

## Il processore ARM: La storia

Nato da un progetto dell' Acorn iniziato nel 1983 dopo il grande successo del BBC Micro rilasciato nel 1982. Siccome l'Acorn era una azienda non comparabile alle aziende americane cercò di sviluppare un processore che soddisfacesse i requisiti interni dell'azienda e che avesse un basso costo di produzione da questo progetto nacque l'ARM1 usato solo per scopi interni. Nel 1987 venne lanciato commercialmente la prima piattaforma ARM Archimedes in versione ARM2 8mhz. La VLSI Technology Inc compagnia partner della Acorn nello sviluppo dell'ARM convince una fetta di mercato ad utilizzare questo tipo di processori a basso costo .Nel 1989 esce l'ARM3 versione potenziata dell' ARM2 integrando 4kbit di cache e un frequenza di clock di 25mhz. Nel 1990 nasce la ARM Ltd composta da Acorn VLSI e Apple. Molte versioni sono uscite sul mercato negli ultimi anni e attualmente la famiglia ARM copre il 75% del mercato mondiale dei processori a 32 bit per applicazioni embedded, essendo a pieno titolo una delle più diffuse architetture a 32 bit del mondo. La sua particolare architettura( System on a Chip )contiene all'interno di un chip interno un intero sistema, o meglio, oltre al processore centrale, anche un chipset ed eventualmente altri controller come quello per la memoria RAM, la circuiteria input/output o il sotto sistema video.

Tra i prodotti più celebri che usano microcontrollori o sistemi on chip basati su questa CPU sono l'iPod della Apple, il Game Boy Advance e il Nintendo DS della Nintendo, la maggior parte dei telefoni Nokia e il Lego Mindstorms NXT inoltre è possibile reperire sulla rete molto materiale informativo riguardo l'installazione di SO su sistemi di questo genere: In generale se si possiede un ARMv4 o successivo, è possibile utilizzare diversi sistemi operativi come ad esempio: Gentoo. (guida reperibile all'indirizzo web <http://www.gentoo.org/doc/it/handbook/handbook-arm.xml?full=1&style=printable> )

I processori ARM sono prevalentemente venduti sotto forma di *cores*, da utilizzarsi nei System-on-Chip (SoC) integrati. I *cores* possono essere venduti nelle seguenti tipologie:

- Hard-Cores: la ARM fornisce un layout fisico del sistema da integrare in una specifica tecnologia;
- Soft-Cores: la ARM fornisce una descrizione ad alto livello del sistema che può essere integrata in una qualunque tecnologia a discrezione del progettista.

In rari casi, i processori ARM vengono venduti sotto forma di dispositivi stand-alone.

## Il processore ARM: Caratteristiche generali

L'Arm è un processore a 32 bit realizzato tramite un circuito integrato contenente approssimativamente 35000 transistor. Possiede la capacità di eseguire la maggior parte delle istruzioni in un singolo periodo di clock,caratterizzando l'architettura come RISC (acronimo dell'inglese Reduced Instruction Set Computer) indica una filosofia di progettazione di architetture per microprocessori formate da un set di istruzioni contenente istruzioni in grado di eseguire operazioni semplici che possono essere eseguite in tempi simili. In una macchina RISC le uniche operazioni in grado di accedere alla memoria sono le operazioni di load e di store, tutte le altre utilizzano solo i registri.

R.I.S.C. : Reduced Instruction Set Computing

- 1) più semplice scalabilità della frequenza
- 2) istruzioni implementate in modo efficiente
- 3) alto numero di registri
- 4) maggiore efficienza

L'architettura inoltre permette la possibilità di eseguire singole istruzioni in modo condizionato rispetto allo stato della CPU, la possibilità di ricevere tre operandi per istruzione e l'ottimizzazione del codice tramite l'utilizzo combinato della ALU e dello shifter. Questo sistema embedded è dotato di un bus verso la memoria limitato e, sebbene il processore possa indirizzare a 32 bit, spesso si utilizzano indirizzamenti a 16 bit chiamata Thumb Instruction set.


## Il processore ARM: Modi d'esecuzione e registri

Il processore ARM supporta sette modi di funzionamento (processor mode). Il modo può essere cambiato via software, in modalità privilegiata, oppure per effetto di una eccezione. Di solito i programmi utente vengono eseguiti in modo User; gli altri modi, noti come modi privilegiati, sono utilizzati per accedere alle risorse protette o per servire le eccezioni. Per questo motivo non è possibile passare dal modo User ad un modo protetto se non attraverso un'eccezione (un'istruzione di software interrupt, un'interruzione esterna o altra eccezione).

Processor mode	Descrizione	Codifica M[4:0]	registri accessibili
1) User (usr)	Modo d'esecuzione dei programmi comuni.	0b10000	pc,r0:r14,cpsr
2 FIQ (fiq)	Gestione di <i>interrupt</i> veloce.	0b10001	pc,r8_fiq : r14_fiq,r0 : r7 , cpsr, spsr_fiq
3 IRQ (irq)	Gestione di <i>interrupt</i> generico.	0b10010	pc,r14_irq, r13_irq,r0 : r12 , cpsr, spsr_irq
4 Supervisor (svc)	Modo protetto per l'esecuzione di codice del sistema operativo.	0b10011	pc,r14_svc, r13_svc,r0 : r12 , cpsr, spsr_svc
5 Abort (abt)	Errore nell'accesso di memoria (anche per implementare memoria virtuale o protezione della memoria).	0b10111	pc,r14_abt, r13_abt,r0 : r12 , cpsr, spsr_abt
6 Undefined (und)	Istruzione illegale.	0b11011	pc,r14_und, r13_und,r0 : r12 , cpsr, spsr_und
7 System (sys)	Modo privilegiato d'esecuzione di un <i>task</i> del sistema operativo.	0b11111	pc,r0 : r14, cpsr

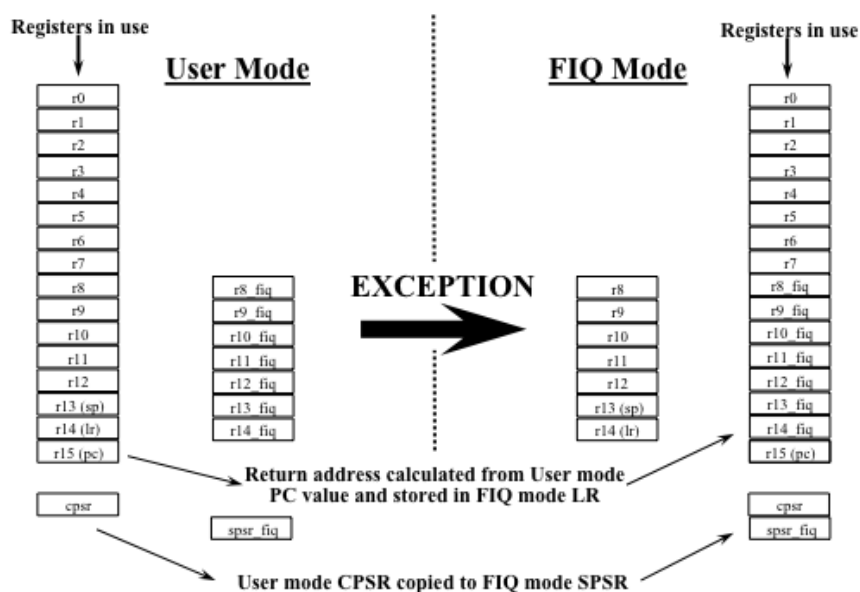
Il “modello software” dell'ARM, mostrato nella figura 1, include un set di 37 registri di interi a 32 bit ciascuno: 31 di tipo GPR (General-Purpose-Register), da usare per operazioni di memorizzazione temporanea, operazioni logiche, intere ed ogni altra applicazione di carattere generico e 6 registri di stato che sono comunemente usati per confrontare e testare condizioni richieste dai programmi.

Modes						
Privileged modes						
Exception modes						
User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

 indicates that the normal register used by User or System mode has been replaced by an alternative register specific to the exception mode

**Figura 1**

In ogni istante sono visibili sempre 15 registri di tipo generale e uno o 2 registri di stato. I primi registri (R0-R7) sono unbanked (puntano sempre alle stesse locazioni fisiche) successivi registri da R8 a R14 sono banked (a seconda della modalità puntano a locazioni fisiche diverse). Questo meccanismo di “switching” è descritto nella Figura 2.



**Figura 2**

Tre dei registri generali hanno anche un significato speciale: R13 (SP) viene utilizzato come Stack Pointer a supporto della realizzazione di subroutine e routine di servizio, R14 (LR) viene utilizzato come Link Register per memorizzare l’indirizzo di ritorno in una chiamata a subroutine (vedi istruzione BL), R15 (PC) è il Program Counter. I registri sono praticamente ordinati in banchi parzialmente sovrapposti, un banco di registri associato ad ogni Processor Mode (il modo System utilizza lo stesso banco del modo User). Nel modo usr ad esempio sono visibili 15 registri (quello corrente ed eventualmente quello di salvataggio) ed il PC.

Il CPSR (Current Program Status Register) è il registro di stato, illustrato nella Figura 3. Include i bit di condizione (i 4 bit più significativi), le maschere per le interruzioni e la codifica del processor mode.



**Figura 3**

- M0-M4 definiscono il modo di processore;
- F indica lo stato di disabilitazione (valore 1) dei "Fast Interrupt" (FIQ) ;
- I indica lo stato di disabilitazione (valore 1) delle interruzioni "generiche" (IRQ).
- V Overflow: bit di condizione overflow per l'aritmetica con segno in complemento a due.
- C Carry: bit di condizione carry per l'aritmetica senza segno e shift.
- Z Zero: indica il risultato di un'operazione pari a zero.
- N Negative: indica il risultato di un'operazione negativo.

V, C, Z e N sono bit modificabili in accordo con il risultato di una qualsiasi istruzione d'elaborazione dei dati (ADD, SUB, RSC ecc.). Tuttavia, con l'eccezione delle istruzioni di "Compare" (CMP e CMN) e di "Test" (TEQ e TST), le istruzioni d'elaborazione dei dati non alterano i bit di condizione a meno che il bit S (Set Condition Codes) dell'istruzione non sia impostato a 1. Questo si ottiene posponendo la lettera al simbolo operativo dell'istruzione (ADDS, SUBS, RSCS, ecc.). Per le istruzioni CMP, CMN, TEQ e TST, il bit S e' sempre attivo.

Il registro SPSR (Saved Program Status Register) ha il compito di preservare il valore di CPSR durante la gestione di un'eccezione. E' presente uno SPSR per ogni modo del processore tranne che per il modo User/System, durante il quale non è prevista la gestione di un'eccezione.

## Il processore ARM: Gli Interrupt

L'interrupt è una particolare caratteristica degli ARM (e dei microprocessori in generale) che consente di intercettare un evento esterno, interrompere momentaneamente il programma in corso, eseguire una porzione di programma specializzata per la gestione dell'evento verificatosi e riprendere l'esecuzione del programma principale.

Qualunque sia l'evento abilitato, al suo manifestarsi l'ARM interrompe l'esecuzione del programma in corso, e salta all'istruzione presente nella locazione di memoria denominata Interrupt Vector Table(vettore di interrupt) che associa a ciascun tipo di eccezione, un indirizzo denominato 'vettore d'eccezione' (hard vector). Uno schema che illustra la struttura di una possibile Interrupt Vector Table è visibile in Figura 4.

Hard vector (Hex)	Funzione	Modo
0x00000000	Reset	SVC
0x00000004	Undef. Inst.	UNDEF
0x00000008	Soft. Int. (SWI)	SVC
0x0000000c	Prefetch Abort	ABORT
0x00000010	Data Abort	ABORT
0x00000014	Address Excep.	SVC
0x00000018	IRQ Interrupt	IRQ
0x0000001c	FIQ Interrupt	FIQ

**Figura 4**

L'elaborazione di una eccezione, tra l'altro, forza il PC ad assumere il valore del corrispondente vettore. Di norma a quell'indirizzo è collocata un'istruzione di salto alla corrispondente routine di servizio: solo per l'ultima eccezione (FIQ), non essendoci altri vettori che seguono nella tabella, all'indirizzo del corrispondente vettore può direttamente iniziare la sua RSI. Quando viene sollevata un'eccezione e inizia la corrispondente elaborazione, vengono eseguite le seguenti azioni:

1) il processore salva il contenuto del PC nel registro LR corrispondente all'eccezione in corso di elaborazione

```
R14_<exception_mode> = PC
```

2) salva il registro di stato nella copia di salvataggio corrispondente all'eccezione in corso di elaborazione

```
SPSR_<exception_mode> = CPSR
```

3) aggiorna il registro di stato col modo di processore.

```
CPSR[4:0] = <identificativo eccezione>
```

4) se il modo d'eccezione è Reset o FIQ allora disabilita le interruzioni 'veloci' (imposta a 1 il bit F)

```
if <exception_mode> == (Reset or FIQ) then CPSR[6] = 1
```

5) disabilita le interruzioni 'comuni' (imposta a 1 bit I)

```
CPSR[7] = 1
```

6) salta all'indirizzo del corrispondente vettore

```
PC = <exception hard vector>
```

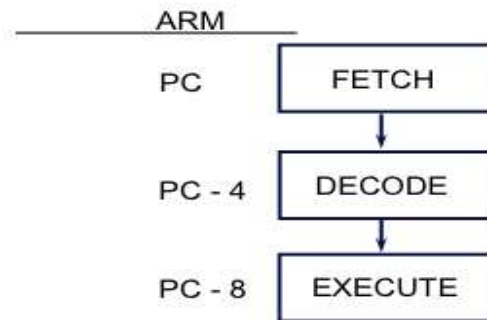
Per tornare alla esecuzione del programma invece si devono effettuare le seguenti operazioni:

- ricaricare il CPSR dal backup effettuato nel registro SPSR\_<mode>
- riportare il PC al valore coerente dal registro LR\_<mode>

# Il processore ARM: La Pipeline

L'organizzazione dell'esecuzione di un'istruzione come una catena di stadi viene detta pipeline, e il concetto è analogo a quello della catena di montaggio in un'industria, dove il prodotto (nel nostro caso l'istruzione) attraversa diverse fasi prima di essere completato, e la catena viene continuamente alimentata in modo che ogni unità lavori costantemente. Il vantaggio di questo tipo di organizzazione sta nel fatto che il tempo necessario a terminare un prodotto (ad eseguire completamente un'istruzione) non è dato dalla somma dei tempi di attraversamento delle singole fasi, ma, idealmente, dal tempo di un singolo stadio. Nel nostro specifico caso le varie versioni di Arm nel tempo hanno scandito una vasta gamma di implementazioni diverse riguardo la gestione della pipeline. L'arm9 ad esempio possiede una pipeline a 5 stadi. In questo documento verrà preso in riferimento l'arm 7 che possiede tre stadi di pipeline. L'unità di controllo del processore esegue una istruzione svolgendo le seguenti tre operazioni di base:

- Fetch (lettura)
- Decode (decodifica)
- Execute (esecuzione)



Un programma è eseguito reiterando il ciclo fetch-decode-execute (ciclo macchina) per eseguire ordinatamente le sue istruzioni. In caso di pipeline alcune di queste fasi possono essere sovrapposte come nelle fasi di catena di montaggio permettendo quindi l'ottimizzazione del tempo di throughput. Nel caso dell'ARMv7 il parallelismo di tre stadi consente in caso di best case contemporaneamente, l'esecuzione di una istruzione, la decodifica della istruzione successiva e la fase di fetch della terza istruzione da eseguire. Il caso di worst case invece come si vede dall'esempio visibile sotto ci accorgiamo che durante la decodifica della istruzione D1 al tempo T2 il processore si accorge di avere eseguito una jmp quindi elimina la istruzione fetchata F2 e riprende a caricare l'istruzione FN perdendo quindi il ritmo di elaborazione. Dimostrazione è il fatto che al tempo T4 nessuna istruzione può essere eseguita.

BEST CASE					WORST CASE				
T0	T1	T2	T3		T0	T1	T2	T3	T4
F0	F1	F2	F3		F0	F1	F2	FN	FN+1
	D0	D1	D2			D0	D1	X	DN
		E0	E1				E0	E1	X

## ARM Instruction Set : Istruzioni Condizionate

Molte istruzioni in linguaggio Assembler per ARM sono codificate in funzione della possibilità di poterle eseguire in maniera condizionata. L'utilizzo di queste funzioni però rallentano l'esecuzione delle istruzioni poiché, dovendo il processore controllare i flag di per determinare l'eseguibilità della istruzione, può eseguire un ciclo di istruzioni condizionate alla volta annullando concretamente il lavoro effettuato dalla pipeline. Come illustrato in figura 5, quattro bit della codifica di ogni istruzione sono riservati per indicare le condizioni che devono essere preliminarmente soddisfatte (valutando i bit di condizione) perché l'istruzione venga effettivamente eseguita.

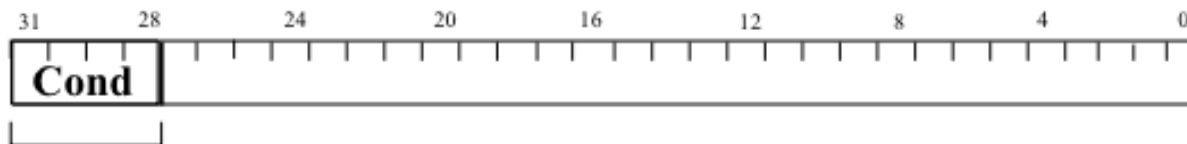


Figura 5

A questo scopo, ogni simbolo operativo d’istruzione può essere esteso con alcuni postfissi. Nel caso in cui non sia specificata alcuna condizione, si assume implicitamente la condizione AL. In figura 6 sono mostrati i postfissi possibili.

Estensione mnemonica	Significato	Flag di condizione	Opcode [31:28]
EQ	Uguali	Z=1	0000
NE	Non uguali	Z=0	0001
CS/HS	Carry Attivato / Senza segno maggiore o uguale	C=1	0010
CC/LO	Carry Disattivato / Senza segno minore	C=0	0011
MI	Negativo	N=1	0100
PL	Positivo o Zero	N=0	0101
VS	Overflow	V=1	0110
VC	Non Overflow	V=0	0111
HI	Senza segno maggiore	C=1 e Z=0	1000
LS	Senza segno minore o uguale	C=0 o Z=1	1001
GE	Con segno maggiore o uguale	N=V	1010
LT	Con segno minore	N!=V	1011
GT	Con segno maggiore	Z=0 e N=V	1100
LE	Con segno minore o uguale	Z=1 o N!=V	1101
AL	Sempre (è il default)	-	1110
NV	Mai	-	1111

Figura 6

Per eseguire una istruzione condizionata basta aggiungere il postfisso desiderato. Ad esempio:

- supponiamo di eseguire l'operazione ADD

-ADD r0,r1,r2 ; r0 = r1 + r2 (ADDAL)

- e adesso desideriamo eseguirla solo in caso in cui il flag zero è impostato a livello alto.

-ADDEQ r0,r1,r2 ; If zero flag set then... r0 = r1 + r2 else  
; passa alla prossima istruzione

- Inoltre, i flag non vengono aggiornati di default. Se vogliamo effettuare l'operazione di somma con aggiornamento dei flag dobbiamo aggiungere il postfisso “S”

-ADDS r0,r1,r2 ; r0 = r1 + r2 and set flags

## ARM Instruction Set : Istruzioni di Ramificazione (Branch)

Queste istruzioni prevedono la possibilità di modificare il valore del PC come registro destinazione, indirizzandolo con un indirizzo di salto nell’intero spazio di indirizzamento da 4Gbyte, i salti possono essere ottenuti eseguendo una delle due sintassi apposite di salto relativo( B e BL ). La struttura di queste istruzioni è visualizzata in figura 7.

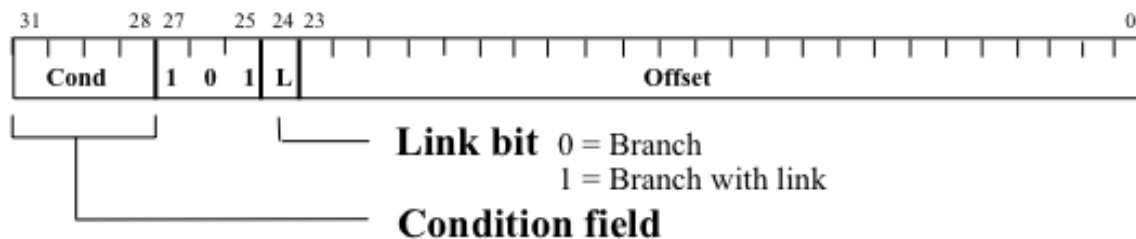


Figura 7

Queste istruzioni caricano nel registro PC (R15), il valore che viene calcolato come somma del PC corrente e dell'offset relativo specificato nella codifica dell'istruzione. È compito dell'assemblatore calcolare tale offset dall'istruzione. L'assemblatore fa uso di solito di label: l'offset è riferito all'indirizzo di caricamento dell'istruzione di salto +8. Il suo valore, diviso per 4, viene codificato in un campo da 24 bit con segno, consentendo pertanto un salto relativo rispetto alla posizione dell'istruzione di salto di circa +/- 32 Mbyte. BL preliminarmente copia il contenuto del PC nel registro LR (R14): tale contenuto è pari all'indirizzo dell'istruzione successiva a BL. L'istruzione corrisponde quindi ad un salto a subroutine con salvataggio (in LR) dell'indirizzo di ritorno.

La sintassi dell'istruzione B (Branch) è la seguente:

`B{<cond>} <target_address>`

Carica nel registro PC l'indirizzo ricavato dalla somma del PC corrente (indirizzo dell'istruzione Branch + 8) e del valore dell'offset codificato nell'istruzione, premoltiplicato per 4.

Es.:

```
B LAB1      ; salta al label "LAB1"
BEQ LAB2     ; se Z=1 allora salta al label "LAB", altrimenti
              ; prosegue con l'istruzione successiva
```

La sintassi dell'istruzione BL (Branch and Link) è la seguente:

`BL{<cond>} <target_address>`

Copia nel registro LR (R14) l'indirizzo di ritorno (indirizzo dell'istruzione BL + 4) e carica nel registro PC l'indirizzo ricavato dalla somma del PC corrente (indirizzo dell'istruzione BL + 8) e del valore dell'offset codificato nell'istruzione, premoltiplicato per 4.

Es.: `BLVC SUBR1 ; se V=0 chiama la subroutine "SUBR1"`

## ARM Instruction Set : Istruzioni per processare i dati

La maggior parte delle istruzioni dell'ARM hanno lo stesso formato. Essendo l'ARM un architettura di tipo load/store queste istruzioni operano solamente sui registri e non sulla memoria. Ognuna di queste istruzioni utilizza uno o due operandi. Il primo operando è sempre un registro, il secondo operando prima di essere mandato alla ALU passa dal barrel shifter.

Le istruzioni possono essere suddivise in: Istruzioni Aritmetiche, Istruzioni di Confronto, Istruzioni logiche, Istruzioni di Spostamento Dati.



## ARM Instruction Set : Istruzioni Aritmetiche

Le istruzioni aritmetiche sono:

- ADD                primo operando + secondo operando
- ADC                primo operando + secondo operando + carry
- SUB                primo operando - secondo operando
- SBC                primo operando - secondo operando + carry -1
- RSB                secondo operando - primo operando
- RSC                secondo operando - primo operando + carry - 1

Sintassi:

- <Istruzione>{<cond>}{S} Rd, Rn, secondo operando

Esempi:

- ADD r0, r1, r2
- SUBGT r3, r3, #1
- RSBLES r4, r5, #5

## ARM Instruction Set : Istruzioni di Confronto

Tutte le istruzioni di confronto non restituiscono risultati, ma aggiornano solamente i Condition Flags, quindi non abbiamo bisogno di specificare il postfisso S nella sintassi dell'istruzione.

Le istruzioni di confronto sono :

- CMP                primo operando - secondo operando
- CMN                primo operando + secondo operando
- TST                primo operando AND secondo operando
- TEQ                primo operando EOR secondo operando

Sintassi:

- <Istruzione>{<cond>} Rn, secondo operando

Esempi:

- CMP                r0, r1
- TSTEQ              r2, #5

## ARM Instruction Set : Istruzioni Logiche

Le istruzioni logiche sono:

- AND                primo operando AND secondo operando
- EOR                primo operando EOR secondo operando
- ORR                primo operando OR secondo operando
- BIC                primo operando AND NOT secondo operando

Sintassi:

- <Istruzione>{<cond>}{S} Rd, Rn, secondo operando

Esempi:

- AND                      r0, r1, r2
- BICEQ                  r2, r3, #7
- EORS                    r1, r3, r0

## ARM Instruction Set : Spostamento di dati tra registri

Le istruzioni di spostamento di dati tra registri sono:

- MOV                      secondo operando
- MVN                    NOT secondo operando

Si noti che queste istruzioni non fanno uso del campo relativo al primo operando.

Sintassi:

- <Istruzioni>{<cond>}{S} Rd, secondo operando

Esempi:

- MOV                    r0, r1
- MOVS                r2, #10
- MVNEQ r1, #0

## ARM Instruction Set : Barrel Shifter

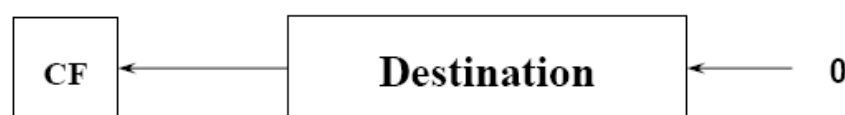
L'Assembler dell'ARM non ha effettive istruzioni di shifting. E' però dotato di un meccanismo per effettuare shift come parte di altre istruzioni, detto *barrel shifter*.

Il barrel shift supporta diverse operazioni:

- Logical Shift Left (LSL)

L'operando è rappresentato dal valore ottenuto dallo scorrimento logico a sinistra del contenuto della destinazione di un numero di posizioni specificato. Le posizioni vuote vengono riempite con il valore 0. Il Carry Flag resta invariato se il numero di posizioni è pari a 0, altrimenti viene impostato al valore del MSB della destinazione. Il funzionamento dell'istruzione LSL è mostrato in figura 8.

**Logical Shift Left (LSL)**



**Figura 8**

- Logical Shift Right (LSR)

L'operando è rappresentato dal valore ottenuto dallo scorrimento logico a destra del contenuto della destinazione di un numero di posizioni specificato. Le posizioni vuote vengono riempite con il valore 0. Il Carry Flag resta invariato se il numero di posizioni è pari a 0, altrimenti viene impostato al valore del LSB della destinazione. Il funzionamento dell'istruzione LSR è mostrato in figura 9.

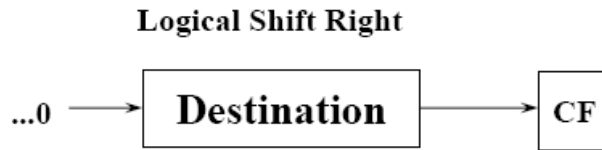


Figura 9

- Arithmetical Shift Right (ASR)

Effettua l'operazione di shift preservando il bit di segno (utile per i numeri rappresentati in complemento a due). L'operazione di shifting avviene solo a partire dal secondo bit più significativo. Le posizioni vuote vengono riempite con il valore 0. Il Carry Flag resta invariato se il numero di posizioni è pari a 0, altrimenti viene impostato al valore del LSB della destinazione. Il funzionamento dell'istruzione ASR è mostrato in figura 10.

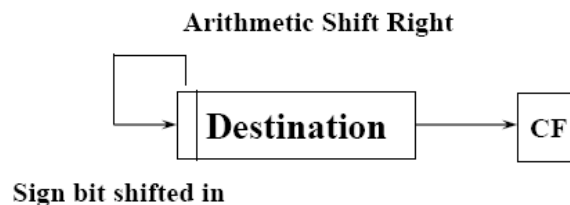


Figura 10

- Rotate On Right (ROR)

Il funzionamento è simile al ASR, ma anziché riempire le posizioni vuote con il valore 0, per ogni spostamento a destra memorizza come MSB il bit uscente. Il Carry Flag al termine dell'operazione memorizza l'ultimo bit spostato a destra. Il funzionamento dell'istruzione ROR è mostrato in figura 11.

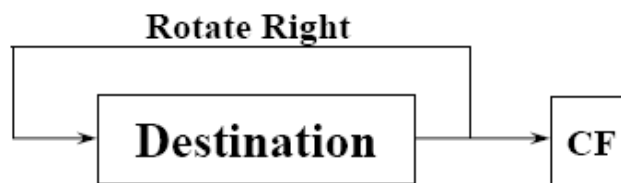


Figura 11

- Rotate Right eXtended (RRX)

Questa operazione utilizza il carry flag del CPSR (Current Process Status Register) come bit aggiuntivo ai 32 bit della destinazione. RRX quindi si comporta come l'istruzione ROR, ma facendo shiftare il LSB della destinazione sul carry flag e quest'ultimo sul MSB della destinazione. Il funzionamento dell'istruzione RRX è mostrato in figura 12.

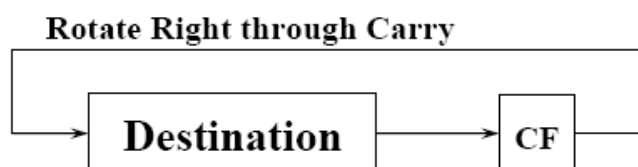


Figura 12

## ARM Instruction Set : Barrel Shifter (Second Operand)

Il secondo operando del barrel shifter può assumere solo una delle seguenti forme:

- Registro

L'operando è un registro di uso comune, che a sua volta può facoltativamente essere integrato con le operazioni di shifting. Il valore di shifting può essere specificato attraverso un intero di 5 bit senza segno, oppure negli ultimi 8 bit di un altro registro.

- Valore immediato #<immediate>

L'operando è un valore immediato (costante). Il valore <immediate> è un valore costante da 8-bit ruotato a destra di un numero pari di posizioni (0,2,4,...,28,30) in una parola da 32 bit. Nella codifica si usano, pertanto, 8 bit per memorizzare il valore da ruotare e 4 bit per memorizzare la metà del numero di posizioni di rotazione da applicare. Ne consegue che non tutti i valori rappresentabili con 32 bit sono riconducibili ad un valore da 8 bit ruotato; in particolare non sono costruibili in questo modo valori rappresentati in binario con più di 8 bit pari a 1.

Esempio di valore valido: 0xf000000f che equivale a 0xff ruotato a destra di 4 bit (in una parola da 32 bit).

Esempio di valore non valido: 0xffff non è valido in quanto composto da più di 8 bit pari a 1.

## ARM Instruction Set : Shifted Register

Il numero di posizioni che determina lo shifting di un registro può essere contenuta:

- Nel campo di 5 bit seguente l'istruzione. In questo caso l'operazione di shift sarà eseguita in un singolo ciclo e eviteremo l'overhead.
- Negli ultimi 8 bit di un registro (che non sia il PC). In questo caso l'operazione durerà più cicli, poiché L'ARM non è in grado di leggere 3 registri simultaneamente. Le prestazioni sono identiche a quelle dei processori dove lo shift è un'istruzione separata.

Se non specificato sarà applicato uno shift nullo di default: LSL #0

Lo Shifted Register utilizzato in combinazione con istruzioni quali MOV, ADD, SUB permette di eseguire moltiplicazioni per costanti in un singolo periodo di clock. Invece di utilizzare le istruzioni di moltiplicazione (che richiederebbero numerosi cicli per essere eseguite), viene effettuata la scomposizione della costante nella notazione  $(2^n) \pm 1$ , e successivamente si applicano operazioni di shifting.

```
Es.: r0 = r1 * 5
      r0 = r1 + (r1 * 4)
      ADD r0, r1, r1, LSL #2
```

```
Es.: r2 = r3 * 105
      r2 = r3 * 15 * 7
      r2 = r3 * (16 - 1) * (8 - 1)
      RSB r2, r3, r3, LSL #4 ; r2 = r3 * 15
      RSB r2, r2, r2, LSL #3 ; r2 = r2 * 7
```