# CPU e programmazione (Parte 1)

## CPU

- La CPU (Central Processing Unit) e` in grado di eseguire dei **programmi**, cioe` sequenze di istruzioni elementari ("**istruzioni macchina**")
- Idea fondamentale dell'architettura di Von Neumann: programmi e dati risiedono entrambi in memoria RAM
- Per poter essere eseguiti i programmi devono risiedere nella RAM, e quindi sono codificati digitalmente



## Elementi della CPU

- Central Processing Unit, processore
  - **Unita' logica/aritmetica**: **elaborazione** dati
  - **Unita' di controllo**: **coordina** le attivita'
  - **Registri**: **memoria temporanea**, simili a celle di memoria principale
    - ***Generici***: per gli operandi di un'operazione logica/aritmetica, e il risultato
    - ***Speciali***: per operazioni particolari



## ALU e Registri della CPU

- L'ALU e' l'unità aritmetico-logica (ALU e' un acronimo dall'inglese) che **esegue** le istruzioni e **gestisce i registri** della CPU
- I registri servono per **memorizzare** gli operandi per le istruzioni di calcolo dell'ALU
- Registri particolari
  - **PC** (program counter): contiene l'**indirizzo** RAM **della prossima istruzione** da eseguire
  - **IR** (instruction register): contiene l'**istruzione** da eseguire

## 3 tipi di istruzioni macchina

1) **trasferimento** tra RAM e registri di calcolo della CPU
2) **operazioni aritmetiche**: somma, differenza, moltiplicazione e divisione
3) **operazioni di controllo:** confronto, salto e stop

## Esecuzione delle istruzioni macchina

- Le istruzioni vengono eseguite una di seguito all'altra nell'ordine in cui esse sono memorizzate nella RAM
- Il terzo gruppo di istruzioni (confronto e salto) permette di modificare questo ordine.

## Istruzioni per trasferimento dati

- In realta', non e' un trasferimento ma una copia
- Load: da memoria a registro
- Store: da registro a memoria
- Anche trasferimento tra la memoria RAM e le unita' input/output

## Istruzioni logico/aritmetiche

- Operazioni aritmetiche: somma, sottrazione…
- Operazioni logiche: and, or, xor …

# Istruzioni di controllo

- Regolano l'esecuzione del programma
- Es.: stop
- Anche istruzioni di salto: se l'istruzione da eseguire non e' la successiva nella lista
- Salto condizionato o no
  - Es.: salta al passo 5, o salta al passo 5 se il valore ottenuto e' 0

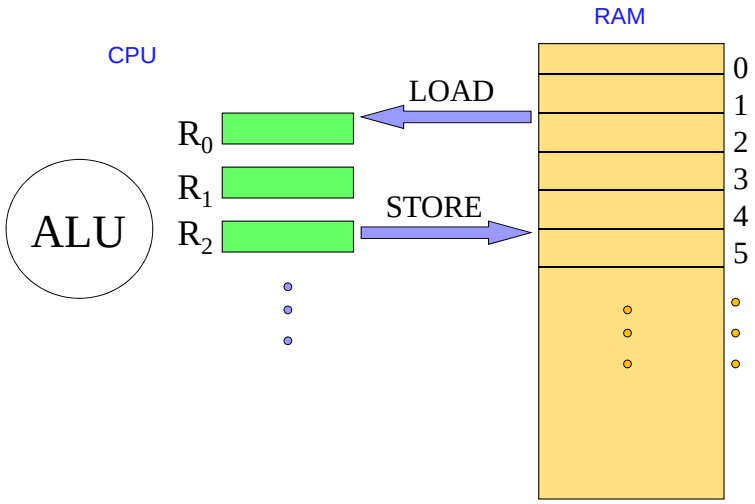
# Divisione di due valori in memoria

1. **Carica** in un registro un valore in memoria (LOAD)
2. **Carica** in un altro registro un altro valore in memoria (LOAD)
3. **Se** questo secondo valore e' 0, **salta** al passo 6 (salto condizionato)
4. **Dividi** il contenuto del primo registro per quello del secondo registro e metti il risultato in un terzo registro (op. aritmetica)
5. **Archivia** il contenuto del terzo registro in memoria (STORE)
6. STOP

# Istruzione macchina

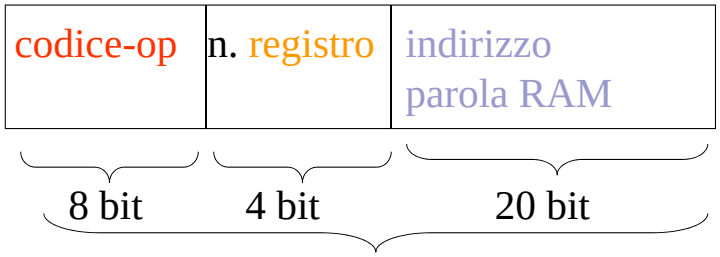
- Ogni istruzione macchina viene memorizzata in una parola di memoria (32 bit)
- Due parti (campi):
  - Campo codice operativo: quale operazione eseguire (8bit)
  - Campo operando: diverso a seconda dell'operazione (24 bit)

# Istruzioni di trasferimento registri ⇔ RAM



## Istruzioni di trasferimento in linguaggio macchina

in binario!



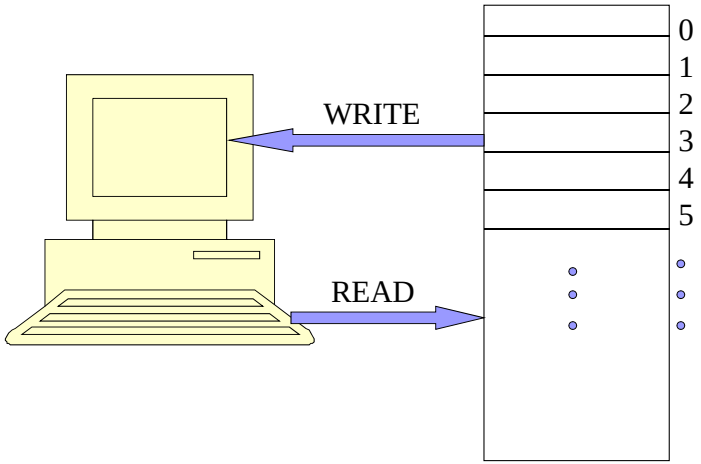

Codici:

```
LOAD     00000000
STORE    00000001
```

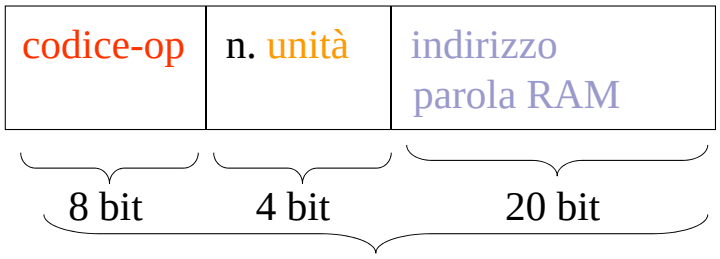
Esempio:

```
00000000 0000 00000000000000000010
00000001 0010 00000000000000000101
```

## Istruzioni di input/output:
## unità I/O ⇔ RAM




## Istruzioni input/output in linguaggio macchina




Codici:

```
READ     00010000
WRITE    00010001
```

Unità:

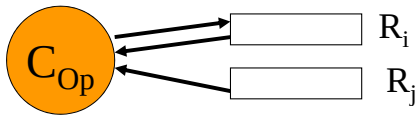
```
INP    0000 (tastiera)
OUT    0001 (video)
```

Esempio:

```
00010000 0000 00000000000000000010
00010001 0001 00000000000000000101
```

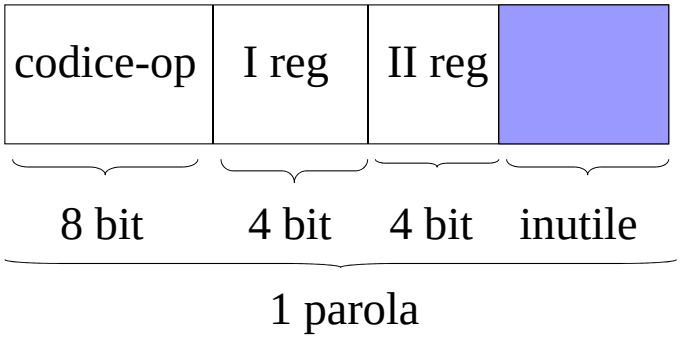
## ISTRUZIONI ARITMETICHE

Eseguono somma, differenza, moltiplicazione e divisione usando i registri come operandi




```
ADD    00000010      FADD    00000011
SUB    00000100      FSUB    00000101
MUL    00000110      FMUL    00000111
DIV    00001000      FDIV    00001001
MOD    00001010
```

## Istruzioni aritmetiche in linguaggio macchina




Esempio:

```
00000010 0011 0001 xxxxxxxxxxxxxxxx
ADD      R3   R1
```

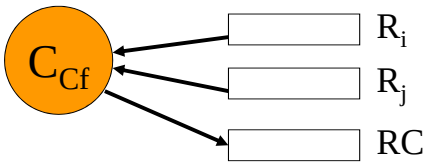
Notare che il **risultato** dell'operazione aritmetica viene messo nel **primo registro**

## Istruzione di confronto

Paragona il contenuto di 2 registri $R_i$ ed $R_j$ e:

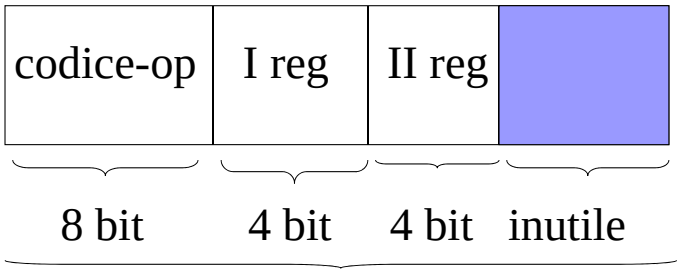
- se $R_i < R_j$ mette -1 nel registro RC
- se $R_i = R_j$ mette 0 in RC
- se $R_i > R_j$ mette 1 in RC

Registro di confronto




Codici: `COMP 00100000`

## Istruzione di confronto in linguaggio macchina:

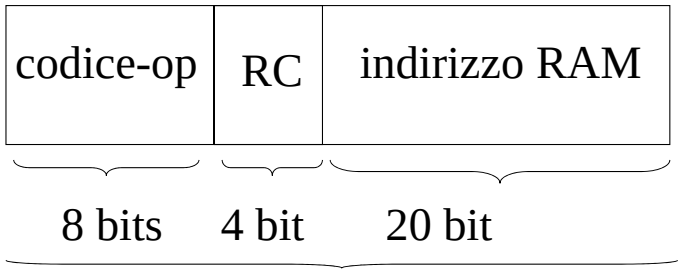



Codici: `COMP 00100000`

Esempio:

```
00100000 0010 0101 xxxxxxxxxxxxxxxx
```

## Istruzioni di salto in linguaggio macchina




Se il contenuto di RC soddisfa la condizione di salto aggiorna il PC all'indirizzo RAM indicato, altrimenti incrementa il PC di 1

Esempio:

```
01000001 xxxx 00000000000000001001
 BRLT     RC        RAM Word 9
10000000 xxxx 00000000000000001010
 BRANCH   (RC)      RAM Word 10
```

## Istruzioni di salto in linguaggio macchina

| | | | |
|---|---|---|---|
| BRLT | 01000001 | BRNE | 01000100 |
| BRLE | 01000010 | BRGE | 01000110 |
| BREQ | 01000011 | BRGT | 01000101 |
| | BRANCH | 10000000 | |

## Istruzioni di salto in linguaggio macchina (1)

- BRLT J: se RC=-1 pone PC=J altrimenti incrementa PC di 1
- BRLE J: se RC<1 pone PC=J altrimenti incrementa PC di 1
- BREQ J: se RC=0 pone PC=J altrimenti incrementa PC di 1

PC=program counter

## Istruzioni di salto in linguaggio macchina (2)

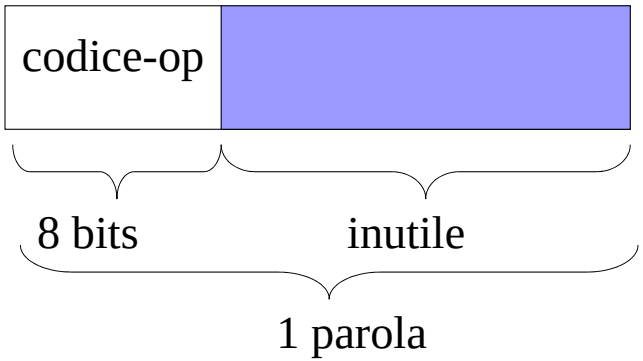
- BRNE J: se RC≠0 pone PC=J altrimenti incrementa PC di 1
- BRGT J: se RC=1 pone P=J altrimenti incrementa PC di 1
- BRGE J: se RC>-1 pone PC=J altrimenti incrementa P di 1

## Istruzione di STOP

termina il programma

Codice: **STOP 10000001**

## Istruzione di STOP in linguaggio macchina




Esempio: **10000001** xxxxxxxxxxxxxxxxxxxxxxx

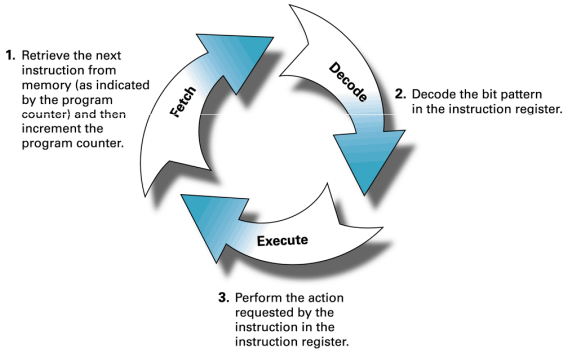
## Ciclo della CPU

- La CPU esegue un programma memorizzato in RAM copiando ad una ad una le istruzioni nell'Instruction Register dell'unita' di controllo
- **Ordine**: quello in cui sono memorizzate, a meno di istruzioni di salto
- **Registri speciali:**
  - contatore di programma (program counter)
  - Istruzione corrente (instruction register)

## Ciclo della CPU

- **Reperimento** dell'istruzione:
  - lettura della cella di RAM il cui indirizzo e' contenuto nel contatore di programma
  - caricamento del registro istruzione con l'istruzione
  - Incremento del contatore programma
- **Decodifica** dell'istruzione:
  - Trova gli operandi a seconda del codice operativo
  - Modifica contatore programma se istruzione di salto
- **Esecuzione** dell'istruzione:
  - Attiva i circuiti necessari

## Ciclo della CPU



# Esempio

Scriviamo un **programma in linguaggio macchina** che fa le operazioni seguenti:

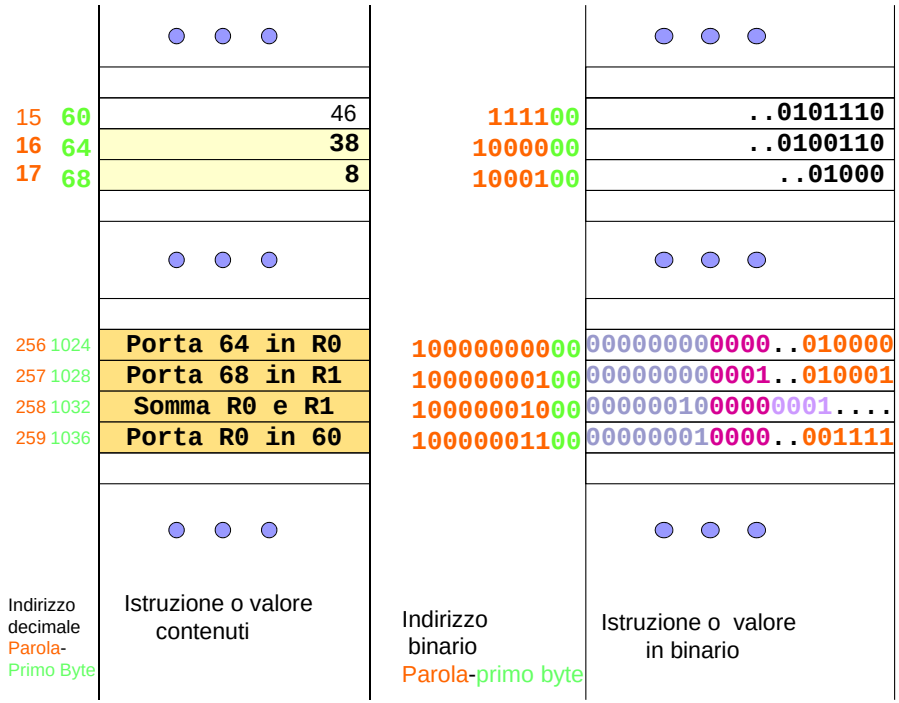
- **trasferisce** gli interi contenuti in 2 parole aventi come indirizzo del loro primo byte **64** e **68** della RAM → **nei registri $R_0$ ed $R_1$**
- **li somma**
- **trasferisce** la somma nella parola di indirizzo **60** della RAM

# Codici delle operazioni

- LOAD RAM → CPU: **00000000**
- STORE CPU → RAM: **00000001**
- ADD : **00000010**

# Numerazione binaria degli indirizzi

| Indirizzo parola | Indirizzo byte (decimale) | Indirizzo byte (binario) |
|---|---|---|
| 0 | 0 | 000000 00 |
| | 1 | 000000 01 |
| | 2 | 000000 10 |
| | 3 | 000000 11 |
| 1 | 4 | 000001 00 |
| | 5 | 000001 01 |
| | 6 | 000001 10 |
| | 7 | 000001 11 |
| 2 | 8 | 000010 00 |
| | 9 | 000010 01 |
| | 10 | 000010 10 |
| | 11 | 000010 11 |



## Svantaggi del linguaggio macchina

- programmi in binario sono difficili da scrivere, capire e cambiare
- il programmatore deve occuparsi di gestire la RAM: difficile ed inefficiente

primo passo → Assembler

## Novità dell'Assembler

- codici mnemonici per le operazioni
- nomi mnemonici (identificatori) al posto degli indirizzi RAM per i dati
- nomi mnemonici (etichette) al posto degli indirizzi RAM delle istruzioni usate nei salti
- tipi dei dati **INT** e **FLOAT**

## Codice-op mnemonici (1)

- **Trasferimento**
  - **LOAD** (RAM → CPU)
  - **STORE** (CPU → RAM)
- **Aritmetiche**
  - **ADD, FADD**
  - **SUB, FSUB**
  - **DIV, FDIV**
  - **MUL, FMUL**
  - **MOD, FMOD**
- **input/output:**
  - **READ** (INP → RAM)
  - **WRITE** (RAM → OUT)

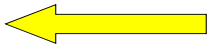
## Codice-op mnemonici (2)

- **Test**
  - **COMP**
  - **FCOMP**
- **Salto**
  - **BREQ**
  - **BRGT**
  - **BRLT**
  - **BRGE**
  - **BRLE**
  - **BRANCH**
- **Terminazione**
  - **STOP**

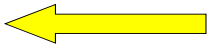
# Stesso esempio del linguaggio macchina

```
Z: INT;
X: INT 38;
Y: INT 8;

LOAD R0 X;
LOAD R1 Y;
ADD R0 R1;
STORE R0 Z;
```



dichiarazioni degli **identificatori dei dati**



istruzioni assembler