Architettura degli Elaboratori I - B

Formato di Istruzione

ARM

Daniel Riccio/Alberto Aloisio

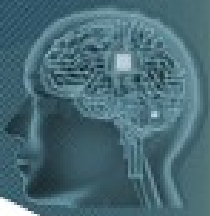
Università di Napoli, Federico II

15 marzo 2018

Rif. Capitolo 6

Digital Design and Computer Architecture-ARM, Harris-Harris, Edition-Morgan Kaufmann.

Il linguaggio e la progettazione



Il primo passo per capire qualsiasi architettura è quello di conoscerne il **linguaggio**. Le parole nel linguaggio di un computer sono chiamate **istruzioni**. il vocabolario di tale linguaggio è chiamato il **set di istruzioni**.

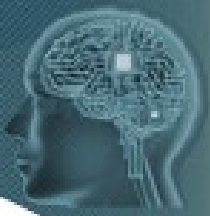
L'hardware di un computer è in grado di elaborare solo 0 e 1, per cui le istruzioni sono codificate come numeri binari in un formato chiamato **linguaggio macchina**.

Le istruzioni indicano sia l'operazione da eseguire, che gli operandi da utilizzare. Gli operandi possono risiedere nella memoria, nei registri o all'interno dell'istruzione stessa.

L'architettura ARM rappresenta ogni istruzione come una parola di 32 bit e segue quattro principi di progettazione fondamentali:

- la regolarità è sinonimo di semplicità;
- il caso più frequente deve essere implementato in modo veloce;
- dimensioni ridotte implicano una maggiore velocità;
- una buona progettazione si basa su buoni compromessi.

Linguaggio macchina



L'**Assembly** è una rappresentazione comprensibile al programmatore del linguaggio macchina di un processore.

L'istruzione più comune è la **somma** di due operandi.

High-Level Code	ARM Assembly Code
`a = b + c;`	`ADD a, b, c`

La prima parte delle istruzioni di somma, **ADD**, è chiamata termine mnemonico ed indica l'operazione da eseguire. L'operazione viene eseguita su **b**, **c**, detti **operandi sorgente**, e il risultato viene scritto in **a**, ovvero l'**operando destinazione**.

High-Level Code	ARM Assembly Code
`a = b - c;`	`SUB a, b, c`

La differenza è analoga alla somma (esempio del principio 1).

RISC vs CISC

Istruzioni di un linguaggio ad alto livello, sono spesso scomposte in istruzioni più semplici nel linguaggio assembly (esempio del principio 2).

High-Level Code	ARM Assembly Code

`a = b + c - d;  // single-line comment`
`/* multiple-line`
`comment */`
	`ADD t, b, c  ; t = b + c`
	`SUB a, t, d  ; a = t - d`

Il set di istruzioni **ARM** rende il caso comune veloce includendo solo semplici istruzioni di uso comune. Il numero di istruzioni viene mantenuto ridotto, cosicché l'hardware necessario per decodificare l'istruzione e suoi operandi sia semplice, piccolo e veloce.

ARM è un **reduced instruction set computer** (**RISC**), a differenza di architetture con molte istruzioni complesse, come x86 di Intel, che è invece considerato un **complex instruction set computer** (**CISC**).

Le istruzioni e i loro operandi

Un'**istruzione** esegue su **operandi**. Gli operandi possono risiedere nei registri, in celle di memoria, in una parte dell'istruzione stessa.

Gli operandi memorizzati come costanti o nei registri sono accessibili rapidamente, ma i registri possono contenere solo una piccola quantità di dati.

Dati aggiuntivi devono essere conservati nella memoria, che è più capiente ma anche più lenta.

Nell'architettura ARM, le istruzioni operano esclusivamente sui registri, quindi i dati residenti nella memoria devono essere spostati in un registro prima di poter essere elaborati.

Usando una combinazione di memoria e registri, un programma può eseguire velocemente, pur accedendo a una grande quantità di dati.

Le istruzioni

L'architettura ARM utilizza istruzioni a **32-bit**. Dato che la regolarità supporta semplicità, la scelta migliore è di codificare tutte le istruzioni come parole che possono essere salvate nella memoria.

Anche se alcune istruzioni possono non richiedere tutti i 32 bit di codifica, gestire istruzioni di lunghezza variabile aumenterebbe la complessità.

Al fine di massimizzare la semplicità sarebbe preferibile anche avere un unico formato per le istruzioni. Tuttavia, questa scelta risulterebbe troppo restrittiva.

Il compromesso scelto da ARM (esempio del principio 4) è di avere **tre** possibili **formati di istruzione**:

- Data-processing;
- Memory;
- Branch.

Istruzioni di memoria

Nel linguaggio assembly del processore ARM, le istruzioni di memoria hanno la seguente sintassi.

LDR	Load a word into a register	*Rd ← mem32[address]*
STR	Store a word from a register to memory	*Rd → mem32[address]*
LDRB	Load a byte into a register	*Rd ← mem8[address]*
STRB	Store a byte from a register to memory	*Rd → mem8[address]*
LDRH	Load a half-word into a register	*Rd ← mem16[address]*
STRH	Store a half-word into a register	*Rd → mem16[address]*
LDRSB	Load a signed byte into a register	*Rd ← SignExtend (mem8[address])*
LDRSH	Load a signed half-word into a register	*Rd ← SignExtend (mem16[address])*

Istruzioni di memoria

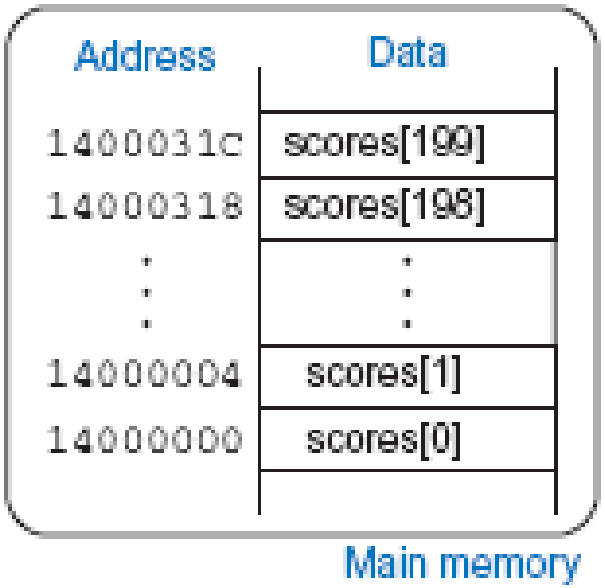
Istruzioni di Memoria

Per facilità di gestione e di accesso, dati simili possono essere raggruppati in un array.

In un array i dati sono conservati in modo sequenziale.

Ogni elemento dell'array è identificato da un numero chiamato suo indice. Il numero di elementi nell'array è detto lunghezza dell'array.

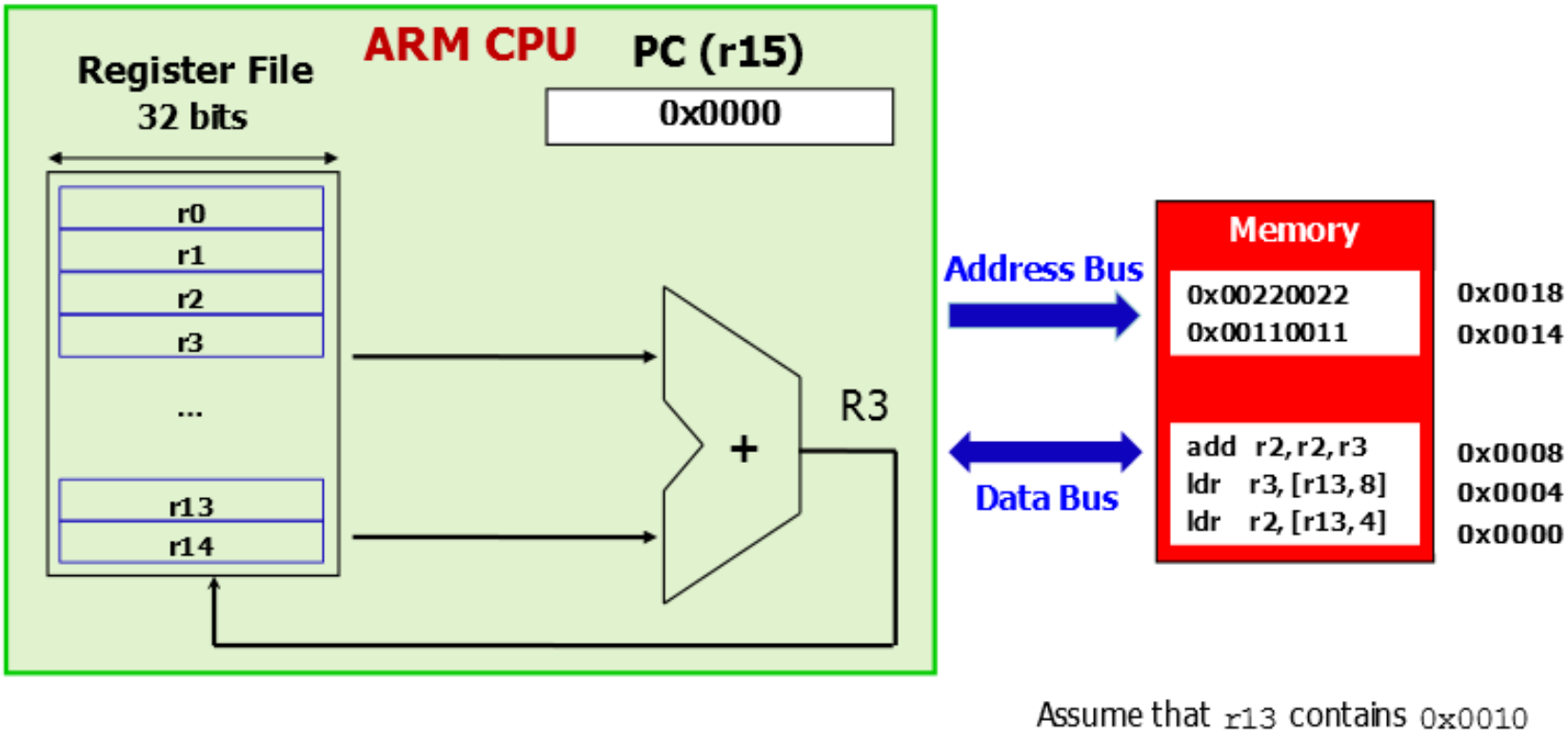
Nell'ARM con una singola istruzione è possibile moltiplicare l'indice, aggiungerlo all'indirizzo di base, e caricare il valore contenuto nella corrispondente locazione.



Esempio: **LDR R3, [R0, R1, LSL #2]**
R1 è scalato (spostato a sinistra di due) quindi aggiunto all'indirizzo di base (R0). Quindi, l'indirizzo di memoria è R0 + (R1 × 4).

Istruzioni di memoria

Un esempio di come il processore accede in memoria:

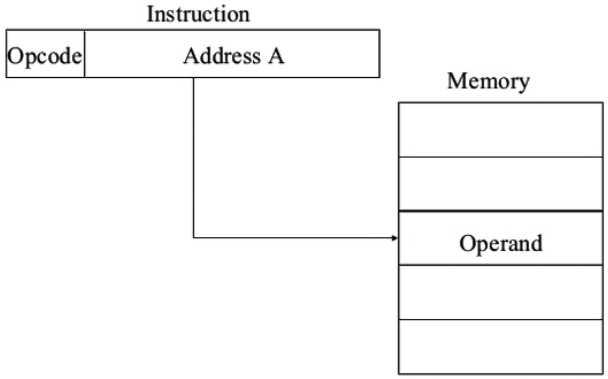


Assume that `r13` contains `0x0010`

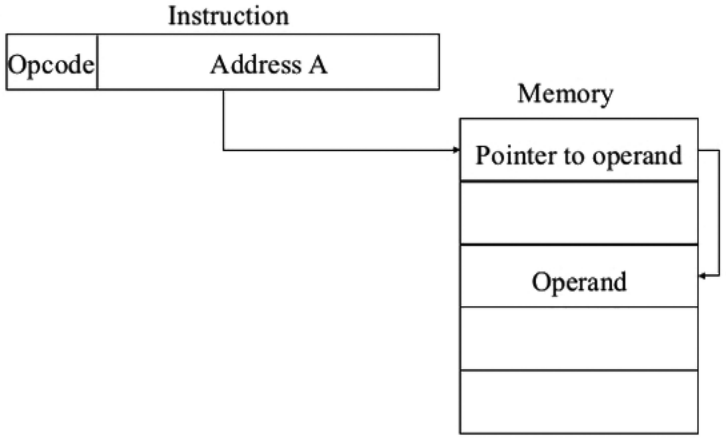
Istruzioni di memoria

Modi di indirizzamento

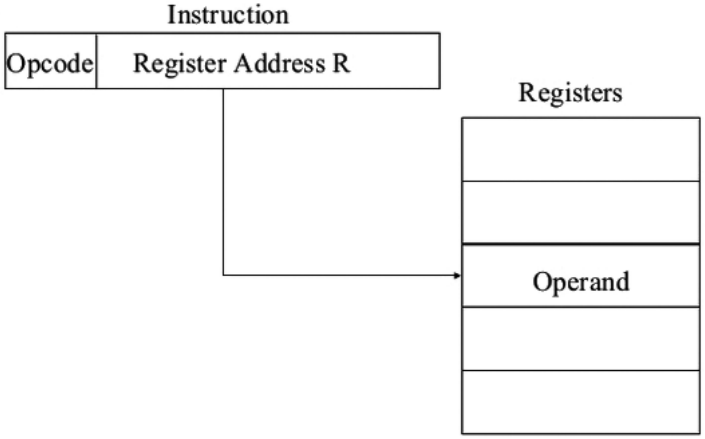
Indirizzamento diretto



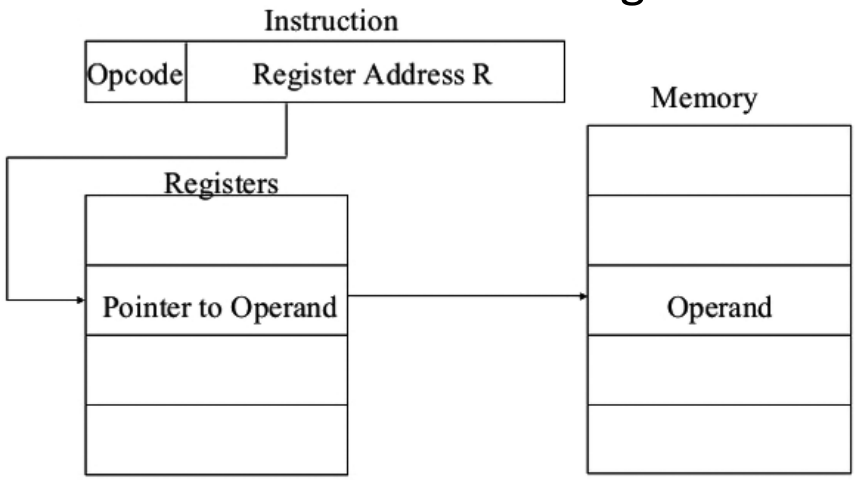
Indirizzamento indiretto



Indirizzamento diretto con registro



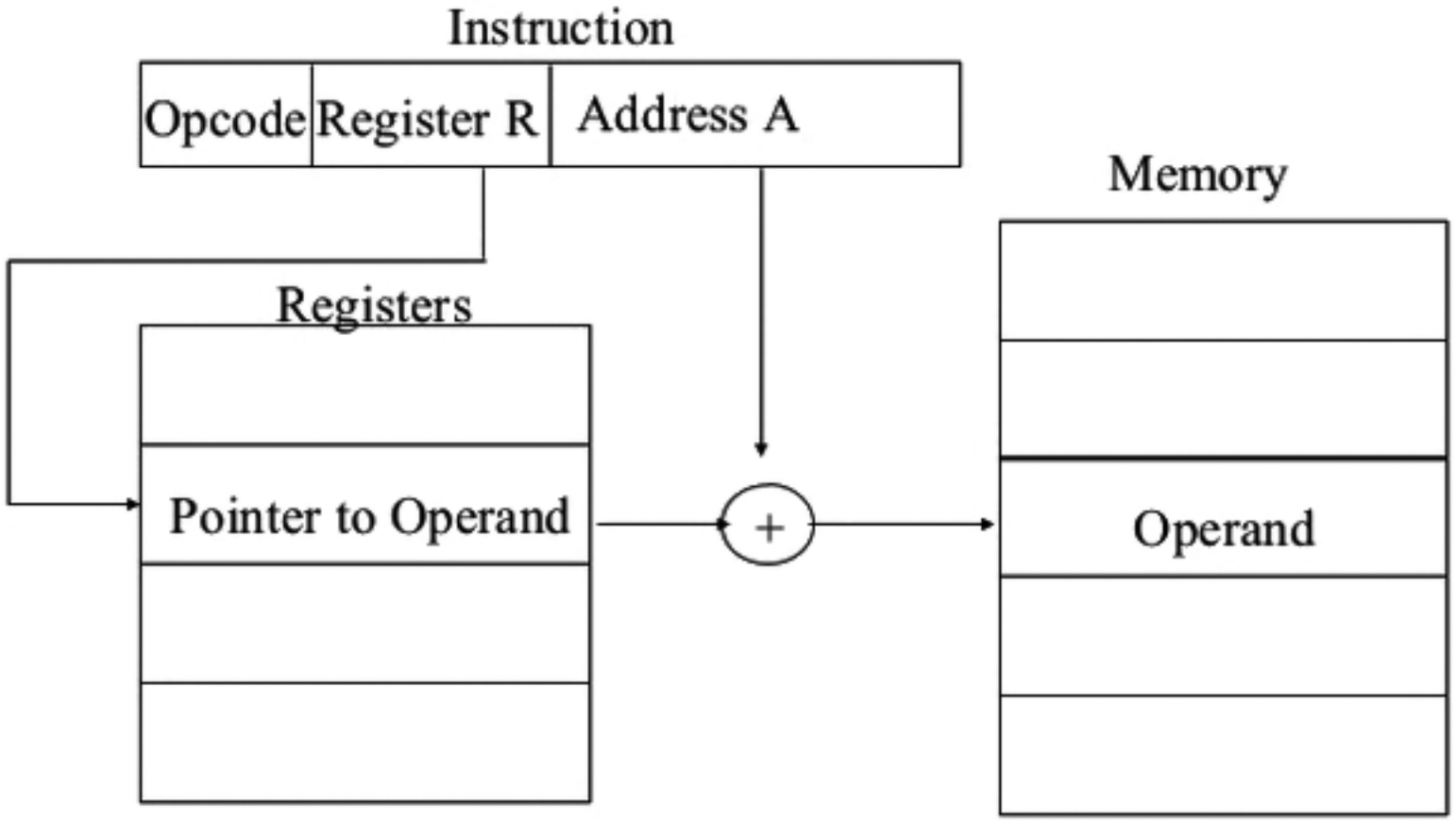
Indirizzamento indiretto con registro



Istruzioni di memoria

Modi di indirizzamento

Indirizzamento con displacement



Istruzioni di memoria (indirizzamento)

In tutti i casi, il registro di base è R1. L'offset può essere il registro R2 (sommato), -R2 (sottratto), o una costante nell'intervallo [0, 4095]. ARM fornisce tre tipi di indicizzazione:

Offset:
L'indirizzo è calcolato sommando o sottraendo un offset al contenuto del registro di base.

Preindex
L'indirizzo viene calcolato come nel caso dell'offset e viene scritto nel registro di base;

Postindex
L'indirizzo corrisponde al contenuto del registro di base;
L'offset viene aggiunto o sottratto al contenuto del registro di base e riscritto nel registro di base

Table 6.4 ARM indexing modes

Mode	ARM Assembly	Address	Base Register
Offset	`LDR R0, [R1, R2]`	R1 + R2	Unchanged
Pre-index	`LDR R0, [R1, R2]!`	R1 + R2	R1 = R1 + R2
Post-index	`LDR R0, [R1], R2`	R1	R1 = R1 + R2

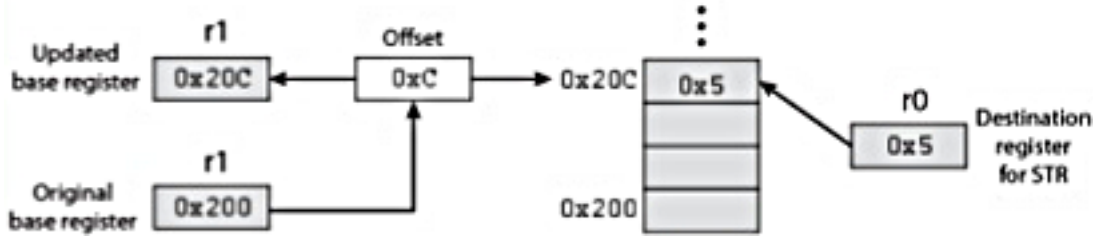
Istruzioni di memoria (indirizzamento)

Esempi di modi di indirizzamento in ARM

```
STRB r0, [r1, #12]
```

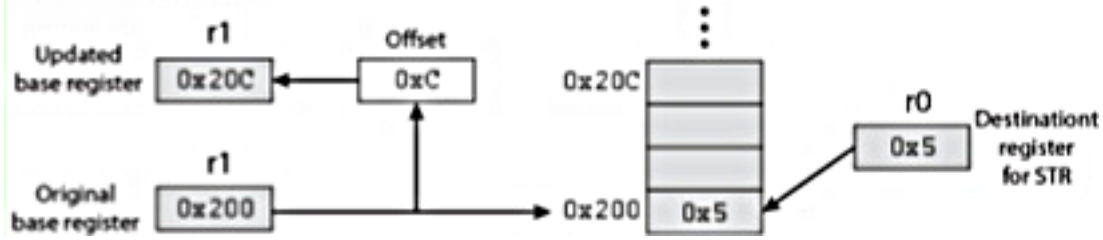
STRB r0, [r1, #12]

```
STRB r0, [r1, #12]!
```



STRB r0, [r1, #12]!

```
STRBv r0, [r1], #12
```



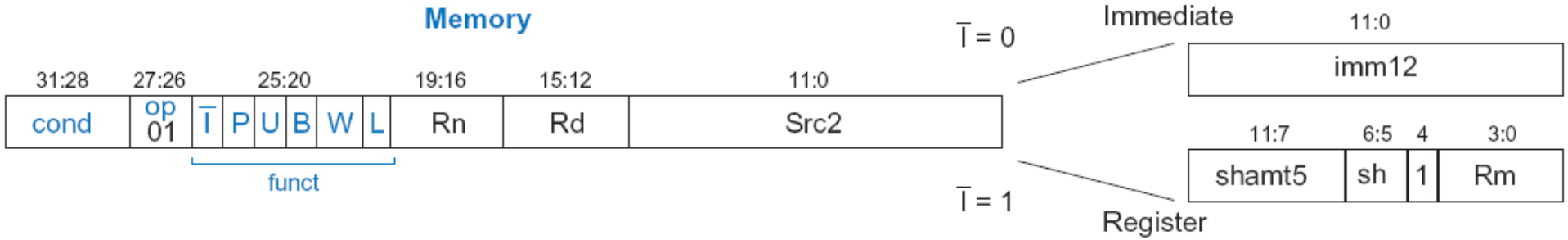
STRB r0, [r1], #12

Istruzioni di Memoria

Bit	Meaning: $\bar{I}$	Meaning: U
0	Immediate offset in Src2	Subtract offset from base
1	Register offset in Src2	Add offset to base

P	W	Index Mode
0	0	Post-index
0	1	Not supported
1	0	Offset
1	1	Pre-index

L	B	Instruction
0	0	STR
0	1	STRB
1	0	LDR
1	1	LDRB



Istruzioni di Memoria

Le istruzioni di **memoria** utilizzano un formato simile a quello delle istruzioni di elaborazione dati, con gli stessi sei campi generali **cond**, **op**, **funct**, **Rn**, **Rd** e **Src2**. Tuttavia, essi assumono una codifica ed un significato diverso.

op è il codice operazione e vale 01_2.

funct è composto da sei bit di controllo:

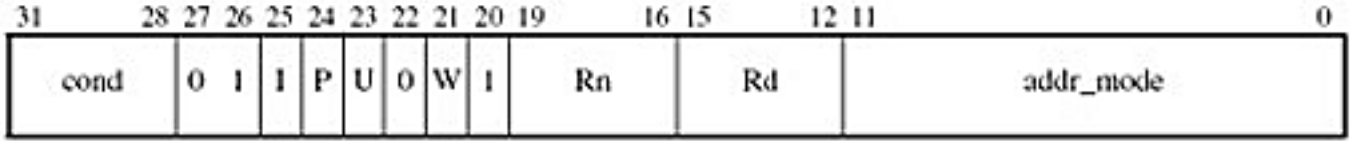
- **I** – l'offset è una costante o un registro;
- **U** – l'offset deve essere aggiunto o sottratto;
- **P** – indicizzazione di tipo pre-index;
- **W** – indicizzazione di tipo writeback;
- **L** – load o store;
- **B** – byte.

Rn è il registro di base e **Rd** è il registro di destinazione in un caricamento o il registro sorgente in un salvataggio.

SRC2 rappresenta l'offset a 12 bit, ovvero una costante senza segno (**imm12**) o un registro (**Rm**) che può essere opzionalmente shiftato di una costante (**shamt5**).

Istruzioni di memoria (LDR)

Istruzione: **LDR**



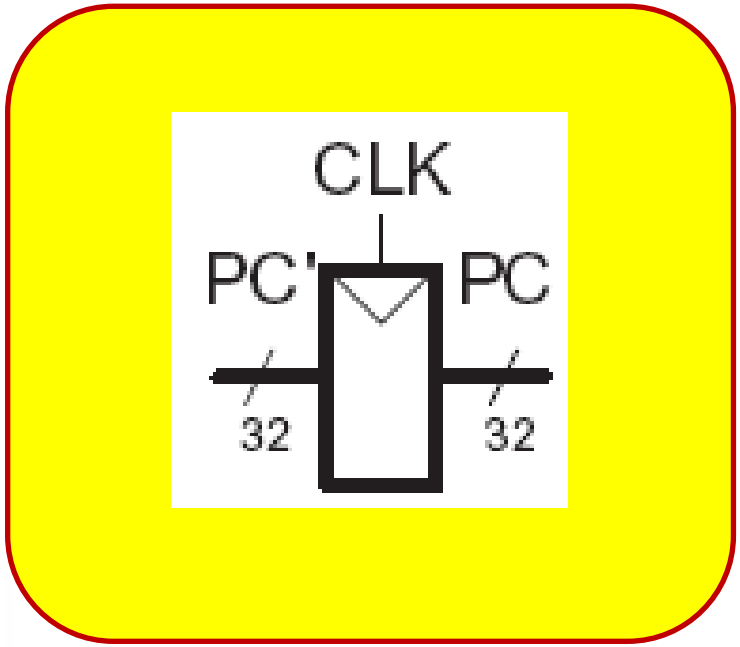
LDR (Load Register) loads a word from a memory address.

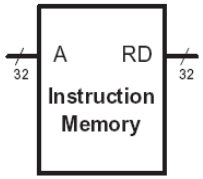
L'istruzione **LDR** legge una parola di 32 bit dalla memoria in un registro. Il modo in cui questa operazione viene effettuata dipende dalla politica di indirizzamento specificata.

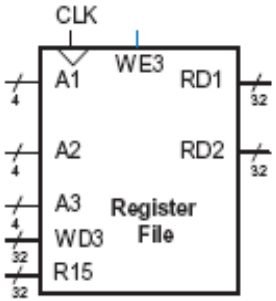
```
// Assume R1 = 0x0000_2000
LDR R0, [R1]  // R0 ← [R1]
LDR R0, [R1, #16] // R0 ← [R1+16]; 0x0000_2010
```

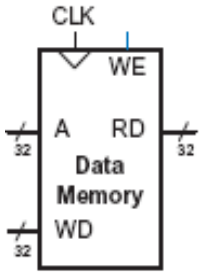
Il datapath

Program Counter: sebbene il PC sia logicamente parte del file registro, esso viene letto e scritto ad ogni ciclo indipendentemente dagli altri registri ed è quindi implementato come un registro autonomo 32-bit. La sua uscita, PC, indica l'istruzione corrente. Il suo ingresso, PC', indica l'indirizzo della successiva istruzione.



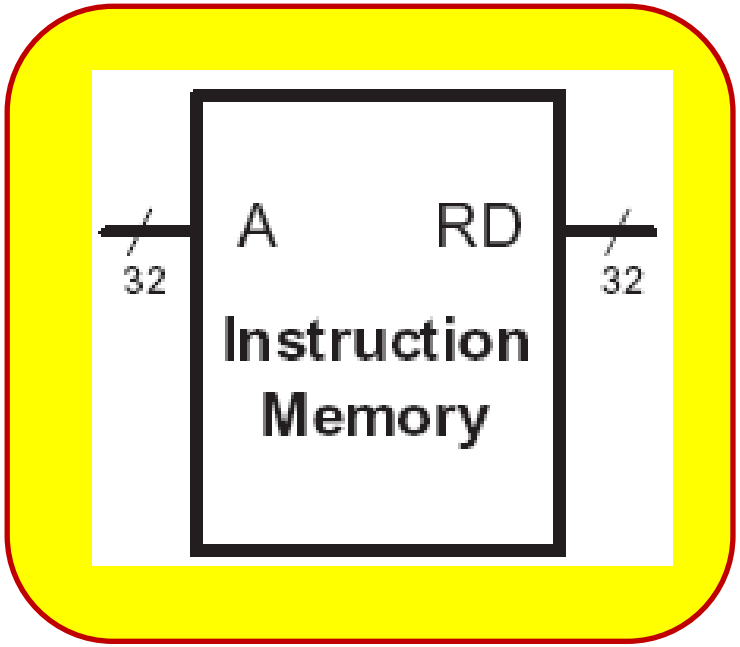


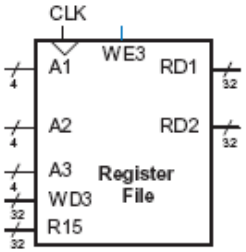


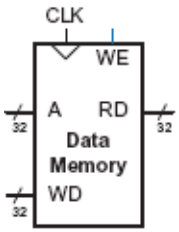


Caratteristiche dei singoli componenti

Instruction Memory: essa ha una singola porta di sola lettura (semplificazione). Prende in input un indirizzo **A** (32 bit) e legge l'istruzione nel registro **RD**.

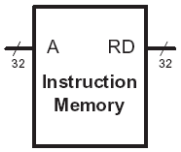


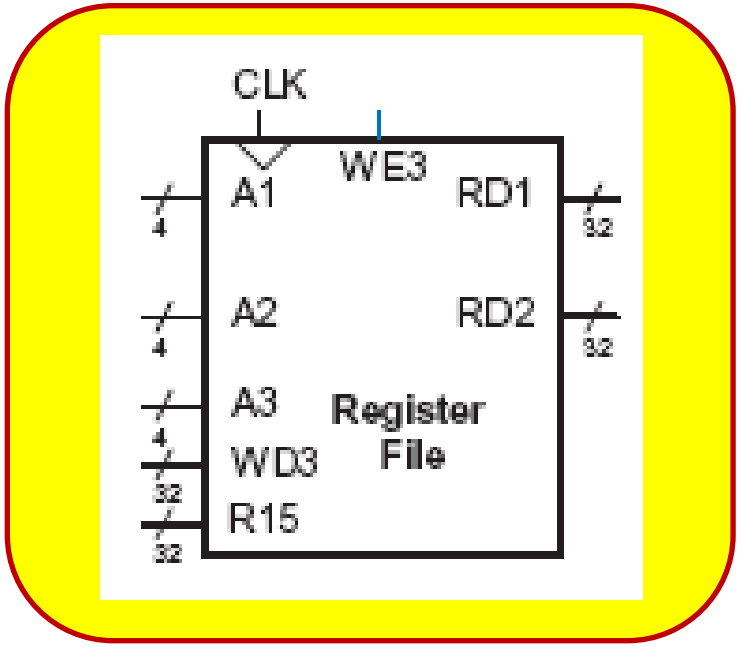


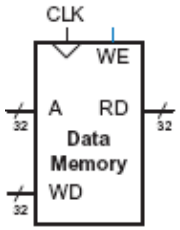


Caratteristiche dei singoli componenti

File Register: consiste di 15 registri di 32 bit ciascuno da **R0** a **R14**, oltre ad un input aggiuntivo **R15** proveniente da **PC**.





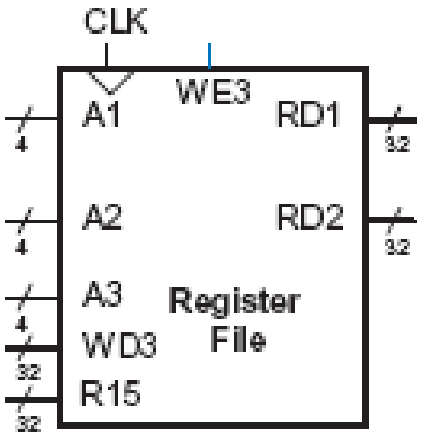


Caratteristiche dei singoli componenti

File Register: consiste di 15 registri di 32 bit ciascuno da **R0** a **R14**, oltre ad un input aggiuntivo **R15** proveniente da **PC**.

Esso ha due porte per la lettura **A1** e **A2** ed una per la scrittura **A3**.

Ciascuna porta di lettura ha un input di 4 bit (2^4=16), che specificano uno dei registri come operando, che viene letto nei registri **RD1** o **RD2**, rispettivamente.



La porta di scrittura ha:

- un indirizzo di 4 bit;
- un elemento di 32 bit **WD3**;
- un segnale di Write Enable (**WE3**);
- il clock.

Se **WE** è 1, il dato è scritto nel registro specificato. Inoltre, viene letto **PC** da **R15**, gli si somma 8 e viene riscritto in **R15**.

Caratteristiche dei singoli componenti

Data Memory: essa ha una singola porta di lettura/scrittura. Quando il segnale **WE** (Write Enable) è attivo, essa scrive i dati nella cella puntata dall'indirizzo **A** durante il fronte alto del clock. Quando il segnale **WE** (Write Enable) è 0, essa legge i dati nella cella puntata dall'indirizzo **A** durante il fronte alto del clock e li pone nel registro **RD**.