Istruzione di salto

`Branch_if_[R2]>0 CICLO`

- L'istruzione di salto (branch) permette di (eseguire il corpo del ciclo più volte e) tornare all'indirizzo di un'istruzione che era già stata eseguita
- L'istruzione di salto scrive un nuovo valore nel registro **PC**, e tale indirizzo è detto destinazione di salto (branch target), invece di proseguire in ordine sequenziale
- L'istruzione di salto è condizionata, ovvero il salto avviene solo se si verifica una condizione. Se la condizione è falsa si procede in modo sequenziale
- Per l'istruzione di salto suddetta, la condizione consiste nel controllare se il contenuto di `R2` è maggiore di zero
- Altre istruzioni di salto condizionale potrebbero servirsi di due registri

  `Branch_if_[R4]>[R5] CICLO`
- L'istruzione di salto si serve dell'indirizzo simbolico `CICLO`
- Dentro il ciclo, l'istruzione di lettura dalla memoria deve usare un indirizzo aggiornato ad ogni passata

Modi di indirizzamento

- Occorre avere vari modi per specificare l'indirizzo di un operando di un'istruzione per venir incontro a varie esigenze di accesso ai dati di strutture dati (liste, array, etc.)
- I vari modi con cui un'istruzione può specificare la posizione degli operandi si chiamano **modi di indirizzamento** (addressing mode)
- **Modo di registro**: l'operando o il risultato è contenuto in un registro del processore, il cui nome (che è anche l'indirizzo del registro) è dato nell'istruzione. Es.

  `Add R3, R3, R5`
- **Modo assoluto (o diretto)**: l'operando o il risultato è contenuto in una parola di memoria il cui indirizzo è dato nell'istruzione. Es.

  `Load R2, NUM1`
- In un linguaggio ad alto livello, il programma dichiara la variabile `NUM1` e il compilatore ha associato ad essa un indirizzo di memoria

Modi di indirizzamento

- **Modo immediato (o di costante)**: l'operando è dato esplicitamente nell'istruzione. Es.

  `Add R4, R6, #200`
- Si aggiunge 200 al valore contenuto nel registro `R6` e si pone il risultato in `R4`. Si usa il simbolo # per denotare la cosante numerica
- **Modo indiretto**: l'indirizzo effettivo di operando o risultato è contenuto in un registro del processore, il cui nome è dato nell'istruzione. Es.

  `Load R2, (R5)`
- Il registro `R5` agisce come puntatore alla lista, si accede a un elemento della lista usando l'indirizzo nel registro `R5`. Si indica il modo indiretto tramite le parentesi tonde

Programma per la somma di una lista di numeri

```
        Load            R2, N       Carica dimensione lista
        Clear           R3          Inizializza la somma a 0
        Move            R4, #NUM1   Carica indirizzo primo numero
CICLO:  Load            R5, (R4)    Preleva prossimo numero
        Add             R3, R3, R5  Aggiungi numero alla somma
        Add             R4, R4, #4  Incrementa puntatore a lista
        Subtract        R2, R2, #1  Decrementa contatore
        Branch_if_[R2]>0 CICLO      Salta indietro se non ha finito
        Store           R3, SOMMA   Immagazzina somma finale
```

```
        MOV     R2, #N
        MOV     R3, #0
        MOV     R4, #NUM1
CICLO   LDR     R5, [R4]
        ADD     R3, R3, R5
        ADD     R4, R4, #4
        SUB     R2, R2, #1
        CMP     R2, #0
        BNE     CICLO
        STR     R3, [R4]
```

- `N` indica la locazione di memoria del valore *n* (numero di elementi)
- `Clear` pone a zero `R3`
- `Move` inserisce un valore in un registro (non accede in memoria). `NUM1` è un numero dato nell'istruzione, e indica la locazione di memoria del primo numero della lista. `Move` usa il modo di indirizzamento immediato. `Move` potrebbe essere una pseudo istruzione, effettuata con `Add R4, R0, #NUM1` dove `R0` vale `0`
- `Load R5, (R4)` usa il modo di indirizzamento indiretto

Esempio di indirizzamento indiretto

- Un'istruzione in linguaggio C

```
A = *B
```

- dove B è un puntatore e il simbolo * è l'operatore per accessi indiretti, può essere compilata come segue

```
Load R2, B
Load R3, (R2)
Store R3, A
```


Modo con indice e spiazzamento

- Un modo di indirizzamento utile nel gestire vettori e liste è il seguente
- **Modo con indice e spiazzamento**: l'indirizzo effettivo di un operando o risultato è ottenuto addizionando un valore costante, chiamato *spiazzamento*, al contenuto di un registro, dove sia spiazzamento che registro sono dati nell'istruzione
- Il registro viene chiamato registro indice
- Si denota con X(Ri), dove X indica lo spiazzamento (costante) e Ri il nome del registro indice
- In notazione RTN, l'indirizzo effettivo EA si calcola con EA = X + [Ri]
- In linguaggio assemblativo, la costante X si può dare in forma numerica o simbolica

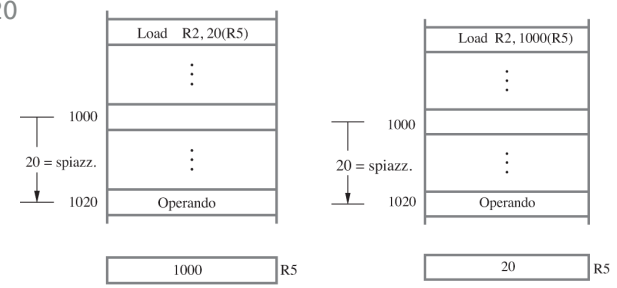
```
Load R2, 20(R5)
```

- L'istruzione sopra indica spiazzamento 20 e il registro R5 usato come registro indice, se in R5 vi è il valore 1000, si preleva il contenuto della locazione 1020

Modo con indice e spiazzamento

```
Load R2, 1000(R5)
```

- L'istruzione sopra indica spiazzamento 1000 e il registro R5 usato come registro indice, se in R5 vi è il valore 20, si preleva il contenuto della locazione 1020



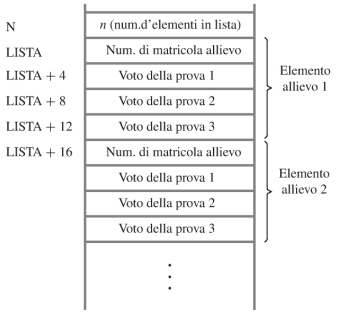
(a) Spiazzamento dato come costante

(b) Spiazzamento dato nel registro indice

Uso indirizzamento con indice e spiazzamento

- Per ciascuna di tre prove, sommare i voti degli studenti

	Move	R2, #LISTA	Carica l'indirizzo LISTA
	Clear	R3	
	Clear	R4	
	Clear	R5	
	Load	R6, N	Carica il valore *n*
CICLO:	Load	R7, 4(R2)	Aggiungi il voto della prova 1 del prossimo studente alla somma parziale
	Add	R3, R3, R7	
	Load	R7, 8(R2)	Aggiungi il voto della prova 2 dello studente alla somma parziale
	Add R4, R4, R7		
	Load	R7, 12(R2)	Aggiungi il voto della prova 3 dello studente alla somma parziale
	Add	R5, R5, R7	
	Add	R2, R2, #16	Incrementa il puntatore
	Subtract	R6, R6, #1	Decrementa il contatore
	Branch_if_[R6]>0	CICLO	Salta indietro se non ancora finito
	Store	R3, SOMMA1	Immagazzina il totale per la prova 1
	Store	R4, SOMMA2	Immagazzina il totale per la prova 2
	Store	R5, SOMMA3	Immagazzina il totale per la prova 3



Modo con base e indice

- Una variante del modo con indice e spiazzamento si ottiene usando due registri: uno che fa da base e uno che fa da indice
- Il modo con base e indice si denota con `(Ri, Rj)`
- L'indirizzo effettivo è la somma di `Ri` e `Rj`, quindi `EA = [Ri] + [Rj]`
- Quindi per questo modo di indirizzamento lo spiazzamento non è costante ma può variare

Linguaggio assemblativo (S. 2.5)

- L'istruzione macchina, memorizzata in una parola di memoria, consiste di una stringa di bit 0 e 1
- Il linguaggio macchina simbolico è più adatto ad essere letto. Le istruzioni hanno nome espressivo (`Load`, `Store`, etc.)
- L'insieme di regole che definisce sintassi e semantica di un linguaggio macchina si chiama **linguaggio assemblativo (Assembly language)**
- L'analisi di correttezza è svolta dall'assemblatore (assembler)
- Le istruzioni macchina in forma simbolica hanno la forma codice_mnemonico operandi, es.

```
Add R2, R3, #5
```


Direttive di assemblatore

- Le direttive sono informazioni accessorie che servono all'assemblatore (assembler) per tradurre il programma da codice sorgente simbolico a codice macchina
- Servono a indicare il valore di un simbolo, es. di uguaglianza (equate)

```
VENTI EQU 20
```

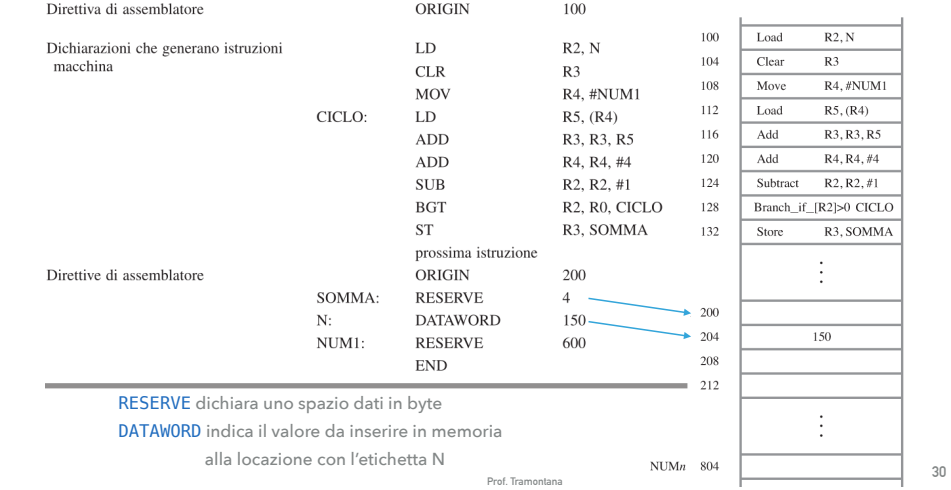
  - L'assemblatore sostituisce 20 ovunque sia presente `VENTI` nel programma
- Servono a indicare dove in memoria mettere le istruzioni
- Servono a dire dove collocare i dati (operandi e risultati)

```
ORIGIN
```


	Etichetta di indirizzo di memoria	Operazione	Indirizzi o dati
Direttiva di assemblatore		ORIGIN	100
Dichiarazioni che generano istruzioni macchina		LD	R2, N
		CLR	R3
		MOV	R4, #NUM1
	CICLO:	LD	R5, (R4)
		ADD	R3, R3, R5
		ADD	R4, R4, #4
		SUB	R2, R2, #1
		BGT	R2, R0, CICLO
		ST	R3, SOMMA
		prossima istruzione	
Direttive di assemblatore		ORIGIN	200
	SOMMA:	RESERVE	4
	N:	DATAWORD	150
	NUM1:	RESERVE	600
		END	

`RESERVE` dichiara uno spazio dati in byte

`DATAWORD` indica il valore da inserire in memoria alla locazione con l'etichetta N



Assemblaggio ed esecuzione

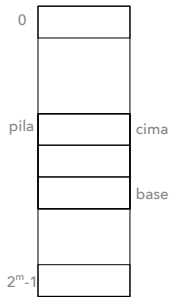
- L'assemblatore trasforma il codice sorgente in linguaggio macchina simbolico in codice oggetto
  - A ogni simbolo (etichetta, codice mnemonico, nome registro e modo di indirizzamento) sarà sostituito il valore numerico binario associato
  - Bisogna decidere a quali indirizzi associare le istruzioni e i dati, e per far questo l'assemblatore si serve delle direttive `ORIGIN`, `DATAWORD`, `RESERVE`
  - Le etichette (es. `CICLO`) vanno associate a indirizzi di memoria, per questo l'assemblatore deve tener traccia delle istruzioni presenti e dello spazio occupato da esse (una parola per istruzione per RISC)
  - Tipicamente nell'istruzione di salto va inserito lo spiazzamento rispetto all'indirizzo corrente

Assemblaggio ed esecuzione

- L'assemblatore crea un file contenente il programma oggetto
- Il programma oggetto viene caricato in memoria quando si richiede l'esecuzione. Lo fa un componente del SO chiamato loader (caricatore)
- Il loader deve sapere dove mettere in memoria il programma, e l'informazione è data dalla direttiva origin
- Quindi si inizia a eseguire dal punto specificato dalla direttiva `START`, che è inserita nel file
- Eventualmente si può eseguire in modalità debug
- Per dare agli assemblato una indicazione sulla codifica del numero immediato presente in una istruzione si usa un simbolo, per es. %01011 per numero binario, 0x come prefisso per un numero esadecimale

Gestione pila (S. 2.6)

- Una struttura dati fondamentale è la pila: lista di elementi per la quale si può inserire ed estrarre elementi solo dall'alto (cima o top)
- Quando si toglie un elemento dalla pila, dall'alto, l'elemento appena sotto diventa la nuova cima
- Solo l'ultimo elemento inserito è estraibile (**LIFO**: last-in-first-out)
- Sono disponibili solo due operazioni: **push** (impilare), ovvero porre in cima alla pila; **pop** (spilare), ovvero togliere dalla cima
- Per convenzione, si disegna la memoria in modo che le parole di memoria hanno indirizzi crescenti verso il basso
- La pila cresce per nel verso di indirizzi decrescenti



Operazione push e pop

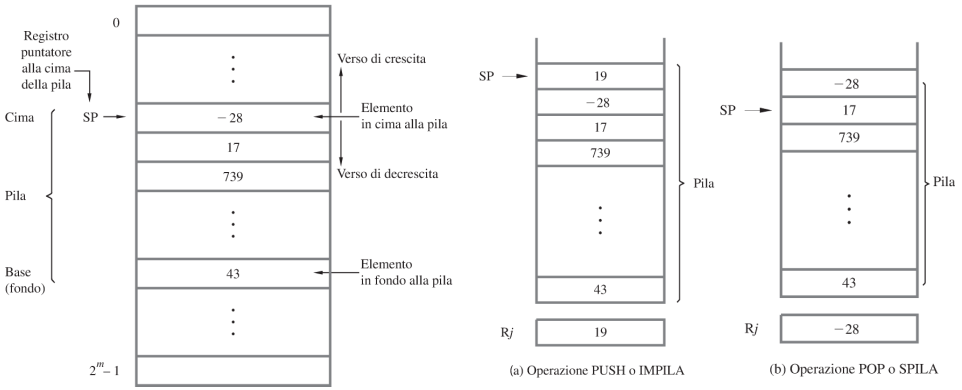
- Nel processore si trova un registro chiamato **SP** (Stack Pointer) che punta all'elemento in cima alla pila
- Con la memoria indirizzabile per byte e parola di 4 byte, l'operazione **push** (impila) del valore del registro `Rj` si ottiene tramite

```
Subtract SP, SP, #4
Store Rj, (SP)
```

- La prima istruzione aggiorna `SP`, diminuendo di 4 unità il contenuto, per farlo puntare alla parola di memoria appena sopra la cima. L'istruzione `Store` memorizza il registro `Rj` in cima alla pila
- L'operazione **pop** sarà quindi

```
Load Rj, (SP)
Add SP, SP, #4
```

Pila in memoria



Gestione di sottoprogrammi (S. 2.7)

- Un sottoprogramma permette di implementare una funzione (routine) che è richiamata (eseguita) più volte all'interno di un programma, evitando quindi di scrivere più volte le stesse istruzioni a livello macchina
- Il programma che effettua il salto al sottoprogramma è detto chiamante, mentre il sottoprogramma a cui si passa è detto chiamato
- L'istruzione di salto è detta istruzione di chiamata a sottoprogramma

`CALL SUB`

- Quando la routine termina, il programma chiamante dovrà riprendere la sua esecuzione dall'istruzione consecutiva a quella di chiamata. La routine **rientra** al programma chiamante e l'istruzione macchina è detta **istruzione di rientro** (**return** instruction)
- La routine può essere chiamata da vari punti del programma chiamante. L'indirizzo del punto di rientro è il valore del registro PC quando si esegue l'istruzione di chiamata a routine

Metodo di collegamento

- Per gestire chiamata e rientro si usa un meccanismo di **collegamento a routine**
- Occorre conservare l'indirizzo di rientro in memoria. Tipicamente si usa un **registro di collegamento** (**link register**) per questo
- Quando la routine termina, l'istruzione return usa il link register per aggiornare il PC e ritornare al punto corretto del chiamante, mediante un salto
- Quindi la chiamata a sottoprogramma è un salto speciale che
  - Copia il contenuto del PC nel registro di collegamento
  - Salta all'indirizzo di destinazione dato nell'istruzione stessa