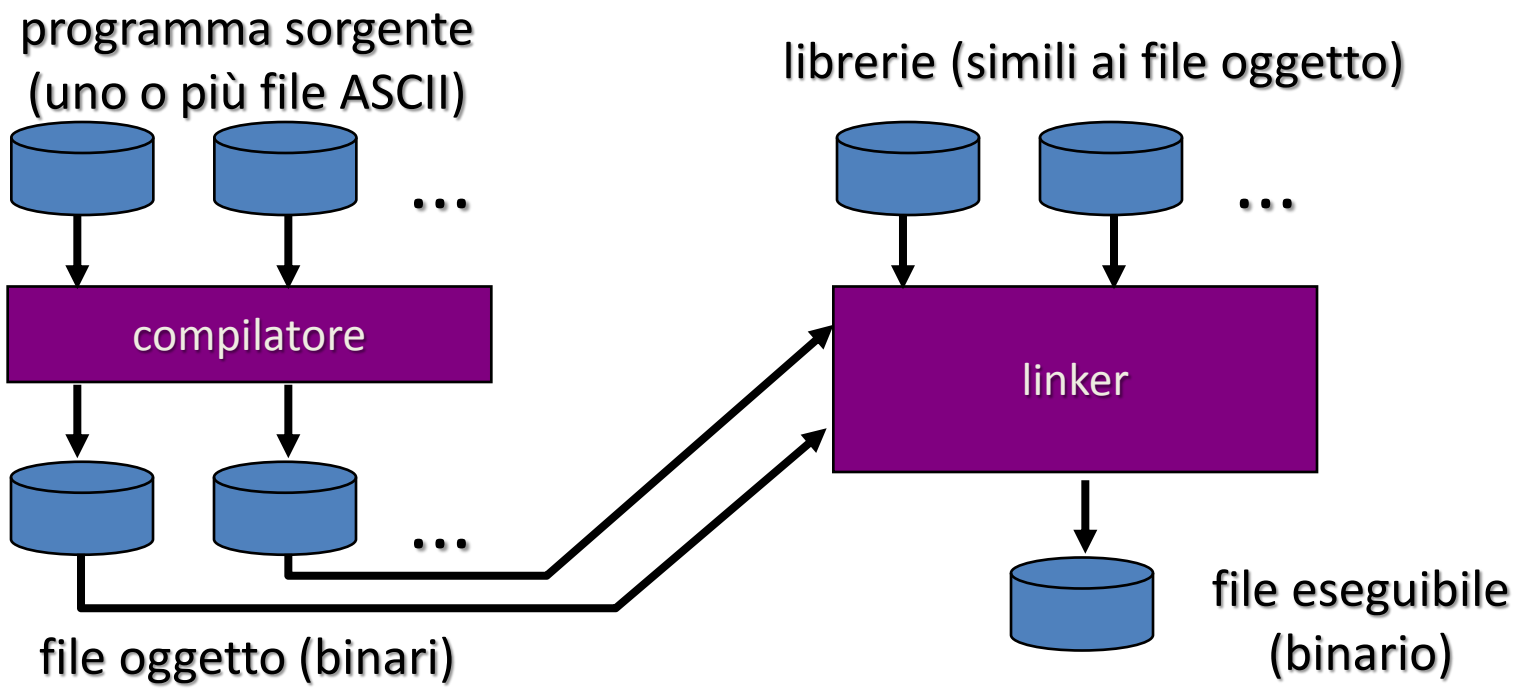
FONDAMENTI DI INFORMATICA

Prof. PIER LUCA MONTESSORO
Università degli Studi di Udine

Dal linguaggio macchina al linguaggio C

Programma sorgente, compilatore, file oggetto, file eseguibile

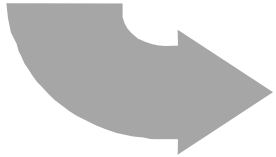


Traduzione effettuata dal compilatore

- Dichiarazione di variabili
  - spazi di memoria, di dimensioni opportune, riservati ai dati
- Espressioni
  - traduzione in sequenze di istruzioni aritmetico-logiche
- Istruzioni di controllo
  - traduzione con sequenze di istruzioni di controllo

Dichiarazione di variabili (esempi)

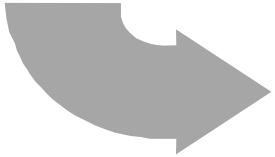
```
int n = 5;
char v[5];
```



```
n:   word 5
v:   byte 0
     byte 0
     byte 0
     byte 0
     byte 0
```

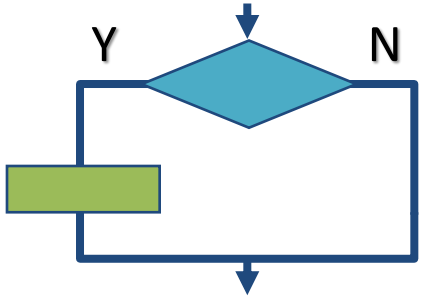
Espressioni (esempio)

```
int n, x = 12, y = 64;
n = x + y + 3;
```



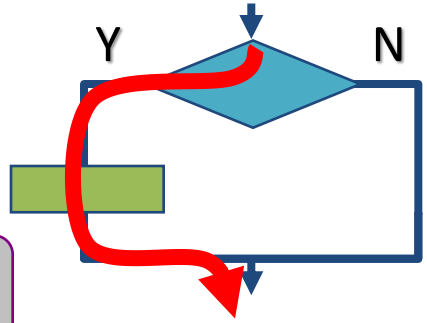
```
n:   word  0
x:   word  C
y:   word 40
     ...
     LDWI R0 3
     LDWA R1 x
     ADD R1 R0
     LDWA R1 y
     ADD R1 R0
     STWA R0 n
```

```
if <condizione>
{
    <istruzioni>
}
```




```
; l'espressione della condizione ha
; modificato il flag Z
; ipotesi: Z=0 → "vero", Z=1 → "falso"

        JMPZ cont
        ...             ; istruzioni
cont:   ...             ; seguito del programma
```

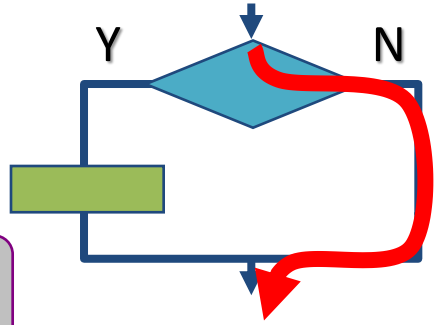
```
if <condizione>
{
    <istruzioni>
}
```



SE LA CONDIZIONE È VERA ...

```
; l'espressione della condizione ha
; modificato il flag Z
; ipotesi: Z=0 → "vero", Z=1 → "falso"

        JMPZ cont
        ...          ; istruzioni
cont:   ...          ; seguito del programma
```

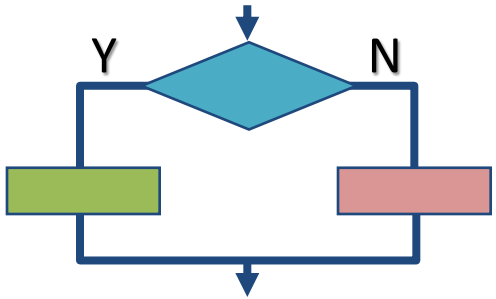
```
if <condizione>
{
    <istruzioni>
}
```



SE LA CONDIZIONE È FALSA ...

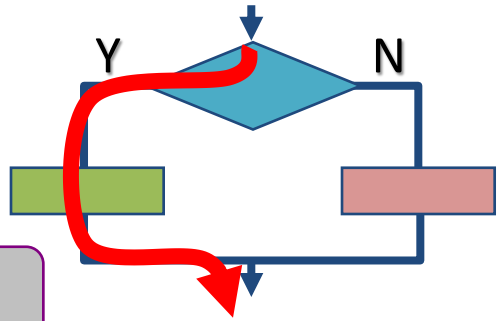
```
; l'espressione della condizione ha
; modificato il flag Z
; ipotesi: Z=0 → "vero", Z=1 → "falso"

        JMPZ cont
        ...          ; istruzioni
cont:   ...          ; seguito del programma
```

```
if <condizione> {
    <istruzioni ramo then>
} else {
    <istruzioni ramo else>
}
```




```
; l'espressione della condizione ha
; modificato il flag Z
; ipotesi: Z=0 → "vero", Z=1 → "falso"
        JMPZ else
        ...             ; istruzioni ramo "then"
        ...             ; istruzioni ramo "then"
        JMP cont
else:   ...             ; istruzioni ramo "else"
        ...             ; istruzioni ramo "else"
cont:   ...             ; seguito del programma
```

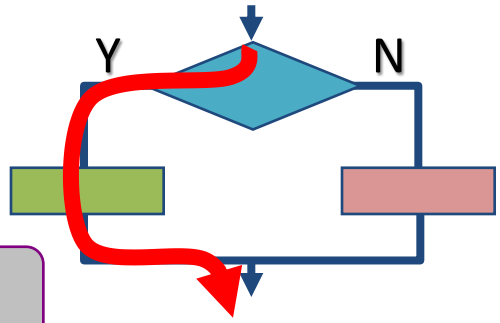
```
if <condizione> {
    <istruzioni ramo then>
} else {
    <istruzioni ramo else>
}
```



SE LA CONDIZIONE È VERA ...

```
; l'espressione della condizione ha
; modificato il flag Z
; ipotesi: Z=0 → "vero", Z=1 → "falso"
        JMPZ else
        ...             ; istruzioni ramo "then"
        ...             ; istruzioni ramo "then"
        JMP cont
else:   ...             ; istruzioni ramo "else"
        ...             ; istruzioni ramo "else"
cont:   ...             ; seguito del programma
```

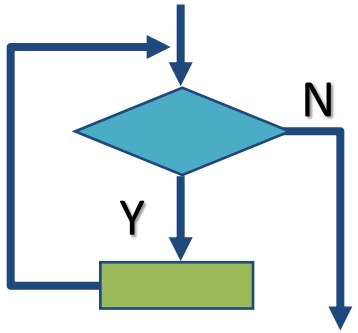
```
if <condizione> {
    <istruzioni ramo then>
} else {
    <istruzioni ramo else>
}
```



SE LA CONDIZIONE È FALSA...

```
; l'espressione della condizione ha
; modificato il flag Z
; ipotesi: Z=0 → "vero", Z=1 → "falso"
        JMPZ else
        ...             ; istruzioni ramo "then"
        ...             ; istruzioni ramo "then"
        JMP cont
else:   ...             ; istruzioni ramo "else"
        ...             ; istruzioni ramo "else"
cont:   ...             ; seguito del programma
```

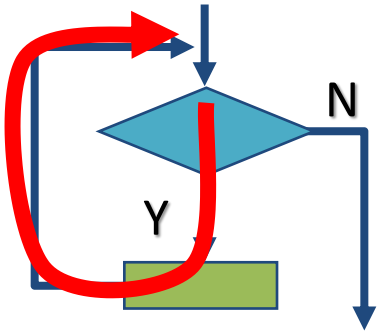
```
while <condizione>
{
    <istruzioni>
}
```




```
loop:   ...             ; valutazione della
        ...             ; condizione
; l'espressione della condizione ha
; modificato il flag Z
; ipotesi: Z=0 → "vero", Z=1 → "falso"
        JMPZ cont
        ...             ; istruzioni del ciclo
        ...             ; istruzioni del ciclo
        JMP loop
cont:   ...             ; seguito del programma
```

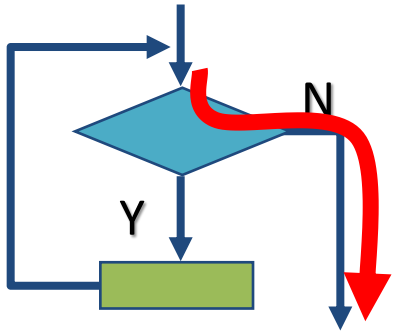
```
while <condizione>
{
    <istruzioni>
}
```

SE LA CONDIZIONE È VERA ...




```
loop:   ...             ; valutazione della
        ...             ; condizione
; l'espressione della condizione ha
; modificato il flag Z
; ipotesi: Z=0 → "vero", Z=1 → "falso"
        JMPZ cont
        ...             ; istruzioni del ciclo
        ...             ; istruzioni del ciclo
        JMP loop
cont:   ...             ; seguito del programma
```

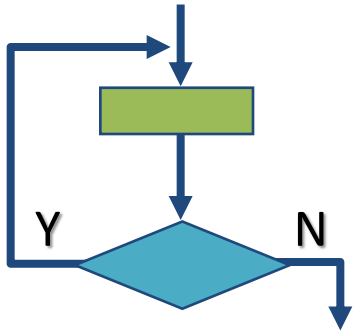
```
while <condizione>
{
    <istruzioni>
}
```



SE LA CONDIZIONE È FALSA ...

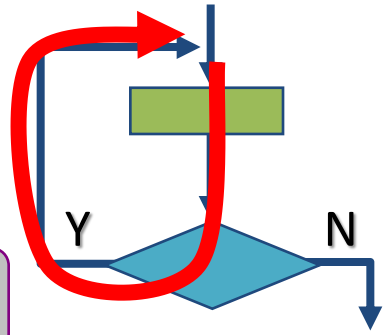
```
loop:   ...             ; valutazione della
        ...             ; condizione
; l'espressione della condizione ha
; modificato il flag Z
; ipotesi: Z=0 → "vero", Z=1 → "falso"
        JMPZ cont
        ...             ; istruzioni del ciclo
        ...             ; istruzioni del ciclo
        JMP loop
cont:   ...             ; seguito del programma
```

```
do
{
    <istruzioni>
} while <condizione>
```




```
loop:   ...             ; istruzioni del ciclo
        ...             ; istruzioni del ciclo
        ...             ; valutazione della
        ...             ; condizione
; l'espressione della condizione ha
; modificato il flag Z
; ipotesi: Z=0 → "vero", Z=1 → "falso"
        JMPNZ loop
cont:   ...             ; seguito del programma
```

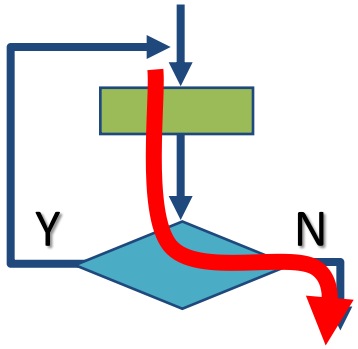
```
do
{
    <istruzioni>
} while <condizione>
```



SE LA CONDIZIONE È VERA ...

```
loop:   ...             ; istruzioni del ciclo
        ...             ; istruzioni del ciclo
        ...             ; valutazione della
        ...             ; condizione
; l'espressione della condizione ha
; modificato il flag Z
; ipotesi: Z=0 → "vero", Z=1 → "falso"
        JMPNZ loop
cont:   ...             ; seguito del programma
```

```
do
{
    <istruzioni>
} while <condizione>
```



SE LA CONDIZIONE È FALSA ...

```
loop:   ...             ; istruzioni del ciclo
        ...             ; istruzioni del ciclo
        ...             ; valutazione della
        ...             ; condizione
; l'espressione della condizione ha
; modificato il flag Z
; ipotesi: Z=0 → "vero", Z=1 → "falso"
        JMPNZ loop
cont:   ...             ; seguito del programma
```

Funzioni: chiamata

- Per gestire le chiamate di funzioni viene utilizzato lo stack di sistema:
  - push (variabili locali)
  - copia degli argomenti della funzione in registri o aree di memoria convenzionali
  - push (program counter corrente)
  - program counter ← indirizzo della funzione

Funzioni: ritorno

- copia del valore di ritorno in un registro o area di memoria convenzionale
- program counter ← pop ()
- pop (variabili locali)

- L'utilizzo dello stack anche per le variabili consente la ricorsione!

Esempio: Fibonacci

- Si scriva in linguaggio assembler una funzione recursiva che restituisca in R0 il valore del termine F_n della successione di Fibonacci il cui numero d'ordine, *n*, è passato alla funzione in R1
- Si ricorda che:

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

Esempio: Fibonacci

- Partiamo dal programma in linguaggio C

- *Nota: in questo esercizio, per semplicità, useremo soltanto variabili nei registri (cioè non in memoria centrale) e non faremo uso di funzioni di input/output*

Esempio: Fibonacci

```
int main()
{
    int R1 = 10;
    int R0 = fibo (R1);
}

int fibo (int R1)
{
    if (R1 == 0)
        return 0;

    if (R1 == 1)
        return 1;

    return fibo (R1-1)
           + fibo (R1-2);
}
```

Esempio: Fibonacci (main)

```
                        ; int main()
                        ; {
START: LDWI R10 0F000
       SPWR R10
       LDWI R1 0A       ;     int R1 = 10;
       CALL FIBO        ;     int R0 = fibo (R1);
       HLT              ; }
```

Funzione FIBO - Parte I

```
                    ; int fibo (int R1)
                    ; {
FIBO:   MV R1 R2
        JMPNZ CONT_1 ;     if (R1 == 0)
        XOR R0 R0    ;        return 0;
        RET

CONT_1: LDWI R2 1
        SUB R1 R2
        JMPNZ CONT_2 ;     if (R1 == 1)
        LDWI R0 1    ;        return 1;
        RET
```

Funzione FIBO - Parte II

```
CONT_2: DEC R1
        PUSH R1
        CALL FIBO ;     /* chiama fibo (R1-1); */
        POP R1
        MV R0 R2
        DEC R1
        PUSH R1
        PUSH R2
        CALL FIBO ;     /* chiama fibo (R1-2); */
        POP R2
        POP R1
        ADD R2 R0
        RET       ;     return fibo (R1-1)
                                + fibo (R1-2);
                  ; }
```