Calcolatori Elettronici
Linguaggio macchina

Ing. Gestionale e delle Telecomunicazioni

A.A. 2009/10

Gabriele Cecchetti

Linguaggio Macchina

- Sommario:
  - Generalità
  - Direttive di assemblatore
  - Chiamata a sottoprogramma
  - Gestione completa di sottoprogramma
- Riferimenti
  - C. Hamacher, “Introduzione all’architettura del Calcolatore”, cap. 5.
  - G. Bucci “Architetture e organizzazione dei Calcolatori Elettronici – Fondamenti”, Cap. ?

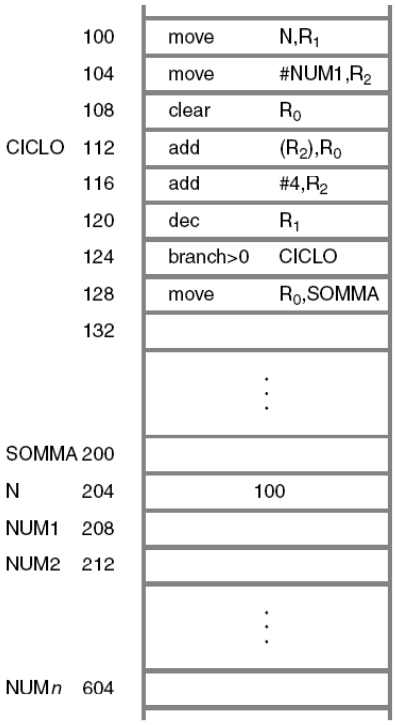
Alcune nozioni fondamentali

GENERALITÀ

Programma e Memoria

- Il codice macchina del programma in esecuzione si trova in memoria centrale.
- Le istruzioni macchina sono codificate in forma numerica e occupano una o più parole ciascuna.
- I dati sono pure collocati in memoria centrale, in un intervallo di indirizzi separato da quello dove si trovano le istruzioni macchina.

Esempio (notazione GAS)



esempio di collocazione in memoria centrale di programma e relativi dati

qui le istruzioni macchina sono denotate in forma simbolica, ma in realtà sono in forma numerica

per semplicità si suppone occupino ciascuna una sola parola di memoria

Processo di Assemblaggio

- Il processo di assemblaggio deve in qualche modo definire dove (cioè a quali indirizzi) collocare programma e dati.
- Ci sono anche altre questioni da affrontare:
  - come definire costanti simboliche ed etichette
  - come risolvere le etichette in indirizzi numerici
  - come designare porzioni di memoria per dati

Processo di Assemblaggio

- La collocazione del programma dipende in modo essenziale dal sistema operativo:
- Sistema operativo assente o non concorrente:
  - collocazione e indirizzi stabiliti in modo definitivo durante l'assemblaggio
- Sistema operativo concorrente con meccanismo di memoria virtuale:
  - collocazione e indirizzi stabiliti in modo provvisorio durante l'assemblaggio, e finalizzati da parte del S.O. al momento della messa in esecuzione del processo

Processo di Assemblaggio

- Il programma macchina in forma simbolica si presenta come una successione di:
  - istruzioni macchina simboliche
  - direttive (pure simboliche)
- Il processo di assemblaggio è sequenziale e procede nel modo seguente:
  - riconosce e trasforma le istruzioni in forma numerica, collocandole a indirizzi successivi
  - riconosce e interpreta le direttive

Come pilotare il processo di generazione del codice macchina

DIRETTIVE DI ASSEMBLATORE

Sistema di Assemblaggio GAS (1/2)

- Il sistema GAS (**G**nu **AS**sembler) è un assemblatore generico, capace di riconoscere il linguaggio macchina di numerose famiglie di processori (Motorola, Intel, HP, ARM e altre).
- Le direttive e il formato generale simbolico di GAS sono comuni a tutti i linguaggi.
- GAS si specializza solo nel riconoscere i nomi delle istruzioni macchina, che variano secondo il processore commerciale specifico da trattare.

Sistema di Assemblaggio GAS (2/2)

- Il sistema GAS è integrato nel sistema di compilazione GNU C/C++, per ambiente Unix/Linux (o simulato sotto Windows).
- Comunque GAS è usabile anche in forma separate dal compilatore C/C++.
- Dispone inoltre di sistema di simulazione (*debugger*), per vari tipi di processore.

- Per i dettagli sull'uso pratico di GAS, si vedano indicazioni e riferimenti nel libro.

Sistemi di assemblaggio MASM e MASM32

- Il sistema MASM (Microsoft ASseMbler) è assemblatore sviluppato da Microsoft per architettura x86 utilizzabile in ambiente DOS e Microsoft Windows.
- E' indirizzato a piattaforme a 16, 32 e 64 bit.
- Utilizza una sintassi diversa dal GAS, derivata dagli assemblatori AT&T a Intel.
- MASM non è più in vendita sebbene Microsoft continui a supportarlo.
- MASM32 è un progetto freeware compatibile con MASM.

Formato di Linea

etichetta: codice mnemonico lista argomenti commento

- Il formato di linea di programma (in linguaggio macchina simbolico) per entrambi gli assemblatori è mostrato sopra.
- Il "codice mnemonico" è il nome dell'istruzione macchina, e dipende dal processore su cui andrà eseguito il programma.
- Talvolta, invece dell'istruzione (simbolica), compare una direttiva, cioè un comando per il sistema di assemblaggio stesso.

Formato di Numero

- Decimale (intero, con segno):
  1, 17, -20, ecc
- Esadecimale (intero, positivo o nullo):
  0x1, 0x2, 0xA, 0xB, 0xF7, ecc
- Binario (intero, positivo o nullo):
  0b0, 0b1, 0b0101101, ecc
- Talvolta è anche disponibile il formato ottale (in base 8).

Direttiva di Assemblatore

- La direttiva somiglia a un'istruzione macchina in forma simbolica, ma non genera un'istruzione per il processore.
- Piuttosto, è un comando dato al sistema assemblatore, per pilotare il modo di generazione del codice macchina.
- L'assemblatore riconosce la direttiva, la interpreta e mette in opera, e la toglie dal programma (il processore non la vedrà).

Formato di Direttiva

- Per le istruzioni macchina specifiche di Motorola e Intel, si rimanda a:
  - Motorola 680x0 - cap. 6
  - Intel IA-32 - cap. 7
- Il libro (Hamacher) illustra anche una forma "generica" per le direttive, peraltro molto simile alla sintassi adottata dal sistema GAS.
- Nel seguito si adotta sia la sintassi GAS per che la sintassi MASM per direttive e notazione generale.

Dichiarazione di Simbolo

GNU GAS			Significato in GAS
	.equ	SIM, val_{32}	dichiara il simbolo "SIM" con valore "val_{32}" a 32 bit
SIM	=	val_{32}	

Sintassi MASM

```
SIM     EQU     0Ah
```

- La costante SIM è usabile come indirizzo simbolico, o come costante in calcoli.
- La direttiva è molto usata per associare simboli a indirizzi, rendendoli mnemonici e facilmente interpretabili.

Indirizzo Iniziale di Programma (GAS)

ETI	.org	num	allinea l'indirizzo (virtuale) di impianto del programma al valore "num"

- L'etichetta ETI è il valore simbolico dell'indirizzo di impianto del programma.
- Se non c'è memoria virtuale, "num" sarà l'indirizzo effettivo dove caricare il programma in memoria.
- Se c'è memoria virtuale, "num" sarà soggetto al meccanismo di rilocazione o di paginazione del S.O.
- Si può ripetere la direttiva, dividendo il programma in tronconi (segmenti) collocati a indirizzi diversi.
- Può anche definire dove collocare dati in memoria.

Modello di programma (MASM)

```
.MODEL small ; indica al compilatore il
       ; modello di memoria da usare
.STACK 100h ; dimensiona lo Stack
.DATA  ; inizio del segmento dati
  …
.CODE  ; inizio del segmento di codice
start: ; inizio del programma
  … ; codice
END start ; fine del programma
```


Modello semplificato di programma (MASM)

```
.MODEL small
.STACK ; Usa per default 1Kb di stack,
       ; altrimenti si specifica quanto ci
       ; serve es. .STACK 2048
.DATA  ; Inizio data segment
 …     ; dichiarazioni, variabili ecc..
.CODE  ; Inizio del segmento codice
.STARTUP ; Genera codice di avvio
 …     ; Le istruzioni del vostro programma
.EXIT  ; Genera codice di uscita
END    ; Deve stare alla fine di ogni modulo
```

Direttive di Supporto per il S.O. (GAS)

ETI .data	dichiara il segmento dati del programma
ETI .text	dichiara il segmento codice del programma

- Le due direttive servono per dichiarare i segmenti codice e dati, se è presente un sistema operativo con memoria virtuale e segmentazione del processo.
- Altrimenti, non sono utilizzabili.

Indirizzo Iniziale di Programma (MASM)

- L'indirizzo iniziale del programma viene dato applicando un'etichetta alla prima procedura che deve essere eseguita e indicando tale etichetta nella direttiva END.

```
start:
  ...
end start
```

Blocco di Dati (non inizializzati) (GAS)

ETI .space num	alloca come spazio di memoria un numero di byte pari a "num" (senza dare loro valore iniziale)

- L'etichetta ETI è il valore simbolico dell'indirizzo iniziale del blocco di byte consecutivi.
- La direttiva serve per riservare in memoria una regione destinata a contenere dati vari.
- Si può ripetere per dichiarare due o più regioni.

Blocco di dati non inizializzati (MASM)

- Esempio:

```
.data? ; uninitialised data section
    align 4
    blankhandle dd (?)
    txtbuffer1 db 128 dup (?)
    txtbuffer2 db 128 dup (?)
```

Dati Inizializzati (GAS)

ETI	.byte	val_8 [, val_8, …]	alloca come spazio di memoria un byte (o una lista di byte), dandogli (dando loro) valore iniziale "val_8"
ETI	.word	val_{16} [, val_{16}, …]	idem, ma con parole a 16 bit
ETI	.long	val_{32} [, val_{32}, …]	idem, ma con parole a 32 bit
ETI	.quad	val_{32} [, val_{32}, …]	idem, ma con parole a 64 bit

- La direttiva funziona come ".space", ma inizializza i dati dichiarati.

Dati inizializzati (MASM)

```
VAR1 DB   255          ; byte
VAR2 DW   FFFFh        ; 16 bit
VAR3 DQ   FFFFFFFFh    ; 32 bit

VAR4 DB 10 DUP (3); VAR4 è definita da
   ; 10 byte inizializzati al valore 3.
VAR5 DB 15 DUP (?) ; VAR5 è definita da
   ; 15 byte non inizializzati.
```

Dati di Tipo Stringa (GAS)

ETI	.ascii	“stringa”	alloca come spazio di memoria un numero di byte pari alla lunghezza della stringa, e dà loro come valori iniziali i car. della stringa
ETI	.asciz	“stringa”	idem, aggiunge `\0´

- La direttiva è simile a ".space", ma dimensiona e inizializza il blocco di byte con la stringa data come argomento.

Dati di tipo stringa (MASM)

```
MSG DB “Hello world”, 0
; La stringa MSG è inizializzata al
  valore “Hello world” ed è terminata da
  uno 0.
```

Esempio (per M68000, GAS)

commento	etichetta	dir. / istr.	argomento / i
direttive dell'assemblatore	C	=	0x202200
(o equivalentemente		.equ	C, 0x202200)
		.org	0x201150
	A:	.word	639
	B:	.word	−215
		.org	0x201200
istruzioni macchina che genera codice	MAIN:	MOVE	A, D0
		ADD	B, D0
		MOVE	D0, C

D0 è un generico registro interno nel processore M68000

Esempio (per IA-32, MASM)

```
C EQU      0x202200
.MODEL small
.DATA
 A    DW    639
 B    DW    -215
.CODE
.STARTUP
 MOV AX, A
 ADD AX, B
 MOV C, AX
.EXIT
END
```

Alcune nozioni fondamentali chiamata - rientro - parametri

CHIAMATA A SOTTOPROGRAMMA

Chiamata a Sottoprogramma

- Nei linguaggi di alto livello, come il C, la chiamata a sottoprogramma (routine) ha come effetto la creazione sulla pila di un'area (o record) di attivazione, associata alla chiamata, contenente le informazioni seguenti:
  - **i parametri formali** (e il loro valore)
  - **l'indirizzo di rientro al chiamante**
  - **informazioni per gestire l'area di attivazione**
  - **le variabili locali del sottoprogramma**
  - **l'eventuale valore restituito** (se si ha una funzione)

Chiamata a Sottoprogramma

- A livello di linguaggio macchina, la chiamata a sottoprogramma viene espansa in più istruzioni macchina, eseguite in “ambiti diversi”:
  - **il chiamante gestisce la parte relativa al passaggio dei parametri;**
  - **il chiamante attiva il sottoprogramma con l’istruzione di chiamata a routine;**
  - **l’esecuzione dell’istruzione di chiamata a routine gestisce la parte relativa al salvataggio dell’indirizzo di rientro (PC) sulla pila e attiva il sottoprogramma;**
  - **la routine gestisce l’allocazione sulla pila delle variabili locali e dell’eventuale valore restituito.**

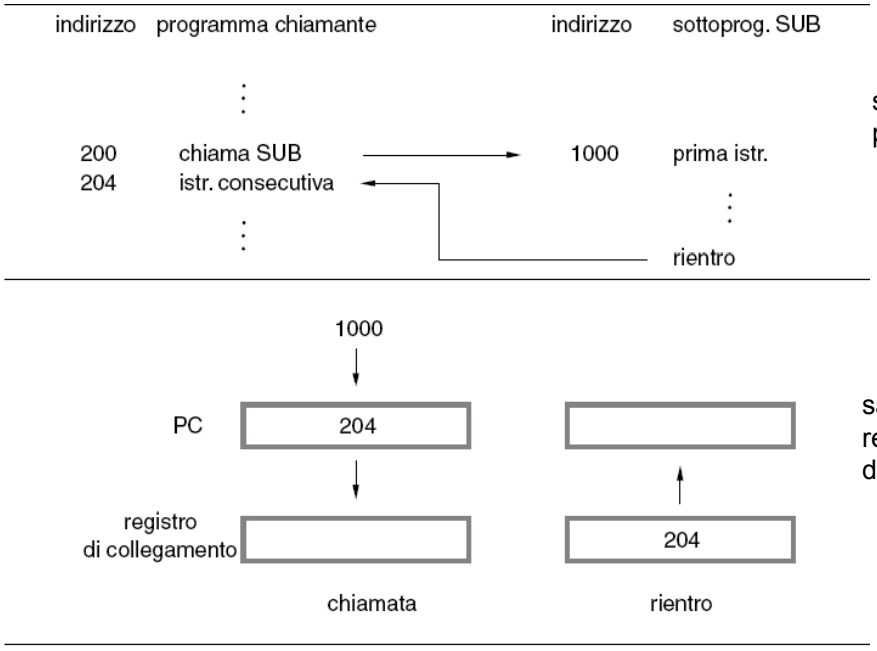
Chiamata a Sottoprogramma

- La chiamata a routine segue l'espansione vista, ma il modello per la chiamata non è identico in tutti i processori:
  - alcuni processori vincolano la modalità
  - in altri è possibile operare delle scelte
- Dove salvare i parametri attuali (registri o pila) ?
- In quale ordine passare i parametri attuali ?
- Dove salvare i registri usati dalla routine (e chi li salva ?)
- Dove restituire il valore calcolato (registro o pila) ?
- Chi deve deallocare lo spazio in pila eventualmente utilizzato per parametri e variabili locali ?
- Si definisce **l’area di attivazione** ? Come viene gestita ?

Soluzione Semplice

- In casi semplici il meccanismo di chiamata e rientro a e da routine si basa su un registro di collegamento nel processore (*link register*).
- **Un'istruzione apposita salta alla routine, e salva nel registro l'indirizzo di rientro** (di norma quello dell'istruzione consecutiva alla chiamata).
- **Un'istruzione apposita pone termine alla routine ricopiando l'indirizzo di rientro dal registro di collegamento nel contatore di programma**.
- Non si può gestire l'annidamento di routine.

Registro di Collegamento



Uso del registro di collegamento.

Uso della Pila

- **Per gestire l'annidamento di routine** (e la ricorsione) **si usa la pila** (stack).
- Ma **la pila può anche servire per passare parametri alla routine, e per recuperare il risultato della routine** (se è una funzione).
- Inoltre **la pila può servire alla routine per tenere le proprie variabili locali** (se ne ha).
- Infine **la pila è usabile per salvare i registri** (che di solito sono pochi di numero), quando occorre fare nuovo spazio nel processore.

Istruzione di Chiamata

- Istruzione di salto a sottoprogramma:
  - **BSR** **indirizzo_routine** (*branch to subroutine*)
  - **JSR** **indirizzo_routine** (*jump to subroutine*)
  - **CALL** **indirizzo_routine** (*call subroutine*)
- Salto incondizionato che salva sulla pila il contatore di programma (cioè l' indirizzo di ritorno) e salta all'indirizzo specificato come argomento:
- Operazione in RTL (istruzione ingombra 4 byte):

  SP ← [SP] − 4

  [SP] ← [PC]

  PC ← indirizzo routine

Istruzione di Rientro

- Istruzione di rientro da sottoprogramma:
  - **RTS** (*return from subroutine*)
  - **RET** (return)
- *Spila* l'indirizzo di rientro a programma chiamante (quello successivo all'istruzione di chiamata a routine) e salta a tale indirizzo:
- Operazione in RTL (istruzione ingombra 4 byte):

  PC ← [[SP]]

  SP ← [SP] + 4

Passaggio di Parametri

- **L'operazione di passaggio dei parametri viene svolta dal programma chiamante**.
- Ci sono due modi distinti per passare parametri.
  - **Nei registri** (soluzione spiccia, ma limitata):
    - i registri sono limitati e si esauriscono rapidamente
    - in chiamate ricorsive si deve poi impilarli (o piuttosto usare da subito la pila, vedi sotto)
  - **Sulla pila** (è la scelta prevalente):
    - **con salvataggio esplicito da parte del chiamante**
    - **passando i parametri nell'ordine di elencazione** (il primo parametro è il primo a essere impilato)

Restituzione di Valore

- **L'operazione di restituzione viene svolta dalla routine** (se è una funzione, naturalmente).
- Di nuovo, ci sono due modi distinti per farla.
  - **Nei registri** (soluzione spiccia, ma limitata),
  - **Sulla pila, sovrascrivendo il primo parametro passato** (è la scelta prevalente):
    - **se la routine non ha parametri, il chiamante deve allocare sulla pila una parola per il valore restituito;**
    - **il chiamante deve prevedere, dopo l'istruzione di chiamata, un'istruzione che *spili* il valore restituito e lo salvi in memoria o registro.**

Salvataggio dei Registri

- *Se una routine usa registri del processore dove si trovano valori utili per il chiamante al rientro, essi vanno salvati impilandoli*.
- **Il salvataggio viene svolto dalla routine, di solito non appena essa parte**.
- **Il recupero (o ripristino) dei registri salvati viene svolto dalla routine, di solito subito prima di terminare**.

Due Esempi

- Programma che calcola la somma di una lista (vettore, array) di numeri interi (da una parola).
- **La scansione della lista e il calcolo della somma sono svolti da una routine**:
  - parametri: lunghezza e indirizzo iniziale lista
  - valore restituito: somma degli elementi
- Due versioni:
  - con parametri e somma nei registri
  - con parametri e somma in pila
- Si suppone di lavorare con un processore generico (registri interni denotati R0, R1, ecc).

Passaggio in Registri (notaz. GAS: sorg. - dest.)

etichetta	istruzione		commento
programma chiamante			
	move	N, R_1	carica in R_1 (che contiene la lunghezza della lista) la parola di memoria di indirizzo N
	move	#NUM1, R_2	carica in R_2 la costante di valore NUM1 (secondo parametro)
	call	TOTALE	impila l'indirizzo di rientro e passa alla routine di indirizzo TOTALE
... ora avviene l'esecuzione della routine TOTALE ...			
	move	R_0, SOMMA	memorizza R_0 (che contiene il risultato calcolato dalla routine TOTALE) nella parola di indirizzo SOMMA
	...	...	

Calcolo della somma di una lista di numeri.

Passaggio in Registri

sottoprogramma chiamato			
TOTALE	clear	R_0	azzera il registro R_0
CICLO	add	$(R_2)+, R_0$	addiziona a R_0 la parola di memoria puntata da R_2 e scrivi la somma in R_0; ciò fatto incrementa il contenuto di R_2
	dec	R_1	decrementa il registro R_1
	branch > 0	CICLO	se l'istruzione aritmetica precedente ha dato risultato maggiore di zero salta all'istruzione di etichetta CICLO; altrimenti passa all'istruzione consecutiva
	return		spila l'indirizzo di rientro e torna al programma chiamante

Routine che effettua il calcolo.

Passaggio su Pila

etichetta	istruzione		commento
... la cima della pila si trova a livello 1 ...			
	move	#NUM1, – (SP)	impila la costante di valore NUM1 (primo parametro)
	move	N, – (SP)	impila la parola di memoria di indirizzo N (secondo parametro)
	call	TOTALE	impila l'indirizzo di rientro e passa alla routine di indirizzo TOTALE - *cima della pila a livello* 2
... ora avviene l'esecuzione della routine TOTALE ...			

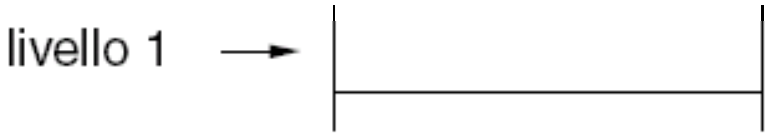
programma chiamante (prima parte - impila parametri)

Passaggio su Pila

... ora avviene l'esecuzione della routine TOTALE ...		
move	4 (SP), SOMMA	prendendola dall'interno della pila, copia nella parola di memoria di indirizzo SOMMA la parola che in origine conteneva l'indirizzo di testa della lista NUM1 (primo parametro) *e che ora contiene il risultato calcolato dalla routine TOTALE*
add	#8, SP	addiziona a SP la costante 8 abbandonando i due elementi in cima alla pila - *cima della pila a livello* 1
...	...	

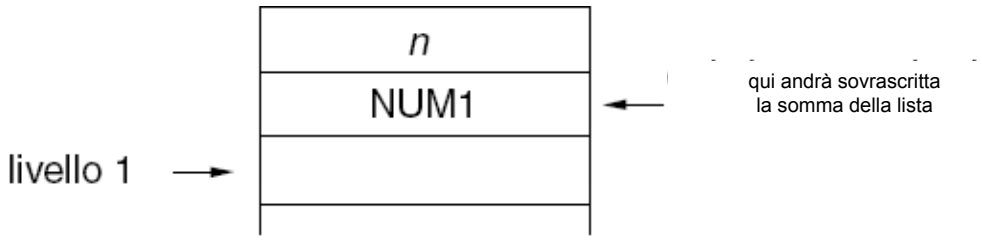
programma chiamante (seconda parte - spila valore restituito)

Struttura della Pila



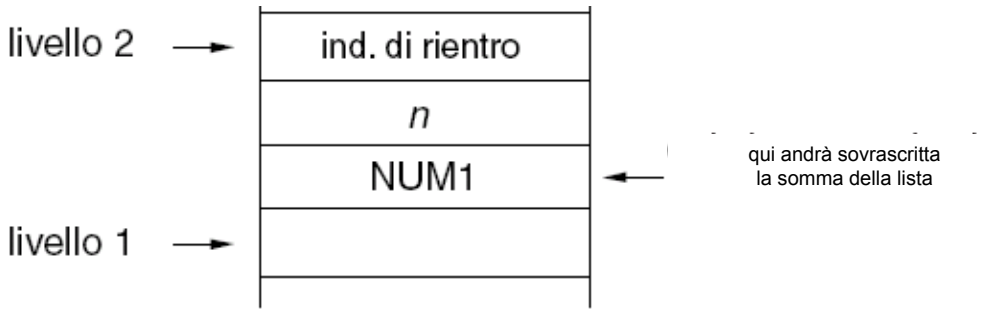
struttura iniziale della pila

Struttura della Pila



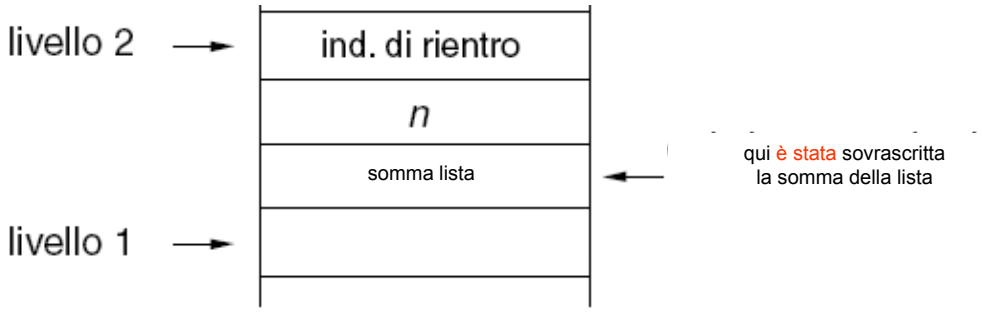
struttura della pila dopo avere passato i parametri
e subito prima di chiamare la routine

Struttura della Pila



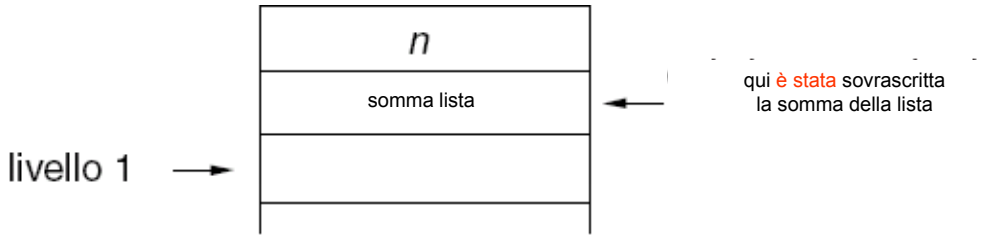
struttura della pila non appena la routine parte

Struttura della Pila



struttura della pila subito prima che la routine termini

Struttura della Pila



struttura della pila subito dopo che la routine è terminata

Struttura della Pila

struttura finale della pila, dopo che il programma chiamante
ha recuperato la somma della lista e abbandonato i parametri

Passaggio su Pila

etichetta	istruzione		commento
TOTALE	move block	R_0-R_2, – (SP)	impila la terna di registri R_0, R_1 e R_2, salvandoli in pila - *cima della pila a livello* 3
	move	16 (SP), R_1	prendendola dall'interno della pila, carica in R_1 la lunghezza della lista *n* (secondo parametro)
	move	20 (SP), R_2	prendendola dall'interno della pila, carica in R_2 l'indirizzo di testa della lista NUM1 (primo parametro)
	clear	R_0	azzera il registro R_0

routine chiamata (prima parte - recupero parametri)

Passaggio su Pila

CICLO	add	$(R_2)+, R_0$	addiziona la parola di memoria puntata da R_2 al contenuto di R_0 e scrivi la somma in R_0; poi incrementa il registro R_2
	dec	R_1	decrementa il registro R_1
	branch > 0	CICLO	se l'istruzione aritmetica precedente ha dato risultato maggiore di zero salta all'istruzione di etichetta CICLO; altrimenti passa all'istruzione consecutiva

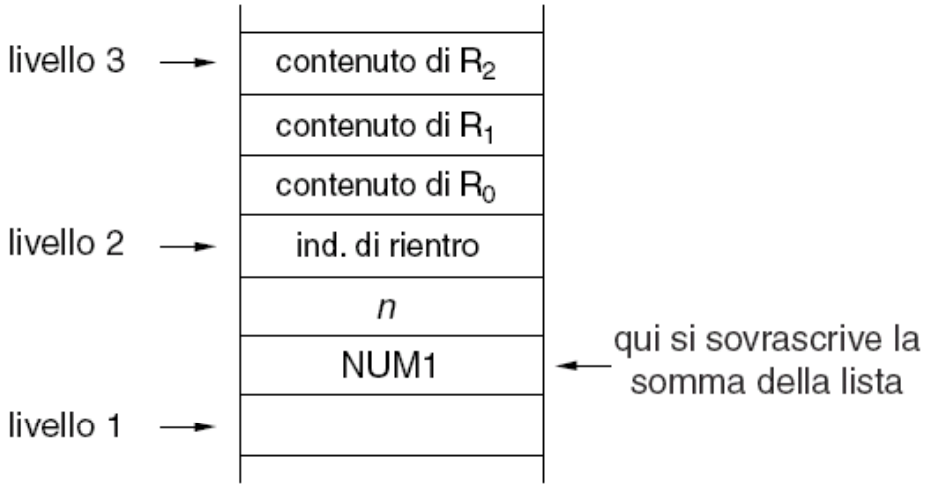
routine chiamata (seconda parte - ciclo)

Passaggio su Pila

move	R_0, 20 (SP)	prendendola da R_0, memorizza all'interno della pila la somma della lista, sovrascrivendo l'indirizzo di testa della lista NUM1 (primo parametro)
move block	$(SP)+, R_0$-R_2	spila la terna di registri R_0, R_1 e R_2, ripristinandoli nel processore - *cima della pila a livello* 2
return		spila l'indirizzo di rientro e torna al programma chiamante

routine chiamata (terza parte - restituzione valore)

Movimentazione della Pila



struttura della pila nel momento di massima espansione

Area di attivazione, parametri e variabili locali

GESTIONE COMPLETA DI SOTTOPROGRAMMA

Area di Attivazione

- Per realizzare meglio il meccanismo di chiamata a routine, si usa l'area di attivazione (*stack frame* o *activation record*).
- **L'area di attivazione è una porzione di pila** (formata da celle consecutive), **che contiene tutto il "pacchetto" di informazioni di cui la routine abbisogna per funzionare correttamente**.
- Le aree sono impilate una sopra l'altra, in ordine di chiamata delle routine corrispondenti.
- L'area della routine correntemente attiva è quella sommitale in pila.

Area di Attivazione (1/2)

- L'area di attivazione ha struttura a strati (come un sandwich).
- Strati in ordine (dal basso):
  - parametri in ingresso alla routine / valore restituito al chiamante (viene sovrascritto ai parametri)
  - elementi di servizio:
    - indirizzo di rientro al chiamante
    - informazioni di servizio (frame pointer)
  - variabili locali della routine
  - eventuali registri salvati da parte della routine
- Sopra la sua area di attivazione, la routine attiva ha spazio per eventuali calcoli di espressione.

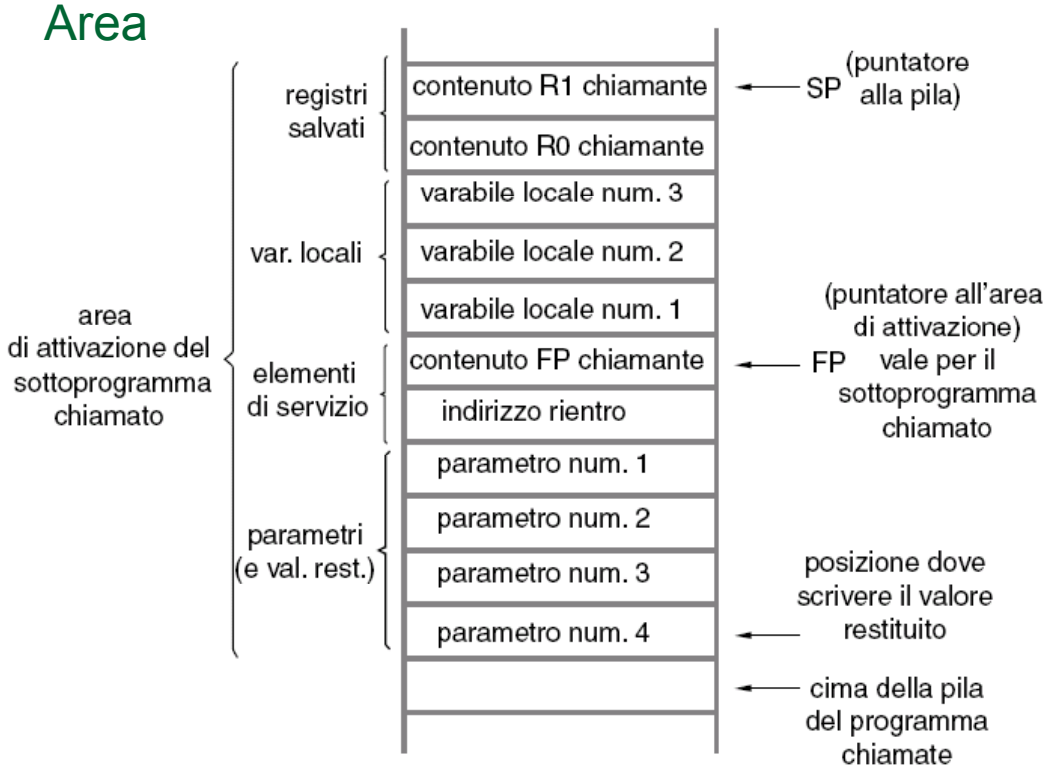
Area di Attivazione (2/2)

- Le aree di attivazione impilate formano una lista concatenata.
- Il puntatore all'area di attivazione corrente (*frame pointer* o FP) sta in un registro di uso generale, designato a tale scopo.
- I parametri hanno spiazzamento positivo rispetto a FP.
- Le variabili locali hanno spiazzamento negativo rispetto a FP.

Esempio di Area

- Routine (funzione) dotata di:
  - quattro parametri (da una parola ciascuno)
  - un valore restituito (una parola, viene sovrascritto al primo parametro impilato)
  - tre variabili locali (da una parola ciascuna)
- La routine deve usare due registri (R0 e R1), in uso anche da parte del chiamante, e li salva in pila nella parte sommitale della sua area.
- Infine, l'area contiene le informazioni di servizio:
  - indirizzo di rientro al chiamante
  - valore del puntatore all'area del chiamante

Area



Puntatore all'Area

- Il registro FP del processore contiene il puntatore alla posizione di mezzo dell'area.
- Tale posizione di mezzo è precisamente la parola destinata a salvare il contenuto di FP valevole nel cotesto del chiamante.
- Usando tale registro FP in modo di indirizzamento con spiazzamento, si accede ai componenti dell'area:
  - parametro: spiazzamento positivo
  - variabile locale: spiazzamento negativo
- I registri da salvare si impilano e spilano con “push” e “pop” (o operazioni equivalenti).

Puntatore all'Area

- Siccome la routine attiva usa il registro FP, il valore che esso conteneva quando era attivo il programma chiamante va salvato da qualche parte.
- In concreto, lo si salva proprio nella posizione di mezzo dell'area corrente, dove punta il nuovo FP usato dalla routine.
- Il salvataggio e il ripristino di FP del chiamante vanno svolti dalla routine stessa, subito dopo essere partita e subito prima di terminare, rispettivamente.
- Alcuni processori dispongono di istruzioni macchina apposite per salvare e ripristinare lo FP del chiamante (vedi p. es. M68000 – istruzioni LINK e UNLK).
- Altrimenti, si usano istruzioni generiche di pila.

Gestione dell'Area

- La costruzione dell'area di attivazione viene:
  - iniziata dal chiamante, subito prima della chiamata:
    - impila i parametri da passare alla routine
    - chiama la routine, impilando l'indirizzo di rientro per salvarlo
  - completata dalla routine, subito dopo essere partita:
    - impila lo FP del chiamante, per salvarlo
    - fa spazio in pila per le variabili locali (fallo innalzando la pila) e se serve inizializzale (variabili con valore iniziale dichiarato)
    - eventualmente impila alcuni (o tutti) i registri, per salvarli
- Poi la routine esegue i suoi calcoli interni, usando parametri e variabili nell'area.
- Eventualmente essa chiama un'altra routine, o anche sé stessa ricorsivamente ...

Gestione dell'Area

- La distruzione dell'area di attivazione viene:
  - iniziata dalla routine, subito prima di terminare:
    - sovrascrivi il risultato (se c'è) ai parametri
    - se serve spila i registri salvati, per ripristinarli
    - abbandona le variabili locali (fallo abbassando la pila)
    - spila lo FP del chiamante, per ripristinarlo
    - rientra al chiamante, spilando l'indirizzo di rientro e mettendolo nel contatore di programma
  - completata dal chiamante, subito dopo il rientro:
    - spila il valore restituito (se c'è)
    - abbandona i parametri (fallo abbassando la pila)
- Poi il chiamante prosegue con i suoi calcoli interni, usando il risultato della routine.

Struttura della Pila

- La gestione dell'area fa sì che quando ci sono chiamate a routine innestate (o ricorsive):
  - la pila contiene altrettante aree impilate una sopra l'altra, nello stesso ordine delle chiamate a routine
  - l'area sulla sommità della pila corrisponde alla routine correntemente attiva (chiamata più di recente)
  - quella in fondo alla pila corrisponde al programma principale (attivato per primo da parte del S.O.)
  - le aree sono collegate come lista concatenata: in ogni area lo FP salvato punta alla posizione di mezzo dell'area immediatamente sottostante ... e così via
  - l'area in fondo alla pila (prog. princ.) ha uno FP salvato non significativo (idealmente punta al S.O.)

Esempio Strutturato - PRO

- Di seguito si dà un esempio strutturato, con chiamate annidate di routine.
- Per costruire (e demolire) le aree di attivazione si usano istruzioni generiche.
- Per processori commerciali ci sono alcune istruzioni specializzate (vedi Motorola e Intel).
- PRO: programma principale
  - impila due parametri PAR1 e PAR2
  - chiama la routine SUB1
  - usa il risultato RIS di SUB1
- Nota: SUB1 a sua volta chiama SUB2 …

Stato della Pila

regione di memoria dove verranno progressivamente impilate le aree di attivazione di SUB1 (e SUB 2)

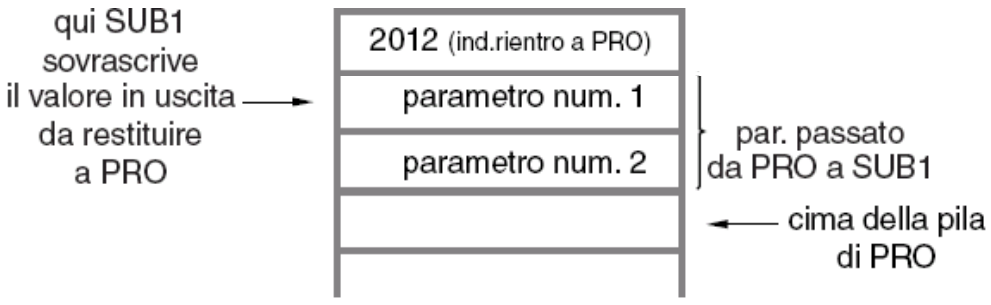
area di attivazione di PRO

Codice

ind.	etich.		istruzione	commento
...		...	...	PRO svolge attività che precedono la chiamata a SUB1 e verosimilmente calcola i parametri num. 1 e 2 da passare a SUB1
2000	PRO	move	PAR2, – (SP)	leggendolo dalla parola di memoria all'indirizzo PAR2, impila il parametro num. 2 per passarlo a SUB1 - *pila su*
2004		move	PAR1, – (SP)	leggendolo dalla parola di memoria all'indirizzo PAR1, impila il parametro num. 1 per passarlo a SUB1 - *pila su*
2008		call	SUB1	leggendolo da PC, impila l'indirizzo di rientro 2012 e poi scrivi in PC l'indirizzo SUB1 (2100) - *pila su*
... ora avviene l'esecuzione di SUB1 ...				

programma principale PRO (prima parte)

Stato della Pila



stato della pila quando SUB1 parte

Codice

... ora avviene l'esecuzione di SUB1 ...			
2012	move	(SP), RIS	leggendolo dalla cima della pila, copia nella parola di memoria all'indirizzo RIS il risultato di SUB1 - *lavora in area*
2016	add	#8, SP	addiziona 8 a SP, riposizionando SP in modo da abbandonare i due parametri posti sulla cima della pila - *pila giù*
...	...	...	PRO svolge attività che seguono la chiamata a SUB1 e verosimilmente si serve del risultato di SUB1

programma principale PRO (seconda parte)

Stato della Pila

qui c'erano le aree di attivazione di SUB1 e SUB2 poi demolite via via che SUB2 e SUB1 sono terminate

area di attivazione di PRO

Esempio Strutturato - SUB1

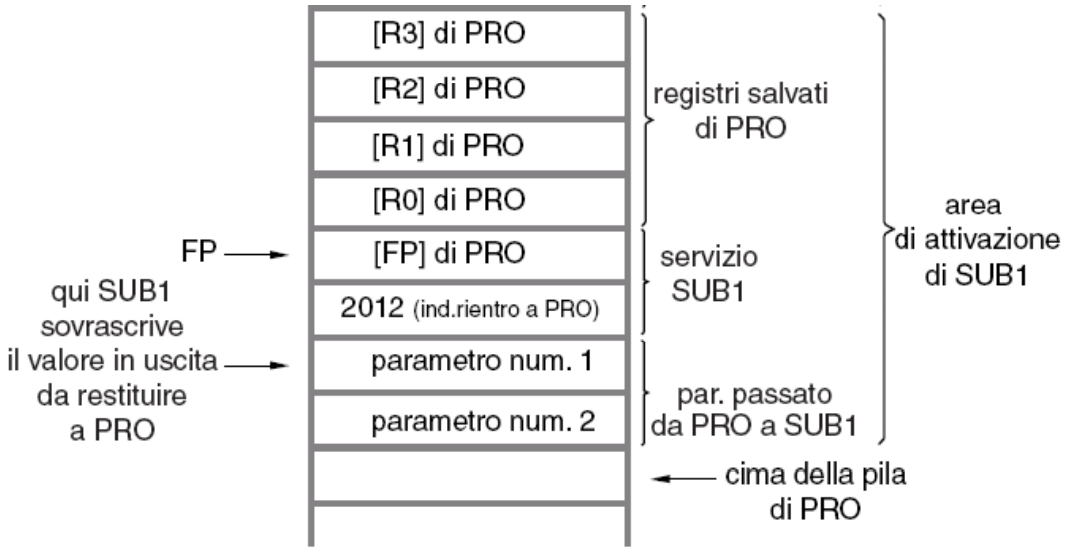
- SUB1: routine chiamata (1º livello)
  - impila il registro FP di PRO, per salvarlo
  - scrive nel registro FP un nuovo valore
  - impila il blocco di registri R0-R3, per salvarli
  - legge dalla pila i suoi parametri PAR1 e PAR2 e li carica nei registri
- Nota bene: SUB1 non ha variabili locali (ma le potrebbe benissimo avere).

Codice

ind.	etich.	istruzione	commento
2100	SUB1 move	FP, – (SP)	leggendolo da FP, impila il puntatore all'area di attivazione di PRO, salvandolo per recupero futuro - *pila su*
2104	move	SP, FP	copia SP in FP, posizionando FP in modo da riferirsi all'area di attivazione di SUB1 - *cambio di area di attivazione*
2108	move block	R_0-R_3, – (SP)	impila la quaterna di registri R_0, R_1, R_2 e R_3, salvandoli per recupero futuro - *pila su*
2112	move	8 (FP), R_0	leggendolo dall'area di attivazione corrente, carica in R_0 il parametro num. 1 per usarlo - *lavora in area*
2112	move	12 (FP), R_1	leggendolo dall'area di attivazione corrente, carica in R_1 il parametro num. 2 per usarlo - *lavora in area*

routine SUB1 (prima parte)

Stato della Pila



stato della pila di SUB1

Esempio Strutturato - SUB1

- SUB1: routine chiamata (1º livello)
  - esegue suoi calcoli interni
  - impila un parametro PAR3
  - chiama la routine SUB2
- Ora è attiva la routine SUB2 (vedi più avanti per la descrizione).

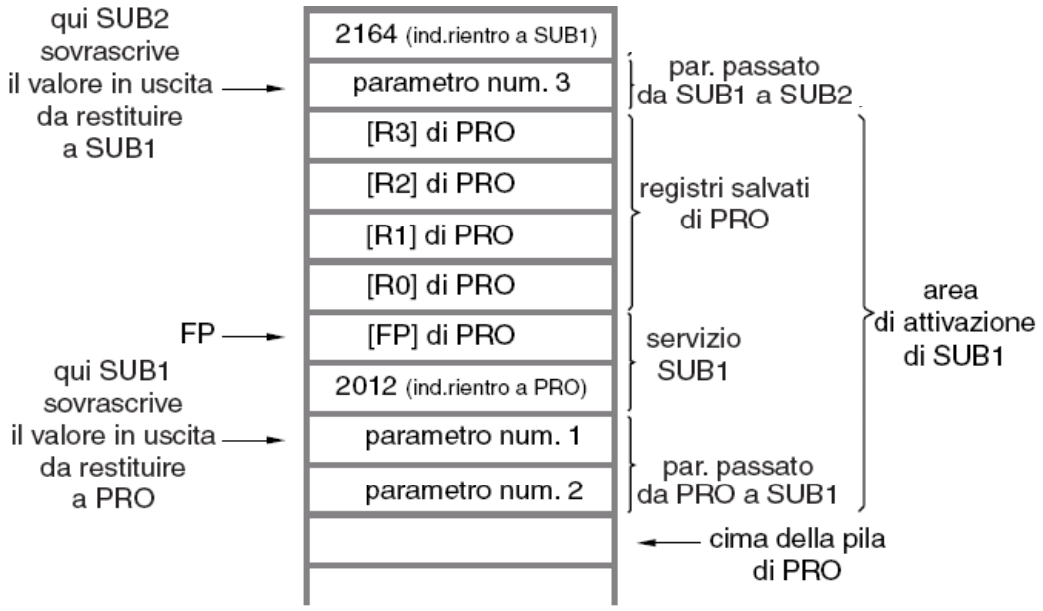
Codice

...	...	...	SUB1 svolge attività che precedono la chiamata a SUB2 e verosimilmente calcola il parametro num. 3 servendosi dei parametri num. 1 e 2
...	move	PAR3, – (SP)	leggendolo dalla parola di memoria all'indirizzo PAR3, impila il parametro num. 3 per passarlo a SUB2 - *pila su*
2160	call	SUB2	leggendolo da PC, impila l'indirizzo di rientro 2164 e poi scrivi in PC l'indirizzo SUB2 (3000) - *pila su*

... ora avviene l'esecuzione di SUB2 ...

routine SUB1 (seconda parte)

Stato della Pila



stato della pila di SUB1 e SUB2

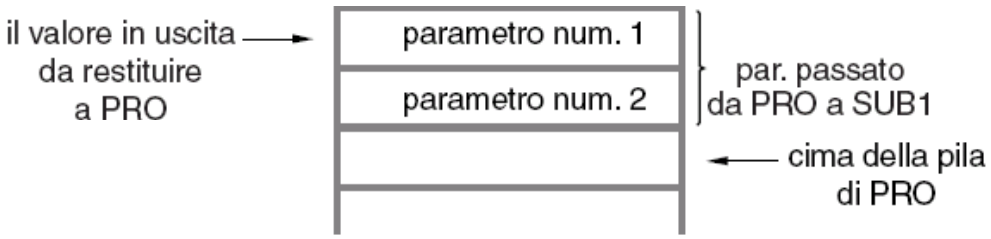
Esempio Strutturato - SUB1

- Prima o poi SUB2 termina e SUB1 riprende, nel modo seguente:
  - legge dalla pila il risultato di SUB2 (che SUB2 avrà sovrascritto a PAR3) e lo carica in un registro
  - esegue i suoi calcoli interni
  - sovrascrive in pila il suo risultato a PAR1
  - spila il blocco di registri R0-R3, per ripristinarli
  - spila lo FP di PRO, per ripristinarlo
  - rientra al chiamante PRO, spilando l'indirizzo di rientro e ricaricandolo nel contatore di programma

Codice

2164	move	(SP) +, R_2	spila il risultato di SUB2, scrivendolo in R_2 - *pila giù*
...	...	...	SUB1 calcola il proprio risultato, verosimilmente servendosi di quello di SUB2 nonché dei parametri num. 1 e 2, e lo lascia in R_3
...	move	R_3, 8 (FP)	leggendolo da R_3, memorizza nell'area di attivazione corrente il risultato di SUB1, sovrascrivendo il parametro num. 1 - *lavora in area*
...	move block	(SP) +, R_0-R_3	spila, scrivendola in R_0, R_1, R_2 e R_3, la quaterna di registri salvati prima, così ripristinandoli - *pila giù*
...	move	(SP) +, FP	spila, scrivendolo in FP, il puntatore all'area di attivazione di PRO salvato prima, così ripristinandola - *pila giù e cambio di area di attivazione*
...	return		spila l'indirizzo di rientro a PRO, scrivendolo in PC - *pila giù*

Stato della Pila



stato della pila di SUB1

Esempio Strutturato - SUB2

- SUB2: routine chiamata (2º livello)
  - procede alla costruzione della sua area di attivazione, similmente a SUB1
  - l'area di SUB2 ha dimensioni diverse da quella di SUB1 (vedi schema avanti)
  - al termine demolisce la sua area, idem SUB1
- Nota bene: SUB2 non ha variabili locali (ma le potrebbe benissimo avere).

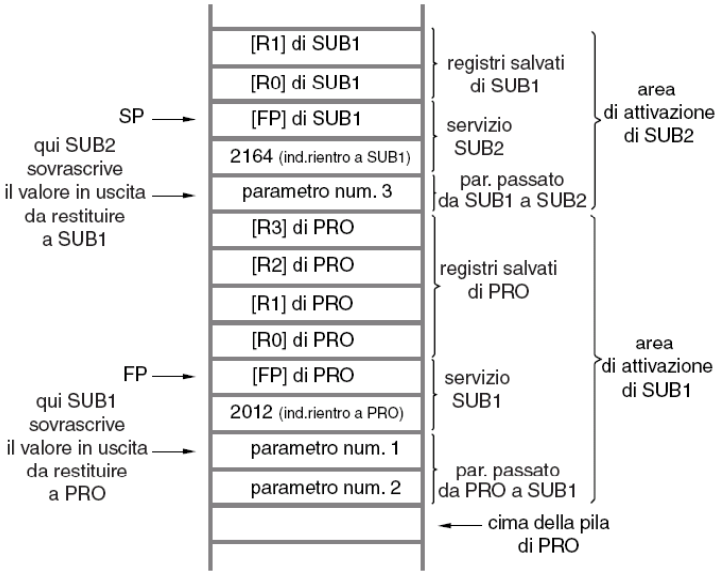
Codice

3000 SUB2	move	FP, – (SP)	leggendolo da FP, impila il puntatore all'area di attivazione di SUB1, salvandolo per recupero futuro - *pila su*
...	move	SP, FP	copia SP in FP, posizionando FP in modo da riferirsi all'area di attivazione di SUB2 - *cambio di area di attivazione*
...	move block	R_0-R_1, – (SP)	impila la coppia di registri R_0 e R_1, salvandoli per recupero futuro - *pila su*
...	move	8 (SP), R_0	leggendolo dall'area di attivazione corrente, carica in R_0 il parametro num. 3 per usarlo - *lavora in area*
...	...	...	SUB2 calcola il proprio risultato, verosimilmente servendosi del parametro num. 3, e lo lascia nel registro R_1

Codice

...	...	...	SUB2 calcola il proprio risultato, verosimilmente servendosi del parametro num. 3, e lo lascia nel registro R_1
...	move	R_1, 8 (SP)	leggendolo da R_1, memorizza nell'area di attivazione corrente il risultato di SUB2, sovrascrivendo il parametro num. 3 - *lavora in area*
...	move block	(SP) +, R_0-R_1	spila, scrivendola in R_0 e R_1, la coppia di registri salvati prima, così ripristinandoli - *pila giù*
...	move	(SP) +, FP	spila, scrivendolo in FP, il puntatore all'area di attivazione di SUB1 salvato prima, così ripristinandola - *pila giù e cambio di area di attivazione*
...	return		spila l'indirizzo di rientro a SUB1, scrivendolo in PC - *pila giù*

Stato della Pila



stato della pila di SUB1 e SUB2,
nel momento di massima espansione