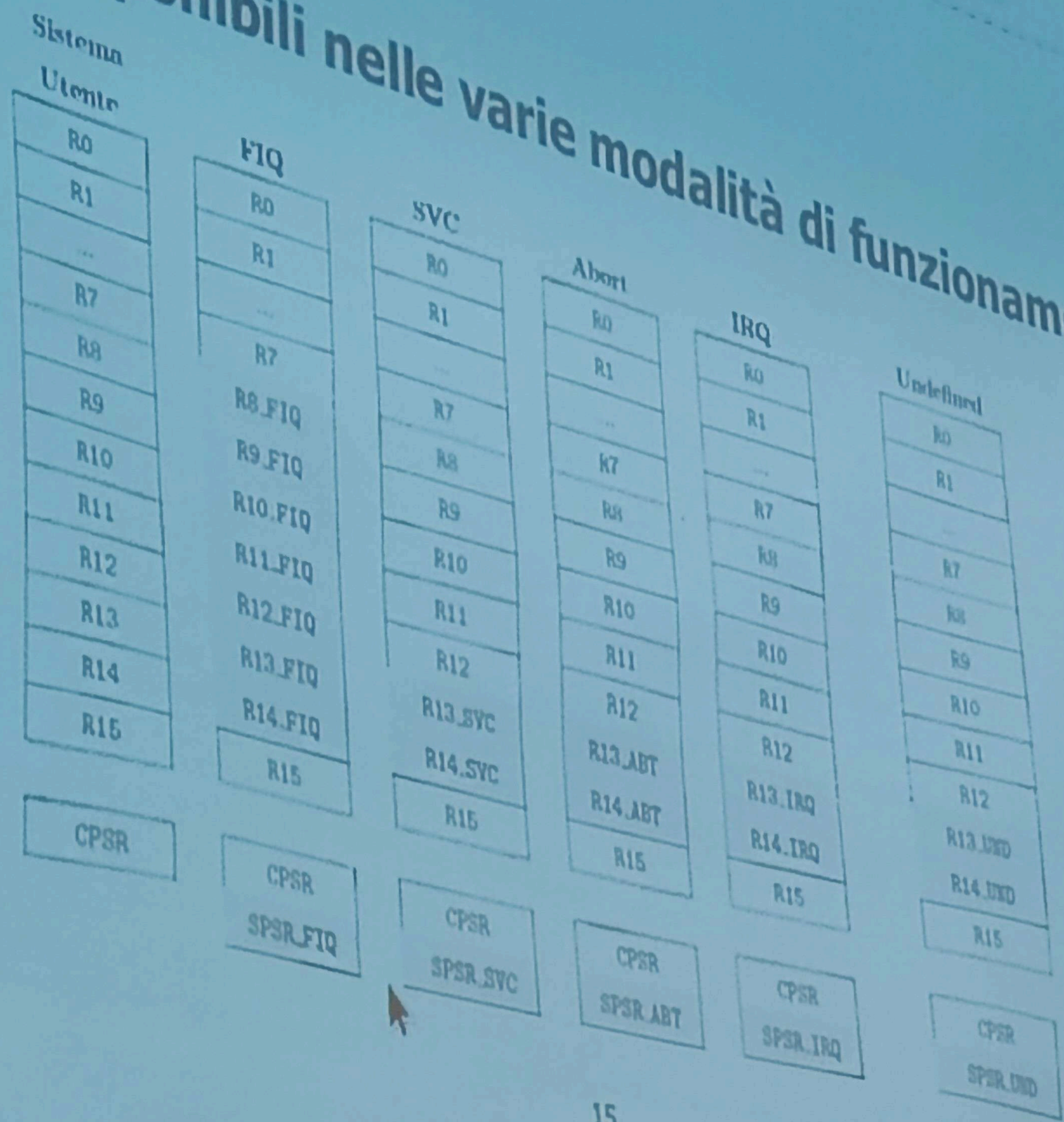


Flag Field	
N	Negative result from ALU
Z	Zero result from ALU
C	ALU operation caused Carry
V	ALU operation oVerflowed
Q	ALU operation saturated
J	Java Byte Code Execution

Control bits	
I	1: disables IRQ
F	1: disables FIQ
T	1: Thumb, 0: ARM

Mode bits M[4:0]	
0b10000	User
0b11111	System
0b10001	FIQ
0b10010	IRQ
0b10011	SVC(Supervisor)
0b10111	Abort
0b11011	Undefined

Registri disponibili nelle varie modalità di funzionamento



Pre-condizioni

La condizione determina l'esecuzione dell'istruzione
La pre-condizione si può esprimere attraverso il valore che assumono i flag (Z, C, N, V) del registro di stato CPSR.

PreCond	PreCond	Significato	Segno	Espr. Logica
0000	EQ	Uguale =		Z
0001	NE	Diverso ≠		$\overline{Z}$
0010	CS/HS	Riporto | ≥	No	C
0011	CC/LO	No riporto | <	No	$\overline{C}$
0100	MI	Meno | < 0		N
0101	PL	Più | ≥ 0		$\overline{N}$
0110	VS	overflow		V
0111	VC	No overflow		$\overline{V}$
1000	HI	>		$C \cdot \overline{Z}$
1001	LS	≤	No	$\overline{C} + Z$
1010	GE	≥	No	$\overline{N \oplus V}$
1011	LT	<	✓	$N \oplus V$
1100	GT	.	✓	$Z \cdot \overline{N \oplus V}$
1101	LE	≤	✓	$Z + (N \oplus V)$
1111	AL	Sempre	✓	1
1101	NV	Mai		0

Architettura interna del processore

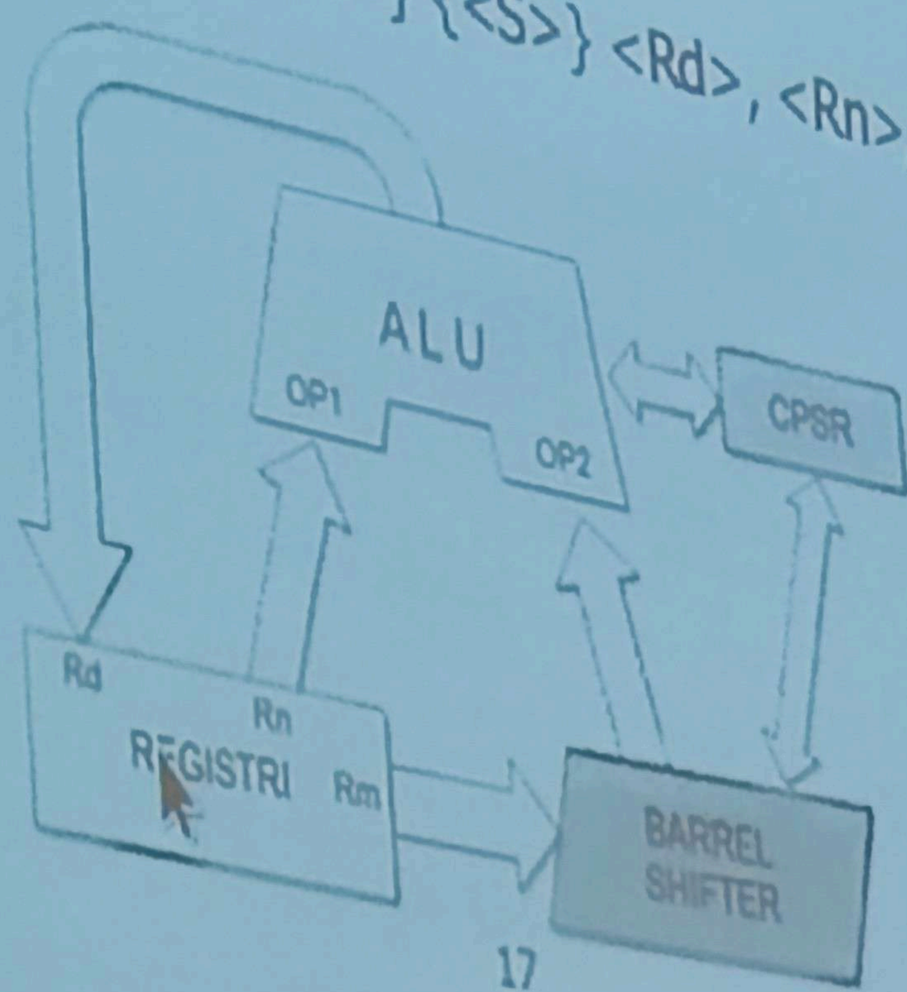
L'organizzazione interna dell'ARM prevede l'utilizzo di tre registri:

Rd ⇨ Registro destinazione dove verrà scritto il risultato

Rn ⇨ Registro sorgente che contiene il primo operando ed è utilizzato direttamente

Rm⇨ Registro sorgente che contiene il secondo operando ed è utilizzato dopo il passaggio nel barrel shifter

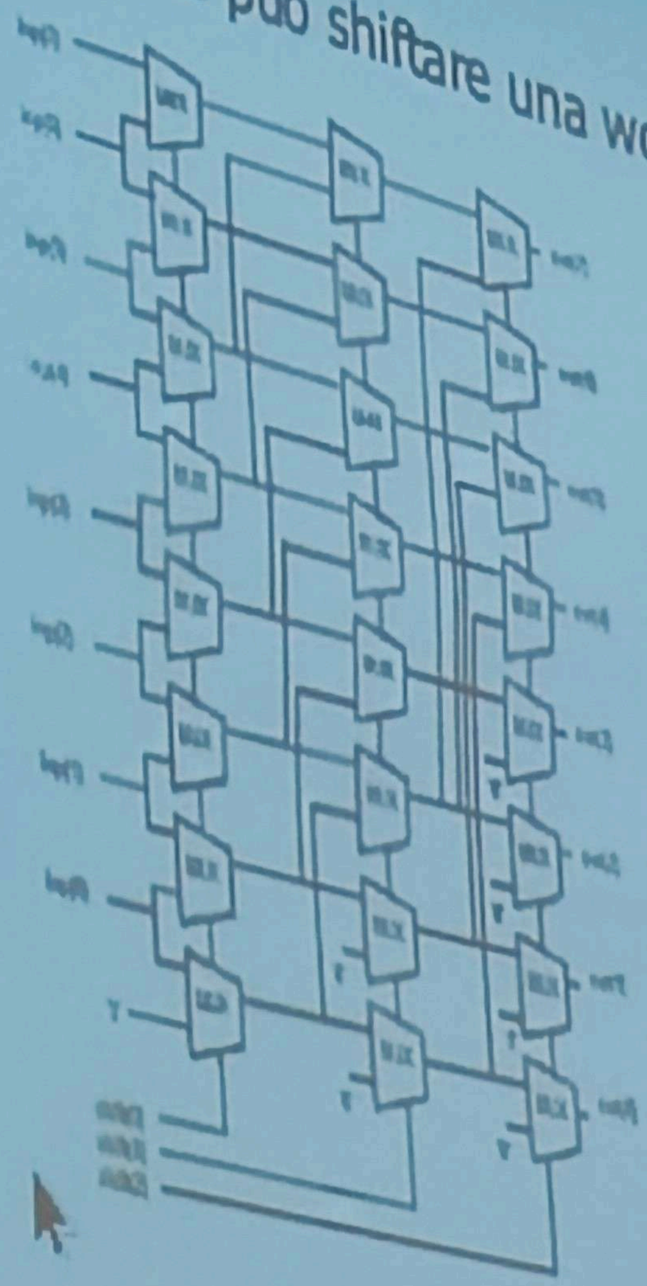
<MNEM>{<PreCond>}{<S>}<Rd>,<Rn>,OP2



65,3%

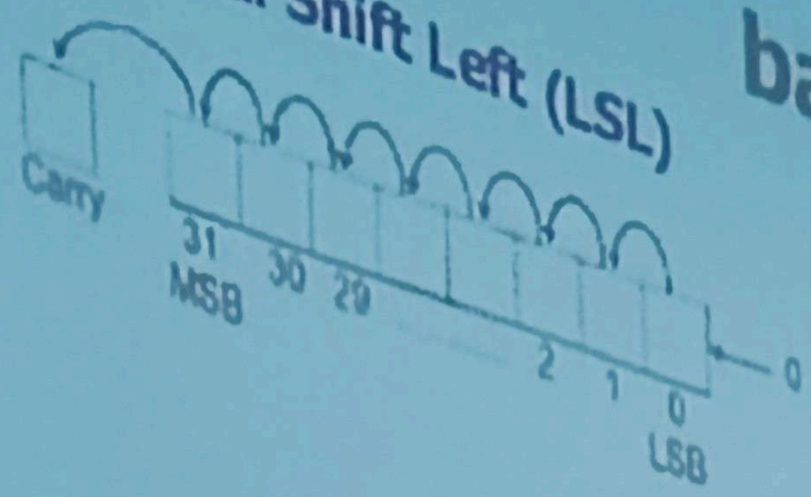
barrel shifter

Un barrel shifter è un circuito digitale che può shiftare una word di un numero specificato di bit in un solo ciclo di clock

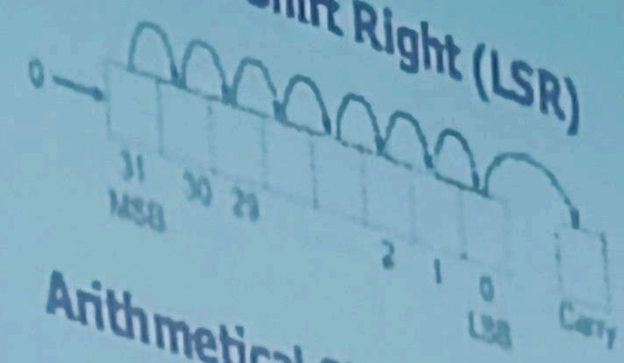


barrel shifter

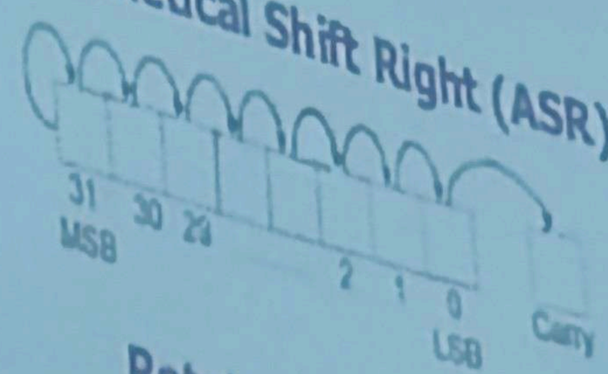
Logical Shift Left (LSL)



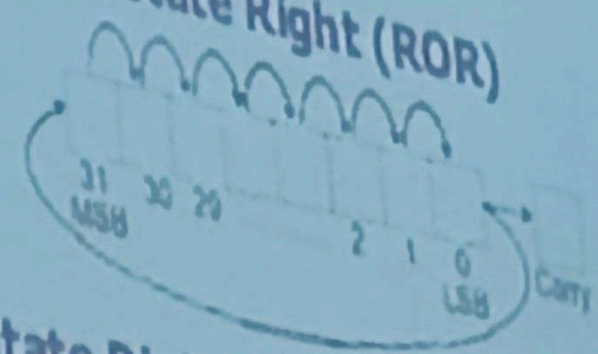
Logical Shift Right (LSR)



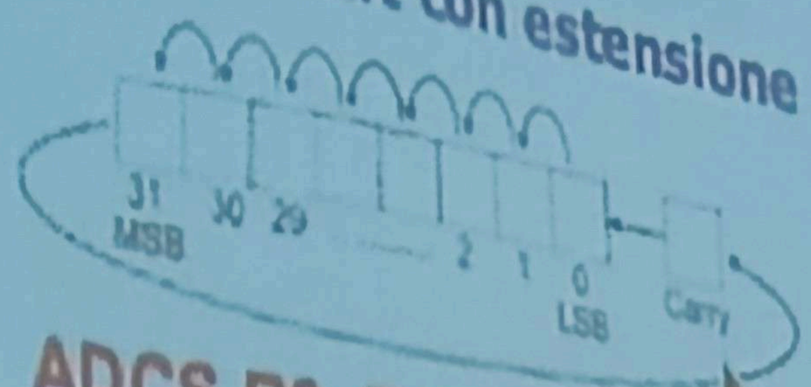
Arithmetical Shift Right (ASR)



Rotate Right (ROR)

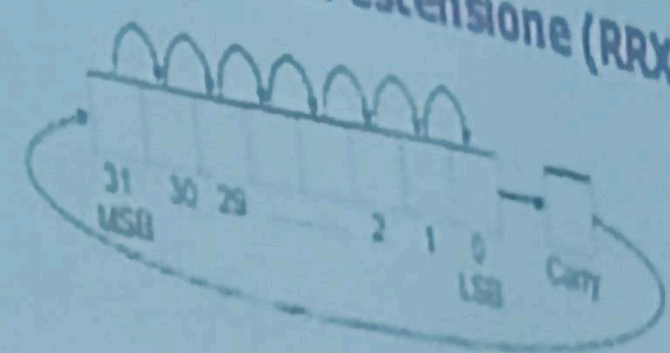


Rotate Left con estensione




```
ADCS R0, R0, R0 // RLX R0
```

Rotate Right con estensione (RRX)



65,3%

Tipi di dati

I tipi di dati sulle quali le istruzioni assembly ARM operano sono:

- Byte 1-byte, 8-bit
- Half word 2-byte, 16-bit
- Full word o word 4-byte, 32-bit

Il linguaggio assemblativo può interpretare questi tipi di dati ed esprimere diversi formati quali:

- Numeri interi espressi in diverse basi
- Numeri in virgola mobile
- Valori booleani
- Singoli caratteri
- Stringhe di caratteri

Modalità di Indirizzamento

L'architettura ARM individua 5 modalità di indirizzamento:

- **Modo 1**: Indirizzamento nelle istruzioni di elaborazione dati,
- **Modo 2**: Indirizzamento nelle load/store su word o byte senza segno;
- **Modo 3**: Indirizzamento nelle load/store su byte, word o doubleword;
- **Modo 4**: Indirizzamento nelle load/store multiple;
- **Modo 5:** Indirizzamento nelle load/store con il coinvolgimento dei coprocessori.

Modalità di Indirizzamento

Tutte le istruzioni di load/store utilizzano un registro base e un offset che, sommato o sottratto al registro base, restituisce l'indirizzo di memoria ove effettuare l'operazione.

Per le operazioni di load/store sono possibili le seguenti modalità di indirizzamento:

- con **offset immediato**, l'offset è un valore senza segno specificato nell'istruzione;
- con **offset a registro**, l'offset è un valore contenuto in un registro generico (con esclusione del PC);
- con **offset a registro scalato**: l'offset è un valore calcolato per tra- slazione o rotazione del contenuto in un registro generico (con esclusione del PC);

Modalità di indicizzazione

Per favorire l'accesso a blocchi di memoria o a strutture complesse quali gli array, sono state aggiunte le modalià indicizzate:

- **pre-indicizzata** prima si calcola l'indirizzo e successivamente si scrive il registro base
- **post-indicizzata** prima si utilizza il registro base come riferimento di memoria e dopo si aggiorna con il nuovo valore.

Quindi alle 3 modalità di indirizzamento (immediato, a registro e registro scalato) si vanno a sovrapporre le 3 modalità di aggiornamento (senza indicizzazione, pre- e post-indicizzato) per un totale di **9 combinazioni!**

...non tutte le istruzioni di load/store possono sfruttare queste combinazioni in quanto dipende dallo spazio disponibile nei 32-bit di codifica delle istruzioni stesse.

Esempi di indirizzamento

Nome	Esempio
A registro	MOV R0, R3
Assoluto	LDR R0, #memoria
Immediato	MOV R0, #15
Registro indiretto	ADD R2, R3, #15
Registro indiretto con offset	LDR R0, [R3]
Registro indiretto pre-incremento	LDR R0, [R4, #4]
Registro indiretto post-incremento	LDR R0, [R5, #4]!
Registro indiretto Registro indicizzato	LDR R0, [R7], #4
Registro indiretto indicizzato con shift	LDR R0, [R3, R2]
	LDR R0, [R3, R4, LSL #4]

Modalità di indicizzazione

Per favorire l'accesso a blocchi di memoria o a strutture complesse quali gli array, sono state aggiunte le modalià indicizzate:

- **pre-indicizzata** prima si calcola l'indirizzo e successivamente si scrive il registro base
- **post-indicizzata** prima si utilizza il registro base come riferimento di memoria e dopo si aggiorna con il nuovo valore.

Quindi alle 3 modalità di indirizzamento (immediato, a registro e registro scalato) si vanno a sovrapporre le 3 modalità di aggiornamento (senza indicizzazione, pre- e post-indicizzato) per un totale di **9 combinazioni!**

...non tutte le istruzioni di load/store possono sfruttare queste combinazioni in quanto dipende dallo spazio disponibile nei 32-bit di codifica delle istruzioni stesse.

Tipo di istruzioni

Le istruzioni ARM possono essere classificate in base alla tipologia e alla sintassi che le accomuna in diverse classi di istruzioni di:

- elaborazione e trasferimento dati;
- diramazione del flusso o di branch;
- accesso in memoria (load/store);
- cambiamento di stato;
- verso i coprocessori;
- per generare eccezioni;
- per la gestione dei dati;
- pseudo-istruzioni

Istruzioni di elaborazione e trasferimento dati

- Aritmetiche e logiche
- Aritmetiche con saturazione
- Confronto
- SIMD (Single Instruction Multiple Data)
- Selezione dei byte
- Moltiplicazione
- Trasferimento dati

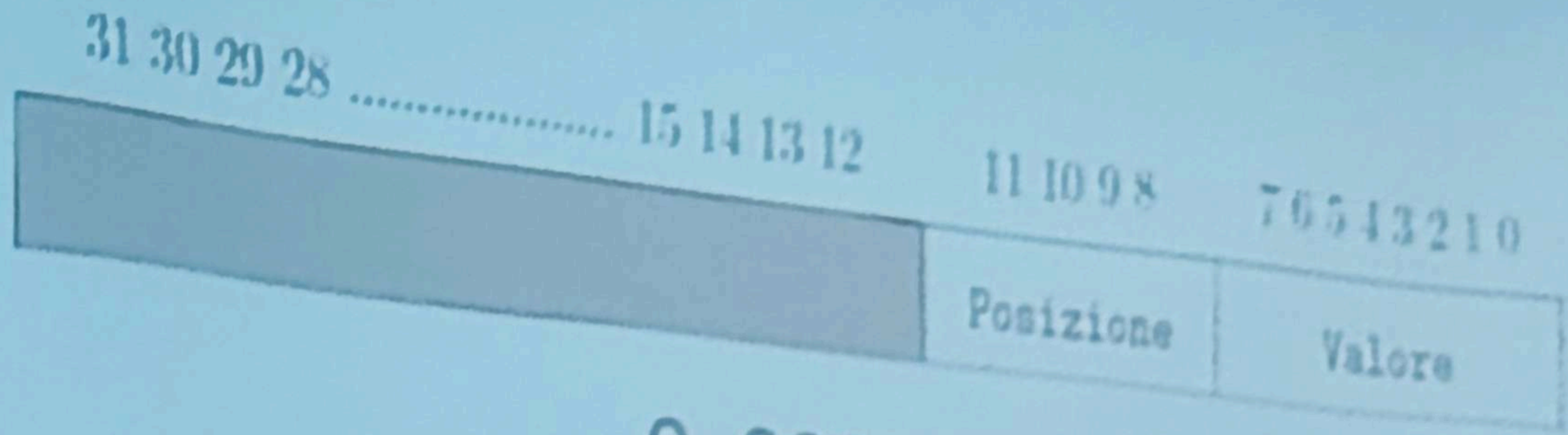
Istruzioni aritmetiche e logiche

31..28	27 26 25	24 .. 21	20	19..16	15..12	11 10 9 .... 2 1 0
PreCond	0 0 I	OPCODE	S	Rn	Rd	OP_2

<MNEM>{<PreCond>}{<S>} <Rd>, <Rn>, Op_2

OPCODE	MNEM	Descrizione	Semantica
0000	AND	Congiunzione logica	$Rd \leftarrow Rn \cdot OP_2$
0001	EOR	OR esclusivo	$Rd \leftarrow Rn \oplus OP_2$
0010	SUB	Sottrazione	$Rd \leftarrow Rn - OP_2$
0011	RSB	Sottrazione invertita	$Rd \leftarrow OP_2 - Rn$
0100	ADD	Somma	$Rd \leftarrow Rn + OP_2$
0101	ADC	Somma con riporto	$Rd \leftarrow Rn + OP_2 + C$
0110	SBC	Sottrazione con riporto	$Rd \leftarrow Rn - OP_2 - \bar{C}$
0111	RSC	Sottrazione invertita con riporto	$Rd \leftarrow OP_2 - Rn - C$
1110	BIC	Azzera bit	$Rd \leftarrow Rn \cdot \overline{OP_2}$
1100	ORR	Disgiunzione logica	$Rd \leftarrow Rn + OP_2$

Indirizzamento Immediato



0x00002040

.... 0010 0000 0100 0000

Indirizzamento Immediato

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00000081																									1	0	0	0	0	0	0	1
0x40000020	0	1																									1	0	0	0	0	0
0x10000008	0	0	0	1																									1	0	0	0
0x04000002	0	0	0	0	0	1																									1	0
0x81000000	1	0	0	0	0	0	0	1																								
0x20400000			1	0	0	0	0	0	0	1																						
0x08100000					1	0	0	0	0	0	0	1																				
0x02040000							1	0	0	0	0	0	0	1																		
0x00810000									1	0	0	0	0	0	0	1																
0x00204000											1	0	0	0	0	0	0	1														
0x00081000													1	0	0	0	0	0	0	1												
0x00020400															1	0	0	0	0	0	0	1										
0x00008100																	1	0	0	0	0	0	0	1								
0x00002040																			1	0	0	0	0	0	0	1						
																									1	0	0	0	0	0	0	1

Superare le difficoltà

MOV R0, #0xFFFFFFFF → MVN R0, #0
R0 = NOT 0x00000000 = 0xFFFFFFFF

MOV R0, #0x55550000 → Errore a compile-time

Istruzioni aritmetiche con saturazione

Se si utilizzano 32-bit per codificare i numeri interi con segno l'intervallo di rappresentazione si divide in due sottoinsiemi: da 1 a $2^{31} - 1$ per i valori positivi e da -1 a -2^{31} per quelli negativi.

Se sommando i numeri $7 \cdot 2^{28}$ e $2 \cdot 2^{28}$, il risultato che si ottiene ($8 \cdot 2^{28}$) non è positivo ma -2^{31}

```
0x70000000   +
0x10000000   =
----------
0x80000000
```

Quando un valore supera un estremo, quest'ultimo e ricondotto al valore limite e si asserisce il fag Q nel registro di stato CPSR.

Istruzioni aritmetiche con saturazione

Se si utilizzano 32-bit per codificare i numeri interi con segno l'intervallo di rappresentazione si divide in due sottoinsiemi: da 1 a $2^{31} - 1$ per i valori positivi e da -1 a -2^{31} per quelli negativi.

Se sommando i numeri $7 \cdot 2^{28}$ e $2 \cdot 2^{28}$, il risultato che si ottiene ($8 \cdot 2^{28}$) non è positivo ma -2^{31}

```
0x70000000   +
0x10000000   =
----------
0x80000000
```

Quando un valore supera un estremo, quest'ultimo e ricondotto al valore limite e si asserisce il fag Q nel registro di stato CPSR.

Principali istruzioni aritmetiche con saturazione

<MNEM>{<PreCond>} <Rd>, <Rm>, <Rn>

Descrizione	MNEM	CODE	Semantica
Addizione con saturazione	QADD	10000	Rd ← sat(Rm + Rn)
Addizione con doppia saturazione	QDADD	10100	Rd ← sat(Rm + sat(2 · Rn))
Sottrazione con saturazione	QSUB	10010	Rd ← sat(Rm − Rn)
Sottrazione con doppia saturazione	QDSUB	10110	Rd ← sat(Rm − sat(2 · Rn))

$$sat(x) = max(min(x; 2^{31}-1); -2^{31})$$

Istruzioni di Confronto

Le istruzioni di confronto utilizzano in ingresso due operandi ma non hanno un registro destinazione poiché il loro effetto è nella scrittura del registro di stato in base al valore risultato.

<MNEM>{<PreCond>} <Rn>, OP_2

utilizzano lo stesso formato delle istruzioni aritmetiche/logiche → stesse regole per la modalità di indirizzamento del secondo operando.

Descrizione	MNEM	CODE	Semantica
Test bit	TST	1000	CPSR ← <Rn> · OP_2
Test uguaglianza bit-a-bit	TEQ	1001	CPSR ← <Rn> ⊕ OP_2
Comparazione	CMP	1010	CPSR ← <Rn> - OP_2
Comparazione negativa	CMN	1011	CPSR ← <Rn> + OP_2

Istruzioni di Confronto

- **CMP** si comporta come **SUB**, **CMN** come **ADD**, perchè sottrarre un numero negativo equivale a sommarlo,
- **TST** come **AND**, **TEQ** come **EOR** poichè l'EXOR di numeri identici da risultato zero.

```
// Indirizzamento Immediato
CMP R1, #256          // Compara R1 con 256 e agg. flag
TST R2, #1            // Test bit LSB di R2 e agg. flag
// Indirizzamento a registro
TEQ R2, R3            // Test R2=R3? e agg. flag
// Indirizzamento con shift immediato
CMN R3, R3, LSL #4    // R3 + R3 * 16 e agg. flag
TEQ R0, R1, RRX       // R0 XOR R1 AND RRX(R1) e agg. flag
// Indirizzamento con shift a registro (5 bit LSB di R3)
TST R0, R1, LSL R3    // Test R0 = R1*(2^R3) e agg. flag
CMN R0, R1, LSR R3    // R0 + R1/(2^R3) e agg. flag
```

Istruzioni SIMD

L'architettura ARMv6 ha introdotto delle istruzioni aritmetiche che agiscono sui registri considerandoli come contenitori di strutture array definiti su word o halfword.

La sintassi generica di una istruzione aritmetica SIMD è la seguente:

```
<MNEM>{<PreCond>} <Rd>, <Rn>, <Rm>
```

Il codice mnemonico <MNEM> si ottiene dalle combinazioni di:

- prefisso
  - U ➡ Unsigned
  - S ➡ Signed
  - Q ➡ Saturazione
  - H ➡ Risultati dimezzati per prevenire overflow
- tipo di istruzione / dimensione del elemento dell'array
  - 8 ➡ ADD8 SUB8
  - 16 ➡ ADD16 SUB16
  - X ➡ ADDSUBX SUBADDX