

Instruction Set Architecture (2)

Argomenti:

- Metodi di indirizzamento
- Codifica delle istruzioni

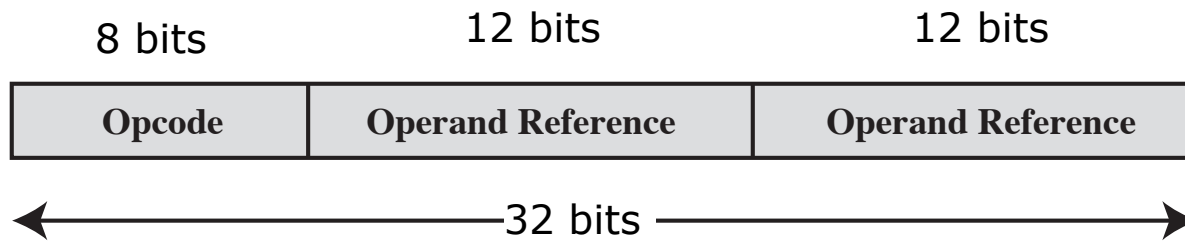
Materiale didattico:

- Capitolo 13 [S15] (esclusa sezione "x86 Addressing Modes")

Metodi di indirizzamento

Rappresentazione operandi

- Ogni istruzione è rappresentata da una sequenza di bit, organizzati in campi.



- Come possiamo indicare dove si trova l'operando?
 - Memoria o registro?
 - A che indirizzo?
 - L'indirizzo è noto quando scrivo il codice?

Metodi di indirizzamento

- Immediato
- Diretto
- Indiretto
- Registro
- Registro indiretto
- Con offset
- Su stack

Indirizzamento immediato



- Il valore da utilizzare è indicato nel campo operando
- Può essere usato per:
 - definire o usare costanti
 - impostare il valore di una variabile
- Valore rappresentato con complemento a due

Indirizzamento immediato

- **Vantaggi:**

- Indirizzamento più semplice
- Non c'è accesso alla memoria, oltre al fetch dell'istruzione

- **Svantaggi:**

- La dimensione del numero è limitata dalla dimensione del campo indirizzi (spesso inferiore ad una word)

ARM: esempio

- Linguaggio ad alto livello

```
int a = 5
```

```
int b = a + 0xD
```

- Registri R0 e R1 contengono a e b, rispettivamente

```
MOV R0, #5
```

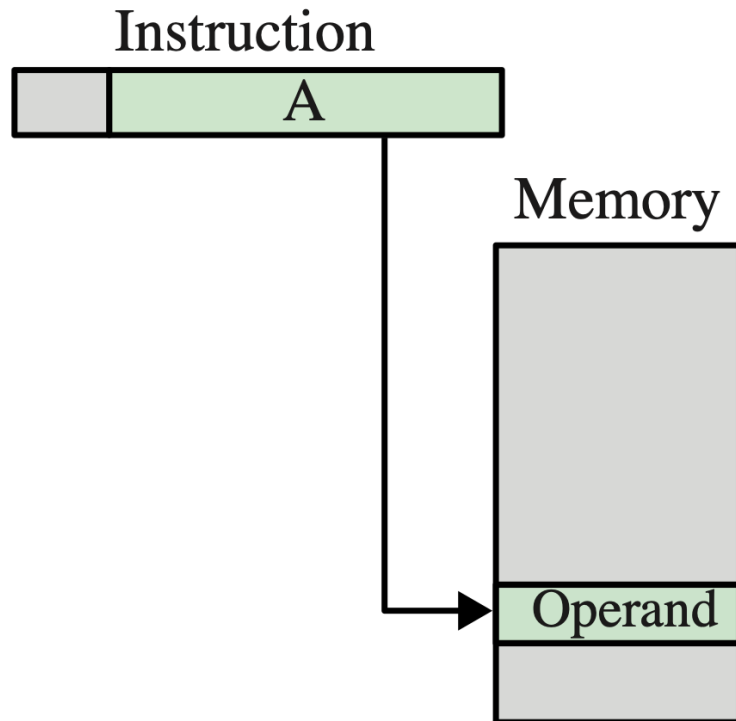
```
ADD R1, R0, #0xD
```

Indirizzamento
immediato → #



- Ci sono dei limiti nel valore numerico utilizzabile

Indirizzamento diretto



Il campo indirizzo A
contiene l'indirizzo
in memoria
dell'operando

Operando = (A)

Indirizzamento diretto (2)

- Utilizzato frequentemente nelle prime generazioni di computer
- **Vantaggi**
 - Richiede 1 accesso di memoria oltre al fetch dell'istruzione
 - Veloce da calcolare
- **Svantaggi:**
 - Spazio limitato di indirizzamento: $2^{|A|}$ dove $|A|$ è la lunghezza in bit di A

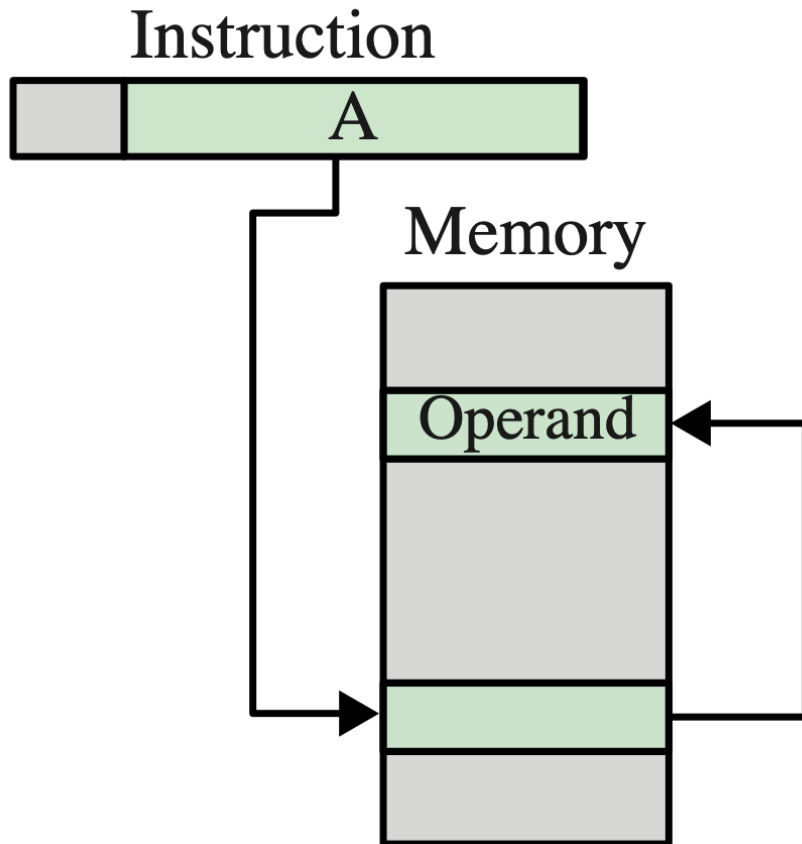
Esempio

- L'indirizzamento diretto **non è possibile in ARM**
 - Ogni istruzione ARM richiede 1 word, non sufficiente per contenere un indirizzo di memoria (1 word)
- Vogliamo copiare il word contenuto all'indirizzo 0x00AA nel registro R0

MOV R0, (0x00AA) (non è un'istruzione ARM!)

Indirizzamento
diretto

Indirizzamento indiretto



Il campo indirizzo A contiene l'indirizzo in memoria di una word contenente l'indirizzo dell'operando

$$\text{Operando} = ((A))$$

Indirizzamento indiretto (2)

- **Vantaggi:**

- Ampio spazio di indirizzamento: con word di n bits, lo spazio indirizzabile è 2^n (n è in genere 32/64 bit)

- **Svantaggi:**

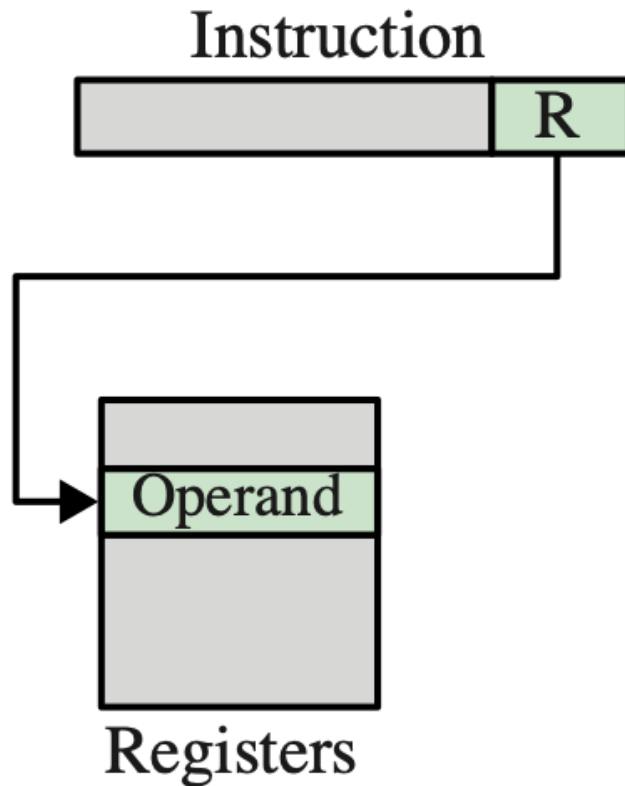
- Richiede 2 accessi in memoria oltre al fetch dell'istruzione
 - Per ottenere l'indirizzo
 - Per ottenere il valore
- È possibile una cascata di indirizzamenti indiretti:
 - Operando = ((... (A) ...))
 - Richiede 3 o più accessi in memoria

Esempio

- L'indirizzamento indiretto **non è possibile in ARM**
 - Ogni istruzione ARM richiede 1 word, non sufficiente per contenere un indirizzo di memoria (1 word)
- Vogliamo copiare il word il cui indirizzo è contenuto all'indirizzo 0x00AA nel registro R0
MOV R0, ((0x00AA)) (non è un'istruzione ARM!)

Indirizzamento
indiretto

Diretto di registro



Il campo indirizzo R
indica un registro
che contiene
l'operando
 $\text{Operando} = (R)$

Diretto di registro (2)

- **Vantaggi:**

- Sono richiesti pochi bit per il campo indirizzi (circa 4-5 bit)
- No accessi alla memoria oltre al fetch dell'istruzione

- **Svantaggi:**

- Lo spazio degli indirizzi è molto limitato (numero di registri)

ARM: esempio

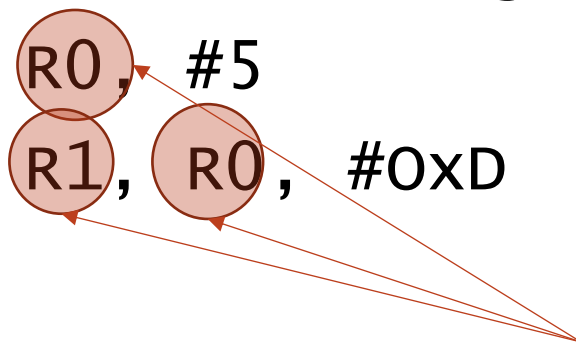
- Linguaggio ad alto livello

```
int a = 5
```

```
int b = a + 0xD
```

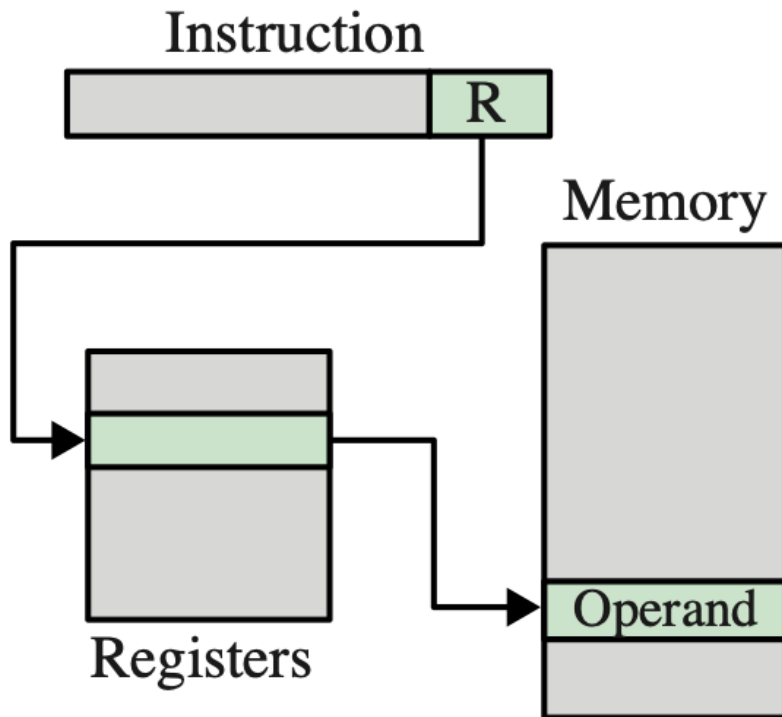
- Registri R0 e R1 contengono a e b, rispettivamente

```
MOV R0, #5  
ADD R1, R0, #0xD
```



Indirizzamento
di registro

Indiretto di registro



Il campo indirizzo R indica un registro che contiene l'indirizzo in memoria dell'operando

Operando = ((R))

Indiretto di registro (2)

- **Vantaggi:**

- Con word di n bits, lo spazio indirizzabile è 2^n

- **Svantaggi:**

- Richiede 1 accesso in memoria, oltre al fetch dell'istruzione
 - 1 accesso in più rispetto al diretto di registro
 - 1 accesso in meno rispetto l'indirizzamento indiretto.

ARM: esempio

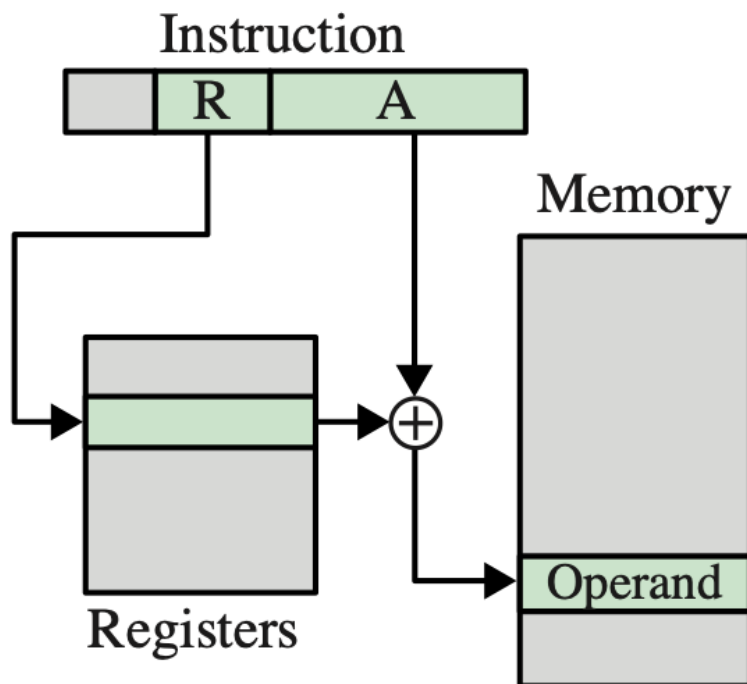
- Linguaggio ad alto livello (esempio precedente)
`int a = 5`
- Supponiamo che l'indirizzo in memoria di **a** sia nel registro R2.

`LDR R0, [R2]`

Indirizzamento
indiretto di registro



Indirizzamento con offset



L'istruzione ha due campi indirizzo: A e R.

Il campo indirizzo R indica un registro che contiene un valore che viene aggiunto ad A per calcolare l'indirizzo in memoria dell'operando

$$\text{Operando} = ((R) + A)$$

Indirizzamento con offset (2)

- L'istruzione ha due campi indirizzo (almeno uno esplicito)
 - Un campo con valore A, usato direttamente
 - Un campo con il registro R
- Varianti dell'indirizzamento con offset:
 - Indirizzamento registro base
 - Indirizzamento relativo
 - Indirizzamento indice

Indirizzamento con registro base

- Il registro R contiene un indirizzo in memoria e il campo indirizzo A contiene lo spostamento (offset) da questo indirizzo
 - Il riferimento al registro può essere esplicito o implicito
 - Sfrutta la località in memoria per risparmiare bit

ARM: indirizzamento con registro base

- Supponiamo di voler copiare in R1 il valore all'indirizzo di memoria 0x1100AABB
- Il registro base R0 contiene il valore 0x1100AA00
- L'offset è quindi 0xBB
- Istruzione ARM: `LDR R1, [R0, #0xBB]`

Indirizzamento (auto)relativo

- Il registro R è il **program counter** (PC)
 - Il valore di R può essere implicito
 - Il valore A è in complemento due (quindi può essere positivo o negativo)
 - L'indirizzo effettivo è uno spostamento rispetto al PC
- Sfrutta il concetto di località: permette di risparmiare bit nel campo indirizzo dell'istruzione.

ARM: Esempio

- Si ottiene tramite indirizzamento con registro base:

`LDR R0, [PC, #4]`

- A livello di linguaggio macchina, l'indirizzamento relativo viene usato nell'istruzioni di salto B
 - A livello di linguaggio assembly, B usa l'indirizzamento immediato che viene convertito in autorelativo dall'assemblatore

Indirizzamento indice

- Il campo indirizzo contiene un indirizzo in memoria mentre il registro uno spostamento da questo indirizzo.
 - R è chiamato registro indice
 - Il contenuto di R e A è l'opposto dell'indirizzamento con registro base.
- Viene utilizzato per iterare un'operazione: e.g. leggere un vettore

Indirizzamento indice: autoindicizzazione

Autoindicizzazione

- Automaticamente aumenta o decrementa il contenuto del registro indice

Operando = (A + (R))

(R) = (R) + 1

Indirizzamento indice indiretto

- E' possibile unire indirizzamento di registro con indirizzamento indiretto
- **Pre-indicizzazione**
 - L'indice è aggiunto prima dell'indirettezza
 - Operando = $((A + (R)))$
- **Post-indicizzazione**
 - L'indice è aggiunto dopo l'indirettezza
 - Operando = $((A) + (R))$

ARM: Esempio

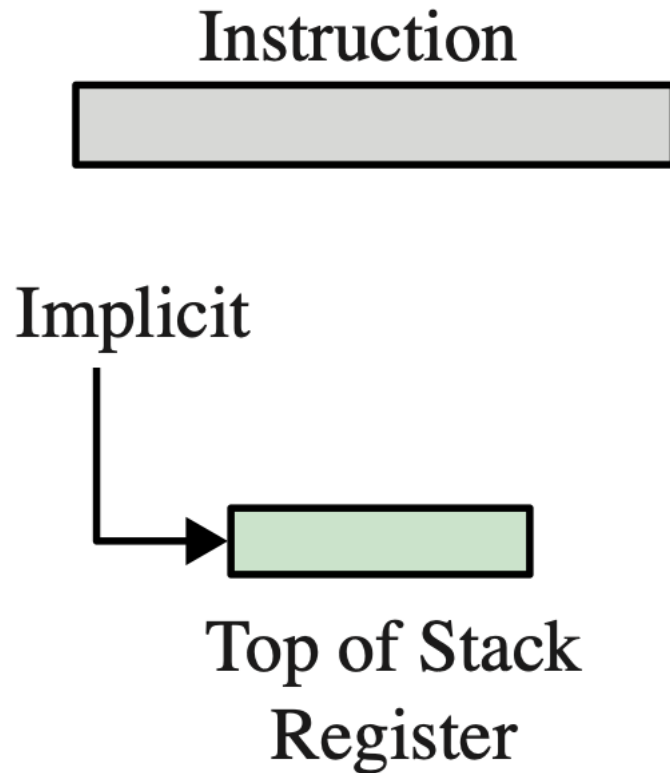
- L'indirizzamento indice **non è possibile in ARM:**
 - Ogni istruzione ARM richiede 1 word, non sufficiente per contenere un indirizzo di memoria (1 word)
- In ARM esiste una variante dell'indirizzamento indice dove l'indirizzo base e l'offset sono mantenuti in due registri
 - Permette l'autoindicizzazione

ARM: Esempio

Esempi di istruzioni:

- `LDR R1, [R0, R1]` → copia in R1 la word in memoria all'indirizzo $R0+R1$.
- `LDR R1, [R0, #1]` → copia in R1 la word in memoria all'indirizzo $R0+1$
- `LDR R1, [R0, #1]!` → calcola $R0=R0+1$ e copia in R1 la word in memoria all'indirizzo R0 (nuovo valore di R0)
- `LDR R1, [R0], #1` → copia in R1 la word in memoria all'indirizzo R0 e poi calcola $R0=R0+1$

Indirizzamento con stack



L'operando si trova nella testa di uno stack

L'indirizzamento è implicito

Operando = Top of Stack

ARM: Esempio

- La testa dello stack viene mantenuta nel registro SP (Stack Pointer)
- L'inserimento e la rimozione di un elemento si realizzano con le istruzioni assembly POP e PUSH
- PUSH {R0, R2-R4} → inserisce nello stack il contenuto dei registri indicati, nell'ordine R4, R3, R2, R0
- POP {R0, R2-R4} → rimuove i primi elementi dello stack e copia nei registri indicati, nell'ordine R0, R2, R3, R4

Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	$\text{Operand} = A$	No memory reference	Limited operand magnitude
Direct	$EA = A$	Simple	Limited address space
Indirect	$EA = (A)$	Large address space	Multiple memory references
Register	$EA = R$	No memory reference	Limited address space
Register indirect	$EA = (R)$	Large address space	Extra memory reference
Displacement	$EA = A + (R)$	Flexibility	Complexity
Stack	$EA = \text{top of stack}$	No memory reference	Limited applicability

Notazione:

- A = contenuto del campo indirizzo di un'istruzione
- R = contenuto del campo indirizzo di un'istruzione che si riferisce ad un registro
- EA = indirizzo effettivo (Effective Address) della locazione contenente l'operando
- (X) = contenuto della locazione di memoria X o del registro X

Codifica delle istruzioni

Formati delle istruzioni

- I bit che codificano un'istruzione sono raggruppati in campi
- I campi devono includere:
 - Tipo di istruzione (**opcode**)
 - Tipo di indirizzamento di ogni operando
 - Indirizzo di ogni operando
- Nella stessa architettura, possono venire usati formati diversi per tipi diversi di istruzioni

Lunghezza delle istruzioni

- La lunghezza delle istruzioni dipende da:
 - Dimensione della memoria
 - Organizzazione della memoria
 - Struttura della memoria
 - Complessità del processore
 - Velocità del processore
 - Deve essere uguale (o un multiplo di) alla lunghezza di una word (unità minima in memoria)
 - Deve essere un multiplo della lunghezza di un carattere (in genere 8 bit)

Allocazione dei bit

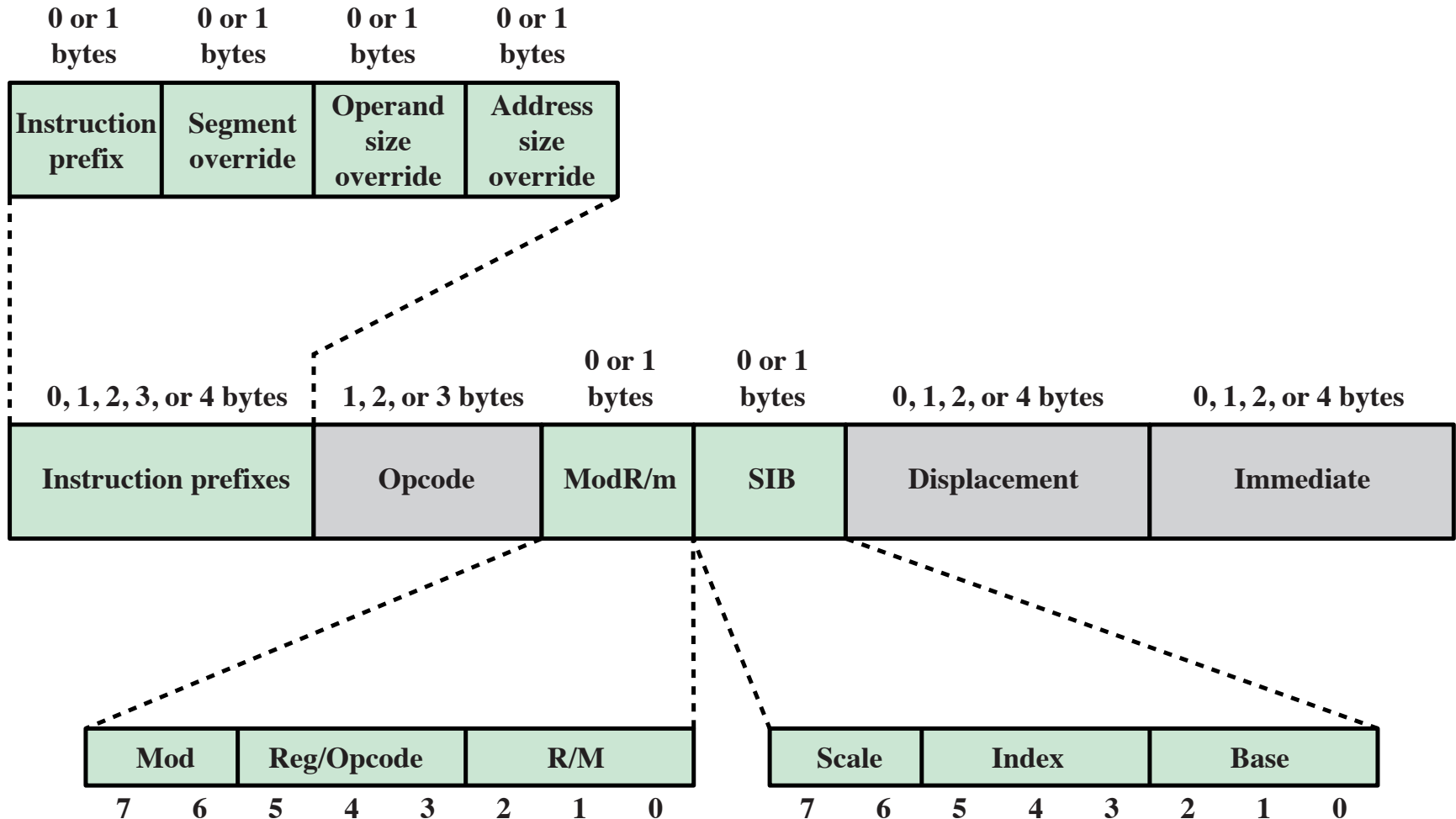
La codifica di un'istruzione è divisa in campi.
Le dimensioni dei campi dipende da:

- Numero di modi di indirizzamento
- Numero di operandi
- Registri vs memoria
- Numero di gruppi di registri
- Intervallo degli indirizzi
- Granularità degli indirizzi

Istruzioni a lunghezza variabile

- In alcune architetture (e.g., x86) le istruzioni hanno lunghezza diversa
- Permette di aumentare il numero di istruzioni (opcode) e aumentare la flessibilità degli indirizzamenti
- Aumenta la complessità del processore per:
 - Riconoscere istruzioni di lunghezza diversa
 - Eseguire istruzioni molto più articolate
- Possono essere richiesti più fetch per leggere un'istruzione
 - Istruzioni più lunghe richiedono più fetch
 - Le istruzioni più frequenti vengono codificate con sequenze corte di bit
 - Le istruzioni più rare vengono codificate con sequenze di bit più lunghe

Codifica nell'architettura x86



Istruzioni a lunghezza fissa

- In alcune architetture (e.g., ARM) le istruzioni hanno lunghezza fissa (e.g. 32 bit)
- Comporta ad una riduzione del tipo di operazioni e di indirizzamenti
- Semplifica la complessità del processore:
 - Istruzioni di ugual lunghezza
 - Semplicità di interpretazione

Codifica nell'architettura ARM

	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
data processing immediate shift	cond			0 0 0			opcode			S	Rn				Rd				shift amount				shift		0	Rm						
data processing register shift	cond			0 0 0			opcode			S	Rn				Rd				Rs				0	shift		1	Rm					
data processing immediate	cond			0 0 1			opcode			S	Rn				Rd				rotate				immediate									
load/store immediate offset	cond			0 1 0			P	U	B	W	L	Rn				Rd				immediate												
load/store register offset	cond			0 1 1			P	U	B	W	L	Rn				Rd				shift amount				shift		0	Rm					
load/store multiple	cond			1 0 0			P	U	S	W	L	Rn				register list																
branch/branch with link	cond			1 0 1			L	24-bit offset																								

S = For data processing instructions, signifies that the instruction updates the condition codes

S = For load/store multiple instructions, signifies whether instruction execution is restricted to supervisor mode

P, U, W = bits that distinguish among different types of addressing_mode

B = Distinguishes between an unsigned byte (B==1) and a word (B==0) access

L = For load/store instructions, distinguishes between a Load (L==1) and a Store (L==0)

L = For branch instructions, determines whether a return address is stored in the link register

Esempio

Istruzione: ADD R0, R1, #0x104

Codifica (hex): E2810F41

Codifica (bin):

1110	0010	1000	0001	0000	1111	0100	0001
------	------	------	------	------	------	------	------

Cond: esegui sempre

Classe: Data processing immediate

Tipo: ADD

Modifica registro di stato: no

Registro sorgente: R1

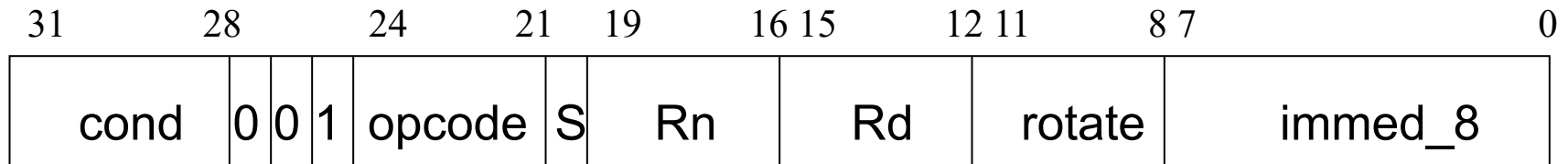
Registro destinazione: R0

Valore immediato: 0x104

Codifica di valori immediati

- Esempio: `ADD R0, R1, #0x104`
- 12 bit a disposizione per l'operando immediato, con la seguente struttura:
 - 8 bit (bit c) definiscono un valore c ($0 \leq c \leq 255$);
 - 4 bit (bit r) specificano una rotazione verso destra di $2r$ posizioni

Codifica di valori immediati (2)



$0x00 \leq c \leq 0xFF$

$0 \leq r \leq 15$

$0 \leq 2r \leq 30$

$\#<\text{immediato}> = c >>_{\text{rot}} 2r$

r

c

- Immediato valido: $\#0x104$ ($c=0x41$, $r=15$)
- Immediato non valido: $\#0x102$
 - ($\#0x102 = 1.0000.0010$ non può essere ottenuto con un valore da 8 bit ruotato un numero pari di posizioni)