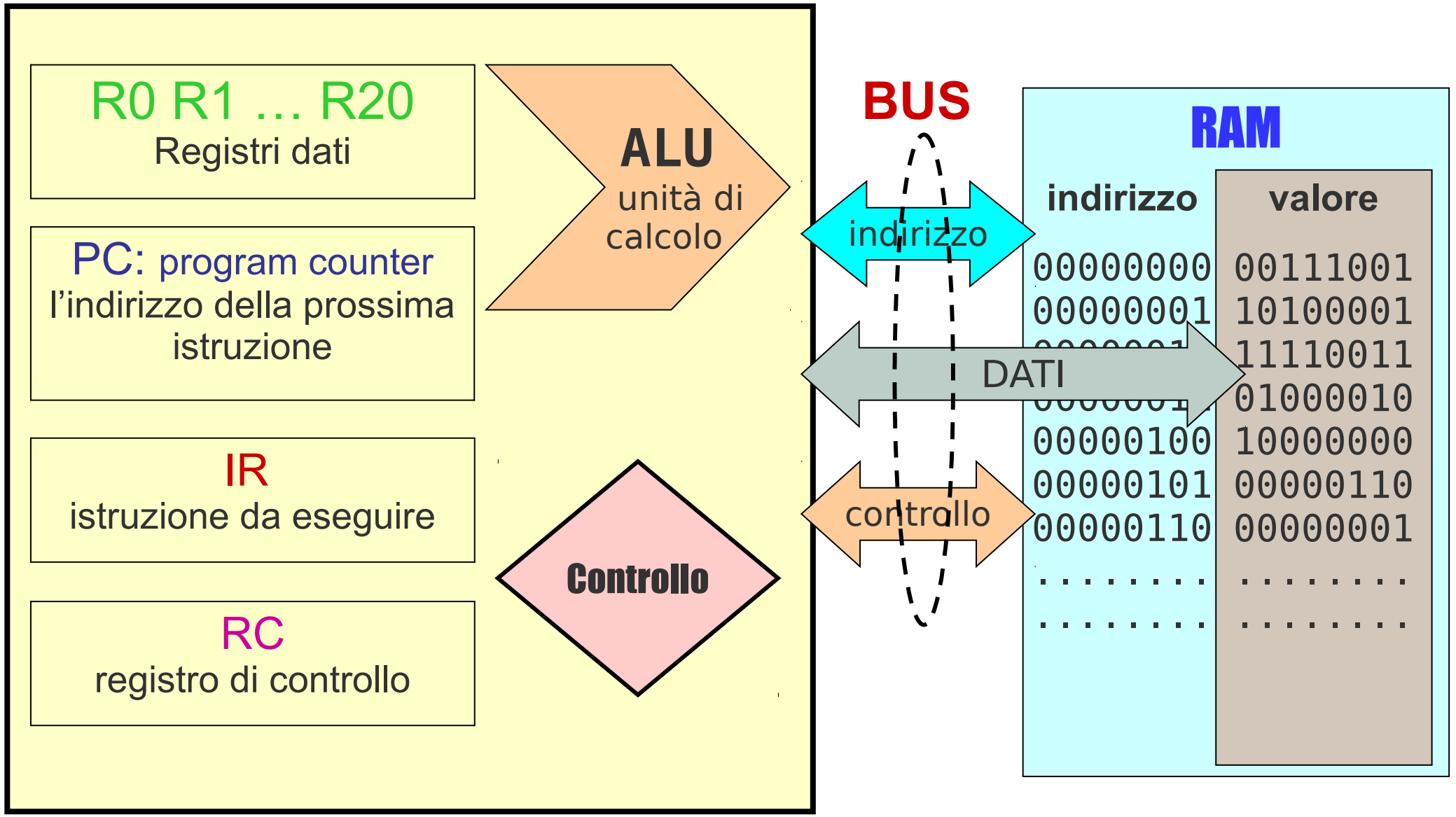
Laboratorio CPU

Ivilin Stoianov, Alessia Ceccato
www.stoianov.it

CPU


R0 R1 … R20
Registri dati
PC: program counter
l'indirizzo della prossima istruzione
IR
istruzione da eseguire
RC
registro di controllo
ALU
unità di calcolo
Controllo
BUS
indirizzo
DATI
controllo
RAM
indirizzo
valore
00000000 00111001
00000001 10100001
11110011
01000010
00000100 10000000
00000101 00000110
00000110 00000001

Un CPU virtuale: **microcalc**

- Simulatore & Assembler:
  - **istruzioni** mnemonici (**LOAD, STORE, ADD**, ...)
  - **memoria** accessibile tramite variabili: **X: INT 10**
  - istruzioni indirizzabili con eticetta: Good: **FINE**
  - Il programma scritto in un semplice file testuale:

    nano `il-mio-programma.cpu`
  - un'istruzione per riga, che finisce con **;**
  - dopo “;” segue un'opzionale **commento libero**
  - Il programma inizia con blocco definizione dati; segue una riga vuota e poi blocco istruzioni.

microcalc (2)

- Per accedere al microcalc:

1) creare una cartella **prg**

2) al interno di **prg**, creare un link simbolico al microcalc:
   ln -s /home/0/2011/stoianov/bin/microcalc

3) provare che microcalc funziona:

   ./microcalc

microcalc [-d] <nome programma>
v.2.1-20110530 v.1.0: A.Ceccato; v.2.0+: I.Stoianov

Simulatore di un CPU. Registri R0...R19. Variabili int e float.

...

microcalc (3)

- Prima di eseguire un programma, *microcalc* visualizza la sua interpretazione del programma e lo stato dei registri e delle variabili.
- Se trova errore nel programma, lo segnala e ferma l'esecuzione del programma.
- Se non ci sono errori nel programma, alla fine dell' esecuzione stampa lo stato delle variabili e dei registri.
- *microcalc* eseguito nella modalità “**debug**” (con “-d”) visualizza lo stato delle variabili e dei registri dopo l'esecuzione di ogni istruzione.

Il primo programma (**trasferimento dati)**

oBBIETIVO:

- imparare a scrivere un programma;
- imparare ad eseguire il programma;

IL COMPITO: Assegnare ad una variabile X la costante 10

- scrivere un programma in un file testuale “**1-load.cpu**”
- eseguire il programma tramite l'interpretatore “microcalc”:

  microcalc 1-load.cpu
- eseguire il programma passo per passo

  microcalc -d 1-load.cpu
-Osservare come cambiano i registri e la memoria ad ogni passo

```
X: INT;        Definire una variabile X

LOAD   R0 10; caricare la costante 10 nel Registro R0
STORE  R0 x;  Salvare il valore del R0 nella variabile X
STOP;         fine programma
```

Primi passi: **Aritmetica**

OBBIETIVI:

- imparare ad inizializzare variabili con valori
- imparare ad usare registri per fare calcoli

Compito: X=X+Costante (X=X+1) **2-add**

- definire una variabile ed assegnarne un valore iniziale (X=10)
- caricare la variabile **X** in un registro **R0**
- aggiungere al registro la costante **1**
- trasferire il risultato (rimasto nel registro) nella variabile **X**

```
X: INT 10;     Definire una variabile tipo int; assegnare valore 10

LOAD R0 X;     R0=X
ADD R0 1;      R0=R0+1
STORE R0 X;    X=R0
STOP;          FINE del Programma
```

Primi passi: **Aritmetica con 2 variabili**

OBBIETIVO:

- imparare ad eseguire calcoli con due registri dati

Compito: X=X+Y **3-add**

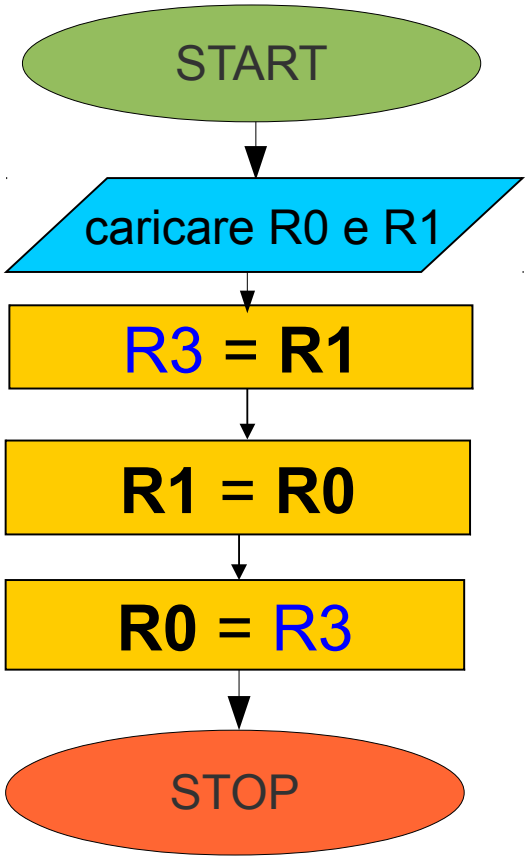
- definire due variabili, X e Y, e darne valori iniziali
- caricare X in registro R0 e Y in registro R1
- aggiungere R1 ad R0
- copiare R1 in X

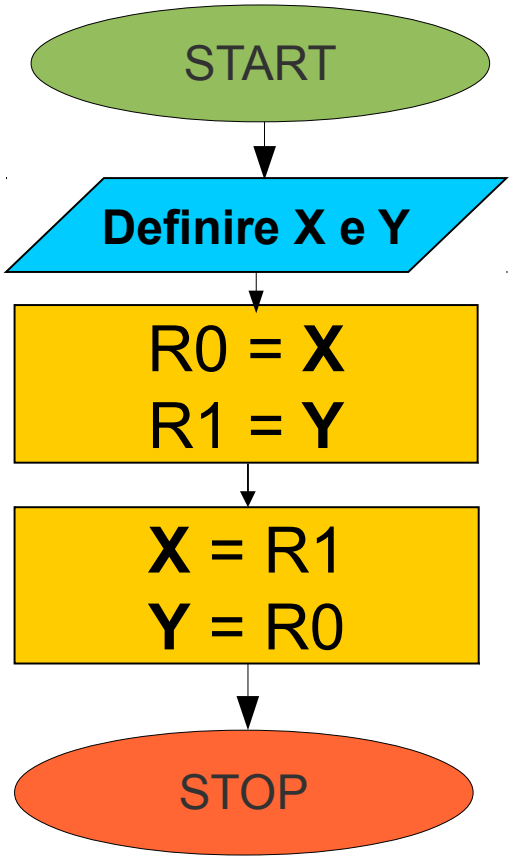
```
X: INT 10;     X=10
Y: INT  2;     Y=2

LOAD R0 X;     R0=X
LOAD R1 Y;     R1=Y
ADD R0 R1;     R0=R0+R1
STORE R0 X;    X=R0
STOP;          FINE del Programma
```

Esecuzione non-condizionale

ESERCIZIO
Scambiare il contenuto di:
(a) due registri R0 e R1
(b) due variabili X e Y





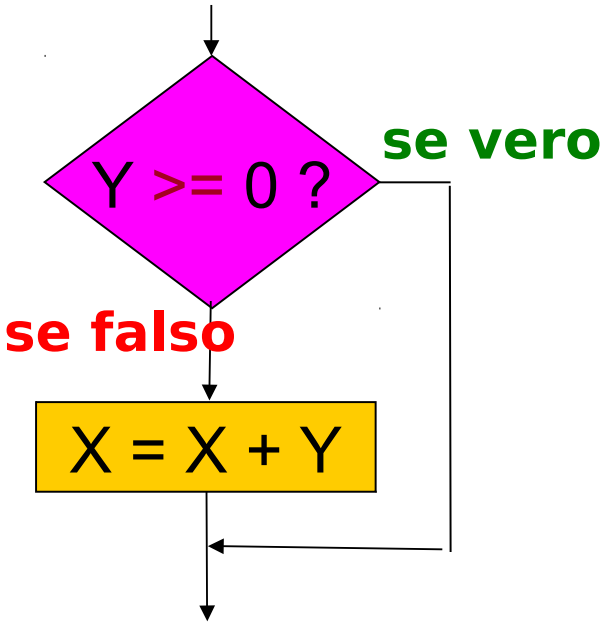
COMPITO

Scrivere due programmi
in Assembler
per la nostro CPU
che realizzano gli algoritmi

La logica “complimentare”: Saltare un blocco di elaborazioni

Per implementare l'esecuzione di un blocco di elaborazioni solo se “A” è vera, allora saltarlo se la condizione complimentare (not A) è vera.

Esempio:
Aggiungere Y al X
solo se Y<0



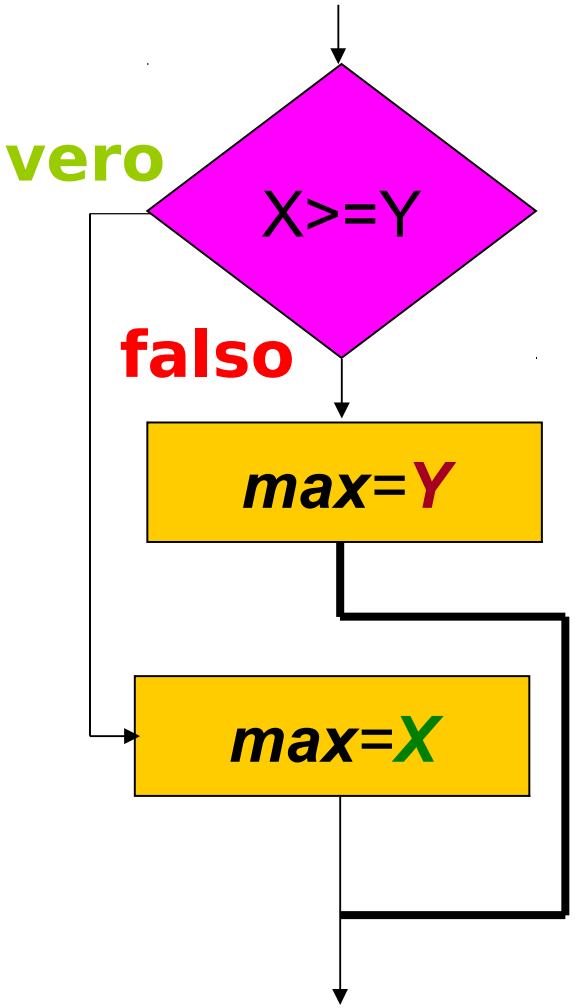

```
X: INT 5;
Y: INT -3;

LOAD R0 X;
LOAD R1 Y;
COMP R1 0;      Y>=0 ?
BRGE  Fine;     YES? → FINE
ADD R0 R1;      Acc=Acc+X
STORE R0 X;     Y+X → X
FINE: STOP;
```

6-somma-condizionale

Esempio: scegliere uno dei due numeri

- **Compito:** **assegnare alla variabile Z il massimo dei due numeri X e Y**



Non - eseguire questo codice utilizzando salto

non-condizionale

```
X: INT 5;
Y: INT -3;
Z: INT;

LOAD R0 X;
LOAD R1 Y;
COMP R0 R1;
BRGE  Big0;   Se R0>R1
STORE R1 Z;   R1>R0=>Z=R1
BRANCH FINE;
Big0: STORE R0 Z;   R0>R1=>Z=R0
FINE: STOP
```

Elaborazione condizionale complessa

OBBIETIVO: imparare a gestire **elaborazioni condizionali complesse**

Compito: Implementare la funzione **sign(X)** **8-sign**

```
X:INT 10;
Y:INT;

LOAD R0 X;
COMP R0 0;
BREQ Store;     X=0 => Y= 0
BRGT Pos;       X>0 => Y=+1
LOAD R0 -1;     X<0 => Y=-1
BRANCH Store;
Pos: LOAD R0 1;
Store: STORE R0 Y;
STOP;
```

Elaborazione condizionale (3)

OBBIETIVO: calcolo condizionale complesso

Compito: X=X+|Y| **9-somma-abs**

Proposta: Scomporre il problemma: (i) AssY=|Y| (ii) X=X+AssY

```
X: INT 10;
Y: INT -5;

LOAD R0 X;
LOAD R1 Y;
COMP R1 0;
BRGE Add;
MUL R1 -1;        Y<0 => R1=Y*(-1)
Add:ADD R0 R1;    R0=X+|Y|
STORE R0 X;
STOP;
```