Laboratorio di Segnali e Sistemi
- Capitolo 7 -

Z80

Claudio Luci
SAPIENZA
UNIVERSITÀ DI ROMA

last update : 070117

Sommario del capitolo:

- Introduzione storica
- Macchina di von Neumann
- Linguaggi di programmazione
- Il microprocessore Z80
- Z80: diagrammi a blocchi della CPU
- Z80: registri interni
- Z80: timing
- Programmazione dello Z80
- Illustrazione della scheda didattica dello Z80
- Spiegazione dell'esercitazione sullo Z80

Introduzione ai microprocessori

- In base a quanto abbiamo visto finora studiando l'elettronica digitale, fondamentalmente abbiamo la corrispondenza:

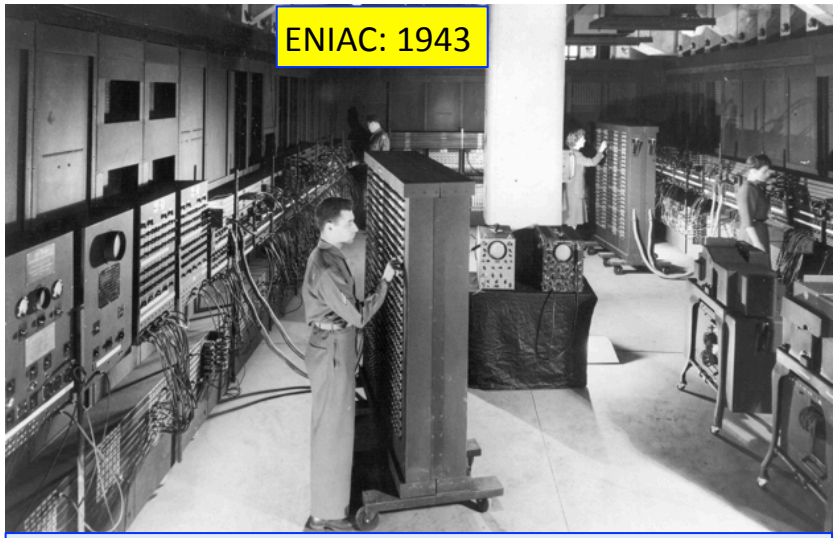
un progetto → un circuito (hardware)

- La risoluzione di un problema con questa tecnica porta alla soluzione in **logica cablata** dello stesso; la soluzione si dice che è solo di tipo **hardware**.
- Questo modo di procedere aveva portato allo sviluppo di nuove applicazioni nei campi dell'elettronica e dell'automazione, ma alla fine degli anni 60 ci si era trovati davanti ad un vicolo cieco dato che ogni nuova applicazione richiedeva una complessa e costosa fase progettuale e realizzativa.
- Con l'avvento e la commercializzazione a basso costo del microprocessore lo sviluppo di un progetto viene fatto con la corrispondenza:

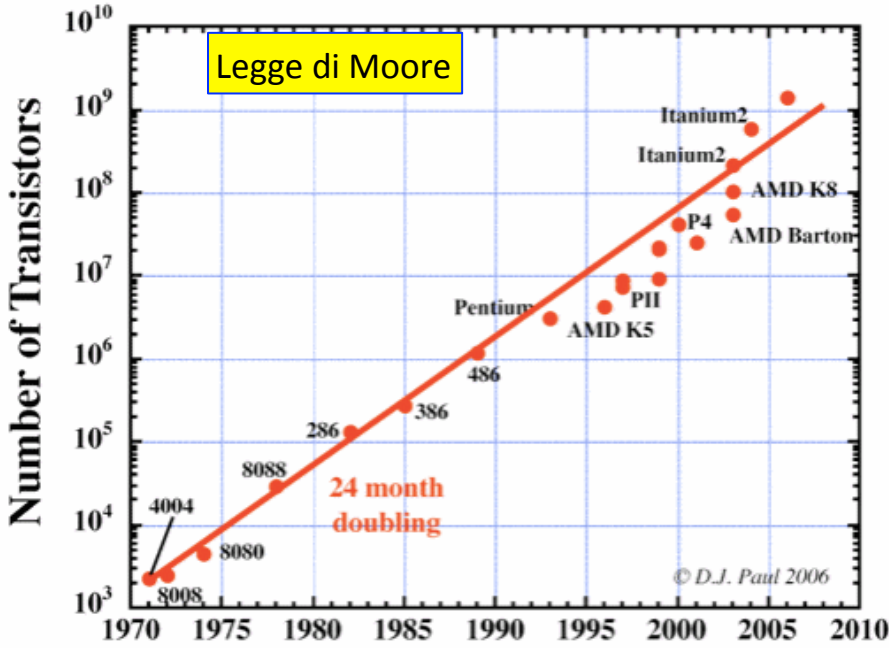
un progetto → un circuito (hardware) + un programma specifico (software)

- Fermo restando che si ha comunque una parte difficile da realizzare (l'hardware del microprocessore e la circuiteria di supporto), ciascun progetto (di una stessa tipologia) può essere realizzato ponendo in esecuzione una particolare procedura di funzionamento dell'hardware data dal **programma** specifico per quel progetto.
- Eseguendo un altro programma, il sistema funziona in un altro modo completamente differente pur utilizzando la stessa piattaforma hardware.
- L'avvento del microprocessore ha rivoluzionato con l'informatica il modo di vivere e di lavorare dell'uomo.

Computer e tecnologia



È il primo computer elettronico omni purpose della storia. Era fatto da 18000 valvole e 1500 rele'.



Prima legge di Moore (enunciata nel 1971): la complessità di un microcircuito, misurata ad esempio tramite il numero di transitor per chip, raddoppia ogni 18 mesi.

DOWNSIZING AND UPGRADING

The inception of computing inspired a remarkable race for faster, smaller, lighter, cheaper hardware.

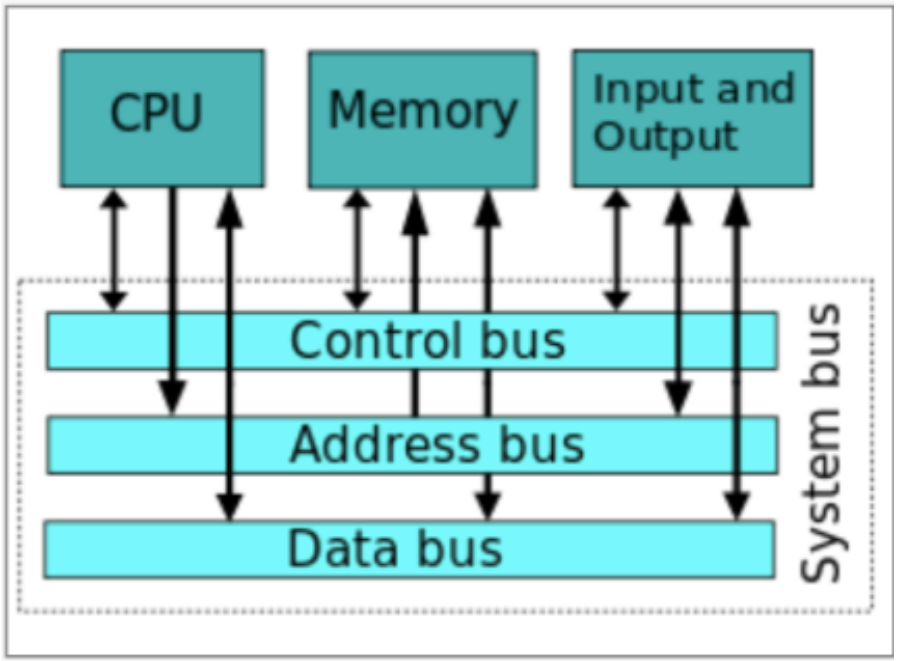
	ENIAC	Intel Core Duo chip
Debut	1946	2006
Performance	5,000 addition problems/sec	21.6 billion ops/sec
Power use	170,000 watts	31 watts max
Weight	28 tons	negligible
Size	80' w x 8' h	90.3 sq. mm.
What's inside	17,840 vacuum tubes	151.6 M transistors
Cost	$487,000	$637

Generation	Approximate Dates	Technology	Typical Speed (operations per second)
1	1946–1957	Vacuum tube	40,000
2	1958–1964	Transistor	200,000
3	1965–1971	Small and medium scale integration	1,000,000
4	1972–1977	Large scale integration	10,000,000
5	1978–1991	Very large scale integration	100,000,000
6	1991-	Ultra large scale integration	1,000,000,000

Architettura di un computer

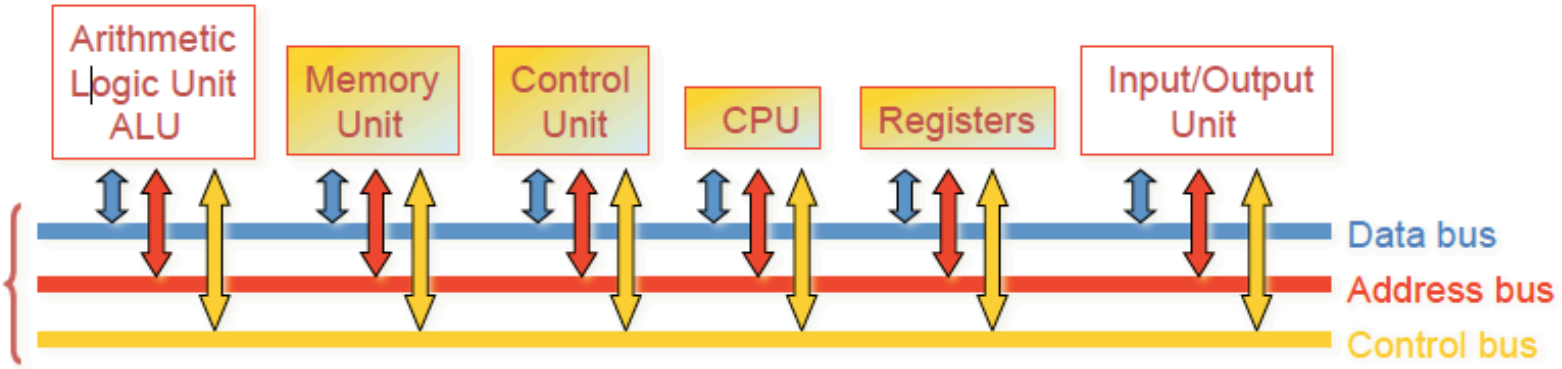
- Gran parte dei processori moderni sono basati sulll'architettura di John von Neumann, che contribuì a svilupparla per l'EDVAC, che può essere considerato il successore dell'ENIAC.

Macchina di von Neumann

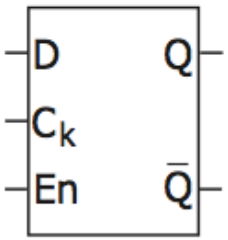


- Nei primi computer la CPU era costituita da un enorme volume di circuiti elettronici;
- Oggi i progressi fatti nel campo dell'integrazione hanno reso possibile racchiudere tutte le funzioni in un unico chip di silicio, in cui sono racchiuse migliaia o decine di migliaia di porte logiche elementari.
- Questi integrati prendono il nome di microprocessori.

Architettura a bus

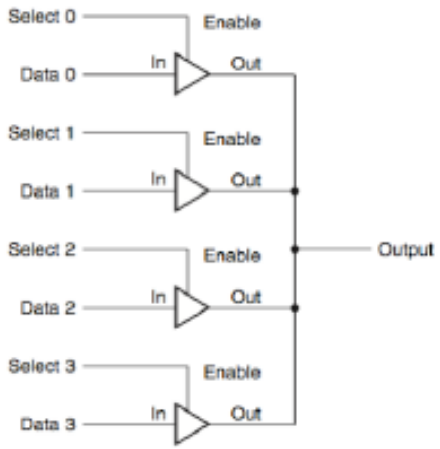


- Le unita' funzionali si scambiano informazioni utilizzando strutture condivise: i *BUS*
- Il BUS e' una collezione di linee elettriche con un protocollo di comunicazione che permette di interpretare correttamente e sincronizzare le varie operazioni di trasferimento dati
- Le singole unita' funzionali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



Enable	D_n	Q_{n+1}
1	0	0
1	1	1
0	x	H

H: alta impedenza



Microprocessori

I computer (e i microprocessori) si contraddistinguono dalla dimensione del bus dei dati e da quella del bus degli indirizzi.

Costruttore	Sigla	Dati (bit)	Indirizzi (bit)
Intel	8080	8	16
Motorola	6800	8	16
Zilog	Z80	8	16
Intel	8086	16	16
Motorola	68000	16	16
Intel	80386	32	32
Motorola	68020	32	32

Tabella 1: *Alcuni microprocessori "storici"*

I computer (e i microprocessori) sono macchine "sincrone": la sequenza delle operazioni è scandita da un clock.

Il microprocessore Zilog Z80

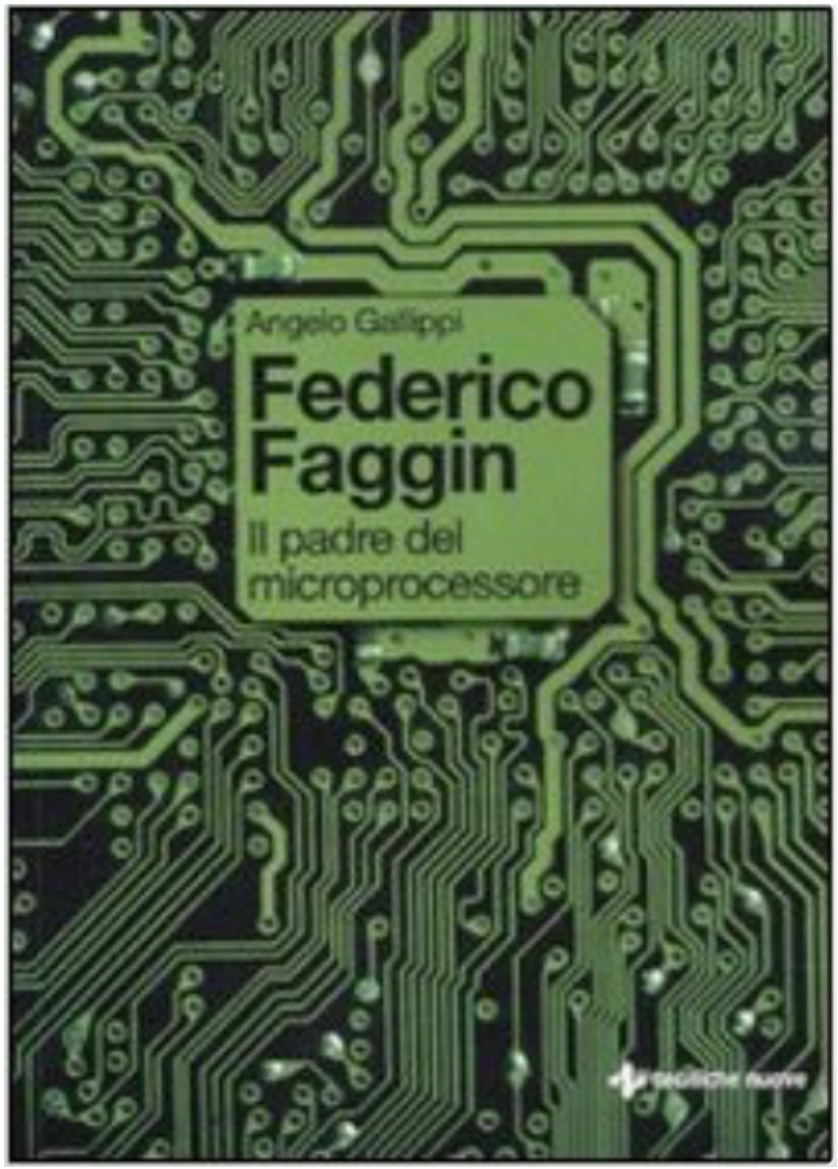
Lo Z80 nacque nel luglio del 1976 per opera di Federico Faggin che, lasciata la Intel dopo aver lavorato sull'8080, aveva fondato la Zilog. Era progettato per offrire compatibilità binaria con l'Intel 8080 in modo che il codice 8080 (in particolare il sistema operativo CP/M) potesse essere eseguito sullo Z80 senza modifiche.

In breve lo Z80 conquistò il mercato dell'8080, e divenne la più popolare CPU a 8 bit di tutti i tempi (e, tenendo in considerazione la dimensione del mercato di allora, la CPU più popolare in generale). Versioni successive dello Z80 ne hanno aumentato la velocità dai 2,5 MHz iniziali fino a 20 MHz.

Viene prodotto in grandi volumi ancora oggi (2009), finora ne sono stati prodotti più di 2 miliardi di unità. Lo si produce ancora e in tali quantità soprattutto per usarlo come microcontrollore o per motivi di istruzione: la semplicità di programmazione, la semplice architettura a 8 bit e la grande quantità di informazioni lo rendono uno strumento perfetto per l'insegnamento dell' elettronica programmata.

Zilog concedeva in licenza il core dello Z80 senza royalty a tutte le aziende che volessero costruire il chip. Questo fece si che il prodotto della piccola azienda guadagnasse consensi nel mercato mondiale, in quanto aziende di gran lunga più grandi come Toshiba iniziarono a produrre lo Z80.

Federico Faggin



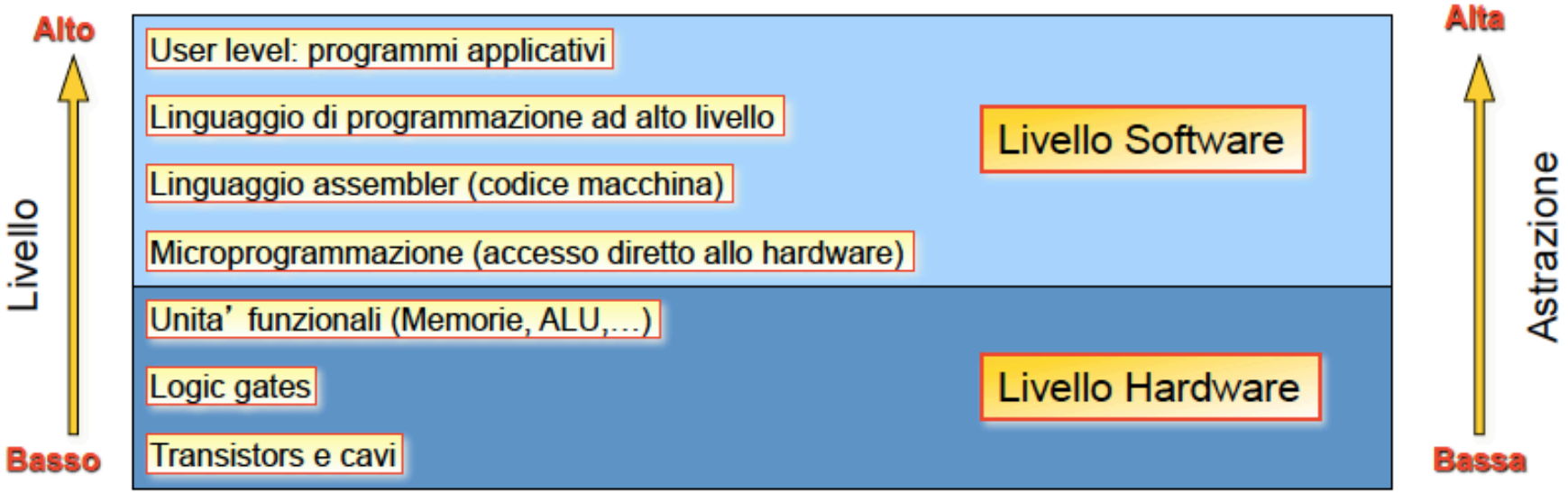
Federico Faggin
il padre del microprocessore

Autore: Angello Gallippi

Da wikipedia

Federico Faggin (Vicenza, 1º dicembre 1941) è un fisico, inventore e imprenditore italiano. Dal 1968 Faggin risiede negli Stati Uniti ed ha assunto anche la cittadinanza statunitense. Fu capo progetto dell'Intel 4004, il primo microprocessore al mondo, e di tutti i primi microprocessori dell'Intel (8008, 4040 e 8080) e creò anche l'architettura del 4040 e dell'8080, il primi microprocessore ad alta prestazione. Fu anche lo sviluppatore della tecnologia MOS con porta di silicio (MOS silicon gate technology), un contributo fondamentale che permise la fabbricazione dei primi microprocessori e delle memorie EPROM e RAM dinamiche e sensori CCD, gli elementi essenziali per la digitilizzazione dell'informazione. Nel 1974 fondó e diresse la ditta Zilog, la prima ditta dedicata esclusivamente ai microprocessori, presso cui dette vita al famoso microprocessore Z80 ancora in produzione fino ai nostri giorni. Nel 1986 Faggin fondò e diresse la Synaptics, ditta che sviluppò I primi Touchpad e Touch screen.

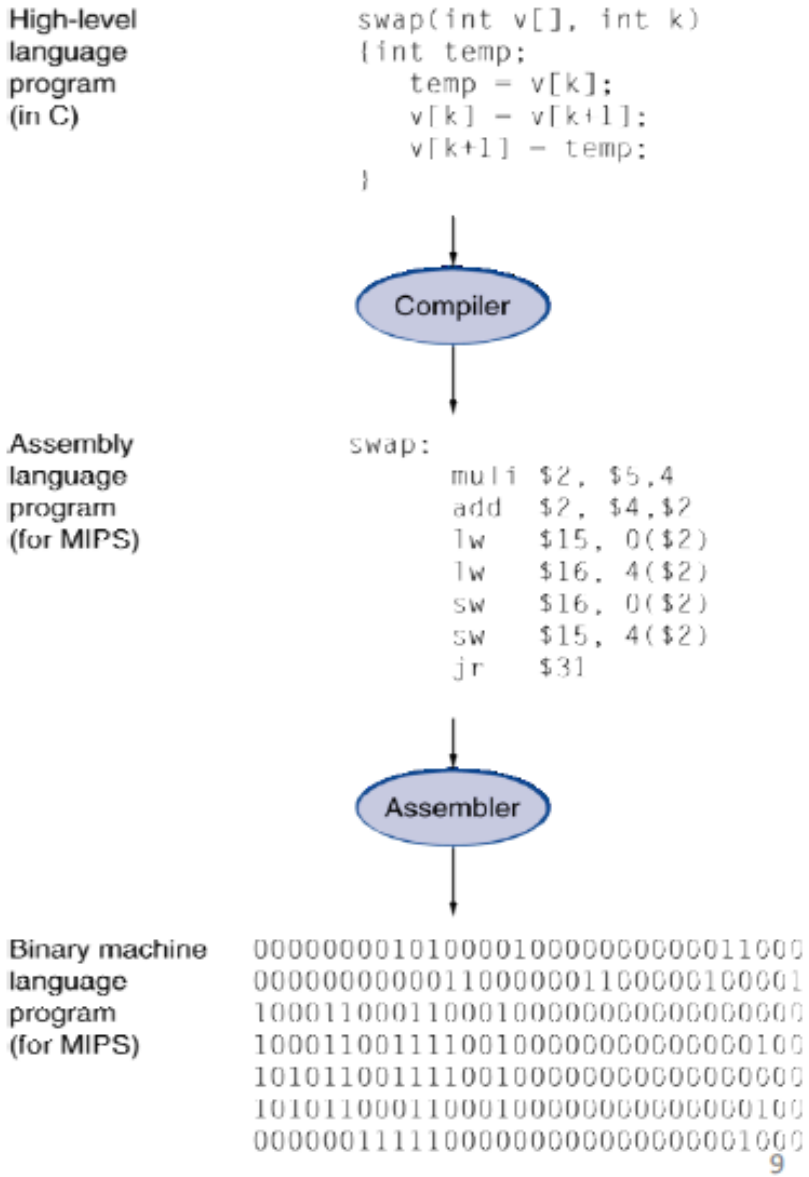
Livelli di astrazione di un architettura di calcolo



- 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

La gerarchia del codice

- Linguaggio di alto livello
  - "User friendly and intelligible"
  - Assicura produttività e (a volte...) portabilita' 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



Quale linguaggio scegliere

- La scelta del linguaggio di programmazione dipende dal tipo di applicazione richiesta.
- In generale si può dire che un linguaggio assembly è utilizzato:
  - nei programmi di piccola-media lunghezza;
  - Nelle applicazioni in cui è importante contenere il costo (un assemblatore richiede molto meno memoria di un compilatore e quindi è più economico);
  - nelle applicazioni molto veloci o in tempo reale (real time)
  - nei casi in cui sono ricorrenti operazioni input/output e di controllo.

- Un linguaggio evoluto, o ad alto livello, è invece usato:
  - nei programmi di media-grande lunghezza;
  - Nelle applicazioni che richiedono una grande capacità di memoria;
  - Nei casi in cui sono richiesti più calcoli che non operazioni di input/ouput o di controllo

- Benché i linguaggi evoluti rendano la programmazione più semplice e più rapida rispetto ai linguaggi a basso livello, essi sono raramente utilizzati nei piccoli sistemi di controllo e applicazioni industriali dove il costo e la velocità sono i fattori più importanti di cui tener conto. È per questo motivo che in questi sistemi sono spesso utilizzati i linguaggi assembly.

- Lo Z80 utilizza l'assembly come linguaggio di programmazione e vedremo anche la sua codifica in linguaggio macchina.

Numerazione esadecimale

- La numerazione binaria è molto pesante (troppe cifre), quindi si preferisce usare la numerazione esadecimale:

Num. decimale	Num. binaria	Num. esadecimale
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10
17	10001	11
...	...	...
255	11111111	FF
256	100000000	100
...	...	

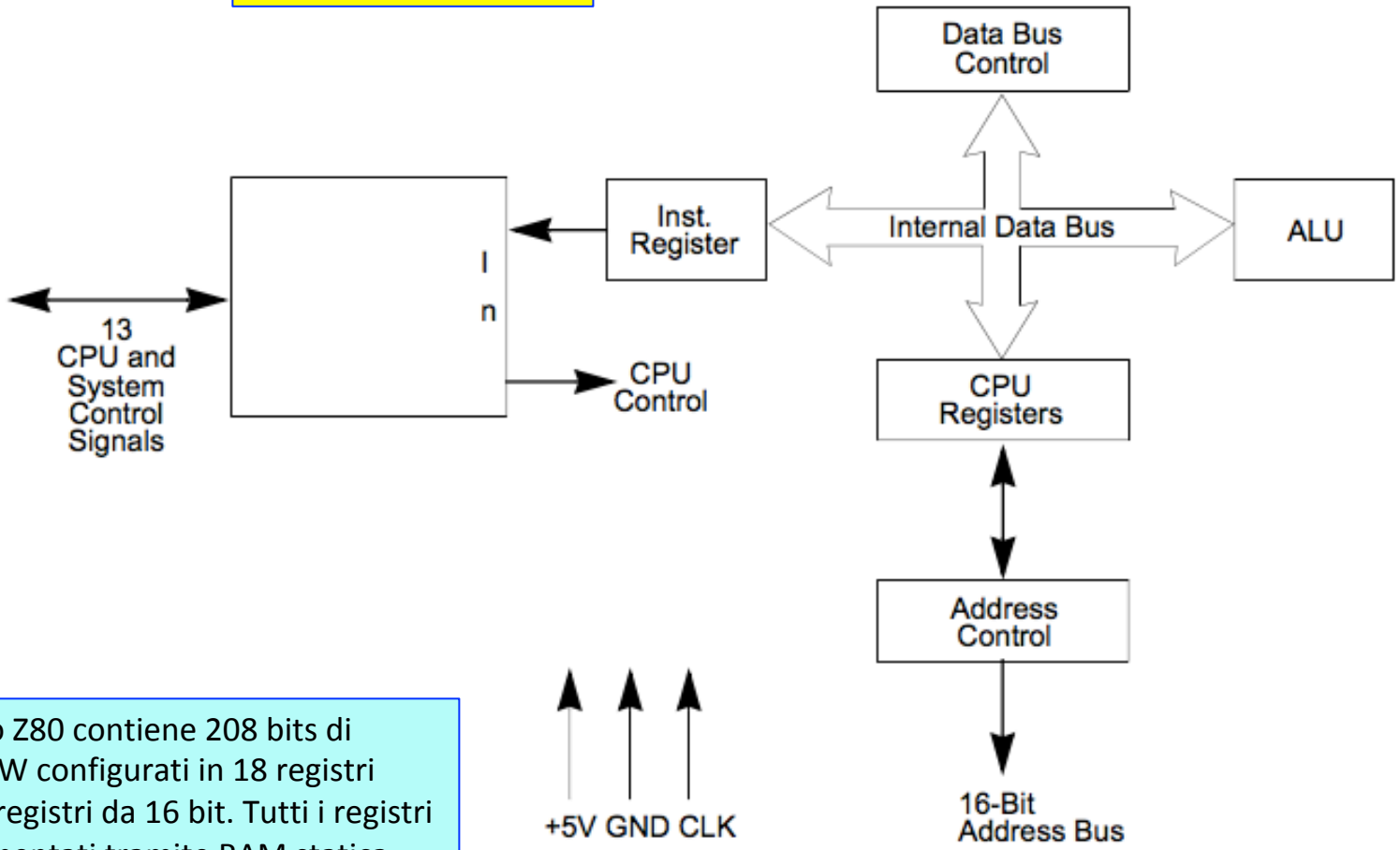
Parole a 8 bit corrispondono a parole a 2 cifre esadecimali

Z80: diagrammi a blocchi della CPU

Z80 CPU
User's Manual

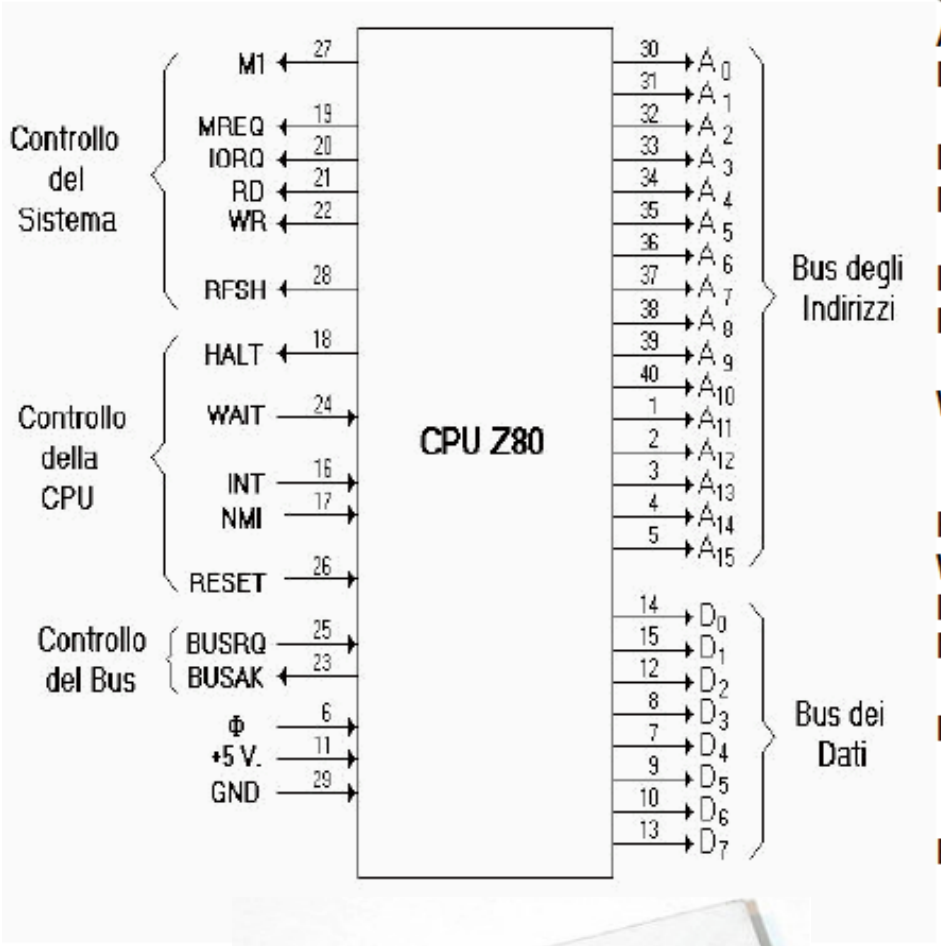
ZiLOG

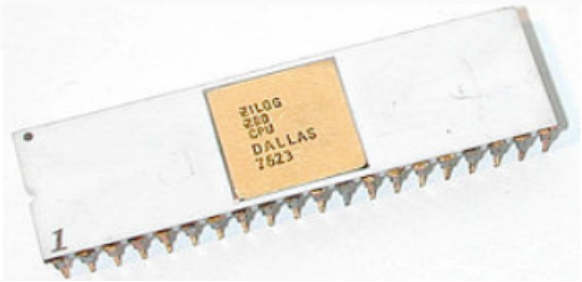
(lo trovate sul sito web)



La CPU dello Z80 contiene 208 bits di memoria R/W configurati in 18 registri da 8 bit e 4 registri da 16 bit. Tutti i registri sono implementati tramite RAM statica.

Z80 Pinout





20 + 20 piedini

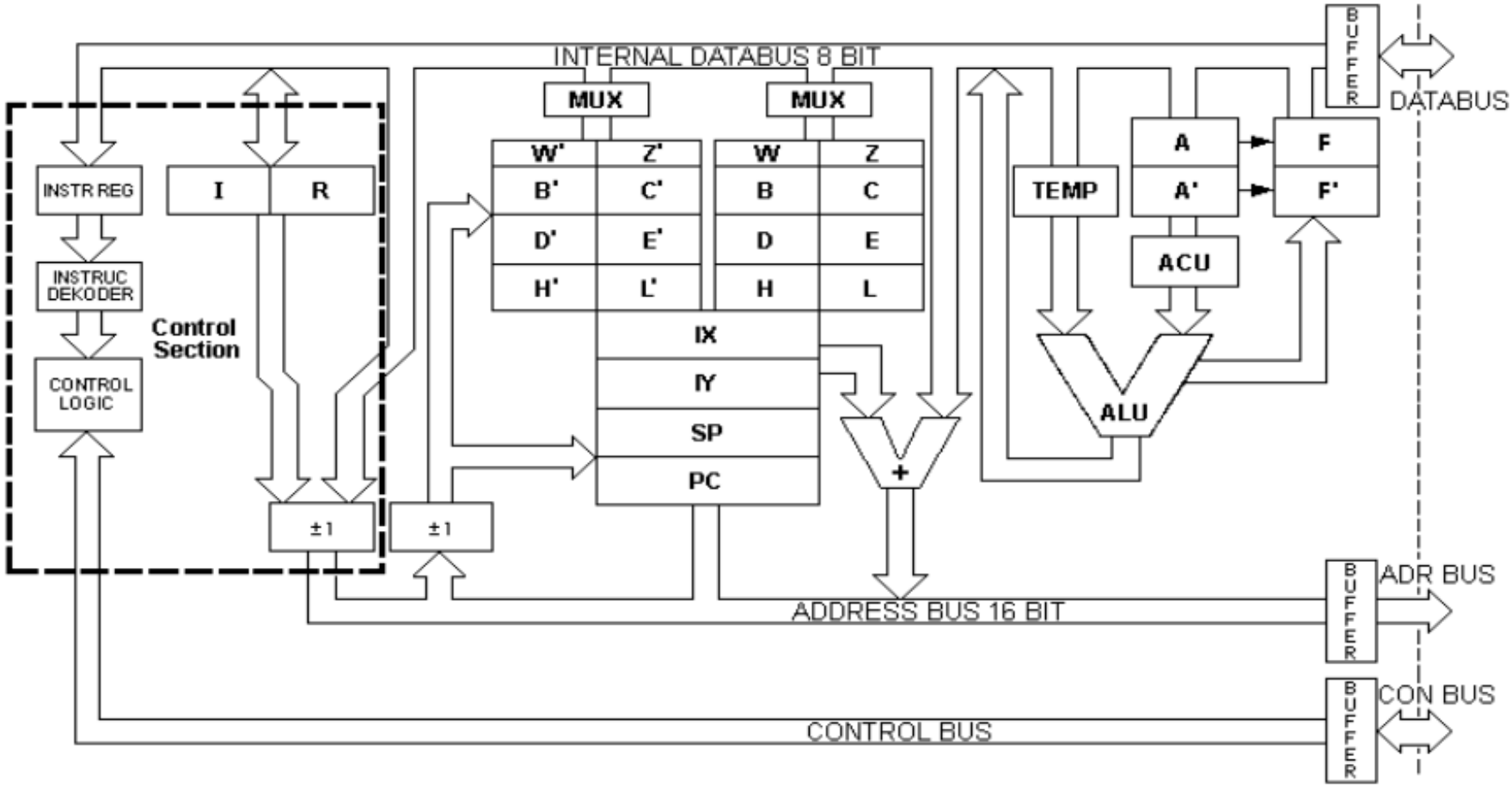
Φ Clock 0 to 4MHz
A0-A15 Bus degli indirizzi,
D0-D7 Bus dei dati (tristate–bidirezionale)

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

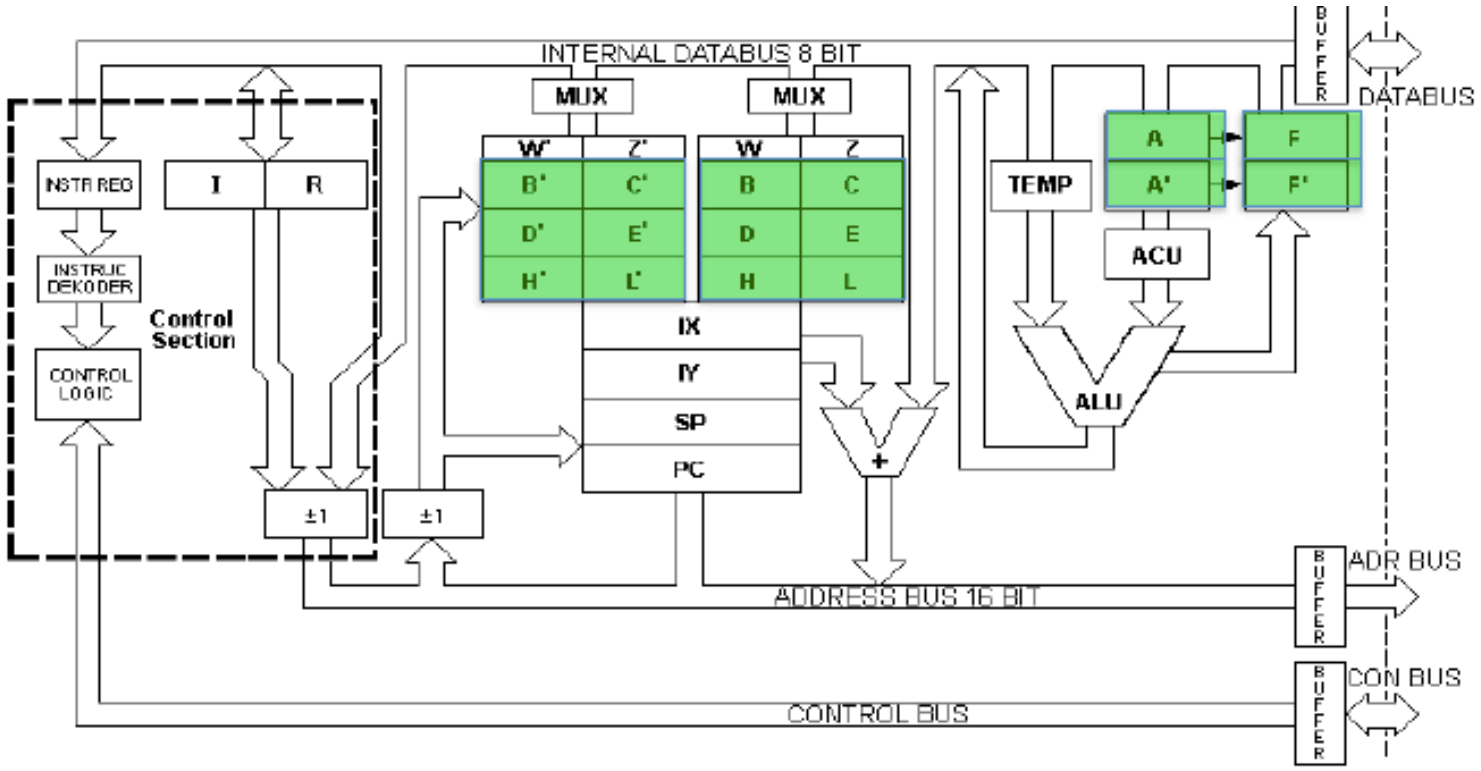
HALT (attivo basso) stato di halt raggiunto
WAIT (attivo basso) La CPU rimane in stato di attesa
INT (attivo basso) Richiesta di interruzione
NMI Richiesta di interrupt di non mascherabile. Costringe la CPU a ripartire da un indirizzo noto (0x66)
RESET (attivo basso) "resetta" la CPU per iniziare le operazioni

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

Z80 schema logico



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)

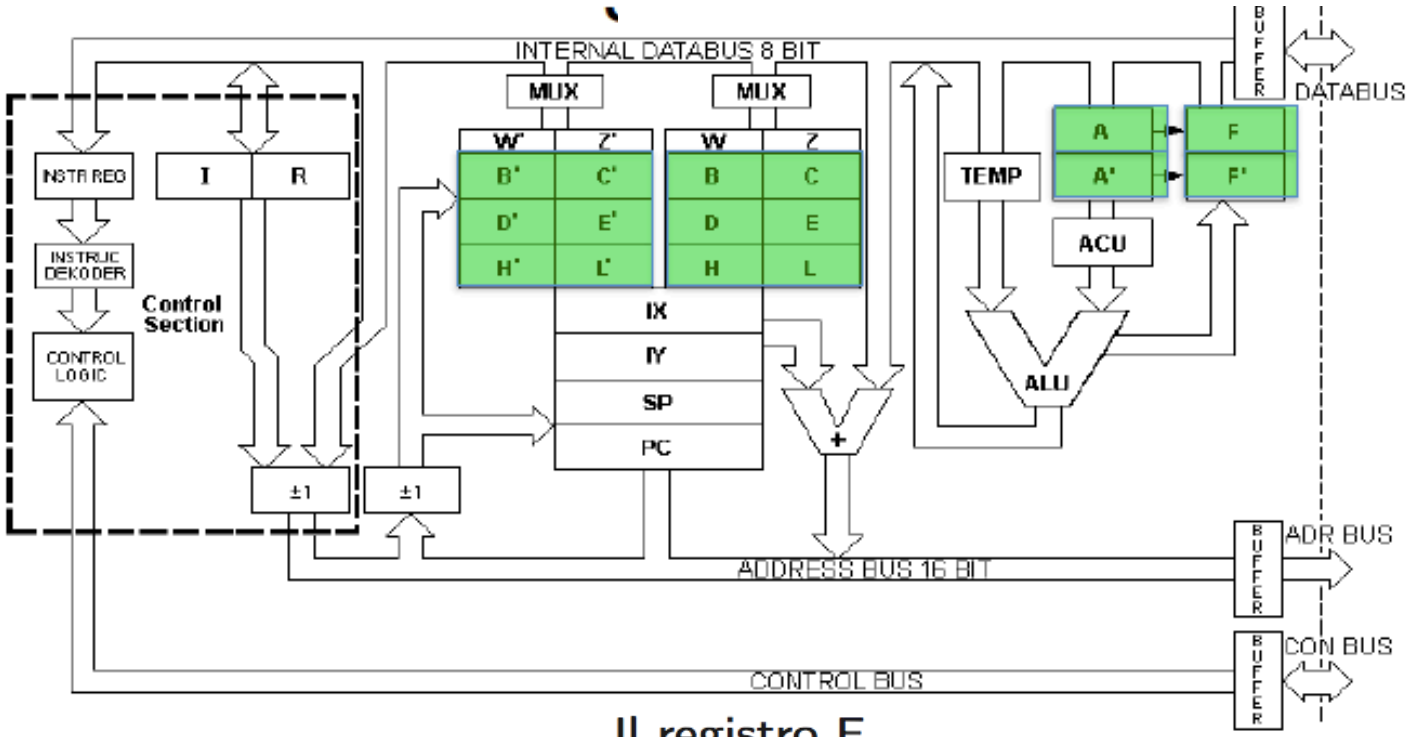
A (accumulatore) risultato dell' alu

F registro di "flag" indicano particolari stati della CPU (Es. non-zero,overflow,…)

Registri "secondari": "context switching" efficente B-> B' C->C'

Registri "speciali" per controllo ed indirizzamento

Z80 registri interni: registro F

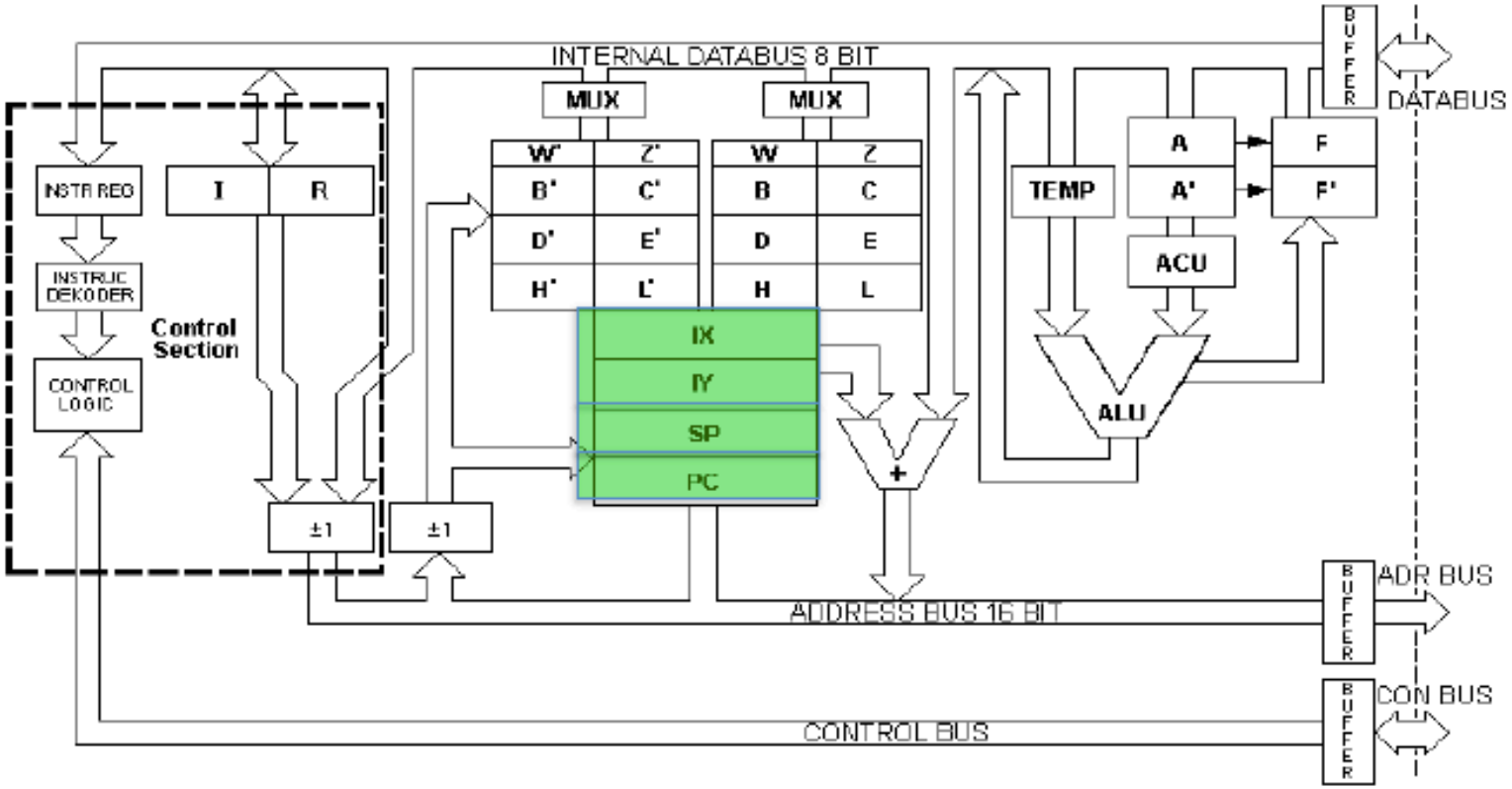


Il registro F

Registro di "Flag"
Indica uno stato particolare della CPU
(esempio: non zero, overflow, etc..)

Bit number	Simbolo	Significato
0	C	Carry
1	N	Add/Subtract
2	P/V	Parity/Overflow
3	X	Non usato
4	H	Half carry
5	X	Non usato
6	Z	Zero
7	S	Sign

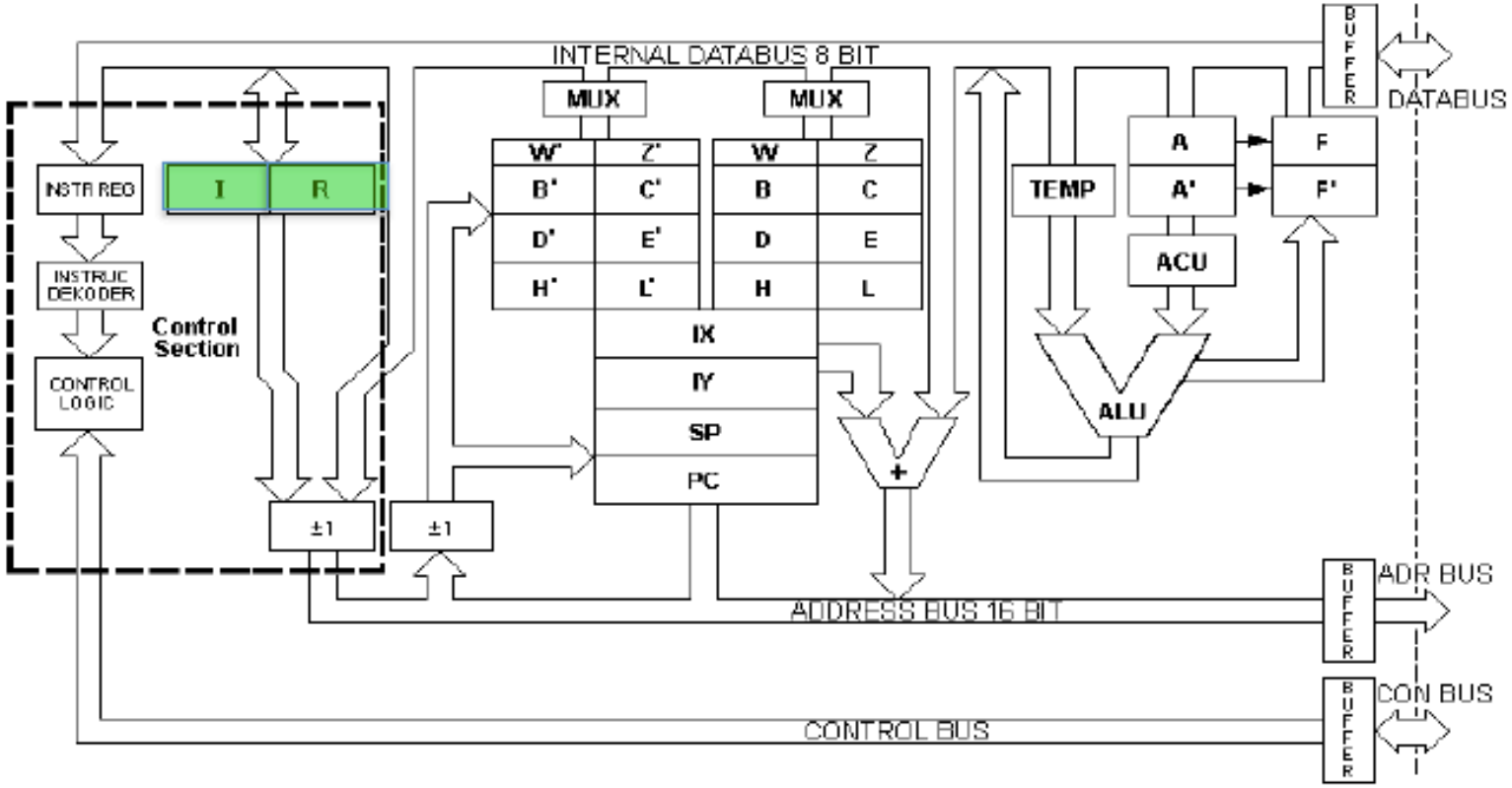
Z80 registri interni



Registri speciali per la gestione dell' indirizzamento dei dati/programma

- **IX,IY** registri "indice" per "indirizzamento indicizzato" (contiene "base address")
- **SP** "stack pointer" per "salto" da programma principale a sub-routine (e ritorno)
- **PC** "program counter" contiene l' indirizzo a 16 bit della istruzione da eseguire

Z80 registri interni

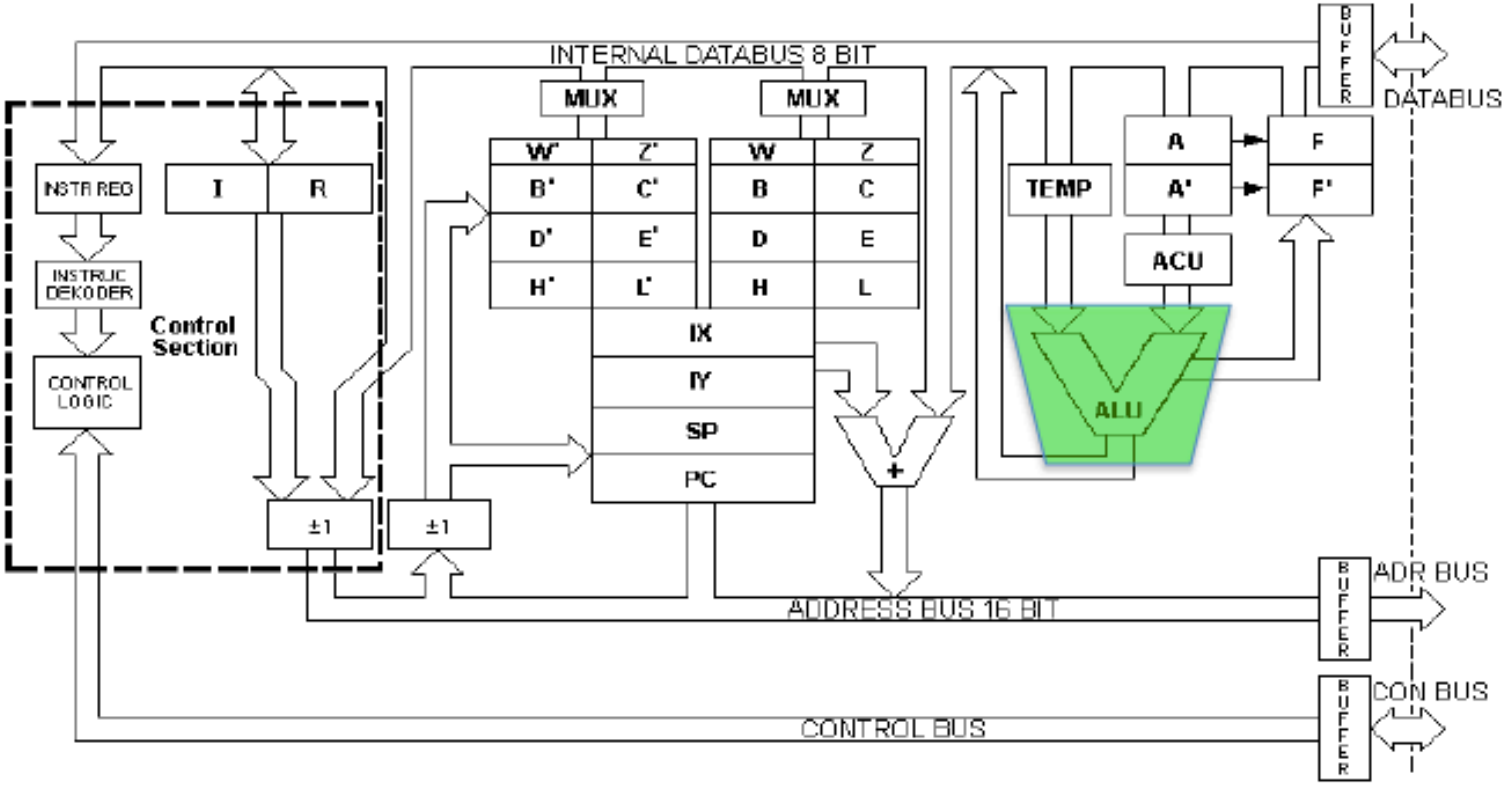


Registri accessori per la gestione della memoria e delle interruzioni

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

Refresh Vector (R) per gestione corretta delle memorie dinamiche

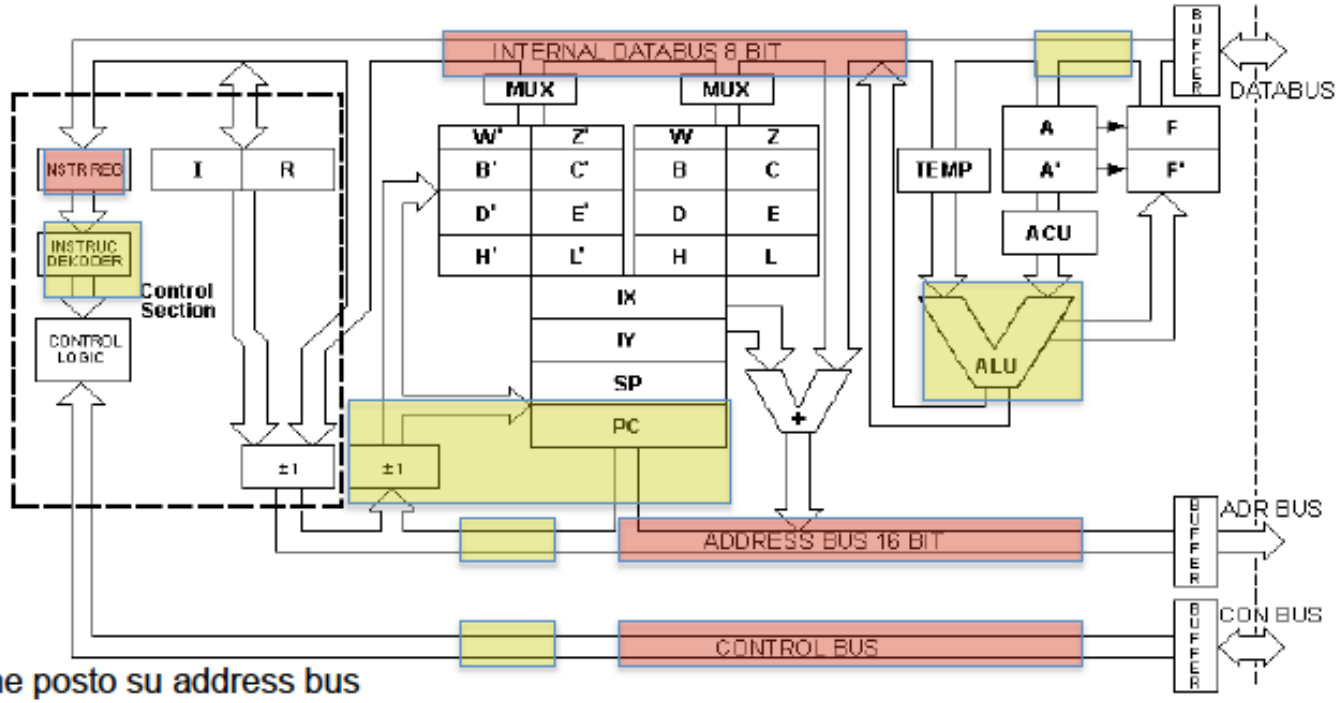
Z80 ALU



ALU (Arithmetic Logic Unit) esegue tutte le operazioni aritmetiche su dati in ingresso

- Somma, Sottrazione, Moltiplicazione,..
- AND logico, OR logico, XOR logico,
- Confronto
- Shift e Rotate (destra e sinistra)
- Incremento, Decremento
- Set, Reset, Test dei bit

Funzionamento dello Z80

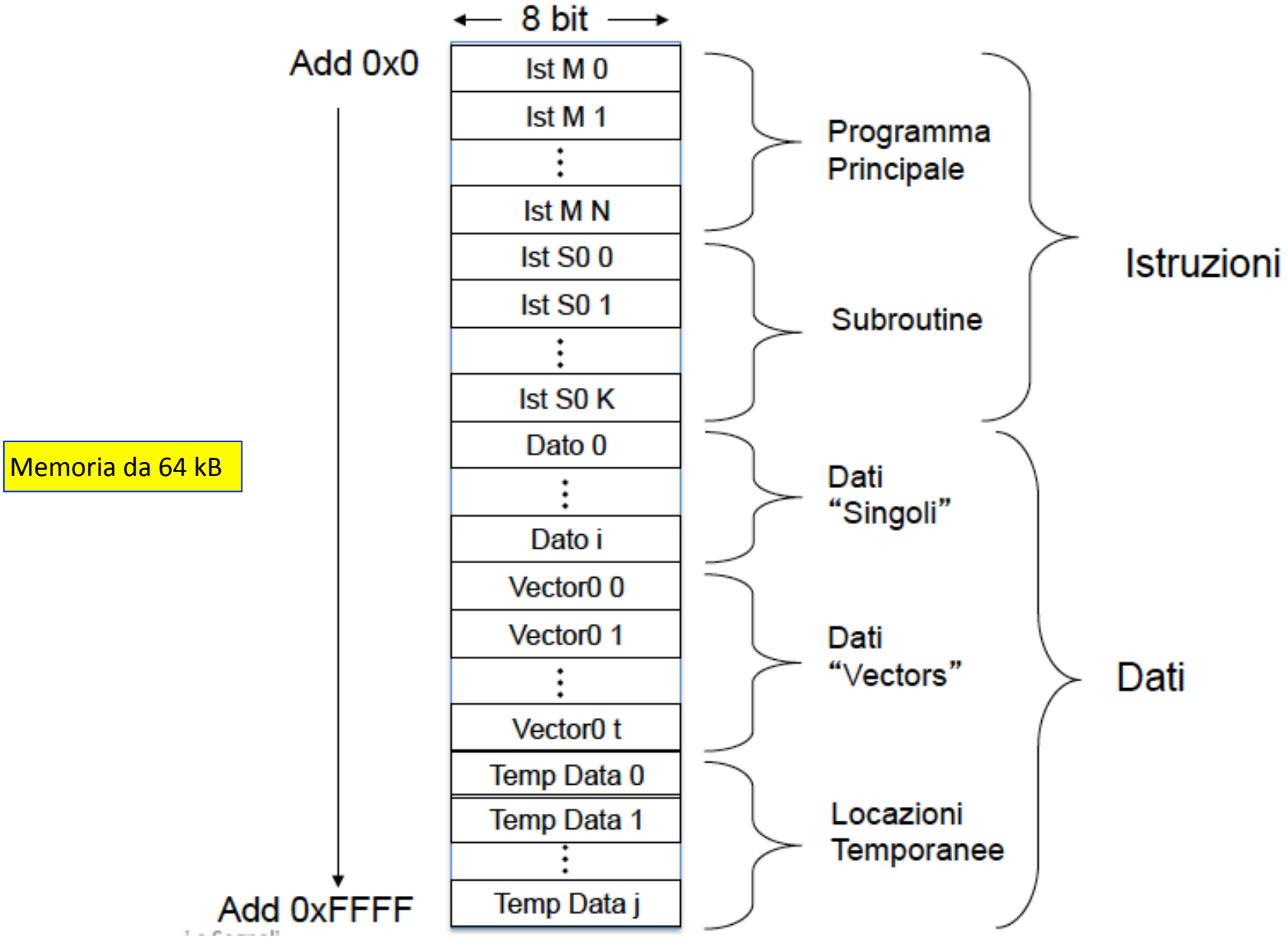


Ciclo di **Fetch**:
1.indirizzo PC viene posto su address bus
2.generazione sul control bus dei segnali necessari a leggere l'istruzione (ad add PC) dalla memoria
3.lettura dell'istruzione dalla memoria e scrittura nell' INSTR REG (via data bus).

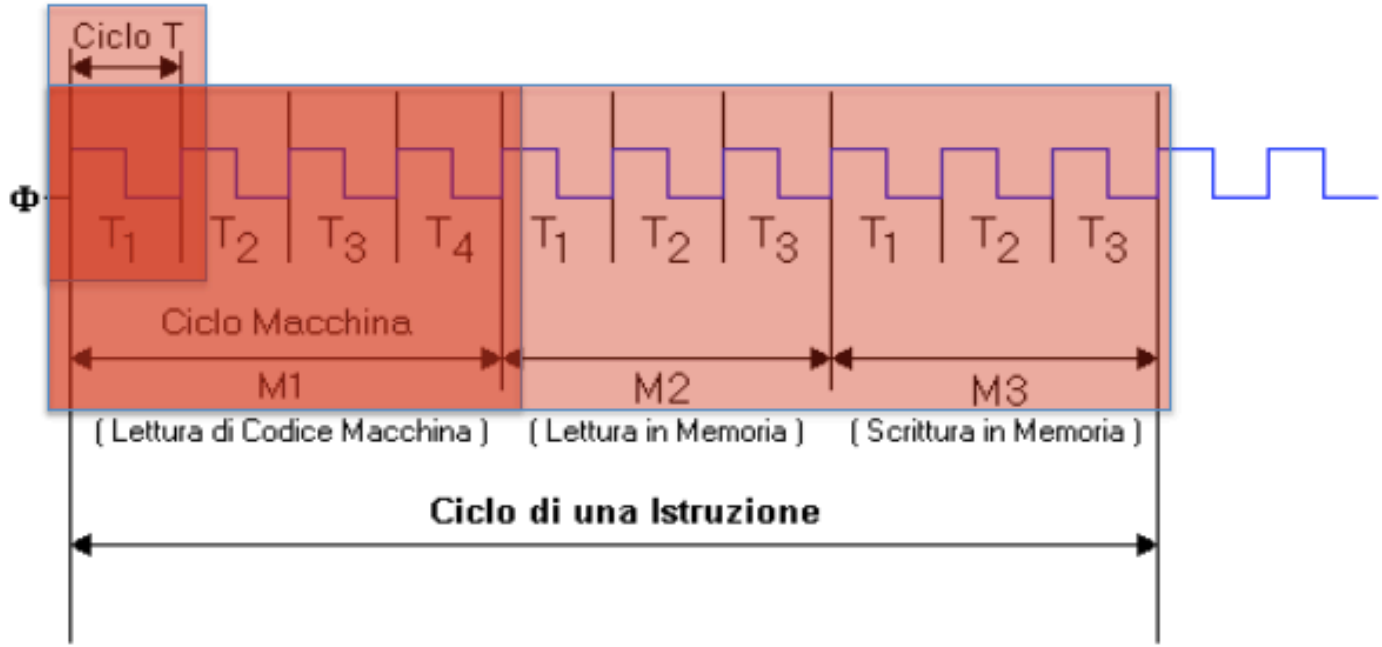
Ciclo di **Execute**:
1.incremento del PC (per prossima istruzione)
2.decodifica dell'istruzione
3.eventuale lettura dei dati
4.esecuzione dell'istruzione

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

Organizzazione della memoria



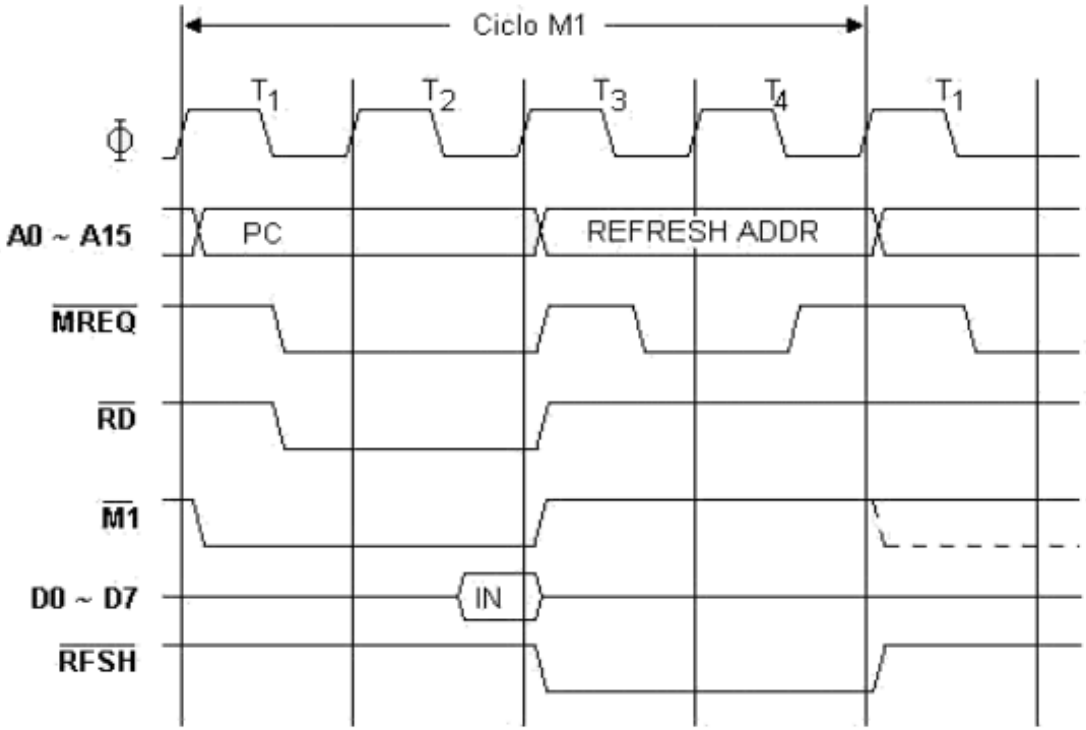
Z80 timing



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)

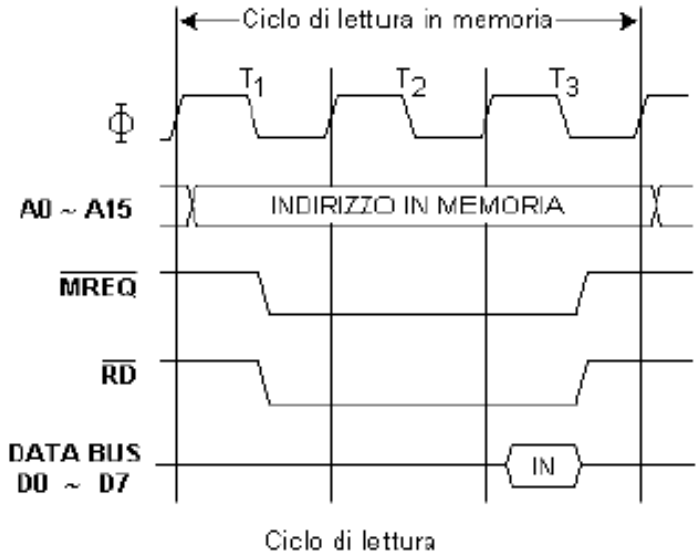
Φ Clock 0 to 4MHz
A0-A15 Bus degli indirizzi,
D0-D7 Bus dei dati (tristate–bidirezionale)

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

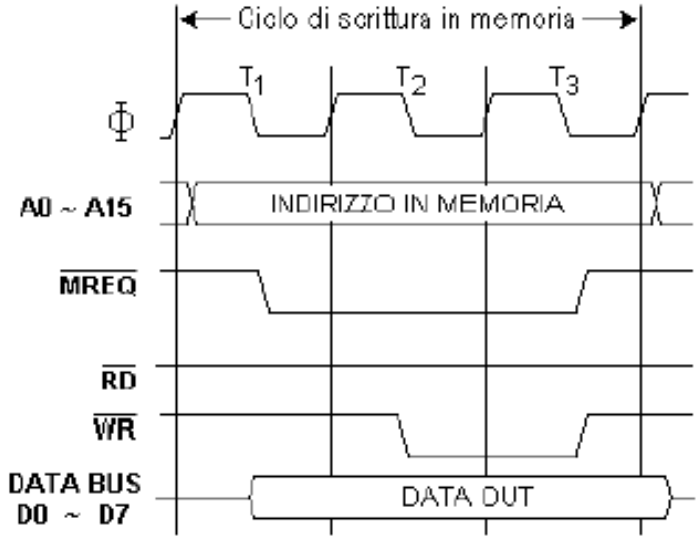
HALT (attivo basso) stato di halt raggiunto
WAIT (attivo basso) La CPU rimane in stato di attesa
INT (attivo basso) Richiesta di interruzione
NMI Richiesta di interrupt di non mascherabile. Costringe la CPU a ripartire da un indirizzo noto (0x66)
RESET (attivo basso) "resetta" la CPU per iniziare le operazioni

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

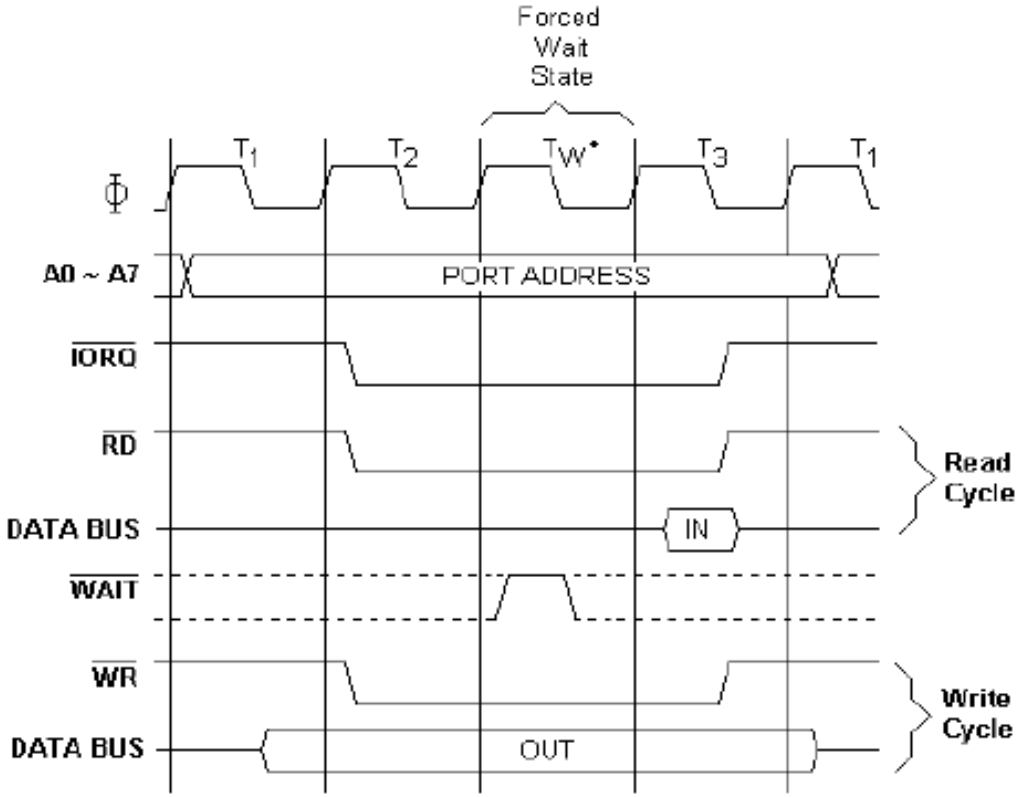
Z80 timing: READ-WRITE-I/O



Ciclo di lettura



Ciclo di scrittura



Programmazione dello Z80

Il microprocessore esegue un compito definito leggendo dalla memoria una serie di istruzioni, che costituiscono il programma.

Fornendo un comando di $RESET$, lo Z80 comincia ad eseguire il programma partendo dalla locazione 0000.

Esempio: loop infinito

Indirizzo	Istruzione	Mnemonico	Commento
0000	00	NOP	Non fa niente
0001	C3	JP 0000	Salta alla locazione 0000
0002	00		
0003	00		

Assembly (o assembler)

Le istruzioni devono essere date in linguaggio macchina, cioe' in codice binario: e' quindi utile al programmatore fare ricorso ad una forma mnemonica per indicare ogni istruzione, e scrivere anche delle note.

Non tutte le istruzioni dello Z80 si esauriscono in un byte; ci sono invece istruzioni piu' complesse che richiedono 2,3 o anche 4 bytes. Di conseguenza l'istruzione deve essere memorizzata in piu' locazioni di memoria consecutive.

Programmazione dello Z80

Esempio: decrementa il registro fino a zero

Indirizzo	Istruzione	Mnemonico	Commento
0000	06	LDB, 64	Carica nel registro B il numero
0001	64		esadecimale 64
0002	05	DEC B	Decrementa il registro B di 1
0003	C2	JPNZ, 0002	Salta alla locazione 002 se
0004	02		l'ultima operazione non ha dato
0005	00		risultato zero
0006	76	HALT	Si ferma

Questo programma esegue un loop 64 volte (esadecimale) e poi si ferma. In esso abbiamo usato istruzioni lunghe 1,2 e 3 bytes. Abbiamo caricato in B un numero e successivamente lo abbiamo decrementato a passi di 1, eseguendo ogni volta un test per verificare il contenuto del registro. Questo tipo di istruzioni usa proprio il bit 6 del registro F, che viene posto ad 1 quando l'ultima operazione fatta (in questo caso DEC B) fornisce un risultato zero. Si noti poi il modo con cui abbiamo dato l'indirizzo nell'istruzione JPNZ, 0002: prima il byte meno significativo (02) e poi quello piu' significativo (00). Questa convenzione e' del tutto generale e deve essere seguita in tutte le istruzioni che contengono un indirizzo.

Programmazione dello Z80

Esempio: somma di due numeri

Indirizzo	Label	Istruzione	Mnemonico	Commento
0100		3A	LDA,(0200)	Carica in A
0101		00		il primo addendo
0102		02		
0103		2A	LDHL, 0201	Carica in HL
0104		01		l'indirizzo del
0105		02		secondo addendo
0106		86	ADDA,(HL)	Somma
0107		32	LD (0202),A	Scrive in memoria
0108		02		il risultato
0109		02		
010A		76	HALT	

Il programma e' memorizzato alla locazione 0100; gli operandi nelle locazioni di memoria 0200 e 0201, mentre il risultato viene salvato nella locazione 0202.
Qui abbiamo usato varie tecniche di indirizzamento
Come facciamo a mettere in esecuzione questo programma?

Indirizzo	Label	Istruzione	Mnemonico	Commento
0000		C3	JP, 0100	Salta a 0100
0001		00		
0002		01		

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)
  DEC H (1 operando <-> 2 byte)
  ADD A,1 (2 operandi <-> 3 byte)

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

  Es: NOP -> 0x00
  INCB -> 0x04

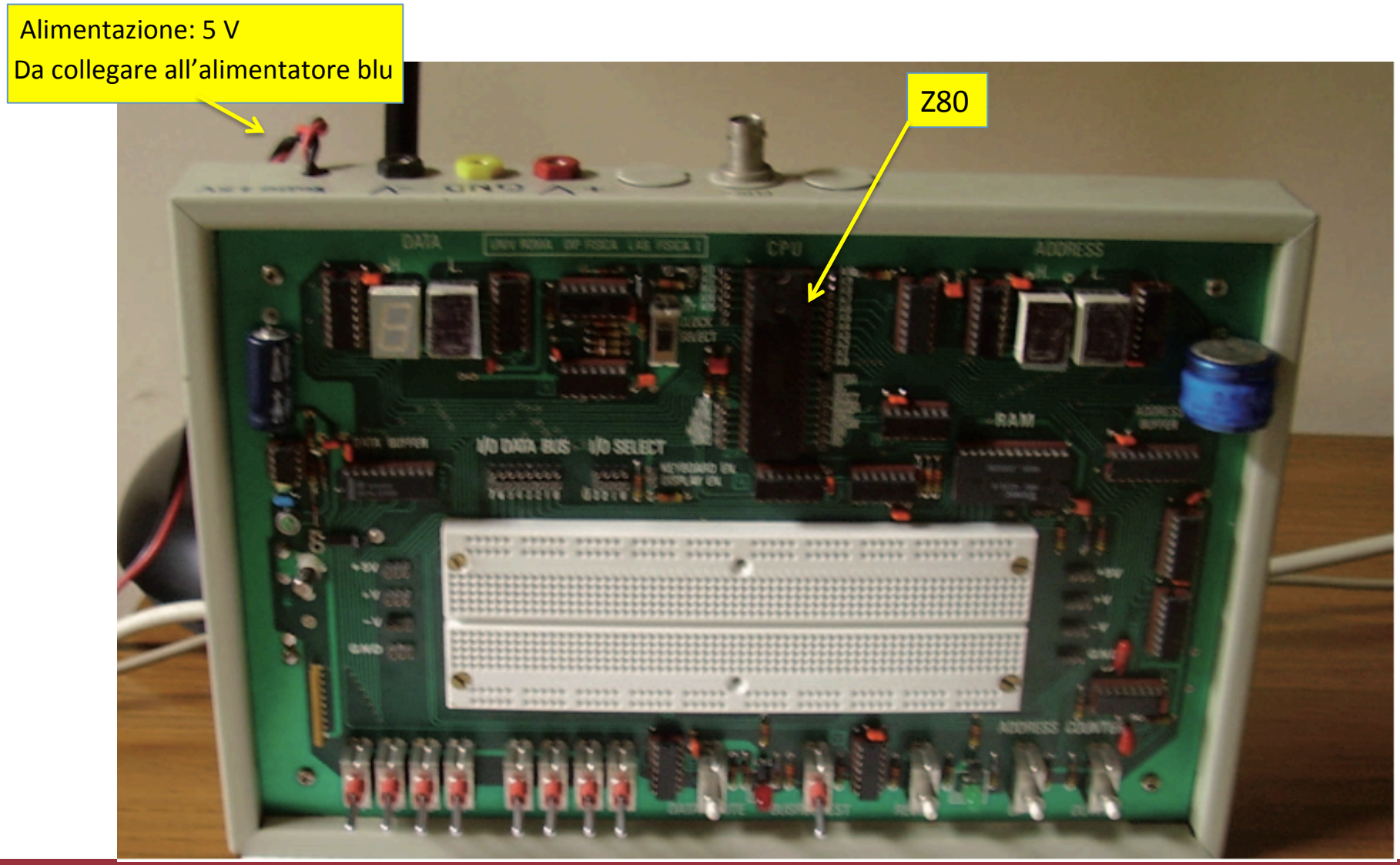
Z80 Instruction set table

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	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
1	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
2	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
3	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
4	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
5	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
6	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
7	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
8	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
9	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
A	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	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
C	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
D	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
E	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
F	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

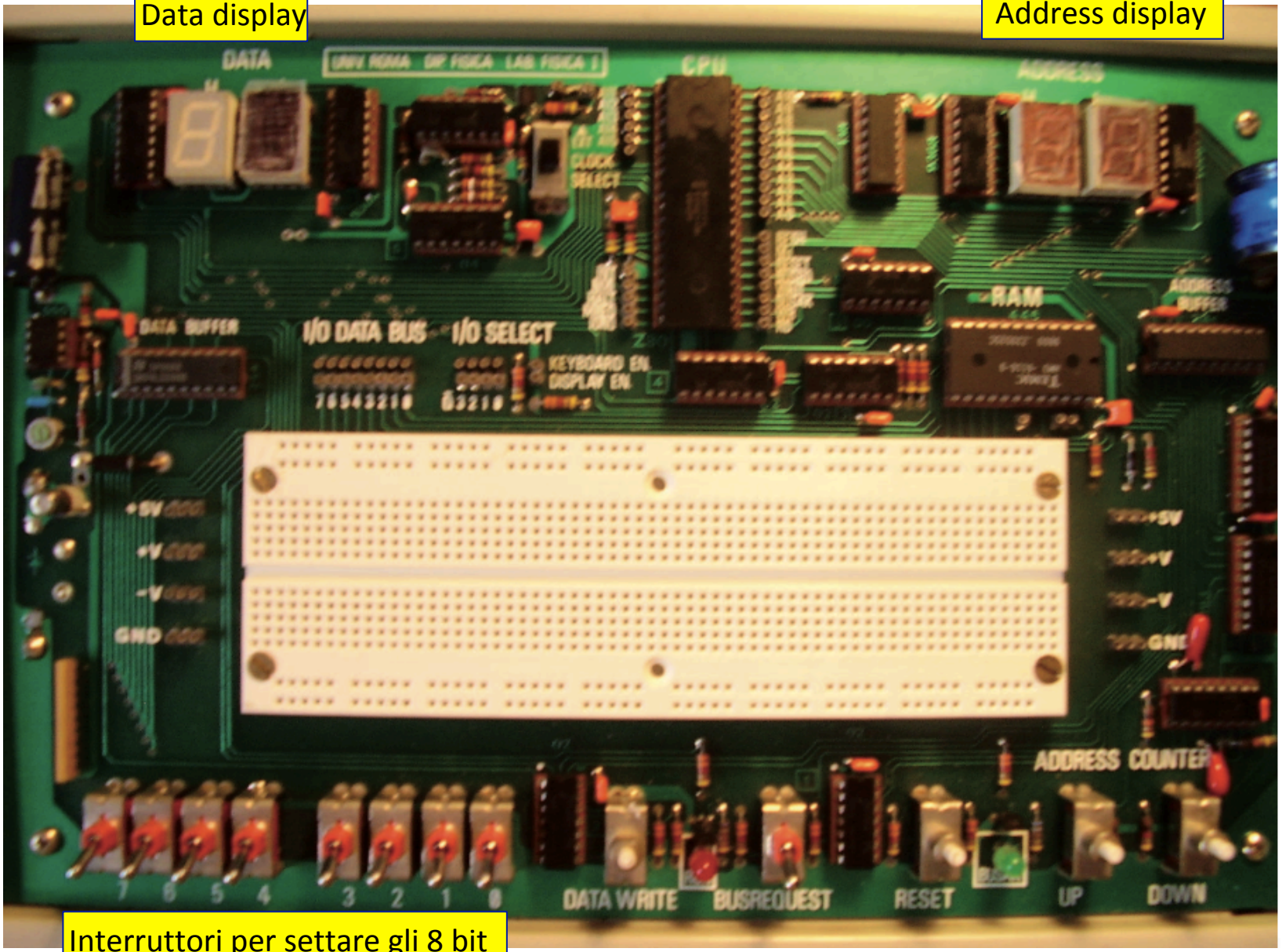
(1) Primo byte di una istruzione con codice a piu' bytes;
(*): L'istruzione si completa con un byte;
(**): L'istruzione si completa con due bytes

La scheda didattica Z80

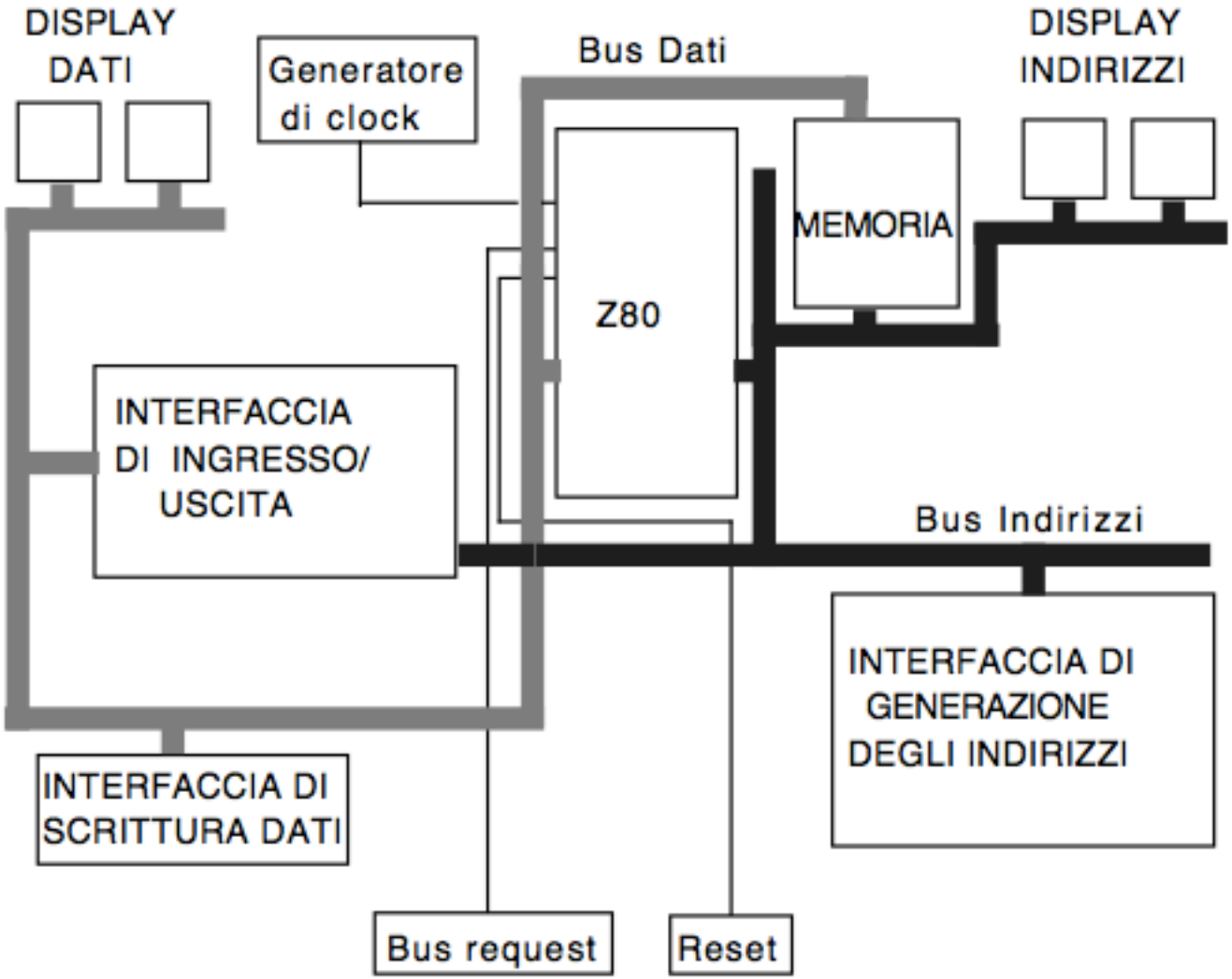
- Questa scheda fu realizzata dallo studente I.Vannucci come progetto nell'ambito del corso di Laboratorio di Fisica 2 (un esame del quarto d'anno di Fisica) nel 1986



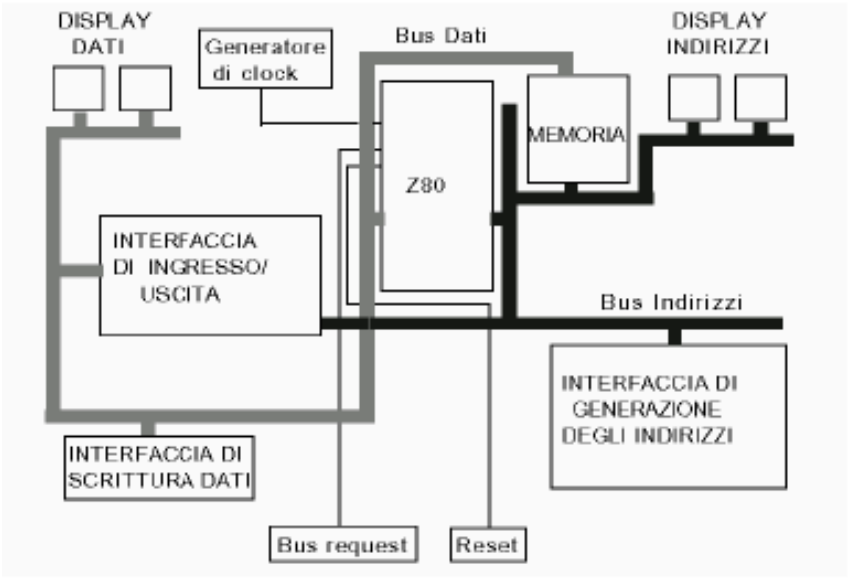
La scheda didattica Z80



Scheda didattica Z80: schema a blocchi



Scheda didattica Z80: schema a blocchi



Due stati di funzionamento:

RUN mode

SYSTEM mode

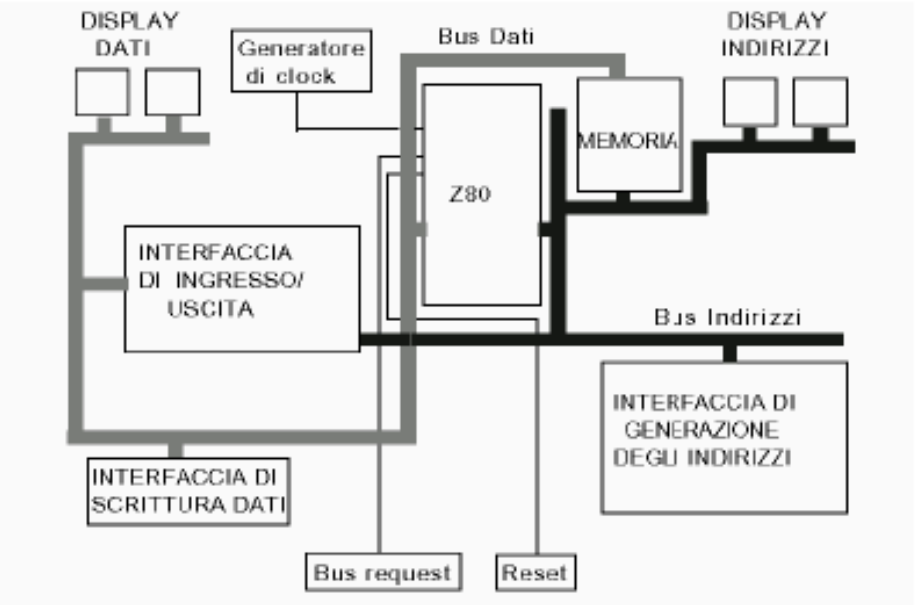
SYSTEM mode: (BUS request attivo)

- 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

Scheda didattica Z80: schema a blocchi



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.
Puo' 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: 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;

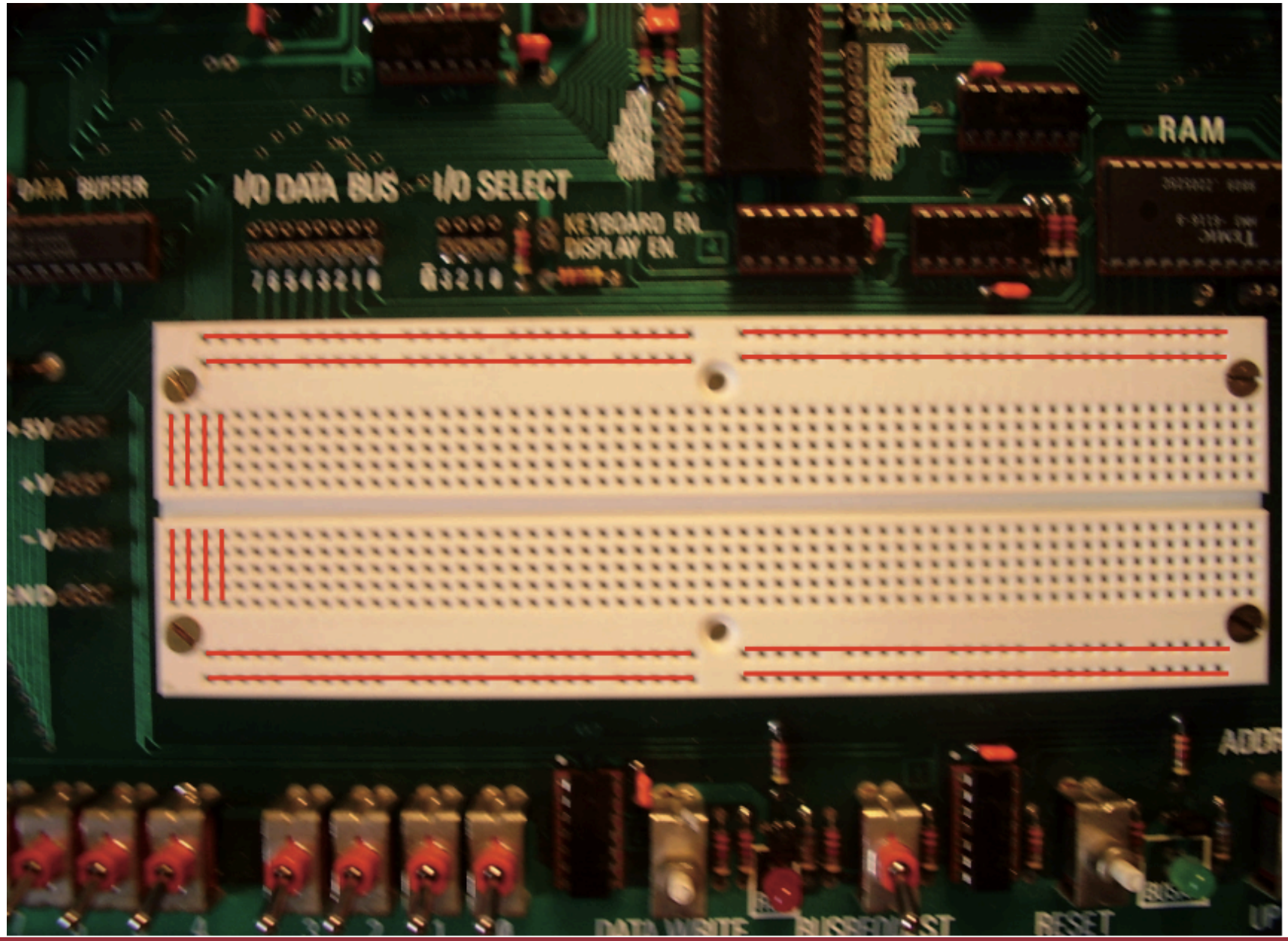
6)Ripetere la sequenza 3-4-5 fino al termine del programma;

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:

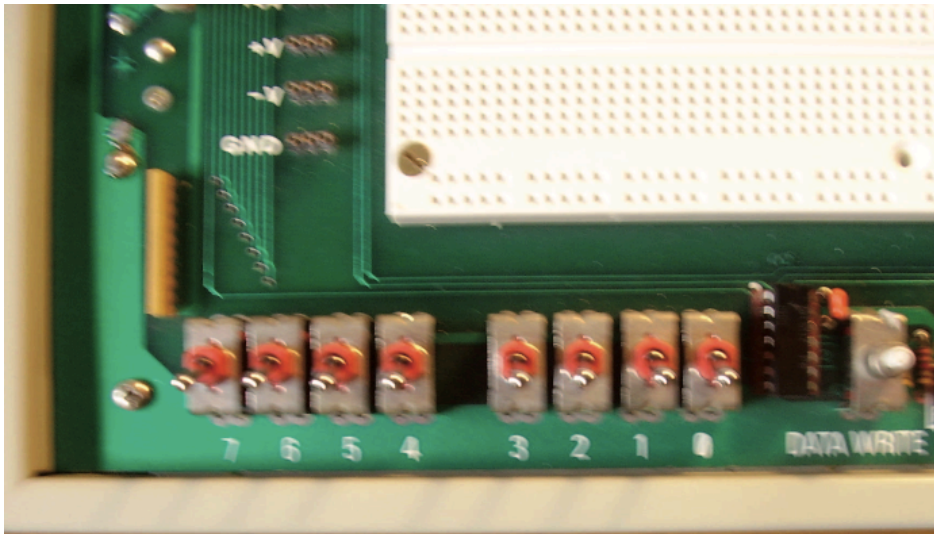
8)Restituire i bus alla CPU mediante l'interruttore BUSREQUEST (il led verde si spegne);

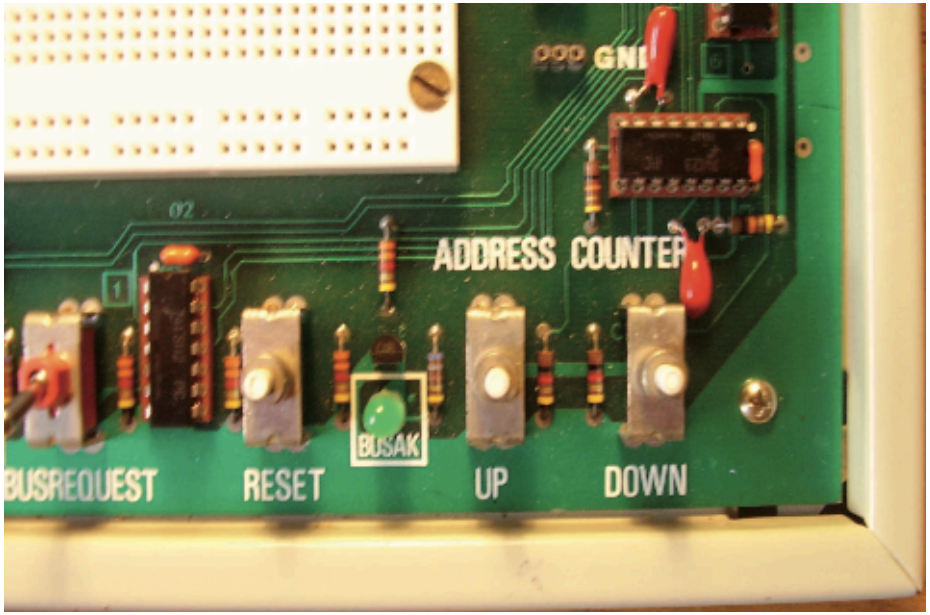
9)Premere momentaneamente il pulsante di RESET; si accende il led rosso di RUN e la CPU cerca la prima istruzione da eseguire in 0000_{HEX}.

La scheda didattica Z80



La scheda didattica Z80





1)Prendere il controllo del bus mediante l'interruttore **BUSREQUEST**; si ha il controllo quando é acceso il led verde **BUSAK**;

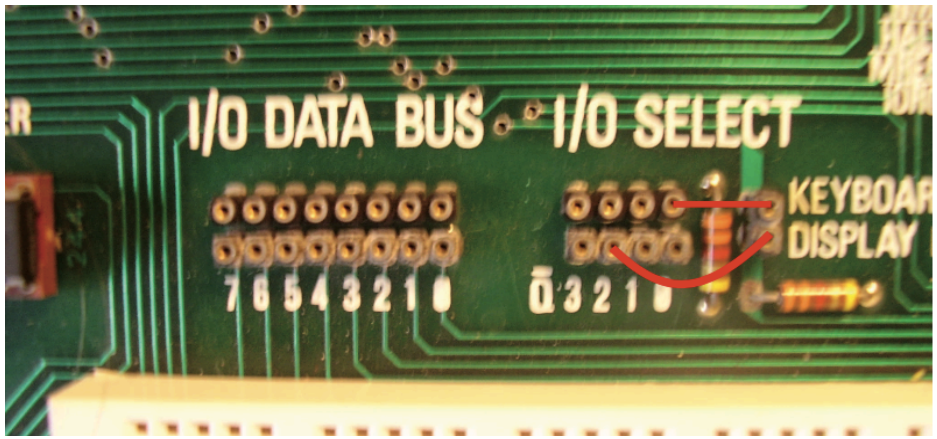
2)Mediante i pulsanti **UP o DOWN** posizionare il contatore degli indirizzi (ADDRESS COUNTER) nella locazione di memoria da cui si desidera far partire il programma; l'indirizzo relativo appare sul visualizzatore degli ADDRESS in forma esadecimale (nibble High e Low).

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;

La scheda didattica Z80

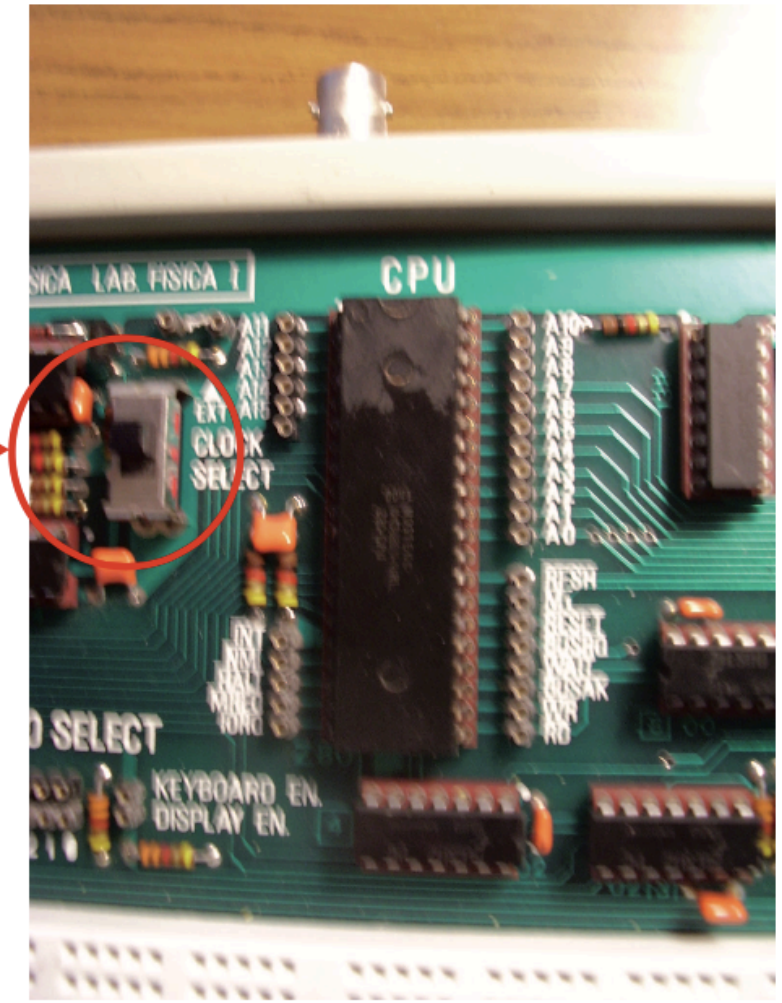


Le periferiche utilizzate, a cui viene assegnato un nome simbolico, sono i visualizzatori dei dati (uscita) e gli interruttori di predisposizione dei dati (ingresso). Per connettere in hardware tali periferiche é necessario (si veda lo schema elettrico):

- inserire un ponticello tra l'uscita desiderata (Q0-Q3) del decoder connessa al connettore I/O SELECT e il piedino del connettore KEYBOARD EN. ; viene quindi abilitata la periferica di ingresso;
- inserire un ponticello tra l'uscita desiderata (Q0-Q3) del decoder connessa al medesimo connettore e il piedino del connettore DISPLAY EN. ; viene quindi abilitata la periferica di uscita.

La scheda didattica Z80

SELEZIONE DEL CLOCK



- **Generatore di clock**

Clk interno 1 MHz, ingresso per Clk esterno

Spiegazione esperienza Z80

- **Verifica della temporizzazione dei comandi e cicli di istruzione.**
- **Programma di temporizzazione.**
- **Scrittura di un programma di temporizzazione per cambiare il tempo di esecuzione**
- **Esempio di input/output**

Verifica della temporizzazione

- **Scrivere un loop infinito.**

indirizzo	dato	istruzione	commento
00	00	NOP	nessuna operazione
01	C3	JP0001	salto incondizionato
02	01		all'indirizzo 0001
03	00		

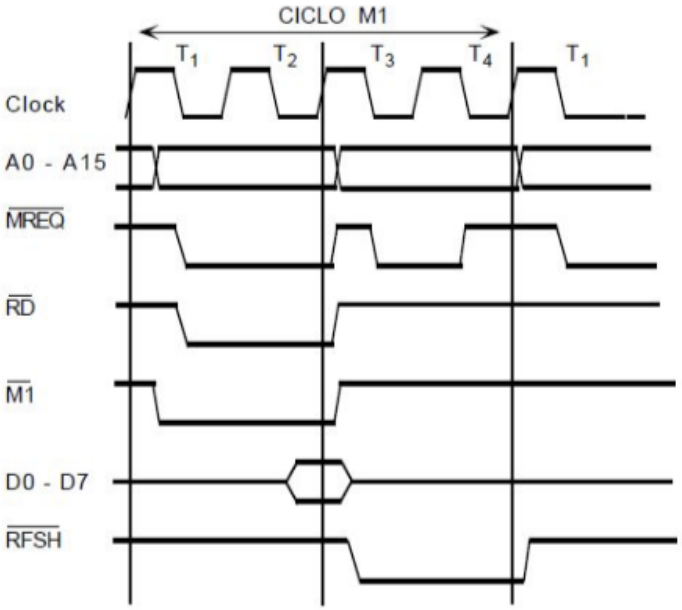
Dopo aver memorizzato il programma farlo eseguire con il clock interno ad 1 MHz. Sincronizzare esternamente l'oscilloscopio a doppia traccia con il segnale M1*; visualizzare su una traccia il clock della CPU (pin 6) e sull'altra traccia i seguenti segnali:

- MI* (pin 27) ciclo di fetch del codice operativo
- MREQ* (pin 19) richiesta di accesso in memoria
- RD* (pin 21) lettura della memoria
- D0 (pin 14) dato meno significativo
- A0 (pin 30) indirizzo meno significativo
- RFSH* (pin 26) segnale di refresh

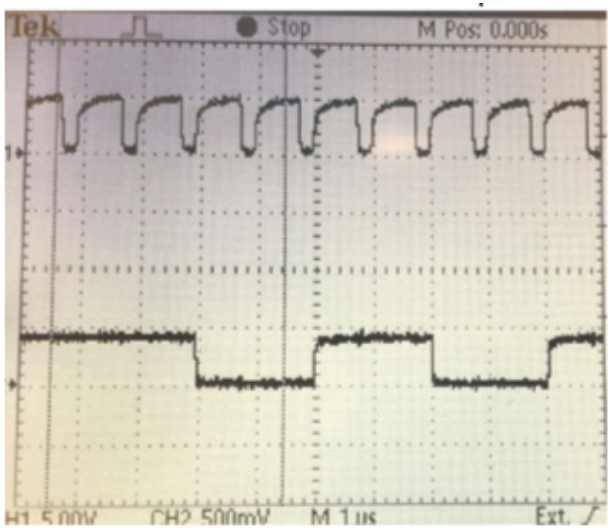
Studiare l'andamento, la fase ed i tempi rispetto al clock, delle forme d'onda osservate.

Utilizzando invece un clock esterno a bassissima frequenza é possibile osservare lo svolgimento del programma e l'esecuzione di ogni istruzione sul visualizzatore degli indirizzi.

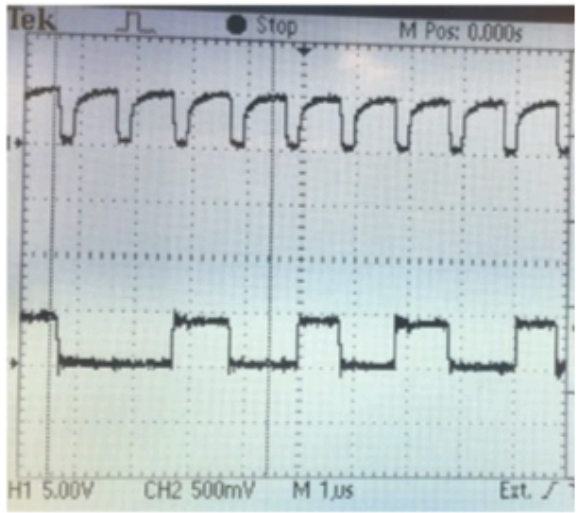
Esempio di cosa si dovrebbe vedere



MI* (pin 27) ciclo di fetch del codice operativo
MREQ* (pin 19) richiesta di accesso in memoria
RD* (pin 21) lettura della memoria
D0 (pin 14) dato meno significativo
A0 (pin 30) indirizzo meno significativo
RFSH* (pin 26) segnale di refresh

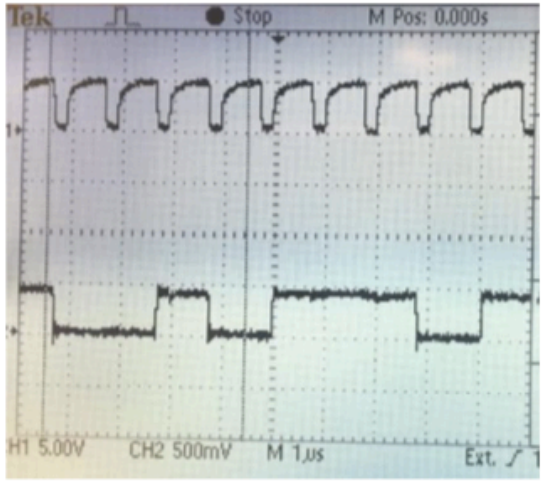


Andamento misurato del segnale MI.

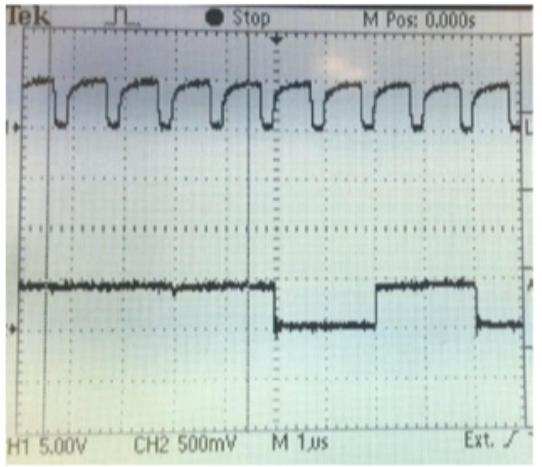


Andamento misurato del segnale MREQ.

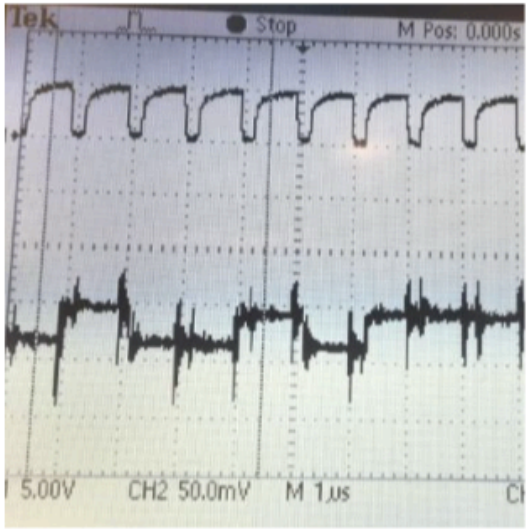
Esempio di cosa si dovrebbe vedere



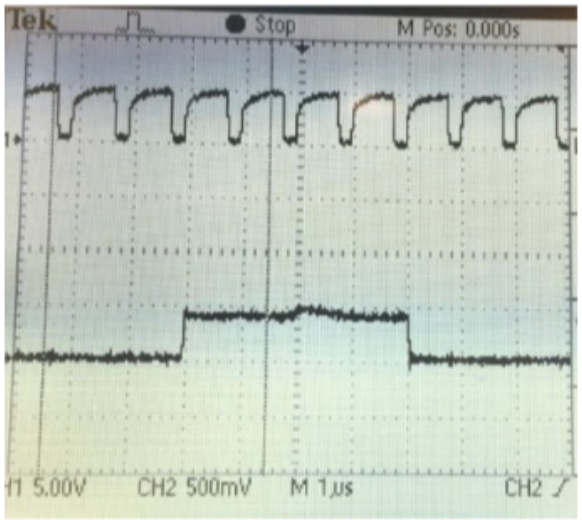
Andamento misurato del segnale RD.



Andamento misurato del segnale RFSH



Andamento misurato del segnale D0.



Andamento misurato del segnale A0.

Programma di temporizzazione

Programma di temporizzazione: (non c'e' sulla guida)
Questo programma é frequentemente usato per effettuare operazioni ad intervalli uniformi di tempo; per il conteggio viene usato il registro B.

indirizzo	Dati	label	istruzione	commento
00	06 64		LD B,100	carica il registro B con 100 (decimale)
02	05	LOOP:	DEC B	decrementa B : B=B-1
03	C2 02 00		JP NZ,LOOP	salto indietro se il risultato non é zero,
06	76		HALT	altrimenti ferma

Dopo aver memorizzato il programma, farlo eseguire con il clock interno ad 1 MHz.
Al termine dell'esecuzione il led rosso di RUN collegato al pin 18 (HALT*) della CPU si spegne.

Il tempo di esecuzione del programma e':

$$100 * (10 + 4) * T$$

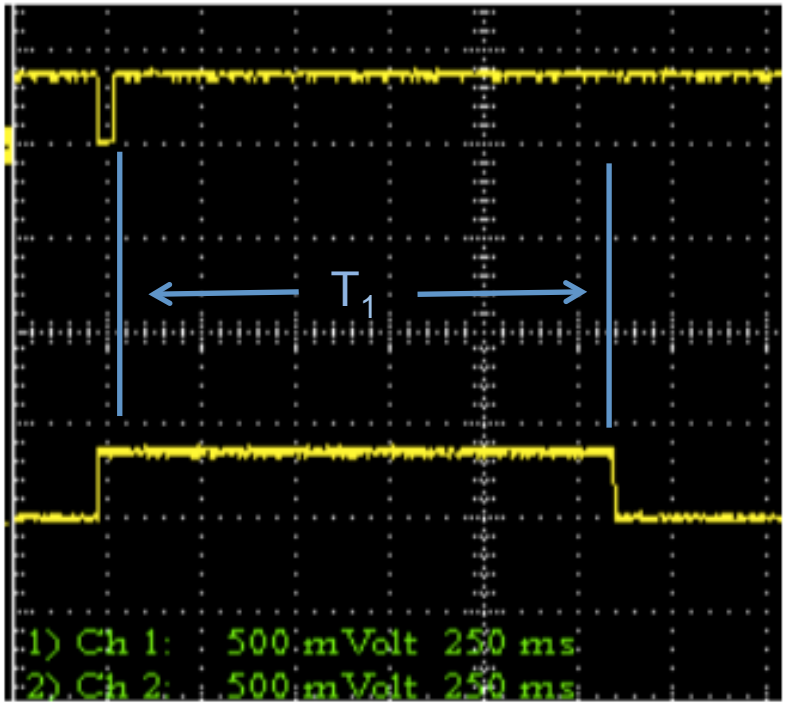
dove T é il periodo del clock, 100 e' il contenuto del registro B, 10 e 4 sono rispettivamente la durata in cicli delle istruzioni JP e DEC.

Programma di temporizzazione

- Misure da fare sulla temporizzazione

Verificare la durata di questo loop, per varie frequenze del clock esterno (per es. 1 kHz, 10 kHz, 100 kHz); a questo scopo si inserisca sulla traccia 1 dell?oscilloscopio il segnale di RESET* e sulla traccia 2 il segnale di HALT*; come segnale sincronizzante si utilizzi quello della traccia 1. Per una velocità di scansione opportuna, agendo sul comando di livello del sincronismo, ogni volta che il pulsante di RESET viene rilasciato si potrà osservare sulla traccia 2 un segnale che indica la durata del programma.

Esempio di cosa si dovrebbe ottenere

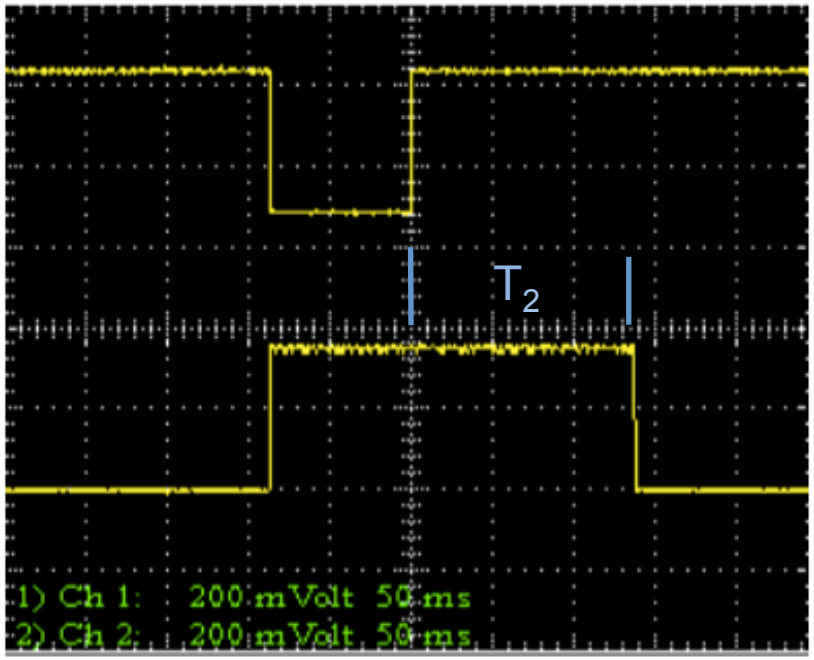


N.B. La durata del programma è tra la fine del RESET e la fine dell'HALT

$$f_1 = (1.06 \pm 0.2)kHz \Rightarrow T_1 = (1.42 \pm 0.1)s$$

$$100 * (10 + 4) * T \Rightarrow 1.32\ s$$

Esempio di cosa si dovrebbe ottenere

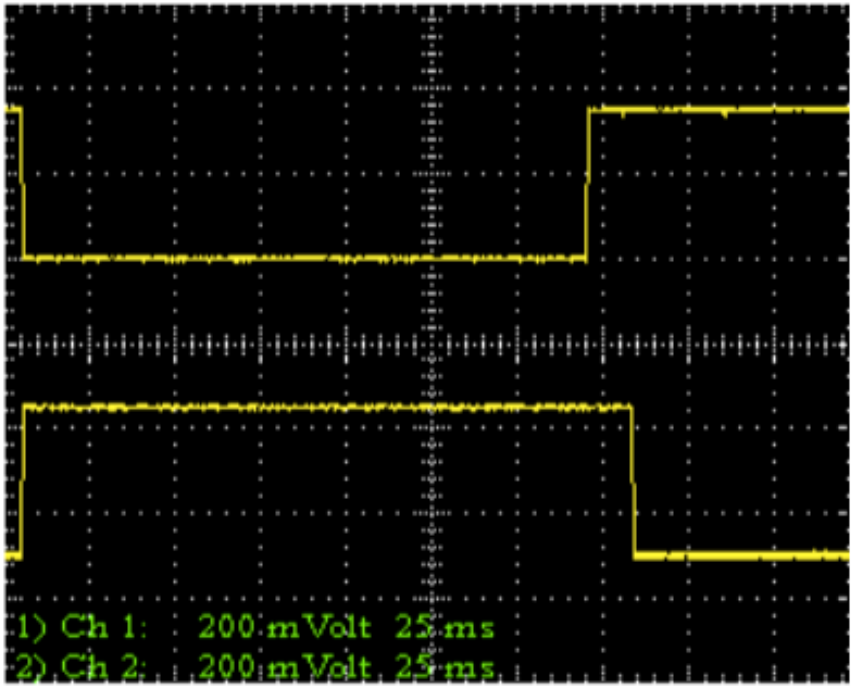


N.B. La durata del programma è tra la fine del RESET e la fine dell'HALT

$(10.4 \pm 0.3)\text{kHz}$ ⇒ $T_2 = (136 \pm 1)ms$

$100 * (10 + 4) * T$ ⇒ 135 ms

Esempio di cosa si dovrebbe ottenere



$f_3 = (101.3 \pm 0.8)\text{kHz}$ ⇒ $T_3 = (13.6 \pm 0.2)ms$

$100 * (10 + 4) * T$ ⇒ 13.8 ms

Verificare che i vari tempi di esecuzione misurati variino linearmente con il periodo del clock

Tempo di esecuzione programmato

Poiché il massimo contenuto del registro B é 255 (FF_{HEX}), vi é un limite al ritardo che si può ottenere da questo programma (si consideri il clock di sistema di 1 MHz, che corrisponde ad un periodo di 1 µs; il massimo ritardo ottenibile é di 3.570 ms).
D'altra parte, si possono ottenere durate variabili a piacere aumentando il numero dei registri da decrementare.

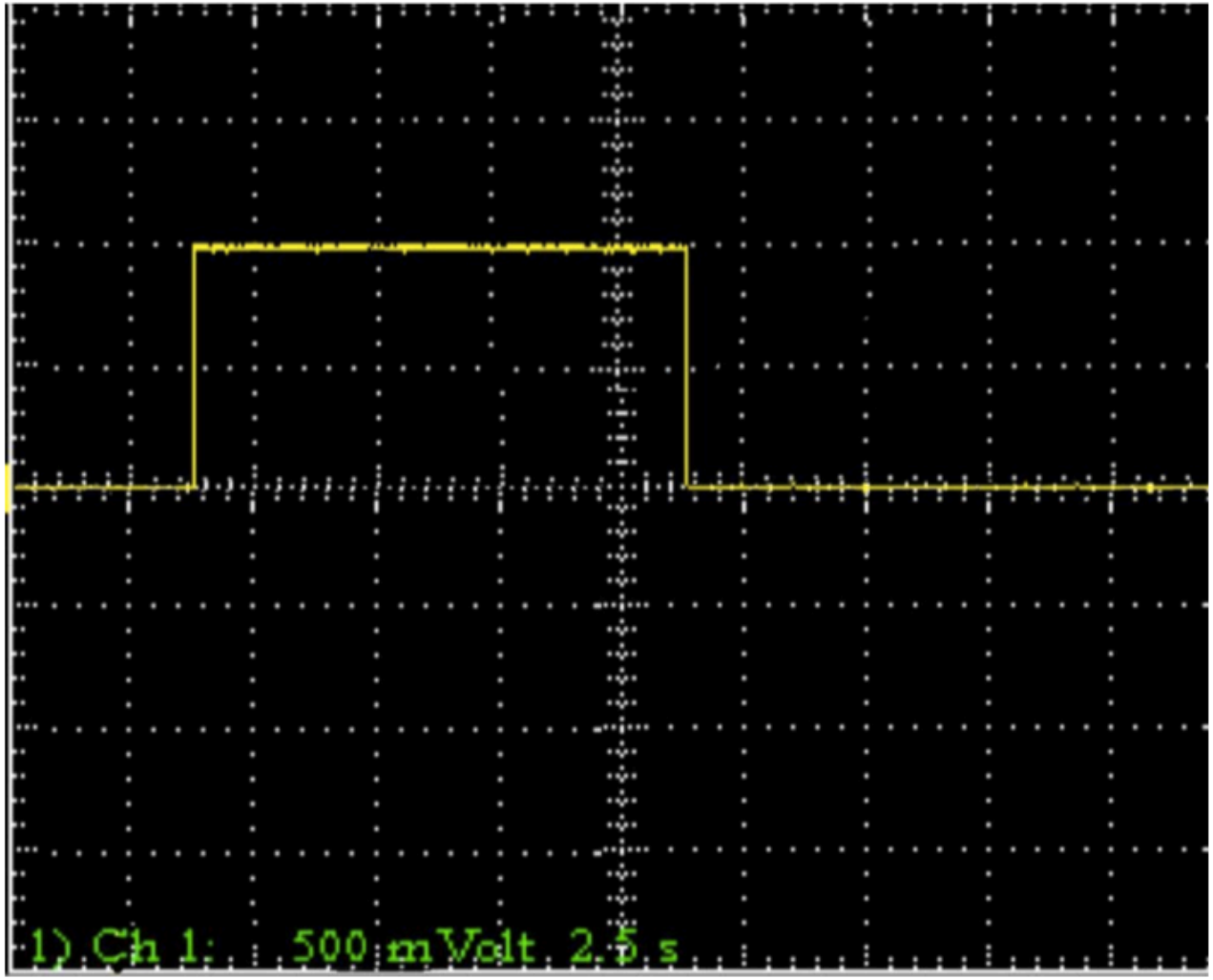
$$255 * (10 + 4) * T$$

Si scriva un programma che effettua un ritardo di 10 sec alla frequenza di 1 MHz.

- Ad esempio si possono fare loop in cascata oppure dei loop che contengono dei loop (nested loop)

Esempio di cosa si dovrebbe ottenere

Eseguendo questo programma, il led si spegne dopo circa 10s. Analizzando il segnale all'oscilloscopio, si ottiene il seguente grafico:

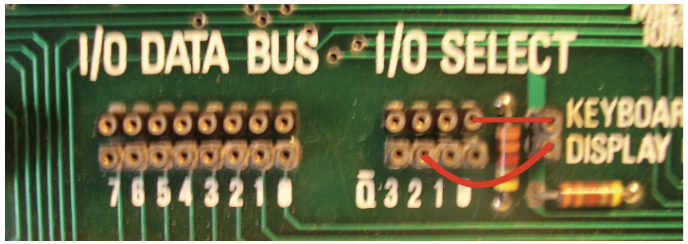


Programma di input/output

I programmi che seguono mostrano la base delle tecniche di I/O dello Z80.
Quando viene eseguita una istruzione di uscita (o ingresso), che in assembler é scritta OUT (n),A (oppure IN A,(n)) dove n=operando (numero esadecimale) e A=Accumulatore, l'operando viene posto sulle linee di indirizzo A0-A7.
Poiché tali linee sono connesse ad un decodificatore, l'uscita del decoder corrispondente all'operando fornito in ingresso verrà resa attiva. Per esempio, l'istruzione OUT $03,A significa che viene resa attiva l'uscita Q3 del decoder.

Le periferiche utilizzate, a cui viene assegnato un nome simbolico, sono i visualizzatori dei dati (uscita) e gli interruttori di predisposizione dei dati (ingresso). Per connettere in hardware tali periferiche é necessario (si veda lo schema elettrico):

- inserire un ponticello tra l'uscita desiderata (Q0-Q3) del decoder connessa al connettore I/O SELECT e il piedino del connettore KEYBOARD EN. ; viene quindi abilitata la periferica di ingresso;
- inserire un ponticello tra l'uscita desiderata (Q0-Q3) del decoder connessa al medesimo connettore e il piedino del connettore DISPLAY EN. ; viene quindi abilitata la periferica di uscita.



Programma di input/output

indirizzo	Dati	Label	istruzione	commento
00	06 64	INIZIO	B,100	; carica il registro B con 100
02	05	LOOP	DEC B	; decrementa B:B=B-1
03	C2 02 00		JP NZ,LOOP	; torna indietro se il risultato non é zero
06	C6 xx		ADD A,xx	; altrimenti somma la costante xx al contenuto dell'accumulatore
08	D3 01		OUT (O1),a	; metti il contenuto dell'accumulatore in uscita (DISPL)
0A	C3 00 00		JP INIZIO	; salta indietro e continua il ciclo

Nel memorizzare il programma scegliere il valore della costante xx da inserire nella locazione 07_{HEX}.
Con un clock esterno di 1 kHz verificare, durante l'esecuzione, i valori presenti sul visualizzatore dei dati inserendo il valore 03 nella locazione 01.
Con un clock esterno di 2-3 Hz osservare, sul visualizzatore degli indirizzi, il flusso del programma.

Programma di input/output

- Vediamo un altro programma di input/output

indirizzo	dati	label	istruzione	commento
00	06 64	INIZIO	LD B,100	; carica il registro B con 100
02	05	LOOP	DEC B	; decrementa B:B=B-1
03	C2 02 00		JP NZ, LOOP	; torna indietro se il risultato non é zero
06	DB 00		IN A,(OO)	; leggi gli interruttori e metti il valore hex nell'accumulatore
08	81		ADD A,C	; aggiungi ad A il totale contenuto nel registro C
09	00		NOP	; nessuna operazione
0A	4F		LD C,A	; trasferisci il nuovo totale nel registro C
0B	D3 01		OUT (O1) , A	; metti in uscita il nuovo totale
0D	C3 00 00		JP INIZIO	; torna indietro e ripeti il ciclo

Eseguire il programma con un clock a bassa frequenza in modo da poter seguire il cambiamento dei digit sul display.
Se è abbastanza lento si riesce a cambiare il numero di input mentre il programma è in RUN

SAPIENZA
UNIVERSITÀ DI ROMA

Fine del capitolo 7