

Z80:  
architettura ed esperienze di  
laboratorio

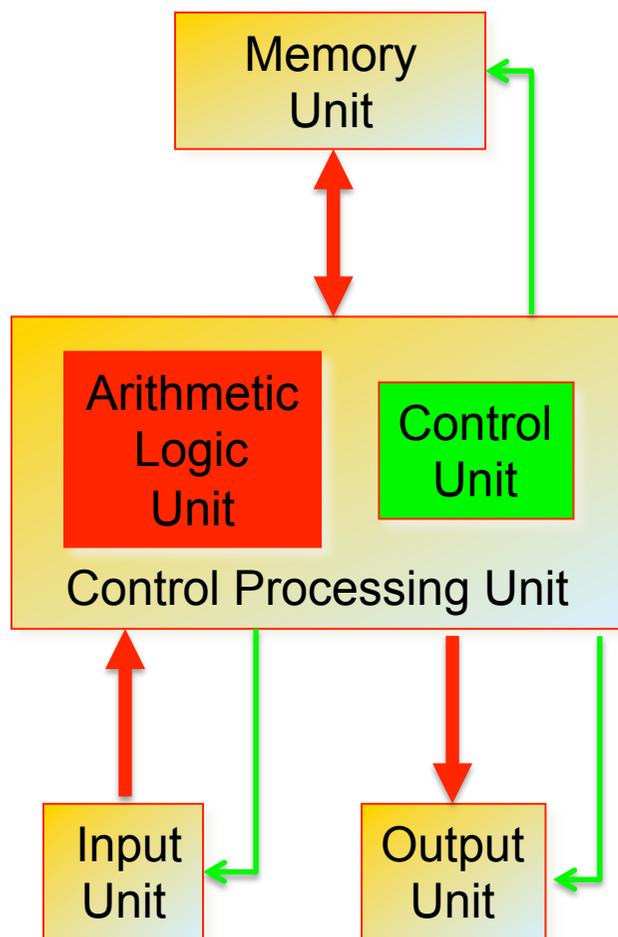
P. Vicini

# Calcolatori

- Prodotto di una tecnologia estremamente vitale con alto impatto economico e sociale
- Tecnologia pervasiva: calcolo, controllo,.....
- ... che rende possibili nuove applicazioni
  - Calcolatori nelle automobili
  - Telefoni cellulari
  - Mappatura del genoma umana
  - Imaging medico
  - WorldWideWeb e motori di ricerca
  - Approccio alla risoluzione di problemi di fisica (biologia, chimica, geologia,...) tramite simulazioni al computer
- .....



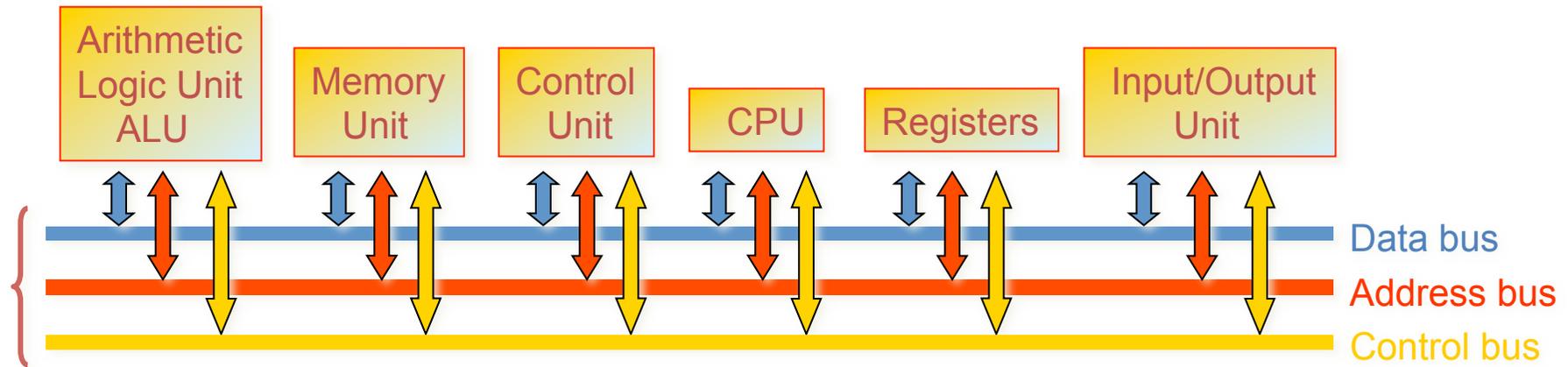
# Architettura di un calcolatore



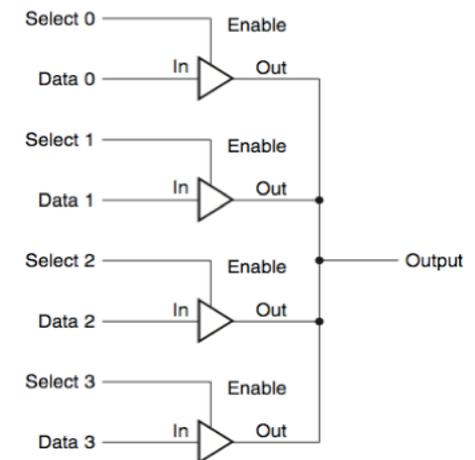
Modello di John von Neumann

- Un *calcolatore* si distingue da una macchina calcolatrice perche' e' *programmabile* i.e. la sua funzionalita' dipende da un *codice esterno* e non dalla configurazione del sistema
  - Il sommatore basato su AmpOp e' un calcolatore?
- Il suo hardware e' in grado di eseguire diversi compiti eseguendo la sequenza di *istruzioni* contenute in un *programma*.
- Secondo il modello di *Von Neumann* un calcolatore deve essere composto da:
  - *CPU* (Control Processing Unit)
    - Blocco aritmetico (esegue calcoli...)
    - Unita' per controllo e sincronizzazione dei vari componenti
  - Unita' di *I/O* (input/output)
    - Tastiera, mouse, network...
    - Display, stampanti, diffusori audio, network...
  - Unita' di *memoria*
    - *Cache*, RAM, Hard Disk,...

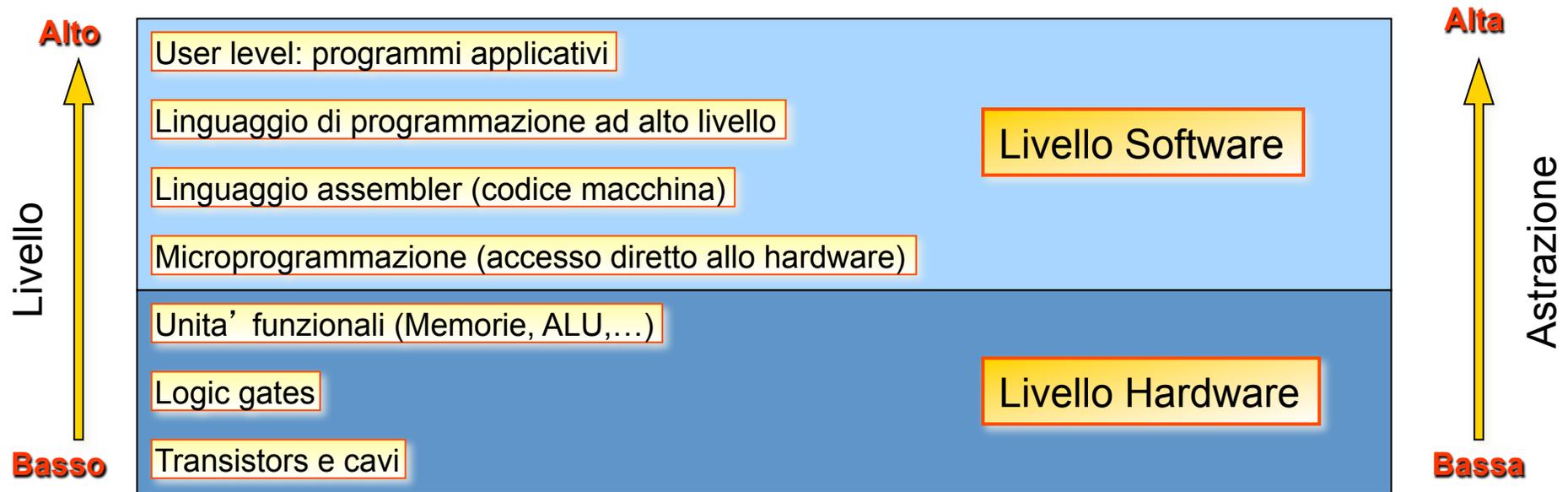
# Architettura a bus



- Le unità funzionali si scambiano informazioni utilizzando strutture condivise: i **BUS**
- Il BUS è una collezione di linee elettriche con un protocollo di comunicazione che permette di interpretare correttamente e sincronizzare le varie operazioni di trasferimento dati
- Le singole unità funzionali e/o i loro componenti interni sono univocamente determinati da un **indirizzo**
- La **condivisione** di un bus è possibile grazie a **logiche tri-state** che permettono ad agenti diversi di pilotare lo stesso filo (ovviamente in momenti diversi....)



# Livelli di astrazione di un'architettura di calcolo



- Un architettura di calcolo puo' essere scomposta in diversi *livelli di astrazione*
- Questa descrizione definisce interfacce chiare tra funzionalita' diverse nascondendo i dettagli del singolo livello
  - Un cambiamento di un componente di un certo livello non comporta (non dovrebbe comportare...) cambiamenti negli altri livelli
- L'*astrazione* cresce dai livelli hardware fino al livello applicativo

# Il linguaggio dei calcolatori: *Instruction Set*

- L'*Instruction Set* (**IS**) e' l'insieme delle istruzioni del processore i.e. il suo "vocabolario"
- Riflette l'architettura interna del processore
- Ogni processore ha il suo IS specifico ([http://en.wikipedia.org/wiki/List\\_of\\_instruction\\_sets](http://en.wikipedia.org/wiki/List_of_instruction_sets))
- Esistono varie categorie di IS che si differenziano per la loro struttura
  - **CISC** (Complex Instruction Set Computer), **RISC** (Reduced....), **VLIW** ma anche cose piu' esotiche quali **ZISC** e **NISC**...
- Comunque i vari IS hanno molti aspetti in comune...
  - Istruzioni per operazioni aritmetiche (add, sub, inc, cp, ...)
  - Istruzioni per operazioni aritmetico/logiche (and, or, shift, rotate, ...)
  - Istruzioni per trasferimento dati tra registri e memoria (load, store, ...)
  - Istruzioni per *salti condizionati e incondizionati* (jump, call, ...)
  - Istruzioni per operazioni di I/O (in, out, ...)
  - Istruzioni per gestione della CPU (nop, halt, ...)

## Il linguaggio dei calcolatori: *il programma*

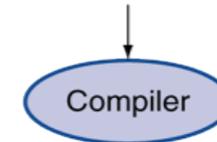
- Un **programma** e' una sequenza opportuna di istruzioni che devono essere eseguite in ordine per completare una determinata operazione.
- Il programma risiede in memoria ed ogni istruzione per essere correttamente eseguita e sincronizzata deve avere un formato noto al processore e un protocollo di lettura definito.
- Ogni istruzione deve essere poi decodificata dalla CPU prima di essere eseguita
- L'esecuzione di un programma e' una iterazione (dalla prima all'ultima istruzione) del cosiddetto ciclo di **fetch-execute**
  1. Preleva (**Fetch**) la prossima istruzione da eseguire dalla memoria
  2. Decodifica l'istruzione da eseguire (detta **OPCODE**)
  3. Legge gli eventuali operandi dalla memoria
  4. Esegue (**Execute**) le istruzioni ed immagazzina i risultati

# Uno sguardo alla gerarchia del codice

- Linguaggio di **alto livello**
  - "User friendly and intelligible"
  - Assicura produttività e (a volte...) portabilità tra diverse piattaforme
  - Strumenti software (Compilatore) traducono in assembler (o anche codice eseguibile)
- Linguaggio **Assembler**
  - Rappresentazione testuale, mnemonica delle istruzioni di un computer
  - Strumenti SW (**Assembler**) traducono "assembly code" nel linguaggio dell'Hardware
- Rappresentazione Hardware
  - Linguaggio **Macchina** dove le istruzioni ed i dati sono rappresentati da stringhe di bit

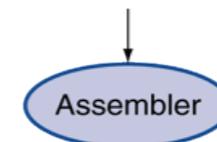
High-level  
language  
program  
(in C)

```
swap(int v[], int k)
{int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```



Assembly  
language  
program  
(for MIPS)

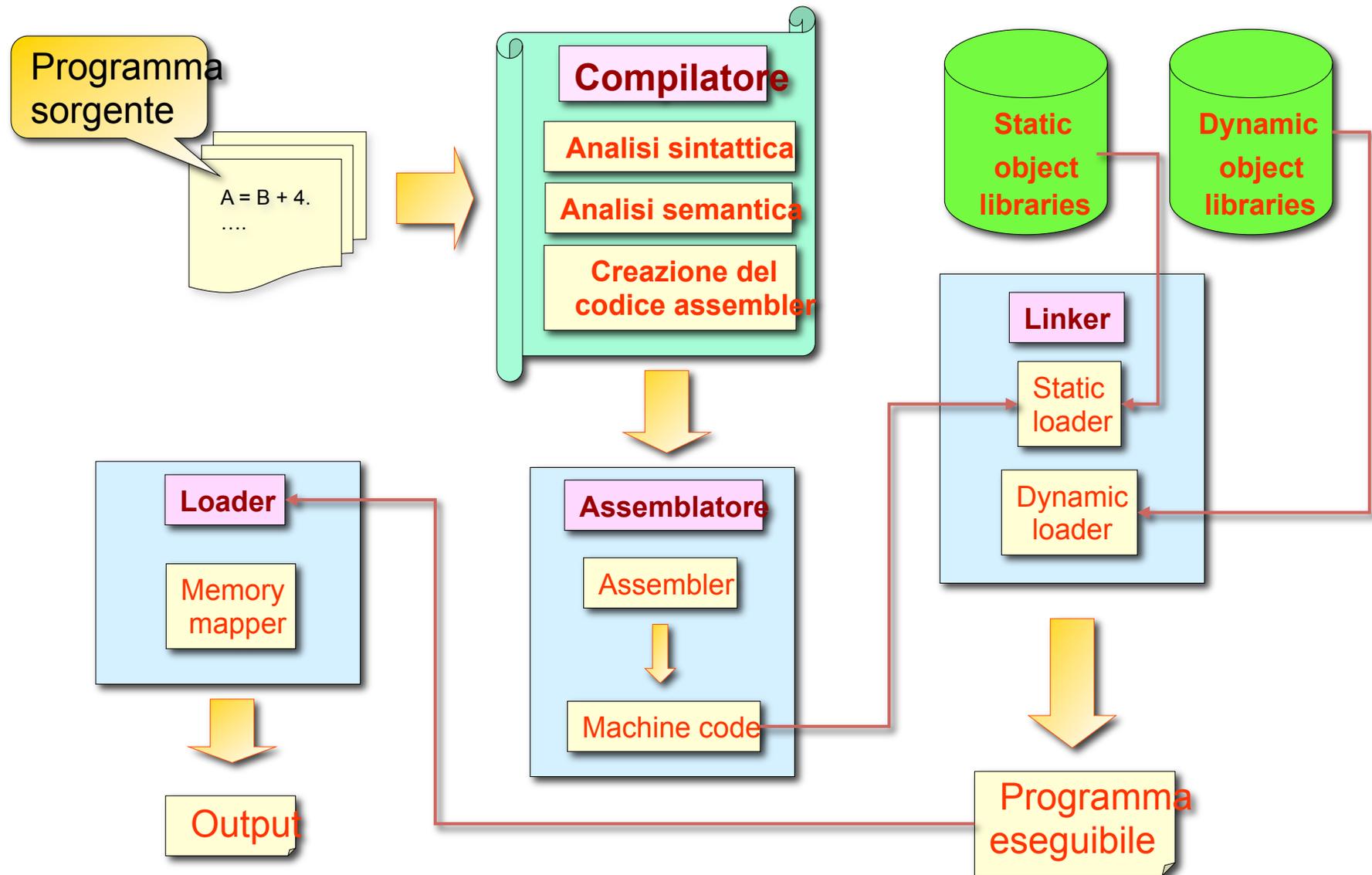
```
swap:
    muli $2, $5, 4
    add  $2, $4, $2
    lw   $15, 0($2)
    lw   $16, 4($2)
    sw   $16, 0($2)
    sw   $15, 4($2)
    jr   $31
```



Binary machine  
language  
program  
(for MIPS)

```
000000001010000100000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

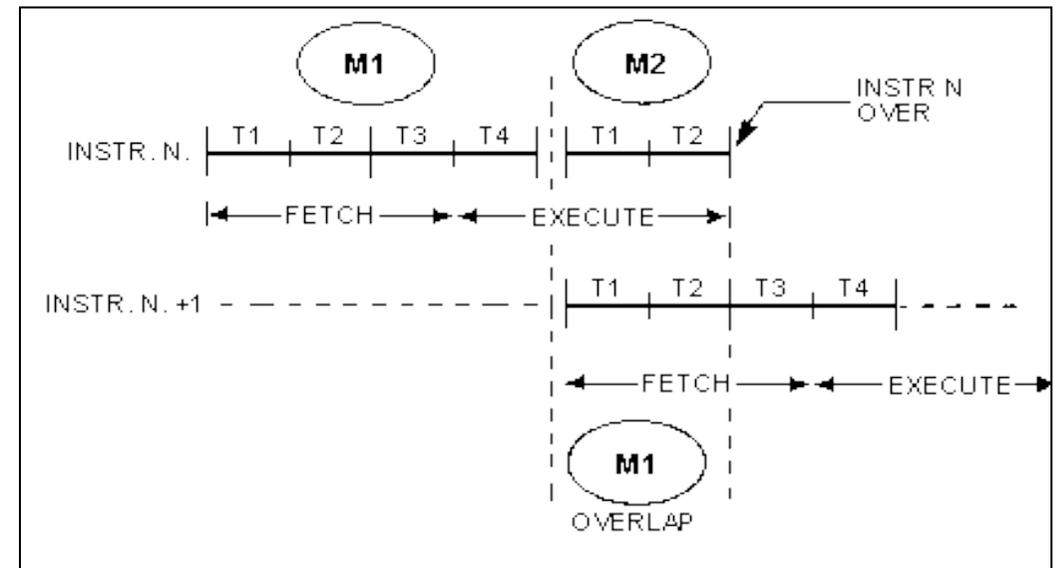
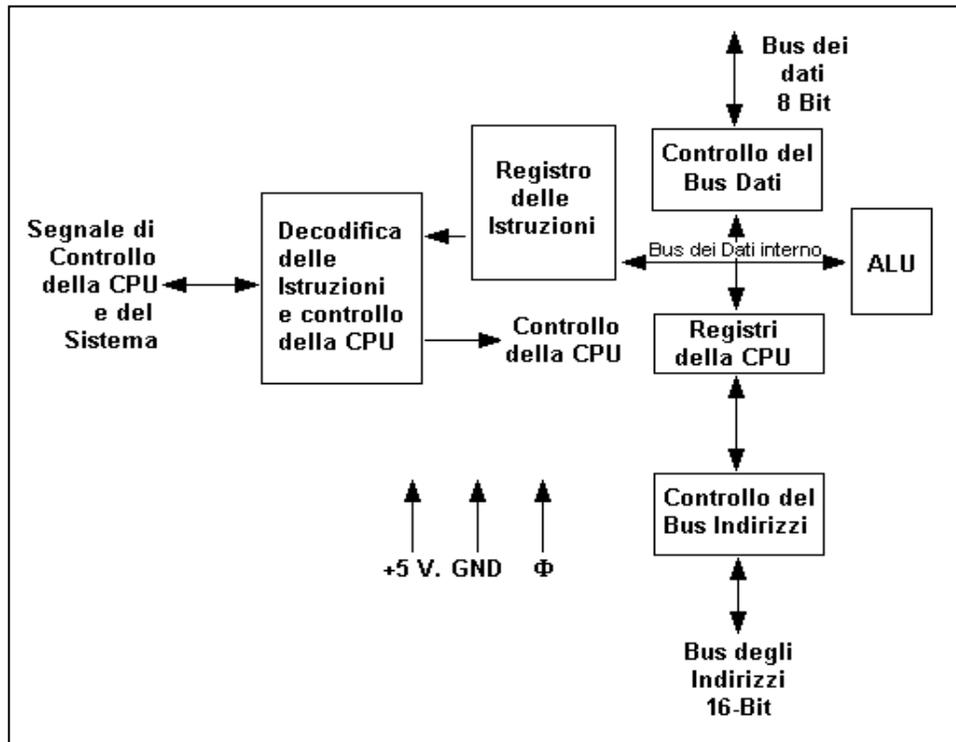
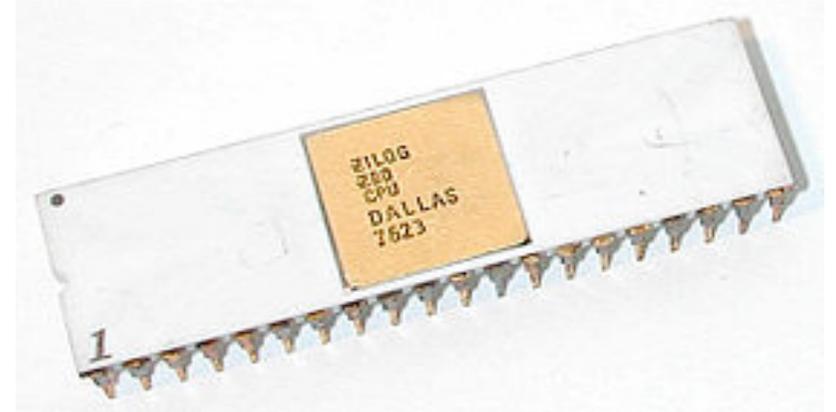
# Ciclo completo di produzione di un programma eseguibile



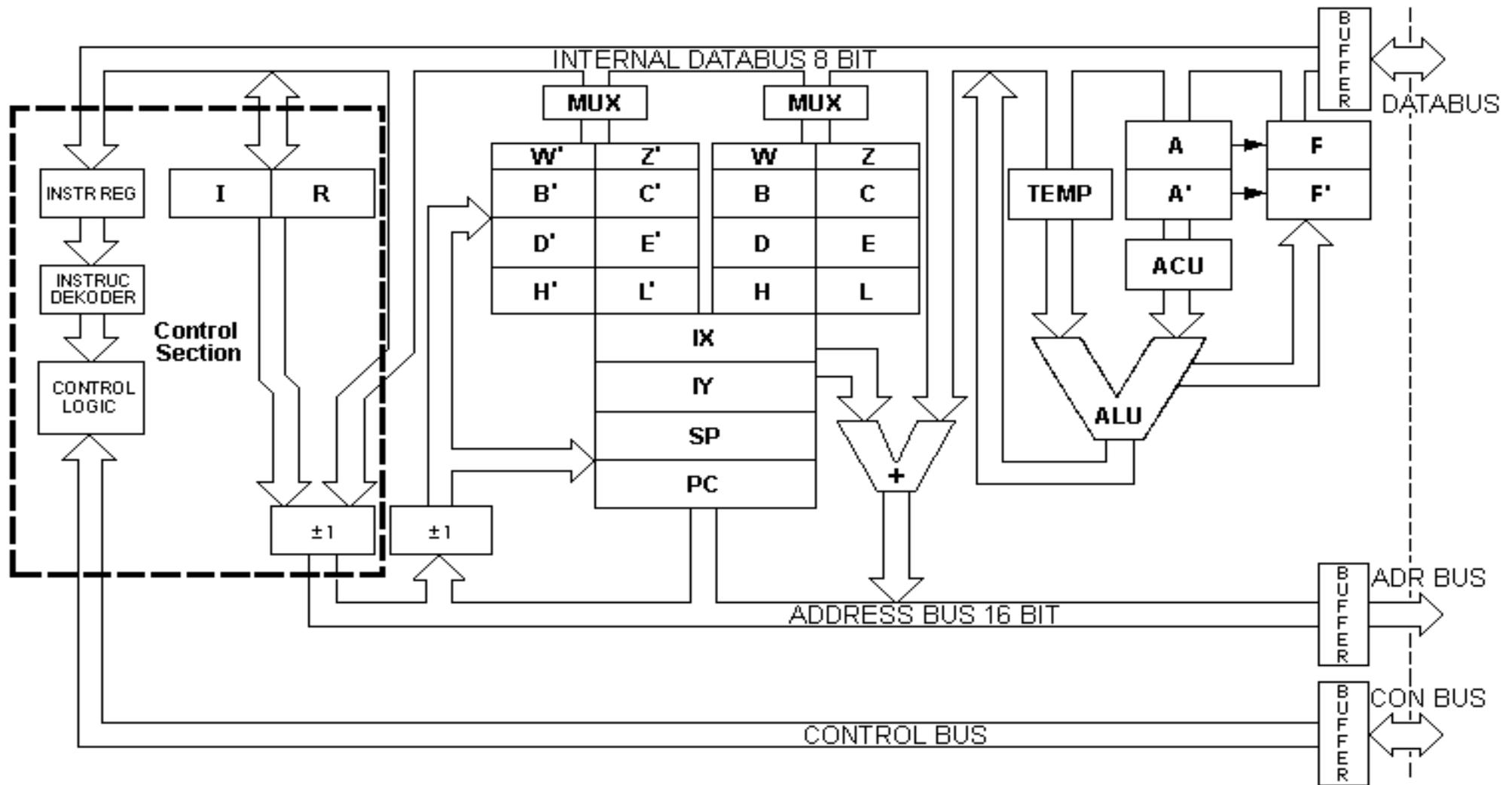
# Z80 Intro

Z80 uP:

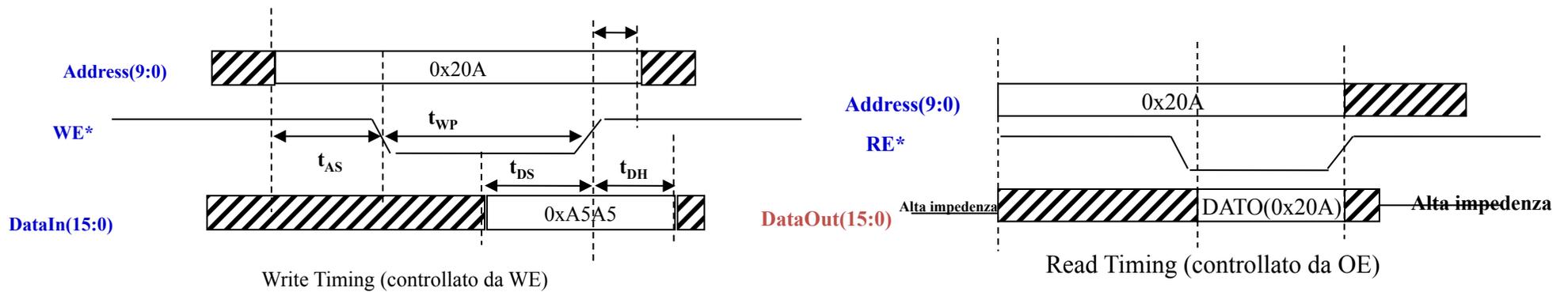
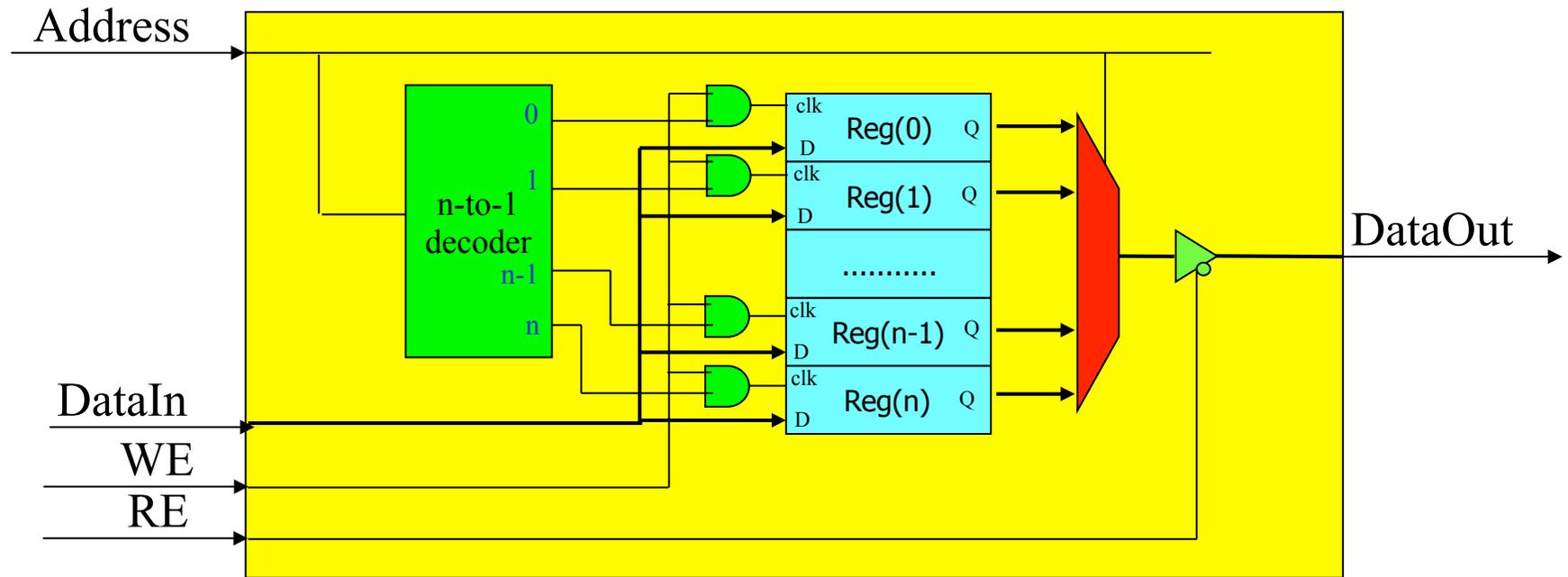
- uP di tipo CISC (*Complex Instruction Set Computer*) del 1976
  - uP piu' diffuso (calcolo prima, embedded recentemente)
  - 2 miliardi di processori realizzati!!!
- 8 bit "data word"
- 16 bit "address"
- Overlap tra fetch/execute



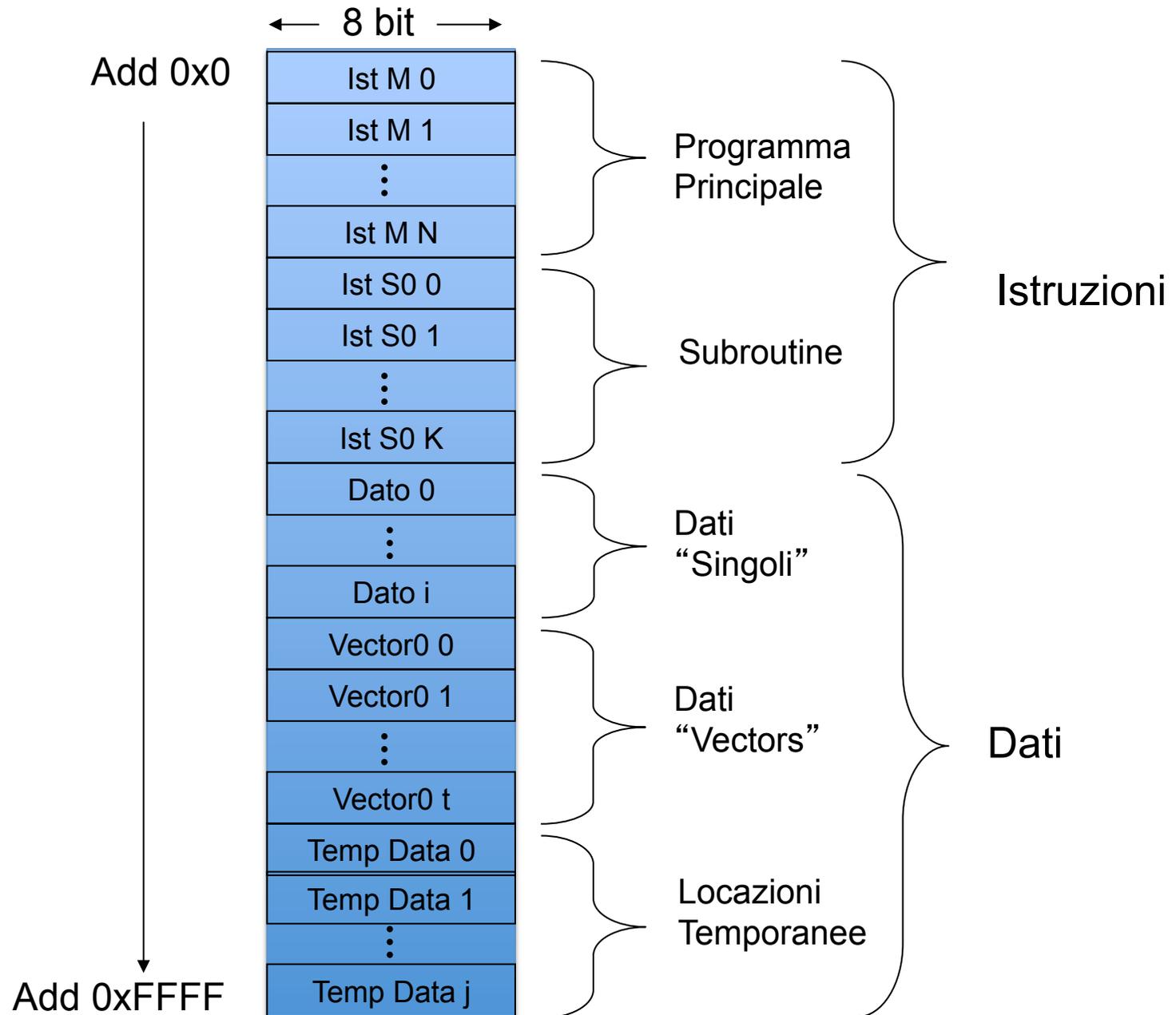
# Z80 schema logico



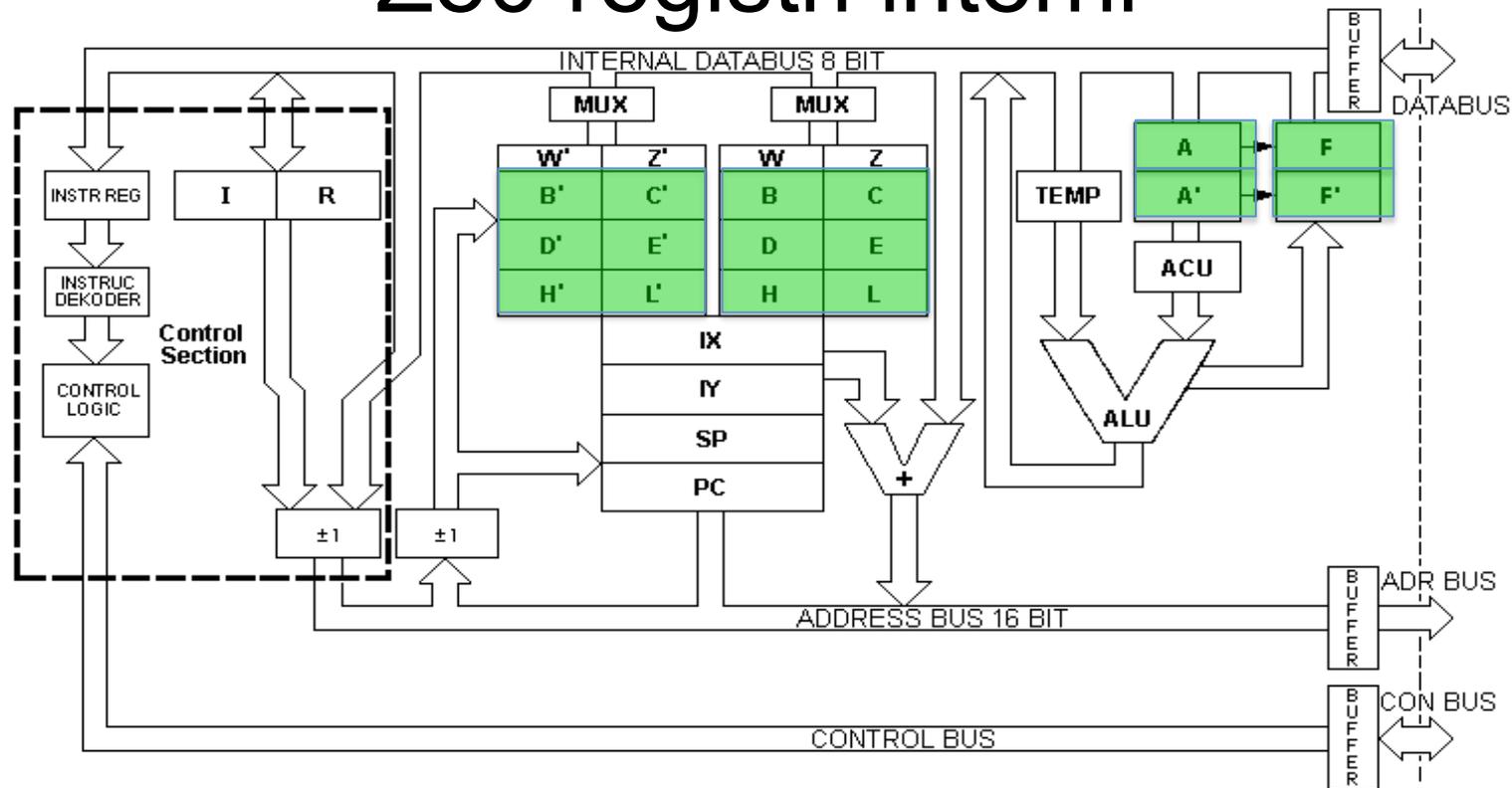
# Memoria RAM (Random Access Memory)



# Organizzazione della memoria



# Z80 registri interni



Esistono 18 registri a 8 bit e 4 a 16 bit

Registri “principali” di uso generale:

**B,C,D,E,H,L** registri per appoggio dati (operandi e risultati)  
 possono essere usati singoli o a coppie (BC,DE...16 bit)

**A** (accumulatore) risultato dell'alu

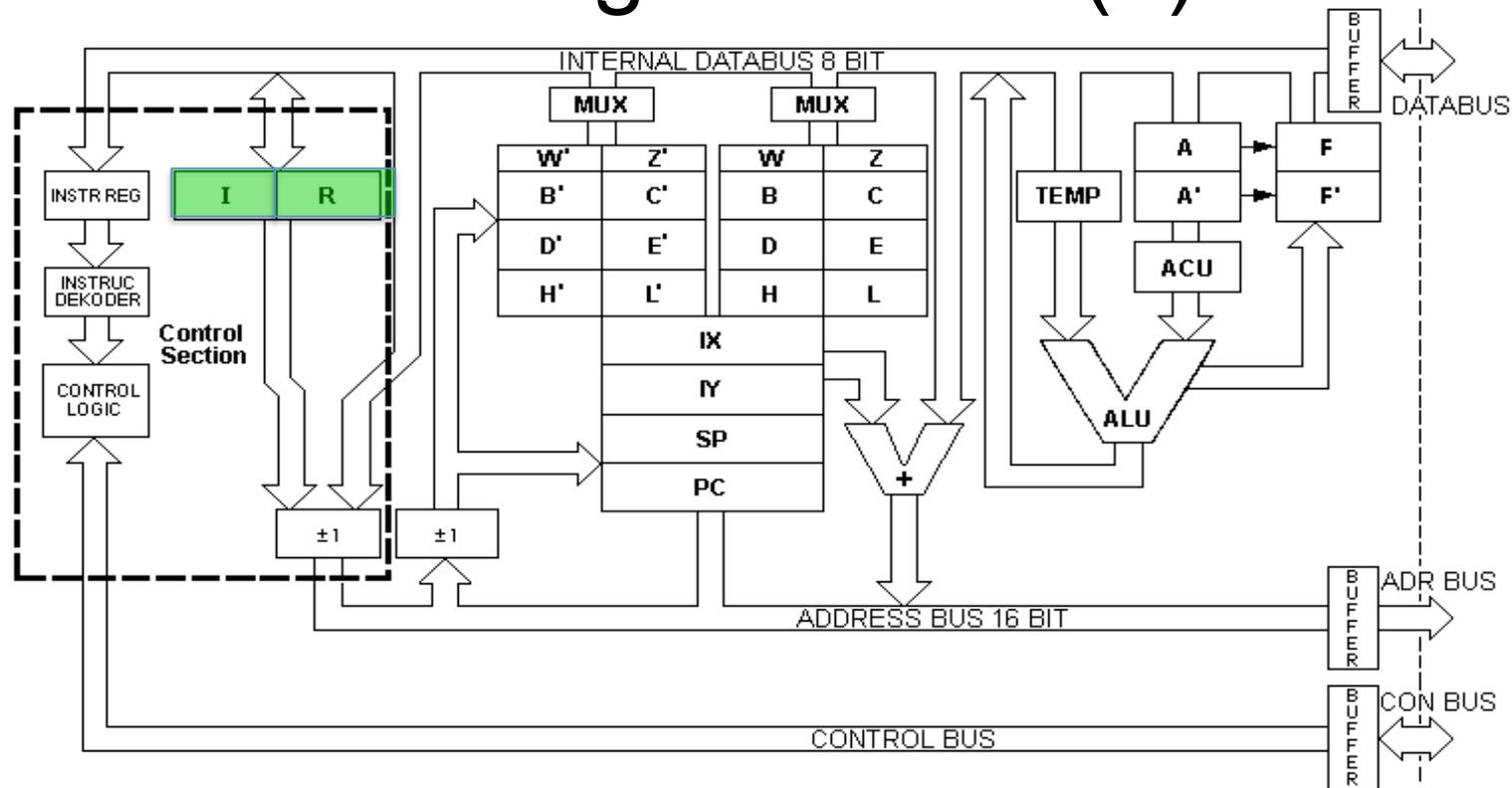
**F** registro di “flag” indicano particolari stati della CPU (Es. non-zero,overflow,...)

Registri “secondari”: “context switching” efficiente B-> B' C->C'

Registri “speciali” per controllo ed indirizzamento



# Z80 registri interni(3)



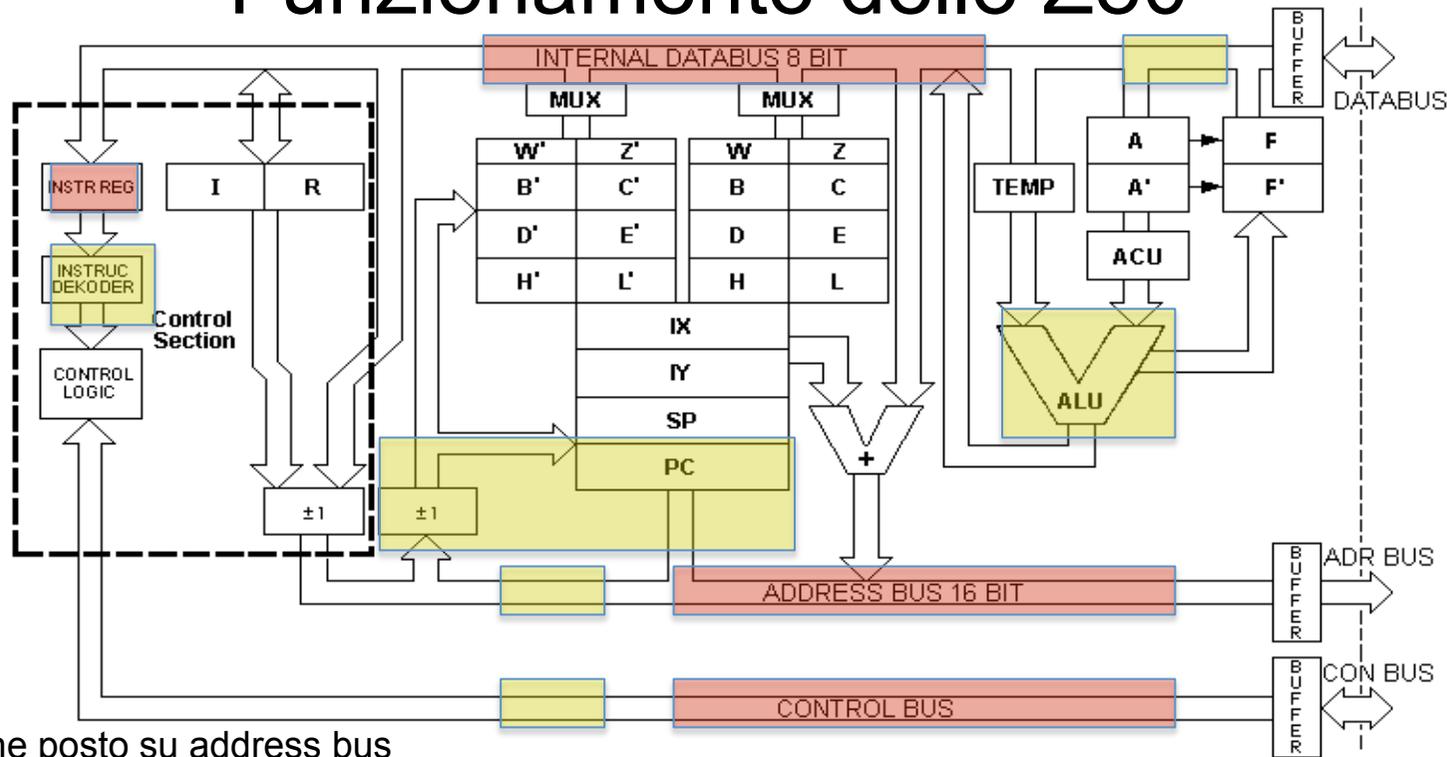
Registri accessori per la gestione della memoria e delle interruzioni

**Interrupt Vector (I)** registro per la gestione delle “interruzioni”

**Refresh Vector (R)** per gestione corretta delle memorie dinamiche



# Funzionamento dello Z80



## Ciclo di **Fetch**:

1. indirizzo PC viene posto su address bus
2. generazione sul control bus dei segnali necessari a leggere l'istruzione (ad add PC) dalla memoria
3. lettura dell'istruzione dalla memoria e scrittura nell' INSTR REG (via data bus).

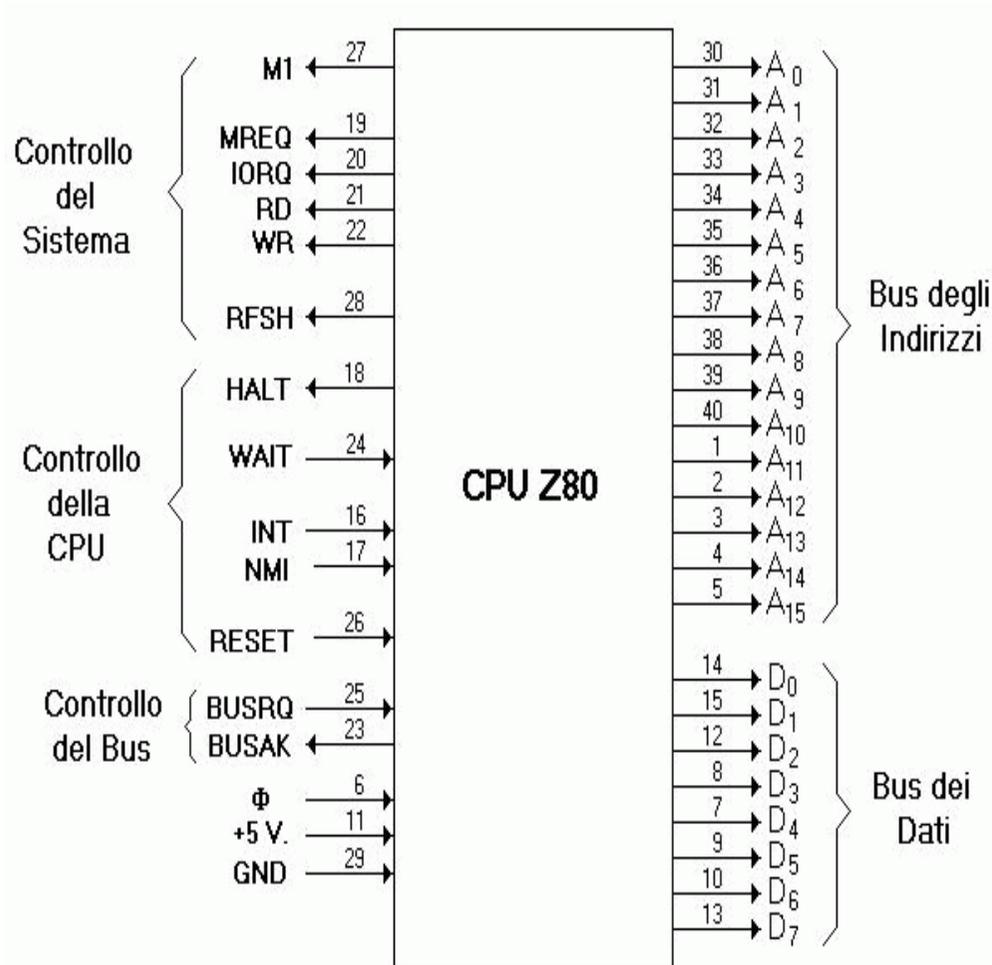
## Ciclo di **Execute**:

1. incremento del PC (per prossima istruzione)
2. decodifica dell'istruzione
3. eventuale lettura dei dati
4. esecuzione dell'istruzione

La control logic si occupa di coordinare le varie unita' di decodifica e logico/aritmetiche

**Nota!!** Lo Z80 comincia sempre ad eseguire il programma dalla locazione di memoria 0x0000 ovvero carica come prima istruzione il contenuto di tale locazione

# Z80 Pinout



$\Phi$  Clock 0 to 4MHz

**A0-A15** Bus degli indirizzi,

**D0-D7** Bus dei dati (tristate–bidirezionale)

**M1** (attivo basso) La CPU si trova nel ciclo di “fetch”

**MREQ** (attivo basso) Indirizzo valido per operazione in memoria

**IORQ** (attivo basso) Indirizzo valido per operazione di I/O

**RD** (attivo basso) CPU vuole effettuare una lettura dalla memoria

**WR** (attivo basso) CPU vuole effettuare una scrittura in memoria

**HALT** (attivo basso) stato di halt raggiunto

**WAIT** (attivo basso) La CPU rimane in stato di attesa

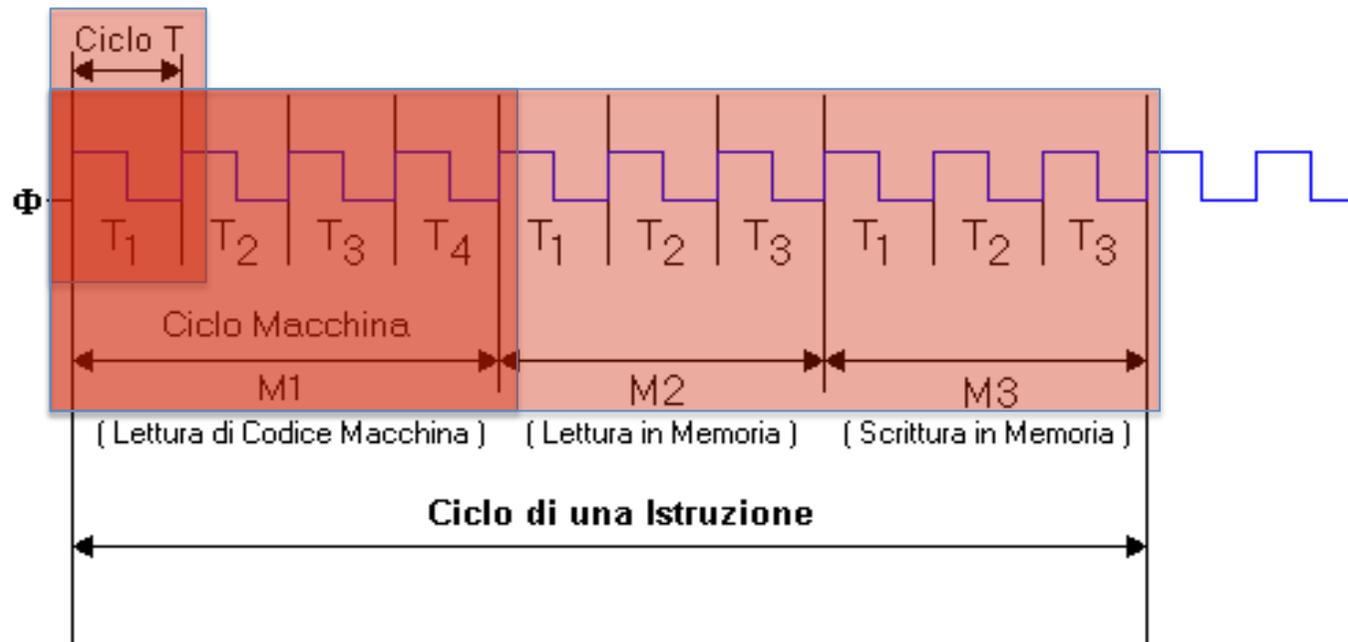
**INT** (attivo basso) Richiesta di interruzione

**NMI** Richiesta di interrupt di non mascherabile. Costringe la CPU a ripartire da un indirizzo noto (0x66)

**RESET** (attivo basso) “resetta” la CPU per iniziare le operazioni

**BUSRQ** (attivo basso) la CPU porta controlli, indirizzi e dati in stato di alta impedenza e attiva il segnale **BUSAK** (attivo basso)

# Z80 timing



**Ciclo T** ciclo di clock

**Ciclo macchina** ciclo dell'operazione elementare

**Ciclo di istruzione** composto da piu' cicli macchina

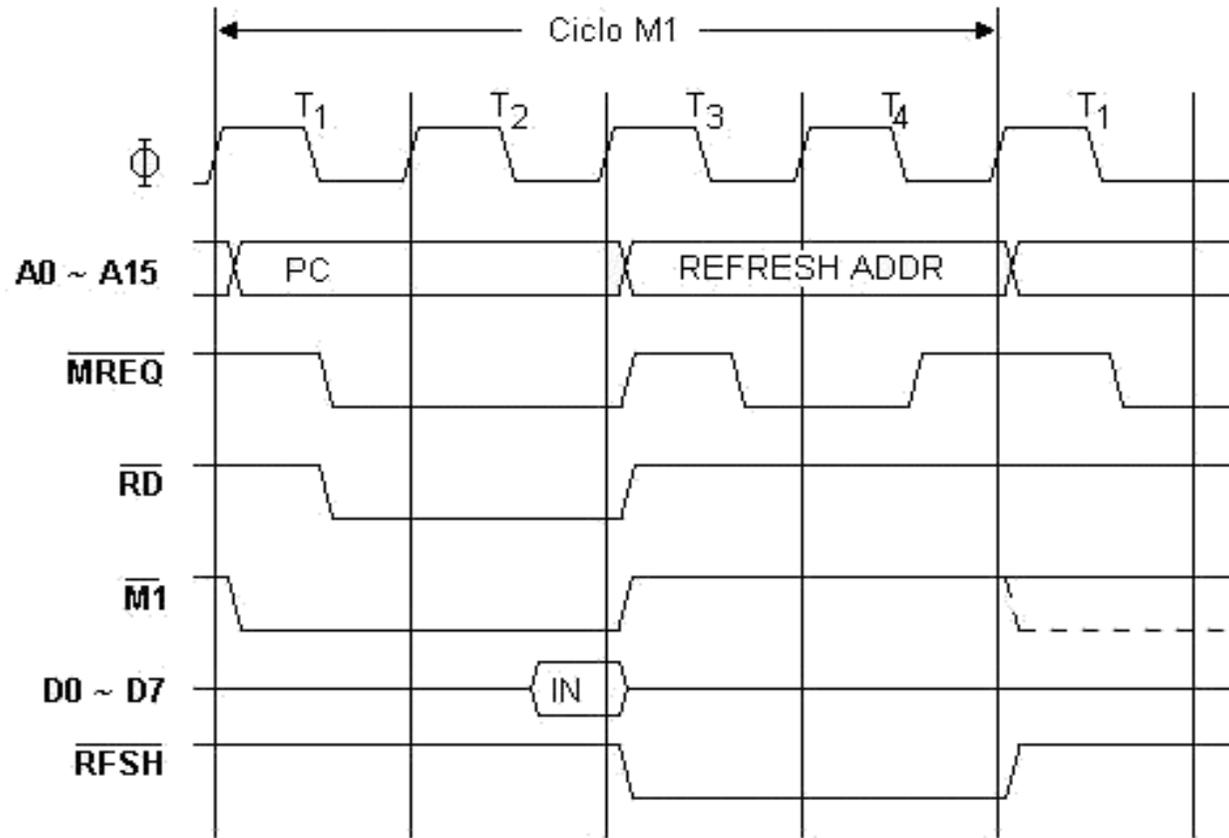
Esempio

**M1** lettura e decodifica dell'istruzione (4 periodi di clock)

**M2** lettura dati tra memoria/dispositivi di I/O (3-5 periodi di clock)

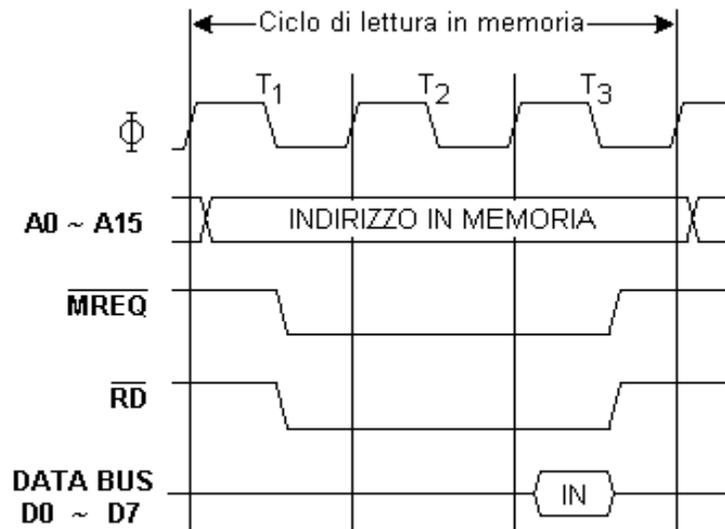
**M3** scrittura dati in memoria/dispositivi di I/O (3-5 periodi di clock)

# Z80 timing: fetch

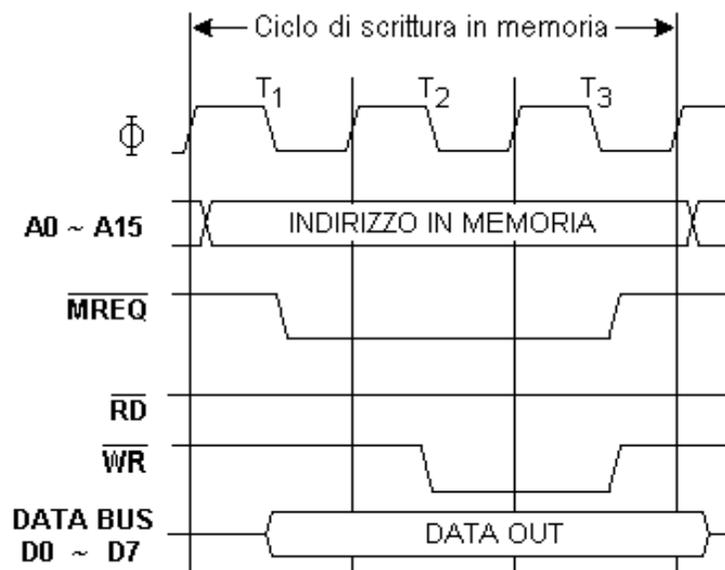


Temporizzazione della CPU Z80 nel ciclo M1 (prelievo del codice operativo)

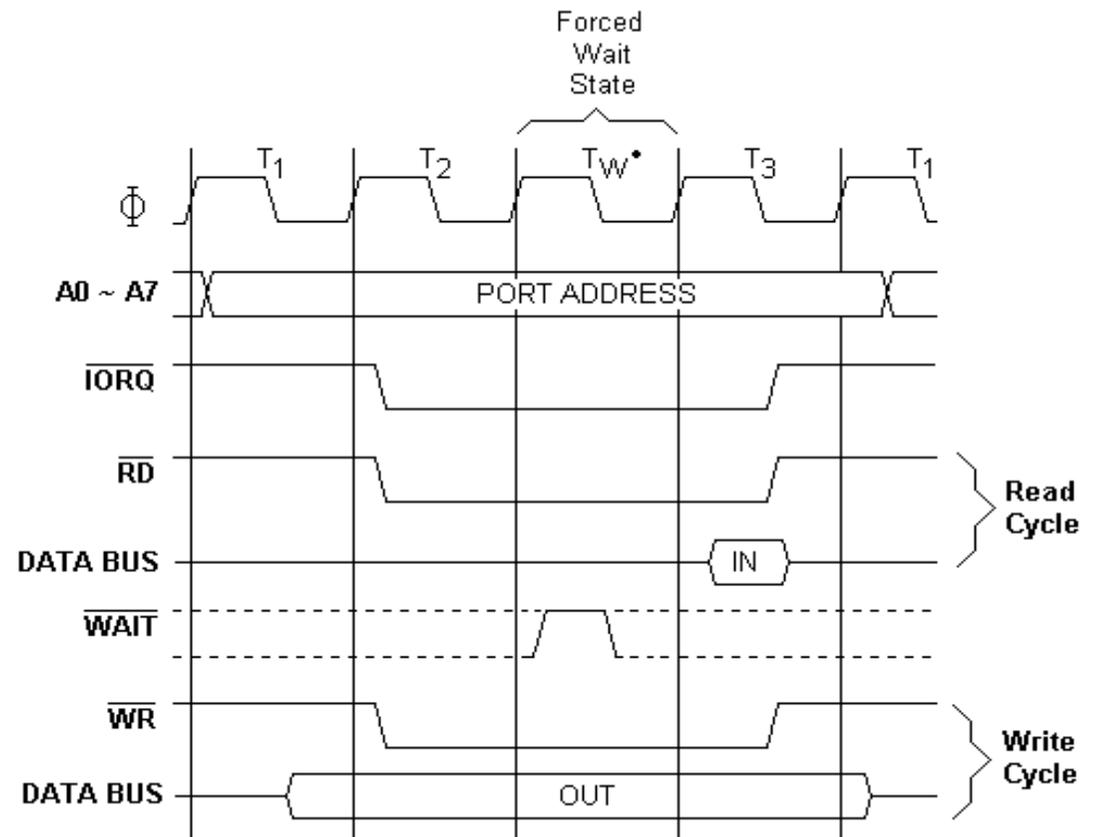
# Z80 timing: READ-WRITE-I/O



Ciclo di lettura



Ciclo di scrittura



# Z80 Instruction Set

- 5 classi di istruzioni:

1. “Data transfer” registro-registro, registro-memoria
2. Aritmetiche e logiche
3. Salto/chiamata/ritorno
4. Input/Output
5. Controllo

- Formato in forma mnemonica:

OpCode Destinazione, Sorgente

**LD A, (2000)**

Carica nel registro A il contenuto della locazione di memoria 2000

- Esistono istruzioni a 0,1,2 operandi:

Esempio

**HALT** (0 operandi <-> 1 byte)

**DECH** (1 operando <-> 2 byte)

**ADDA,1** (2 operandi <-> 3 byte)

- Ogni OpCode e' codificato in linguaggio macchina (codice binario) da un byte differente

Es: NOP -> 0x00

INCB -> 0x04

# Z80 Instruction Set Table

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
<b>0</b>	NOP	LD BC, **	LD (BC),A	INC BC	INC B	DEC B	LD B,*	RLCA	EXAF, AF'	ADD HL, BC	LD A, (BC)	DEC BC	INC C	DEC C	LD C, *	RRCA
<b>1</b>	DJNZ *	LD DE, **	LD( DE),A	INC DE	INC D	DEC D	LD D, *	RLA	JR *	ADD HL, DE	LD A, (DE)	DEC DE	INC E	DEC E	LD E, *	RRA
<b>2</b>	JRNZ *	LD HL, **	LD (**), HL	INC HL	INC H	DEC H	LD H, *	DAA	JRZ, *	ADD HL, HL	LD HL, (**)	DEC HL	INC L	DEC L	LD L, *	CPL
<b>3</b>	JRNC *	LD SP, **	LD (**), A	INC SP	INC (HL)	DEC (HL)	LD (HL), *	SCF	JRC, *	ADD HL, SP	LD A, (**)	DEC SP	INC A	DEC A	LD A, *	CCF
<b>4</b>	LD B, B	LDB , C	LD B, D	LD B, E	LD B, H	LDB , L	LD B, (HL)	LD B, A	LD C, B	LD C, C	LD C, D	LD C, E	LD C, H	LD C, L	LD C, (HL)	LD C, A
<b>5</b>	LD D, B	LD D, C	LD D, D	LD D, E	LD D, H	LD D, L	LD D, (HL)	LD D, A	LD E, B	LD E, C	LD E, D	LD E, E	LD E, H	LD E, L	LD E, (HL)	LD E, A
<b>6</b>	LD H, B	LD H, C	LD H, D	LD H, E	LD H, H	LD H, L	LD H, (HL)	LD H, A	LD L, B	LD L, C	LD L, D	LD L, E	LD L,H	LD L, L	LD L, (HL)	LD L, A
<b>7</b>	LD (HL), B	LD (HL), C	LD (HL), D	LD (HL), E	LD (HL), H	LD (HL), L	HALT	LD (HL), A	LD A, B	LD A, C	LD A, D	LD A, E	LD A, H	LD A, L	LD A, (HL)	LDA, A
<b>8</b>	ADD A, B	ADD A, C	ADD A, D	ADD A, E	ADD A, H	ADD A, L	ADD A, (HL)	ADD A, A	ADC A, B	ADC A, C	ADC A, D	ADC A, E	ADC A, H	ADC A, L	ADC A, (HL)	ADC A, A
<b>9</b>	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB (HL)	SUB A	SBC A, B	SBC A, C	SBC A, D	SBC A, E	SBC A, H	SBC A, L	SBC A, (HL)	SBC, A
<b>A</b>	AND B	AND C	AND D	AND E	AND H	AND L	AND (HL)	AND A	XOR B	XOR C	XOR D	XOR E	XOR H	XOR L	XOR (HL)	XOR A
<b>B</b>	OR B	OR C	OR D	OR E	OR H	OR L	OR (HL)	OR A	CP B	CP C	CP D	CP E	CP H	CP L	CP (HL)	CP A
<b>C</b>	RETNZ	POP BC	JPNZ, **	JP **	CALL NZ, **	PUSH BC	ADD A, *	RST 0H	RET Z	RET	JPZ, **	[1]	CALL Z, **	CALL **	ADC A, *	RST 8H
<b>D</b>	RET NC	POP DE	JPNC, **	OUT (*), A	CALL NC, **	PUSH DE	SUB *	RST 10H	RET C	EXX	JPC, **	IN A, (*)	CALL C, **	[1]	SBC A, *	RST 18H
<b>E</b>	RET PO	POP HL	JP PO, **	EX(SP) HL	CALL PO, **	PUSH HL	AND *	RST 20H	RET PE	JP (HL)	JP PE, **	EX DE,HL	CALL PE,**	[1]	XOR *	RST 28H
<b>F</b>	RET P	POP AF	JP P, **	DI	CALL P, **	PUSH AF	OR *	RST 30H	RET M	LD SP, HL	JPM, *	EI	CALL m, **	[1]	CP *	RST 38H

(1) Primo byte di una istruzione con codice a piu' bytes;

(\*): L'istruzione si completa con un byte;

(\*\*): L'istruzione si completa con due bytes

# Z80 Programmazione(1)

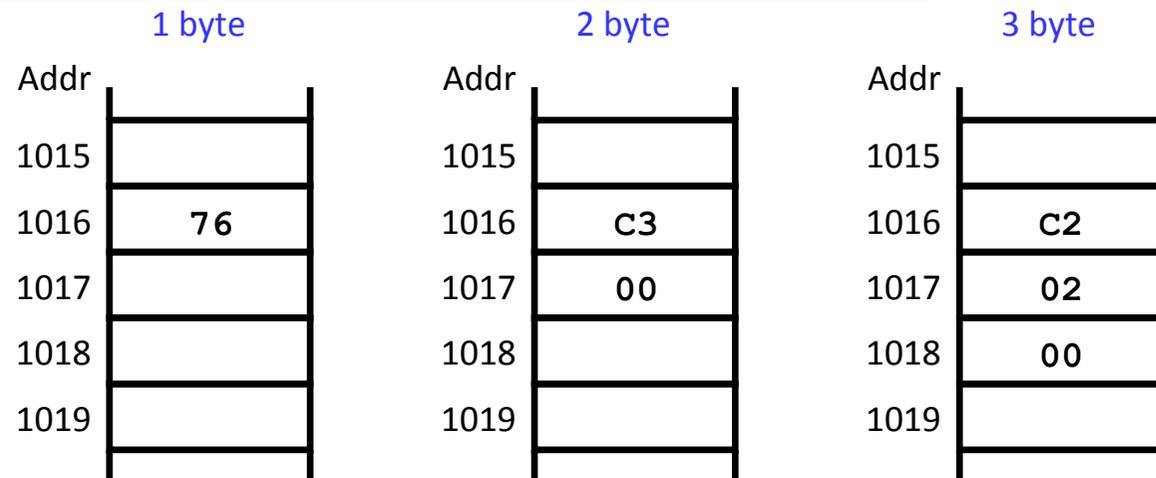
Es: 1 Loop infinito

Indirizzo	Istruzione	Mnemonico	Significato
0000	00	NOP	Nessuna operazione
0001	C3	JP, 0000	Salta a locazione 00
0002	00		
0003	00	NOP	

Es: 2 Decrementa il registro fino a 0

Indirizzo	Label	Istruzione	Mnemonico	Significato
0000		06	LDB,64	Carica nel registro B il valore 64 (Hex)
0001		64		
0002	loop	05	DEC B	Decrementa registro B
0003		C2	JPNZ, 0002(loop)	Salta a loc. 0002 se ultima operazione $\neq$ 0
0004		02		
0005		00		
0006		76	HALT	

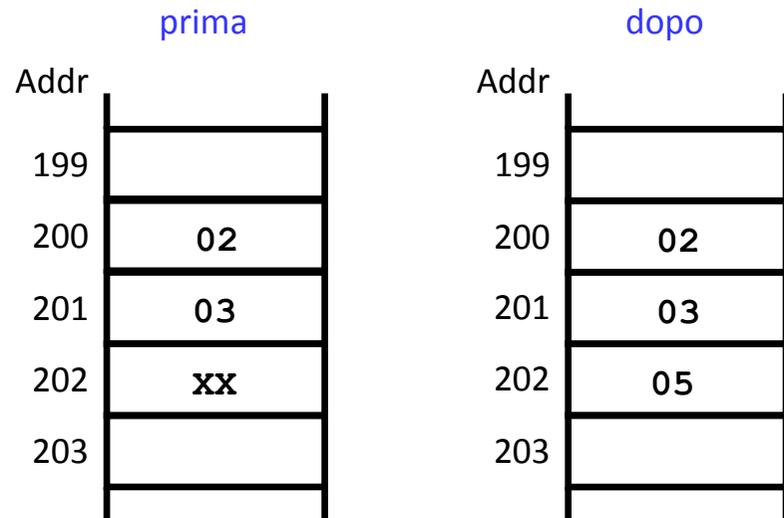
Organizzazione delle istruzioni in memoria



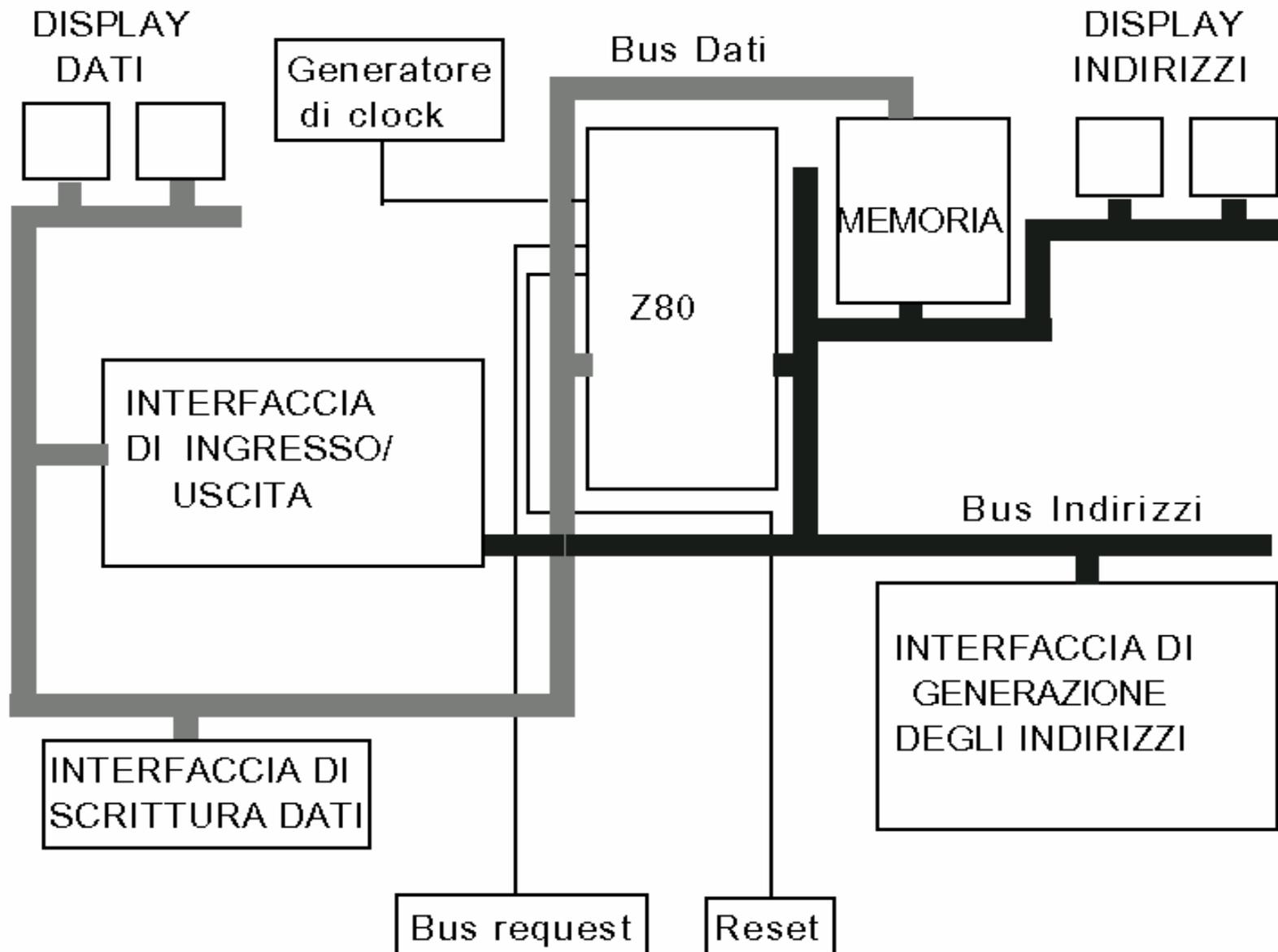
# Z80 Programmazione(2)

Es: 3 Somma di due addendi

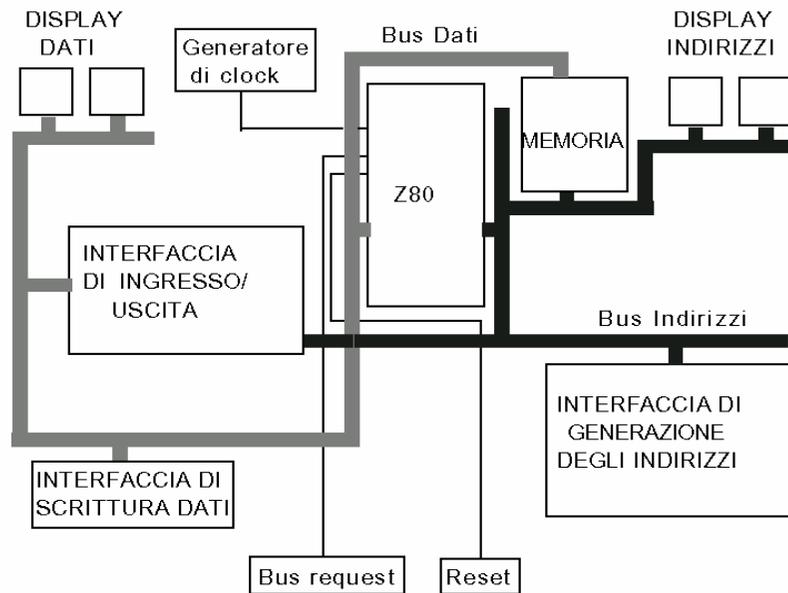
Indirizzo	Label	Istruzione	Mnemonico	Significato
0000		3A	LDA,(0200)	Carica A con il contenuto della locazione 200
0001		00		
0002		02		
0003		2A	LDHL,0201	Carica HL con l'indirizzo del secondo addendo
0004		01		
0005		02		
0006		86	ADDA,(HL)	Somma di A e loc. HL 0007
		32	LD (0202),A	Scrivo A in memoria a locazione 202
0008		02		
0009		02		
000A		76	HALT	Fine programma



# Z80 uC



# Z80 uC(2)



Due stati di funzionamento:  
**RUN mode**  
**SYSTEM mode**

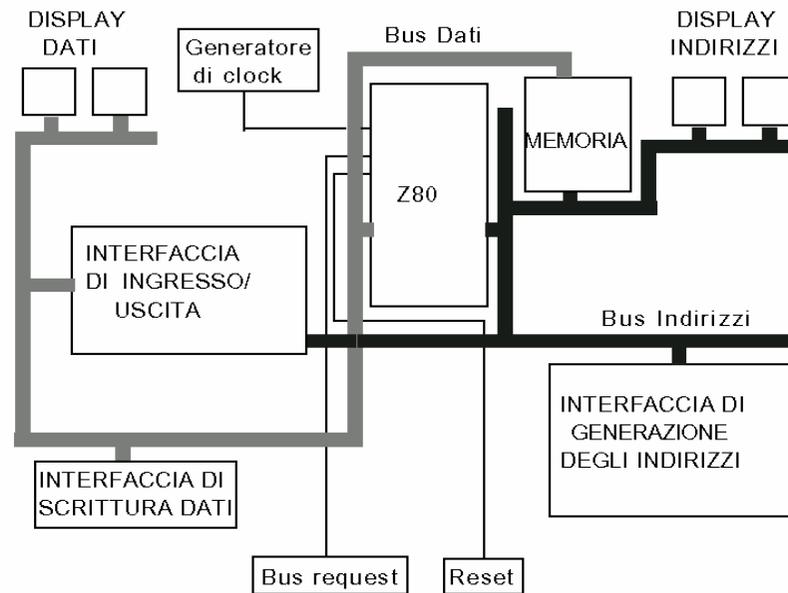
## **SYSTEM mode:** (BUS request attivo)

- Caricamento della memoria con dati e programma
- Gli indirizzi vengono generati da due contatori up/down collegati in cascata (4+4 bit) pilotati da due pulsanti (up e down).
- I dati vengono predisposti da 8 interruttori divisi in due gruppi (3:0) e (7:4)
- Comando scrittura attraverso pulsante DATA\_WRITE

## **RUN mode:** (Bus request non attivo) Z80 pilota bus dati ed indirizzi

- Esecuzione del programma a partire dalla locazione 0x0
- Z80 controlla bus dati ed indirizzi

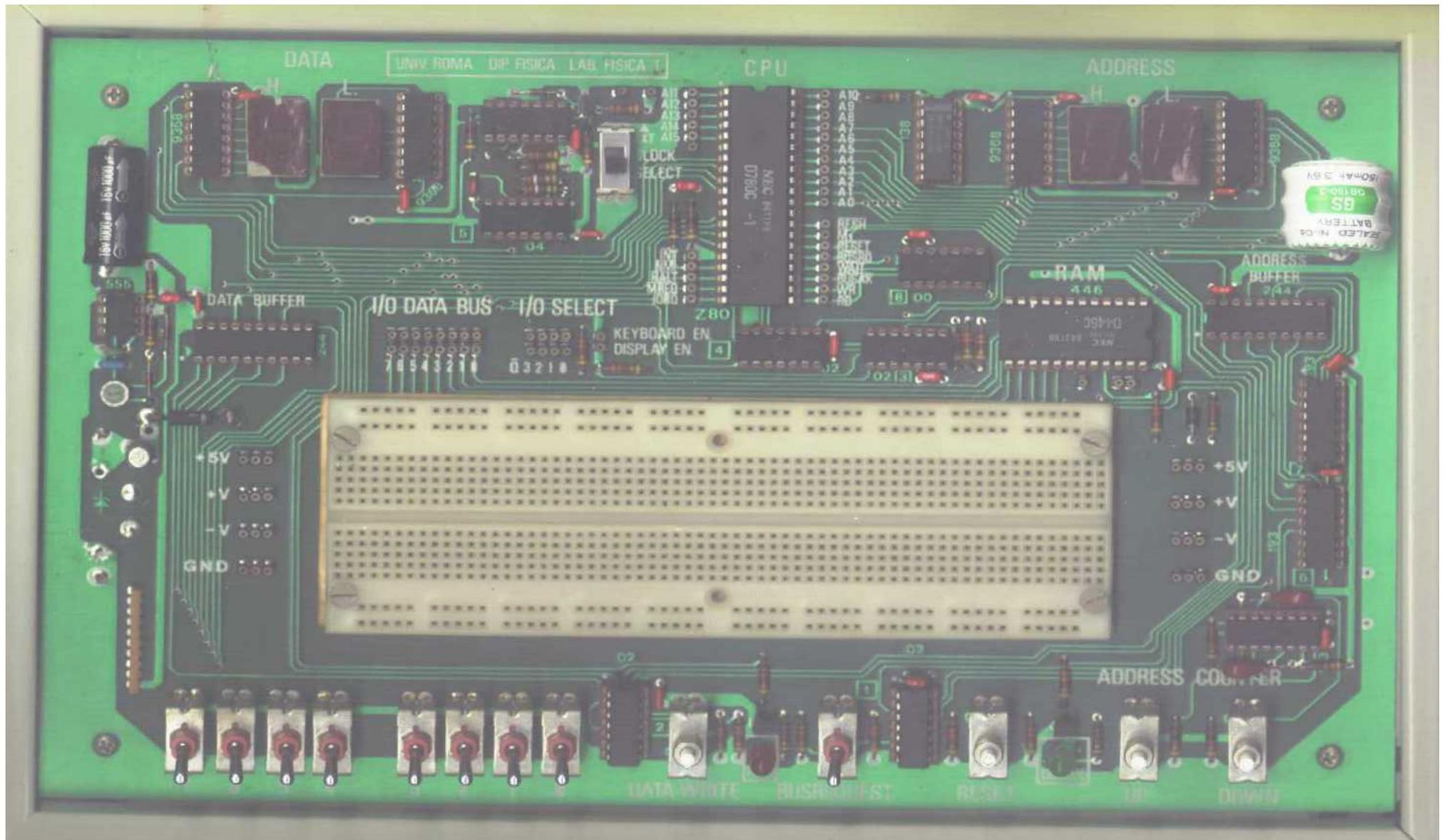
# Z80 uC(2)



Sono presenti sulla piastra:

- **Memoria RAM** da 2 KByte indirizzata attraverso gli 8 bit meno significativi
- **Display a due cifre Hex per dati**
- **Display a due cifre Hex per address**
  
- **Interfaccia di I/O:** attivata dal segnale IORQ.  
Può pilotare fino a 8 periferiche (3 address bit) Data bus -> Display dati
  
- **Generatore di clock**  
Clk interno 1 MHz, ingresso per Clk esterno
  
- **“breadboard”** per esperimenti con Z80

# Z80 uC(4)



# Z80 uC: esecuzione di programmi

Per immettere un programma e farlo funzionare si deve eseguire la seguente sequenza.

- 1)Prendere il controllo del bus mediante l'interruttore **BUSREQUEST**; si ha il controllo quando é acceso il led verde **BUSAK**;
- 2)Mediante i pulsanti **UP** o **DOWN** posizionare il contatore degli indirizzi (ADDRESS COUNTER) nella locazione di memoria da cui si desidera far partire il programma; l'indirizzo relativo appare sul visualizzatore degli ADDRESS in forma esadecimale (nibble High e Low).
- 3)Impostare (in forma binaria) i byte delle istruzioni del programma da eseguire mediante gli interruttori 0-3, 4-7 (negli esempi seguenti tali byte vengono indicati come "dato");
- 4)Trasferire nella locazione di memoria indirizzata il dato impostato mediante il pulsante DATA WRITE;
- 5)Incrementare di uno la posizione dell'ADDRESS COUNTER mediante il pulsante UP;
- 6)Ripetere la sequenza 3-4-5 fino al termine del programma;
- 7)Per controllare l'esattezza dei dati impostati si può decrementare l'ADDRESS COUNTER mediante il pulsante DOWN verificando, locazione per locazione, il contenuto della memoria e correggendo gli eventuali errori:
- 8)Restituire i bus alla CPU mediante l'interruttore BUSREQUEST (il led verde si spegne);
- 9)Premere momentaneamente il pulsante di RESET; si accende il led rosso di RUN e la CPU cerca la prima istruzione da eseguire in 0000<sub>HEX</sub>.