Set Istruzioni della CPU

Classificazione del set istruzioni

Modi di indirizzamento

Classificazione del set Istruzioni

Organizzazione.

Il set istruzioni dei μP della famiglia '86 è diviso nelle seguenti classi:

- Trasferimento dati
- Aritmetiche
- Manipolazione di bit
- Manipolazione di stringhe
- Trasferimento del controllo
- Controllo processore

Istruzioni di assegnazione

Trasferimento generico

Le istruzioni di trasferimento, servono a copiare un byte o una word tra una locazione di memoria e un registro interno, oppure tra registri. La copia diretta tra locazioni di memoria è eseguita mediante le istruzioni per la gestione delle stringhe.

MOV destinazione, sorgente

L'operando sorgente viene copiato nell'operando destinazione. Si assume sempre che l'operando in memoria si trovi in un segmento dati accessibile tramite il valore corrente del registro DS.

Si supponga di voler assegnare a tre variabili intere il valore 1. Un modo per farlo è il seguente:

- var1 = 1;
- var2 = 1;
- var3 = 1;

Si ottengono tre istruzioni di trasferimento con modo di indirizzamento immediato, cioè il valore 1, di 16 bit perché intero, viene codificato nell'istruzione.

Per ottenere una traduzione ottimizzata, in C si usa l'assegnazione multipla:

var1 = var2 = var3 = 1;

Infatti in questo caso il compilatore prima trasferisce 1 in un registro, ad esempio AX, e poi tradurrà le assegnazioni con istruzioni mov var, ax, che occupano meno spazio delle precedenti.

Ottimizzazione dell'assegnazione

Assegnazione dello stesso valore iniziale a 3 variabili:

Assegnazione	Istruzione ASM	Indirizzo	Codice macchina
var1 = 1	MOV word ptr [0004], 0001	CS:0000	C7 06 04 00 01 00
var2 = 1	MOV word ptr [0006], 0001	CS:0006	C7 06 06 00 01 00
var3 = 1	MOV word ptr [0008], 0001	CS:000C	C7 06 08 00 01 00

Nota: tutti i valori sono espressi in esadecimale.

Ciascuna istruzione è lunga 6 byte. Tutte occupano 24 byte.

C706 è il codice operativo dell'istruzione MOV con indirizzamento diretto e specificatore del tipo (word pointer) dell'operando destinazione.

0004, 0006, 0008 sono gli indirizzi delle 3 variabili (scritti in notazione little-endian, cioè prima il byte di ordine inferiore: 04, 06, 08 e poi il byte di ordine superiore: 00) e si trovano nel segmento Dati.

0001 è il valore dell'operando sorgente, anch'esso scritto in notazione little-endian.

La prima istruzione è memorizzata a partire dall'indirizzo 0000, nel segmento Codice, la seconda inizia dall'indirizzo 0006, la terza inizia dal'indirizzo 000C.

Ottimizzazione dell'assegnazione

Assegnazione multipla:

var1 = var2 = var3 = 1

Assegnazione	Istruzione ASM	Indirizzo	Codice macchina
MOV AX, 1	MOV AX, 0001	CS:0000	B8 01 00
MOV var1, AX	MOV [0004], AX	CS:0003	A3 04 00
MOV var2, AX	MOV [0006], AX	CS:0006	A3 06 00
MOV var3, AX	MOV [0008], AX	CS:0009	A3 08 00

Ciascuna istruzione è lunga 3 byte. In totale occupano 12 byte.

Il codice operativo A3 indica che l'istruzione specifica l'operando destinazione in memoria e l'operando sorgente in AX.

Lo specificatore di tipo (word ptr), questa volta, non è necessario perché la presenza di AX indica che un operando è di 16 bit, di conseguenza lo deve essere anche l'altro.

Lo stack

- Lo stack è un'area di memoria nel segmento dello stack, la cui dimensione viene fissata dal programmatore.
- I dati memorizzati nello stack vengono elaborati nell'ordine: ultimo inserito primo estratto (Last In First Out - LIFO).
- L'ultimo elemento inserito, e quindi il prossimo da prelevare, si trova sulla *testa* dello stack.
- Sullo stack si possono depositare solo operandi di 16 bit (2 byte), o di tipo Word.
- Il registro SP (Stack Pointer) contiene l'indirizzo della testa dello stack.
- Quando si inseriscono dati sullo stack, la *testa* dello stack si sposta verso indirizzi bassi.

Operazioni con lo stack

PUSH sorgente

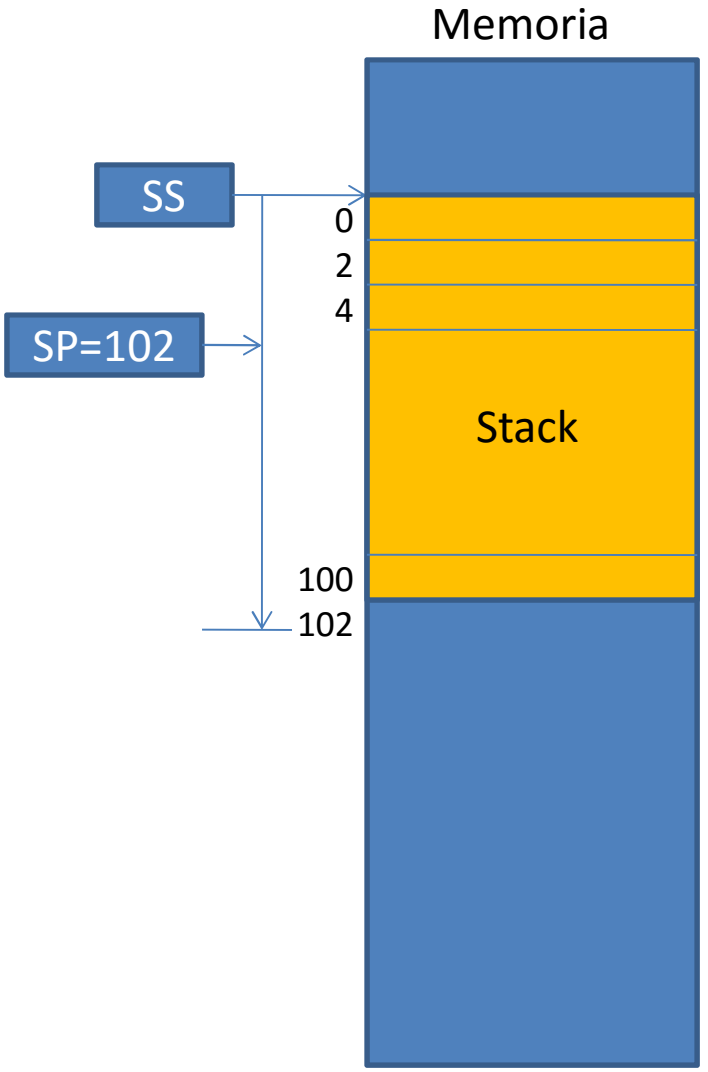
deposita l'operando, di tipo word, sullo stack. Il registro SP (Stack Pointer) è decrementato di 2 e l'operando viene trasferito nella locazione indirizzata da SP.

POP destinazione

Trasferisce la word di testa dello stack (puntata da SP) nell'operando destinazione. Il registro SP è incrementato di 2.

Lo stack: Esempio

- Il programmatore riserva uno spazio di memoria nel segmento SS e assegna al registro SP l'indirizzo della prossima locazione da cui prelevare il dato.
- Se la dimensione dello stack è 100 byte, inizialmente, il registro SP punta alla locazione 102.
- Il valore SP=102 indica che lo stack è vuoto.
- Il valore SP=0 indica che lo stack è pieno



PUSH

Esempio:

PUSH AX

PUSH BX

Il programmatore deve prima assicurarsi che lo stack non sia pieno (SP≠0).

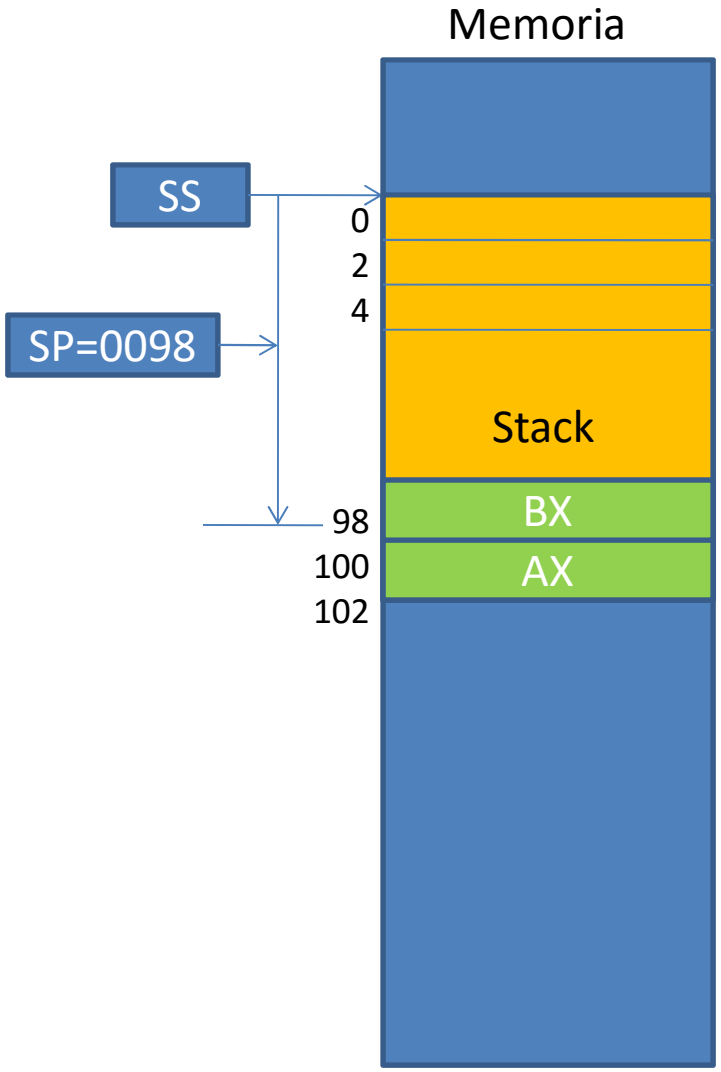
La CPU esegue un'istruzione PUSH in due passi:

PUSH AX

- Decrementa SP di 2, diventa SP=0100
- Trasferisce AX nella locazione puntata da SP.

PUSH BX

- Decrementa SP di 2, diventa SP=0098
- Trasferisce BX nella locazione puntata da SP.



POP

Esempio:

POP BX

POP AX

Il programmatore deve prima assicurarsi che lo stack non sia vuoto (SP≠0102).

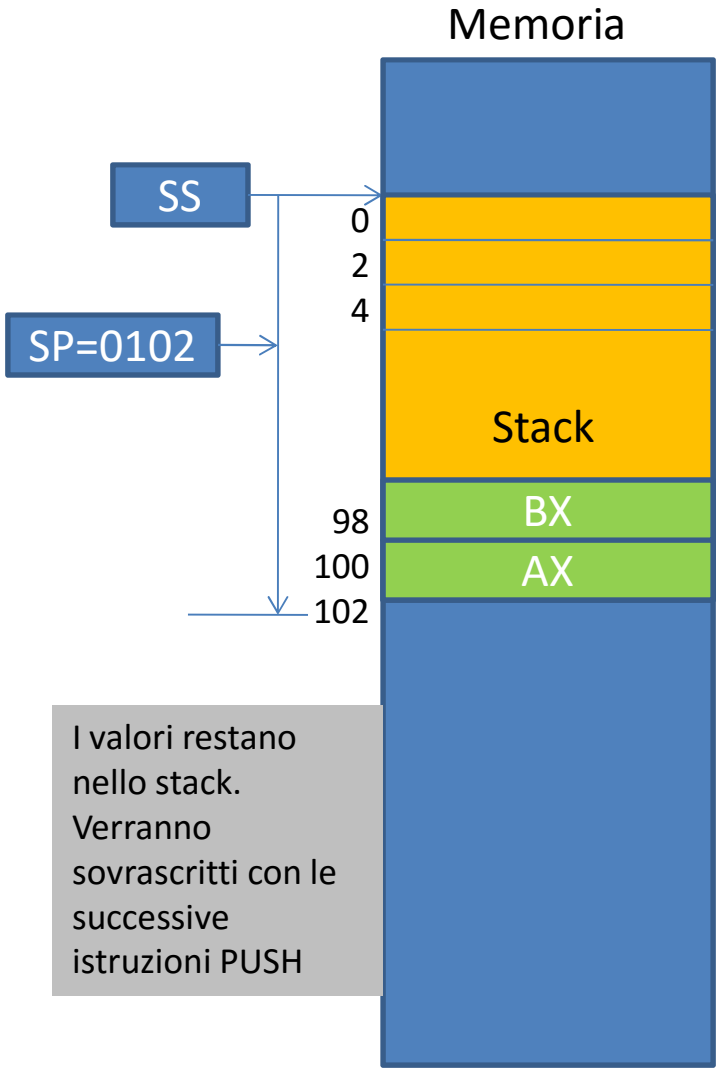
La CPU esegue un'istruzione POP in due passi:

POP BX

- Trasferisce in BX il valore contenuto nella locazione puntata da SP
- Incrementa SP di 2, diventa SP=0100.

POP AX

- Trasferisce in AX il valore contenuto nella locazione puntata da SP
- Incrementa SP di 2, diventa SP=0102.



Istruzioni di IN e di OUT

Le istruzioni di trasferimento di dati con una periferica operano solo con il registro AX o AL, a seconda se la periferica comunica dati di 16 bit o di 8 bit, non può essere usato nessun altro registro per trasferire dati con la periferica. Generalmente la periferica è collegata al bus della CPU tramite un'interfaccia, il cui registro di scambio è detto 'port'.

IN AL, port

trasferimento, di un byte o di una word, da un port di ingresso al registro AL o AX. Il port può essere indirizzato in modo immediato (max 256 port) oppure in modo indiretto tramite il registro DX (max 65536 port).

OUT port, AL

trasferimento di un byte o di una word da AL o AX al port di uscita. Il possibile indirizzamento del port è identico a quello dell'istruzione IN.

Trasferimento di indirizzi

Queste istruzioni trasferiscono gli indirizzi delle variabili piuttosto che i valori delle variabili: trattano puntatori e sono utili per le elaborazioni di liste, variabili strutturate e operazioni su stringhe di byte.

LEA destinazione, sorgente

(load effettive address) trasferisce la parte offset del puntatore all'operando sorgente (anziché il suo valore) nell'operando destinazione. L'operando sorgente deve essere un operando in memoria, e l'operando destinazione deve essere un registro di 16 bit.

Esempio: LEA SI, array. L'offset dalla base del segmento di appartenenza della variabile array è copiato nel registro SI.

LDS destinazione, sorgente

(load pointer using DS) la parte offset del puntatore (a 32 bit) dell'operando sorgente viene trasferita nell'operando destinazione mentre la parte registro segmento viene copiata nel registro DS.

Esempio: LDS DI, array. Lo spiazzamento, dalla base del segmento di appartenenza, della variabile array è copiato nel registro DI, mentre l'indirizzo base del segmento è copiato in DS.

LES destinazione, sorgente

(load pointer using ES) la parte offset del puntatore (a 32 bit) dell'operando sorgente viene trasferita nell'operando destinazione mentre la parte registro segmento viene copiata nel registro ES.

Operazioni logiche

Dopo ogni istruzione logica le flag di carry e di overflow sono sempre zero, mentre le flag di segno e di zero riflettono l'esito dell'operazione, e possono essere interrogate con le istruzioni di salto condizionato.

NOT destinazione

i singoli bit dell'operando vengono complementati.

AND destinazione, sorgente

viene eseguita un'operazione di And tra i singoli bit degli operandi, il risultato viene messo nell'operando destinazione. L'and è usata anche per forzare a 0 i bit dell'operando destinazione che sono a livello 0 nell'operando sorgente.

AND AL, 03: se AL contiene 0100 0101 (=45 esa), il risultato dell'and con 0000 0011 (03 esa) è 0000 0001.

AND SI, SI: corrisponde a verificare se SI è uguale a zero. In alternativa si dovrebbe usare l'istruzione CMP SI, 0.

OR destinazione, sorgente

viene eseguita un'operazione di Or tra i singoli bit degli operandi, il risultato viene messo nell'operando destinazione. L'or è usata anche per forzare a 1 i bit dell'operando destinazione che sono a 1 nell'operando sorgente.

OR AL, 03: se AL contiene 0100 0101 (=45 esa), il risultato dell'or con 0000 0011 (03 esa) è 0100 0111 (=47 esa),

XOR destinazione, sorgente

viene eseguita un'operazione di Or esclusivo tra i singoli bit degli operandi, il risultato viene messo nell'operando destinazione.

XOR AX, AX: l'or esclusivo tra due valori uguali dà sempre risultato 0, quindi l'operazione XOR è spesso usata per azzerare un operando quando interessa posizionare anche le flag. L'istruzione MOV AX, 0 ha lo stesso effetto (scrive uno 0 in AX), ma non modifica le flag e ha una codifica più lunga.

Applicazioni dell'operatore AND

- Trasformare un carattere da minuscolo a maiuscolo:

  basta portare a 0 il bit 5 del codice ASCII del carattere e lasciare invariati gli altri bit.
- Es:

  Il registro AL contiene il codice ASCII del carattere 'a': $0110\ 0001_2$: 0x61

  Per forzare a 0 il bit 5 del registro AL si deve eseguire l'AND tra il valore contenuto in AL e il byte: $1101\ 1111_2$ = 0xDF

  In ASM:

  ```
  AND AL, 0DF
  ```

  In C:

  ```
  ch = ch & 0xDF
  ```

  (oppure, in forma compatta: ch &= 0xDF)

Codici ASCII (lettere maiuscole)

Binario	Dec	Hex	Car
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F

Codici ASCII (lettere minuscole)

Binario	Dec	Hex	Car
0110 0001	97	61	a
0110 0010	98	62	b
0110 0011	99	63	c
0110 0100	100	64	d
0110 0101	101	65	e
0110 0110	102	66	f

Applicazioni dell'operatore AND

- Un numero, acquisito da tastiera, viene rappresentato in codice ASCII in una variabile. Trasformare il codice ASCII nella rappresentazione in binario del numero.

  I codici ASCII dei numeri hanno i 4 bit di sinistra nella configurazione 0011, mentre gli altri 4 bit rappresentano la codifica in binario della cifra decimale.

  Es:

  il carattere acquisito da tastiera è disponibile in AL. Per azzerare i quattro bit di sinistra, si esegue l'AND tra l'operando e il valore 0F:

  in ASM.

  ```
  AND AL, 0F
  ```

  in C:

  ```
  ch = ch & 0x0F
  ```

  (oppure, in forma compatta: ch &= 0x0F)

Codici ASCII (numeri)

Binario	Dec	Hex	Car
0011 0000	48	30	0
0011 0001	49	31	1
0011 0010	50	32	2
0011 0011	51	33	3
0011 0100	52	34	4
0011 0101	53	35	5
0011 0110	54	36	6
0011 0111	55	37	7
0011 1000	56	38	8
0011 1001	57	39	9

Applicazioni dell'operatore AND

- Un indirizzo IPv4 è rappresentato con 32 bit, ed è composto da 2 parti:
  - il prefisso di rete
  - il numero di host.
- Ad esempio, per un indirizzo IPv4 il numero di bit che formano il prefisso di rete si specifica con la notazione:

  192.168.218.65/26

  Per estrarre il prefisso di rete, bisogna eseguire l'AND tra l'indirizzo IPv4 e un valore a 32 bit, avente i primi 26 bit a livello 1:

  192.168.218.65 AND
  255.255.255.192=
  192.168.218.64

  Il prefisso di rete è 192.168.218.64

Applicazioni dell'operatore OR

- Trasformare un carattere da maiuscolo a minuscolo:

  basta portare a 1 il bit 5 del codice ASCII del carattere e lasciare invariati gli altri bit.
- Es:

  Il registro AL contiene il codice ASCII del carattere 'A': $0100\ 0001_2$: 0x41

  Per forzare a 1 il bit 5 del registro AL si deve eseguire l'OR tra il valore contenuto in AL e il byte: $0010\ 0000_2$ = 0x20

  In ASM:

  ```
  AND AL, 020
  ```

  In C:

  ```
  ch = ch & 0x20
  ```

  (oppure, in forma compatta: ch &= 0x20)

Codici ASCII (lettere minuscole)

Binario	Dec	Hex	Car
0110 0001	97	61	a
0110 0010	98	62	b
0110 0011	99	63	c
0110 0100	100	64	d
0110 0101	101	65	e
0110 0110	102	66	f

Codici ASCII (lettere maiuscole)

Binario	Dec	Hex	Car
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F

Applicazioni dell'operatore OR

- Per mostrare un numero sullo schermo o sulla stampante, bisogna rappresentare le sue cifre in codice ASCII.

  Trasformare il codice binario di una cifra decimale nel suo codice ASCII.

  I codici ASCII dei numeri hanno i 4 bit di sinistra nella configurazione 0011, mentre gli altri 4 bit rappresentano la codifica in binario della cifra decimale.

  Es:

  il codice binario della cifra da stampare è disponibile in AL. Per portare i quattro bit di sinistra nella configurazione 0011, si esegue l'OR tra l'operando e il valore 30h:

  in ASM.

  ```
  OR AL, 30
  ```

  in C:

  ```
  ch = ch | 0x20
  ```

  (oppure, in forma compatta: ch |= 0x20)

Codici ASCII (numeri)

Binario	Dec	Hex	Car
0011 0000	48	30	0
0011 0001	49	31	1
0011 0010	50	32	2
0011 0011	51	33	3
0011 0100	52	34	4
0011 0101	53	35	5
0011 0110	54	36	6
0011 0111	55	37	7
0011 1000	56	38	8
0011 1001	57	39	9

Scorrimento

L'operazione di scorrimento consiste nello spostare tutti i bit di un operando. Lo scorrimento può essere logico o aritmetico. La flag di carry cattura il bit che esce fuori dall'operando, dopo lo scorrimento, le altre flag (di segno, zero, parità, e overflow) vengono posizionate secondo il risultato che resta nell'operando.

SHL / SAL destinazione, N-bit

(Shitf Left e Shift Arithmetic Left) gli scorrimenti a sinistra logico e aritmetico coincidono. L'operando <N-bit>, indica il numero di scorrimenti a sinistra da effettuare sull'operando destinazione. Per ogni scorrimento a sinistra entra un bit 0 a destra, il bit che esce dall'operando è copiato nella flag di carry. Questa istruzione trova utilità in quei casi nei quali interessa esaminare ciascun bit di un operando oppure per moltiplicare rapidamente un operando per 2 o un multiplo di 2.

Esempio: SHL AL, 1: se AL contiene 0100 0100 (=44 esa), dopo lo scorrimento in AL si ritrova 1000 1000 (=88 esa) e la flag di carry contiene 0.

SHR destinazione, N-bit

Scorrimento logico a destra dei bit dell'operando destinazione di <N-bit> posizioni, e introduzione a sinistra di un bit 0 per ogni scorrimento.

Esempio: SHR AL, 1: se AL contiene 1000 1000 (=88 esa), dopo lo scorrimento in AL si ritrova 0100 0100 (=44 esa) e la flag di carry contiene 0.

SAR destinazione, N-bit

scorrimento aritmetico a destra dei bit dell'operando destinazione di <N-bit> posti. Il bit di segno resta inalterato.

Rotazione

Nelle operazioni di rotazione, a differenza degli scorrimenti, i bit che escono dall'operando destinazione, non sono persi ma rientrano dall'altro estremo dell'operando. Nella rotazione può essere inserita anche la flag di carry, consentendo di isolare un bit dell'operando e interrogarlo tramite un'istruzione JC (jump on carry) o JNC (jump on not carry).

ROL destinazione, N-bit

rotazione a sinistra dei bit dell'operando destinazione, di <N-bit> posti.

Esempio: ROL AL, 1: se AL contiene 1000 1000 ((=88 esa), dopo la rotazione contiene 0001 0001 (= 11 esa), cioè il bit che esce a sinistra rientra all'altro estremo e tutti gli altri si spostano di un posto a sinistra.

ROR destinazione, N-bit

rotazione a destra dei bit dell'operando destinazione di <N-bit>. posizioni.

Esempio: ROR AL, 1: se AL contiene 0001 0001 ((=11 esa), dopo la rotazione contiene 1000 1000 (= 88 esa), cioè il bit che esce a destra rientra all'altro estremo e tutti gli altri si spostano di un posto a destra.

RCL destinazione, N-bit

(Rotazione, compreso la flag di Carry, a sinistra) rotazione a sinistra dei bit dell'operando destinazione di <N-bit> posti. Il bit di sinistra viene catturato nella flag di carry, mentre il bit della flag di carry entra a destra.

Esempio: RCL AL, 1: se AL contiene 0001 0001 ((=11 esa), e la flag carry è 1, dopo la rotazione AL contiene 0010 0011 (= 23 esa), e la flag carry contiene 0.

RCR destinazione, N-bit

rotazione a destra dei bit dell'operando destinazione di <N-bit> posti. Il bit di destra viene catturato nella flag di carry, mentre il bit della flag di carry entra a sinistra.

Esempio: RCR AL, 1: se AL contiene 0001 0001 ((=11 esa), e la flag carry è 1, dopo la rotazione AL contiene 1000 1000 (= 88 esa), e la flag carry contiene 1.

Salto incondizionato

CALL nome-procedura

Viene salvato sullo stack l'indirizzo dell'istruzione successiva a CALL e viene eseguita la procedura. Se la procedura chiamata è stata dichiarata di tipo NEAR, cioè è raggiungibile senza cambio di segmento, allora viene aggiornato solo il registro IP, se invece la procedura è di tipo FAR, cioè è contenuta in un segmento diverso da quello corrente, viene aggiornato CS e IP. Il salto alla procedura può essere diretto, se nell'istruzione si specifica il nome della procedura, o indiretto, se nell'istruzione si specifica il registro o la locazione di memoria che contiene l'indirizzo della procedura.

Esempio: CALL NEAR proc1 - il registro IP, che contiene l'offset entro il segmento codice della prossima istruzione da eseguire, è salvato sullo stack e l'offset di proc1 è scritto in IP.

RET

É l'istruzione che termina una procedura e provoca il ritorno al programma chiamante: viene prelevato l'indirizzo di ritorno dallo stack, questo è composto dal solo registro IP se la procedura è NEAR, è composto dalle due word relative a CS e a IP se la procedura è FAR.

JMP destinazione

salto incondizionato all'istruzione identificata da <destinazione>. A differenza del salto a una procedura, non avviene il ritorno. Il salto può essere diretto o indiretto. Il salto diretto è relativo, cioè è calcolato come numero di byte che separano il valore corrente di IP dalla locazione destinazione, e può essere SHORT, se la posizione destinazione si trova nel range da -128 a +127 locazioni, o NEAR, se la posizione destinazione si trova nel range di 32K in più o in meno. Il salto indiretto a una posizione interna al segmento avviene tramite un registro o una locazione di memoria, che contiene il nuovo IP.

Esempio: JMP SHORT -10 - il registro IP punta all'istruzione successiva, gli si deve sottrarre 10 per puntare a un'istruzione situata 10 byte prima. Generalmente il programmatore non deve occuparsi di compiere questi calcoli, peraltro in complemento a 2, ma specifica una label dalla quale continuare l'esecuzione (ad esempio JMP SHORT label1), l'assemblatore provvede a calcolare la distanza tra IP e label1.

Salto condizionato

Viene interrogata una combinazione di flag per verificare l'esistenza di determinate condizioni.

Dopo un'operazione aritmetica o logica su uno o tra due operandi, il programmatore può stabilire l'offset della prossima istruzione da eseguire interrogando le flag. Tutti i salti condizionati sono SHORT, ovvero la posizione destinazione si deve trovare in un range da -128 a +127 rispetto all'istruzione che segue JMP. Il salto viene effettuato se la condizione è vera, altrimenti viene eseguita l'istruzione successiva.

Esempi:

JC label

Jump if carry is set, salta all'istruzione preceduta da label se C=1.

JNC label

Jump if not carry, salta all'istruzione preceduta da label se C=0.

JZ label

Jump if zero flag is set, salta all'istruzione preceduta da label se Z=1.

Iterazione

Controllano la ripetizione dei cicli. Il registro CX contiene il numero di volte in cui il ciclo deve essere ripetuto. Il salto all'istruzione dalla quale ripetere l'esecuzione è relativo, cioè deve essere contenuto nel range -128÷ +127 posizioni misurate da quella successiva al ciclo.

LOOP short-label

il registro CX è decrementato di uno e se CX è diverso da zero il ciclo è ripetuto a partire dall'istruzione identificata dalla label, altrimenti si prosegue ad eseguire dall'istruzione successiva a LOOP. Esempio:

MOV CX, 10 CX è inizializzato con il valore 10, cioè il ciclo deve essere ripetuto 10 volte.

ciclo: INC BX prima operazione del loop, la label 'ciclo' identifica l'istruzione.

:

LOOP ciclo CX è decrementato di 1 e se CX è diverso da 0 si ripete l'esecuzione a partire da 'ciclo', altrimenti si prosegue con l'istruzione successiva.

Cicli condizionati

Due varianti dell'istruzione LOOP permettono di controllare il ciclo non solo sul numero contenuto in CX, ma anche su un'eventuale condizione che può modificare la flag di zero:

LOOPZ short-label

Per ripetere il ciclo deve risultare CX diverso da 0, e la flag di zero=1.

LOOPNZ short-label

Il ciclo è ripetuto se CX è diverso da 0 e la flag di zero=0.

JCXZ short-label

(jump if CX equal to zero) il salto è eseguito se CX è zero.

E' utile prima di un ciclo, per non eseguire il ciclo se CX=0.

Operazioni sulle stringhe

Stringa è una qualsiasi sequenza di byte o di word, che possono costituire una tabella di dati o anche un programma. Gli elementi della stringa possono essere di tipo byte o word. Sono disponibili le istruzioni di trasferimento, confronto e ricerca di elementi nelle stringhe.

Un'istruzione che fa riferimento a stringhe di dati può avere un operando sorgente, un operando destinazione o entrambi. L'hardware accede alla stringa sorgente tramite DS, e alla stringa destinazione tramite ES. L'assemblatore deduce il tipo, byte o word, dagli attributi degli operandi, ma non usa il nome di identificazione della stringa per l'indirizzamento: il contenuto del registro **SI** (source index) è usato come offset dell'elemento corrente nella stringa sorgente, e il contenuto del registro **DI** (destination index) è usato come offset dell'elemento corrente nella stringa destinazione. Questi due registri devono essere inizializzati (con le istruzioni di trasferimento di indirizzi) per puntare alle rispettive stringhe prima di eseguire l'istruzione che opera sulle stringhe.

Le istruzioni aggiornano, automaticamente, i registri SI e/o DI, in preparazione dell'elaborazione del successivo elemento. La flag di direzione (DF) determina se i registri indice devono essere autoincrementati (DF=0) o autodecrementati (DF=1).

Se viene codificato un prefisso di ripetizione, allora il registro CX (count register) è decrementato di uno dopo ogni ripetizione dell'istruzione. CX deve essere inizializzato con il numero di ripetizioni desiderate, cioè con la lunghezza della stringa.

Operazioni sulle stringhe

prefissi di ripetizione.

REP (Repeat) è usato in connessione con le istruzioni MOVS (move string), ed è interpretato come "repeat while not end of string" (CX $\neq$ 0).

REPE e **REPZ** ('repeat while equal' oppure 'repeat while zero') operano nello stesso identico modo e fisicamente coincidono. Questi prefissi sono usati con le istruzioni CMPS (compare string) e SCAS (scan string), e richiedono che la flag di zero (modificata da queste istruzioni) sia 1 prima di iniziare la successiva ripetizione.

REPNE e **REPNZ** ('repeat while not equal' oppure 'repeat while not zero') sono due forme mnemoniche diverse dello stesso byte di prefisso, in questo caso la flag di zero deve essere 0, altrimenti la ripetizione è terminata.

Trasferimento di stringhe

Trasferimento e ricerca in stringhe

MOVS stringa destinazione, stringa sorgente

Copia di un byte o di una word dalla stringa sorgente alla stringa destinazione e aggiornamento di SI e DI per puntare al successivo elemento. Quando viene usato il prefisso REP, l'istruzione MOVS esegue il trasferimento di dati da un blocco di locazioni di memoria ad un altro blocco di locazioni di memoria

```
MOV CX, 200      la stringa è lunga 200 byte
LDS SI, str2     indirizzo della stringa sorgente → DS:SI
LES DI, str1     indirizzo della stringa destinazione → ES:DI
CLD   la flag di direzione è posta a 0, in modo che il trasferimento proceda per indirizzi crescenti.
REP: MOVS str1, str2          trasferisci la stringa
```

MOVSB / MOVSW

Queste istruzioni sono codificate senza operandi. Esse dicono esplicitamente all'assemblatore se bisogna trasferire byte o word. Queste istruzioni sono utili quando l'assemblatore non può determinare il tipo degli elementi delle stringhe, come ad esempio quando si spostano blocchi di istruzioni.

Confronto di stringhe

CMPS stringa destinazione, stringa sorgente

Il byte, o la word, destinazione viene sottratto dal byte, o word, sorgente. CMPS non altera gli operandi, aggiorna SI e DI affinché puntino al successivo elemento della stringa e aggiorna le flag in accordo alla relazione tra gli operandi.

Esempio: i registri SI, DI e CX sono stati inizializzati con gli indirizzi e la lunghezza della stringa, e la flag di direzione é stata opportunamente posizionata, il seguente programma confronta in ciclo gli elementi delle stringhe, e ad ogni ciclo decide l'operazione da compiere.

- **ciclo: CMPS str1, str2** ; confronta una coppia di elementi delle stringhe
- **JG mag**; se l'elemento destinazione è maggiore dell'elemento sorgente continua dalla label 'mag'
- **min: .....**
- **mag: ....**
- **LOOP ciclo;** se la stringa non è finita, cioè se dopo il decremento CX>0, il ciclo viene ripetuto.

Se l'istruzione CMPS è preceduta dal prefisso REPE o REPZ, l'operazione è interpretata: "Fintantochè la stringa non è finita (CX≠0) e gli elementi sono uguali (ZF=1), continua il confronto."

REPE: CMPS str1, str2 Il confronto termina quando si incontrano due elementi diversi, i registri SI e DI ne indicano la posizione entro le stringhe.

Se l'istruzione CMPS è preceduta dal prefisso REPNE o REPNZ, l'operazione è interpretata: "Fintantochè la stringa non è finita (CX≠0) e gli elementi sono diversi (ZF=0), confronta i successivi". Con CMPS si possono individuare identità o differenze tra elementi di due stringhe.

Ricerca in una stringa

SCAS stringa destinazione

(scan string) Scansione della stringa alla ricerca di un elemento uguale a quello contenuto in AL o AX. L'elemento della stringa destinazione, indirizzato da DI, viene sottratto dal contenuto di AL o da AX e vengono aggiornate le flag ma non vengono modificati gli operandi.

SCAS aggiorna anche il puntatore DI. Se l'istruzione SCAS è preceduta dal prefisso REPE o REPZ l'operazione è interpretata: "continua la scansione fintantochè la stringa non è finita (CX≠0) e l'elemento della stringa è uguale ad AX (cioè ZF=1)". Questa forma può essere usata per individuare lo scostamento da un determinato valore.

Se l'istruzione SCAS è preceduta dal prefisso REPNE o REPNZ l'operazione è interpretata: "continua la scansione fintantoché la stringa non è finita (CX≠0) e l'elemento della stringa è diverso da AX (cioè ZF=0)". Questa forma può essere usata per localizzare un determinato valore in una stringa.

Controllo del processore

operazioni sulle flag.

- CLC (clear carry flag) Azzera la flag di carry senza influenzare altre flag.
- CMC (complement carry flag) Inverte il valore della flag di carry.
- STC (set carry flag) Pone a "1" il valore della flag di carry.
- CLD (clear direction flag) Pone a "0" la flag di direzione, così le istruzioni che operano sulle stringhe auto-incrementano i registri SI e DI.
- STD (set direction flag) Pone a "1" la flag di direzione, così le istruzioni che operano sulle stringhe auto-decrementano i registri indice SI e DI.
- CLI (clear interrupt enable flag) Pone a "0" il valore della flag di interruzione, il processore non rileverà richieste di interruzione esterne che potrebbero giungere sull'ingresso INTR.
- STI (set interrupt enable flag) Pone a "1" il valore della flag di interruzione, il processore prenderà in considerazione le richieste di interruzione esterne che potrebbero giungere sull'ingresso INTR dopo l'esecuzione dell'istruzione che segue STI

Gestione delle interruzioni

Queste istruzioni consentono l'esecuzione dei programmi di servizio delle periferiche, così come avviene se il servizio fosse richiesto dalla periferica mediante il segnale hardware di Interrupt Request. A differenza dell'interrupt hardware, in corrispondenza di un'istruzione INT il processore non esegue la sequenza di riconoscimento dell'interruzione, e il programma è richiamato indipendentemente dal valore della flag di interruzione (abilitate o disabilitate).

- INT tipo-interruzione

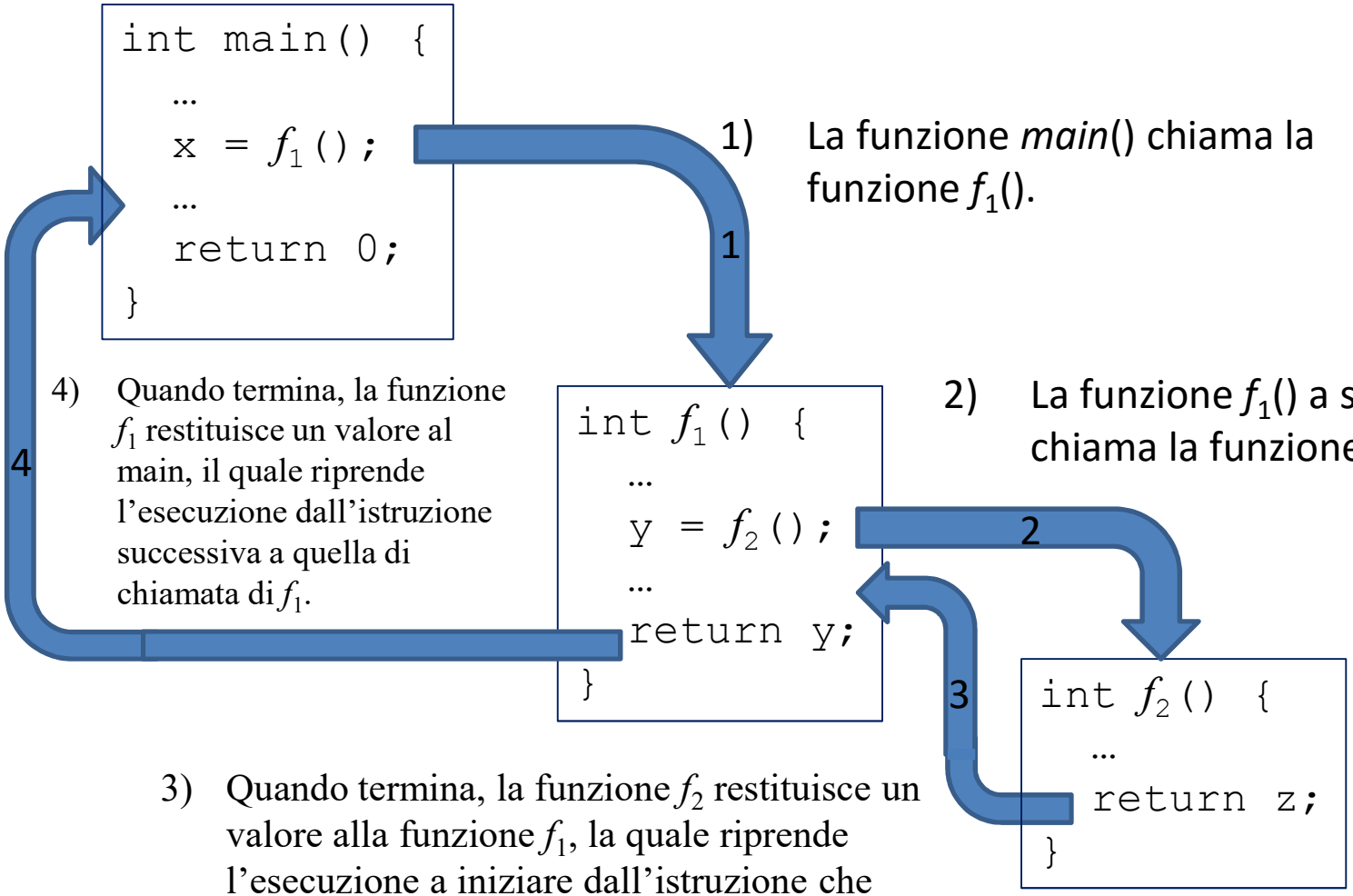
provoca l'esecuzione della procedura di servizio delle interruzioni specificata dall'operando 'tipo-interruzione': L'istruzione INT deposita sullo stack il registro delle flag, il registro CS e il registro IP, per poi recuperarli al termine della procedura di servizio. L'operando 'tipo-interruzione' moltiplicato per quattro fornisce l'indirizzo di memoria in cui è contenuto il puntatore (CS:IP) alla procedura di servizio da eseguire.

Esempio: INT 10h - Viene richiamato il programma di servizio il cui indirizzo è scritto nella locazione 40h

- IRET

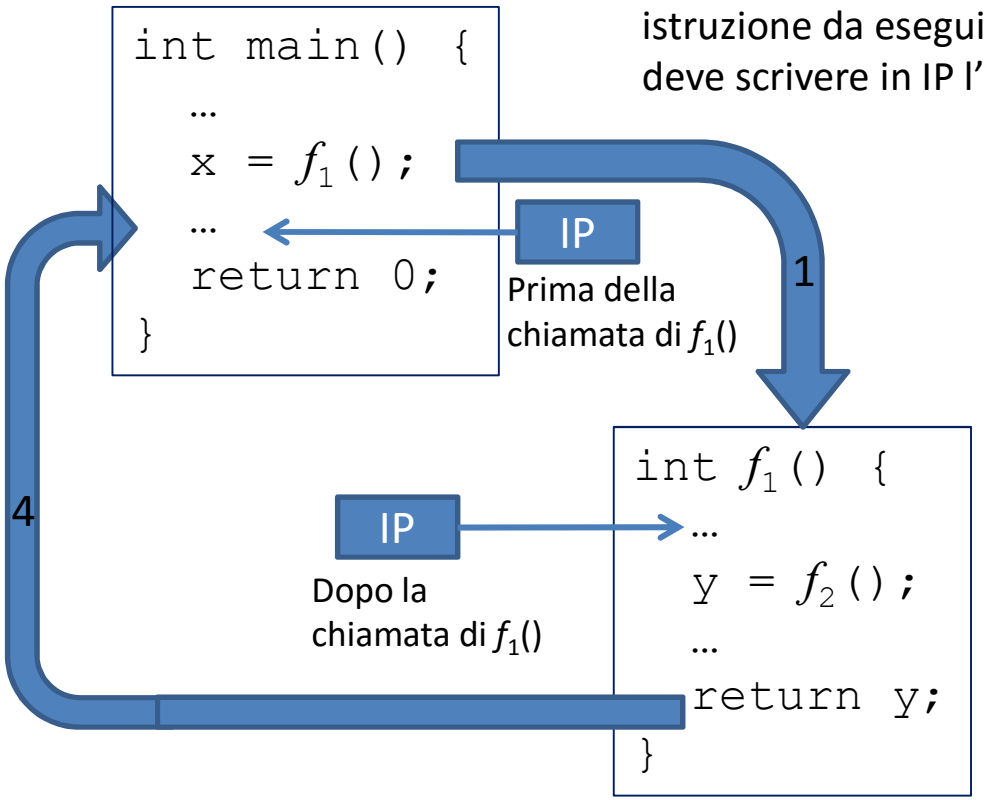
(Interrupt Return) Il controllo viene trasferito al programma interrotto, dallo stack vengono prelevati IP, CS e le flag (NOTA: in seguito all'interruzione la flag di interrupt enable viene resettata, è compito del programma di servizio decidere se riabilitare le interruzioni; in ogni caso l'istruzione IRET ripristina la flag al valore che essa aveva prima dell'interruzione). L'istruzione IRET è usata sia per il ritorno da un interrupt hardware che software.

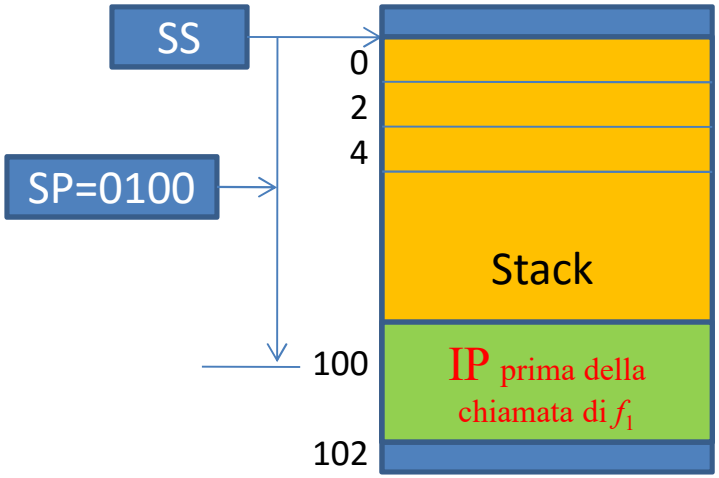
Chiamata di funzione



Lo stack durante la chiamata di funzione

La CPU, mentre esegue l'istruzione di chiamata di $f_1()$, contiene nel registro IP l'*offset* del puntatore alla successiva istruzione da eseguire in *main*(). Per eseguire il salto, la CPU deve scrivere in IP l'indirizzo di $f_1()$.

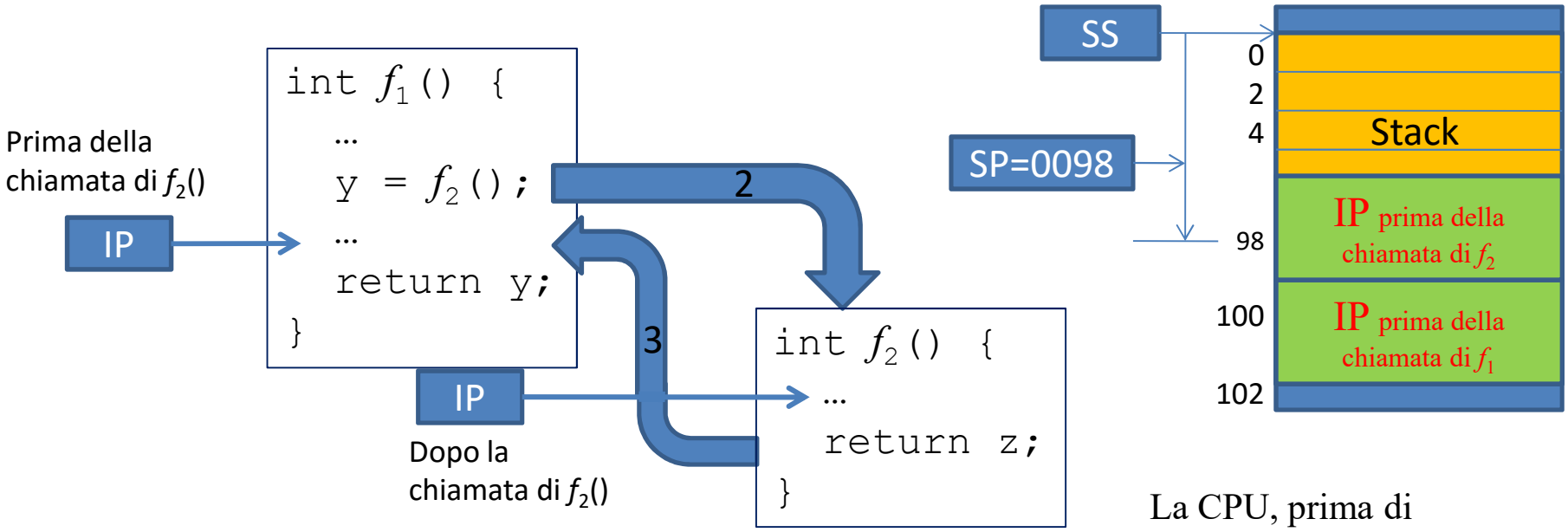




per non distruggere il valore contenuto in IP, che serve per riprendere a eseguire le istruzioni di *main*, la CPU deve salvare sullo stack il registro IP, da cui dovrà riprendere l'esecuzione alla fine di $f_1()$.

Lo stack durante la chiamata di funzione

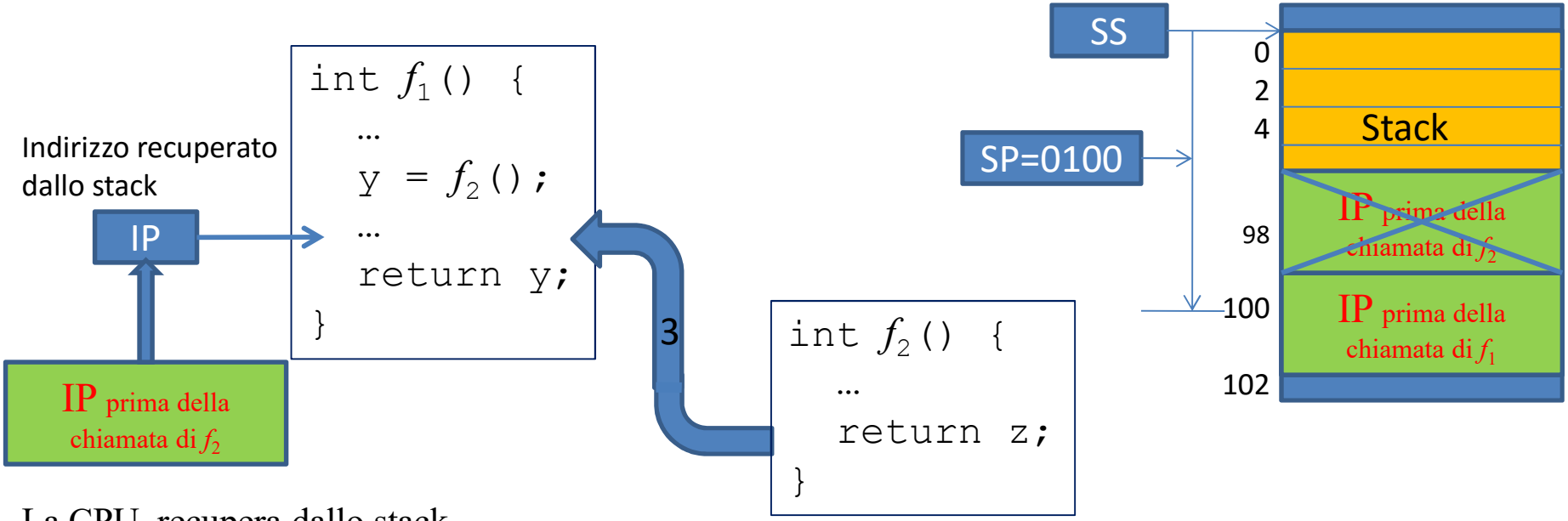
La CPU, mentre esegue l'istruzione di chiamata di $f_2()$, contiene nel registro IP l'*offset* del puntatore alla successiva istruzione da eseguire in $f_1()$. Per eseguire il salto, la CPU deve scrivere in IP l'indirizzo di $f_2()$.



La CPU, prima di aggiornare IP con l'indirizzo di f_2, salva sullo stack l'indirizzo a cui tornare al termine di f_2.

Lo stack durante il ritorno da funzione

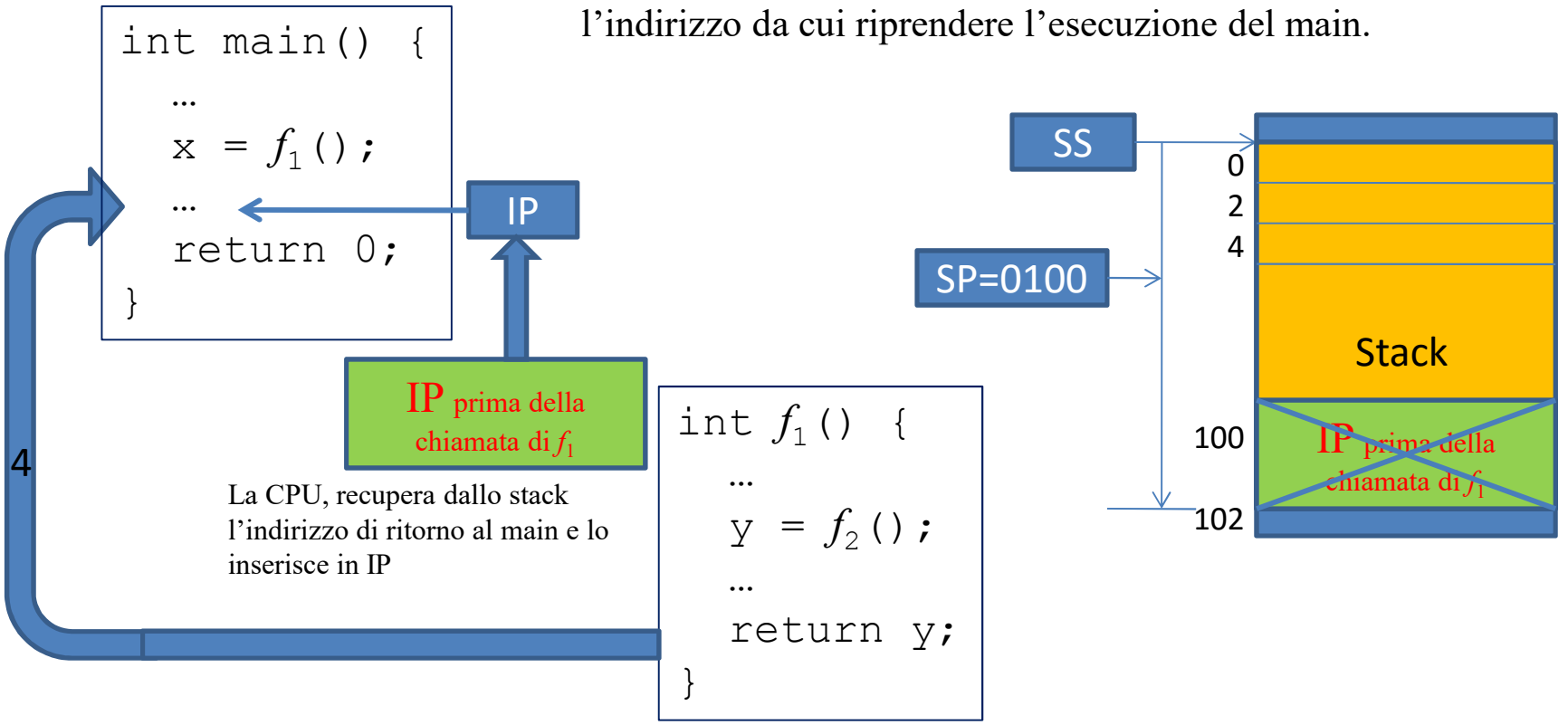
La CPU, al termine dell'esecuzione di $f_2()$, incontra l'istruzione return. Per tornare a $f_1()$ preleva dallo stack l'indirizzo da cui riprendere l'esecuzione di f_1.



La CPU, recupera dallo stack l'indirizzo di ritorno a f_1 e lo inserisce in IP

Lo stack durante il ritorno da funzione

La CPU, al termine dell'esecuzione di $f_1()$, incontra l'istruzione return. Per tornare al main preleva dallo stack l'indirizzo da cui riprendere l'esecuzione del main.



Passaggio dei Parametri

Si possono usare tre tecniche per passare i parametri ad una funzione:

- la funzione chiamante deposita i dati nei registri interni e la funzione chiamata li usa. Questa tecnica è usata dai servizi INT n del DOS;
- la funzione chiamante e la funzione chiamata usano una locazione di memoria comune. Questa tecnica è usata quando si riferiscono variabili globali.
- La funzione chiamante deposita i valori sullo stack poi, eseguendo l'istruzione di chiamata, la CPU deposita l'indirizzo di ritorno. La funzione chiamata tratta lo stack come un array per accedere ai valori presenti sullo stack.

Passaggio dei Parametri

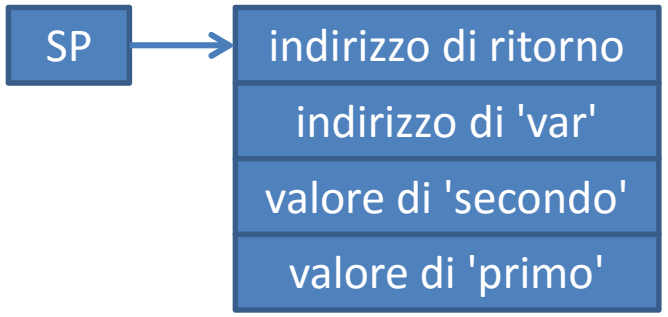
- Si consideri la dichiarazione di funzione:
  *int funzione(int var1, int var2, int *var3);*

Il compilatore traduce una chiamata di funzione del tipo:

risultato = funzione(primo, secondo, &var),

depositando dapprima i parametri sullo stack e poi inserendo un'istruzione di chiamata di funzione, con le seguenti istruzioni:

PUSH primo
PUSH secondo
PUSH offset var
CALL funzione



Lo stack dopo la chiamata di funzione

Passaggio dei Parametri

- Il compilatore traduce la chiamata di funzione, aggiungendo le istruzioni PUSH che depositano sullo stack, nell'ordine, il valore di *primo*, il valore di *secondo* e la parte offset del puntatore a *var*.
- L'istruzione CALL funzione richiede che il registro IP venga modificato con l'indirizzo di *funzione*, quindi la CPU, prima di modificare il valore di IP, salva sullo stack il valore corrente di IP, dal quale dovrà riprendere l'esecuzione.
- La funzione chiamata, per usare i parametri ricevuti, calcola la loro posizione sullo stack. Per non distruggere il contenuto di SP, la funzione chiamata usa il registro BP, previo il salvataggio di BP stesso:

  PUSH BP ; salva BP

  MOV BP, SP ; copia SP in BP
- Il registro SP punta a una word sullo stack contenente il valore di BP appena salvato, al di sotto di questa word c'è il registro IP a cui tornare al termine della funzione. L'elenco dei parametri passati inizia immediatamente dopo, dista cioè 6 byte dalla testa dello stack, quindi la corrispondenza tra i parametri formali e i loro offset è:

  indirizzo di var $\rightarrow$ [BP+6]

  valore di secondo $\rightarrow$ [BP+8]

  valore di primo $\rightarrow$ [BP+10]
- Al termine, la funzione chiamata deve fornire il risultato alla funzione chiamante. Siccome la regione dello stack utilizzata per i parametri non serve più, il valore ritornato dalla funzione viene depositato all'indirizzo BP+6. Eseguita questa operazione la funzione preleva dallo stack il valore di BP (POP BP), quindi esegue l'istruzione return, che corrisponde al prelievo dallo stack del registro IP.
- A questo punto il controllo è ritornato al programma chiamante. Il registro SP indirizza il valore del parametro restituito dalla funzione, il compilatore traduce l'assegnazione del risultato della funzione alla variabile *risultato*, prendendo il valore di testa dello stack e trasferendolo nella variabile *risultato*.
- I parametri sullo stack non servono più, per recuperare lo spazio che essi occupano resta da incrementare SP.
- Nell'esempio descritto si distinguono due casi:

  - passaggio per valore,
  - passaggio per riferimento.
- Il passaggio per valore consiste nel fornire alla funzione il valore da copiare nel parametro formale.
- Il passaggio per riferimento si usa quando la funzione deve restituire più di un parametro. Alla funzione si passa, nel parametro formale, l'indirizzo di memoria, anziché il valore, della variabile sulla quale operare.