Calcolatori Elettronici
Linguaggio macchina IA-32

Ing. Gestionale e delle Telecomunicazioni

A.A. 2008/09

Gabriele Cecchetti

Linguaggio Macchina IA-32

- Sommario:
  - Registri e memoria di IA-32
  - Modi di indirizzamento e istruzioni
  - Direttive e strutture del programma
  - Forma numerica
  - Esempi semplici di programma
  - Sottoprogramma
  - Altri esempi
  - Istruzioni privilegiate
- Riferimenti
  - C. Hamacher, “Introduzione all’architettura del Calcolatore”, cap. 7.

Banco di registri indirizzamento di memoria

REGISTRI E MEMORIA DI IA-32

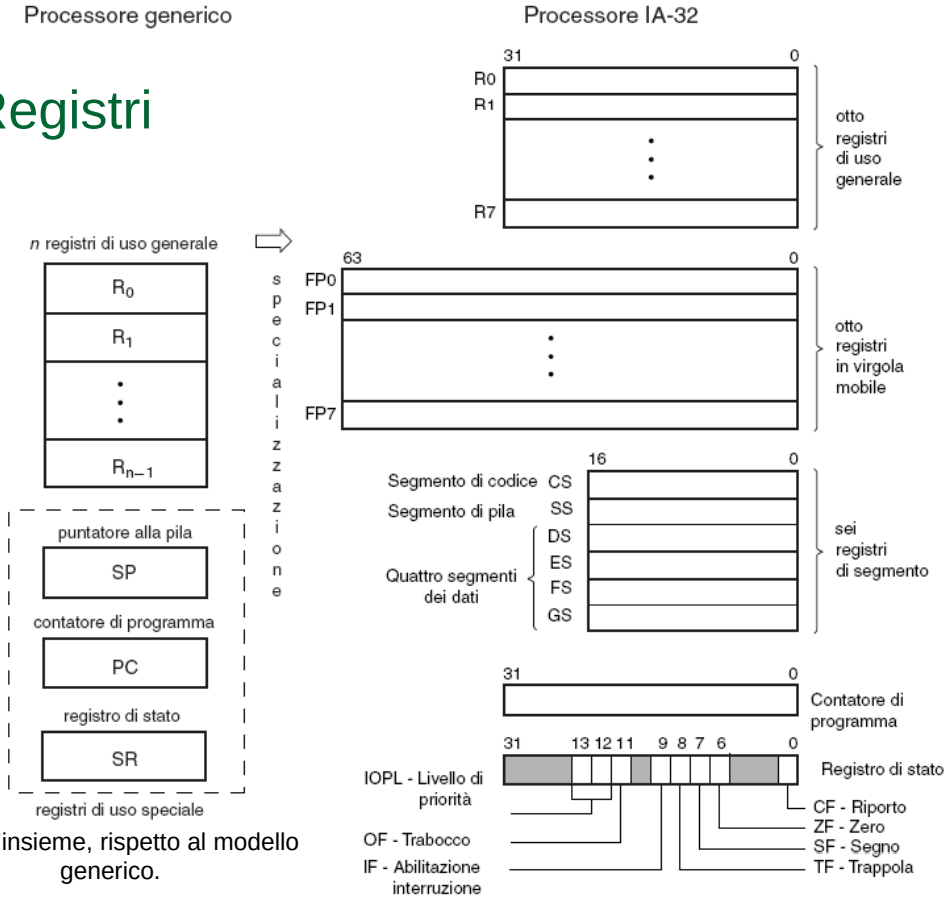
IA-32

- Famiglia di processori piuttosto complessa e articolata, a 32 bit, ma lavora anche con dati da 8 o 16 bit, compresi anche numeri in virgola mobile standard IEEE 754.
- Ha un repertorio di fino ad alcune centinaia di istruzioni macchina, contando tutte le estensioni, a due argomenti (e qualche istruzione è unaria o non ha argomenti).
- Ha svariati modi di indirizzamento e istruzioni per lo più semi-ortogonali, tranne poche eccezioni.
- Contiene un banco di registri relativamente strutturato, con registri a 32 bit e di uso più o meno specializzato.
- Ha diversi modi di esecuzione (uno utente e vari di sistema), con livelli di privilegio differenziati.
- Sono processori di una certa potenza, e ampiamente usati in ambiti diversi: da microcontrollore, a calcolatore da tavolo, server fino a supercalcolatore.

Struttura Generale

- Il processore IA-32 ha diversi banchi di registri.
- Registri di uso generale:
  - per dati in aritmetica intera
  - per dati in virgola mobile IEEE 754
- Registri di uso speciale:
  - contatore di programma
  - puntatore alla pila
  - registro di stato
  - registri di segmento
- Alcuni modelli hanno banchi aggiuntivi, per gruppi di istruzioni specializzate.

Registri



Vista d'insieme, rispetto al modello generico.

Registri

- Registri di uso generale:
  - da R0 a R7, a 8 / 16 / 32 bit, per dati e indirizzi
  - da FP0 a FP7, a 32 / 64 bit, per dati in virgola mobile, aritmetica IEEE 754 in singola o doppia precisione
- I registri Ri (i = 0, …, 7) sono usati:
  - per contenere / elaborare dati, di taglia non troppo grande (8, 16 o 32 bit)
  - come puntatori a strutture dati residenti in memoria
- I registri Ri figurano spesso come argomenti di istruzioni aritmetiche-logiche e di trasferimento.
- Hanno anche nomi specifici, come EAX, ecc, e alcuni di essi possono avere uso speciale.

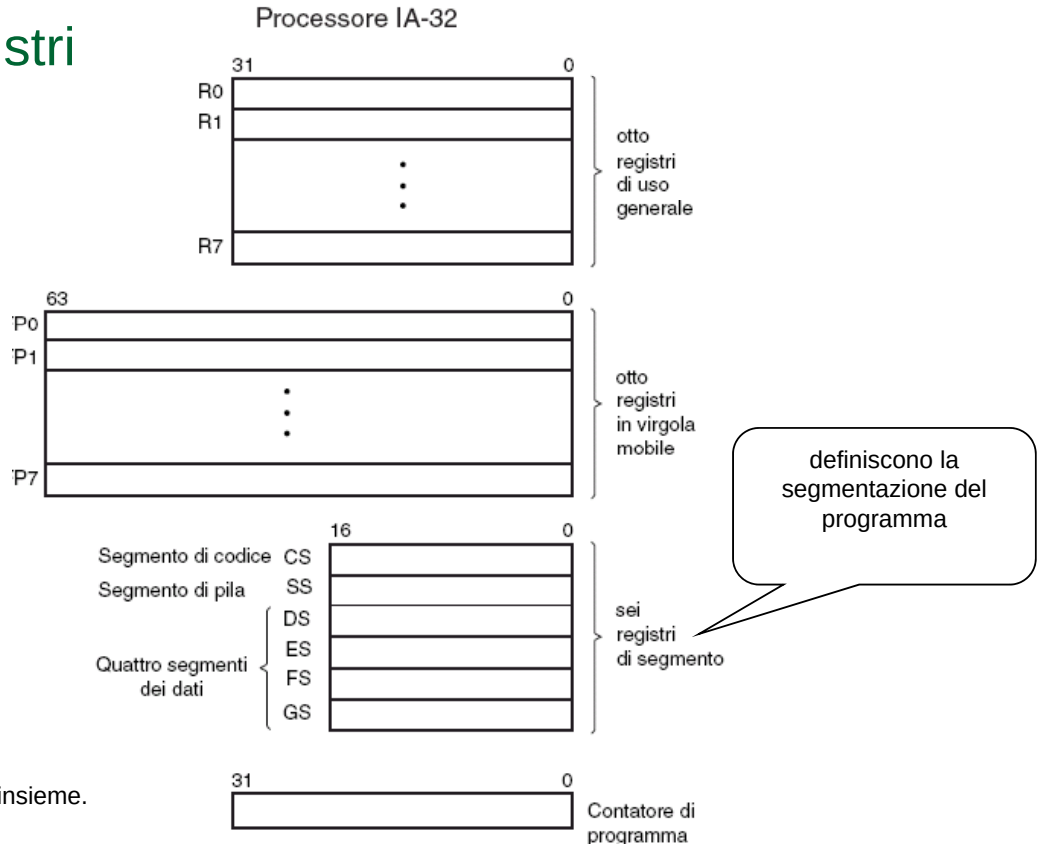
Registri

- Registri di uso speciale:
  - contatore di programma, EIP, a 32 bit
  - puntatore alla pila, ESP, a 32 bit, coincide con R4
    - il puntatore alla pila è unico, il meccanismo di doppia pila (utente e sistema) è gestito in SW
  - registro di stato, EFLAGS, a 32 bit, contenente
    - i quattro bit di esito ZF, SF, CF e OF
    - vari bit di controllo (modo IOPL, abil. interrupt, e altri)
- I registri speciali sono usati in modo implicito (senza nominarli) da svariate istruzioni.

Registri di Segmento

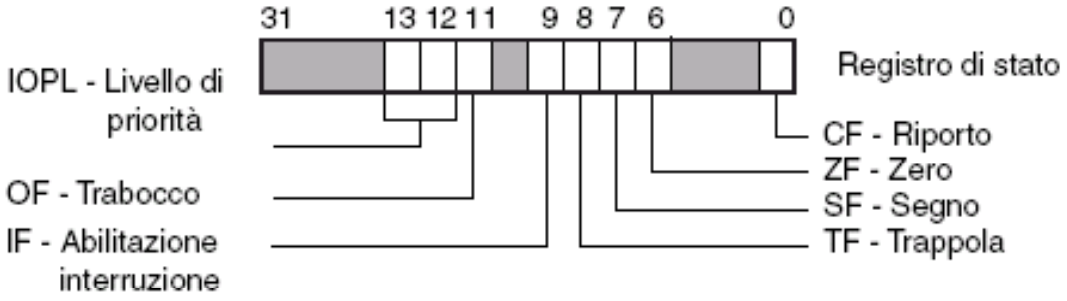
- Sono sei registri da 16 bit ciascuno, che specificano i segmenti del programma.
- Il contenuto di ciascuno di tali registri è un riferimento alla MMU, dove si trovano l'indirizzo (virtuale) del segmento corrispondente e i permessi di accesso.
- I tre segmenti fondamentali del programma sono:
  - il segmento di codice – CS
  - il segmento principale dei dati – DS
  - e il segmento di pila – SS
- Questi tre segmenti sono usati in modo automatico, i segmenti rimanenti sono tutti di dato e sono ausiliari.

Registri



Vista d'insieme.

Registro di Stato - EFLAGS



EFLAGS contiene i bit di esito, di modo IOPL e di controllo di interruzione.

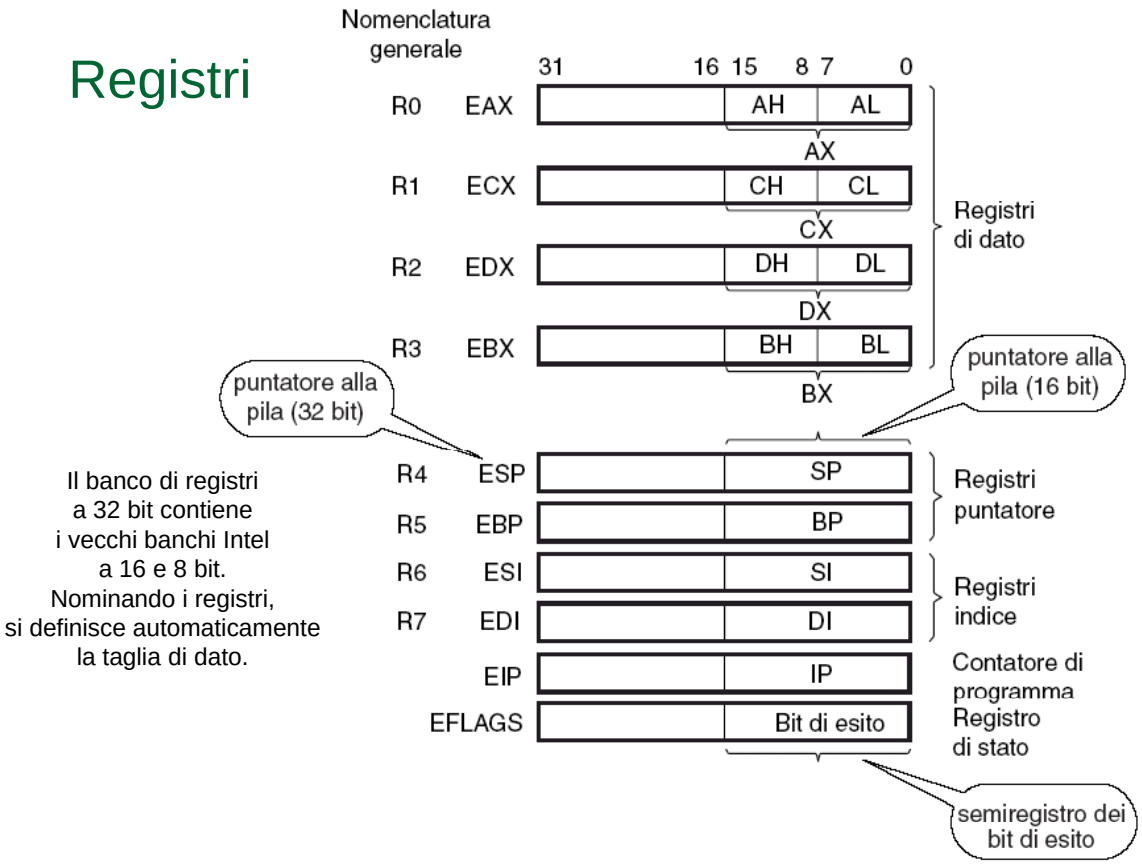
Vedi il capitolo di aritmetica per la definizione dei bit di esito.

I processori IA-32 hanno svariati modi di esecuzione, con livelli di privilegio differenziati.

Nomi dei Registri - GAS

- I registri IA-32 hanno nomi ereditati dalle generazioni precedenti di processori Intel:
  - IA-8 famiglia Intel a 8 bit
  - IA-16 famiglia Intel a 16 bit
- Ciò permette la compatibilità SW tra i linguaggi di tutti i processori Intel, di ogni generazione.
- Vedi la figura successiva per la nomenclatura dei registri, a 32, 16 e 8 bit.
- In notazione GAS, i nomi dei registri vanno sempre preceduti da “%”, per esempio “%EAX” (a 32 bit), “%AX” (a 16 bit) e “%AL” (a 8 bit).

Registri



Il banco di registri a 32 bit contiene i vecchi banchi Intel a 16 e 8 bit. Nominando i registri, si definisce automaticamente la taglia di dato.

Particolarità

- Il registro EIP è il contatore di programma.
- Il registro ESP è il puntatore alla pila.
- Il registro EBP è quello usato più di frequente come registro base del modo di indirizzamento indiretto con base e indice.
- Il registro EBP è usato come puntatore all'area di attivazione (frame pointer) per la gestione di sottoprogramma.
- I registri ESI e EDI sono usati come contatori da certe istruzioni macchina di tipo iterativo.

Modi di indirizzamento classi di istruzioni

MODI DI INDIRIZZAMENTO E ISTRUZIONI

Modo di Indirizzamento

- In IA-32 sono disponibili numerosi e diversificati modi di indirizzamento (vedi elenco), sia di dato sia di istruzione (per salto), o di uso misto.
- Mancanza più vistosa: modo di indirizzamento indiretto da memoria (c'è indiretto da registro), tranne che per l'istruzione di salto (incond.).
- Alcuni modi presentano restrizioni più o meno significative, allo scopo di ottimizzare la codifica numerica dell'istruzione macchina.
- Di seguito si dà la notazione in sintassi GAS.

Elenco di Modi IA-32 - GAS

Modo	Sintassi (GAS)	Funzionamento (in RTL)
Modi di indirizzamento di dato		
immediato (o di costante)	$VAL	operando = VAL
di registro	%R*i*	operando = [R*i*], risultato = R*i*
assoluto (o diretto)	IND	i.e. = IND
indiretto da registro	(%R*i*)	i.e. = [R*i*]
con base e spiazzamento	SPI (%R*i*)	i.e. = SPI + [R*i*]
con indice e spiazzamento	SPI (, %R*j*.F)	i.e. = SPI + [R*j*].F
con base e indice	(%R*i*, %R*j*.F)	i.e. = [R*i*] + [R*j*].F
con base indice e spiazzamento	SPI (%R*i*, %R*j*.F)	i.e. = SPI + [R*i*] + [R*j*].F
Modi di indirizzamento di istruzione (tutte le istr. di salto)		
relativo a contatore di prog.	ETI	i.e. = posizione di ETI
Solo per JMP (salto incondizionato) e CALL (salto a sottoprogramma)		
indiretto da memoria	*IND, oppure	i.e. = [IND]
	SPI (, %R*j*.F)	i.e. = [SPI + [R*j*].F]

si può aggiungere il registro base

Notazione

Simboli a sinistra di "=":

"operando" è il *dato* che l'istruzione userà (argomento di tipo sorgente)

"risultato" è il *dato* che l'istruzione produrrà (argomento di tipo destinazione)

"i.e." indica l'*indirizzo effettivo* di memoria dell'argomento (sorgente o destinazione) su cui lavorerà l'istruzione; nell'istruzione di salto indica l'*indirizzo di destinazione*

Simboli a destra di "=":

"VAL" è un *intero relativo* (a 8 o 32 bit), dato come numero o simbolo

"IND" è un *indirizzo assoluto* (a 32 bit, ≥ 0), dato come numero o simbolo

"ETI" è un'*etichetta* che marca la posizione (cioè l'indirizzo) di un'istruzione

"SPI" è un *intero relativo* (a 8 o 32 bit), dato come numero o simbolo (nel modo con indice e spiazzamento può essere solo a 32 bit)

"R*i*" e "R*j*" sono *registri di uso generale*: EAX, EBX, ECX, EDX, ESP, EBP, ESI e EDI (è vietato usare come registro indice il puntatore alla pila ESP)

"F" è il *fattore di scala* e può valere 1, 2, 4 o 8

Il valore costante (immediato) ha sempre il prefisso "$".
L'indirizzo diretto (assoluto) è senza prefisso.

Osservazioni

- Come registro base si può utilizzare uno qualunque dei registri di uso generale a 32 bit.
- Molto comunemente come registro base si usa EBP (extended base pointer), a 32 bit.
- Come registro indice si può utilizzare uno qualunque dei registri, tranne ESP (a 32 bit, riservato come puntatore alla pila).
- Il registro EBP, a 32 bit, è normalmente usato come puntatore all'area di attivazione di sottoprogramma o "frame pointer".

Osservazioni

- Il registro indice ammette di specificare un fattore di scala, cioè un intero F = 1, 2 o 4.
- Il fattore di scala F definisce se il registro indice sia puntatore a byte, a parola allineata di 16 bit o allineata di 32 bit, rispettivamente.
- Di fatto, l'indirizzo contenuto nel registro indice viene moltiplicato per il fattore F.
- Nota bene: il registro indice resta inalterato.
- Il fattore di scala F = 1 (registro indice puntatore a byte) vale per difetto.

Classi di Istruzioni Macchina

- In IA-32 sono disponibili tutte le classi usuali di istruzioni macchina, vale a dire le seguenti:
  - trasferimento tra registri (copia tra registri), o tra registro e memoria (cioè caricamento e memorizzazione)
  - ingresso e uscita – se lo spazio di indirizzamento di I/O è separato da quello di memoria
  - operazioni aritmetiche e logiche tra registri o tra registri e memoria (ma non solamente in memoria)
  - salto incondizionato, condizionato (ai bit di esito) e gestione di sottoprogramma
  - istruzioni particolari: di controllo e privilegiate
- Nota bene: è sempre possibile unificare gli spazi di indirizzamento di I / O e memoria, e così non usare le istruzioni di I / O sostituendole con MOV.

Argomenti

- Il repertorio IA-32 comprende istruzioni:
  - binarie (2 argomenti)
  - unarie (uno solo)
  - e nullarie (nessuno)
- Non ci sono istruzioni a tre o più argomenti.
- In sintassi GAS l'ordinamento degli argomenti è del tipo **sorgente,destinazione**, mentre in sintassi MASM è del tipo **destinazione,sorgente**.
- I nomi di istruzione (codici mnemonici) sono per lo più autoesplicativi.

Taglia di Dato

- Numerose istruzioni manipolano o elaborano dati (p. es. trasferimento o aritmetica-logica).
- È facoltativo specificare la taglia in bit del dato, concatenando il segnale di taglia al nome dell'istruzione (non agli argomenti):
  - L indica dato da 32 bit (segnale di default)
  - B indica dato da un byte
- Nota bene: in ogni caso, se come argomento di istruzione si usa un nome di registro a 16 o 8 bit, la taglia del dato è fissata automaticamente uguale alla taglia del registro nominato.

Trasferimento

- C'è un solo nome (codice mnemonico):
  - MOV sorgente, destinazione
- L'istruzione MOV ammette l'uso di tutti i modi di indirizzamento (è la più generale).
- Essa è semiortogonale: l'argomento sorgente o quello destinazione deve essere un nome di registro (con qualche modesta eccezione).
- Per esempio, non è permessa la copia diretta di una parola da memoria a memoria.

Notazione - I

reg – uno qualunque degli otto registri di uso generale del processore:

- EAX, EBX, ECX, EDX, ESP, EBP, ESI e EDI
- o le rispettive versioni a 16 e 8 bit

mem – indirizzo effettivo di memoria (a 32 bit) prodotto da uno qualunque dei modi di indirizzamento riferibili allo spazio di memoria, vale a dire i seguenti:

- diretto (o assoluto)
- indiretto da registro
- con base e spiazzamento
- con indice e spiazzamento
- con base e indice
- con base indice e spiazzamento

Notazione - II

ind I / O – indirizzo di I / O (a 8 bit), specificato in modo diretto, riferito a spazio di indirizzamento separato da spazio di memoria

imm – operando immediato (costante) a 8 o 32 bit

imm 8 – operando immediato (costante) a solo 8 bit

dist – indirizzo di destinazione di salto specificato come distanza (a 8 o 32 bit, in complemento a due) relativa a contatore di programma

cod 8 – codice da 8 bit, specificato come argomento nell'istruzione di chiamata a supervisore o SVC (*supervisor call*)

Come d'uso, le istruzioni sono elencate in tabelle
e specificate in formalismo RTL (appendice).

MOV - Semiortogonale

codice mnemonico (nome)	dim. di dato	modo di indirizzamento		operazione svolta	bit di esito aggiornati			
		sorg.	sorg. dest.		S	Z	O	C
MOV move	B, L	reg mem reg imm imm	reg reg mem reg mem	$d \leftarrow s$ (trasferimento di parola)				

Trasferimento di I / O

- L'istruzione MOV può anche svolgere funzione di trasferimento di I / O:
  - se gli spazi di indirizzamento di I/O e memoria sono unificati (caso molto frequente)
- Ma ci sono istruzioni di I / O specifiche:
  - se gli spazi di indirizzamento di I/O e memoria sono separati (meno comune, vedi di seguito)
- Le istruzioni di I / O sono IN e OUT.

IN e OUT (GAS)

IN	IND_DISP, registro
OUT	registro, IND_DISP

- Lo spazio di indirizzamento di I / O ha indirizzo da 16 bit (max 216 porte di I / O da un byte).
- Si può leggere (IN) o scrivere (OUT) da o su periferica un dato da 8 o 32 bit, a scelta.
- IND_DISP è l'indirizzo diretto, a 8 bit, del dispositivo di I / O, cioè della porta cui il dispositivo è collegato.
- La destinazione o sorgente "registro" deve essere il registro AL (dato a 8 bit) oppure EAX (dato a 32 bit).
- Usano solo le prime 256 porte dello spazio di I / O.
- Ci sono però varianti di IN e OUT capaci di usare l'indirizzo di I / O pieno a 16 bit (vedi il testo).
- MASM: notazione con sorgente e destinatario invertiti.

Aritmetica-Logica

- Svariati nomi (codici mnemonici):
  - due operandi

    ADD

    SUB

    AND

    OR
  - un operando

    NEG (comp. a due)

    NOT (neg. logica)

  e poche altre
- Tutte queste istruzioni aggiornano i bit di esito, dove ha senso.

Aritmetica-Logica

- Sono tutte (o quasi) semi-ortogonali:
  - se sorgente è un nome di registro, la destinazione è qualunque
  - se destinazione è un nome di registro, la sorgente è qualunque
- C'è qualche (modesta) eccezione.

ADD - Semiortogonale

codice mnemonico (nome)	dim. di dato	modo di indirizzamento		operazione svolta	bit di esito aggiornati			
		sorg.	sorg. dest.		S	Z	O	C
ADD add	B, L	reg mem reg imm imm	reg reg mem reg mem	$d \leftarrow s_2 + s_1$ (addizione senza riporto in naturale o in comp. a due)	×	×	×	×

- ADD è essenzialmente semiortogonale.
- Almeno un argomento deve essere in modo di registro, l'altro è libero.
- Eccezione: addiziona a memoria una costante.
- ADD aggiorna i bit di esito (nel modo ovvio).

AND - Semiortogonale

codice mnemonico (nome)	dim. di dato	modo di indirizzamento		operazione svolta	bit di esito aggiornati			
		sorg.	sorg. dest.		S	Z	O	C
AND logical and	B, L	reg mem reg imm imm	reg reg mem reg mem	$d \leftarrow s_2 \wedge s_1$ (prodotto logico bit a bit)	×	×	0	0

- AND è essenzialmente semiortogonale.
- Almeno un argomento deve essere in modo di registro, l'altro è libero.
- Eccezione: prodotto logico di parola di memoria e costante.
- AND aggiorna i bit di esito, dove ha senso (riporto e trabocco azzerati, perché non hanno senso per AND).

Salto – in Generale

- Incondizionato:
  - JMP etichetta (modo relativo a PC)
- Condizionato:
  - Jcc etichetta (modo relativo a PC)
  - dove cc è una condizione sui bit di esito
- Sottoprogramma:
  - CALL etichetta (modo rel. a PC) – chiamata
  - RET – rientro
- La pila di sottoprogramma fa riferimento al registro ESP come puntatore alla cima, o SP.

Elenco di Condizioni – cc

Mnemonico	Nome (ing. / ita.) - jump if / salta se	Espressione
JS	signed (negative) ha segno (negativo)	$SF = 1$
JNS	not signed (positive or null) non ha segno (positivo)	$SF = 0$
JE / JZ	equal to / zero uguale / nullo	$ZF = 1$
JNE / JNZ	not equal / not zero diverso / non nullo	$ZF = 0$
JO	overflow (signed) c'è trabocco (comp. a due)	$OF = 1$
JNO	not overflow (signed) non c'è trabocco (comp. a due)	$OF = 0$

Elenco di Condizioni – cc

JC / JB	carry / below (unsigned) c'è riporto / minore (naturale)	$CF = 1$
JNC / JAE	no carry / above or equal (unsigned) non c'è riporto / maggiore o uguale (naturale)	$CF = 0$
JA	above (unsigned) maggiore (naturale)	$CF \vee ZF = 0$
JBE	below or equal (unsigned) minore o uguale (naturale)	$CF \vee ZF = 1$
JGE	greater than or equal to (signed) maggiore o uguale (comp. a due)	$SF \oplus OF = 0$
JL	less than (signed) minore (comp. a due)	$SF \oplus OF = 1$
JG	greater than (signed) maggiore (comp. a due)	$ZF \vee (SF \oplus OF) = 0$
JLE	less than or equal to (signed) minore o uguale (comp. a due)	$ZF \vee (SF \oplus OF) = 1$

Salto Incondizionato - JMP

- L'istruzione di salto incondizionato (JMP) e quelle affini (CALL) ammettono anche il modo di indirizzamento indiretto da memoria.
- Una parola di memoria contiene l'indirizzo effettivo (assoluto) di salto.
- Tale modo, esclusivo di JMP (e CALL), serve per avere in memoria tabelle di indirizzi di salto.
- Notazione e uso di tale modo sono un po' complicati, vedi l'esempio o il testo per dettagli.
- La tabella di indirizzi è utile per tradurre in modo efficiente il salto a più vie (p. es. switch di C).

Esempio – Tabella di Indirizzi

```
TAB_PUNT:   .long   BLOCCO_0, BLOCCO_1, ...
            ...     ...
MAIN:       istr.   ...
            ...     ...
            JMP     TAB_PUNT ( , %ESI, 4)
            ...     ...
BLOCCO_0:   istr.   ...
            ...     ...
BLOCCO_1:   istr.   ...
            ...     ...
```

tabella di indirizzi (puntatori)

blocchi di codice puntati (rami switch)

Effetto dell'istruzione JMP con indirizzamento indiretto da memoria:

$$\text{EIP} \leftarrow [\text{TAB_PUNTATORI} + ([\text{ESI}] \times 4)] \qquad \text{(ricorda che EIP è il PC)}$$

`switch (n) {BLOCCO_0: … ; BLOCCO_1: …; … }`

Ciclo a Conteggio – LOOP

codice mnemonico (nome)	dim. di dato	modo di indirizzamento		operazione svolta	bit di esito aggiornati			
		sorg.	sorg. dest.		S	Z	O	C
LOOP loop		dist		ECX ← [ECX] – 1 se ([ECX] ≠ 0) allora EIP ← indirizzo di salto (ciclo a conteggio in discesa)				

L'istruzione LOOP realizza il ciclo a conteggio in discesa (vedi sotto). Esempio di programma avanti.
L'istruzione ha alcune varianti.

LOOP etichetta

```
for (i == n - 1; i >= 0; i--) { /* istr. */ }
```

Confronto

- Istruzione di confronto:
  - CMP sorgente1, sorgente2
- Effetto:
  - sottrae s1 da s2 e aggiorna i bit di esito
  - la differenza va persa
- In genere è seguita da salto condizionato.
- È essenzialmente semiortogonale, come aritmetica-logica (e con stesse eccezioni).

Gestione di Pila

- Il repertorio IA-32 prevede istruzioni macchina specifiche per gestire la pila.
- Istruzioni:
  - POP e PUSH
- Le due istruzioni hanno modi di indirizzamento:
  - di registro (argomento nome di registro)
  - diretto (argomento indirizzo assoluto di mem.)

Gestione di Pila

- Ricorda che la pila del processore IA-32:
  - cresce verso il basso
  - e che il puntatore alla cima della pila si riferisce all'ultima cella occupata (non alla prima cella libera)
- Incremento e decremento del puntatore alla pila si adattano automaticamente alla taglia, espressa in byte, del dato da spilare o impilare, rispettivamente.
- Il dato impilato o spilato può essere collocato altrove in memoria o in un registro.

Operazioni PUSH e POP

codice mnemonico (nome)	dim. di dato	modo di indirizzamento		operazione svolta	bit di esito aggiornati			
		sorg.	sorg. dest.		S	Z	O	C
POP pop off stack	L		reg mem	$d \leftarrow [[ESP]]$ $ESP \leftarrow [ESP] + 4$ (impilamento di parola doppia)				
PUSH push onto stack	L	reg mem imm		$ESP \leftarrow [ESP] - 4$ $[ESP] \leftarrow s$ (spilamento di parola doppia)				

Varie

- Istruzioni per esaminare o aggiornare un bit in un registro o in una parola di memoria:
  - BT, BTS, BTR e BTC
- Istruzioni per fare scorrere (shift) a sx o dx:
  - SHL e SHR

  o ruotare (rotate) a sx o dx:
  - ROL e ROR

  un registro o una parola di memoria.
- Istruzioni INC e DEC per incrementare o decrementare un registro o una parola di mem.

Estensioni

- Il nucleo del repertorio di istruzioni IA-32 ammette come estensione alcuni gruppi, più o meno ampi, di istruzioni specializzate:
  - gruppo in virgola mobile (IEEE 754), usa il banco di registri FP (floating point, a 32 / 64 bit)
  - gruppo MMX, per applicazioni multimediali, usa il banco di registri MMX (aggiuntivi)
  - gruppo SSE, per applicazioni di calcolo scientifico, usa il banco di registri SSE (a 128 bit)
  - gruppo SSE2, ulteriore estensione di SSE
- Tali gruppi di estensione non sono di necessità disponibili in tutti i modelli di processore IA-32.

Istruzione LEA (GAS)

- L'istruzione Load Effective Address:
  - LEA arg_riferito_a_memoria, registro

  carica nel registro dato come argomento destinazione, l'indirizzo effettivo (di memoria) dell'argomento sorgente.
- Di solito serve per inizializzare "registro" e usarlo in seguito come registro puntatore.
- Il primo argomento (sorgente) deve generare un indirizzo effettivo riferito a memoria (altrimenti LEA non ha senso).
- MASM:
  - LEA REG, memoria

Esempio – Somma di Vettore

	LEA	NUM1, %EBX	EBX è il puntatore all'elemento iniziale della lista
	MOV	N, %ECX	ECX è il contatore del numero di giri del ciclo
	MOV	$0, %EAX	EAX è l'accumulatore della somma di elementi
	MOV	$0, %EDI	EDI è il puntatore all'elemento corrente della lista
CICLO	ADD	(%EBX, %EDI, 4), %EAX	accumula l'elemento corrente in EAX
	INC	%EDI	incrementa il puntatore all'elemento corrente
	DEC	%ECX	decrementa il contatore di giri
	JG	CICLO	se [%ECX] > 0 riesegui il ciclo
	MOV	%EAX, SOMMA	memorizza la somma

Versione estesa (realizza ciclo con salto retroverso).

Esempio – Somma di Vettore (GAS)

	LEA	NUM1, %EBX	EBX è il puntatore all'elemento iniziale della lista
	SUB	$4, %EBC	decrementa EBX di 4 unità (punta a inzio lista)
	MOV	N, %ECX	ECX è il puntatore corrente e il contatore di ciclo
	MOV	$0, %EAX	EAX è l'accumulatore della somma di elementi
CICLO	ADD	(%EBX, %ECX, 4), %EAX	accumula l'elemento corrente in EAX
	LOOP	CICLO	decrementa ECX e poi se [%ECX] > 0 riesegui ciclo
	MOV	%EAX, SOMMA	memorizza la somma

Versione compatta (realizza ciclo con istruzione iterativa LOOP).

Direttive e come organizzare il programma

DIRETTIVE E STRUTTURE DEL PROGRAMMA

Direttive

- In IA-32 valgono tutte le direttive già viste del sistema di assemblaggio GAS.
- Di solito il programma non segmentato dichiara prima i dati e poi la lista di istruzioni (ma il formato è del tutto libero).
- Generalmente l'indirizzo di inizio del programma è segnalato dall'etichetta MAIN (convenzione simile a quella C).

Breve Riassunto

	`.equ`	simbolo, valore
simbolo	=	valore
etichetta:	`.org`	indirizzo
etichetta:	`.byte`	lista_di_valori_a_8_bit
etichetta:	`.word`	lista_di_valori_a_16_bit
etichetta:	`.long`	lista_di_valori_a_32_bit
etichetta:	`.quad`	lista_di_valori_a_64_bit
etichetta:	`.space`	numero_di_byte
etichetta:	`.text`	
etichetta:	`.data`	

Segmentazione

- Il sistema operativo dove andrà eseguito il programma può avere memoria virtuale.
- In tale caso si può strutturare in segmenti il programma, secondo vari modelli.
- Gli indirizzi delle istruzioni e dei dati sono allora da intendere come virtuali.
- Per i dettagli su tipo e uso dei segmenti, si deve vedere il S.O. specifico.

Segmentazione

- Generalmente si hanno almeno:
  - un segmento di codice (text)
  - uno di dati globali (data)
  - e un segmento di pila (stack)
- I segmenti di testo e dei dati possono essere suddivisi in sezioni e riempiti a più riprese.
- Il segmento di pila è dichiarato in modo implicito, pertanto non figura nel codice simbolico ma viene aggiunto su richiesta in esecuzione.
- Altri segmenti possibili (mem. condivisa, ecc).

Esempio GAS – Segmentato

commento	etichetta	dir./istr.	argomento / i
titolo		.title	"prog."
segmento di dati		.data	
ind. virtuale iniz. dati		.org	0x201150
	NUM1:	.long	17, 3, −51, 242, −113
	N:	.long	5
	SOMMA:	.long	0
segmento di codice		.text	
ind. virtuale iniz. codice		.org	0x201200
	MAIN:	LEA	NUM1, %EBX
		SUB	$4, %EBC
		MOV	N, %ECX
		MOV	$0, %EAX
	CICLO:	ADD	(%EBX, %ECX, 4), %EAX
		LOOP	CICLO
		MOV	%EAX, SOMMA

Indirizzamento di Segmento

- In IA-32 il programma è suddiviso in segmenti.
- I segmenti standard sono denominati:
  - CS, codice eseguibile
  - DS, dati (statici, var. globali, e dinamici, heap)
  - SS, pila (aree di attivazione di sottoprogramma)
- I tre registri di segmento omonimi CS, DS e SS ne definiscono l'indirizzo (virtuale) iniziale, che la MMU traduce poi in indirizzo fisico.
- Al segmento sono associati anche permessi.
- Il programma può avere altri tre segmenti, normalmente non usati.

Segmentazione Automatica

- Il processore IA-32 sceglie automaticamente come e quando lavorare nei tre segmenti standard elencati prima.
- Ogniqualvolta il processore genera un indirizzo effettivo, gli associa il segmento (ovvero il registro di segmento) dove localizzarlo.
- La scelta del segmento avviene in funzione dell'istruzione, del modo di indirizzamento e del registro che hanno generato l'indirizzo effettivo.
- Di solito il programmatore non interviene in tale meccanismo, che è appropriato per programmi dalla struttura ordinaria, senza funzioni speciali.

Segmento di Codice - CS

- Qualunque indirizzo effettivo generato:
  - in modo diretto da un'istruzione di salto
  - oppure generato (in un modo qualunque) a partire dal registro EIP (contatore di programma)

  si riferisce allo spazio del segmento di codice.
- Di conseguenza, il prelievo di istruzioni avviene nel segmento CS (come del resto è naturale).
- Ciò vale sia quando si procede in sequenza sia quando c'è salto (di qualunque tipo).

Segmento dei Dati - DS

- Qualunque indirizzo effettivo generato:
  - in modo diretto da un'istruzione che manipola dati (trasferimento, aritmetica o logica)
  - oppure generato a partire da un registro di uso generale utilizzato come registro base o indice (tranne ESP e EBP, se sono usati come base) si riferisce allo spazio del segmento dei dati.
- Di conseguenza caricamento e memorizzazione dei dati avvengono nel segmento DS (come del resto è naturale).

Segmento di Pila - SS

- Qualunque indirizzo effettivo generato a partire dal registro ESP utilizzato:
  - come puntatore, in modo esplicito o implicito (p. es. da POP e PUSH, o CALL e RET)
  - oppure come registro base si riferisce allo spazio del segmento di pila.
- Di conseguenza le istruzioni POP e PUSH e lavorano (come sorgente e destinazione, rispettivamente) in pila.

Segmento di Pila - SS

- Qualunque indirizzo effettivo generato a partire dal registro EBP utilizzato:
  - come registro base si riferisce allo spazio del segmento di pila.
- Ciò spiega perché EBP sia la scelta normale come registro puntatore all'area di attivazione di sottoprogramma o "*frame pointer*" - FP
- Di conseguenza, qualunque argomento di istruzione, specificato in modo con base EBP e spiazzamento, lavora nell'area di attivazione corrente sulla cima della pila (vedi esempi).

Cambio di Segmento

- Si può ridirigere in modo esplicito l'indirizzo effettivo verso un segmento diverso dalla scelta standard automatica.
- Si deve usare una notazione opportuna, che evidenzi il nome del registro di segmento alternativo.
- Ciò si può fare sia per gli argomenti sia per il prelievo dell'istruzione stessa.

Esempio – Argomento (GAS)

- L'indirizzo diretto NUMERO si riferisce al segmento standard dei dati DS:

```
MOV        NUMERO, %EAX
```

- L'indirizzo diretto NUMERO si riferisce al segmento alternativo ES:

```
MOV        %ES: NUMERO, %EAX
```

- Si veda il capitolo 12 per l'illustrazione più dettagliata del modello di memoria IA-32.

Sintassi Nativa

- Sono disponibili sistemi di assemblaggio per IA-32 in sintassi nativa, come definita originariamente da Intel.
- I nomi di istruzione sono identici a GAS.
- La notazione dei modi di indirizzamento è invece MOLTO DIVERSA da GAS.
- Le direttive sono pure diverse da GAS.
- Per i dettagli, vedi documentazione Intel.

Cenni alla forma numerica dell'istruzione

FORMA NUMERICA

Collocazione in Memoria

- L'istruzione macchina IA-32 è codificata numericamente in una successione di byte.
- L'istruzione può iniziare a un indirizzo di byte qualunque, senza bisogno di essere allineata alla parola da 16 o 32 bit:
  - prelevare un'istruzione IA-32 è complicato
- Per codificare un'istruzione occorre al minimo un byte (e spesso basta) fino a 11 byte (per le istruzioni più complesse), e talora anche oltre.
- Il programma consiste in una lista contigua di istruzioni, collocate in memoria.

Regole di Codifica

- La forma numerica dell'istruzione segue regole molto complesse (vedi appendice per i dettagli).
- Strutturazione generale dell'istruzione:
  - codice operativo (codifica nome dell'istruzione e argomenti semplici, è l'unico elemento obbligatorio)
  - modi di indirizzamento degli argomenti (max 2 arg.)
  - spiazzamento o indirizzo diretto (naturale o comp. 2)
  - operando di tipo costante (immediato, nat. o comp. 2)
- Alcune istruzioni ammettono un byte di "prefisso", per generalizzarne le funzioni.

Formato Numerico

codice operativo	modo di indirizzamento	spiazzamento o indirizzo diretto	operando immediato (cost.)
1 o 2 byte	1 o 2 byte	1 o 4 byte	1 o 4 byte

- Le istruzioni semplici e frequenti sono codificate tramite il solo codice operativo, gli altri elementi non servono.
- I campi rimanenti servono per le istruzioni meno semplici oppure poco frequenti, dotate di argomenti ingombranti (indirizzo o costante) o strutturalmente complessi.
- L'ingombro in byte dell'istruzione varia da 1 a 11 (con il prefisso anche di più), e non si estende di necessità su un numero esatto di parole da 16 o 32 bit.

Distanza di Salto

- Le istruzioni di salto più comuni hanno modo di indirizzamento relativo a PC.
- In pratica l'argomento del salto rappresenta la distanza orientata della destinazione rispetto all'istruzione di salto stessa:
  - positiva se salto in avanti
  - negativa se salto all'indietro (salto retroverso)
- Come riferimento si prende però l'indirizzo del primo byte dell'istruzione consecutiva a quella di salto (non quello dell'istruzione di salto stessa).
- Nota bene: JMP e CALL hanno anche modo di indirizzamento indiretto da memoria.

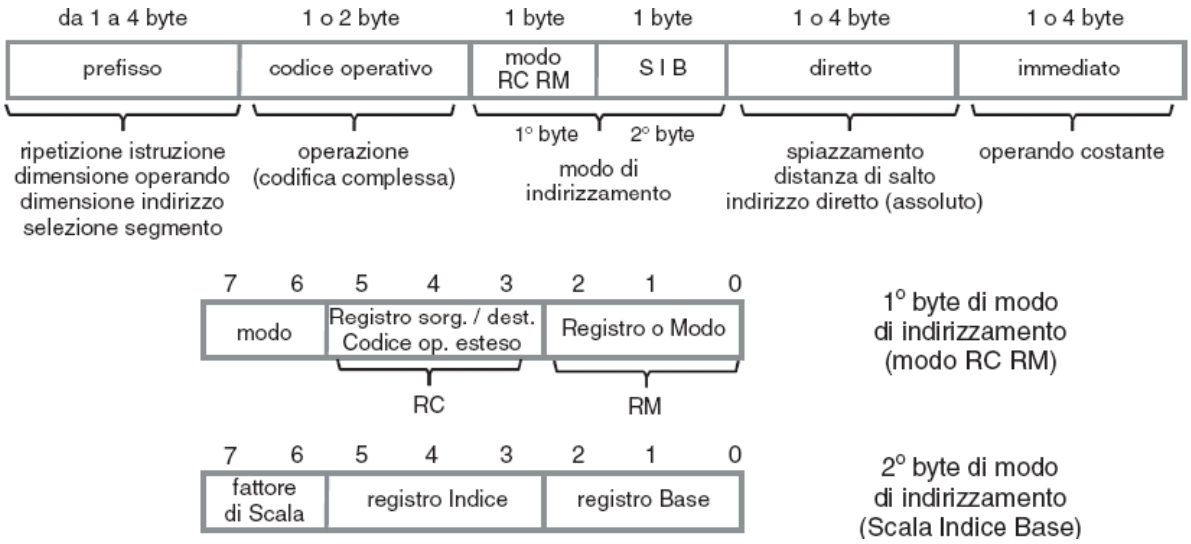
Esempio - Ciclo

codifica hex istruzione			programma in linguaggio macchina		
03	04	BB	CICLO:	ADD	(%EBX, %EDI, 4), %EAX
47				INC	%EDI
49				DEC	%ECX
7F	F9			JG	CICLO

comp. a 2, vale – 7 in decimale

- Si noti come le istruzioni siano codificate numericamente su 3, 1, 1 e 2 byte, rispettivamente; totale 7 byte.
- La distanza di salto (retroverso) è relativa al primo byte dell'istruzione che segue JG (qui non è mostrata).
- Le istruzioni sono contigue e non allineate alla parole.

Formato - Dettagli



Vista d'insieme con qualche dettaglio della codifica di istruzione IA-32.
Si nota la presenza (facoltativa) dei byte di prefisso.
Per la spiegazione, si vedano l'appendice o la documentazione Intel.

Qualche esempio semplice

ESEMPI SEMPLICI DI PROGRAMMA

Concatenamento di Cifre BCD

LEA	DATO, %EBP	predisponi EBP facendolo puntare ai dati
MOV	(%EBP), %AL	carica in AL la prima cifra BCD
SHL	$4, %AL	fa scorrere AL in modo logico di 4 bit a sinistra
MOV	1 (%EBP), %BL	carica in BL la seconda cifra BCD
AND	$0x0F, %BL	azzera in BL i bit in pos. 4, 5, 6, 7
OR	%BL, %AL	concatena in AL le due cifre BCD
MOV	%AL, COPPIA	memorizza in COPPIA il byte di risultato

Programma semplice che usa istruzioni logiche.

I/O – Tastiera e Video – I

Spazio di indirizzamento di I / O unificato con memoria.

	LEA	BLOCCO, %EBP	inizializza EBP facendolo puntare alla testa del blocco di byte in memoria dove scrivere i caratteri introdotti da tastiera
TASTIERA	BT JNC	$3, STATO_ING TASTIERA	attendi la pressione del tasto e l'introduzione del carattere in DATO_ING, monitorando S_ING
	MOV	DATO_ING, %AL	carica in AL il carattere (byte) contenuto in DATO_ING (così S_ING si azzera)
	MOV	%AL, (%EBP)	memorizza AL nella parola (byte) puntata da EBP
	INC	%EBP	incrementa EBP facendolo puntare all'elemento (byte) successivo del buffer di memoria

Gestione di periferica mediante controllo di programma – parte tastiera.

I/O – Tastiera e Video – II

VIDEO	BT JNC	$3, STATO_USC VIDEO	attendi che il video sia pronto, monitorando S_USC
	MOV	%AL, DATO_USC	scrivi in DATO_USC il carattere (byte) contenuto in AL (così S_USC si azzera)
	CMP JNE	$CR, %AL TASTIERA	verifica se il carattere (byte) introdotto da tastiera sia CR: se non lo è salta indietro e attendi la nuova introduzione da tastiera

Gestione di periferica mediante controllo di programma – parte video.

Sottoprogramma e area di attivazione

SOTTOPROGRAMMA

Routine

- Le istruzioni macchina CALL e RET realizzano il meccanismo di chiamata e rientro di sottoprog.
- L'istruzione CALL impila l'indirizzo di rientro, salvandolo:
  - il puntatore alla pila è il registro ESP (cioè SP)
- L'istruzione RET spila l'indirizzo di rientro, ripristinandolo in EIP (cioè in PC).
- La coppia CALL - RET fa uso implicito di SP.
- Di norma la pila è collocata nel segmento standard di pila SS, e la coppia CALL - RET fa riferimento a tale segmento automaticamente.

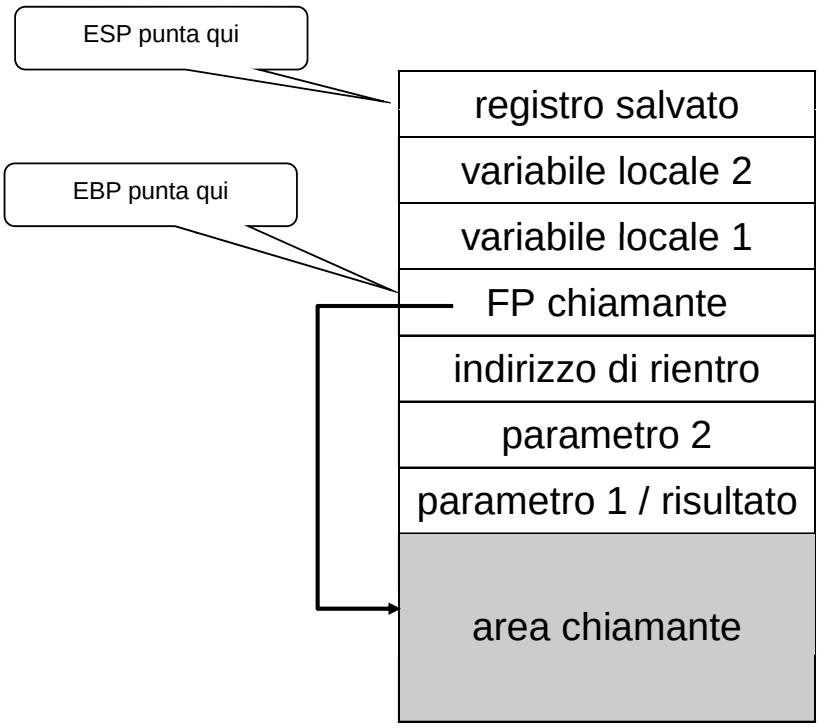
Parametri

- Se la routine (funzione o procedura) ha:
  - pochi parametri e piccoli
  - eventuale risultato pure piccolo
  - e poche o nessuna variabile locale
- gli uni e le altre si possono passare e tenere direttamente nei registri del processore (quelli di uso generale).
- Altrimenti occorre costruire e gestire in pila un'area di attivazione in piena regola.

Area di attivazione

- In IA-32 l'area di attivazione è strutturata secondo il modello generale già visto.
- Di solito il ruolo di registro puntatore all'area di attivazione, "frame pointer" o FP, è dato al registro EBP (ma non è obbligatorio).
- I modi di indirizzamento dove EBP figura come registro base generano un indirizzo effettivo che fa riferimento al segmento standard di pila.
- Ciò spiega perché si usi sempre EBP come FP.

Esempio



Generica area
di attivazione,
con 2 variabili locali
e 2 parametri,
e in cima
un registro salvato.
Si prevede
di sovrascrivere
il risultato
al parametro
numero 1.

Area di Attivazione (2/2)

- In IA-32 non è prevista nessuna istruzione speciale per facilitare costruzione dell'area di attivazione di sottoprogramma.
- La costruzione dell'area è fatta mediante sole istruzioni macchina ordinarie.
- Di solito l'accesso al contenuto dell'area è fatto mediante indirizzamento con registro base EBP e spiazzamento costante (vedi esempi).
- Ci sono comunque le istruzioni PUSH e POP, molto utili sia per salvare e ripristinare i registri sia per impilare e spilare i parametri.

Modello di Chiamante

…

impila eventuali parametri da passare
alla routine (tramite PUSH)

CALL INIZ

leggi o spila (POP) l'eventuale risultato

spila i parametri (tramite POP)
o abbassa la pila e abbandonali
…

Modello di Routine

INIZ: salva vecchio FP e crea spazio var. loc.

salva in pila eventuali registri da usare
(tramite PUSH)

corpo della routine, contenente i calcoli,
che usa le var. locali e i parametri tenuti
nell'area in pila, e che produce il risultato

sovrascrivi in pila il risultato a uno dei
parametri (o allo spazio apposito lasciato)

ripristina da pila eventuali registri salvati
(tramite POP)

togli spazio var. loc. e ripristina vecchio FP

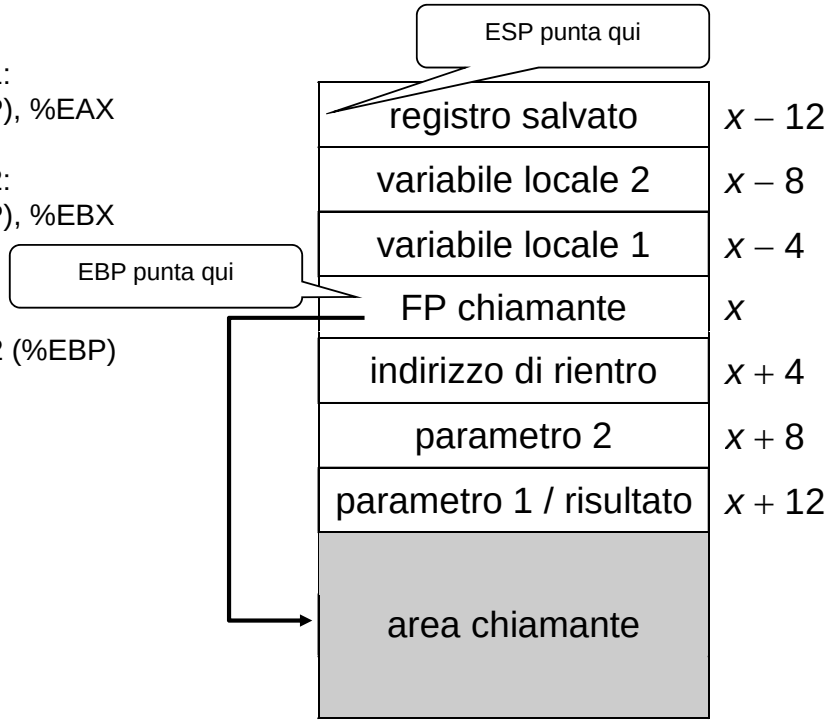
FINE: RET

Accesso a Parametri e Variabili Locali

- Nei suoi calcoli interni, la routine usa il registro FP (EBP) puntatore all'area per accedere:
  - ai parametri passatile dal programma chiamante
  - alle sue variabili locali
  - nonché per scrivere l'eventuale risultato
- Il registro puntatore FP va usato come registro base e lo spiazzamento deve essere:
  - positivo per i parametri e il risultato (che va sovrascritto a uno dei parametri)
  - negativo per le variabili locali
- Per i dettagli si veda l'esempio successivo.

Accesso all'Area

- Dati a 32 bit.
- Carica in EAX var. loc. n. 1:
  MOV –4 (%EBP), %EAX
- Carica in EBX var. loc. n. 2:
  MOV –8 (%EBP), %EBX
- Memorizza il risultato (calcolato in ECX):
  MOV %ECX, 12 (%EBP)

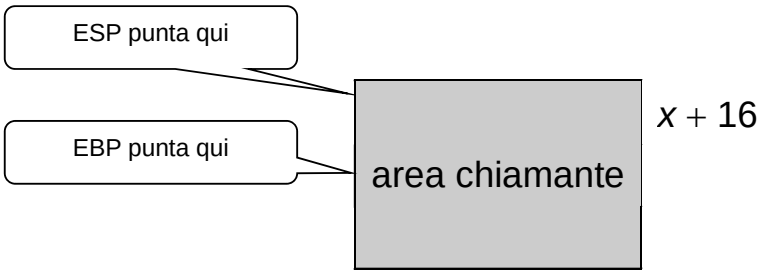


Ricorda che il registro EBP del processore contiene l'indirizzo *x*.

Simulazione

- Situazione iniziale, è attivo il chiamante.
- Tra poco partirà la sequenza per la routine.

L'area di attivazione del chiamante è sulla sommità della pila.

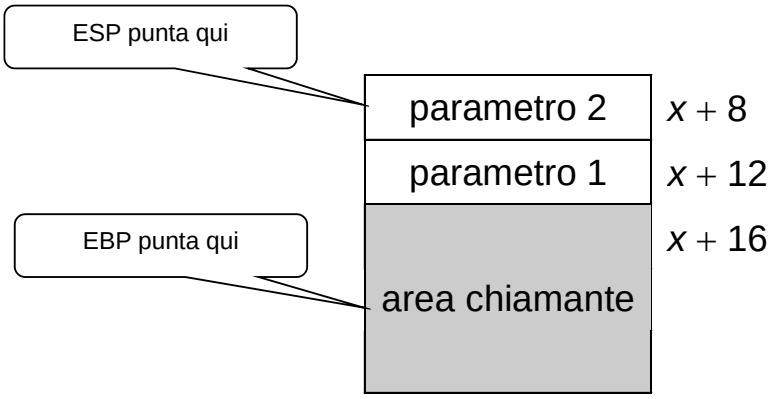


Simulazione

- Il chiamante impila i parametri:

PUSH %EAX

PUSH %EBX

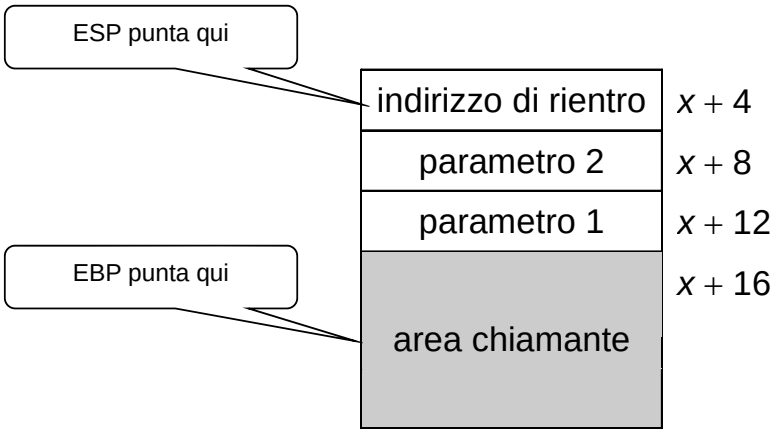


I due parametri vengono impilati.

Simulazione

- Il chiamante chiama la routine:

CALL INIZ

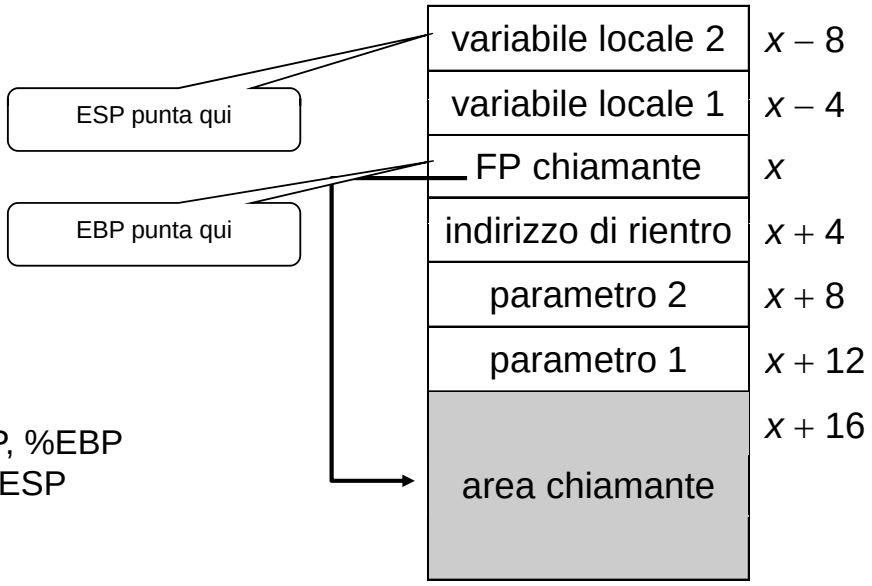


L'indirizzo di rientro viene salvato in pila.

Simulazione

- La routine salva FP e fa spazio per le var. loc.:

Poiché si vogliono avere 2 variabili locali da 32 bit ciascuna, lo spazio da lasciare in pila vale: 2 var. × 4 byte = 8 byte.

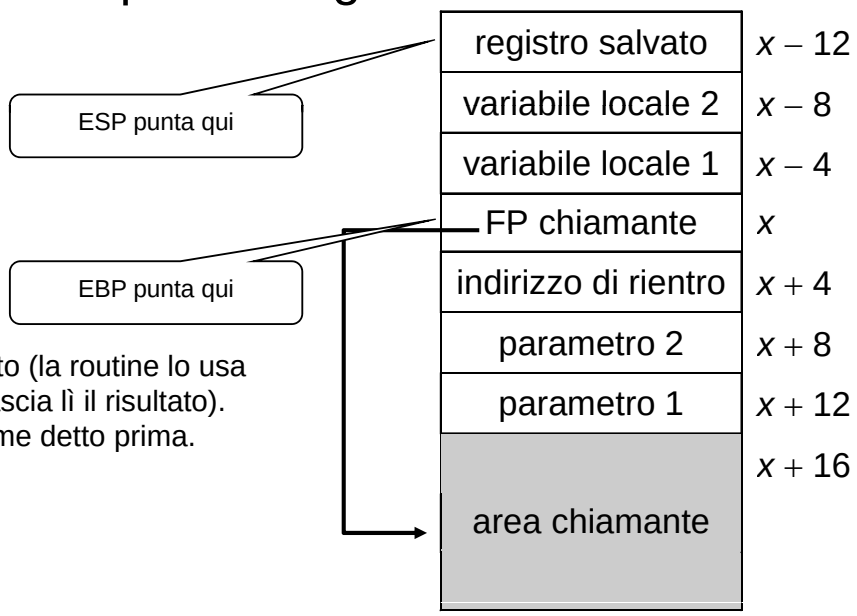



```
PUSH    %EBP
MOV             %ESP, %EBP
SUB             $8, %ESP
```

Simulazione

- La routine salva in pila un registro:

PUSH %ECX

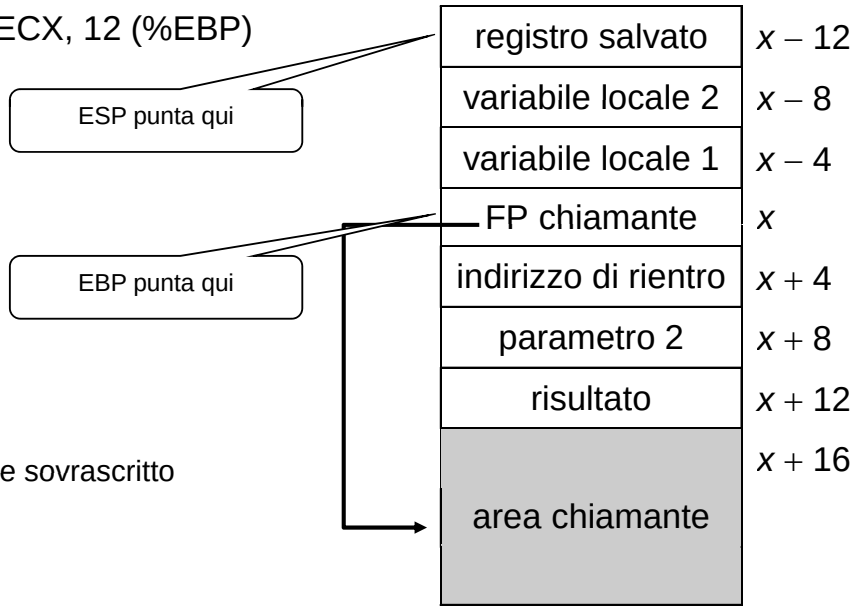


Il registro ECX viene liberato (la routine lo usa per i suoi calcoli interni e lascia lì il risultato). Var. loc. e par. si usano come detto prima.

Simulazione

- La routine sovrascrive in pila il risultato:

MOV %ECX, 12 (%EBP)

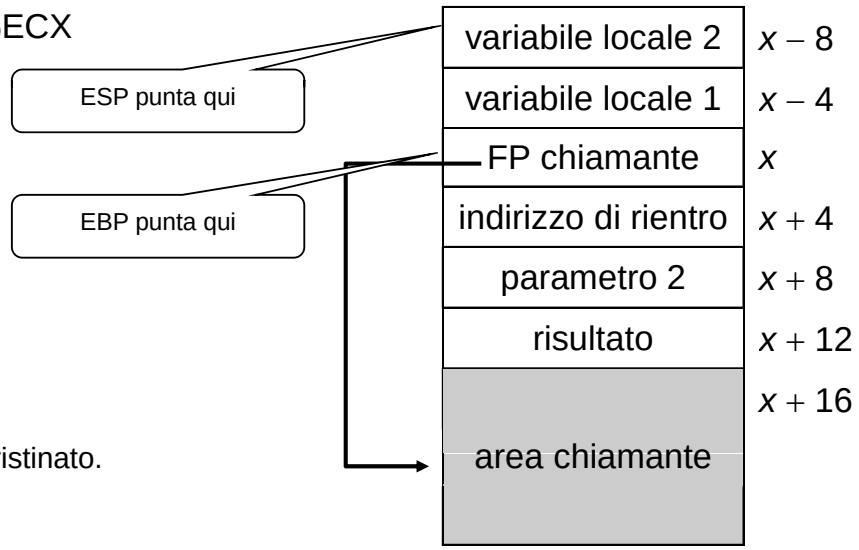


Il parametro numero 1 viene sovrascritto con il risultato (calcolato in ECX).

Simulazione

- La routine recupera dalla pila il registro salvato:

```
POP        %ECX
```

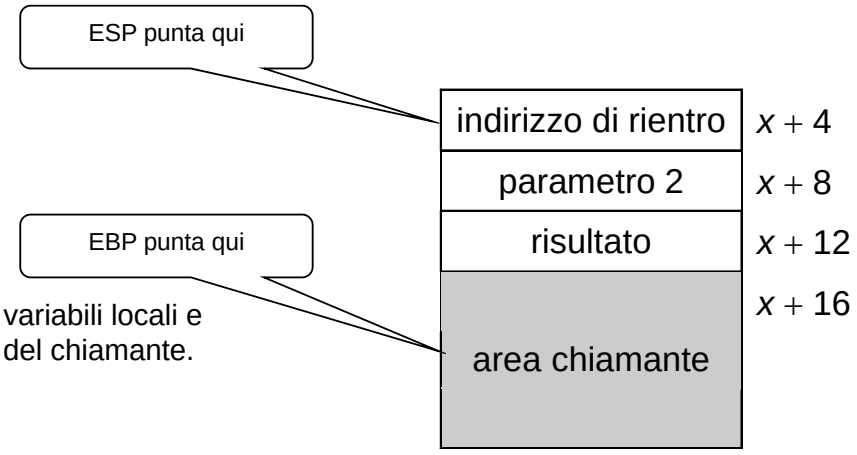


Il registro salvato viene ripristinato.

Simulazione

- La routine toglie spazio per le var. loc. e recupera FP:

```
ADD        $8, %ESP
POP        %EBP
```

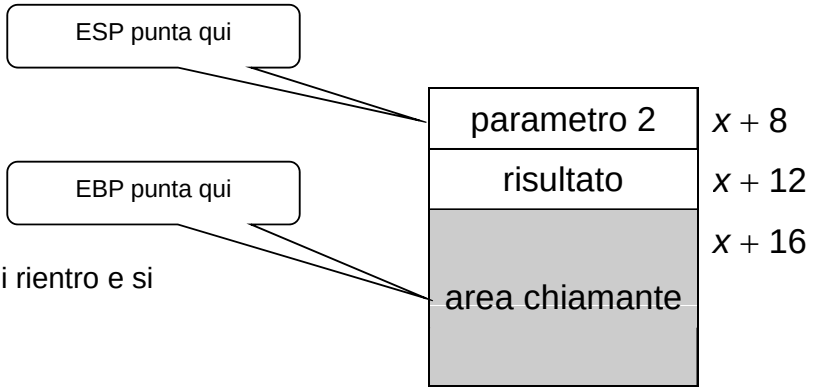


Vengono abbandonate le variabili locali e recuperato il valore di FP del chiamante.

Simulazione

- La routine esegue RET:

RET

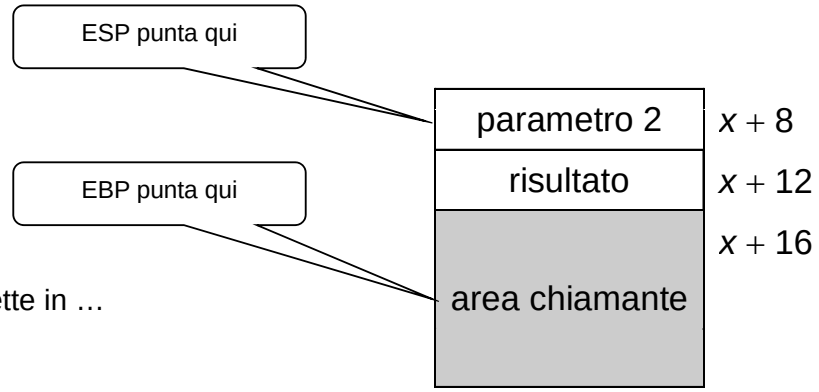


Viene recuperato l'indirizzo di rientro e si torna al chiamante.

Simulazione

- Il chiamante legge dalla pila il risultato:

MOV 4 (%ESP), …



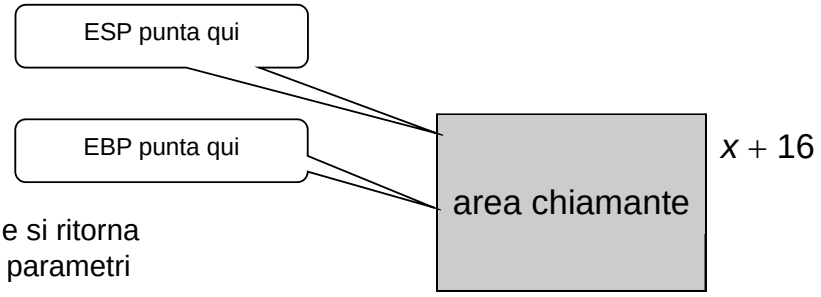
Si legge il risultato e lo si mette in …

Nota bene: si potrebbero anche spilare parametro 2 e risultato mediante POP, ci sono varie soluzioni possibili.

Simulazione

- Il chiamante abbassa la pila:

ADD $8, %ESP



Si abbandonano i parametri e si ritorna alla situazione iniziale (i due parametri occupano 8 byte).

Nota bene: è inutile abbassare se prima si è usata POP.

Digressione – POPAD / PUSAD

- Le istruzioni PUSHAD e POPAD impilano e spilano l'intero banco di registri.
- Funzionano con qualche particolarità, si vedano il testo o l'appendice.
- Per manipolare la pila, talvolta conviene usare MOV, altre volte PUSH e POP.
- In genere è questione di semplice buon senso, e ci sono varie soluzioni possibili.

Routine SUB1

PRIMA PARTE

```
SUB1   PUSH   %EBP                 salva il puntatore all'area
       MOV    %ESP, %EBP           aggiorna il puntatore all'area
       PUSH   %EAX                 salva EAX
       PUSH   %EBX                 salva EBX
       PUSH   %ECX                 salva ECX
       PUSH   %EDX                 salva EDX
       MOV    8 (%EBP), %EAX       carica i parametri: PAR1
       MOV    12 (%EBP), %EBX      e PAR2
       ...    ...                  altri calcoli di SUB1
```

Qui lo spazio previsto per le var. locali è nullo (non ci sono var. loc.).

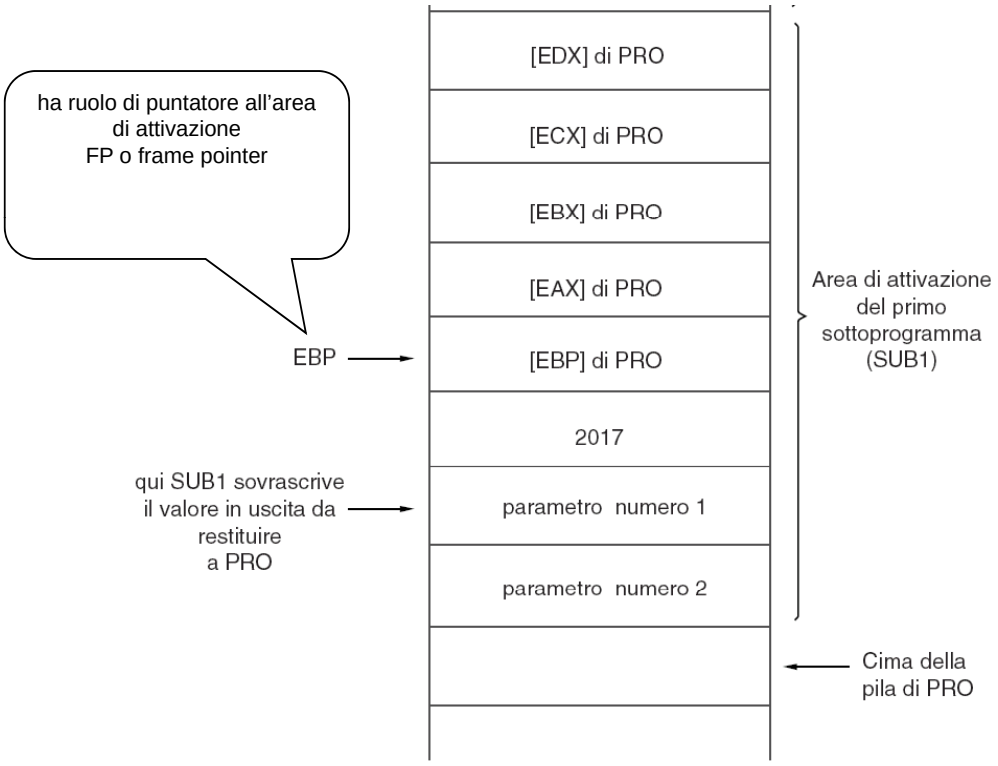
Routine SUB1

SECONDA PARTE

```
PUSH   PAR3                 impila il parametro PAR3
CALL   SUB2                 chiama SUB2
POP    %ECX                 spila il risultato
...    ...                  calcola ris. e lascia in EDX
MOV    %EDX, 8 (%EBP)       memorizza il risultato
POP    %EDX                 ripristina EDX
POP    %ECX                 ripristina ECX
POP    %EBX                 ripristina EBX
POP    %EAX                 ripristina EAX
POP    %EBP                 ripristina il puntatore all'area
RET                         rientra a PRO
```

Si riveda l'esempio completo nel capitolo 5.

Area di Attivazione – SUB1



Qualche esempio conclusivo

ALTRI ESEMPI

Prodotto Scalare di Vettori

	LEA	VETT_A, %EBP	fa puntare EBP al vettore A
	LEA	VETT_B, %EBX	fa puntare EBX al vettore B
	MOV	N, %ECX	carica in ECX la lunghezza n
	MOV	$0, %EAX	inizializza accumulatore EAX
	MOV	$0, %EDI	inizializza registro indice EDI
CICLO	MOV	(%EBP, %EDI, 4), %EAX	carica in EAX il fatt. di A
	IMUL	(%EBX, %EDI, 4), %EDX	e moltiplicalo per il fatt. di B
	INC	%EDI	incrementa l'indice EDI
	ADD	%EDX, %EAX	accumula il prodotto in EAX
	LOOP	CICLO	dec. ECX se [ECX] > 0 ciclo
	MOV	%EAX, PROD	memorizza risultato in PROD

Ciclo a conteggio realizzato tramite l'istruzione LOOP.

Ordinamento di Stringa - C

```
for (p = N - 1; p > 0; p = p - 1) {     /* esterno */
  for (q = p - 1; q >= 0; q = q - 1) { /* interno */
    if (STRINGA[q] > STRINGA[p]) {
      TEMP = STRINGA[q];                /* scambio */
      STRINGA[q] = STRINGA[p];
      STRINGA[p] = TEMP;
    } /* if */
  } /* for */
} /* for */
```

Algoritmo di ordinamento "bubblesort" per ordinare un vettore di byte, modellato come una stringa di caratteri – versione in linguaggio C.

Traduzione

	LEA	STRINGA, %EAX	EAX punta a STRINGA
	MOV	N, %EDI	predisponi l'indice p con
	DEC	%EDI	$n-1$ tenendolo in EDI
ESTERNO	MOV	%EDI, %ECX	predisponi l'indice q con
	DEC	%ECX	$p-1$ tenendolo in ECX
	MOV	(%EAX, %EDI), %DL	carica in DL il max corrente
INTERNO	CMP	%DL, (%EAX, %ECX)	conf. `LST[p]` e `LST[q]`
	JLE	CONTINUA	se falso non scambiare
	XCHG	%DL, (%EAX, %ECX)	scambia `LISTA[p]` = DL
	MOV	%DL, (%EAX, %EDI)	con `LISTA[q]`
CONTINUA	DEC	%ECX	decrementa indice `q`
	JGE	INTERNO	se [ECX] ≥ 0 riesegui ciclo
	DEC	%EDI	decrementa indice `p`
	JG	ESTERNO	se [EDI] > 0 riesegui ciclo

Traduzione di "bubblesort" in linguaggio macchina IA-32.

Per gestire l'interruzione e interagire con il S.O.

ISTRUZIONI SPECIALI E PRIVILEGIATE

Generalità

- Il linguaggio macchina di IA-32 prevede numerose istruzioni privilegiate e speciali.
- Le istruzioni privilegiate sono eseguibili solo quando il processore è in modo S:
  - IA-32 dispone di diversi modi di tipo S
  - con livelli di privilegio differenti
- Se si tenta di eseguire un'istruzione privilegiata con processore in modo U, essa è interpretata come evento di errore e per segnalarlo viene generata un'interruzione apposita.

Interruzione

- I processori IA-32 dispongono di meccanismo di interruzione, di tipo vettorizzato.
- La tabella dei vettori di interruzione sta in memoria e ne contiene 256, ciascuno da 32 bit.
- Per controllare il meccanismo di interruzione dal lato di processore, ci sono le due istruzioni privilegiate seguenti:
  - STI (set interrupt) - per abilitare l'interruzione
  - CLI (clear interrupt) - per disabilitare l'interruzione
- Per le interazioni tra l'interruzione e i vari modi S, si veda la documentazione tecnica Intel.

Rientro da Interruzione

codice mnemonico (nome)	dim. di dato	modo di indirizzamento		operazione svolta	bit di esito aggiornati			
		sorg.	sorg. dest.		S	Z	O	C
IRET return from interrupt				spila registro EIP spila registro EFLAGS (rientro da interruzione)	×	×	×	×

- L'istruzione IRET effettua il rientro da routine di servizio di interruzione.
- Essa ripristina stato e contatore di programma, ed è privilegiata, naturalmente.

Supervisor Call (SVC)

- L'istruzione SVC non è di per sé privilegiata, ma mette il processore in modo S o di supervisore.
- L'istruzione SVC somiglia però a un'interruzione, in quanto attiva una routine di servizio apposita.
- L'indirizzo della routine di servizio è specificato dall'eventuale argomento di SVC, e si trova nella tabella dei vettori di interruzione.
- L'indirizzo di rientro dalla routine viene salvato sulla pila di sistema.
- Il rientro dalla routine avviene come un normale rientro da interruzione, tramite istruzione IRET.

Chiamata a Supervisore (Supervisor Call - SVC)

INT	*s*	impila registro EFLAGS impila registro EIP passa a modo di sistema EIP ← vettore di interruzione (determinato da *s*)

In IA-32, l'istruzione INT realizza la chiamata a supervisore SVC.

L'istruzione INT si serve della tabella dei vettori di interruzione.

Il rientro da routine di servizio di INT si fa mediante istruzione IRET.

L'argomento *s* è un numero da 0 a 255 (256 possibili routine di servizio).

Per i dettagli, vedi la documentazione tecnica Intel.

Doppia Pila

- Il processore IA-32 contiene un solo registro puntatore a pila (ESP).
- Il meccanismo di doppia pila, di utente e di sistema, che è necessario per il funzionamento di un S.O. concorrente, è realizzato via SW, non direttamente via HW.
- Per i dettagli realizzativi si veda la documentazione tecnica del S.O. di interesse.

Ingresso/Uscita

- I processori IA-32 possono avere spazi di indirizzamento I / O e memoria unificati oppure separati.
- Nel primo caso, per leggere o scrivere le porte di I / O si usa l'istruzione di trasferimento MOV.
- Nel secondo caso, ci sono le istruzioni macchina di I / O privilegiate IN e OUT, con vari modi di indirizzamento disponibili (vedi prima).
- Spesso però si usa la soluzione unificata, giacché è più semplice e uniforme, e la protezione degli indirizzi di I / O da accesso non autorizzato è affidata alla MMU.