



NoiStudiamo

DISPENSA DI LABORATORIO

Assembly RISC e ARM VisUAL

Istruzioni macchina, modi di indirizzamento, cicli, pila e sottoprogrammi,
con gli esercizi del laboratorio. Fatto da studenti, per studenti.

A CURA DI

Angelo Faro

Ottobre 2026

Indice dei capitoli

Dispensa sull'assembly del modulo di laboratorio: assembly generico e ARM in VisUAL, con esempi svolti ed esercizi.

1. Il laboratorio di assembly	3
2. Istruzioni di base e modi di indirizzamento	5
3. Direttive e struttura del codice	7
4. Bit di esito, salti e cicli	8
5. La pila	10
6. Sottoprogrammi	12
7. Istruzioni logiche, scorrimenti e aritmetiche	17
8. Esercizi del laboratorio	18

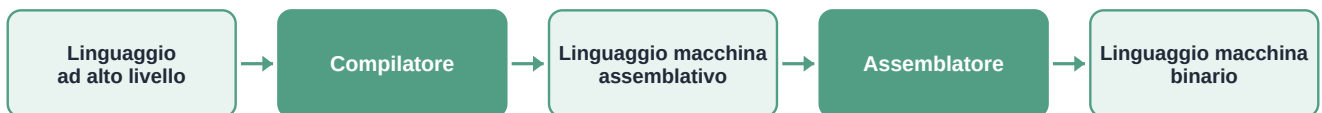
Il laboratorio di assembly

Come si programma un calcolatore a livello di istruzioni macchina: l'assembly generico di riferimento e l'emulatore ARM VisUAL usato in laboratorio.

Come si programma

Il calcolatore esegue **sequenzialmente** una serie di istruzioni. Le istruzioni definiscono le operazioni da eseguire e sono raggruppate in **programmi**; spesso le operazioni devono essere eseguite su dei **dati**. L'utente interagisce con il calcolatore tramite le interfacce di I/O, cioè le periferiche.

Il programmatore scrive i programmi in **linguaggio assemblativo** (assembly), che l'**assemblatore** traduce in sequenze binarie. I linguaggi ad alto livello (C, C++ e simili) sono ancora più espressivi: il **compilatore** traduce il codice ad alto livello in codice assemblativo.



Dal linguaggio ad alto livello al linguaggio macchina binario.

Memoria e dati

L'informazione è immagazzinata in memoria sotto forma di **vettore di parole** in successione, ciascuna di lunghezza n che varia da 16 a 64 bit. A ogni parola è associato un **indirizzo** binario univoco. Per convenzione l'indirizzo è un numero naturale (sempre positivo o al più nullo) e si usano gli indirizzi dell'intervallo $[0, 2^m)$, con m pari al numero di bit dell'indirizzo.

DEFINIZIONE

La quantità totale di parole che la memoria contiene si chiama **spazio di indirizzamento**. Ad esempio con $m = 24$ bit di indirizzo lo spazio di indirizzamento è di 2^{24} parole di memoria.

Istruzioni e dati sono rappresentati da sequenze di cifre binarie (bit). Per convenzione una sequenza di 8 bit è detta **byte**, e i byte si raggruppano in blocchi con un numero di elementi espresso con potenze di 2:

UNITÀ	SIMBOLO	VALORE	ORDINE DI GRANDEZZA
Kilobyte	KB	$2^{10} = 1024$	circa 10^3
Megabyte	MB	$2^{20} = 1024 \cdot 1024$	circa 10^6
Gigabyte	GB	2^{30}	circa 10^9
Terabyte	TB	2^{40}	circa 10^{12}

Di norma l'unità minima di informazione indirizzabile in memoria è il byte. Si assegnano indirizzi consecutivi ai byte contenuti in ciascuna parola e gli indirizzi delle parole sono multipli della loro lunghezza in byte. Esistono due schemi di indirizzamento dei byte:

- **crescente (big endian)**: l'indirizzo aumenta al diminuire del peso aritmetico del byte;
- **decrescente (little endian)**: l'indirizzo aumenta all'aumentare del peso aritmetico del byte.

Insiemi di istruzioni RISC e CISC

- **RISC** (*reduced instruction set computer*): insieme di istruzioni base ridotto; ogni istruzione occupa una sola parola di memoria.
- **CISC** (*complex instruction set computer*): insieme di istruzioni base complesse; ogni istruzione può occupare più di una parola di memoria.

Assembly generico e ARM in VisUAL

Le slide presentano prima un assembly RISC generico (Load, Store, Add, Subtract, Branch_if, Call, Return), utile per capire concetti validi su qualsiasi architettura, e poi lo stesso insieme di istruzioni in **VisUAL**, un emulatore ARM multiplatforma sviluppato per facilitare l'apprendimento dell'ARM assembly.

REGISTRI IN VISUAL

REGISTRO	RUOLO
R0 ... R12	Registri di uso generale: deposito di dati di uso immediato e puntatori a parole di memoria contenenti dati (o altri puntatori) su cui lavorare.
R13, o SP	Stack pointer: indirizzo di memoria dello stack. Il suo uso come stack pointer non è obbligatorio, è solo una convenzione.
R14, o LR	Link register: vi è salvato l'indirizzo di ritorno nel momento in cui viene chiamata una procedura.
R15, o PC	Program counter: contiene l'indirizzo di memoria della prossima istruzione da eseguire.

Nella finestra di VisUAL si vedono l'editor, il pannello dei registri (con le schede per memoria e simboli) e in basso a destra i bit di stato N, Z, C e V.

CORRISPONDENZA TRA ASSEMBLY GENERICO E VISUAL

ASSEMBLY GENERICO	VISUAL	SIGNIFICATO
Load, Store	LDR, STR	Carica una parola dalla memoria in un registro e la salva in memoria
Add, Subtract	ADD, SUB	Somma e sottrazione
Move	MOV	Copia un registro o un valore immediato in un registro
And, Or, XOr	AND, ORR, EOR	Operazioni logiche bit a bit
LShiftL, LShiftR	LSL, LSR	Scorrimento logico a sinistra e a destra
AShiftR, AShiftL	ASR, LSL	Scorrimento aritmetico a destra e a sinistra
RotateL, RotateR	nessuna, ROR	Rotazione a sinistra e a destra
LoadByte, StoreByte	LDRB, STRB	Lettura e scrittura di un singolo byte
Multiply, Divide, ORIGIN	nessuna	Le slide non indicano un equivalente
Call, Return	BL, MOV PC, LR	Chiamata a sottoprogramma e rientro
push, pop	STM, LDM	Salvataggio e ripristino di più registri sulla pila
DATAWORD, RESERVE	DCD, FILL	Riserva una parola o uno spazio di memoria
EQU, END	EQU, END	Eguaglianza simbolica e fine del sorgente

Istruzioni di base e modi di indirizzamento

Le istruzioni per spostare dati e fare somme e sottrazioni, e i modi con cui un'istruzione indica dove si trovano operandi e risultato.

Lista delle istruzioni di base

ISTRUZIONE	EFFETTO
Load R2, LOC	Legge i contenuti della locazione di memoria il cui indirizzo è rappresentato dall'etichetta LOC e li carica nel registro R2. I contenuti originali di LOC sono preservati, quelli di R2 sono sovrascritti.
Add R4, R2, R3	Somma i contenuti dei registri R2 e R3 e inserisce la somma in R4. Gli operandi in R2 e R3 non sono alterati, il precedente valore di R4 è sovrascritto.
Store R4, LOC	Copia l'operando del registro R4 nella locazione di memoria LOC. I contenuti originali di LOC sono sovrascritti, quelli di R4 sono preservati.
Move R4, R3	Carica il contenuto del registro R3 in R4.
Subtract R4, R2, R3	Sottrae i contenuti dei registri R2 e R3 e inserisce il risultato in R4.

Un primo programma: la somma di due valori

Semplice programma che somma due valori presenti in memoria e ne salva il risultato: $C \leftarrow [A] + [B]$. È composto da 4 istruzioni (2 Load, 1 Add e 1 Store), memorizzate in parole di memoria consecutive e lette sequenzialmente. I dati A, B e C si trovano in locazioni successive al segmento di programma.

```
i      Load  R2, A
i + 4   Load  R3, B
i + 8   Add    R4, R2, R3
i + 12  Store  R4, C
```

Modi di indirizzamento

I metodi con cui, nell'istruzione macchina, si specifica la posizione di operandi e risultato si chiamano **modi di indirizzamento**. Ogni istruzione contiene un **codice operativo** (specifica l'operazione da eseguire) e gli **operandi** (i dati su cui eseguirla).

MODO	DESCRIZIONE ED ESEMPIO
Di registro	L'operando o il risultato è in un registro, il cui nome (che è indirizzo) è dato nell'istruzione. In Add R4, R2, R3 tutti e tre gli operandi sono in modo di registro: R2 e R3 contengono i sorgenti, R4 è la destinazione.
Assoluto (diretto)	L'operando o il risultato è in una parola di memoria il cui indirizzo è dato nell'istruzione. Rappresenta le variabili globali: una dichiarazione come Integer NUM1, NUM2, SOMMA fa allocare al compilatore una locazione per ciascuna variabile e i riferimenti successivi usano il modo assoluto. Esempio: Load R2, NUM1.
Immediato	L'operando è dato esplicitamente nell'istruzione, con la costante preceduta dal simbolo #. Esempio: Add R4, R6, #200 somma 200 al contenuto di R6 e pone il risultato in R4.
Indiretto	Il nome di un registro contiene l'indirizzo di memoria dell'operando o del risultato. Si scrive con il nome del registro tra parentesi tonde. Esempio: Load R2, (R5): R5 agisce da puntatore alla lista e si accede a un elemento usando l'indirizzo in R5.
Con indice e spiazzamento	L'indirizzo effettivo si ottiene sommando un valore costante (lo spiazzamento) al contenuto di un registro. Si scrive X(Ri), con X spiazzamento e Ri registro con l'indirizzo. È utile per vettori e liste. Esempio: Load R2, 20(R5); se R5 vale 1000, si preleva il contenuto della locazione 1020.

Le istruzioni in VisUAL

In VisUAL i modi di indirizzamento si scrivono così:

MODO	SINTASSI VISUAL
Registro	Ri
Diretto (o assoluto)	Locazione
Indiretto	[Ri]
Immediato	#Valore
Base e spiazzamento	[Ri, #Valore]

ISTRUZIONE	EFFETTO
LDR Ri, [Rj]	Carica nel registro Ri il contenuto della parola di memoria puntata da Rj.
STR Ri, [Rj]	Salva nella parola di memoria puntata da Rj il contenuto di Ri.
ADD Rd, Ri, Rj	Somma il contenuto di Ri e Rj e lo carica nel registro Rd.
SUB Rd, Ri, Rj	Sottrae il contenuto di Rj da quello di Ri e lo carica in Rd.
MOV Ri, Rj oppure MOV Ri, #Valore	Carica nel registro Ri il contenuto di Rj oppure il valore immediato.
ADDS Rd, Ri, Rj	Come ADD, e in più aggiorna i bit di stato NZCV.
SUBS Rd, Ri, Rj	Come SUB, e in più aggiorna i bit di stato NZCV.

Direttive e struttura del codice

Come si organizza un programma sorgente: le direttive che parlano all'assemblatore e i campi di una linea di codice.

Direttive di assemblatore

L'assemblatore traduce un programma scritto in codice assembly (programma sorgente) in codice macchina binario (programma oggetto). Il linguaggio assembly non contiene solo le istruzioni del programma, ma anche comandi specifici per l'assemblatore, le **direttive di assemblatore**, che non sono tradotte nel programma oggetto e servono a dargli informazioni utili.

DIRETTIVA	SINTASSI	FUNZIONE	VISUAL
DATAWORD	DATAWORD contenuto	Riserva una parola di memoria e la inizializza con il contenuto indicato.	DCD
EQU	NOME EQU valore	Associa un valore numerico a un nome usato nel sorgente: ogni occorrenza di NOME è sostituita dal valore.	EQU
ORIGIN	ORIGIN indirizzo	Indica l'indirizzo di partenza da cui inserire istruzioni e dati definiti nelle righe seguenti.	non indicata
RESERVE	RESERVE byte	Riserva uno spazio di memoria espresso in byte, che non viene inizializzato.	FILL
END	END	Indica la fine del testo del programma sorgente.	END

Negli esercizi sulle stringhe i caratteri ASCII sono dichiarati con la direttiva **DCB**, che definisce una sequenza di byte scritti in esadecimale.

Linea di codice assembly

In genere una linea di codice assembly presenta quattro campi:

Etichetta Operazione Operandi Commento

- **Etichetta**: nome associato all'indirizzo della parola di memoria assegnata all'istruzione o all'indirizzo del blocco di memoria riservato. È facoltativa.
- **Operazione**: il nome (codice operativo) dell'istruzione oppure una direttiva di assemblatore.
- **Operandi**: informazioni di indirizzamento per accedere agli operandi.
- **Commento**: testo ignorato dall'assemblatore, marcato dai caratteri ; oppure % oppure ! o simili.

Bit di esito, salti e cicli

Come il processore tiene traccia dell'esito delle operazioni e come questo permette di ripetere istruzioni con i cicli.

I bit di esito

Il processore tiene traccia di alcune informazioni ausiliarie relative all'esito di svariate operazioni, che servono come condizioni per l'istruzione di salto. Sono espresse come insiemi di bit, detti **codici o bit di condizione** o, in breve, **bit di esito**. Sono raggruppati in un registro apposito interno al processore, il **registro di stato**. Ogni esito è binario e vale 1 o 0 secondo l'andamento dell'operazione cui si riferisce.

BIT DI ESITO	SIGNIFICATO
N (negativo)	1 se il risultato è negativo, 0 se positivo o nullo
Z (zero)	1 se il risultato è nullo, 0 altrimenti
C (riporto)	1 se si ha trabocco in binario naturale, 0 altrimenti (Carry)
V (trabocco)	1 se si ha trabocco in complemento a due, 0 altrimenti (oVerflow)

COME SI AGGIORNANO

I bit di esito si aggiornano quando si esegue un'istruzione logica o aritmetica nella versione che li modifica, o con istruzioni di confronto:

ISTRUZIONE	EFFETTO
ADDS Rd, Ri, Rj	Somma Ri e Rj, carica il risultato in Rd e aggiorna i bit NZCV.
SUBS Rd, Ri, Rj	Sottrae Rj da Ri, carica il risultato in Rd e aggiorna i bit NZCV.
CMP Ri, Rj	Sottrae Rj da Ri e aggiorna i bit di stato. Il risultato viene scartato.
CMN Ri, Rj	Somma Ri e Rj e aggiorna i bit di stato. Il risultato viene scartato.

CODICI DI CONDIZIONE

I codici di condizione sono espressioni sui bit di stato che stabiliscono se un salto condizionato va eseguito. Con $\oplus$ si indica l'XOR:

CODICE	SIGNIFICATO	CONDIZIONE SUI BIT
EQ	equal, uguale	Z
NE	not equal, diverso	$\neg Z$
GE	greater equal, maggiore o uguale	$\neg(N \oplus V)$
LT	lower than, minore	$N \oplus V$
GT	greater than, maggiore	$\neg(Z \text{ OR } (N \oplus V))$
LE	lower equal, minore o uguale	$Z \text{ OR } (N \oplus V)$

Esecuzione sequenziale e con salto

Per sommare N numeri NUM1, NUM2, ..., NUMn si può procedere in due modi. Nell'**esecuzione sequenziale** il programma contiene un Load e un Add per ogni numero, quindi la sua lunghezza cresce con N:

```
Load  R2, NUM1
Load  R3, NUM2
Add   R2, R2, R3
Load  R3, NUM3
Add   R2, R2, R3
...
Load  R3, NUMn
Add   R2, R2, R3
Store R2, SOMMA
```

Nell'**esecuzione con salto** si scrive una sola volta il corpo del ciclo e lo si ripete. Il ciclo inizia all'etichetta CICLO e termina con l'istruzione Branch_if_[R2]>0; N è il numero di numeri e R2 è il registro usato come contatore.

Istruzione di salto condizionato

DEFINIZIONE

Branch_if_condizione_destinazione_salto: salta all'esecuzione di un'istruzione specifica se la condizione di salto è vera. La destinazione del salto è una locazione di memoria contenente l'istruzione da eseguire quando la condizione è vera. La condizione può confrontare valori contenuti nei registri (scritti tra parentesi quadre, [Ri]) oppure valori espressi esplicitamente.

```
Branch_if_[R2]>0 CICLO      ; salta a CICLO se il contenuto di R2 è maggiore di 0
Branch_if_[R2]>[R3] CICLO   ; salta a CICLO se [R2] è maggiore di [R3]
```

Esempio: somma di n numeri con salto

```
Load  R2, N                ; carica la dimensione della lista
Clear R3                   ; inizializza la somma a 0
Move  R4, #NUM1            ; carica l'indirizzo del primo numero
CICLO: Load R5, (R4)       ; preleva il prossimo numero
Add   R3, R3, R5           ; aggiunge questo numero alla somma
Add   R4, R4, #4           ; incrementa il puntatore alla lista
Subtract R2, R2, #1        ; decrementa il contatore
Branch_if_[R2]>0 CICLO     ; salta indietro se non ancora finito
Store R3, SOMMA            ; immagazzina la somma finale
```

Il puntatore R4 avanza di 4 a ogni giro perché ogni numero occupa una parola di 4 byte; il contatore R2 parte da N e il ciclo termina quando arriva a 0.

La pila

La struttura dati su cui poggiano i sottoprogrammi: come si impila e si spila, con le istruzioni generiche e con STM e LDM in VisUAL.

Che cos'è la pila

La pila (*stack*) è una lista di elementi alla quale si applicano restrizioni sul modo di inserirli ed estrarli. L'estremità in alto è detta **cima** (*top*), quella in basso **base** o **fondo** (*bottom*). Si può lavorare solo agendo sulla cima della pila. Lo **stack pointer** (SP, R13) è il registro che punta alla cima.

Push e pop con le istruzioni generiche

Si possono eseguire solo due operazioni. Il **push** (impila) pone un elemento in cima, il **pop** (spila) lo toglie:

```
; PUSH
Subtract SP, SP, #4      ; diminuisce SP di una parola, per puntare alla nuova cima
Store      Rj, (SP)      ; scrive il valore richiesto nella parola puntata da SP

; POP
Load       Rj, (SP)      ; copia il valore della locazione puntata da SP in un registro
Add        SP, SP, #4     ; aumenta SP di una parola, per puntare alla nuova cima
```

Push e pop in VisUAL: STM e LDM

ISTRUZIONE	OPERAZIONE	NOTE
STM[codice] SP[!] {lista di registri}	PUSH	Store Multiple. Il registro di ordine più basso va in cima.
LDM[codice] SP[!] {lista di registri}	POP	Load Multiple. Il registro di ordine più basso è preso dalla cima.

CODICE	DESCRIZIONE
FD	Full descending
FA	Full ascending
ED	Empty descending
EA	Empty ascending
!	Aggiorna il valore del puntatore alla cima a fine operazione

Le quattro varianti descrivono come è organizzata la pila: *descending* o *ascending* indicano se cresce verso indirizzi decrescenti o crescenti, *full* o *empty* se SP punta all'ultimo elemento inserito oppure alla prima locazione libera.

ESEMPIO DI STM

Si supponga che R0 contenga 18, R1 contenga 20, R2 contenga 22, R3 contenga 24, R4 contenga 26 e R5 contenga 28. Si eseguono tre Store Multiple, STM {R0,R1}, STM {R2,R3} e STM {R4,R5}, con lo stesso codice. Alla fine, scrivendo la memoria a indirizzi crescenti, si ha:

VARIANTE	CONTENUTO DELLA MEMORIA	SP PUNTA A
FD	26, 28, 22, 24, 18, 20	26, l'ultimo elemento inserito
FA	18, 20, 22, 24, 26, 28	28, l'ultimo elemento inserito
ED	locazione libera, 26, 28, 22, 24, 18, 20	la locazione libera prima di 26
EA	18, 20, 22, 24, 26, 28, locazione libera	la locazione libera dopo 28

L'ordine in cui si elencano i registri non conta: con FD, STM {R3,R2} produce lo stesso risultato di STM {R2,R3}, perché il registro di ordine più basso va comunque in cima.

ESEMPIO DI LDM

Con la pila ottenuta con FD (dalla cima: 26, 28, 22, 24, 18, 20) e tutti i registri a 0, si eseguono due Load Multiple di seguito:

PASSO	REGISTRI DOPO IL PASSO	SP PUNTA A
Situazione iniziale	R0 = R1 = R2 = R3 = R4 = R5 = 0	26
LDM {R0,R1}	R0 = 26, R1 = 28, gli altri a 0	22
LDM {R3,R4}	R0 = 26, R1 = 28, R3 = 22, R4 = 24, R2 e R5 a 0	18

Anche per LDM l'ordine di elencazione non conta: LDM {R0,R1} è equivalente a LDM {R1,R0}.

Sottoprogrammi

Come si richiama un compito specifico da più punti del programma: chiamata, rientro, area di attivazione, annidamento e passaggio dei parametri.

Chiamata e rientro

Un **sottoprogramma** (o *routine*) è una lista di istruzioni che eseguono un compito specifico e che possono essere richiamate in un qualsiasi momento durante l'esecuzione di un programma. Il programma che effettua il salto è detto **chiamante** (*caller*); l'istruzione che effettua il salto al sottoprogramma è l'**istruzione di chiamata** (*call*). Terminata la routine, l'esecuzione riprende dall'istruzione successiva alla chiamata: la routine **rientra** al programma chiamante, con l'**istruzione di rientro**.

DEFINIZIONE

Il **link register** (LR) è il registro speciale in cui si memorizza l'indirizzo dell'istruzione di rientro durante l'esecuzione di un sottoprogramma.

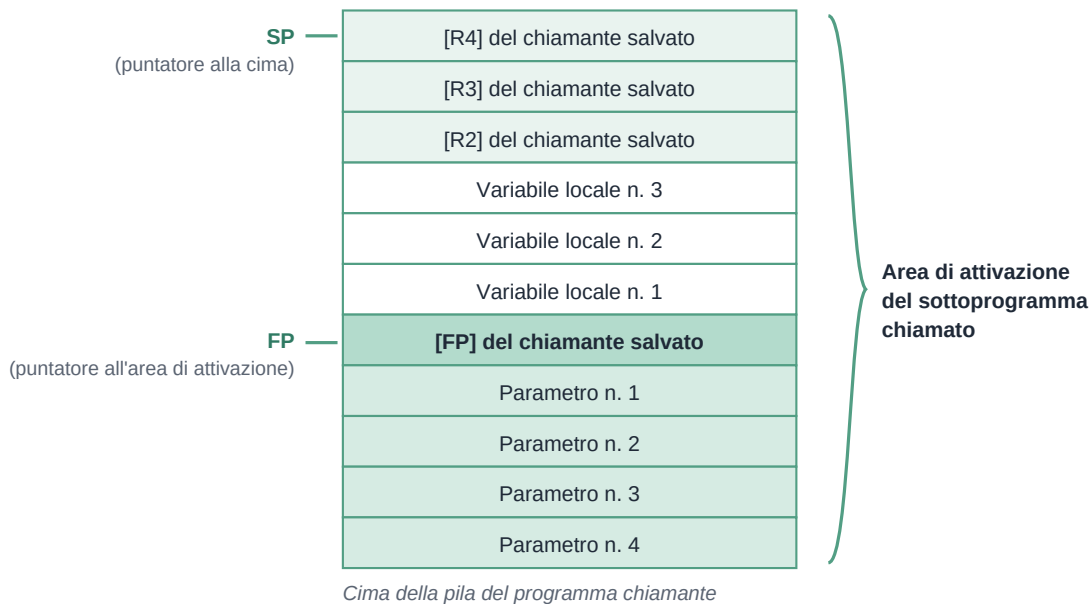
	ASSEMBLY GENERICO	VISUAL
Chiamata	Call INDIRIZZO: salva il PC nel link register e salta all'indirizzo di destinazione.	BL Indirizzo: salva il PC nel LR prima di effettuare il salto.
Rientro	Return: salva il contenuto del link register nel PC.	MOV PC, LR: rientra dal sottoprogramma.

Esempio: il programma chiamante contiene all'indirizzo 200 la chiamata a SUB e all'indirizzo 204 l'istruzione successiva; il sottoprogramma parte dall'indirizzo 1000.

FASE	PC	REGISTRO DI COLLEGAMENTO
Chiamata	1000 (prima istruzione di SUB; prima del salto valeva 204)	204
Rientro	204 (copiato dal registro di collegamento)	204

Area di attivazione in pila

Il blocco di memoria nella pila riservato al sottoprogramma è detto **area di attivazione** (*stack frame*). Il **frame pointer** (FP) è un registro che punta allo stack frame del sottoprogramma in esecuzione. Lo stack frame contiene i parametri, il FP del programma chiamante, le variabili locali e i valori dei registri salvati.



Area di attivazione di un sottoprogramma: SP punta alla cima, FP alla parola che contiene l'FP del chiamante.

Sottoprogrammi annidati

Se i sottoprogrammi sono annidati, prima di chiamare una seconda routine occorre salvare il contenuto del link register per recuperarlo in seguito. Si può usare la pila per salvare gli indirizzi di rientro delle chiamate annidate, nell'area di attivazione dei programmi chiamanti. Il link register è comunque sempre usato dalle funzioni di Call e Return. Alcune architetture non usano il link register e salvano l'indirizzo di rientro solo nella pila: questa strategia è detta **pila a modo implicito**.

Esempio: il programma principale PRO impila due parametri e chiama SUB1, che a sua volta chiama SUB2. Nel codice di SUB1 si noti il salvataggio e il ripristino di LINK_reg in pila.

```
; Programma principale
2000 PRO:  Load  R2, PARAM2          ; impila i parametri
2004        Subtract SP, SP, #4
2008        Store  R2, (SP)
2012        Load   R2, PARAM1
2016        Subtract SP, SP, #4
2020        Store  R2, (SP)
2024        Call   SUB1              ; chiama il sottoprogramma
2028        Load   R2, (SP)          ; immagazzina il risultato
2032        Store  R2, RIS
2036        Add    SP, SP, #8         ; ripristina il livello della pila
2040        prossima istruzione
```

```

; Primo sottoprogramma
2100 SUB1: Subtract SP, SP, #24      ; salva in pila i registri
2104      Store LINK_reg, 20(SP)
2108      Store FP, 16(SP)
2112      Store R2, 12(SP)
2116      Store R3, 8(SP)
2120      Store R4, 4(SP)
2124      Store R5, (SP)
2128      Add    FP, SP, #16        ; inizializza il puntatore all'area di attivazione
2132      Load  R2, 8(FP)          ; preleva il primo parametro
2136      Load  R3, 12(FP)         ; preleva il secondo parametro
      ...
      Load  R4, PARAM3            ; impila un parametro da passare a SUB2
      Subtract SP, SP, #4
      Store  R4, (SP)
      Call  SUB2
      Load  R4, (SP)              ; spila il risultato ottenuto da SUB2
      Add   SP, SP, #4
      ...
      Store  R5, 8(FP)            ; impila la risposta
      Load  R5, (SP)              ; ripristina i registri
      Load  R4, 4(SP)
      Load  R3, 8(SP)
      Load  R2, 12(SP)
      Load  FP, 16(SP)
      Load  LINK_reg, 20(SP)
      Add   SP, SP, #24
      Return                      ; rientro al programma principale

```

```

; Secondo sottoprogramma
3000 SUB2: Subtract SP, SP, #12    ; salva in pila i registri
3004      Store FP, 8(SP)
      Store R2, 4(SP)
      Store R3, (SP)
      Add   FP, SP, #8            ; inizializza il puntatore all'area di attivazione
      Load R2, 4(FP)             ; preleva il parametro
      ...
      Store R3, 4(FP)            ; impila il risultato di SUB2
      Load R3, (SP)              ; ripristina i registri
      Load R2, 4(SP)
      Load FP, 8(SP)
      Add   SP, SP, #12
      Return                      ; rientro al sottoprogramma 1

```

Passaggio di parametri

Una routine ha spesso bisogno di **parametri di ingresso** su cui operare e di **restituire un risultato** al programma chiamante. Esistono due tecniche:

1. **passaggio tramite registri del processore**: si usano alcuni registri generici per i parametri; il numero di parametri è limitato dal numero di registri;
2. **passaggio attraverso la pila**: i parametri si impilano; il numero di parametri è virtualmente illimitato.

ESEMPIO 1: SOMMA DI N NUMERI TRAMITE REGISTRI

Il chiamante passa la dimensione della lista in R2 (primo parametro) e l'indirizzo della lista in R4 (secondo parametro); il risultato torna in R3. Il sottoprogramma TOTALE salva e ripristina R5, che usa per i numeri.

```

; Programma chiamante
    Load R2, N           ; il parametro 1 è la dimensione della lista
    Move R4, #NUM1        ; il parametro 2 è la locazione della lista
    Call TOTALE            ; chiama il sottoprogramma
    Store R3, SOMMA        ; immagazzina il risultato

; Sottoprogramma
TOTALE: Subtract SP, SP, #4 ; salva in pila R5
    Store R5, (SP)
    Clear R3               ; inizializza la somma a 0
CICLO: Load R5, (R4)       ; preleva il prossimo numero
    Add R3, R3, R5          ; aggiunge questo numero alla somma
    Add R4, R4, #4          ; incrementa di 4 il puntatore
    Subtract R2, R2, #1      ; decrementa il contatore
    Branch_if_[R2]>0 CICLO
    Load R5, (SP)          ; ripristina il contenuto di R5
    Add SP, SP, #4
    Return                  ; rientra al programma chiamante

```

ESEMPIO 2: SOMMA DI N NUMERI TRAMITE PILA

Qui i parametri e il risultato passano dalla pila. Il chiamante impila l'indirizzo della lista e poi N; il sottoprogramma li legge a 16(SP) e 20(SP), dopo aver salvato quattro registri, e scrive il risultato nello slot del primo parametro (20(SP)). Al ritorno il chiamante lo spila da 4(SP) e ripristina la cima della pila.

```

; Programma chiamante
    Move R2, #NUM1        ; impila i parametri
    Subtract SP, SP, #4
    Store R2, (SP)
    Load R2, N
    Subtract SP, SP, #4
    Store R2, (SP)
    Call TOTALE            ; chiama il sottoprogramma (cima della pila a livello 2)
    Load R2, 4(SP)        ; spila il risultato e conservalo in SOMMA
    Store R2, SOMMA
    Add SP, SP, #8         ; ripristina la cima della pila (livello 1)

```

```

; Sottoprogramma
TOTALE: Subtract SP, SP, #16      ; salva in pila i registri
      Store R2, 12(SP)
      Store R3, 8(SP)
      Store R4, 4(SP)
      Store R5, (SP)              ; cima della pila a livello 3
      Load R2, 16(SP)            ; inizializza il contatore a n
      Load R4, 20(SP)            ; inizializza il puntatore alla lista
      Clear R3                    ; inizializza la somma a 0
CICLO: Load R5, (R4)              ; preleva il prossimo numero
      Add R3, R3, R5              ; aggiunge questo numero alla somma
      Add R4, R4, #4              ; incrementa di 4 il puntatore
      Subtract R2, R2, #1         ; decrementa il contatore
      Branch_if_[R2]>0 CICLO
      Store R3, 20(SP)            ; impila il risultato
      Load R5, (SP)              ; ripristina i registri
      Load R4, 4(SP)
      Load R3, 8(SP)
      Load R2, 12(SP)
      Add SP, SP, #16             ; cima della pila a livello 2
      Return                      ; rientra al programma chiamante

```

Istruzioni logiche, scorrimenti e aritmetiche

Le operazioni bit a bit, gli scorrimenti, la lettura di singoli byte e le istruzioni di moltiplicazione e divisione.

Lista delle istruzioni

ISTRUZIONE	VISUAL	EFFETTO
And R4, R2, R3	AND	Calcola l'AND bit a bit degli operandi nei registri R2 e R3 e lascia il risultato in R4.
Or R4, R2, R3	ORR	Calcola l'OR bit a bit degli operandi in R2 e R3 e lascia il risultato in R4.
XOr R4, R2, R3	EOR	Calcola l'XOR bit a bit degli operandi in R2 e R3 e lascia il risultato in R4.
LShiftL Ri, Rj, contatore	LSL	Scorrimento logico a sinistra: scorre il contenuto di Rj a sinistra del numero di posizioni dato da contatore e pone il risultato in Ri. Gli spazi vuoti si riempiono con 0.
LShiftR Ri, Rj, contatore	LSR	Scorrimento logico a destra: come sopra, ma verso destra. Gli spazi vuoti si riempiono con 0.
AShiftR R4, R2, #2	ASR	Scorrimento aritmetico a destra: scorre il contenuto di R2 a destra di 2 posizioni e pone il risultato in R4. Le posizioni lasciate libere si riempiono con il valore del bit più significativo.
AShiftL R4, R2, #2	LSL	Scorrimento aritmetico a sinistra: identico allo scorrimento logico a sinistra.
RotateL R4, R2, #2	nessuna	Rotazione verso sinistra.
RotateR R4, R2, #2	ROR	Rotazione verso destra.
LoadByte Rdst, LOCBYTE	LDRB	Legge un singolo byte dalla memoria e lo registra negli 8 bit meno significativi del registro destinazione, mettendo a 0 gli altri bit.
StoreByte Rsrc, LOCBYTE	STRB	Salva gli 8 bit meno significativi del registro sorgente nella locazione di memoria specificata.
Multiply Rk, Ri, Rj	nessuna	Moltiplica due numeri in complemento a due contenuti in Ri e Rj e salva il risultato in Rk. In genere in Rk vanno i bit meno significativi del prodotto; quelli più significativi non vengono calcolati.
Divide Rk, Ri, Rj	nessuna	Divisione intera tra due numeri in complemento a due contenuti in Ri e Rj, con il risultato in Rk. Non tutte le architetture posseggono le istruzioni di moltiplicazione e divisione.

Esercizi del laboratorio

Gli esercizi proposti nelle slide, raggruppati per argomento, con il risultato atteso. I valori con asterisco non sono riportati nelle slide e sono stati calcolati a mano.

Prime istruzioni

ESERCIZIO	RISULTATO ATTESO
Sommare i numeri 2, 3, 4, 1 e salvare in memoria il risultato.	10*
Eseguire le due operazioni $9 + 1$ e 9 meno 1 e salvare in memoria i risultati.	10* e 8*

Cicli e liste

ESERCIZIO	RISULTATO ATTESO
Data la sequenza 1, 2, 3, 4, 5, 6, 7, 8: (I) sommare a tutti i numeri la costante 10; (II) sommare i numeri ottenuti; (III) salvare il valore della somma in memoria.	116
Sequenza 1, 2, 3, 4, 5, 6, 7, 8: sommare i numeri di posizione pari; sottrarre i numeri in posizione dispari; salvare in memoria i due risultati.	20 e meno 16
Moltiplicare i numeri 9 e 3.	27*

Liste di valori

Voti. Tre studenti hanno eseguito 3 prove, ciascuna con voto da 18 a 30. Sommare i voti ottenuti dagli studenti per ogni prova e salvare il risultato in memoria. Il formato di ogni riga è ID ALLIEVO, VOTO1, VOTO2, VOTO3, con la lista 1, 20, 20, 20, 2, 18, 30, 27, 3, 22, 26, 28. Risultato atteso: 60, 76 e 75.

Altre quattro richieste sulla lista 1, 24, 12, 8, 10, 1, 4, 21, 3, 1, 5, 1:

ESERCIZIO	RISULTATO ATTESO
Trovare il valore 10 in memoria e salvare in memoria l'indirizzo corrispondente. Se non viene trovato, salvare meno 1.	L'indirizzo del quinto elemento della lista*
Contare il numero di volte che compare il valore 1 e salvare il risultato in memoria.	4
Sommare i numeri maggiori o uguali di x, con $x = 10$.	67

Pila

ESERCIZIO	RISULTATO ATTESO
Eseguire l'operazione $20 + 10$ meno 15 e salvare il risultato in memoria, usando la pila e conservando i vecchi valori contenuti nei registri.	15*

Minimo, massimo e numeri pari

Data la lista meno 10, 4, 23, 50, 0, 40, 10, 7:

ESERCIZIO	RISULTATO ATTESO
Trovare il minimo e il massimo e salvare il risultato in memoria.	minimo meno 10*, massimo 50*
Sommare i numeri pari e salvare il risultato in memoria.	94*

Operazioni su 1, 2, ..., 10

Data la lista 1, 2, 3, 4, 5, 6, 7, 8, 9, 10:

ESERCIZIO	RISULTATO ATTESO
Moltiplicare per 2 ogni elemento.	2, 4, 6, ..., 20*
Dimezzare tutti gli elementi.	0, 1, 1, 2, 2, 3, 3, 4, 4, 5* (divisione intera)
Contare tutti i numeri dispari e maggiori di 5 e salvare il risultato in memoria.	2* (sono 7 e 9)
Contare tutti i numeri pari e minori di 8 e salvare il risultato in memoria.	3* (sono 2, 4 e 6)

Conteggi e somme con condizioni

ESERCIZIO	RISULTATO ATTESO
Lista 10, 20, 12, 12, 5, 1, 5, 1: contare tutti i numeri diversi da 1 e da 10.	5*
Lista 30, 10, 23, 1, 17, 8, 19, 10: sommare tutti i numeri compresi tra 10 e 20.	56* con gli estremi inclusi, 36* con gli estremi esclusi
Lista 1, 2, 3, ..., 13: contare i numeri minori di 5, quelli pari a 5 e quelli maggiori di 5.	4*, 1* e 8*

Stringhe

Date le due stringhe ASCII T: «Ciao Carla!!» e P: «Ciao», verificare se P è una sottostringa di T e trovare l'indice della prima occorrenza in T. Nelle slide le stringhe sono dichiarate in byte esadecimali con DCB:

```
TSTRING DCB 0x43, 0x49, 0x41, 0x4F, 0x20, 0x43, 0x41, 0x52, 0x4C, 0x41, 0x21, 0x21
SUBSTR DCB 0x41, 0x49, 0x41, 0x4F
```

DA RICORDARE

Nelle slide la sequenza di SUBSTR è 0x41, 0x49, 0x41, 0x4F, che in ASCII è AIAO. Per la parola CIAO, come in TSTRING, il primo byte dovrebbe essere 0x43: probabile refuso della slide. Con CIAO la prima occorrenza è all'indice 0.