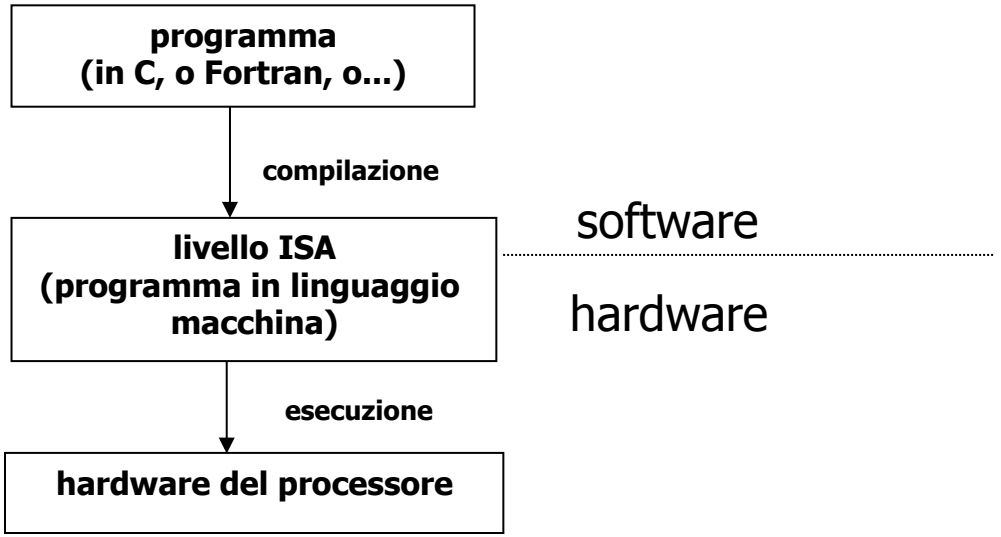
AXO – Achitettura dei Calcolatori e Sistema Operativo

Instruction Set Architecture (ISA) e 68000

introduzione a ISA

- ISA - Instruction Set Architecture
  - insieme delle istruzioni (**instruction set**) che possono essere **direttamente interpretate** dalla microarchitettura del processore
- è il livello a cui si fa riferimento quando si descrive il linguaggio macchina di un calcolatore
  - descrizione di un calcolatore (esecutore di ISA) per scrivere programmi in linguaggio macchina, cioè la descrizione del calcolatore che serve per scrivere un suo compilatore



che cosa lo ISA descrive

- lo ISA descrive
  - **gli elementi a disposizione delle istruzioni**: il modello della memoria e i registri (come sono usabili tramite le istruzioni macchina)
  - **l'insieme delle istruzioni** macchina (formati, operazioni, tipi di dati, tipi di indirizzamento)
  - le **modalità operative** (come usare lo ISA)
- lo ISA può essere fornito
  - esplicitamente (in un documento apposito)
  - implicitamente (p.es. nel "manuale assembler")

ISA e linguaggio assemblatore

- il linguaggio assemblatore è il **linguaggio *simbolico*** che consente di programmare un calcolatore utilizzando le istruzioni del linguaggio macchina (cioè sequenze di bit direttamente interpretabili dal calcolatore)
  - le istruzioni ISA (linguaggio macchina) sono in corrispondenza uno a uno con quelle del linguaggio assembly; nella descrizione di ISA si farà riferimento alla notazione simbolica delle istruzioni e delle variabili (cioè faremo riferimento al linguaggio assembly)
- per poter essere eseguito, un programma scritto in ASSEMBLER deve essere **tradotto in linguaggio macchina**, in modo tale da tradurre i codici mnemonici delle istruzioni in codici operativi, sostituire tutti i riferimenti simbolici degli indirizzi con la loro forma binaria, e riservare spazio di memoria per le variabili
  - l'operazione di traduzione viene eseguita dall'ASSEMBLATORE
  - se nel codice sorgente sono presenti riferimenti simbolici definiti in moduli esterni a quello assemblato è necessario anche il LINKER (collegatore) *[per altri programmi o librerie esterne]*

istruzioni e variabili in linguaggio macchina

- **istruzioni**: espresse in **codice macchina**, cioè nel formato direttamente interpretabile dalla CPU
- **ogni istruzione è costituita** da
  - un **codice operativo** (**OPCODE**) che la identifica in modo univoco e definisce che cosa fa l'istruzione
  - un **riferimento all'operando** (agli operandi) su cui agisce l'istruzione. Il riferimento agli operandi può essere
    - implicito nel codice operativo
    - direttamente il valore numerico dell'operando
    - un registro del processore
    - in modo diretto o indiretto, una locazione di memoria
- **variabili**: espresse in modo da essere accessibili alla CPU
  - il **riferimento** ad una variabile è rappresentato da un **indirizzo** (diretto o indiretto) della memoria di lavoro
  - il **valore** della variabile è contenuto nella parola di memoria associata all'indirizzo ed è rappresentato tramite una **codifica binaria** opportuna, dipendente dal tipo di variabile

memoria – struttura e indirizzamento

- sequenza di parole costituite da uno o più byte (8, 16, 32, … bit)
- byte indirizzabili singolarmente
- **68000: 16 M byte (24 bit di indirizzo)**
- **a partire da 68020: 4 G byte (32 bit di indirizzo)**
- contenuto della parola:
  - intero in cpl2 su 16 o 32 bit
  - carattere (2 o 4 per parola)
- enumerazione dei byte nella parola (indirizzamento per byte)
  - **big-endian:** byte numerati da sinistra a destra (CPU Sparc, ***Motorola*** ..); il byte meno significativo si trova all'indirizzo più alto, e analogamente per gli altri
  - **little-endian**: byte numerati da destra a sinistra (Intel); il byte meno significativo si trova all'indirizzo più basso, e analogamente per gli altri

memoria – indirizzamento a byte e allineamento

- con parole da 32 bit e indirizzamento della memoria per byte, parole consecutive sono collocate a indirizzi multipli di 4
- Little-Endian vs Big-Endian:
  - Supponiamo di salvare in memoria, a partire dall'indirizzo 0, la parola 0x12345678:

Little-Endian	
Indirizzo	Valore
0	78
1	56
2	34
3	12

Big-Endian	
Indirizzo	Valore
0	12
1	34
2	56
3	78

memoria – operazioni a livello ISA

- caricamento (da memoria) = trasferimento del contenuto di una parola di memoria in un registro di CPU
  - lettura da memoria
    - *load sorgente (= memoria), destinazione*
    - ***MOVE sorgente (= memoria), destinazione***
- memorizzazione (in memoria) = trasferimento del contenuto di un registro di CPU in una parola di memoria
  - scrittura in memoria
    - *store sorgente, destinazione (= memoria)*
    - ***MOVE sorgente, destinazione (= memoria)***

registri di CPU

- sono elementi di memoria interni alla CPU; per ISA interessano tutti e soli i registri **referenziabili** nelle istruzioni macchina, che sono un sottoinsieme di quelli effettivamente presenti nella CPU
- sono usati per **dati** e risultati temporanei e **informazioni di controllo**
  - il **valore massimo** memorizzabile è determinato dalle **dimensioni** del registro
- **specifici per informazioni di controllo**
  - Program Counter, Instruction Register …, Program Status Word - PSW- per i flag (esiti) dell'ALU, per il controllo di dispositivi (cache, I/O)
- **di uso generico**

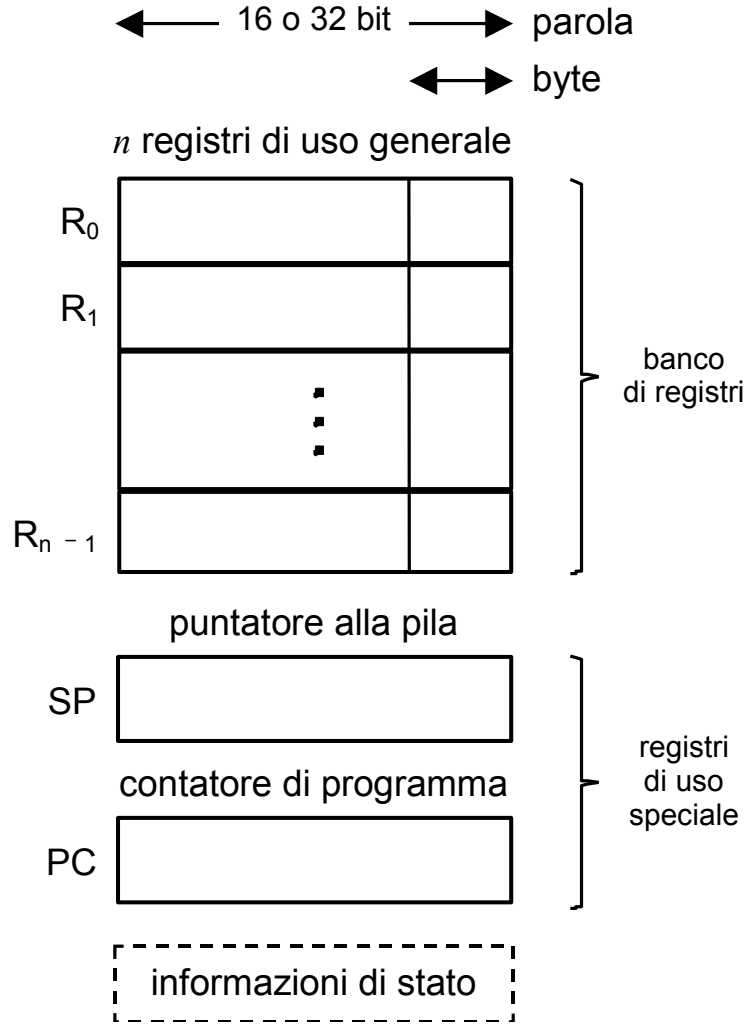
  dedicati: per i dati; per gli indirizzi: puntatori, base, spiazzamenti, stack pointer, …

  con limitazioni d'uso: solo leggibili; ad automodifica: autoincremento/ decremento, …

  preferiti per istruzioni speciali: scorrimenti, operazioni logiche, conversioni di formato, operazioni aritmetiche, …

modello di un generico processore a registri

- sono evidenziati tutti e soli i registri referenziabili da ISA
- 68000 rientra in questo modello con alcune specificità



architettura dei registri in 68000

Di (0 ≤ i ≤ 7): registri di dato a 8, 16, 32 bit

Ai (0 ≤ i ≤ 7): resgistri di indirizzo a 16, 32 bit

PC: Program Counter a 32 bit (24 bit)

SP (= A7): Stack Pointer a 16, 32 bit

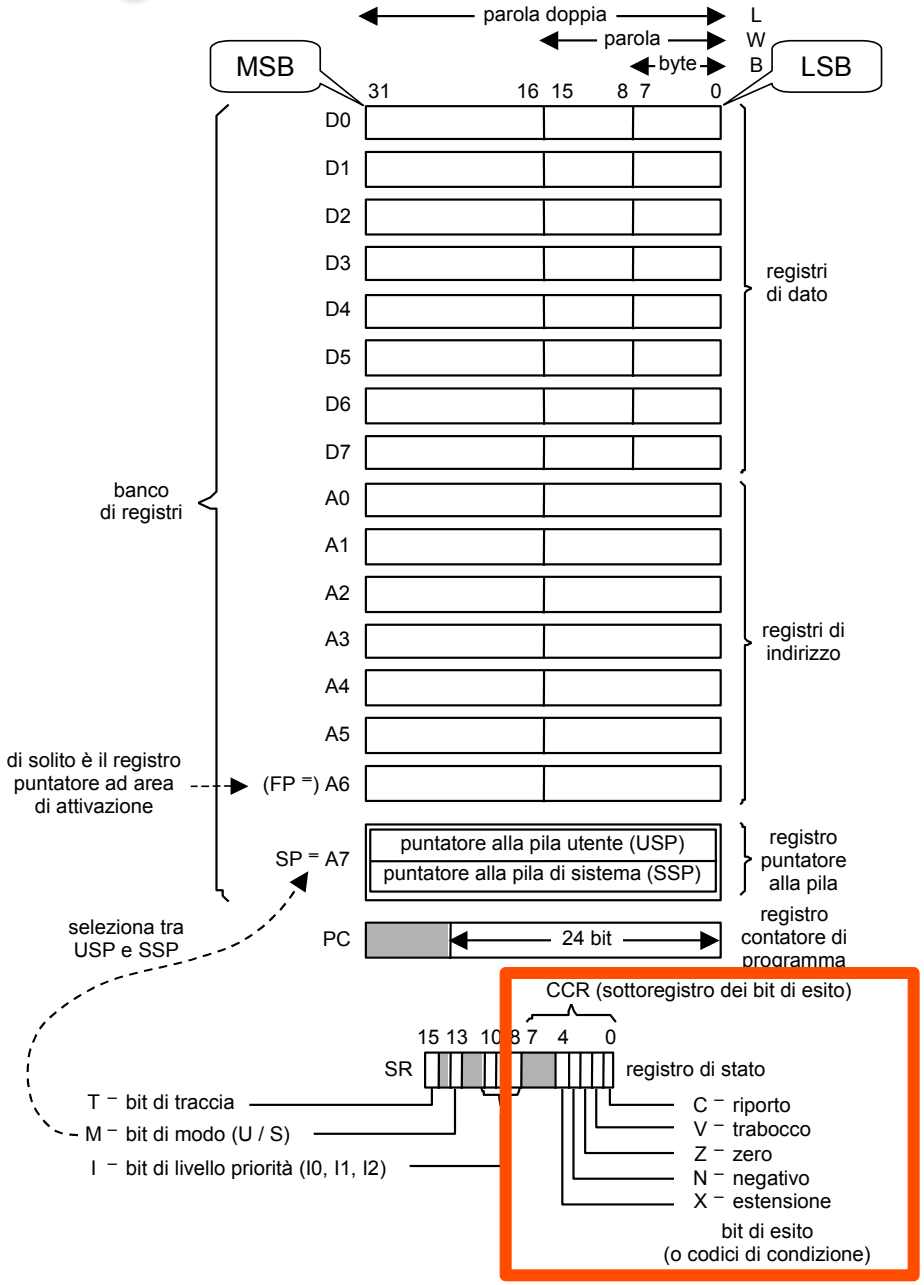
FP (di solito = A6): Frame Pointer

(puntatore all'area di attivazione) a 16, 32 bit

CCR: Condition Code Register (registro dei bit di esito - flag) a 8 bit

SR: Status Register

registro di stato (contiene CCR) a16 bit



istruzioni macchina

istruzioni – sottoinsieme di 68000

- **sottoinsieme** del linguaggio M68000 con queste caratteristiche
  - funzionalmente completo (permette di esprimere qualsiasi programma)
  - pochi codici mnemonici
  - separazione tra modi di indirizzamento di dati e di istruzioni
  - anche istruzioni privilegiate

classi di istruzioni

- **classi di istruzioni** tipiche in linguaggio macchina
  - trasferimento da e in memoria e tra registri
  - aritmetico-logiche
  - modifica del flusso di esecuzione
    - salto: incondizionato, condizionato, a sottoprogramma
  - trasferimento dati da e in periferica (istruzioni di I/O)
  - istruzioni speciali (controllo)

tipi di dato

- sono molto semplici rispetto a quelli disponibili in un linguaggio di alto livello e ovviamente solo “built-in”
- ciascuna istruzione macchina prevede di lavorare su dati di un tipo ben definito e rispettandone le caratteristiche al termine dell’operazione
- le operazioni su tipi non supportati dal linguaggio macchina devono essere svolte SW utilizzando le operazioni sui tipi supportati

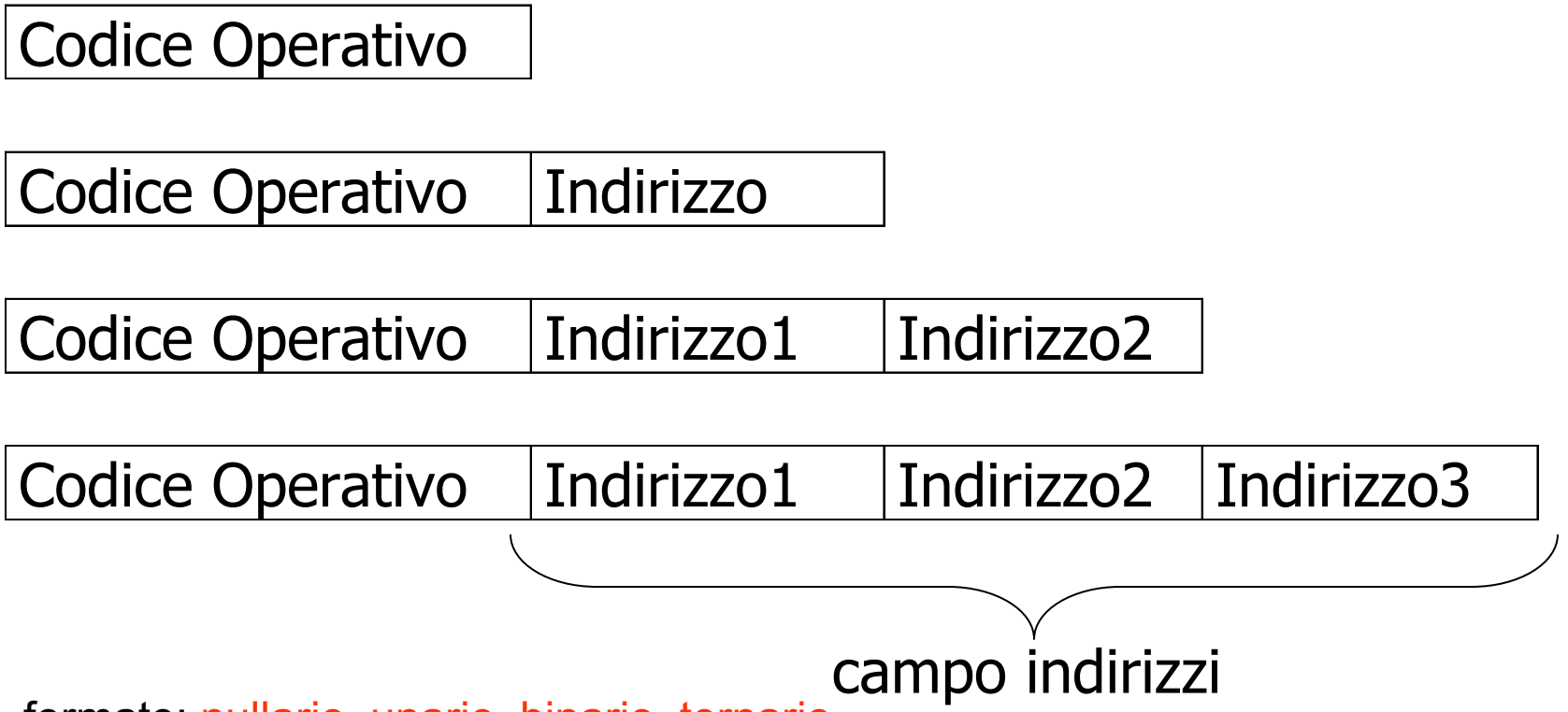
tipi di dati più comuni

- **numerici**
  - interi (con segno e senza segno) da 8, 16, 32, 64 bit
  - floating point da 32 bit (float), 64 bit (double) ma anche di formati superiori
- **non numerici**
  - caratteri ASCII (7 bit) ASCII estesi (8 bit) e UNICODE (16 bit)
  - stringhe
  - booleani, bit e mappe di bit

formato di istruzione

- definisce
  - il codice operativo dell'istruzione
  - dove l'istruzione trova i suoi operandi (indirizzo e modalità di indirizzamento, memoria o registro)
  - dove l'istruzione lascia i suoi risultati (indirizzo e modalità di indirizzamento, memoria o registro)
- inoltre dobbiamo sempre precisare eventuali "effetti collaterali" di ogni istruzione (per esempio se e come modifica i bit di esito, cioè i bit CCR del registro SR nel processore Motorola 68000)

formati di istruzione tipici



- formato: nullario, unario, binario, ternario
- se manca un indirizzo
  - o l'istruzione non ha l'operando/risultato
  - o l'operando/risultato è in una posizione implicitamente determinata

esempi di formati di istruzione per processore generico

modello semplificato	modello espanso	funzionamento in RTL	in 68000 ?
ternario (a tre argomenti)			
add s_1, s_2, d	add s_1, s_2, d	d ← [s_1] + [s_2]	non esiste
binario (a due argomenti)			
add s, d	add s_1, s_2 = d	d ← [s_1] + [s_2] (o d ← [s_1] + [d])	√
unario (a un solo argomento)			
add s	-	*acc* ← [s] + *acc*	√
nullario (a zero argomenti)			
add stack	-	addiziona i due elementi sulla cima della pila, li spila e impila la somma	non esiste

ingombro di istruzione

- una istruzione per parola, con tutte le istruzioni di uguale lunghezza
- più istruzioni per parola, con tutte le istruzioni di uguale lunghezza
- **caso più generale**: istruzioni di lunghezza diversa che possono essere contenute in più parole di memoria (è il caso del Motorola 68000)

Una parola di memoria		
Istruzione		
Istruzione	Istruzione	Istruzione
Istruzione		

massima flessibilità nella definizione delle istruzioni:

- minimo spreco di spazio per il programma in memoria
- massima complessità dell'implementazione nella microarchitettura

descrizione dell'istruzione

notazioni usate nella descrizione

- formato: codice operativo e operandi
  - operandi: sorgente (s), destinazione (d)
  - può anche essere s = d
- funzionamento dell'istruzione in RTL (Register Transfer Language): specifica l'uso degli operandi e l'effetto dell'istruzione sui bit di esito
- esempi

ADD s, d **d, esiti ← $[s_1] + [s_2]$** o meglio **s_2, esiti ← $[s_1] + [s_2]$**

CMP s_1, s_2 **esiti ← $[s_2] - [s_1]$**

modi di indirizzamento

caratteristiche di 68000: ricorda i registri

- 68000 ha 8 registri di dato (D0-D7) e 8 registri di indirizzo (A0-A7)
  - i registri sono da 32 bit (long word) ma si possono usare anche i soli 16 bit meno significativi (word) o gli 8 bit meno significativi (byte)
  - se vengono usati word o byte, i rimanenti nel registro restano invariati
- i registri di dato sono indifferenziati tra loro
- il registro A7 si usa convenzionalmente come Stack Pointer (SP)
- lo Stack Pointer punta sempre alla prima cella piena della pila e la pila cresce dal basso (indirizzi alti della memoria) verso l'alto (indirizzi bassi della memoria)
  - per inserire un nuovo dato in pila (push)
    - decremento dello SP
    - scrittura del dato a quell'indirizzo di memoria
  - per estrarre un dato dalla pila (pop)
    - lettura del dato dall'indirizzo di memoria definito dallo SP
    - incremento dello SP
- il Program Counter è di 24 bit

caratteristiche di 68000: ricorda i suffissi

- molte istruzioni 68000 e alcuni modi di indirizzamento possono avere un suffisso (.X) che indica quanti bit usare per gli operandi
- .B = byte = 8 bit, i meno significativi
- .W = word = 16 bit, i meno significativi (*default*)
- .L = long word = 32 bit
- esempio con istruzioni

```
MOVE.L      #$12345678, D1
CLR         D0
MOVE.B      D1, D0          // D0 contiene $00000078
CLR         D0
MOVE.W      D1, D0          // D0 contiene $00005678
CLR         D0
MOVE.L      D1, D0          // D0 contiene $12345678
```

codifica dell'operando

- Gli operandi, in generale più di uno, possono essere contenuti in memoria o in un registro; la loro codifica dipende:
  - dall'indirizzabilità della memoria (p. es. a byte, a parola di 2/4/8 byte)
  - dalla **modalità di indirizzamento**
- si devono indirizzare
  - dati
  - locazioni di arrivo di salti (scelta della prossima istruzione da eseguire)
- l'operando può essere specificato genericamente nel modo seguente
  - specificando direttamente il **valore** dell'operando
  - specificando un **registro** che contiene l'operando
  - specificando come ottenere **l'indirizzo effettivo** in memoria dell'operando
- prestazioni
  - spazio occupato in memoria dalla codifica del campo
  - velocità: nel reperimento dell'istruzione completa, nella determinazione effettiva dell'indirizzo degli operandi e il loro reperimento

modi di indirizzamento

- distinguiamo tra modalità di indirizzamento di dati e modalità di indirizzamento di istruzioni (destinazioni di salto) per chiarezza
- 68000 ha:
  - 8 modi indirizzamento di dati
  - 4 modi di indirizzamento di istruzioni per identificare le destinazioni di salto
    - per l'indirizzamento di istruzioni, l'indirizzo definito dalla modalità di indirizzamento viene caricato nel Program Counter
- per ciascuna modalità verrà data la sintassi Assembler e l'effetto della modalità
- le modalità di indirizzamento disponibili nel 68000 sono la (quasi) totalità delle modalità di indirizzamento disponibili in generale
- verranno evidenziate le modalità non supportate

modi di indirizzamento di dato - 1

- **di costante o immediato**
  - il valore dell'operando è specificato nell'istruzione

- sintassi assembler #costante

- significato: operando = costante

- con costante a 8, 16, 32 bit, specificata in decimale, esadecimale (prefisso $)

- esempio

```
MOVE.L    #$12345678, D1
```

modi di indirizzamento di dato - 2

- **di registro (diretto)**
  - l'istruzione specifica il registro che contiene il valore dell'operando
  - il registro può essere di dato (Di) o di indirizzo (Ai)

- sintassi assembler **Di** o **Ai**

- significato
  - operando = [Di] o [Ai]
  - risultato = Di o Ai
- esempio

```
MOVE.L      D0, D1          D1 ← [D0]
MOVE.L      A0, D1          D1 ← [A0]
```

modi di indirizzamento di dato - 3

- **indiretto da registro**
  - l'operando è in memoria
  - l'istruzione specifica quale registro di indirizzo contiene l'indirizzo dell'operando in memoria

- sintassi assembler **(Ai)**

- significato
  - indirizzo effettivo = [Ai]

- esempio

```
MOVE.L    (A0), D1          D1 ← [[A0]]
```

modi di indirizzamento di dato - 4

- **con autoincremento (indiretto da registro con autoincremento posticipato)**
  - l'operando è in memoria
  - l'istruzione specifica quale registro di indirizzo contiene l'indirizzo dell'operando in memoria
  - dopo aver utilizzato l'indirizzo, l'istruzione incrementa il suo valore di 1, 2 o 4 a seconda della dimensione dell'operando
  - Implementa l'istruzione di **PUSH** sulla pila
- sintassi assembler **(Ai)+**
- significato
  - indirizzo effettivo = [Ai]
  - e poi Ai ← [Ai] + 1 (dove 1 indica genericamente l'incremento in funzione della dimensione dell'operando)
- esempio

`MOVE.L (A0)+, D1` **`D1 ← [[A0]]`** e poi **`A0 ← [A0] + 4`**

modi di indirizzamento di dato - 5

- **con autodecremento (indiretto da registro con autodecremento anticipato)**
  - l'operando è in memoria
  - l'istruzione specifica quale registro di indirizzo contiene l'indirizzo dell'operando in memoria
  - prima di utilizzare l'indirizzo, l'istruzione decrementa il suo valore di 1, 2 o 4 a seconda della dimensione dell'operando
  - Implementa l'istruzione di **POP** dalla pila

- sintassi assembler **-(Ai)**
- significato
  - Ai ← [Ai] - 1 (dove 1 indica genericamente il decremento in funzione della dimensione dell'operando)
  - indirizzo effettivo = [Ai]
- esempio

`MOVE.L -(A0), D1` `A0 ← [A0] - 4` e poi `D1 ← [[A0]]`

modi di indirizzamento di dato - 6

- **con indice e spiazzamento (indiretto da registro con spiazzamento)**
  - l'operando è in memoria
  - l'indirizzo dell'operando in memoria è la somma di
    - un indirizzo contenuto nel registro di indirizzo (indice) specificato nell'istruzione
    - uno spiazzamento a 16 bit (dotato di segno in complemento a 2) specificato nell'istruzione

- sintassi assembler **spiazzamento_{16} (Ai)**
- significato
  - indirizzo effettivo = spiazzamento_{16} + [Ai]
- esempio

```
MOVE.L 12(A0), D1          D1 ← [12+[A0]]
```

modi di indirizzamento di dato - 7

- **con base, indice e spiazzamento (indiretto da registro con spiazzamento)**
  - l'operando è in memoria
  - l'indirizzo dell'operando in memoria è la somma di
    - un indirizzo contenuto nel registro di indirizzo (indice) specificato nell'istruzione
    - un indirizzo scalato contenuto in un registro (Di o Ai)
    - uno spiazzamento a 8 bit specificato nell'istruzione
  - lo spiazzamento è sempre a 8 bit, l'indirizzo scalato può essere a 16 o 32 bit

- sintassi assembler **$spiazzamento_8$ (Ai, Rb.Y)** con Y = W, L (16, 32 bit)
- significato
  - indirizzo effettivo = $spiazzamento_8$ + [Ai] + [Db_{16}]
  - indirizzo effettivo = $spiazzamento_8$ + [Ai] + [Ab_{16}]
  - indirizzo effettivo = $spiazzamento_8$ + [Ai] + [Db_{32}]
  - indirizzo effettivo = $spiazzamento_8$ + [Ai] + [Ab_{32}]

- esempio

```
MOVE.L 4(A0,D0.W), D1          D1 ← [4+[A0]+[D0.W]]
```

modi di indirizzamento di dato - 8

- **assoluto (o diretto)**
  - l'operando è in memoria
  - l'indirizzo dell'operando in memoria è un valore a 16 o 32 bit specificato nell'istruzione

- sintassi assembler **indirizzo.X** con X = W, L (16, 32 bit)

- significato
  - indirizzo effettivo = indirizzo_{16}
  - indirizzo effettivo = indirizzo_{32}

- esempio

```
MOVE N, D1          D1 ← [N]
```

INDIRIZZAMENTO DI DATO IN M68000		
di costante (o immediato)		
1	#costante$_8$ #costante$_{16}$ #costante$_{32}$	operando = costante$_8$ operando = costante$_{16}$ operando = costante$_{32}$
di registro		
2	Di o Ai	operando = [Di] o [Ai] risultato = Di o Ai
indiretto da registro		
3	(Ai)	i.e. = [Ai]
con autoincremento (indiretto da registro con autoincremento posticipato)		
4	(Ai) +	i.e. = [Ai] e poi Ai ← [Ai] + 1
con autodecremento (indiretto da registro con autodecremento anticipato)		
5	- (Ai)	Ai ← [Ai] - 1 e poi i.e. = [Ai]
con indice e spiazzamento		
6	spiazzamento$_{16}$ (Ai)	i.e. = spiazzamento$_{16}$ + [Ai]
con base indice e spiazzamento		
7	spiazzamento (Ai, Rb.Y) cioè: spiazzamento$_8$ (Ai, Db.W) spiazzamento$_8$ (Ai, Ab.W) spiazzamento$_8$ (Ai, Db.L) spiazzamento$_8$ (Ai, Ab.L)	con Y = W, L (16, 32 bit) i.e. = spiazzamento$_8$ + [Ai] + [Db$_{16}$] i.e. = spiazzamento$_8$ + [Ai] + [Ab$_{16}$] i.e. = spiazzamento$_8$ + [Ai] + [Db$_{32}$] i.e. = spiazzamento$_8$ + [Ai] + [Ab$_{32}$]
assoluto (o diretto)		
8	indirizzo.Y cioè: indirizzo.W indirizzo.L	con Y = W, L (16, 32 bit) i.e. = indirizzo$_{16}$ i.e. = indirizzo$_{32}$

modi di indirizzamento di istruzione - 1

- **indiretto da registro**
  - l'istruzione specifica quale registro di indirizzo contiene l'indirizzo da caricare in PC

- sintassi assembler **(Ai)**

- significato
  - PC = [Ai]

- esempio

JMP (A0) **PC ← [A0]**

modi di indirizzamento di istruzione - 2

- **relativo a contatore di programma**
  - identico a indice con spiazzamento, ma il registro utilizzato è PC
  - lo spiazzamento (intero in complemento a 2) può essere da 8 o 16 bit
  - al posto di dare lo spiazzamento numerico e indicare esplicitamente PC è possibile utilizzare un'etichetta

- sintassi assembler **spiazzamento (PC)** oppure **etichetta**
- significato
  - PC = spiazzamento + [PC]
  - PC = distanza da etichetta + [PC]

- esempio

```
JMP     -48(PC)  PC ← -48 + [PC]
JMP CONTINUA             PC ← distanza da etichetta +[PC]
```

- NOTA: Nel processore Motorola 68000, durante l'esecuzione di una istruzione, il registro PC contiene sempre l'indirizzo dell'istruzione successiva
  - L'esempio precedente si modifica come:
  - `JMP CONTINUA  PC ← distanza da etichetta +[PC_istruzione_succ_alla_jmp]`

modi di indirizzamento di istruzione - 3

- **con base relativo a contatore di programma**
  - identico a con base, indice e spiazzamento, ma il registro utilizzato (come indice) è PC
  - lo spiazzamento (intero in complemento a 2) può essere solo da 8 bit
  - al posto di dare lo spiazzamento numerico e indicare esplicitamente PC è possibile utilizzare un'**etichetta**

- sintassi assembler
  - **spiazzamento$_8$ (PC, Rb.Y)** con Y = W, L (16, 32 bit)
  - **etichetta (Rb.Y)** con Y = W, L (16, 32 bit)

- significato
  - PC = spiazzamento + [PC] + [Rb]
  - PC = distanza da etichetta + [PC] + [Rb]

- esempio

```
JMP -48(PC, A0.L)       PC ← -48 +[PC]+[A0.L]
JMP CONTINUA(A0.L)      PC ← distanza da etichetta +[PC]+[A0.L]
```

modi di indirizzamento di istruzione - 4

- **assoluto (o diretto)**

- sintassi assembler **indirizzo.X** con X = W, L (16, 32 bit)
  - dove **indirizzo.X** può essere ovviamente specificato in simbolico

- significato
  - PC = indirizzo

- esempio

```
JMP SUB1.L    PC ← SUB1.L
```

INDIRIZZAMENTO DI ISTRUZIONE IN M68000		
indiretto da registro		
1	(Ai)	PC = [Ai]
relativo a contatore di programma		
2	spiazzamento (PC) etichetta cioè: $spiazzamento_8$ (PC) $etichetta_8$ $spiazzamento_{16}$ (PC) $etichetta_{16}$	$PC = spiazzamento_8 + [PC]$ $PC = distanza\ da\ etichetta_8 + [PC]$ $PC = spiazzamento_{16} + [PC]$ $PC = distanza\ da\ etichetta_{16} + [PC]$
con base relativo a contatore di programma		
3	spiazzamento (PC, Rb.Y) etichetta (Rb) cioè: $spiazzamento_8$ (PC, Db.W) $etichetta_8$ (Db.W) $spiazzamento_8$ (PC, Ab.L) $etichetta_8$ (Ab.L)	con Y = W, L (16, 32 bit) $PC = spiazzamento_8 + [PC] + [Db_{16}]$ $PC = distanza\ da\ etichetta_8 + [PC] + [Db_{16}]$ $PC = spiazzamento_8 + [PC] + [Ab_{32}]$ $PC = distanza\ da\ etichetta_8 + [PC] + [Ab_{32}]$
assoluto (o diretto)		
4	indirizzo.Y cioè: indirizzo.W indirizzo.L	con Y = W, L (16, 32 bit) $PC = indirizzo_{16}$ $PC = indirizzo_{32}$