Architettura di un elaboratore

Argomenti:

- Instruction Set Architecture
- Caratteristiche di un istruzione
- Tipi di istruzioni

Materiale didattico:

- Capitolo 12 (escluse sezioni "x86 Data Types", "x86 Operation Types")

Introduzione

- L'architettura di un elaboratore è la parte visibile ad un programmatore
- L'architettura include tutto quello che serve ad un programmatore per controllare una macchina
  - Vari tipi di dato
  - Accesso ai dati: memoria, registri
  - Insieme di istruzioni
  - …
- Un'architettura è formalmente definita dall'Instruction Set Architecture (ISA)

Introduzione (2)

- Usiamo l'architettura ARM come caso di studio
  - Altre architetture x86, MIPS,…
- Architetture diverse condividono elementi comuni
  - Assomigliano a dei dialetti regionali, più che a lingue indipendenti

Costruire un programma

```
//Codice C

int a = 10;
int b = 11;
int c = 12;
int d;


int main(){
    d = (a+b)*c;
    return d;
}
```

Compilazione (su architetture ARM):

```
gcc -o sum sum.c
```

File eseguibile sum (in esadecimale)

```
…
0009    0006    3024    e59f
2000    e593    3020    e59f
3000    e593    2003    e082
3018    e59f    3000    e593
0293    e002    3010    e59f
2000    e583    ff1e    e12f
…
```

Istruzioni macchina

Ogni sequenza di 4 byte (8 cifre hex) rappresenta un'**istruzione macchina** interpretabile da un'architettura ARM

(Non è compatibile con architetture X86, MIPS,…)

File eseguibile sum (in esadecimale)

…

3024	e59f	2000	e593
3020	e59f	3000	e593
2003	e082	3018	e59f
3000	e593	0293	e002
3010	e59f	2000	e583

…

Linguaggio assembly

- Il **linguaggio assembly** è un livello di astrazione (leggermente) superiore al linguaggio macchina

- Ogni istruzione macchina viene codificata in una stringa alfanumerica nel linguaggio assembly

Da linguaggio macchina ad assembly

Linguaggio macchina

```
…
3024    e59f
2000    e593
3020    e59f
3000    e593
2003    e082    …
```

Linguaggio assembly

```
…
ldr   r3, [pc, #36]
ldr   r2, [r3]
ldr   r3, [pc, #32]
ldr   r3, [r3]
add   r2, r2, r3
…
```

Da hex a stringa alfanumerica

```
e59f3024  → ldr   r3, [pc, #36]
e5932000  → ldr   r2, [r3]
e59f3020  → ldr   r3, [pc, #32]
e5933000  → ldr   r3, [r3]
e0822003  → add   r2, r2, r3
```

Instruction Set Architecture

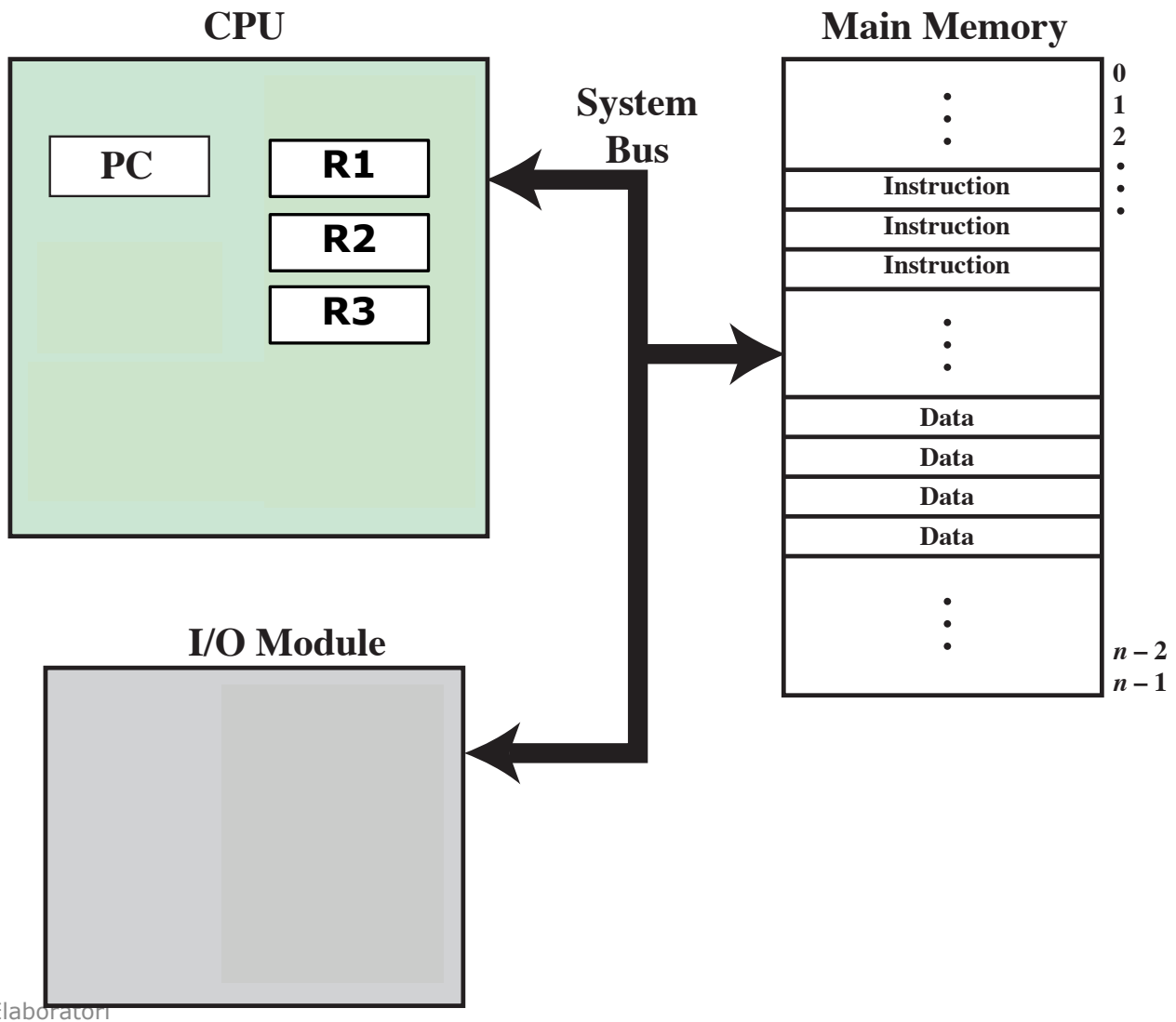
- Ogni istruzione del linguaggio macchina è definita nell'**Instruction Set Architecture (ISA)**
- L'ISA rappresenta il confine tra software e hardware
  - E' il livello più basso di un'architettura visibile ad un programmatore

La macchina semplificata per ISA

- L'ISA ha una visione semplificata della macchina

- Non vede come funzionano i moduli
- Vede solo un sottoinsieme dei registri
- I registri legati a scelte implementative (tipo MAR/MBR) non sono visibili

La macchina semplificata per ISA (2)



Caratteristiche di un'ISA

- **Tipo di dato**
  - Che tipo di dato supportare? Che dimensione?
- **Registri**
  - Quanti registri? Come utilizzarli?
- **Caratteristiche delle istruzioni**
  - Che operazioni supportare? Quanti operandi?
- **Indirizzamento**
  - Quanti e quali modi per specificare gli indirizzi degli operandi?
- **Codifica delle istruzioni**
  - Come codifico un'istruzione?

Dati e registri

Tipi di dati

Numeri

Caratteri

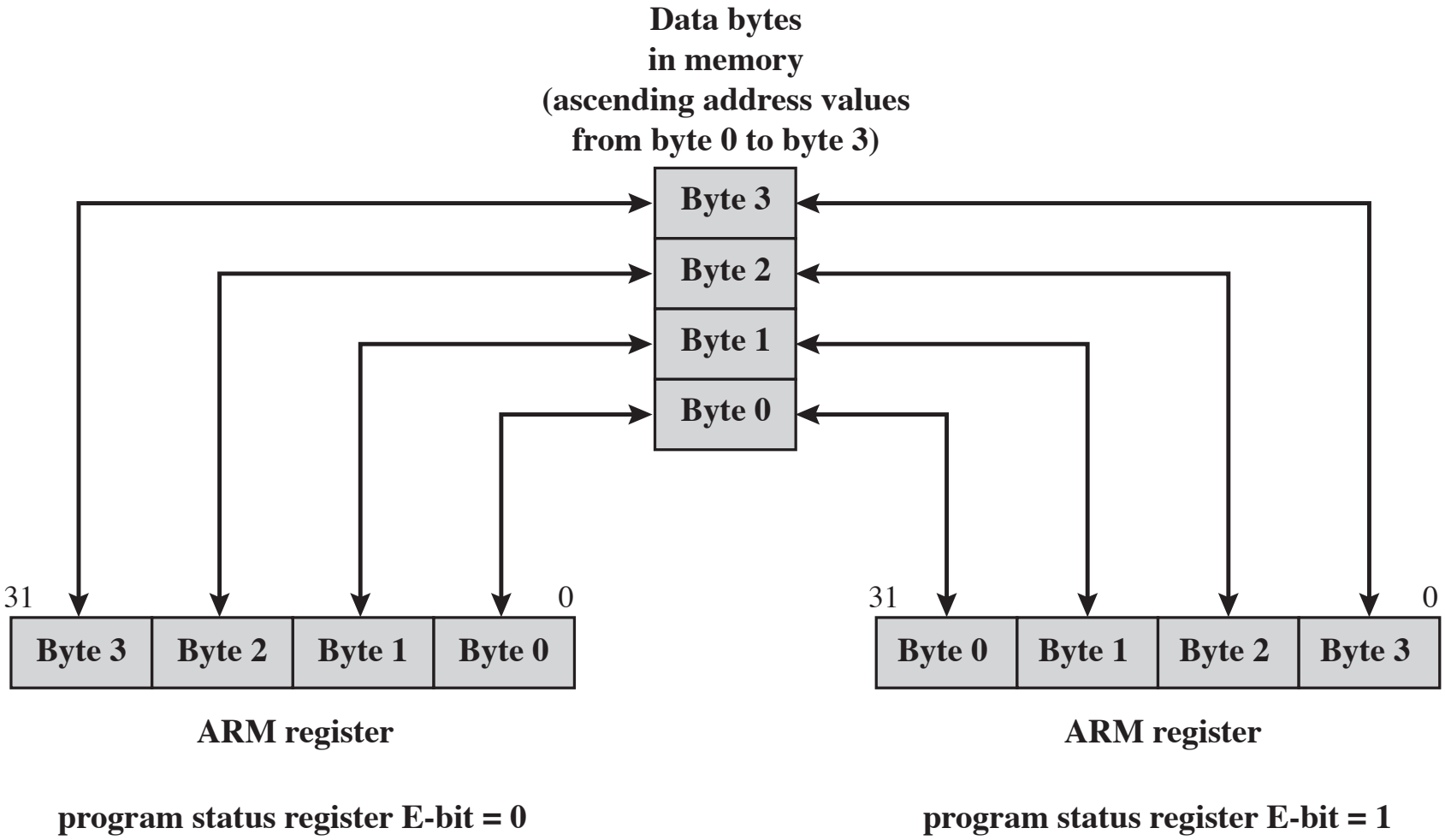
Dati logici

Indirizzi

ARM: tipo di dato

- Il processore ARM supporta dati di lunghezza
  - 8 bit (byte)
  - 16 bit (halfword)
  - 32 bit (word)
- In genere, ARM richiede che dati di k bits siano allocati in indirizzi multiplo di k/8 byte
  - Byte → indirizzi divisibili per 1 (tutti!)
  - Halfword → indirizzi divisibili per 2
  - Word → indirizzi divisibili per 4

ARM: Big/little endian



Registri accessibili

- Le istruzioni macchina hanno accesso in lettura/scrittura a vari tipi di registri
  - **PC (Program Counter):** contiene il puntatore in memoria alla prossima istruzione
  - **SP (Stack Pointer):** contiene un puntatore in memoria alla testa dello stack
  - **FP (Frame Pointer):** contiene un puntatore in memoria alla base del frame pointer corrente
  - **Registri generici (R0, R1,…):** un numero piccolo di registri utilizzati per salvare dati utilizzati frequentemente, per dati temporanei,…
  - **Registro di stato:** contiene informazioni sullo stato del processore

Registro di stato

- Il **registro di stato** (o program status word) è un registro speciale contenente informazioni sullo stato della macchina organizzati in campi
- I campi sono insiemi di bit che rappresentano particolari situazioni (modalità di esecuzione, interrupt…)
- Le flag sono campi di un bip. Le principali flag sono:
  - **N (Negative)**: se N=1 l'ultimo risultato ottenuto era negativo; N=0 altrimenti
  - **Z (Zero)**: se Z=1 l'ultimo risultato ottenuto era uguale a 0; Z=0 altrimenti
  - **C (Carry)**: è il bit di riporto del bit più significativo (il 32-esmo) di un'operazione di riporto
  - **V (oVerflow)**: se V=1, l'ultima operazione aritmetica ha causato un overflow
- **Ogni istruzione può modificare alcuni di questi bit**

Utilizzo dei registri

- La lettura/scrittura dei registri è molto più veloce rispetto alla lettura/scrittura in memoria

- Registri come MAR, MBR, IR, IOAR, IOBR non sono visibili al livello ISA

ARM: Registri

ARM contiene 16 registri

R0-R10: registri generici

R11 (FP): frame pointer

R12 (IP): scratch register

R13 (SP): stack pointer

R14 (LR): link register

R15 (PC): program counter

ARM: Program Counter

- Si consideri l'esecuzione di un'istruzione all'indirizzo di memoria CURRENT_IND
- Per semplicità assumiamo che PC punti all'istruzione successiva in memoria:

PC = CURRENT_IND + 4

- Realtà in ARM: PC = CURRENT_IND + 8
  - Per legacy da quando ARM aveva solo 3 stage
  - Entra in gioco solo con alcune particolari istruzioni

ARM: Registro di stato

- Il **registro CPSR (Current Program Status Register)** è il registro di stato in ARM
- E' possibile scegliere se un istruzione può modificare o meno CPSR
  - Per default, le istruzioni ARM non modificano i bit di stato N,Z,C,V
- Le operazioni di confronto modificano sempre i bit di stato

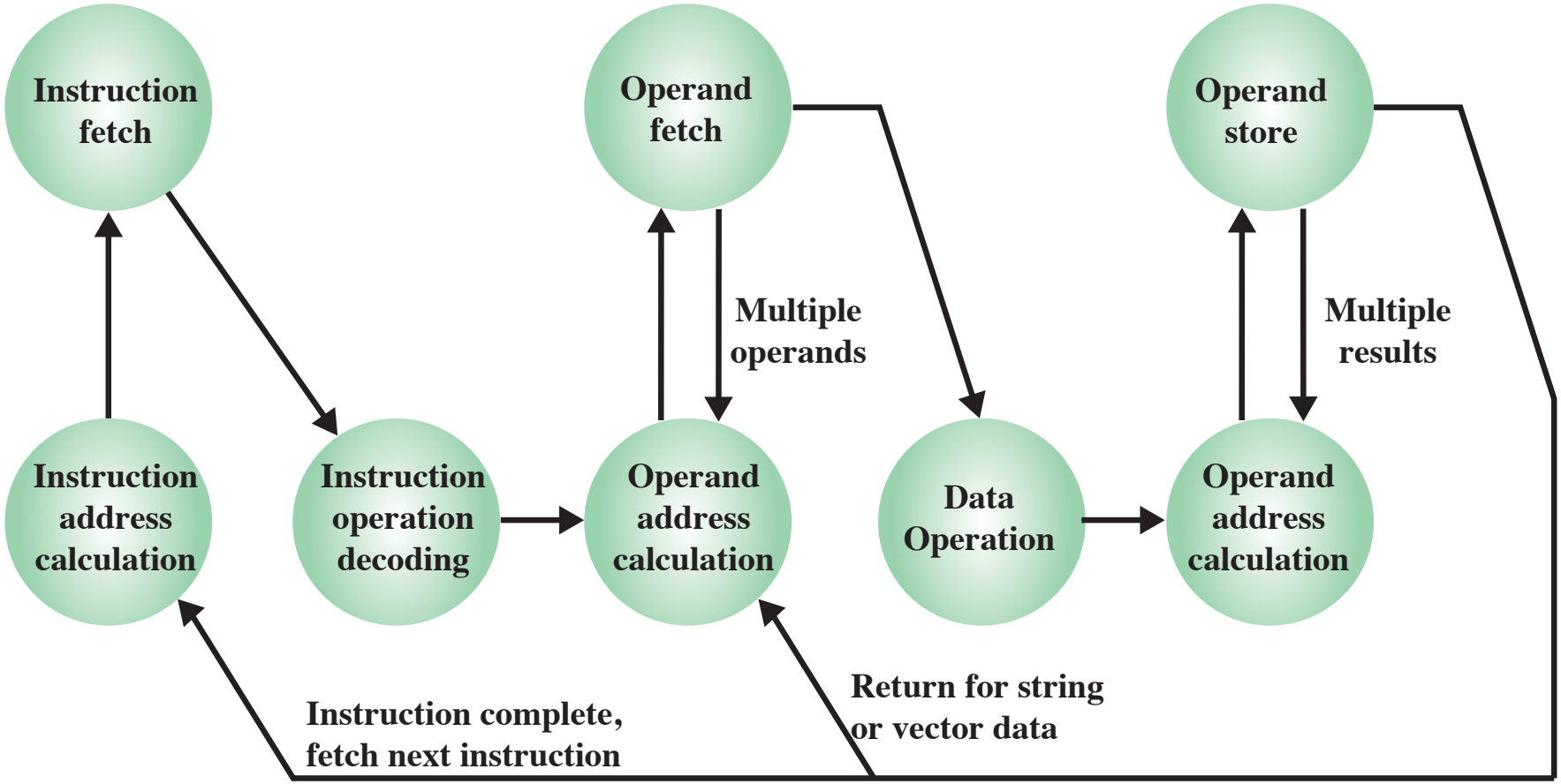
Caratteristiche delle istruzioni

Elementi di un'istruzione macchina

Ogni istruzione deve contenere tutte le informazioni necessarie per l'esecuzione

L'informazione può essere implicita o esplicita

Elementi di un'istruzione macchina (2)



Elementi di un'istruzione macchina (3)

1. Tipo di operazione
2. Riferimento agli operando sorgente e destinazione
3. Indirizzo della prossima istruzione

Tipi di istruzione

- **Aritmetico/Logiche**
  - Operazione aritmetiche e logiche su interi, decimali o sequenze di bit
- **Trasferimento dati**
  - Movimento di dati da e verso registri o locazioni di memoria
- **Input/Output**
  - Accesso in lettura/scrittura a dati e programmi esterni al processore (I/O)
- **Trasferimento di controllo**
  - Determinano le prossime istruzioni da seguire

Type	Operation Name	Description
Data Transfer	Move (transfer)	Transfer word or block from source to destination
	Store	Transfer word from processor to memory
	Load (fetch)	Transfer word from memory to processor
	Exchange	Swap contents of source and destination
	Clear (reset)	Transfer word of 0s to destination
	Set	Transfer word of 1s to destination
	Push	Transfer word from source to top of stack
	Pop	Transfer word from top of stack to destination
Arithmetic	Add	Compute sum of two operands
	Subtract	Compute difference of two operands
	Multiply	Compute product of two operands
	Divide	Compute quotient of two operands
	Absolute	Replace operand by its absolute value
	Negate	Change sign of operand
	Increment	Add 1 to operand
	Decrement	Subtract 1 from operand
Logical	AND	Perform logical AND
	OR	Perform logical OR
	NOT (complement)	Perform logical NOT
	Exclusive-OR	Perform logical XOR
	Test	Test specified condition; set flag(s) based on outcome
	Compare	Make logical or arithmetic comparison of two or more operands; set flag(s) based on outcome
	Set Control Variables	Class of instructions to set controls for protection purposes, interrupt handling, timer control, etc.
	Shift	Left (right) shift operand, introducing constants at end
	Rotate	Left (right) shift operand, with wraparound end

Type	Operation Name	Description
Transfer of Control	Jump (branch)	Unconditional transfer; load PC with specified address
	Jump Conditional	Test specified condition; either load PC with specified address or do nothing, based on condition
	Jump to Subroutine	Place current program control information in known location; jump to specified address
	Return	Replace contents of PC and other register from known location
	Execute	Fetch operand from specified location and execute as instruction; do not modify PC
	Skip	Increment PC to skip next instruction
	Skip Conditional	Test specified condition; either skip or do nothing based on condition
	Halt	Stop program execution
	Wait (hold)	Stop program execution; test specified condition repeatedly; resume execution when condition is satisfied
	No operation	No operation is performed, but program execution is continued
Input/Output	Input (read)	Transfer data from specified I/O port or device to destination (e.g., main memory or processor register)
	Output (write)	Transfer data from specified source to I/O port or device
	Start I/O	Transfer instructions to I/O processor to initiate I/O operation
	Test I/O	Transfer status information from I/O system to specified destination
Conversion	Translate	Translate values in a section of memory based on a table of correspondences
	Convert	Convert the contents of a word from one form to another (e.g., packed decimal to binary)

Operazioni aritmetiche

- Operazioni aritmetiche standard per interi con/senza segno: somma, sottrazione,…
- Spesso vengono fornite operazioni aritmetiche per floating points.
- Altre operazioni includono:
  - valore assoluto, negato, incremento, decremento, operazioni su vettori,…

Operazioni logiche

- Confronti tra valori (<,=,>)
- And, or, xor, not su vettori di bit
- Shift e rotazioni di vettori di bit

AND

0001100
0101011

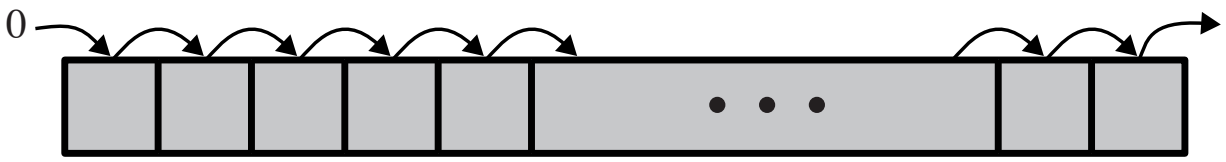
0001000

OR

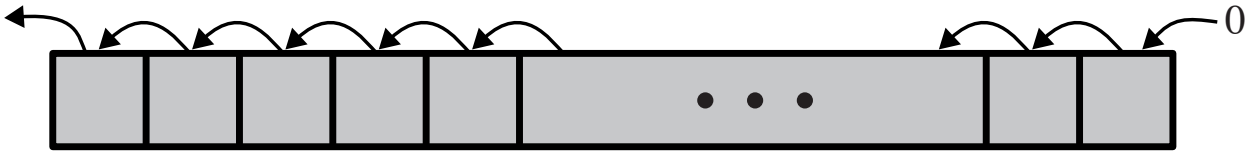
0001100
0101011

0101111

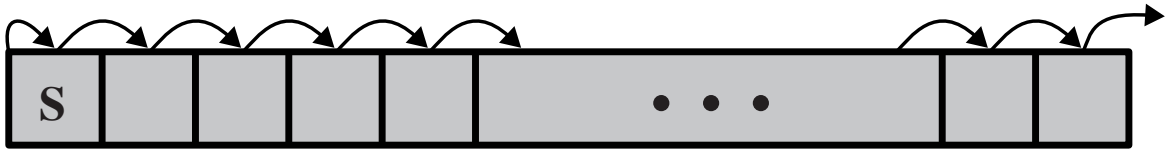
Shift e rotazioni



(a) Logical right shift

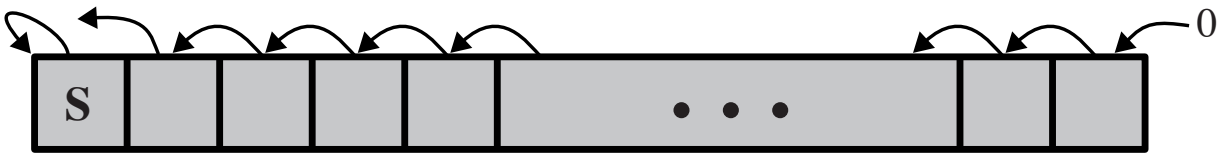


(b) Logical left shift

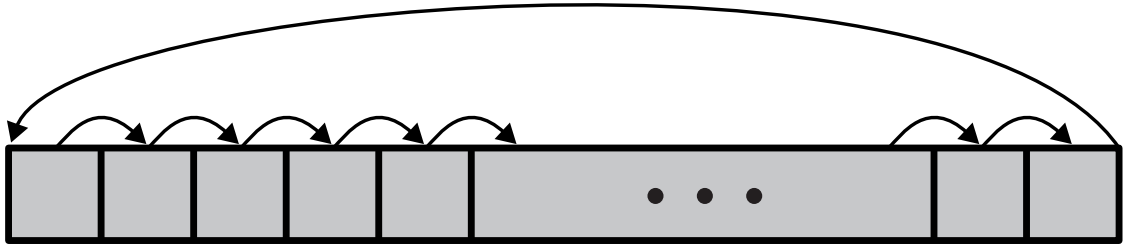


(c) Arithmetic right shift

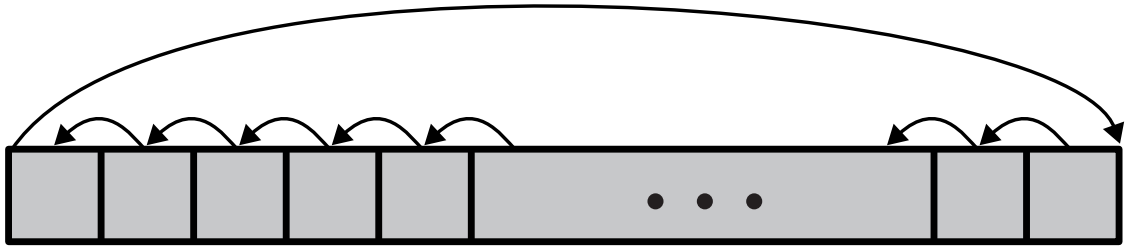
Shift e rotazioni (2)



(d) Arithmetic left shift



(e) Right rotate



(f) Left rotate

Esempi di rotazioni/shift

Input	Operation	Result
10100110	Logical right shift (3 bits)	00010100
10100110	Logical left shift (3 bits)	00110000
10100110	Arithmetic right shift (3 bits)	11110100
10100110	Arithmetic left shift (3 bits)	10110000
10100110	Right rotate (3 bits)	11010100
10100110	Left rotate (3 bits)	00110101

Shift aritmetici e logici

- Gli shift aritmetici sono equivalenti a divisioni/moltiplicazioni per potenze di due per numeri rappresentati a complemento a due

- Lo shift logico a destra non ha interpretazione aritmetica, ma coincide con una divisione nel caso di interi senza segno

- Shift logico a sinistra e shift aritmetico a sinistra coincidono se non c'è overflow.

ARM: istr. aritmetico e logiche

- Istruzioni aritmetiche: ADD, SUB, MUL,…
  - Hanno 3 operandi
  - Esempio: `ADD R1, R2, R3 @ R1 ← R2 + R3`
- Istruzioni logiche: AND, OR, EOR, …
  - Hanno 3 operandi
  - Esempio: `AND R1, R2, R3`
- Istruzioni di rotazione: LSL, LSR, ASR,…
  - Hanno 3 operandi
  - Esempio: LSL R0, R0, #2
- Istruzioni di confronto: CMP, TST, TEQ,..
  - Hanno 2 operandi
  - CMP R1, R2

Trasferimento dati

- Istruzioni per trasferire dati tra memoria e registri
  - Operazioni da registri a registri
  - Operazioni da memoria a registri
  - Operazioni da registri a memoria
  - Operazioni da memoria a memoria
  - Operazioni per pop/push in uno stack

- Informazioni richieste dalle istruzioni:
  - Indirizzo di sorgente/destinazione
  - Modo con cui vengono indicati gli indirizzi
  - Lunghezza del dato da trasferire (byte, halfword, word,…)

ARM: Trasferimento dati

- ARM include le seguenti operazioni
  - da registri a registri (MOV, MVN)
  - da memoria a registri (LDR, LDM, ADR)
  - da registri a memoria (STR, STM)
  - Push, pop (PUSH/POP)
- Si possono trasferire word, halfword, byte
- ARM non ha istruzioni per spostamenti da memoria a memoria

Input/Output

- Istruzioni per gestire I/O e muovere dati da/verso I/O
- Istruzioni dipendono dalla tecnica di I/O utilizzata
  - I/O programmato, memory-mapped I/O, interrupt, DMA
- In ARM: non ci sono istruzioni assembly specifiche, ma si usano quelle per il trasferimento dati da/verso la memoria

Trasferimento di controllo

- Dopo l'esecuzione di un'istruzione viene eseguita l'istruzione immediatamente seguente in memoria

$$PC = PC + 4$$

- In molti casi, è necessario cambiare l'indirizzo della prossima istruzione

$$PC = \text{nuovo valore}$$

Quando modificare il PC?

- Ripetere dei segmenti di codice
  - Costrutti `for, while`,…
- Eseguire delle istruzioni solo quando si verifica una condizione
  - Costrutti `if, switch`,…
- Per riutilizzare del codice già scritto
  - Chiamate a subroutine

Tipi di trasferimento di controllo

- Salto condizionato o incondizionato (branch)
- Salto di istruzione (skip)
- Chiamata a procedura (procedure call)

Istruzione di branch

- Il branch (o istruzione di salto) indica l'indirizzo della prossima istruzione da eseguire

- Il branch può essere:
  - **Incondizionato**: se viene sempre effettuato
  - **Condizionato**: se effettuato quando una certa condizione si realizza

Istruzione di skip

- Lo skip permette di non eseguire un'istruzione se una condizione non viene soddisfatta

- Esempio: esegui somma se il risultato precedente è zero
  - Se si: effettua somma e passa alla prossima istruzione
  - Se no: passa alla prossima istruzione

Chiamate a procedura

- **Procedura o subroutine**: è un programma autonomo all'interno di un programma più grande

- La chiamata a procedura indica l'indirizzo della prima istruzione della procedura

ARM: trasferimento di controllo

- B (branch incondizionato)

```
B R1          @ PC ← R1
```

- Tutte le istruzioni possono essere rese condizionate: la loro effettiva esecuzione può essere fatta dipendere dallo stato dei bit ZNCV nel CPSR

Operandi

- **Memoria**
  - L'istruzione fornisce l'indirizzo di memoria
- **Dispositivo di I/O**
  - L'istruzione fornisce l'indirizzo del dispositivo di I/O
- **Registro**
  - L'istruzione fornisce l'indirizzo del registro
- **Immediato**
  - Il valore da usare è contenuto nell'istruzione

Numero di operandi

- Gli operandi specificano dove trovare i valori di input e dove salvare il risultato
- Gli operandi possono essere espliciti o impliciti
  - **Esplicito**: l'istruzione definisce l'operando
  - **Implicito**: l'operando assume un valore predefinito e non viene rappresentato nell'istruzione

Numero di operandi (2)

- Il numero di operandi dipende dal tipo di operazioni e da scelte architetturali
- Operazioni standard possono avere: 3, 2, 1, 0 operandi
- Operazioni speciali possono avere più di 3 operandi

Esempio a 3 operandi

Programma che calcola

$$\mathbf{Y} = \frac{\mathbf{A} - \mathbf{B}}{\mathbf{C} + (\mathbf{D} \times \mathbf{E})}$$

Instruction	**Comment**
`SUB Y, A, B`	`Y ← A - B`
`MUL T, D, E`	`T ← D * E`
`ADD T, T, C`	`T ← T + C`
`DIV Y, Y, T`	`Y ← Y / T`

Ogni istruzione specifica:

- 2 operandi sorgente
- 1 operando destinazione

Esempio a 2 operandi

Programma che calcola

$$Y = \frac{A - B}{C + (D \times E)}$$

Instruction	Comment
MOV Y, A	Y ← A
SUB Y, B	Y ← Y - B
MOV T, D	T ← D
MUL T, E	T ← T * E
ADD T, C	T ← T + C
DIV Y, T	Y ← Y / T

Ogni istruzione specifica:
- 2 operandi sorgente

L'operando destinazione è implicito:
- Coincide con il primo operando sorgente

Esempio a 1 operandi

Programma che calcola

$$\mathbf{Y} = \frac{\mathbf{A} - \mathbf{B}}{\mathbf{C} + (\mathbf{D} \times \mathbf{E})}$$

Instruction		Comment
LOAD	D	AC ← D
MUL	E	AC ← AC * E
ADD	C	AC ← AC + C
STORE	Y	Y ← AC
LOAD	A	AC ← A
SUB	B	AC ← AC - B
DIV	Y	AC ← AC / Y
STORE	Y	Y ← AC

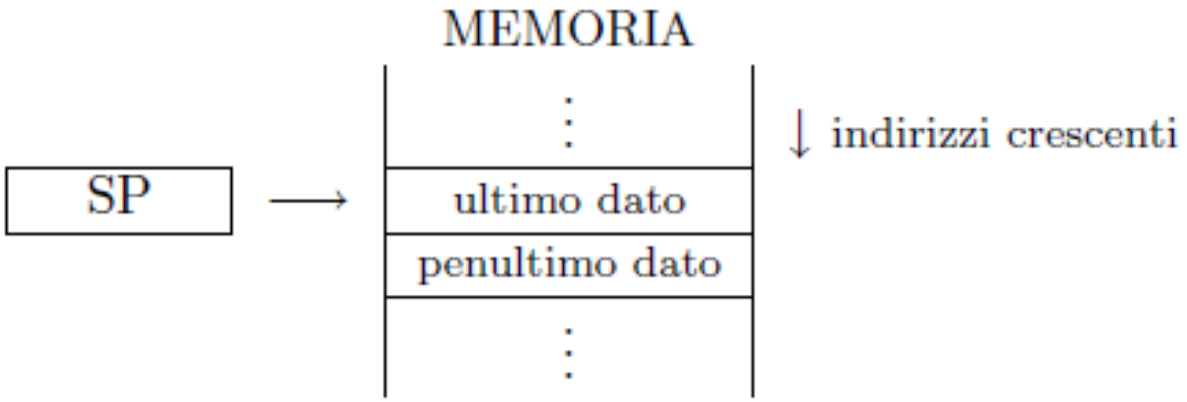
Ogni istruzione specifica:
- 1 operando sorgente

I restanti due operandi sono impliciti
- Il secondo operando si trova nel registro accumulatore (AC)
- Il risultato viene scritto nel registro AC

Istruzioni a 0 indirizzi: stack

- Lo **stack** (o pila) è una struttura cosiddetta "LIFO" (Last In First Out), ovvero che mantiene una serie di impilati uno sull' altro
- Le operazioni sono
  - **PUSH**: aggiungere un nuovo dato sopra agli altri già presenti
  - **POP**: togliere dalla pila il dato che sta in cima
- Ad uno stack è associato un puntatore alla testa dello stack (**stack pointer**)
- In genere, ogni programma ha accesso ad uno stack
  - Lo stack pointer è mantenuto nel **registro SP**

Istruzioni a 0 indirizzi: uso stack



Sia D un word (4 byte)

push D: SP - 4 → SP
D → M[SP]

pop D: M[SP] → D
SP + 4 → SP

Nelle istruzioni a 0 indirizzi, tutti gli operandi si trovano in uno stack.

- Gli operandi sorgente si estraggono con due POP.
- L'operando destinatario si inserisce con un PUSH.

Gli operandi sono impliciti.

Istruzioni a 0 indirizzi

```
ADD     ; pop + pop → push
SUB     ; pop → T, pop - T → push
MUL     ; pop * pop → push
DIV     ; pop → T, pop / T → push
```

Servono due istruzioni per estrarre o inserire un operando in cima allo stack:

```
POP X   ; pop → M[X]
PUSH X  ; M[X] → push
```

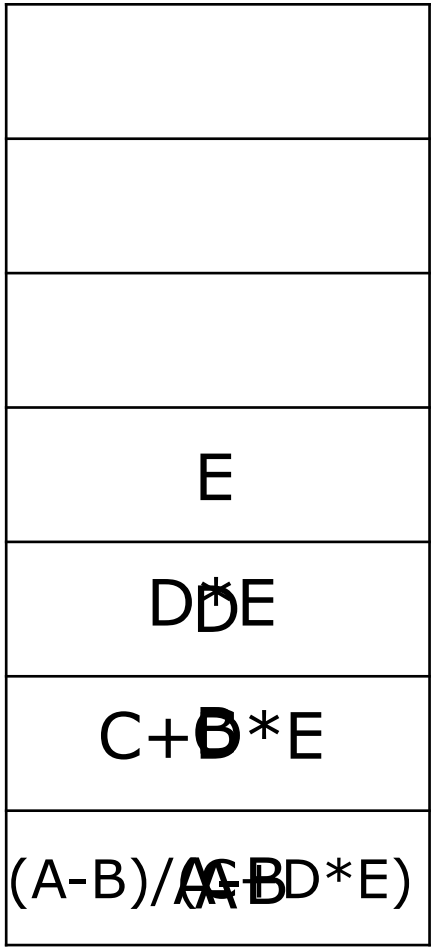
Istruzioni a 0 indirizzi: esempio

```
PUSH A
PUSH B
SUB
PUSH C
PUSH D
PUSH E
MUL
ADD
DIV
```

$$\mathbf{Y} = \frac{\mathbf{A} - \mathbf{B}}{\mathbf{C} + (\mathbf{D} \times \mathbf{E})}$$

Istruzioni a 0 indirizzi: esempio (2)

```
PUSH A
PUSH B
SUB
PUSH C
PUSH D
PUSH E
MUL
ADD
DIV
```



Numero operandi

Number of Addresses	Symbolic Representation	Interpretation
3	OP A, B, C	A ← B OP C
2	OP A, B	A ← A OP B
1	OP A	AC ← AC OP A
0	OP	T ← (T – 1) OP T

AC = accumulator
T = top of stack
(T – 1) = second element of stack
A, B, C = memory or register locations

- Numero di operandi alto → programma corto; istruzioni complicate.
- Numero di operandi basso → programma lungo; istruzioni semplici.

ARM: Numero operandi

- Le istruzioni ARM hanno un numero di operando coerente con la definizione dell'istruzione
  - Somma ha 3 operandi (2 input e 1 output)
  - Confronto ha 2 operandi (2 input)
- ARM ha istruzioni a
  - 4 operandi (prodotto con somma)
  - 3 operandi (somme, sottrazioni,…)
  - 2 operandi (confronti, spostamento dati)
  - 1 operando (salti)
  - 0 operandi (nop)