- Programmazione a livello assembler (o di linguaggio macchina)

> Occorre conoscere l'architettura del processore
  - Come è fatta la parte operativa.
    - Collocazione e dimensione dei registri
    - Definizione e funzionamento dei flag
    - Reperimento degli operandi sorgente e destinazione
    - Possibilità operative della ALU. Rappresentazioni. Algebra.
  - Informazioni sulla parte controllo
    - Condizioni di RESET
    - Tempo di esecuzione delle istruzioni
    - Gestione delle interfacce verso l'esterno
      - Scrittura/Lettura con memoria esterna o I/O
    - Gestione di eventi asincroni (vedi progr. gerarchica)

> Occorre conoscere il SET di ISTRUZIONI

> Occorre una metodologia di programmazione e la padronanza delle tecniche più comuni di programm.
  - Progr. SEQUENZIALE (strutture per gestire il flusso)
  - Progr GERARCHICA (uso di sottoprogrammi)

> Occorre la padronanza di un AMBIENTE DI SVILUPPO
  - CAD di progettazione a basso livello
    - Editor assembler-oriented
    - Assemblatore
    - Simulatore / Debugger
  - Programmatore HW (dipende dal tipo di sistemi)
    - Gestione della ROM di programma (tipico di µC e altri oggetti "piccoli")
    - Sistemi Operativi che "accolgono" l'applicazione
  - Compilatori (cross-)

- ARCHITETTURA

> Flag tipici

Z : zero NZ
C : carry NC
OV : overflow
P : parity

> Indirizzamento

- Immediato
- Diretto
- Indiretto
- Indiretto con auto incremento (o decremento)

> Rappresentazioni numeriche

- Unsigned
- Integer

- ARCHITETTURA

- SET DI ISTRUZIONI

> Istruzioni di controllo

- Salto, salto con condizione

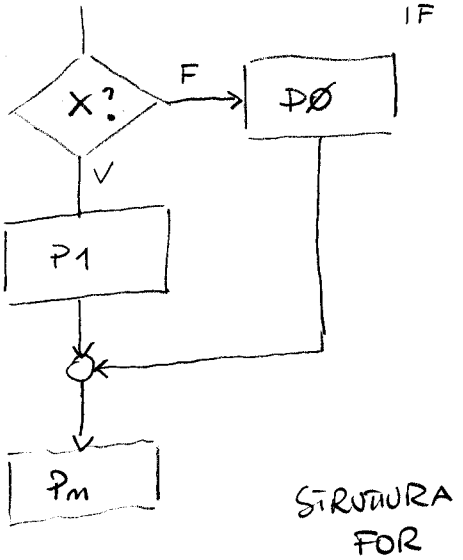
– Interrompe il normale svolgimento SEQUENZIALE, modificando il PROGRAM COUNTER. Può essere specificato
  - Nuovo valore del PC ↦ SALTO ASSOLUTO (non rilocabile)
  - Variazione ±Δ del PC ↦ SALTO RELATIVO (rilocabile)
  - Salto di 1 sola istruzione ↦ insieme al salto incondizion. permette di realizzare OGNI tipo di controllo

– Codici MNemonici tipici

JP, JR, BRANCH, SKIP, GOTO, --- ecc

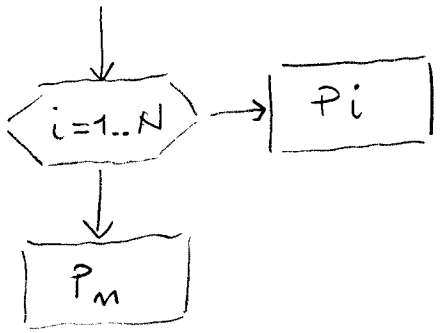
– Realizzazione di STRUTTURE di controllo (con es JP)

STRUTTURA IF



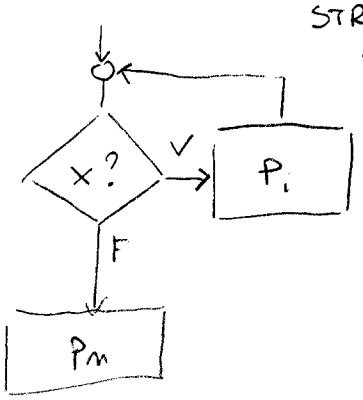

```
        TEST      ; esegui operaz. che altera FLAG
        JP FLAG, vero
falso:  PØ
        JP oltre  ; incondiz.
vero:   P1
oltre:  Pn        ; prosegue nell'es.
```

STRUTTURA FOR



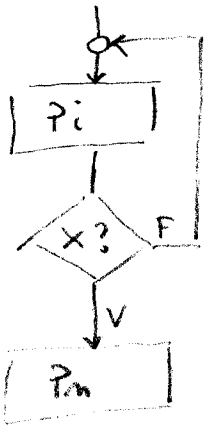

```
        LD CTR, N   ; carica contatore
coop:   Pi
        DEC CTR
        JP NON Z, coop  ; ripetere
        Pn
```

STRUTTURA WHILE




```
ciclo: TEST   ; operazione coi flag
       JP NON FLAG, oltre
       Pi
       JP ciclo
oltre: Pn
```

STRUTTURA REPEAT.. UNTIL

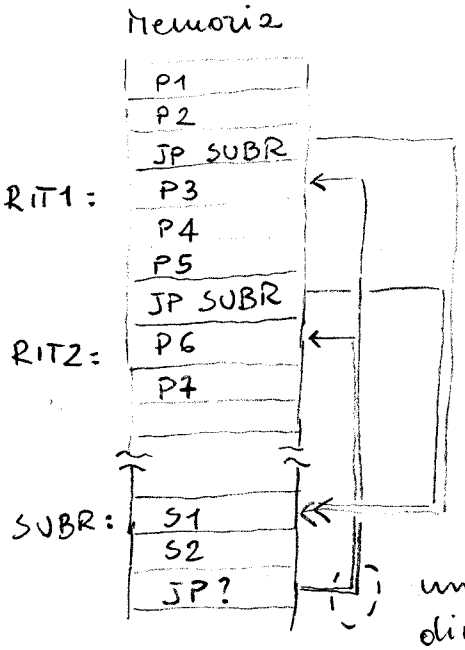



```
ciclo: Pi
       TEST
       JP NON FLAG, ciclo
       Pn
```

- Note 1: TEST è una serie di operazioni che rispondono alla domanda x? ponendo la soluzione in qualche FLAG del registro di stato
- Note 2: Praticamente tutti gli ambienti di sviluppo permettono di specificare la DESTINAZIONE del salto (Nuovo valore di PC o ΔPC) direttamente con una LABEL

– Gestione di chiamate a sottoprogrammi

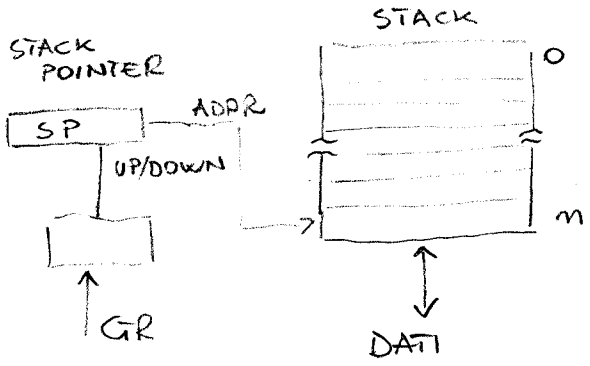
- Quando si esegue un salto, il flusso del programma è alterato e non è in generale possibile ripristinarlo
  > non è possibile chiamare sottoprogrammi con JP e ritornare al programma principale indipendentemente dalla posizione della JP di chiamata



IMPORTANTE: i sottoprogrammi devono poter essere ANNIDATI (almeno fino a un dato punto)

una JP NON può saltare verso due locazioni diverse (RIT1 e RIT2). Occorre "inventare" una struttura dati per memorizzare gli indirizzi di ritorno: es STACK

- lo stack è una struttura di tipo "LAST IN, FIRST OUT" - cioè si tratta di una memoria in cui è possibile scrivere e leggere (non contemporaneamente) con aggiornamento automatico dell'indirizzo.



Valore iniziale di SP: n

SCRIVO (PUSH): cella(SP) ← DATI
SP ← SP − 1

LEGGO (POP): SP ← SP + 1
DATI ← cella(SP)

- Può essere realizzato HW (numero di locazioni limitato) oppure SW (molto più ampio, dipende dalla dimensione della memoria dati.

> la chiamata si effettua con una istruzione particolare
- CALL (oppure SUBR) label
- salva nello stack l'indirizzo di ritorno (es: RIT1)

> la subroutine termina con una istruzione particolare
- RET
- Rimette in PC il valore prelevato dallo stack

> Istruzioni LOGICHE

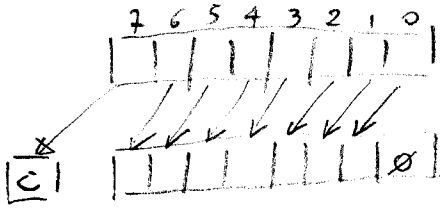
- funzionano bit a bit AND, OR, NOT, XOR (o simili)
- utili per fare MASCHERE

> Istruzioni ARITMETICHE

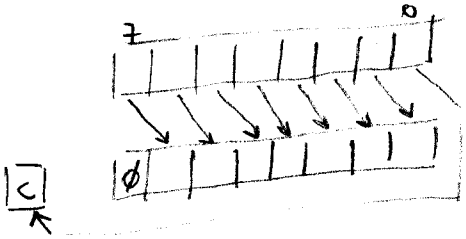
- Le più comuni sono ADD, SUB, CP, INC, DEC (o simili)
  - importante la gestione dei flag.
  - Uso del CARRY per somme MULTIPAROLA

```
ADD
ADC  → ADD with carry : propaga il riporto
ADC
 ⋮
```

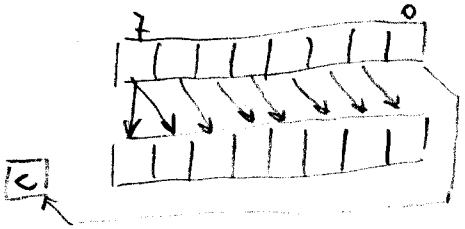
> Istruzioni di spostamento dei bit



Shift SX
equivale a MOLTIPLICARE X 2



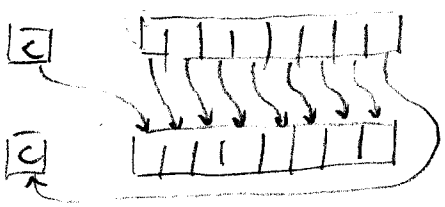
Shift DX logico
divide ÷2 un INTERO ASSOLUTO (UNSIGNED)



Shift DX aritmetico
divide ÷2 un INTERO RELATIVO

$$\frac{1}{2}\left\{-2^7 b_7 + \sum_0^6 i\ b_i 2^i\right\} = -2^6 b_7 + \sum_0^6 i\ b_i 2^{i-1} =$$

$$= \left(-2^7 + 2^6\right) b_7 + \sum_0^6 i\ b_i 2^{i-1}$$



Rotazione DX (o SX) attraverso il carry