Capitolo 6
Il repertorio delle istruzioni

Parte I

Copyright © 2009 – The McGraw-Hill Companies srl

www.isti.cnr.it/people/

Copyright © 2009 – The McGraw-Hill Companies srl

Programmi

Programma = traduzione in un linguaggio formale (linguaggio di programmazione con una sua sintassi) di un algoritmo (procedimento generale che risolve in un numero finito di passi un problema).

Istruzioni come quelle viste fino ad ora sono istruzioni semplice dette istruzioni macchina

Diversi livelli di astrazione.

Assembler è il più rudimentale linguaggio di programmazione.
Sintassi assembler:

LABEL OP OPN1, OPN2, ... ;commento

Codice mnemonico dell'operazione

operandi

Etichetta opzionale che identifica lo statement

Copyright © 2009 – The McGraw-Hill Companies srl

Programmi e processo di esecuzione

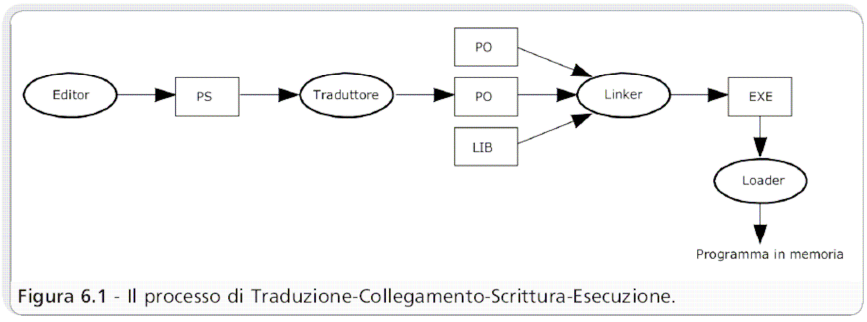

Figura 6.1 - Il processo di Traduzione-Collegamento-Scrittura-Esecuzione.

Legenda:

PS programma sorgente
PO programma oggetto
LIB sottoprogrammi di libreria
EXE modulo in forma eseguibile

Copyright © 2009 – The McGraw-Hill Companies srl

Istruzioni e architettura

Modelli di esecuzione delle istruzioni classificati rispetto all'esecuzione dello statement

a = b + c

La CPU esegue le elaborazioni che richiedono i seguenti passi:
1)Lettura della parola all'indirizzo B
2)Lettura della parola all'indirizzo C
3)Somma tramite la ALU
4)Scrittura del risultato nella parola all'indirizzo A

Come viene eseguita l'istruzione? Dipende dall'architettura

Modello memoria-memoria
Secondo questo modello è sufficiente una sola istruzione macchina
ADD A, B, C
La codifica dell'istruzione dovrebbe avere 3 campi per contenere i 3 indirizzi (richiederebbe un formato molto ampio)
Non sono teoricamente necessari registri nella CPU (in pratica servono 2 registri di appoggio)

Copyright © 2009 – The McGraw-Hill Companies srl

Istruzioni e architettura

Modello registro-registro
Per ognuno dei 4 passi, una specifica istruzione

LD R2, B
LD R3, C
ADD R1, R2, R3
ST A, R1
Operandi e risultato sempre dai/nei registri. Architetture RISC

Modello memoria-registro
Soluzione intermedia

LD R1, B
ADD R1, C
ST A, R1

Somma fra dato contenuto in un registro e dato in memoria.
Necessario un solo registro. Soluzione rigida. Codice compatto.

Copyright © 2009 – The McGraw-Hill Companies srl

Istruzioni e architettura

Modello a stack

Il programmatore non considera i registri in quanto le operazioni si effettuano usando lo stack (pila)

PUSH B
PUSH C
ADD
POP A

(PUSH prelievo, POP deposito)

Copyright © 2009 – The McGraw-Hill Companies srl

Istruzioni e architettura

Modelli di esecuzione delle istruzioni classificati rispetto all'esecuzione dello statement

a = b + c

Memoria-Memoria	Registro-Registro	Memoria-Registro	Stack
ADD A,B,C	LOAD R1,B LOAD R2,C ADD R3,R1,R2 STORE A,R3	LOAD B ADD C STORE A	PUSH B PUSH C ADD POP A

Figura 6.2 - Confronto di diversi modelli di esecuzione, per lo statement di alto livello a = b+c.

+ compattezza del codice
+ apparente efficienza
- Prestazioni scadenti, molte operazioni in memoria

- Prestazioni molto scadenti alto traffico con la memoria

Copyright © 2009 – The McGraw-Hill Companies srl

Sequenzializzazione delle istruzioni

Le istruzioni vengono eseguite in sequenza incrementando il PC (Program Counter).

Alcune istruzioni permettono il trasferimento del controllo non sequenziale:

- Istruzioni di salto (*jump*) condizionato o incondizionato
- Istruzioni di diramazione (*branch*) che a differenza delle prime prevedono il ritorno all'indirizzo successivo all'istruzione di diramazione (salvato sullo stack o in un apposito registro della CPU)

Inoltre il normale flusso sequenziale può essere modificato dalle *interruzioni*: eventi che alterano il normale flusso del programma, imponendo alla CPU di abbandonare l'esecuzione del programma corrente per passare a eseguire un altro programma dipendente dalla natura dell'interruzione stessa.

Copyright © 2009 – The McGraw-Hill Companies srl

Sequenzializzazione delle istruzioni

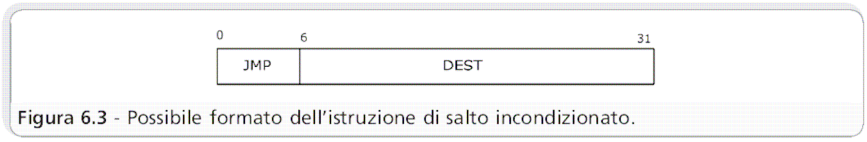

Figura 6.3 - Possibile formato dell'istruzione di salto incondizionato.

JMP DEST ; PC ← DEST

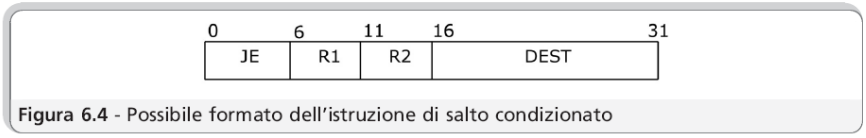

Figura 6.4 - Possibile formato dell'istruzione di salto condizionato

JE R1, R2, DEST ; if (R1==R2) PC ← DEST else PC ← PC + 4

Copyright © 2009 – The McGraw-Hill Companies srl

Sequenzializzazione delle istruzioni

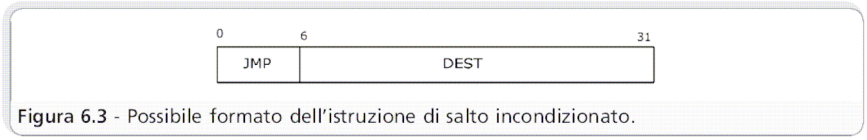

Figura 6.3 - Possibile formato dell'istruzione di salto incondizionato.

JMP DEST ; PC ← DEST

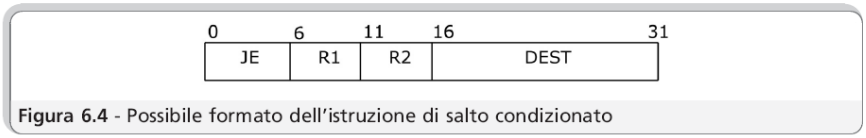

Figura 6.4 - Possibile formato dell'istruzione di salto condizionato

JE R1, R2, DEST ; if (R1==R2) PC ← DEST else PC ← PC + 4

Copyright © 2009 – The McGraw-Hill Companies srl

Il repertorio delle istruzioni

Repertorio stile RISC

- le istruzioni hanno tutte la stessa dimensione
- il campo del codice di operazione occupa uno spazio predefinito
- esiste un numero estremamente limitato di formati

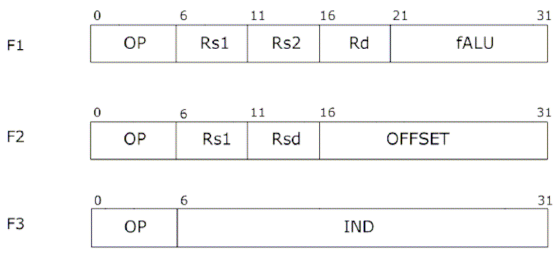

Figura 6.6 - Formati delle istruzioni. Il campo OP (bit 0-5) contiene il codice di operazione. Da esso dipende l'interpretazione degli altri campi. La regolarità del formato permette una facile decodifica dell'istruzione.

Copyright © 2009 – The McGraw-Hill Companies srl

Repertorio delle istruzioni di riferimento

Si assuma un'architettura con le seguenti caratteristiche:

- L'unità operativa presenta 32 registri di uso generale R0, R1, .. , R31
  R0 contiene il valore zero e non può essere sovrascritto
- La macchina è a 32 bit: registri, bus dati e bus indirizzi sono a 32 bit
- Gli indirizzi di memoria sono riferiti ai byte. Le istruzioni occupano sempre una parola di 32 bit.
- Per le istruzioni sono previsti solo tre formati:

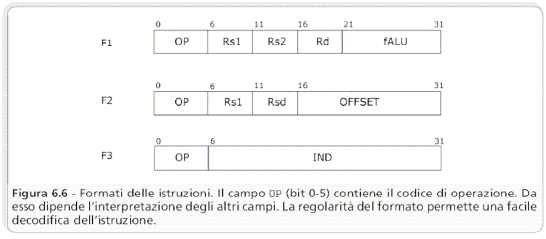

Figura 6.6 - Formati delle istruzioni. Il campo OP (bit 0-5) contiene il codice di operazione. Da esso dipende l'interpretazione degli altri campi. La regolarità del formato permette una facile decodifica dell'istruzione.

Copyright © 2009 – The McGraw-Hill Companies srl

Repertorio delle istruzioni di riferimento

- Istruzioni aritmetiche (formato F1) : prevedono sempre due registri sorgente e uno di destinazione.

  Il campo OP non specifica il tipo di operazione aritmetica (esso contiene il generico id. ARITM), discriminato invece attraverso il campo fALU.

ADD R1, R2, R8 R1 ← R3 + R8

Figura 6.6 - Formati delle istruzioni. Il campo OP (bit 0-5) contiene il codice di operazione. Da esso dipende l'interpretazione degli altri campi. La regolarità del formato permette una facile decodifica dell'istruzione.

Copyright © 2009 – The McGraw-Hill Companies srl

Repertorio delle istruzioni di riferimento

- Istruzioni che fanno riferimento alla memoria (formato F2) : prevedono sempre due registri, uno per il dato e l'altro per l'indirizzamento.

Il campo OFFSET, di 16 bit, indica lo scostamento rispetto al registro di indirizzamento e viene portato a 32 bit (con segno) mediante la ripetizione del bit più signiticativo.

LD R1, 100(R10) R1 ← M [100 + R10]

ST 200(R3), R7 M [200 + R3] ← R7

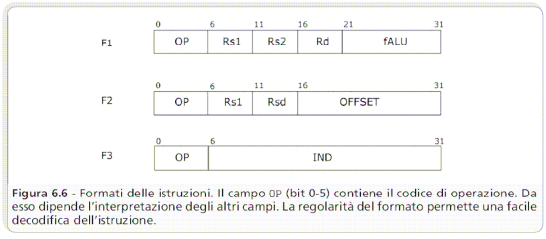

Figura 6.6 - Formati delle istruzioni. Il campo OP (bit 0-5) contiene il codice di operazione. Da esso dipende l'interpretazione degli altri campi. La regolarità del formato permette una facile decodifica dell'istruzione.

Copyright © 2009 – The McGraw-Hill Companies srl

Repertorio delle istruzioni di riferimento

- Istruzioni di salto incondizionato (formato F3) : prevedono sempre il campo IND di indirizzamento assoluto.

Jump (JMP) salta incondizionatamente alla destinazione:

JMP Label PC ← indirizzo Label

Jump and Link (JAL) salta incondizionatamente alla destinazione e salva il contenuto del PC nel registro R31:

JAL Label PC ← indirizzo Label; R31 ← PC + 4

Figura 6.6 - Formati delle istruzioni. Il campo OP (bit 0-5) contiene il codice di operazione. Da esso dipende l'interpretazione degli altri campi. La regolarità del formato permette una facile decodifica dell'istruzione.

Copyright © 2009 – The McGraw-Hill Companies srl

Repertorio delle istruzioni di riferimento

- Istruzioni di salto condizionato (formato F2) : prevedono sempre due registri per valutare la condizione e un OFFSET rispetto al PC.

  Jump if equal (JE) controlla se i due registri contengono lo stesso valore:

  JE R1, R2, Offset if (R1==R2) then PC ← PC + Offset

  Jump on sign (JS) controlla se il valore del secondo è maggiore di quello del primo registro:

  JS R1, R2, Offset if (R2>R1) then PC ← PC + Offset

Figura 6.6 - Formati delle istruzioni. Il campo OP (bit 0-5) contiene il codice di operazione. Da esso dipende l'interpretazione degli altri campi. La regolarità del formato permette una facile decodifica dell'istruzione.

Copyright © 2009 – The McGraw-Hill Companies srl

Repertorio delle istruzioni di riferimento

- Istruzioni di salto relativo (formato F2) : hanno un formato degenere perché prevedono solo un registro e il campo OP.

  Jump Register (JR) salta all'indirizzo indicato nel registro:

  JR R3 PC ← R3

- Istruzioni di non operazione (NOP): hanno l'unico effetto di incrementare il PC.

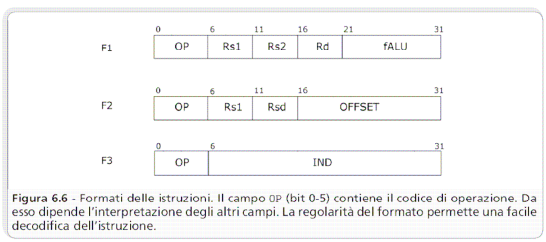

Figura 6.6 - Formati delle istruzioni. Il campo OP (bit 0-5) contiene il codice di operazione. Da esso dipende l'interpretazione degli altri campi. La regolarità del formato permette una facile decodifica dell'istruzione.

Copyright © 2009 – The McGraw-Hill Companies srl

Esempio: frammento di programma C

```
int s, i;
int v[10];

....

i=0;
s=0;
while (i<10) { s=s + v[i]; i=i+1; }
```


Copyright © 2009 – The McGraw-Hill Companies srl

Esempio: frammento di programma C

int s, i;
int v[10];
....
i=0;
s=0;
while (i<10) { s=s + v[i]; i=i+1; }

i = 0;
s = 0;

Il numero 0 servirà molte volte, quindi supponiamo di inserire nel registro R0 il valore 0 e di lasciarlo immutato.
Indichiamo con I, S, V gli indirizzi rispettivamente delle variabili i e s e Della prima parola che compone il vettore v.

Assembler:

```
SUB    R0, R0, R0      ; R0<- 0;          (formato F1)
ST     I(R0), R0       ; M[I+0] <- 0;     (formato F2)
ST     S(R0), R0       ; M[S+0] <- 0;     (formato F2)
```


Copyright © 2009 – The McGraw-Hill Companies srl

Esempio: frammento di programma C

```
int s, i;
int v[10];
....
i=0;
s=0;
while (i<10) { s=s + v[i]; i=i+1; }
```

s = s + v[i];

I registri R2, R3 e R4 della CPU servono per contenere rispettivamente l'indice i, la variabile s e il generico (i-esimo) elemento del vettore v.

Assembler:

```
LD    R2, I(R0)     ; R2<- M[I+0];    (formato F2)
LD    R3, S(R0)     ; R3<- M[S+0];    (formato F2)
LD    R4, V(R2)     ; R4<- M[V+R2];   (formato F2)
ADD   R3, R3, R4    ; R3<- R3+R4;     (formato F1)
ST    S(R0), R3     ; M[S+0] <- R3;   (formato F2)
```


Copyright © 2009 – The McGraw-Hill Companies srl

Esempio: frammento di programma C

```
int s, i;
int v[10];
....
i=0;
s=0;
while (i<10) { s=s + v[i]; i=i+1; }
```

i = i+ 1;

I registri R2, R3 e R4 della CPU servono per contenere rispettivamente l'indice i, la variabile s e il generico elemento i-esimo del vettore v.

Assembler:

i è un indice di un vettore quindi l'incremento deve essere di 4

```
LD    R2, I(R0)     ; R2<- M[I+0];    (formato F2)
ADDI  R2, R2, 4     ; R2<- R2+4;      (formato F2)
ST    I(R0), R2     ; M[I+0] <- R2;   (formato F2)
```

Add immediate: istruzione che somma al contenuto del registro R2 il *valore immediato* 4 e lo va a scrivere in R2 stesso. Il valore *immediato* 4 è contenuto nella parte OFFSET del formato F2.

Copyright © 2009 – The McGraw-Hill Companies srl

Esempio: frammento di programma C

```
int s, i;
int v[10];
....
i=0;
s=0;
while (i<10) { s=s + v[i]; i=i+1; }
```

while (i < 10)

I registri R2, R3 e R4 della CPU servono per contenere rispettivamente l'indice i, la variabile s e il generico elemento del vettore v. Il registro R5 è usato per contenere la costante 40 (JGE lavora sui registri)

Assembler:

```
While   LD      R2, I(R0)       ; R2<- M[I+0];          (formato F2)
        ADDI    R5, R0, 40      ; R5<- 40;              (formato F2)
        JGE     R2, R5, fine    ; if(R2>=40) PC<- fine; (formato F2)
```

JUMP IF GREATER OR EQUAL

Copyright © 2009 – The McGraw-Hill Companies srl

Esempio: frammento di programma C


```
int s, i;
int v[10];
....
i=0;
s=0;
while (i<10) { s=s + v[i]; i=i+1; }
```


```
        SUB     R0, R0, R0      ; R0<- 0;               (formato F1)
        ST      I(R0), R0       ; M[I+0] <- 0;          (formato F2)
        ST      S(R0), R0       ; M[S+0] <- 0;          (formato F2)

; while (i<10)
While   LD      R2, I(R0)       ; R2<- M[I+0];          (formato F2)
        ADDI    R5, R0, 40      ; R5<- 40;              (formato F2)
        JGE     R2, R5, fine    ; if(R2>=40) PC<- fine; (formato F2)
;s=s+v[i]
        LD      R3, S(R0)       ; R3<- M[S+0];          (formato F2)
        LD      R4, V(R2)       ; R2<- M[V+R2];         (formato F2)
        ADD     R3, R3, R4      ; R3<- R3+R4;           (formato F1)
        ST      S(R0), R3       ; M[S+0] <- R3;         (formato F2)
;i=i+1
        ADDI    R2, R2, 4       ; R2<- R2+4;            (formato F2)
        ST      I(R0), R2       ; M[I+0] <- R2;         (formato F2)
        JMP     While           ; PC <- while           (formato F3)
fine
```


Copyright © 2009 – The McGraw-Hill Companies srl

Capitolo 6
Il repertorio delle istruzioni

parte II

Copyright © 2009 – The McGraw-Hill Companies srl

Il repertorio delle istruzioni

Repertorio stile RISC (*Reduced Instruction Set Computers*)

- le istruzioni hanno tutte la stessa dimensione
- il campo del codice di operazione occupa uno spazio predefinito
- esiste un numero estremamente limitato di formati

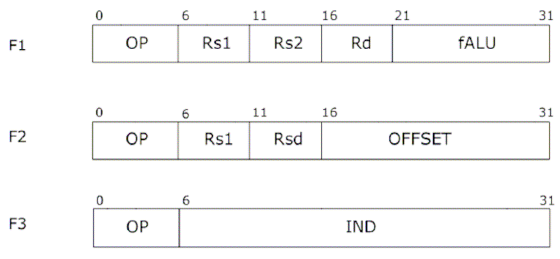

Figura 6.6 - Formati delle istruzioni. Il campo OP (bit 0-5) contiene il codice di operazione. Da esso dipende l'interpretazione degli altri campi. La regolarità del formato permette una facile decodifica dell'istruzione.

Copyright © 2009 – The McGraw-Hill Companies srl

Il repertorio delle istruzioni

Repertorio stile CISC (*Complex Instruction Set Computers*)

- le istruzioni non hanno dimensione fissa
- il campo del codice di operazione occupa un numero variabile di bit
- esiste un numero estremamente ampio di formati

PREFISSI				ISTRUZIONE				
Istruzione	Dimensione Indirizzo	Dimensione Operando	Segmento	Codice	MOD R/M	SIB	Scostamento	Immediato
0/1	0/1	0/1	0/1	1/2	0/1	0/1	0/1/2/4	0/1/2/4

Figura 6.7 - Formato delle istruzioni nell'architettura ×86, per macchine a 32 bit. Si noti che l'istruzione vera e propria può essere preceduta da ben quattro tipi di prefisso, che ne alterano l'interpretazione usuale. Il campo del codice di operazione può essere di 1 o 2 byte. Tre bit del campo MOD R/M sono da considerare come estensione del codice di operazione. MOD R/M determina quali registri sono implicati, oltre a indicare se c'è riferimento alla memoria e a condizionare la presenza del campo SIB (Scale-Index-Base). Il campo SIB determina il fattore di scala nell'accesso ai dati. Infine il campo scostamento e il campo immediato rappresentano rispettivamente lo scostamento dell'operando in memoria o una quantità codificata nell'istruzione.

Copyright © 2009 – The McGraw-Hill Companies srl

RISC vs. CISC

CISC:
+ compilatori più semplici , distanza fra istruzioni di alto livello e istruzioni macchina ridotta
+ uso più efficiente memoria centrale

- Memoria sempre più veloce, memoria cache
- 20% istruzioni del repertorio rappresentano l'80% di quelle eseguite

•Meglio investire su processori sempre più veloci e ottimizzare i compilatori
•Istruzioni semplici che richiedano pochi cicli di clock. Codice più grande ma di più facile interpretazione (memoria non costosa)
•Funzionalità a livello di Microcodice rende più difficile le modifiche. La memoria centrale ha velocità comparabile al controllo (meglio le libreria di sistema)
•Compilatori ottimizzati che svolgono il lavoro di risolvere le complicazioni del passaggio fra alto e basso livello

Copyright © 2009 – The McGraw-Hill Companies srl

Prestazioni della CPU

Il tempo di CPU richiesto per l'esecuzione del programma è

$$T_{CPU} = N/f$$

N numero di cicli di clock per l'esecuzione di un programma

f frequenza di clock

Il numero medio di clock per istruzioni macchina è

$$C_{PI} = N/N_{ist}$$

$$T_{CPU} = N/f = (N_{ist} \times C_{PI}) / f = N_{ist} \times C_{PI} \times T$$

dove T= 1/f rappresenta il periodo del clock

Copyright © 2009 – The McGraw-Hill Companies srl

Prestazioni della CPU

Legge di Amdahl

Definisce *accelerazione* (*speedup*) il rapporto fra le prestazioni di un programma dopo ($1/t_n$) e quelle prima ($1/t_v$) di un miglioramento:

$$a = t_v / t_n$$

Indice MIPS (milioni di istruzioni per secondo)

$$MIPS = N_{ist} / (T_e \times 10^6)$$

con T_e tempo di esecuzione del programma in secondi

Copyright © 2009 – The McGraw-Hill Companies srl

Prestazioni della CPU

Indice MFLOPS (milioni di istruzioni in virgola mobile per secondo)

$$MFLOPS = Nvm / (Te \times 10^6)$$

con Nvm numero di operazioni in virgola mobile del programma

Programmi campione

Programmi di *benchmark*, appositamente studiati e documentati per la quantificazione delle prestazioni in precisi campi applicativi e con differenti tipologie di carichi di lavoro.

Copyright © 2009 – The McGraw-Hill Companies srl

Modello di Memoria

Gli indirizzi di memoria vengono determinati come indirizzi assoluti (istruzioni JMP/JAL – campo IND) oppure come somma fra il campo OFFSET (numero con segno) e il contenuto di un registro (PC per JE/JS, Rb per LD/ST).

Effective Address (*EA*): il valore che risulta dal calcolo dell'indirizzo attraverso i componenti espliciti rappresentati nel codice di istruzione (fino ad ora *EA corrisponde al calcolo dell'indirizzo fisico)*

Per alcune architetture l'indirizzo fisico non corrisponde all'EA perchè l'indirizzo fisico viene calcolato sommando l'EA al contenuto di un registro non esplicitamente riferito nell'istruzione .

Nel modello di memoria segmentata questo registro è il registro di segmento

Memoria segmentata: suddivisa in segmenti di dimensione variabile i cui indirizzi di partenza sono contenuti a tempo di esecuzione in specifici registri di CPU

Copyright © 2009 – The McGraw-Hill Companies srl

Rilocazione della memoria

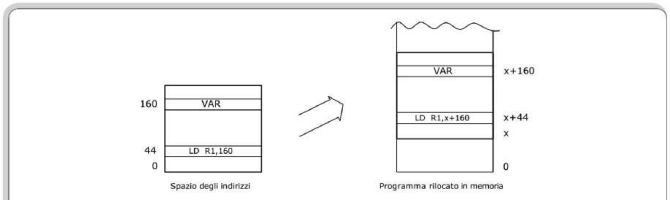

Figura 6.8 - Rilocazione di un programma in uno spazio di memoria lineare. I campi relativi agli indirizzi sono calcolati dall'assemblatore o dal compilatore in riferimento alla posizione convenzionale 0 nello spazio degli indirizzi. All'atto del caricamento in memoria, questi campi vengono modificati in modo da tener conto della posizione di caricamento del programma in memoria (indicata con x).

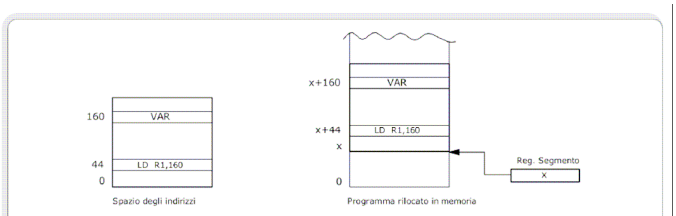

Figura 6.9 - Rilocazione di un programma in uno spazio di memoria segmentata. I campi relativi agli indirizzi, calcolati dall'assemblatore o dal compilatore in riferimento alla posizione convenzionale 0, non necessitano di essere modificati. Per tener conto della posizione di caricamento del programma viene usato un registro di segmento.

Copyright © 2009 – The McGraw-Hill Companies srl

Indirizzamento dei dati

Effective Address (EA): il valore che risulta dal calcolo dell'indirizzo attraverso i componenti espliciti rappresentati nel codice di istruzione.

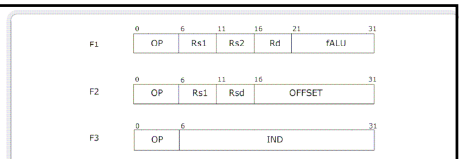

Figura 6.6 - Formati delle istruzioni. Il campo OP (bit 0-5) contiene il codice di operazione. Da esso dipende l'interpretazione degli altri campi. La regolarità del formato permette una facile decodifica dell'istruzione.

Indirizzamento diretto:

LD R1, Var EA = IND, (R1 ← M[EA]) (F2)

Indirizzamento relativo ai registri:

ST Var(R2), R5 EA = IND + R2 (F2)

Indirizzamento indiretto rispetto ai registri (senza uso di un campo indirizzo):

LD R1, (R2) EA = R2

Indirizzamento relativo ai registri, scalato e con indice:

LD R1, Var(R2), (R6) EA = IND + R2 + R6 * *d*

dove *d* è la dimensione dell'elemento (utile per strutture dati tipo vettori, matrici, ..) il secondo registro rappresenta l'indice nella struttura)

Copyright © 2009 – The McGraw-Hill Companies srl

Indirizzamento dei dati

Indirizzamento indiretto rispetto ai registri con autoincremento:

LD R1, (R2) + EA = R2, (R2 ← R2 + d)

Indirizzamento immediato (non c'è un vero e proprio indirizzamento):

LD R1, 2467 R1 ← 2467

Indirizzamento dei registri: (non c'è un vero e proprio indirizzamento)

LD R1, R2 R1 ← R2

Copyright © 2009 – The McGraw-Hill Companies srl

Macchine con stack

Meccanismo '*naturale*' per la chiamata/Ritorno e il passaggio dei prametri

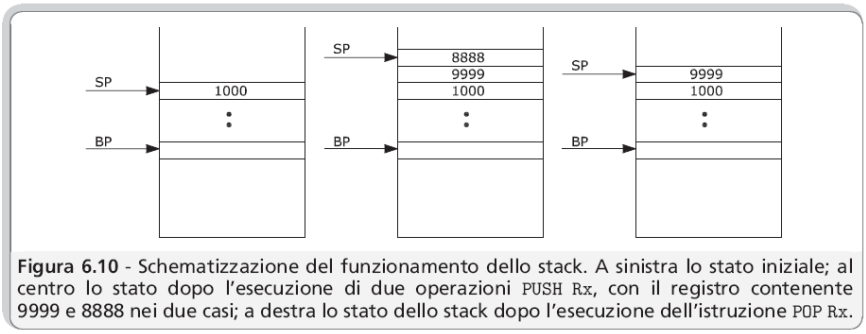

Figura 6.10 - Schematizzazione del funzionamento dello stack. A sinistra lo stato iniziale; al centro lo stato dopo l'esecuzione di due operazioni `PUSH Rx`, con il registro contenente 9999 e 8888 nei due casi; a destra lo stato dello stack dopo l'esecuzione dell'istruzione `POP Rx`.

BS *Base pointer* SP *Stack Pointer*

PUSH Rx SP ← SP + 4, M[SP] ← Rx

POP Rx Rx ← M[SP], SP ← SP - 4

Copyright © 2009 – The McGraw-Hill Companies srl

Chiamata a sottoprogrammi

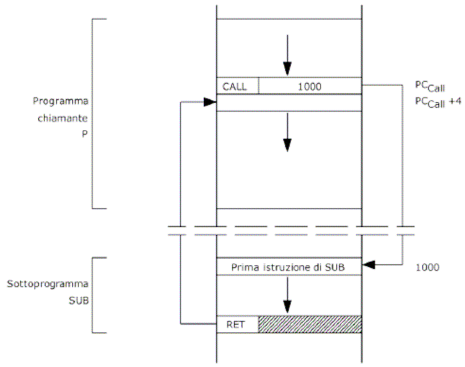

Figura 6.11 - Schematizzazione della chiamata e del ritorno a un sottoprogramma. L'istruzione `CALL` si trova all'indirizzo PC_{Call} nel programma chiamante P. L'istruzione contiene nel campo a ciò preposto il numero 1000, indirizzo della prima istruzione del sottoprogramma SUB. L'esecuzione dell'istruzione di chiamata deposita $PC_{Call}+4$ nello stack (non mostrato in figura). Il sottoprogramma termina con l'istruzione `RET` che preleva la parola in testa allo stack, riportando il contatore di programma a $PC_{Call}+4$. Naturalmente, affinché `RET` possa avere questo effetto, lo stack deve essere nella stessa condizione in cui è stato lasciato dall'istruzione `CALL`.

Copyright © 2009 – The McGraw-Hill Companies srl

Chiamata a sottoprogrammi

y = f (p1, p2)

CHIAMATA

a) Due PUSH per depositare p1 e p2 sullo stack e una CALL che fa saltare a f e salva l'indirizzo di ritorno sullo stack
b) Salvataggio di BP (come BP0) sullo stack e aggiornamento di BP con SP
c) Allocazione dello spazio per le due variabili locali

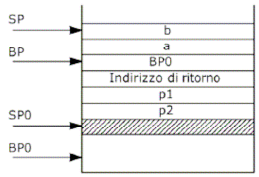

Figura 6.12 - Stato dello stack dopo che sono stati effettuati i tre passi a), b) e c) descritti nel testo. BP0 e SP0 rappresentano i valori in BP e SP prima dello statement `y=f(p1, p2)`.

Copyright © 2009 – The McGraw-Hill Companies srl

Chiamata a sottoprogrammi

y = f (p1, p2)

RITORNO

a) Due POP a perdere per eliminare le due variabili locali a e b;
b) POP per prelevare BP0 e assegnarlo a BP ripristinando il valore prima della chiamata
c) POP in un registro (es. R2) dell'indirizzo di ritorno, due POP a perdere per eliminare p1 e p2.
d) PUSH di R2 ed escuzione di RET
e) Assegnamento di R1 a y

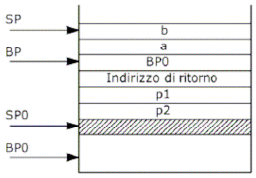

Figura 6.12 - Stato dello stack dopo che sono stati effettuati i tre passi a), b) e c) descritti nel testo. BP0 e SP0 rappresentano i valori in BP e SP prima dello statement `y=f(p1, p2)`.

Copyright © 2009 – The McGraw-Hill Companies srl

Interruzioni

Evento che, pur non essendo un salto o una chiamata/ritorno da un sottoprogramma, altera il normale flusso di esecuzione del programma.

Classificazione

- interruzioni esterne: generate dall'esterno del programma, asincrone e impredicibili
- eccezioni: causate da anomalie durante l'esecuzione del programma, sincrone e impredicibili
- trappole: generate da apposite istruzioni, sincrone e predicibili.

Copyright © 2009 – The McGraw-Hill Companies srl

Interruzioni

Trattamento

Bisogna stabilire un modo per passare dal programma corrente alla *routine di servizio* dell'interruzione

Nel passaggio deve essere salvato lo *stato di macchina*, in modo da ripristinarlo al termine della routine, come se niente fosse accaduto (*trasparenza* dell'interruzione)

Lo stato di macchina è il contenuto nei registri di CPU: l'azione del suo salvataggio non deve poter essere interrotta (*atomicità* dell'interruzione)

Se ad ogni interruzione è associato un selettore (*vettorizzazione: vettore di interruzione che porta a eseguire la specifica routine e non una routine generale per tutte le interruzioni)*, è possibile rendere più efficiente il passaggio alla routine di servizio

Copyright © 2009 – The McGraw-Hill Companies srl