

ARM Instruction Set : Second Operand (Valori Immediati)

L'Assembler dell'ARM è costituito unicamente da istruzioni a 32 bit, dei quali 12 sono dedicati alla memorizzazione del secondo operando. L'utilizzo diretto dei 12 bit introdurrebbe una limitazione del range di valori dell'operando da 0 a 4096 (2^{12}). Per questo motivo, si adotta un meccanismo basato sull'utilizzo del Barrel Shifter. Dei 12 bit a disposizione, ne vengono effettivamente utilizzati solo 8, sui quali si opera successivamente una rotazione a destra (ROR) di un numero pari di posizioni. Questo aumenta considerevolmente il numero di costanti numeriche a disposizione, sebbene introduca una perdita di granularità (poiché non tutti i numeri nell'intervallo saranno realizzabili in questo modo). Infatti, come possiamo osservare:

- 0 - 255 [0 - 0xff]
- 256,260,264,...,1020 [0x100-0x3fc, step 4, 0x40-0xff ror 30]
- 1024,1040,1056,...,4080 [0x400-0xff0, step 16, 0x40-0xff ror 28]
- 4096,4160, 4224,...,16320 [0x1000-0x3fc0, step 64, 0x40-0xff ror 26]

(Per ottenere i valori intermedi, si può adottare un meccanismo di somme successive). L'assemblatore ARM incapsula questo meccanismo, in modo che il programmatore possa direttamente inserire la costante all'interno del codice.

- MOV r0, #4096 ; => MOV r0, #0x1000 (ie 0x40 ror 26)

Un metodo più semplice (anche se meno performante) consiste nell'accedere ad una *look-up table* in memoria tramite l'istruzione LDR:

- LDR rd,=numeric constant

Questa istruzione consente di caricare qualsiasi costante a 32 bit. Inoltre, se la costante può essere generata tramite l'istruzione MOV, l'assemblatore convertirà l'istruzione LDR nella relativa MOV.

- LDR r0,=0x42 ; generates MOV r0,#0x42

Per questo motivo l'istruzione LDR rappresenta il modo migliore per effettuare il caricamento di costanti a valori immediati.

ARM Instruction Set : Istruzioni di Moltiplicazione

L'assembler ARM fornisce due istruzioni di moltiplicazione, su operandi con e senza segno:

MUL{<cond>}{S} Rd, Rm, Rs ; $Rd = Rm * Rs$

MLA{<cond>}{S} Rd, Rm, Rs,Rn ; $Rd = (Rm * Rs) + Rn$

L'istruzione MLA (multiply accumulate) consente di eseguire una somma al risultato della moltiplicazione nella stessa istruzione.

Si applicano inoltre le seguenti restrizioni:

- Rd e Rm devono essere registri diversi
- Non si può utilizzare il PC

ARM Instruction Set : Istruzioni Load/Store

L'architettura ARM non può eseguire operazioni da memoria a memoria, e pertanto deve trasferire i dati sui registri prima di poterli elaborare. Le istruzioni di tipo Load/Store eseguono questo tipo di trasferimenti.

Le istruzioni di tipo “Single Register” (LDR, STR) consentono di specificare la dimensione e il segno del dato da trasferire: Byte, Word (32 bit), Half-Word(16 bit), Signed Byte, Signed Word. Inoltre, possono essere eseguite in modo condizionale inserendo l'apposito codice per specificare una condizione.

- e.g. LDREQB ; LDR=mnemonico – EQ=condizione – B: dimensione (Byte)

L'indirizzamento di una locazione di memoria avviene mediante indirizzamento indiretto, specificando un *Base Register* che contiene l'indirizzo della locazione di memoria di riferimento, come illustrato nella Figura 1.

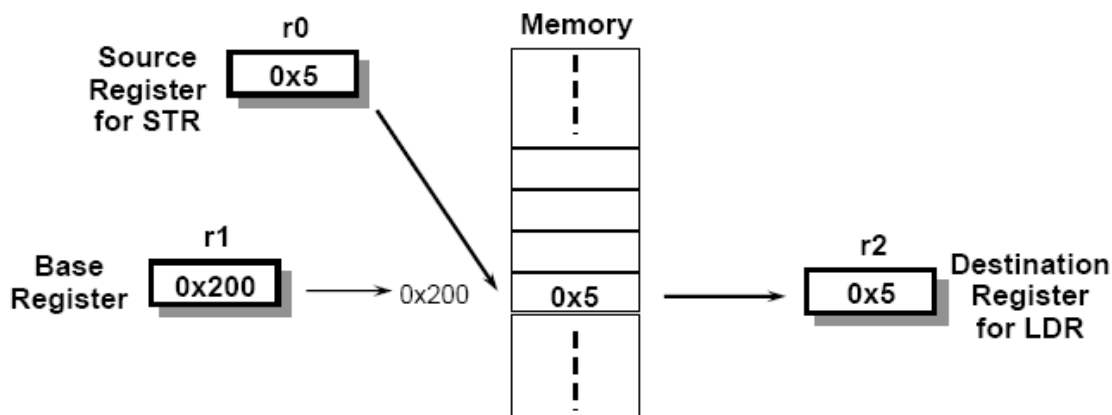


Figura 1

Inoltre, è possibile specificare un *offset* calcolato a partire dal BR, in modo immediato (su 12 bit, quindi in un range da 0 a 4096; è possibile utilizzare le tecniche di shifting, introdotte relativamente agli operandi immediati, per ampliare detto intervallo) oppure specificando un registro che ne contiene il valore. L'offset può essere applicato prima o dopo il trasferimento (*Pre-Indexed/Post-Indexed Addressing*).

- **Pre-Indexed Addressing:** l'istruzione sarà applicata alla locazione puntata dal Base Register incrementato dell'offset. E' possibile incrementare il Base Register al termine dell'istruzione, tramite l'operatore '!'
- **Post-Indexed Addressing:** l'istruzione sarà applicata alla locazione puntata dal Base Register. Successivamente, questo verrà incrementato dell'offset.

Le Figure 2 e 3 illustrano il funzionamento delle due metodologie di indirizzamento.

Load and Store Word or Byte: Pre-indexed Addressing

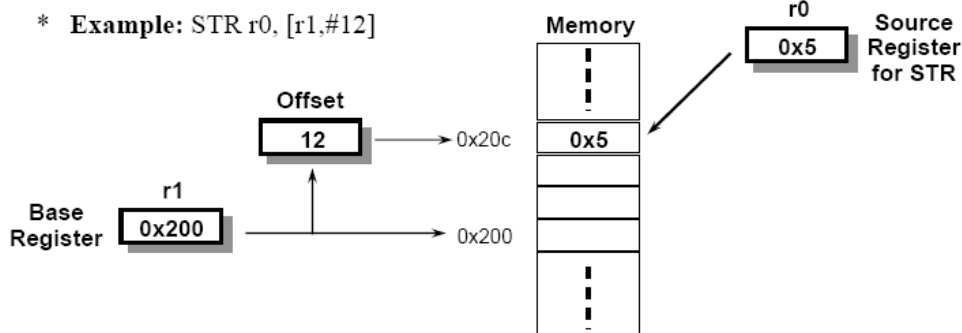


Figura 2

Load and Store Word or Byte: Post-indexed Addressing

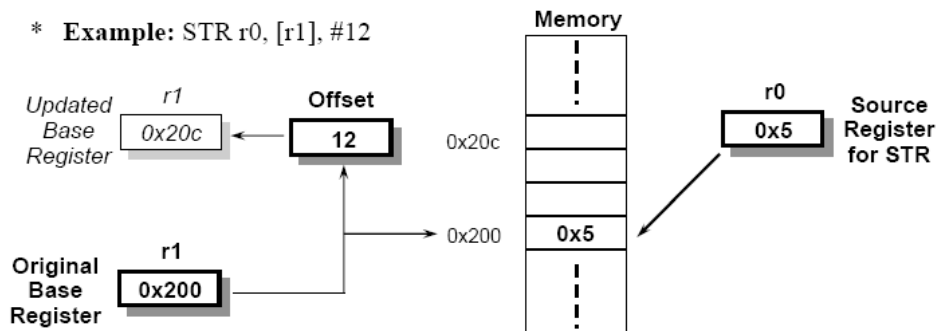


Figura 3

Si noti che in caso di operazioni Load/Store applicate a valori Signed, o di tipo HalfWord, il formato dell'offset è ancora più ridotto (8 bit, range 0-256) e qualora sia fornito mediante registro, a detto registro non sono applicabili operazioni di shifting.

ARM Instruction Set : Endianess

Il processore ARM può accedere ai dati in format Big Endian o Little Endian.

- Big Endian: il byte meno significativo (LSB) del dato è memorizzato nei bit 0-7.
- Little Endian: il LSB è memorizzato nei bit 24-31.

L'Endianess è rilevante solo nel caso in cui i dati siano memorizzati in Word, e successivamente elaborati tramite quantità di dimensioni minori (Byte, Half-Word). La Figura 4 illustra questa problematica.

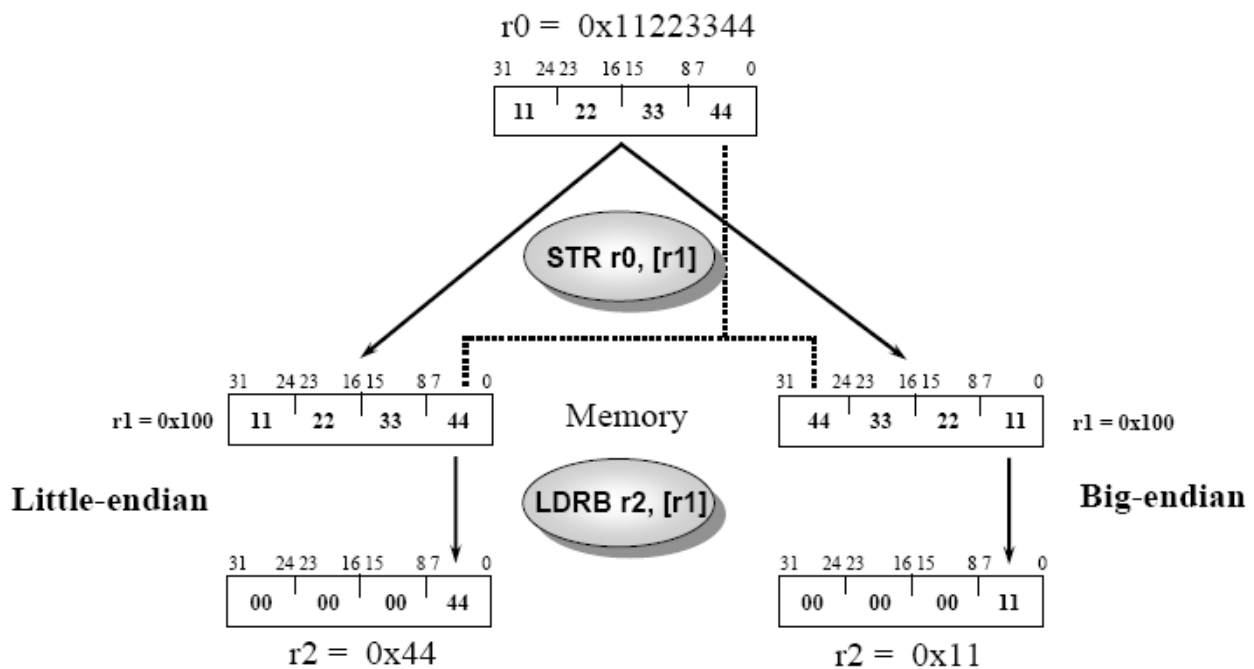


Figura 4

ARM Instruction Set : Trasferimenti A Blocchi

L'assembler ARM fornisce anche delle istruzioni per il trasferimento di blocchi di dati: sono le cosiddette istruzioni "Block Data Transfer" STM, LDM. Queste istruzioni hanno una doppia funzionalità: quella di poter registrare e ripristinare le informazioni di stato del processore (accedendo alla memoria come un vero e proprio *stack*) e quella di incrementare la velocità di trasferimento dei dati.

La struttura dello Stack dell'ARM è funzionalmente identica a quella dell'architettura x86. I limiti dello Stack sono definiti da due registri, un registro base (*Base Pointer, BP*) che punta sempre alla prima locazione dello Stack, ed uno *Stack Pointer (SP)* che punta alla "cima" dello stack. Tradizionalmente, lo Stack si "espande" in senso discendente, assegnando quindi ai valori inseriti (*pushed*) indirizzi di memoria progressivamente decrescenti. L'architettura ARM supporta però anche Stack di tipo *ascendente*, dove gli indirizzi vengono assegnati in modo crescente.

Inoltre, essa consente anche di specificare se lo Stack Pointer si riferisca alla prima locazione libera dello Stack (*Empty Stack*) oppure all'ultima locazione occupata (*Full Stack*). La tipologia di Stack da utilizzare viene specificata dal suffisso delle istruzioni STM, LDM:

- STMFD / LDMFD : Full Descending stack
- STMFA / LDMFA : Full Ascending stack.
- STMED / LDMED : Empty Descending stack
- STMEA / LDMEA : Empty Ascending stack

La Figura 5 illustra il funzionamento delle diverse tipologie di Stack.

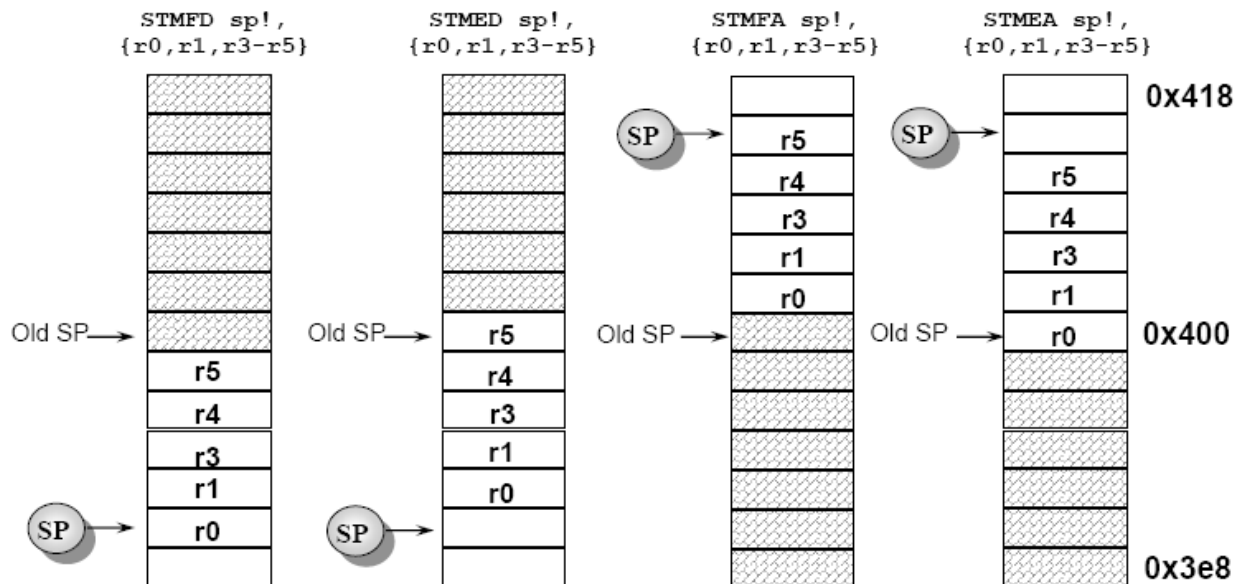


Figura 5

Come nell'Assembler dell'architettura x86, un comune utilizzo dello Stack nell'ARM consiste nel memorizzare il contenuto di uno o più registri prima di effettuare la chiamata ad una procedura, in modo da riservare tali registri come *workspace* temporaneo per la sua esecuzione, e successivamente ripristinarne il contenuto originario al termine della procedura.

```
STMFD sp!, {r0-r12, lr} ; stack all registers
.....                ; and the return address
.....
LDMFD sp!, {r0-r12, pc} ; load all the registers
.....                ; and return automatically
```

ARM Instruction Set : Swap / Swap Byte

L'istruzione di Swap (SWP) implementa un'operazione atomica, composta di un accesso in lettura ed uno in scrittura alla memoria, per scambiare dati in quantità Word o Byte fra memoria e registri. La figura 6 ne illustra il funzionamento.

SWP{<cond>}{B} Rd, Rm, [Rn]

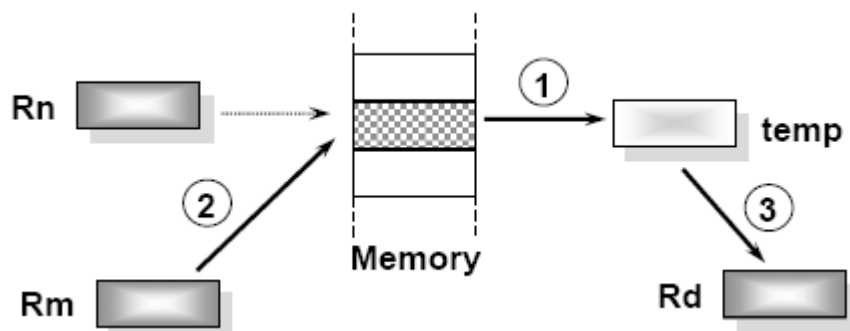


Figura 6

Questa operazione è particolarmente efficace in senso prestazionale.

ARM Instruction Set : Software Interrupt (SWI)

L'istruzione SWI consente al programmatore di eseguire operazioni di tipo privilegiato, modificando lo stato del processore ARM in Supervisor Mode. Nel momento in cui viene eseguita, SWI notifica un'eccezione di tipo Trap (non mascherabile) al vettore delle interruzioni hardware, che conseguentemente imposta il processore in modalità Supervisor, provvede al salvataggio della PSW ed esegue la specifica procedura di gestione dell'eccezione. Al termine dell'esecuzione, viene ripristinato il PC per continuare l'esecuzione del processo chiamante.

Inoltre, il gestore delle interruzioni verifica il contenuto del campo di commento dell'interruzione SWI, in base al quale determina l'indirizzo della procedura da eseguire. Utilizzando questo meccanismo, un sistema operativo può implementare operazioni di tipo privilegiato eseguendo l'istruzione SWI e specificando nel campo di commento una locazione di memoria che contiene la procedura da eseguire.

ARM Instruction Set : IRQ e FIQ

Il processore ARM può gestire due tipi di interrupt: IRQ e FIQ (Fast Interrupt). Questo particolare tipo di interrupt ha una maggiore priorità rispetto agli interrupt normali, quindi in caso di concorrenza di interrupt, una richiesta di tipo FIQ viene servita prima. Inoltre, gli IRQ sono disabilitati per tutta la durata della routine di gestione degli interrupt FIQ.

La maggiore velocità degli interrupt FIQ è realizzata grazie ad un numero di registri del processore ad essi dedicati maggiore rispetto agli interrupt normali. Questo consente di ridurre l'*overhead* temporale dovuto al salvataggio dello stato del processore che avviene prima di ogni procedura di interrupt. La figura 7 illustra il timing degli interrupt.

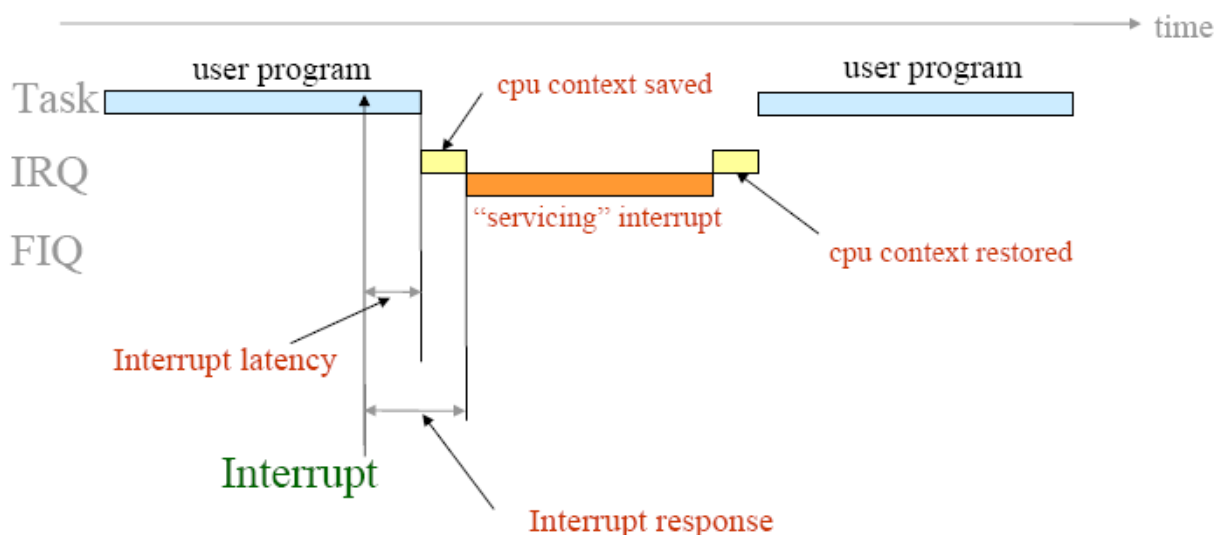


Figura 7

In aggiunta, il vettore degli interrupt di tipo FIQ occupa l'ultima posizione della *vector table*, quindi il gestore delle interruzioni FIQ può essere posizionato nella locazione immediatamente successiva in modo da essere eseguito sequenzialmente, aumentandone la velocità d'esecuzione.

ARM Instruction Set: Istruzioni Thumb

Alcuni processori della famiglia ARM forniscono un supporto aggiuntivo per un set di istruzioni semplificate, detto Thumb Instruction Set. Questi processori sono contrassegnati da una “T” nell’acronimo del nome del modello.

Le istruzioni di tipo Thumb hanno le seguenti caratteristiche:

- Codificate su 16 bit
- Meno potenti delle istruzioni standard
- In numero minore delle istruzioni standard

Di conseguenza, implementare un algoritmo tramite istruzioni Thumb comporta un maggiore numero di istruzioni ed un’esecuzione più lenta, a fronte di un guadagno in termini di memoria e consumo energetico. Per questo motivo, le istruzioni Thumb vengono utilizzate prevalentemente per applicazioni a basso profilo prestazionale.

Il meccanismo di switching fra istruzioni normali e istruzioni Thumb è implementato tramite il bit T all’interno del CPRS. Inoltre, è richiesta una circuiteria addizionale per effettuare la traduzione da istruzioni Thumb a istruzioni ARM standard. Lo schema a blocchi del circuito richiesto è illustrato in figura 8.

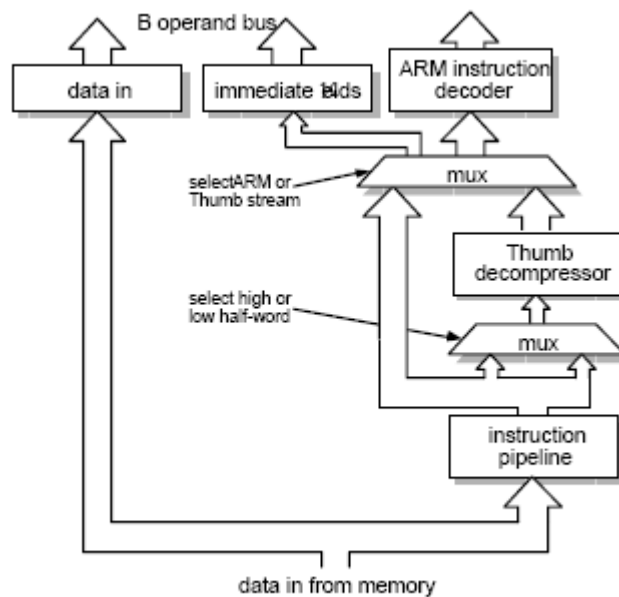


Figura 8

Processore ARM: Sottosistema di Memoria

L'interfaccia del processore ARM con il sottosistema di memoria è costituita da un Bus Dati bidirezionale (DBUS) e un Bus Indirizzi (ABUS), entrambi a 32 bit, e da svariati segnali di controllo utili a specificare i parametri operazionali degli accessi alla memoria (read/write, accessi sequenziali, informazioni di *timing*).

Questa interfaccia di memoria è in grado di gestire memorie di tipo ROM e Static RAM. In questo caso, gli indirizzi di memoria rimangono stabili sull'ABUS per tutta la durata del ciclo di lettura o scrittura, e i moduli di memoria sono direttamente collegati ai Bus. La Figura 9 illustra il modello delle interconnessioni.

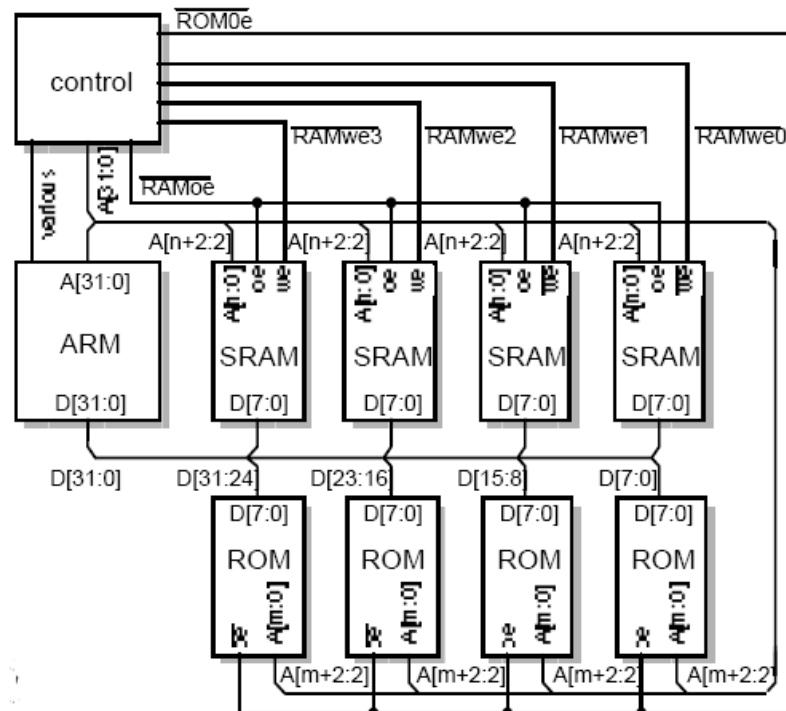


Figura 9

Come possiamo notare, nello schema è presente una circuiteria di controllo, che svolge diversi compiti. Innanzitutto, determina se l'operazione di lettura/scrittura è relativa alla ROM o alla RAM, e successivamente attiva i corrispondenti segnali *read/write*; inoltre, in caso di un'operazione di scrittura relativa alla RAM, è compito del circuito di controllo stabilire quale modulo di RAM attivare, mediante il segnale *RAMwe* ad esso relativo. Infine, il circuito di controllo deve verificare la disponibilità dei dati sul DBUS, e notificarla al processore che potrà quindi procedere con l'esecuzione.

Queste informazioni vengono ricavate dall'unità di controllo in base all'indirizzo presente sull'ABUS. Esso è infatti composto da $[n+2:2]$ bit, ove

- n indica la profondità dei blocchi di memoria;
- 2 bit servono per indirizzare uno specifico modulo di memoria;
- un bit determina il tipo di operazione;
- l'ultimo bit ($A[31]$) determina se l'operazione è relativa alla ROM o alla RAM.

Lo schema logico del circuito di controllo è visualizzato nella Figura 10.

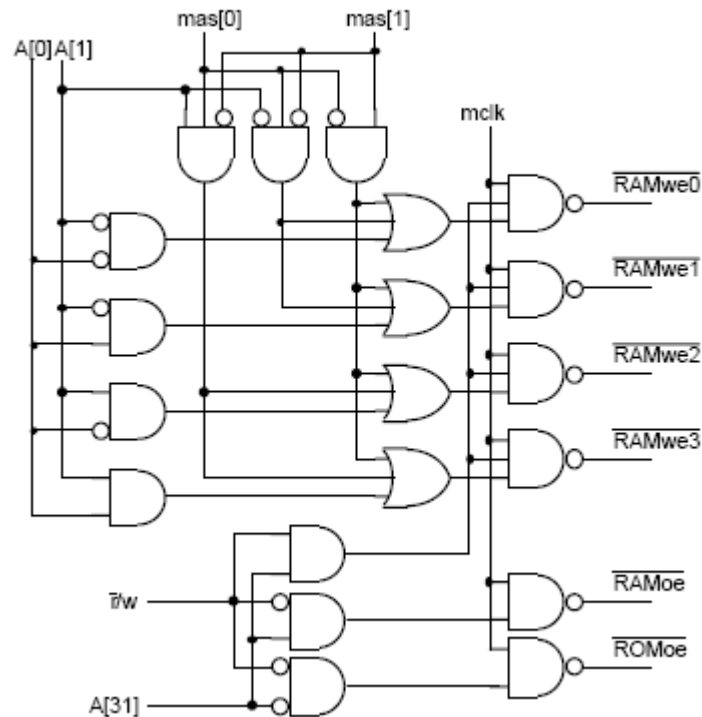


Figura 10

Processore ARM: Stati di Wait

Poiché l'architettura ARM supporta svariati tipi di memorie, è probabile che i loro tempi di accesso siano differenti. Per ottimizzare l'accesso alle memorie senza modificare il clock di sistema, il processore ARM può inserire dei cicli di inattività (*wait*) in modo da sincronizzarsi con i dispositivi di memorizzazione. Mediante un apposito segnale, il processore riceve la notifica di un accesso ad un modulo di memoria con un elevato tempo di risposta, e non riprenderà l'operazione di lettura/scrittura fino alla disabilitazione del segnale.

Processore ARM: Memorie DRAM

La tecnologia DRAM (Dynamic RAM) è attualmente la meno costosa nell'ambito delle memorie a semiconduttori. Esse sono però caratterizzate da tempi di accesso sensibilmente più elevati rispetto alle RAM statiche, quindi il loro utilizzo tende a degradare sensibilmente le prestazioni complessive del sistema. Per ovviare a questo inconveniente, l'architettura ARM fa ricorso alle tecniche di accesso sequenziale alla memoria.

La struttura di un modulo di DRAM è a matrice, come mostrato nella Figura 11.

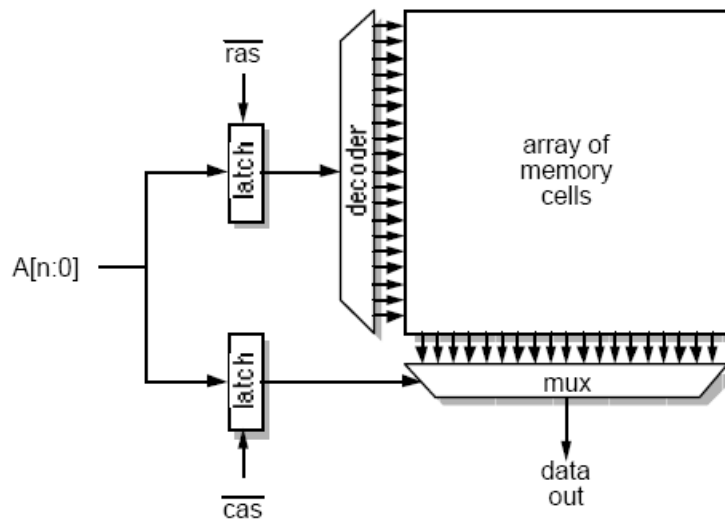


Figura 11

Due segnali (Row Address Strobe, RAS, e Column Address Strobe, CAS) selezionano una cella di memoria all'interno del modulo. Si parla di accesso sequenziale quando i dati richiesti in più accessi consecutivi sono contenuti in celle che giacciono sulla stessa riga. (Si noti che questo tipo di accessi risulta essere pari al 75% degli accessi totali alla memoria). Quando ciò avviene, è possibile ridurre significativamente il tempo di accesso poiché a variare è soltanto l'indirizzo relativo alla colonna (e il rispettivo segnale CAS).

Grazie ad un apposito segnale (*seq*) il processore notifica al modulo di memoria un accesso sequenziale; quindi, il modulo dovrà solo specificare il nuovo segnale CAS, ma potrà mantenere abilitata la riga dell'accesso precedente. Questo comporta un tempo di accesso molto minore, come mostrato nel diagramma dei tempi nella Figura 12 (*N-cycle* identifica un ciclo di accesso standard; *S-cycle* identifica un ciclo di accesso sequenziale).

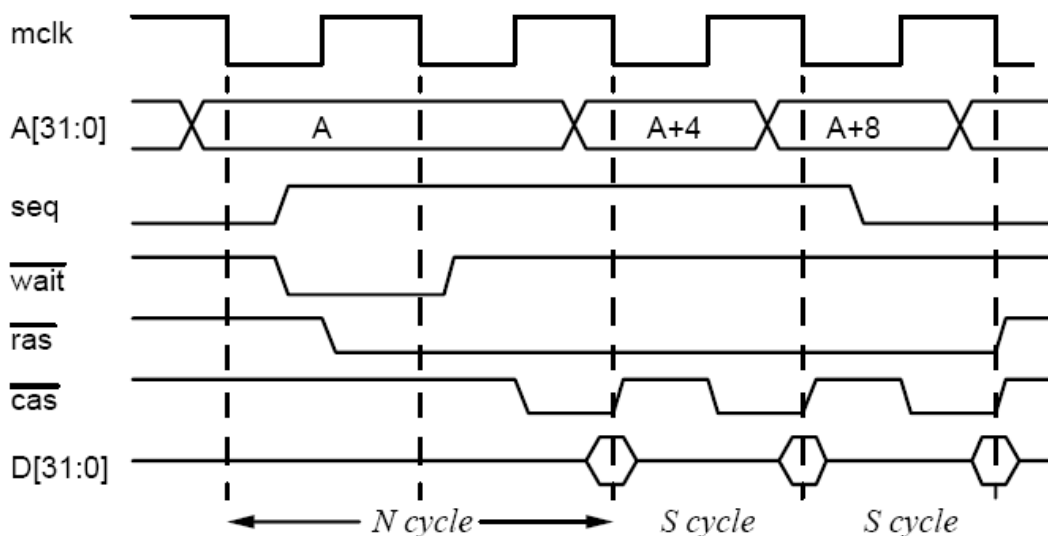


Figura 12

Processore ARM: Architettura di Bus

L'architettura ARM dispone di uno standard per la progettazione dei bus, chiamato AMBA (Advanced Microcontroller Bus Architecture). Questo standard definisce un'architettura con 3 diverse tipologie di Bus:

- AHB (*Advanced High-Performance Bus*) utilizzato per i moduli ad alte prestazioni.
- ASB (*Advanced System Bus*) obsoleto, rimpiazzato dall'AHB nelle architetture più recenti
- APB (*Advanced Peripheral Bus*) utilizzato per le periferiche a basse prestazioni, o come bus secondario (*slave*) per l'AHB.

La Figura 13 mostra una tipica architettura AMBA.

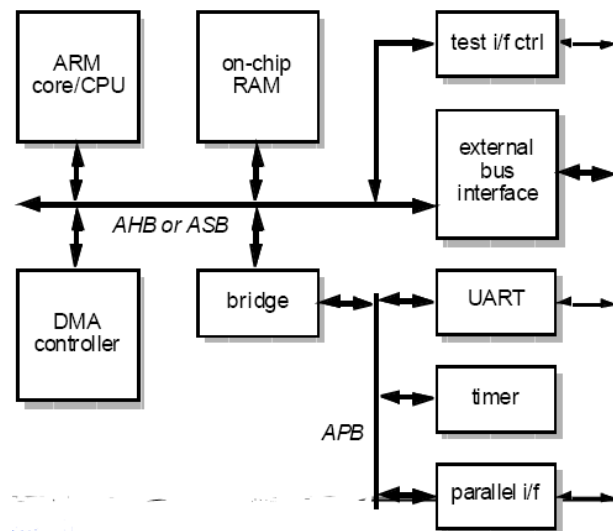


Figura 13

L'arbitraggio del Bus è di tipo centralizzato, gestito da una circuiteria apposita mediante l'utilizzo di segnali di tipo Request (AREQx) e Grant (AGNTx) per ogni dispositivo connesso al Bus. La politica di arbitraggio e le priorità dei dispositivi devono essere definite dal progettista del sistema.

Architettura AMBA: ASB

Entrando nel merito dell'architettura AMBA, analizziamo con maggiore dettaglio la struttura del bus ASB. Esso è utilizzato per connettere il processore con la memoria e altri dispositivi sofisticati ad alte prestazioni. Lo schema del bus ASB è illustrato nella Figura 14.

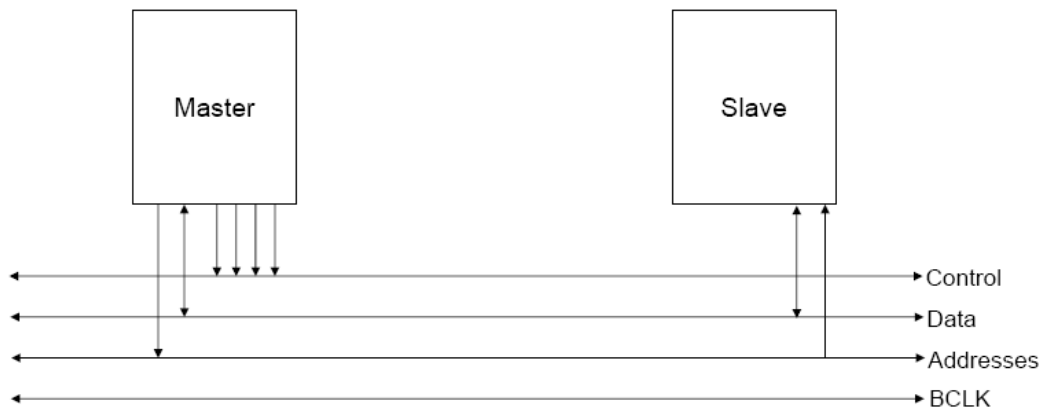


Figura 14

Come possiamo notare, il bus ASB è di tipo sincrono in quanto presenta al suo interno una linea di clock. Questo implica che questo tipo di bus deve essere usato per connessioni brevi, onde evitare ritardi nella trasmissione del clock.

I dispositivi che un ASB può connettere sono di tre tipi: Master, Slave, e il dispositivo di arbitraggio. Le seguenti figure li illustrano in dettaglio.

ASB Master

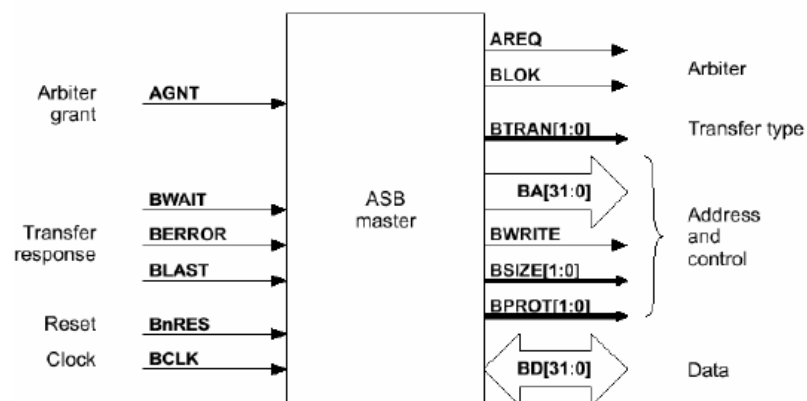


Figura 15

ASM Slave

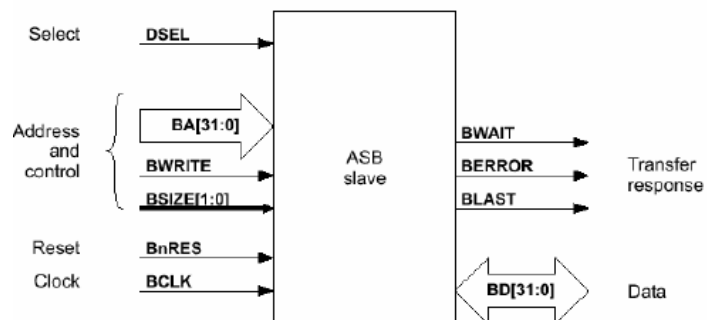


Figura 16

ASB Arbiter

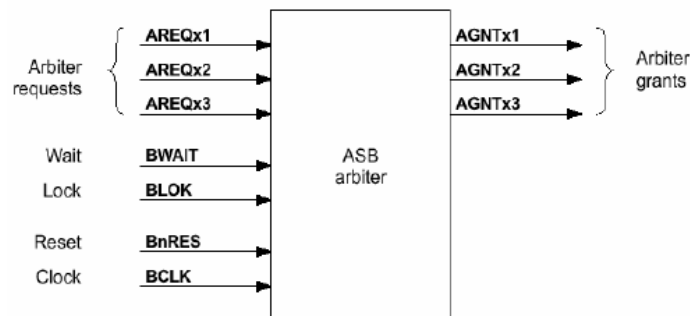


Figura 17

Come possiamo notare, sono presenti numerosi segnali di controllo:

- **BWRITE**: definisce la direzione della transazione
- **BTRAN[1:0]**: definisce il tipo di transazione
- **BPROT[1:0]**: trasmette informazioni relative ai privilegi di accesso
- **BSIZE[1:0]**: definisce la dimensione del trasferimento (Byte, Word, HalfWord...)
- **BLAST**: consente ad un dispositivo *slave* di segnalare la terminazione di un ciclo di burst
- **BWAIT**: consente ad un dispositivo *slave* di richiedere l'inserimento di cicli di *wait*
- **BERROR**: indica una transazione che non può essere completata

Architettura AMBA: APB

Il bus APB è usato per la connessione di dispositivi periferici a basse prestazioni. Esso fornisce un'interfaccia semplificata, che prevede i seguenti segnali di controllo:

- PADDR[n:0]: Bus Indirizzi ($n \leq 32$)
- PRDATA[m:0]: Bus Dati ($m=7, 15, \text{ or } 31$)
- PWRITE: indica la direzione del trasferimento
- PSELx: seleziona un singolo dispositivo periferico
- PENABLE: segnale di timing del dispositivo
- PCLK: APB clock
- PRESETn: APB reset.

Architettura AMBA: AHB

Il bus AHB, come già accennato, sostituisce il bus ASB nei sistemi ad alte prestazioni. Le principali differenze rispetto al bus ASB sono:

- Opera su un singolo fronte di clock
- Supporta un maggiore parallelismo del bus Dati (64 o 128 bit)
- Supporta le split transactions
- Implementa un meccanismo di decodifica e multiplexing degli indirizzi

L'architettura del bus AHB è illustrata nella Figura 18.

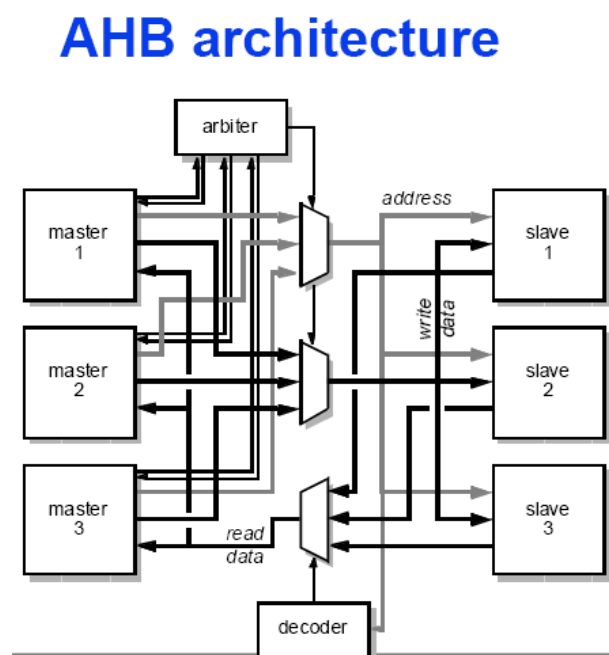


Figura 18