Il Microprocessore

Prof. Ivaldi Giuliano

Sommario

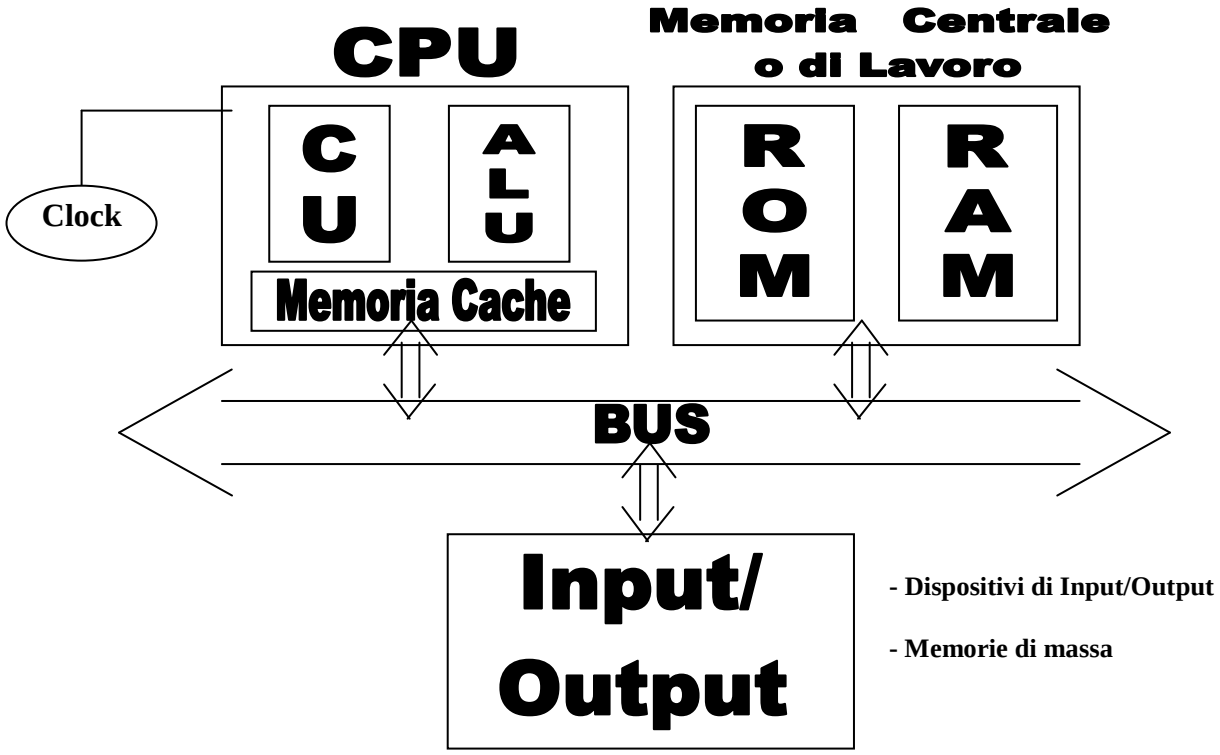
1 – Architettura di un computer

1a. Il microprocessore (μP) come macchina

Nella vita quotidiana siamo abituati ad interagire con molte macchine inventate dall'uomo, quasi sempre si tratta di macchinari elettromeccanici, come l'automobile ad esempio, ed è quindi difficile vedere un "semplice" microchip anch'esso come una macchina di tipo elettronico però, non meccanico.
Proseguendo con l'analogia dell'automobile, in essa si possono individuare vari componenti (motore, volante, acceleratore, freno, …) tutti realizzati per un solo scopo, permetterci di guidarla facendoci spostare con rapidità da un luogo all'altro; allo stesso modo come vedremo anche il microprocessore è formato da vari componenti (ALU, UC, registri) per permetterci di "guidarlo" facendogli eseguire dei programmi, cioè un insieme di istruzioni con lo scopo di eseguire automaticamente dei lavori ripetitivi per l'uomo.

Analogia automobile – computer	
Automobile	**Computer**
Motore (pistoni,…)	Microprocessore (ALU, UC, registri)
Radiatore	Ventola
Volante, cambio, acceleratore, freno	Tastiera, mouse
Tachimetro	Monitor
Benzina	Energia elettrica
Carrozzeria	Case

1b. La macchina di Von Neumann[1] (il computer più semplice)



[1] **Cenni storici:**
Von Neumann giunse a questa struttura dopo aver letto un libro dal titolo "modelli matematici del collegamento neuronale" da cui sviluppò anche il concetto di programmabilità di un calcolatore eliminando così l'estenuante lavoro di giorni che ogni volta dovevano eseguire i tecnici attraverso tutti gli interruttori di un calcolatore per riprogrammarlo.
In tale schema si possono quindi riconoscere, con le dovute differenze, le varie parti che a grandi linee compongono il sistema "uomo": parte elaborativa del cervello (UC), parte "matematica" del cervello (ALU), memoria a breve termine (RAM), memoria a lungo termine(ROM), sistema nervoso (BUS), i cinque sensi più il nostro corpo (dispositivi di Input/Output), libri (memorie di massa)

- **CPU** (**C**entral **P**rocessing **U**nit - Unità di Elaborazione Centrale): è il cuore/cervello del sistema, che decodifica ed esegue le istruzioni ed è formata dalla CU e dall'ALU:
  1) CU (**C**ontrol **U**nit – Unità di Controllo): controlla il funzionamento di tutti gli elementi che compongono il sistema di elaborazione. In particolare, l'unità di controllo ha il compito di decodificare e interpretare le istruzioni, generando i segnali di attivazione di tutti gli organi esecutivi contenuti nel computer, al ritmo degli impulsi forniti da un temporizzatore *(clock).*
  2) ALU (**A**rithmetic **L**ogic **U**nit - Unità Logico Aritmetica): *è* essenzialmente formata da una rete combinatoria (circuito con porte logiche AND, OR, NOT) che realizza le principali funzioni logiche ed è in grado di eseguire le operazioni aritmetiche elementari(somma, sottrazione, moltiplicazione, divisione) e di confronto (di uguaglianza, maggiore, minore, …), nonchè quelle di scorrimento a destra e a sinistra dei bit dell'operando coinvolto(*shift*).
     La ALU riceve in ingresso dei comandi che specificano l'operazione da effettuare ed è collegata con il *registro dati* e uno o più registri *accumulatore,* dai quali verranno prelevati gli operandi.
     A operazione ultimata l'unità aritmetico-logica, oltre a restituire il risultato nell'accumulatore, coinvolgerà un altro registro della CPU, chiamato *registro di stato,* modificandone alcuni bit *(flag di condizione),* bit che successivamente forniranno alla CPU stessa valide informazioni sull'operazione appena eseguita.

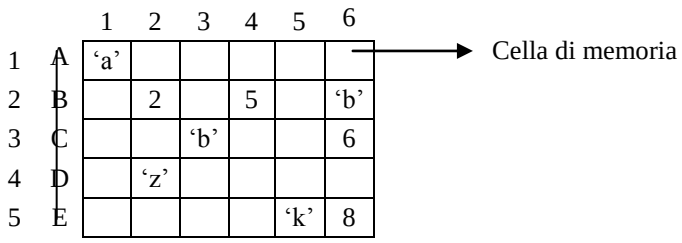
In un normale elaboratore la CPU è rappresentata dal microprocessore (es. Pentium Intel) che non è altro che un microchip, che si può immaginare come un insieme di circuiti formati da transistor, resistenze ed altri componenti elettronici.

- **MEMORIA CENTRALE o di lavoro (RAM,ROM):**

ROM (**R**ead **O**nly **M**emory – Memoria a sola lettura): contiene i programmi del BIOS necessari per l'accensione corretta del calcolatore. Questi programmi controllano che l'intero sistema funzioni correttamente ed al termine caricano il sistema operativo (es. Windows o Linux). Dopo l'accensione non viene praticamente più usata.
Caratteristiche: 1 – E' possibile solo leggere i dati in essa contenuti
2 – Memoria non volatile (se si spegne il computer i dati contenuti nella memoria non vengono persi)
3 – Molto veloce rispetto alle memorie di massa
4 – Capacità limitate rispetto alle memorie di massa
5 – Molto costosa rispetto alle memorie di massa

RAM (**R**andom **A**ccess **M**emory – Memoria ad accesso casuale): contiene i programmi aperti quando il computer è acceso (es. se state lavorando con Word, esso è contenuto nella RAM).
La RAM si può paragonare ad una bacinella: ogni volta che si apre un programma è come se si versasse dell'acqua nella bacinella, è quindi ovvio che se si aprono troppi programmi insieme il catino si riempie fino a traboccare, il che equivale per il calcolatore a rallentare o a bloccarsi del tutto perché non ha più spazio nella memoria per elaborare i programmi.
Ne consegue che non è buona norma aprire troppe applicazioni contemporaneamente, ma è sempre meglio chiudere quelle che non servono.
Caratteristiche: 1 – E' possibile scriverci o leggervi dei dati
2 – Memoria di tipo volatile (se si spegne il computer i dati contenuti nella memoria vengono persi)
3 – Molto veloce rispetto alle memorie di massa
4 – Capacità limitate rispetto alle memorie di massa
5 – Molto costosa rispetto alle memorie di massa

Struttura di una RAM: la RAM si può rappresentare come una griglia del gioco "Battaglia navale", solo che al posto delle lettere sulle righe vi sono dei numeri



Memoria ad accesso casuale significa che si può leggere o scrivere un dato qualsiasi, "a caso", con la stessa velocità.
Es. se la lettera 'a' contenuta nella cella di memoria 1-1 viene restituita dalla memoria in 3nanosecondi, anche il numero 5 della cella 2-4 sarà letto in 3nanosecondi e lo stesso vale per il numero 8 della cella 5-6 anche se si trova nell'ultima posizione.
Sembra una cosa scontata invece non è così, infatti esistono delle memorie che non sfruttano questo principio, per esempio le cassette, nelle quali il tempo di ricerca di un brano musicale dipende dalla posizione in cui esso si trova (primo, secondo, …, ultimo)

- **BUS**: fascio di linee (fili di rame o piste di circuiti stampati) che mettono in comunicazione le varie parti del sistema di elaborazione

- **Periferiche di INPUT/OUTPUT:** è l'insieme dei dispositivi che permettono al microprocessore di interagire con il mondo esterno

  Si dividono in due grandi categorie:

  1) Dispositivi di INPUT/OUTPUT

     INPUT deriva dall'inglese e significa mettere (to PUT) dentro (IN)

     Esempi di dispositivi di input: tastiera, mouse, scanner, joystick, trackball

     OUTPUT invece significa mettere fuori (OUT)

     Esempi di dispositivi di output: monitor, casse, stampante, plotter

     Una domanda sorge spontanea: mettere dentro o fuori, ma rispetto a che cosa? Rispetto alla CPU

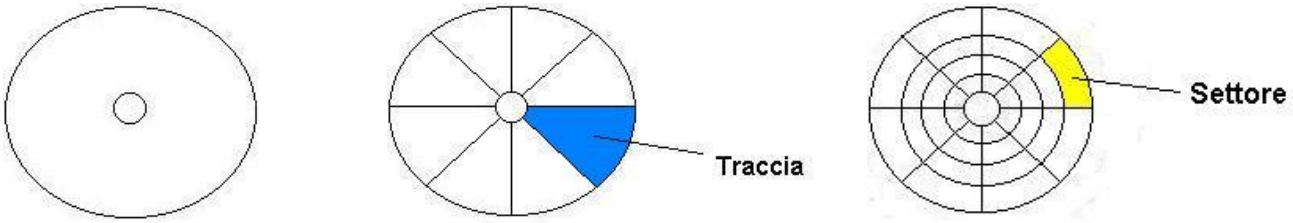
  2) Memorie di massa: sono memorie di tipo permanente (non volatili) perché permettono la memorizzazione di grandi quantità di dati per lungo tempo (es. hard-disk, cd, dvd, chiavette)

     Contengono i programmi quando sono chiusi (non aperti), perché quando li si apre, passano nella RAM

     Caratteristiche: 1 – E' possibile scriverci o leggervi dei dati
     2 – Non volatili
     3 – Più lente rispetto alla RAM e alla ROM
     4 – Grandi capacità rispetto alla RAM e alla ROM
     5 – Più economiche rispetto alla RAM e alla ROM
     6 – Fanno parte dei dispositivi di I/O, non della memoria centrale

     Le memorie di massa sono dispositivi sia di INPUT che di OUTPUT, perché è possibile sia leggervi che scriverci dei dati

  Struttura di un disco

  1) 2) 3)

  Un disco vergine, sia esso un hard-disk o un floppy-disk si presenta come in figura 1) e non può essere utilizzato.
  Perché vi si possano registrare dei dati è necessario formattarlo: il computer esegue questa operazione suddividendolo in tanti spicchi, detti tracce (fig. 2) ed in cerchi concentrici, che intersecandosi con le tracce danno origine ai cosiddetti settori (fig. 3)

  Salvataggio dei dati: Un file non viene salvato in un blocco unico, ma viene diviso in più parti, ognuna di esse sarà grande come un settore, quindi il file verrà memorizzato in diversi settori, che saranno contigui se vi è molto spazio libero sul disco, altrimenti saranno sparpagliati.
  Nel secondo caso qualcuno si potrebbe porre il problema di come successivamente il computer riuscirà a ricomporre i programmi, ciò è possibile perché nel disco esiste una tabella in cui viene memorizzato il settore iniziale di ogni file, dopodiché alla fine di ogni blocco di programma salvato vi è l'indicazione del settore successivo da andare a leggere, fino all'ultimo, che chiude il caricamento del file.
  E' ovvio che più saranno sparpagliati i blocchi più tempo ci vorrà per caricare il programma; per ordinarli e compattare i dati è stata inventata un'applicazione che si chiama "deframmentazione", richiede molto tempo per essere eseguita, tempo che poi si guadagna all'apertura e nell'esecuzione dei programmi.

- **Clock:** come in un gruppo musicale si batte il piede per andare tutti a tempo e suonare correttamente un brano, così il calcolatore ha bisogno di un temporizzatore che permetta di sincronizzare tutte le componenti del sistema, per poter svolgere correttamente tutte le operazioni, questo è il clock

- **Cache Memory:** sebbene sia una componente della CPU ho deciso di descriverla a parte per evitare confusione con gli altri tipi di memorie. I computer hanno bisogno di sempre maggiori velocità per elaborare la grande mole di dati a cui vengono sottoposti, per questo ad un certo punto si è pensato di inserire direttamente all'interno del microprocessore una piccola memoria ancora più veloce della RAM, in questo modo le informazioni necessarie alla CPU per eseguire i programmi non vengono più spostate ogni volta nella RAM, ma tenute proprio all'interno del microprocessore stesso che così risparmia tempo non dovendole caricare ogni volta.
  Analogia: se si ha bisogno di una bibita per una festa, la si può acquistare in un grande magazzino in periferia (memorie di massa) oppure nel bar sottocasa (memoria RAM) oppure se ne si ha un bisogno immediato, se ne possono stipare un certo numero nel proprio frigorifero (cache memory).

Funzionamento della macchina di von-Neumann:

Accensione (Boot Strap)

1 – i programmi del Bios contenuti nella ROM passano nella RAM
2 – dalla RAM, attraverso il Bus, le istruzioni dei programmi vengono portate una per una nella CPU
3 – la CPU elabora tutte le istruzioni e restituisce gli eventuali risultati alla RAM e/o ai dispositivi di I/O (per esempio al monitor che visualizza una schermata nera iniziale con l'analisi del sistema)

Apertura di un programma

1 – il programma dalla memoria di massa (es. hard-disk, cd, dvd, chiavetta) passa nella RAM
2 – dalla RAM, attraverso il Bus, le istruzioni del programma vengono portate una per una nella CPU
3 – la CPU elabora tutte le istruzioni e restituisci gli eventuali risultati alla RAM e/o ai dispositivi di I/O (per esempio al monitor che visualizza la schermata di apertura di word o di un gioco)

Esercizio: provare ad indicare, disegnando delle frecce sulla macchina di von-Neumann, i passaggi appena spiegati.

A questo punto sarebbe utile per chi ne ha la possibilità, smontare un computer e cercare di individuare le varie componenti della macchina di von Neumann.

Fattori che influenzano le prestazioni di un computer (utili anche all'acquisto di un computer):

- Frequenza processore (Hz, velocità del processore)
- Capacità Ram (byte)
- Capacità Hard-Disk (byte)
- Frequenza Bus di sistema (Hz, velocità del bus di sistema)
- Buona Scheda Video
- Capacità Cache memory (byte)

Esercizio: provare a leggere un volantino pubblicitario sulla vendita di computer

Appendice - Microprocessori e Scale di integrazione (SSI, MSI, LSI, VLSI)

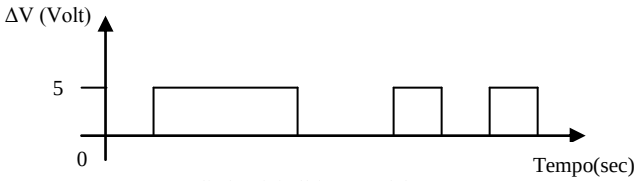
Come già detto la CPU è costruita su un unico microchip, questi vengono classificati in elettronica in base al numero di porte logiche che contengono, maggiore è il numero di porte maggiore sarà la capacità di "elaborazione".

Scala di integrazione	N° di porte logiche	Categorie
SSI (Small scale of Integration)	<= 12	Porte logiche fondamentali, Flip-Flop
MSI(Medium Scale of Integration)	12 - 100	Decoder, encoder, contatori, registri, multiplexer, circuiti aritmetici, piccole memorie
LSI (Large Scale of Integration)	100 – 1000	Memorie e alcuni tipi di Microprocessori (μP)
VLSI(Very Large Scale of Integration)	1000 – 100000 e oltre	Memorie grandi, Microprocessori, Computer su unico chip

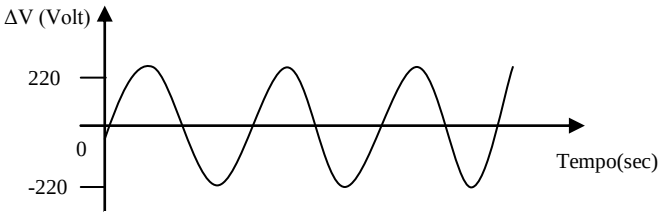
2 - Bit, Byte e Sistemi di numerazione

2a. Il computer e i Segnali elettrici

Il computer come ogni altra macchina elettronica funziona grazie a dei segnali elettrici, in particolare utilizza degli impulsi a forma di onda quadra che possono assumere solo due valori 0Volt o 5Volt.



Segnali elettrici all'interno del computer



Segnale elettrico di una normale presa di corrente

Dopo i primi anni di vita dei calcolatori elettronici si cominciò ad astrarre l'elaborazione delle informazioni dalla struttura fisica delle macchine, così facendo i due segnali elettrici sopraccitati diventarono due valori numerici

0 Volt -> 0
5 Volt -> 1

Nacquero così il **bit (binary digit – cifra binaria,** perché sono possibili solo 2 valori) e il **Byte** (1Byte = 8bit).

Definizione di bit: *è l'unità di informazione minima che può trattare e memorizzare un elaboratore.*

Con l'evoluzione tecnologica vennero poi introdotti dei multipli anche per il byte

Unità di misura	**Simbolo**	**Equivale a…**
Byte	B	8 bit
Kilobyte	KB	1024 byte = 2^{10} byte
Megabyte	MB	1024 kilobyte = 1.048.576 byte = 2^{20} byte
Gigabyte	GB	1024 megabyte = 1.048.576 kilobyte = 1.073.741.824 byte = 2^{30} byte
Terabyte	TB	1024 gigabyte = 1.048.576 megabyte = 1.073.741.824 kilobyte = 1099.511.627.776 byte = 2^{40} byte

Facciamo qualche conversione di esempio:

3GB = 3*1024 MB = 3.072 MB 3GB = 3*1024*1024 KB = 3*1024^2 KB = 3.145.728 KB 3GB = 3*1024*1024*1024 B = 3*1024^3 B = 3.221.225.472 B = 3.221.225.472*8 bit = 25.769.803.776 bit	489.626.271.744 bit = 489.626.271.744/8 B = 61.203.283.968 B 61.203.283.968 B = 61.203.283.968 / 1024 KB =59.768.832 KB 61.203.283.968 B = 61.203.283.968 / (1024*1024) MB = 61.203.283.968 / 1024^2 = 58368 MB 61.203.283.968 B = 61.203.283.968 / (1024*1024*1024) GB = 61.203.283.968 / 1024^3 = 57 GB
122.880 MB = 122.880 /1024 GB = 120 GB 122.880 MB = 122.880*1024 KB =125.829.120 KB 122.880 MB = 122.880*1024*1024 B =128.849.018.880 B = 128.849.018.880*8 bit = 1.030.792.151.040 bit	Di compito convertire in tutte le possibili unità di misura, 10GB, 61440MB, 92.274.688KB, 37.580.963.840B, 584.115.552.256bit,

Perchè studiare questi argomenti?

Nonostante l'astrazione il significato fisico dei bit non è mutato, questo significa che allo 0 corrispondono ancora 0Volt, mentre all'1 corrispondono 5Volt. Questo significa che con un computer è possibile pilotare anche dei circuiti elettronici, accendendo o spegnendo delle luci o un motorino con una sequenza di bit, tramite opportuni comandi in un linguaggio di programmazione.

A questo punto è chiaro che per poter comprendere a pieno il funzionamento di un computer e della circuiteria annessa è inevitabile studiare come si conta con i bit, questo ci porterà in un "mondo parallelo" in cui non si conta più in base 10, ma in base 2, tanti quanti sono i valori possibili per un bit, per poi passare alle basi 8 e 16 che permettono di semplificare i valori numerici troppo grandi che risulterebbero proprio dalle operazioni svolte nel sistema binario.

2b. I sistemi di numerazione

Il nostro sistema di numerazione è il sistema decimale, formato da dieci cifre da 0 a 9, in realtà esistono anche altre modalità per contare, ad esempio un sistema vecchio di 4000anni è quello sessagesimale, usato ancora oggi negli orologi, che risale però agli antichi Babilonesi.
Analizzando la struttura della base 10 è però intuibile che possano esistere un'infinità di altre basi, base 2, base 3, base 4, base 8, base 16 e così via, vediamo come è possibile costruirle, studiando la seguente tabella

Base 10	Base 2	Base 8	Base 16
…00000	…00000000	…00000	…00000
…00001	…00000001	…00001	…00001
…00002	…00000010	…00002	…00002
…00003	…00000011	…00003	…00003
…00004	…00000100	…00004	…00004
…00005	…00000101	…00005	…00005
…00006	…00000110	…00006	…00006
…00007	…00000111	…00007	…00007
…00008	…00001000	…00010	…00008
…00009	…00001001	…00011	…00009
…00010	…00001010	…00012	…0000A
…00011	…00001011	…00013	…0000B
…00012	…00001100	…00014	…0000C
…00013	…00001101	…00015	…0000D
…00014	…00001110	…00016	…0000E
…00015	…00001111	…00017	…0000F
…00016	…00010000	…00020	…00010
…00017	…00010001	…00021	…00011
…00018	…00010010	…00022	…00012
…00019	…00010011	…00023	…00013
…00020	…00010100	…00024	…00014
…00021	…00010101	…00025	…00015
…00022	…00010110	…00026	…00016
…00023	…00010111	…00027	…00017
…00024	…00011000	…00030	…00018
…00025	…00011001	…00031	…00019
…00026	…00011010	…00032	…0001A
…00027	…00011011	…00033	…0001B
…00028	…00011100	…00034	…0001C
…00029	…00011101	…00035	…0001D
…	…	…	…
…00098	…01100010	…00142	…00062
…00099	…01100011	…00143	…00063
…00100	…01100100	…00144	…00064
…00101	…01100101	…00145	…00065
…00102	…01100110	…00146	…00066

Base 10
Come si può notare per la base 10, esistono dieci cifre da 0 a 9 e ogni numero si può scrivere immaginandolo preceduto da un'infinità di zeri (anche se normalmente non si mettono per comodità).
Arrivati al termine delle dieci cifre si prosegue sostituendo allo zero delle decine il primo valore successivo che è 1, e per le unità si riinizia il conteggio da 0, fino a 19, quando l'1 delle decine diventa 2 e per le unità si ricomincia da 0. Giunti al 99, lo 0 delle centinaia diventa 1, mentre per le decine e le unità si ricomincia da 0, cioè 100 e così via….

Lo stesso discorso vale per le altre basi
Base 2, esistono due cifre da 0 a 1 e ogni numero si può scrivere immaginandolo preceduto da un'infinità di zeri.
Arrivati al termine delle due cifre si prosegue sostituendo allo zero delle "decine" il primo valore successivo che è 1, e per le "unità" si riinizia il conteggio da 0, fino a 11. Giunti a 11, lo 0 delle centinaia diventa 1, mentre per le decine e le unità si ricomincia da 0, cioè 100, (che però in questo caso equivale al 4 della base 10), e così via…

Base 8, esistono otto cifre da 0 a 7 e ogni numero si può scrivere immaginandolo preceduto da un'infinità di zeri.
Arrivati al termine delle otto cifre si prosegue sostituendo allo zero delle "decine" il primo valore successivo che è 1, e per le "unità" si riinizia il conteggio da 0, fino a 17, quando l'1 delle decine diventa 2 e per le unità si ricomincia da 0. Giunti a 77, lo 0 delle centinaia diventa 1, mentre per le decine e le unità si ricomincia da 0, cioè 100, (che però in questo caso equivale al 64 della base 10), e così via…

Base 16, esistono sedici cifre da 0 a F, infatti siccome i valori da 10 a 15 vengono già utilizzati in seguito non si potevano più usare per indicare i numeri da dieci a quindici, allora si è pensato a dei nuovi simboli, che potevano anche essere il "?" o il "!", ma cosa vi è di più semplice delle lettere dell'alfabeto?
Arrivati al termine delle sedici cifre si prosegue sostituendo allo zero delle "decine" il primo valore successivo che è 1, e per le "unità" si riinizia il conteggio da 0, fino a 1F, quando l'1 delle decine diventa 2 e per le unità si ricomincia da 0. Giunti a FF, lo 0 delle centinaia diventa 1, mentre per le decine e le unità si ricomincia da 0, cioè 100,

Da quanto detto si intuisce che esistono più numeri in basi diverse rappresentati allo stesso modo: ad esempio il 10 della base dieci o il 10 della base 2 che equivale però al numero 2 in base 10 o il 10 della base 8 che equivale però all'8 della base 10 e così di seguito…
Come distinguerli ? Semplicemente con un pedice, così $10_{(10)}$, $10_{(2)}$, $10_{(8)}$, $10_{(16)}$, …
E' da notare che mentre il $10_{(10)}$ si legge dieci, il 10 di tutte le altre basi si legge uno-zero seguito dal nome della base e lo stesso vale per tutti gli altri numeri composti: ad esempio $11_{(2)}$ si leggerà uno-uno in base 2, $16_{(8)}$ si leggerà uno-sei in base 8, $2A_{(16)}$ si leggerà due-a in base 16.
E' infine abbastanza ovvio che debbano esistere dei metodi convenienti per poter convertire i numeri da una base all'altra senza dover far uso continuo di tabelle, i prossimi paragrafi sono dedicati proprio a questo scopo.

Esercizi di compito, costruire le basi 3, e 12 e confrontarle con la base 10 (scrivere i valori fino a $30_{(10)}$).

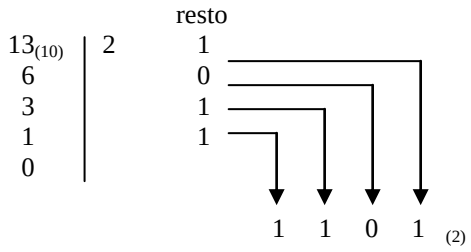
Il sistema binario

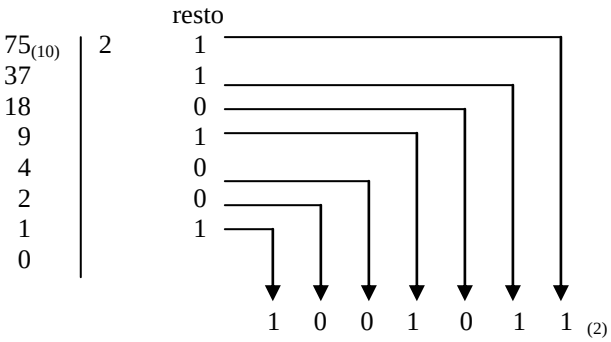
Simboli, 0 e 1

Conversioni decimale – binario

Si prende il numero da convertire e si eseguono una serie di divisioni intere per 2 fino a quando si ottiene un quoziente uguale a 0, annotando tutti resti. Una volta terminato il processo si prendono i resti dall'ultimo al primo e si ha il numero convertito in base 2.

Esempi,





Esercizi di compito: convertire in base 2 i seguenti numeri, $7_{(10)}$, $84_{(10)}$, $236_{(10)}$, $1255_{(10)}$

Conversioni binario – decimale

Partiamo dal concetto noto che un numero in base 10 si possa scrivere nella seguente forma

Esempio, $435_{(10)} = 4*100 + 3*10 + 5*1 = 4*10^2 + 3*10^1 + 5*10^0$

Quindi le cifre occupano delle posizioni ben precise: a partire da destra le unità si trovano nella posizione 0 (e si dice che hanno peso 10^0), le decine nella posizione 1 (peso 10^1), le centinaia nella posizione 2 (peso 10^2), e così via…

Lo stesso vale per la base 2 in cui invece di moltiplicare per potenze del dieci si moltiplicherà per potenze del due, ottenendo alla fine dei calcoli il numero convertito in base 10.

Esempi, $10111_{(2)} = 1*2^0 + 1*2^1 + 1*2^2 + 0*2^3 + 1*2^4 = 1 + 2 + 4 + 0 + 16 = 23_{(10)}$

$1110101_{(2)} = 1*2^0 + 0*2^1 + 1*2^2 + 0*2^3 + 1*2^4 + 1*2^5 + 1*2^6 = 1 + 0 + 4 + 0 + 16 +32 + 64 = 117_{(10)}$

Esercizi di compito: convertire in base 10 i seguenti numeri, $11111_{(2)}$, $101010_{(2)}$, $1000011_{(2)}$, $11101001_{(2)}$

Operazioni in base 2

Regole della somma	Regole della sottrazione	Regole della moltiplicazione
$0_{(2)} + 0_{(2)} = 0_{(2)}$	$0_{(2)} - 0_{(2)} = 0_{(2)}$	$0_{(2)} * 0_{(2)} = 0_{(2)}$
$0_{(2)} + 1_{(2)} = 1_{(2)}$	$0_{(2)} - 1_{(2)}$ = impossibile	$0_{(2)} * 1_{(2)} = 0_{(2)}$
$1_{(2)} + 0_{(2)} = 1_{(2)}$	$1_{(2)} - 0_{(2)} = 1_{(2)}$	$1_{(2)} * 0_{(2)} = 0_{(2)}$
$1_{(2)} + 1_{(2)} = 0_{(2)}$ riporto 1	$1_{(2)} - 1_{(2)} = 0_{(2)}$	$1_{(2)} * 1_{(2)} = 1_{(2)}$

Esempi,

$$\begin{array}{r} {\scriptstyle 1\,1\,1\,1\,\,1\,1} \\ 1011011_{(2)} + \\ 111111_{(2)} = \\ \hline 10011010_{(2)} \end{array}$$

$$\begin{array}{r} {\scriptstyle 1} \\ {\scriptstyle 1\,1\,1\,1} \\ 101_{(2)} + \\ 111_{(2)} + \\ 111_{(2)} = \\ \hline 10011_{(2)} \end{array}$$

$$\begin{array}{r} {\scriptstyle 0\,1_1\,0\,\,1_0\,1_1\,0\,\,1\,\,1} \\ 1\,1\,0\,1\,1\,0_{(2)} = \\ \hline 1\,0\,0\,1\,0\,1_{(2)} \end{array}$$

$$\begin{array}{r} 101_{(2)} * \\ 111_{(2)} = \\ \hline 101 \\ 101— \\ 101— \\ \hline 100011_{(2)} \end{array}$$

Esercizi di compito: svolgere le seguenti operazioni

$101010_{(2)} + 11111_{(2)} = \ldots$ $\qquad$ $11101001_{(2)} + 1000011_{(2)} = \ldots$

$101010_{(2)} - 11111_{(2)} = \ldots$ $\qquad$ $11101001_{(2)} - 1000011_{(2)} = \ldots$

$101010_{(2)} * 11111_{(2)} = \ldots$ $\qquad$ $11101001_{(2)} * 1000011_{(2)} = \ldots$

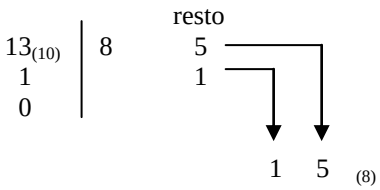
Il sistema ottale

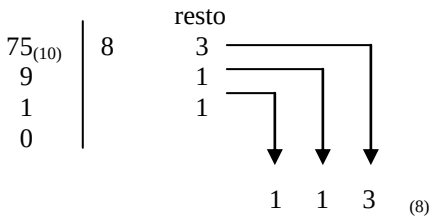
Simboli, 0, 1, 2, 3, 4, 5, 6, 7

Conversioni decimale – ottale

Si prende il numero da convertire e si eseguono una serie di divisioni intere per 8 fino a quando si ottiene un quoziente uguale a 0, annotando tutti resti. Una volta terminato il processo si prendono i resti dall'ultimo al primo e si ha il numero convertito in base 8.

Esempi,





Esercizi di compito: convertire in base 8 i seguenti numeri, $7_{(10)}$, $84_{(10)}$, $236_{(10)}$, $1255_{(10)}$

Conversioni ottale – decimale

Partiamo dal concetto noto che un numero in base 10 si possa scrivere nella seguente forma

Esempio, $435_{(10)} = 4*100 + 3*10 + 5*1 = 4*10^2 + 3*10^1 + 5*10^0$

Quindi le cifre occupano delle posizioni ben precise: a partire da destra le unità si trovano nella posizione 0 (e si dice che hanno peso 10^0), le decine nella posizione 1 (peso 10^1), le centinaia nella posizione 2 (peso 10^2), e così via…

Lo stesso vale per la base 8 in cui invece di moltiplicare le cifre per le potenze del dieci si moltiplicheranno per le potenze dell'otto, ottenendo alla fine dei calcoli il numero convertito in base 10.

Esempi, $75_{(8)} = 5*8^0 + 7*8^1 = 5 + 56 = 61_{(10)}$

$225_{(8)} = 5*8^0 + 2*8^1 + 2*8^2 = 5 + 16 + 128 = 149_{(10)}$

Esercizi di compito: convertire in base 10 i seguenti numeri, $36_{(8)}$, $174_{(8)}$, $5432_{(8)}$, $7243_{(8)}$

Conversioni binario – ottale

Siccome il numero 8 è una potenza del 2, in particolare 2^3, è molto semplice convertire i numeri da base binaria a base ottale, basta costruirsi una tabella come quella di pagina 5, sono sufficienti le prime 8 cifre, e associare a ogni tre cifre binarie del numero da convertire, partendo da destra, una cifra in base 8.

Esempi,

$100101010_{(2)}$ = 100 101 010$_{(2)}$ → 4 5 2 $_{(8)}$

$1111110_{(2)}$ = 001 111 110$_{(2)}$ → 1 7 6 $_{(8)}$

Esercizi di compito: convertire in base 8 i seguenti numeri, $1001101_{(2)}$, $1010101010_{(2)}$, $11100011100_{(2)}$, $1110110111_{(2)}$

Conversioni ottale – binario

Il procedimento contrario invece permette di convertire facilmente numeri dalla base ottale a quella binaria: basta associare a ogni cifra in base 8 tre cifre in base 2, laddove non vi è la corrispondenza con le tre cifre bisogna aggiungere degli zeri.

Esempi,

$327_{(8)}$ = 3 2 7 $_{(8)}$ → 011 010 111 = $11010111_{(2)}$

$612_{(8)}$ = 6 1 2 $_{(8)}$ → 110 001 010 = $110001010_{(2)}$

Esercizi di compito: convertire in base 2 i seguenti numeri, $36_{(8)}$, $174_{(8)}$, $5432_{(8)}$, $7243_{(8)}$

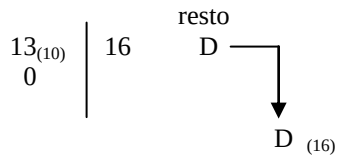
Il sistema esadecimale

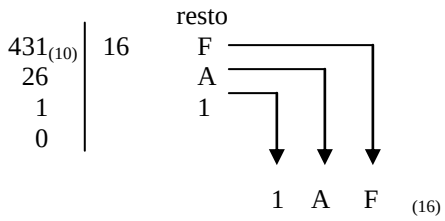
Simboli, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Conversioni decimale – esadecimale

Si prende il numero da convertire e si eseguono una serie di divisioni intere per 16 fino a quando si ottiene un quoziente uguale a 0, annotando tutti resti. Una volta terminato il processo si prendono i resti dall'ultimo al primo e si ha il numero convertito in base 16.

Esempi,





Esercizi di compito: convertire in base 16 i seguenti numeri, $7_{(10)}$, $84_{(10)}$, $236_{(10)}$, $1255_{(10)}$

Conversioni esadecimale – decimale

Partiamo dal concetto noto che un numero in base 10 si possa scrivere nella seguente forma

Esempio, $435_{(10)} = 4*100 + 3*10 + 5*1 = 4*10^2 + 3*10^1 + 5*10^0$

Quindi le cifre occupano delle posizioni ben precise: a partire da destra le unità si trovano nella posizione 0 (e si dice che hanno peso 10^0), le decine nella posizione 1 (peso 10^1), le centinaia nella posizione 2 (peso 10^2), e così via…

Lo stesso vale per la base 8 in cui invece di moltiplicare le cifre per le potenze del dieci si moltiplicheranno per le potenze dell'otto, ottenendo alla fine dei calcoli il numero convertito in base 10.

Esempi, $7A_{(16)} = 10*16^0 + 7*16^1 = 10 + 112 = 122_{(10)}$

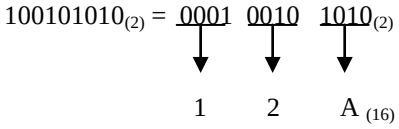
$2C5_{(16)} = 5*16^0 + 12*16^1 + 2*16^2 = 5 + 192 + 512 = 709_{(10)}$

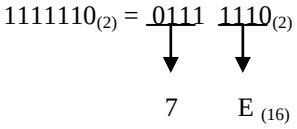
Esercizi di compito: convertire in base 10 i seguenti numeri, $FC_{(16)}$, $B94_{(16)}$, $5ED2_{(16)}$, $79EB_{(16)}$

Conversioni binario – esadecimale

Siccome il numero 16 è una potenza del 2, in particolare 2^4, è molto semplice convertire i numeri da base binaria a base esadecimale, basta costruirsi una tabella come quella di pagina 5, sono sufficienti le prime 16 cifre, e associare a ogni quattro cifre binarie del numero da convertire, partendo da destra, una cifra in base 16.

Esempi,



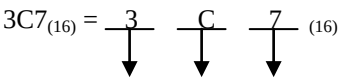


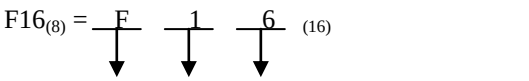
Esercizi di compito: convertire in base 16 i seguenti numeri, $1001101_{(2)}$, $1010101010_{(2)}$, $11100011100_{(2)}$, $1110110111_{(2)}$

Conversioni esadecimale – binario

Il procedimento contrario invece permette di convertire facilmente numeri dalla base esadecimale a quella binaria: basta associare a ogni cifra in base 16 quattro cifre in base 2, laddove non vi è la corrispondenza con le quattro cifre bisogna aggiungere degli zeri.

Esempi,





Esercizi di compito: convertire in base 2 i seguenti numeri, $FC_{(16)}$, $B94_{(16)}$, $5ED2_{(16)}$, $79EB_{(16)}$

convertire in base 16, senza eseguire calcoli, i seguenti numeri, $23_{(8)}$, $654_{(8)}$, $772_{(8)}$, $4363_{(8)}$

convertire in base 8, senza eseguire calcoli, i seguenti numeri, $BE_{(16)}$, $C76_{(16)}$, $32C1_{(16)}$, $97CA_{(16)}$

2f. Codici alfanumerici

Dopo aver approfondito lo studio dei vari sistemi di numerazione resta ancora da studiare come il computer possa "capire" simboli che non siano i numeri, come ad esempio le lettere dell'alfabeto o il "?" o il "!", eccetera…
La risposta è molto semplice, infatti anch'essi sono "visti" come numeri dai calcolatori che usano apposite tabelle per convertirli in valori numerici trattabili dagli elaboratori stessi: per riconoscerli dalla tastiera e per visualizzare i pixel necessari a video.
Uno dei codici alfanumerici più utilizzati è senza dubbio il codice ASCII (American Standard Code for Information Interchange – codice standard americano per lo scambio di informazioni), in cui ogni simbolo viene rappresentato con 7bit (per un totale di 128caratteri, vedere paragrafo successivo "Le variabili"), ma ne esiste una versione estesa a 8bit (per un totale di 256caratteri) cioè 1Byte per carattere e infatti l'unita di misura minima per una cella di memoria ram nei normali PC è proprio di 1Byte: quindi una cella di memoria è in grado di contenere un carattere alfanumerico o un numero da 0 a 255 (perché con 1Byte si possono rappresentare come detto 256 valori, il motivo lo capiremo in seguito).

Segue la tabella di conversione del codice ASCII esteso (i caratteri sono visualizzabili su un normale PC premendo il tasto Alt seguito dal codice del carattere, ad esempio usando il Blocco Note)

0		32		64	@	96	`	128	Ç	160	á	192	└	224	Ó
1	☺	33	!	65	A	97	a	129	ü	161	í	193	┴	225	ß
2	☻	34	"	66	B	98	b	130	é	162	ó	194	┬	226	Ô
3	♥	35	#	67	C	99	c	131	â	163	ú	195	├	227	Ò
4	♦	36	$	68	D	100	d	132	ä	164	ñ	196	─	228	õ
5	♣	37	%	69	E	101	e	133	à	165	Ñ	197	┼	229	Õ
6	♠	38	&	70	F	102	f	134	å	166	ª	198	ã	230	µ
7	•	39	'	71	G	103	g	135	ç	167	º	199	Ã	231	þ
8	◘	40	(	72	H	104	h	136	ê	168	¿	200	╚	232	Þ
9	○	41	)	73	I	105	i	137	ë	169	®	201	╔	233	Ú
10	◙	42	*	74	J	106	j	138	è	170	¬	202	╩	234	Û
11	♂	43	+	75	K	107	k	139	ï	171	½	203	╦	235	Ù
12	♀	44	,	76	L	108	l	140	î	172	¼	204	╠	236	ý
13	♪	45	-	77	M	109	m	141	ì	173	¡	205	═	237	Ý
14	♫	46	.	78	N	110	n	142	Ä	174	«	206	╬	238	¯
15	☼	47	/	79	O	111	o	143	Å	175	»	207	¤	239	´
16	►	48	0	80	P	112	p	144	É	176	░	208	ð	240	-
17	◄	49	1	81	Q	113	q	145	æ	177	▒	209	Ð	241	±
18	↕	50	2	82	R	114	r	146	Æ	178	▓	210	Ê	242	‗
19	‼	51	3	83	S	115	s	147	ô	179	│	211	Ë	243	¾
20	¶	52	4	84	T	116	t	148	ö	180	┤	212	È	244	¶
21	§	53	5	85	U	117	u	149	ò	181	Á	213	ı	245	§
22	▬	54	6	86	V	118	v	150	û	182	Â	214	Í	246	÷
23	↨	55	7	87	W	119	w	151	ù	183	À	215	Î	247	¸
24	↑	56	8	88	X	120	x	152	ÿ	184	©	216	Ï	248	°
25	↓	57	9	89	Y	121	y	153	Ö	185	╣	217	┘	249	¨
26	→	58	:	90	Z	122	z	154	Ü	186	║	218	┌	250	·
27	←	59	;	91	[	123	{	155	ø	187	╗	219	█	251	¹
28	∟	60	<	92	\	124	|	156	£	188	╝	220	▄	252	³
29	↔	61	=	93	]	125	}	157	Ø	189	¢	221	¦	253	²
30	▲	62	>	94	^	126	~	158	×	190	¥	222	Ì	254	■
31	▼	63	?	95	_	127	⌂	159	ƒ	191	┐	223	▀	255	

2g. Le variabili

Le variabili sono contenitori di valori, numerici o di tipo carattere, elaborati durante l'esecuzione di un programma, risiedono nella memoria RAM. Ad ogni variabile sono associati un nome, detto **identificatore**, che la identifica univocamente all'interno del programma e un valore che può cambiare durante l'esecuzione del programma.
Nomi delle variabili ammessi: qualsiasi lettera o parola è ammessa, se si usano nomi formati da più parole, queste dovranno essere separate da "_". Consiglio: come si dice in campo informatico è utile dare dei nomi "parlanti" alle variabili cioè dei nomi che permettano di capire cosa contengono le variabili stesse. Esempi: x, y, a, b, c, n1, n2, somma, risultato, resto_divisione, num_casuale.
Analogie: il concetto di variabile in matematica dovrebbe essere ben noto, qui si espande e oltre che contenitore di valori numerici diventa anche contenitore di caratteri o parole. Per chi ha dimestichezza con i fogli di calcolo (es, Excel, Calc) può immaginare una variabile come una cella di Excel o Calc, che come è noto ha anch'essa un nome, nome_colonna-nome_riga (es. A1, D4) e può contenere numeri interi, con la virgola, lettere o parole e può essere sommata, sottratta, moltiplicata o divisa per altre celle (non è un caso visto che il foglio di calcolo è un programma ed è stato costruito utilizzando tutti gli argomenti che stiamo trattando in questo corso). Tutti i concetti che valgono per le celle di Excel e Calc possono essere trasportati in ambito informatico con il vantaggio che vi è maggiore flessibilità per i nomi delle variabili.

Struttura delle variabili, si riparte dal concetto di bit, cifra binaria(binary digit, 0 o 1), 8 bit = 1Byte

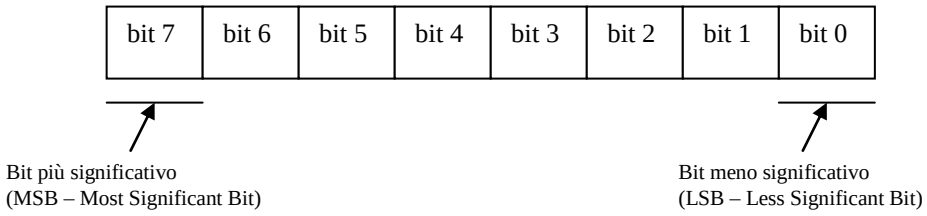
Nel sistema binario valgono inoltre le seguenti regole,

- numero configurazioni con n bit -> 2^n
- valori possibili $[0 \div (2^n-1)]$

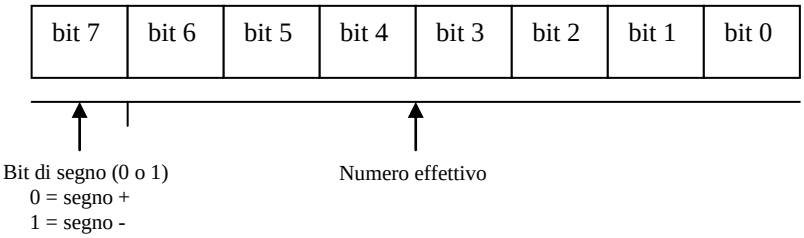
esempi

- date n = 2 cifre binarie, cioè n = 2 bit si possono avere $2^n = 2^2 = 4$ configurazioni valide, cioè 4 valori diversi che vanno da 0 a (2^n-1), cioè da 0 a (2^2-1), cioè da 0 a 3 (i valori binari saranno i seguenti, 00, 01, 10, 11)
- date n = 8 cifre binarie, cioè n = 8 bit si possono avere $2^n = 2^8 = 256$ configurazioni valide, cioè 256 valori diversi che vanno da 0 a (2^n-1), cioè da 0 a (2^8-1), cioè da 0 a 255

(valori binari, 00000000, 00000001, 00000010, 00000011, 00000100, …, 11111111)



- bit di segno, il più a sinistra di un insieme di bit è usato per indicare il segno di un numero (+ o -)
  se vale 0 si è in presenza di un numero positivo, quindi 0 = segno +
  se vale 1 si è in presenza di un numero negativo, quindi 1 = segno –



Tipi di variabili, numeriche(numeri interi, reali) o di caratteri (variabile carattere, stringa)

variabili carattere -> 1byte = 8 bit

$2^8 = 256$ configurazioni valide, cioè 256 valori diversi che vanno da 0 a (2^8-1), cioè da 0 a 255 (vedi codice ASCII)

variabili stringa -> sono un insieme di caratteri

variabili per numeri interi -> 2byte = 16bit

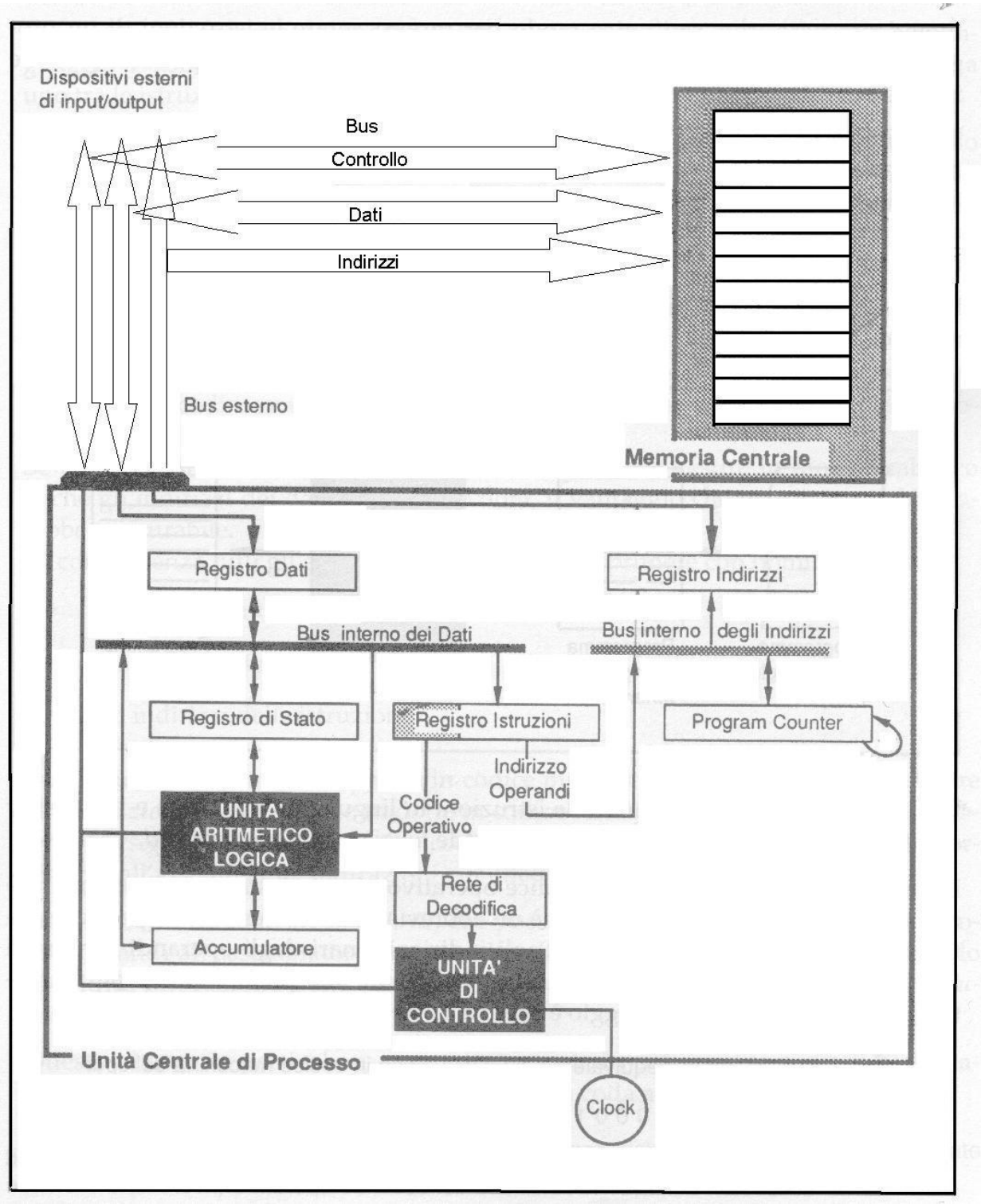
$2^{16} = 65536$ configurazioni valide, cioè 65536 valori(numeri) diversi che vanno da 0 a $(2^{16}-1)$, cioè da 0 a 65535

se però si considera il primo bit a sinistra per il segno allora i bit da considerare per le configurazioni scende a 15, perciò $2^{15} = 32768$ configurazioni valide, cioè 32768 valori(numeri) diversi che vanno da 0 a $(2^{15}-1)$, cioè da 0 a 32767
siccome però il bit di segno può anche dare numeri negativi i valori andranno da –32768 a +32767

variabili per numeri reali -> 4byte = 32bit (numeri espressi in virgola mobile)

Dichiarazione delle variabili, bisogna dichiararle all'inizio del programma, all'inizio del programma bisogna "dire" al computer: "Fra poco eseguirò dei calcoli, quindi lascia dello spazio libero nella RAM per i passaggi e i risultati", se non si fa questo il calcolatore poi non sa dove mettere tali valori e segnalerà un errore (analogia: dichiarare le variabili è un po' come dover affittare dei locali per ufficio, la ricerca va fatta in anticipo se non si vuole poi rimanere per strada!)

3 - Architettura di un microprocessore(µP) generico



Oss: il µP è fatto così non c'è niente da capire, come in un'automobile c'è il volante o il cruscotto qui ci sono i registri, l'ALU,… -> il µP è una macchina anche se elettronica !

3a. I registri

I registri sono delle piccole aree di memoria interne alla CPU e contengono i dati temporaneamente; a seconda della struttura si dividono in registri di tipi PIPO(Parallel In Parallel Out) o FIFO(First In First Out).
<Schema di un latch S-R> per un singolo bit di un registro (pag. 22)
Come abbiamo visto, la CPU è dotata di una propria memoria locale, composta da diversi registri con funzioni ben definite. In questo paragrafo esamineremo dettagliatamente le funzioni logiche svolte dai principali registri che compongono la memoria locale della CPU, prestando particolare attenzione al modo in cui vengono coinvolti nell'esecuzione delle istruzioni.

- I registri di uso generale

Questi registri, la cui dimensione in bit indica l'ampiezza di parola *(word)* del processore, sono quelli in cui vengono depositati operandi e risultati intermedi dell'elaborazione.
Tra i registri di uso generale riveste particolare importanza il registro accumulatore, in quanto è l'unico a poter essere coinvolto in operazioni particolari come quelle di lettura e scrittura sulle porte di I/O.

- Il registro istruzioni

Es. di istruzione : printf("%d", var);
Cod. operat. Indir. oper.(1000)
Si tratta di un registro in cui ogni istruzione di un programma, dopo essere stata letta dalla memoria, rimane depositata per il tempo necessario alla sua esecuzione.
Questo registro, in accordo con la struttura interna delle istruzioni in codice macchina, è suddiviso in due parti:
- una, collegata con un decodificatore e quindi indirettamente con la rete logica di controllo, atta a contenere il *codice operativo* dell'istruzione;
- l'altra, direttamente collegata con il bus interno, destinata a contenere l'indirizzo (o gli indirizzi) dell'operando (o degli operandi).

- Il registro contatore di programma

Il processore, nel corso dell'esecuzione di un programma, deve essere in grado di reperire e prelevare dalla memoria, di volta in volta, la singola istruzione da eseguire. Pertanto, poiché ogni istruzione è reperibile in memoria attraverso il proprio indirizzo, la CPU deve possedere un registro, il *program counter* all'interno del quale vengono generati gli indirizzi delle istruzioni da eseguire.
Dato che le istruzioni sono registrate in locazioni contigue, il program counter deve *autoincrementarsi* ad ogni passo del numero di locazioni occupate dall'istruzione in esecuzione, per permettere di passare automaticamente all'istruzione successiva.
L'incremento del program counter avviene subito dopo il caricamento dell'istruzione nel registro istruzioni (fase di *fetch)* e quindi in fase di esecuzione *(execute)* contiene l'indirizzo della *successiva* istruzione da eseguire.

- Il registro dati e il registro indirizzi

Si tratta di registri che hanno il compito di collegare la CPU con i bus esterni e sono spesso indicati come *buffer di ingresso/uscita* e *buffer degli indirizzi.*
Es. di buffer : stampante,masterizzatore

- Il registro stack pointer

In generale i calcolatori dispongono in memoria di una particolare area, detta *pila di sistema (stack),* che viene usata in situazioni particolari, come la chiamata a un sottoprogramma, per conservare lo stato del processore e dei registri.
La gestione di quest'area particolare di memoria viene realizzata attraverso un particolare registro, detto *stack pointer,* che consente di accedere in ogni momento all'elemento di testa della pila.

- Il registro di stato

È un registro in cui ogni bit rappresenta logicamente un indicatore *(flag)* che consente di conservare particolari informazioni relative alle condizioni verificatesi nel corso dell'ultima operazione eseguita.
Tra i flag più significativi citiamo:
- il fiag *di carry,* che viene posto a 1 quando un'operazione aritmetica produce un riporto (positivo o negativo), mentre assume il valore 0 negli altri casi;
- il *flag di overflow,* che viene posto a 1 se il risultato di un'operazione aritmetica tra numeri relativi oltrepassa la capacità di elaborazione della ALU, cioè quando il bit più significativo trabocca sul bit di segno;

- il *fiag di zero,* che vale 1 quando il risultato dell'ultima operazione effettuata è zero e invece vale 0 in tutti gli altri casi;
- il *flag di negatività,* che riproduce il bit di segno del risultato.

Analizzando i singoli bit di questo registro è possibile subordinare l'esecuzione di una o più istruzioni al verificarsi di un certo evento (per esempio eseguire la divisione solo se il risultato dell'ultima operazione svolta è stato diverso da zero).

3b. I BUS

Dato che l'implementazione reale di un altissimo numero di collegamenti crea diversi problemi per quanto riguarda l'affidabilità, dal punto di vista tecnico si è preferito optare per una soluzione alternativa che permettesse di razionalizzare la trasmissione delle informazioni, collegando tutte le unità del calcolatore ad uno stesso sistema di trasmissione, detto *bus,* in cui si possono distinguere:

- un insieme di *linee bidirezionali* dedicate alla trasmissione dei dati tra le varie unità *(bus dati);*
- un insieme di *linee unidirezionali,* dedicate alla trasmissione di indirizzi, tramite i quali il processore seleziona l'unità con la quale deve essere stabilita la comunicazione *(bus indirizzi);*
- un insieme di *linee bidirezionali* riservate alla trasmissione dei segnali di controllo *(bus di controllo).*

Le caratteristiche salienti di una struttura di comunicazione di questo tipo possono essere così riassunte:

- la risorsa bus è *unica* e consente una sola comunicazione per volta;
- durante la fase di comunicazione *una sola unità alla volta può trasmettere, mentre più unità possono* contemporaneamente *ricevere;*
- i t*rasferimenti sono comandati dall'unità di controllo* che abilita di volta in volta le porte di accesso dalle unità al bus e viceversa.

Notiamo, inoltre, che l'operazione di trasferimento dati necessita di tre segnali di sincronizzazione: la *richiesta* del bus da parte di un'unità, la *concessione* della risorsa da parte dell'unità di controllo e un terzo segnale emesso dal richiedente per avvertire l'unità di controllo che l'operazione di trasmissione è terminata.

3c. Tipi di istruzioni in assembler

Campi di un'istruzione assembler

	Etichetta	Codice operativo	Campo operandi	Commento
Es.	INIZIO:	MOV	AX,BX	; AX=BX

Tipi di istruzioni in assembler

- I. di trasferimento dati — es. MOV
- I. aritmetiche — es. ADD, SUB, MUL, DIV
- I. logiche — es. AND, OR, NOT
- I. di scorrimento — es. Shift (SHL, SHR), Rotate(ROL, ROR)
- I. di manipolazione di singoli bit — es. SET pos
- I. per modificare la sequenza di elaborazione — es. CMP(I. di confronto), JMP

3d. Fasi di elaborazione di un'istruzione

2 Fasi : - Fetch : caricamento istruzione nel reg. istruzioni
- Execute : esecuzione istruzione

Esplicitazione delle fasi di fetch e di execute sul seguente programma

```
Indir. di memoria      .DATA
        0000                  DATO1 DB 5
        0001                  DATO2 DB 7
        0010                  DATO3 DB ?
                       .CODE
           0                  [2]inizio:  MOV AL,DATO1
           1                              ADD AL,DATO2
           2                              MOV DATO3,AL
           3                              JMP INIZIO
```

[2] Per semplicità in questo caso si trascura il gruppo di istruzioni poste subito dopo il .CODE

Fetch viene trasferito nel registro istruzioni il contenuto della locazione indirizzata dal program counter (0).

Execute viene incrementato il valore del program counter (da 0 a 1), viene decodificato il codice operativo (MOV) e viene eseguita l'istruzione. Nel nostro caso,viene inviato nel registro indirizzi il contenuto del campo operando dell'istruzione (0 - DATO1), viene eseguito un ciclo di lettura dalla memoria che trasferisce nel registro dati il valore contenuto nella locazione 0 - DATO1 (5) e viene trasferito il contenuto del registro dati nell'accumulatore.

Fetch viene trasferito nel registro istruzioni il contenuto della locazione indirizzata dal program counter (1).

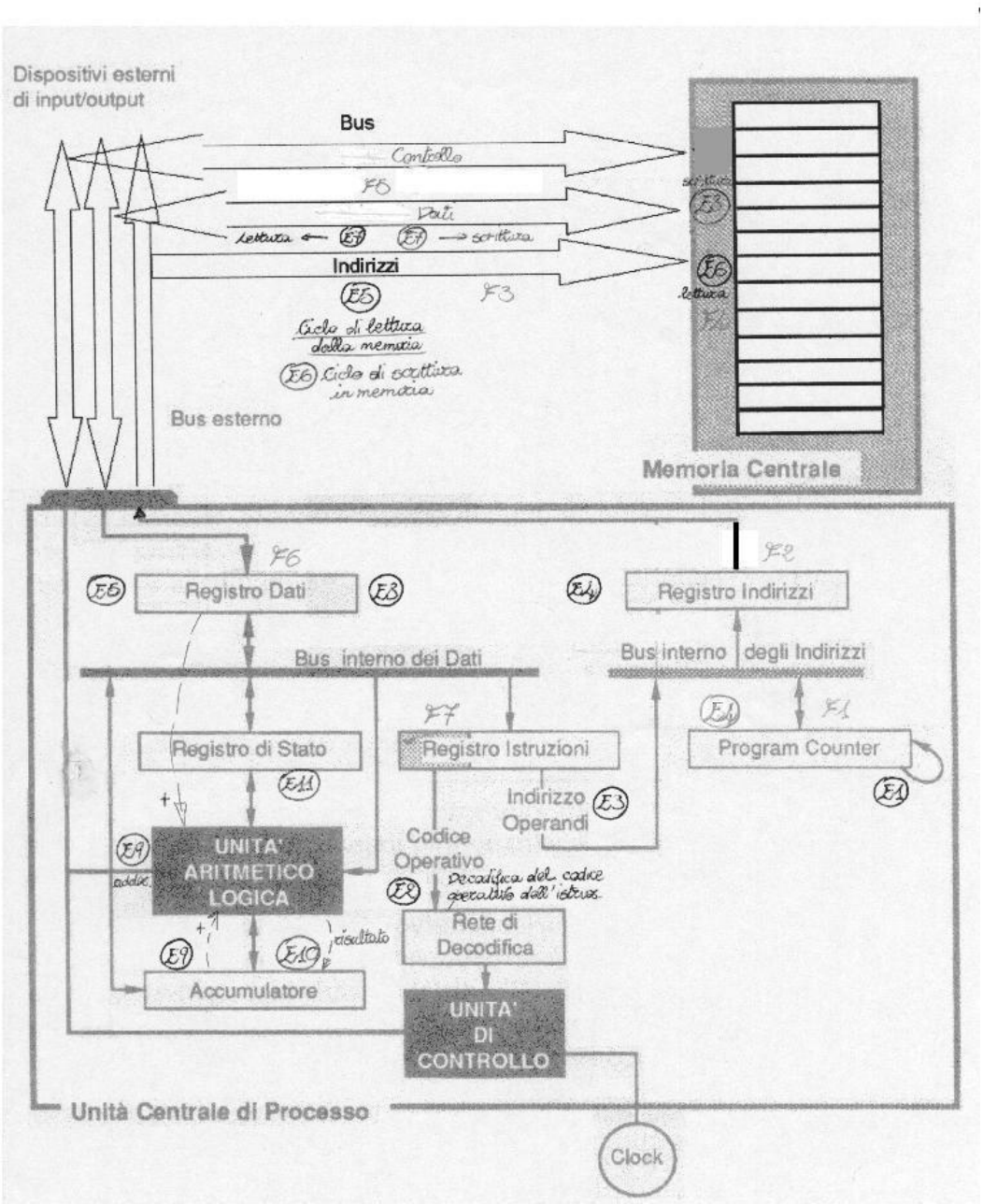
Execute viene incrementato il valore del program counter (da 1 a 2),viene decodificato il codice operativo (ADD) dell'istruzione e viene eseguita l'istruzione.Nel nostro caso, viene inviato nel registro indirizzi il contenuto del campo operando dell'istruzione (0001 – DATO2)viene eseguito un ciclo di lettura dalla memoria, che trasferisce nel registro dati il valore contenuto nella locazione 0001 (7), viene inviato alla ALU il codice dell'operazione da compiere(addizione), la ALU preleva il contenuto del registro dati e dell' accumulatore,esegue l'operazione, trasferisce il risultato(12) nell'accumulatore e modifica in base a esso i flag del registro di stato.

Fetch viene trasferito nel registro istruzioni il contenuto della locazione indirizzata dal program counter (2).

Execute viene incrementato il valore del program counter (da 2 a 3),
viene decodificato il codice operativo (MOV) e viene eseguita l'istruzione. Nel nostro caso,viene inviato nel registro indirizzi il contenuto del campo operando dell'istruzione (0010 – DATO3), viene inviato al registro dati il contenuto dell' accumulatore, viene eseguito un ciclo di scrittura in memoria, che trasferisce dal registro dati il valore (12) nella locazione 0010 (DATO3).

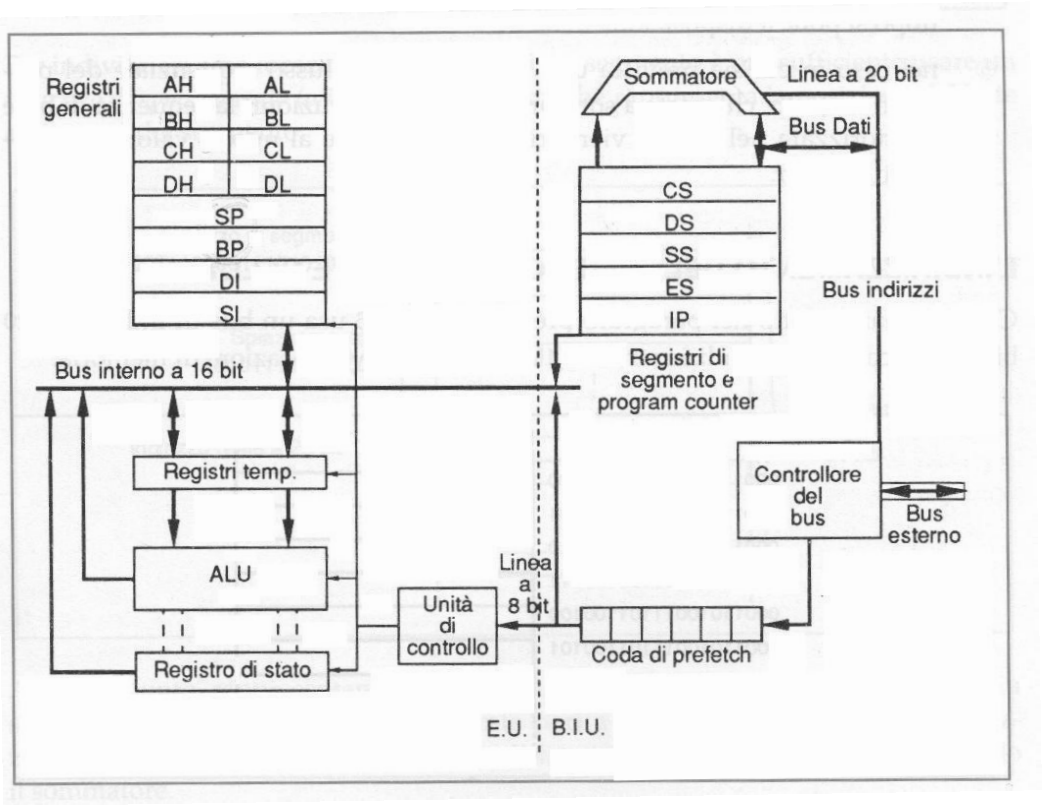
Fetch viene trasferito nel registro istruzioni il contenuto della locazione indirizzata dal program counter (3).

Execute viene incrementato il valore del program counter (da 3 a 4), viene decodificato il codice operativo (JMP) e viene eseguita l'istruzione. Nel nostro caso,viene scritto nel program counter il contenuto del campo operando dell'istruzione (0): il programma riprende quindi dalla prima fase di fetch.



4 - Architettura del microprocessore INTEL 8086

Passiamo ora a studiare un microprocessore specifico l'Intel 8086



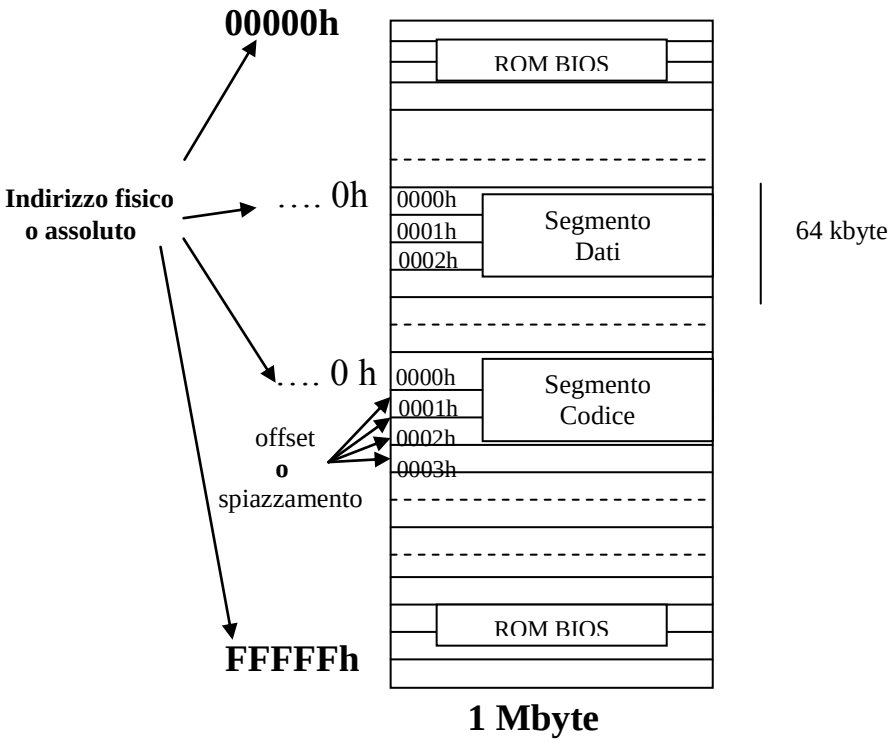
Oss : il µP è fatto così non c'è niente da capire,come un tornio è costruito in modo da modellare pezzi meccanici, il µP è costruito in modo da eseguire programmi -> il µP è una macchina anche se elettronica !

A questo punto vi consiglio di confrontare lo schema del µP8086 (quello qui sopra) con quello del µP generico

Caratteristiche generali :

1- BUS DATI a 16 bit
2- BUS INDIRIZZI a 20 bit (2^{20}=1 Mb di memoria, da 00000h a FFFFFh)
3- MULTIPLEXAGGIO bus dati e indirizzi : dati ed indirizzi viaggiano sulle stesse linee fisiche,ma in tempi diversi
4- 14 registri interni a 16 bit (4 utilizzabili come coppia di registri a 8 bit)
5- 2 modalità di funzionamento : minima e Massima
6- Struttura a Pipeline (EU e BIU) : mentre la EU sta eseguendo un'istruzione la BIU ne precarica altre nella coda di prefetch

4a. Indirizzamento della memoria da parte del μP 8086



Rom caricata nella Ram all'accensione del PC con operazione di shadowing (ombra)

Indirizzo fisico o assoluto : è lo spostamento in byte dall'inizio della memoria

Problema : bus indirizzi a 20 bit,mentre tutti i registri sono a 16 bit,come posso allora indirizzare 1 Mb di memoria (=2^{20} locaz da 1 byte)?
Con 16 bit posso indirizzare solo fino a 64 kbytes.

Soluzione dei progettisti Intel : la memoria non la utilizzerò tutta,una parte sarà usata dal Rom-Bios,mentre il resto potrò utilizzarla suddividendola in varie parti(segmenti) adibite ognuna ad un compito,
es., Segmento dati o codice
Come regola fissa però,si impone che ogni segmento inizi SEMPRE da una locazione di memoria il cui indirizzo termini con 4 zeri binari (…0000b),cioè uno 0 esadecimale (…0h).
All'interno del segmento,il conteggio delle locazioni di memoria può ricominciare da zero,uno,due → offset o spiazzamento

Offset o spiazzamento : è lo spostamento in byte dall'inizio del Segmento. In memoria però,si può accedere solo con l'indirizzo fisico e non con l'offset,quindi bisognerà trovare il modo di convertire l'offset in indirizzo fisico con l'aiuto dell'indirizzo fisico di inizio segmento.
Le 4 cifre esadecimali prima dello 0h sono contenute nei registri di segmento CS,DS,SS,ES,mentre l'offset è contenuto nei registri puntatori : per esempio quello del code segment è in IP

Calcolo dell'indirizzo fisico:
es, nel caso di un programma,le 4 cifre che precedono lo 0h sono contenute in CS, il cui contenuto nel momento del calcolo dell'indirizzo fisico,viene shiftato 4 volte verso sinistra dal sommatore,introducendo così i 4 zeri binari,dopodichè si aggiunge l'offset contenuto in IP,cioè il program counter,che essendo a 16 bit sarà formato da 4 cifre esadecimali.

se CS → 1F58 h
allora l'inizio del segmento di codice sarà…

```
CS →   1F580  h  +
IP →    0000  h  =   1° istruz. Pgm
       ----------
       1F580  h
```

che giustamente coincide con l'inizio del segmento la 2° istruzione avrà indirizzo fisico 1F581h,la 3° 1F582h
e così via…fino a che l'IP avrà raggiunto il suo massimo, che è FFFFh,

```
CS →   1F580 h  +
IP →    FFFF h  =
       ----------
       2F57F h   →   fine segmento
```

quindi un segmento potrà avere una dimensione massima di 2^{16} byte = 64 kbytes e questo varrà per tutti i segmenti

A questo punto consiglio a chi ha un minimo di dimestichezza con i sistemi di numerazione di fare qualche: esercizio sul calcolo dell'indirizzo fisico anche passando dalla base 2

4b. I registri interni del µP 8086

Il microprocessore 8086 utilizza, al suo interno,14 registri a 16 bit che, in base alla loro funzione, possono essere suddivisi in quattro categorie: i registri di transito, i registri puntatori, i registri di segmento e il registro dei fiag.

I registri di transito

I 4 registri di transito, chiamati AX,BX, CX, DX vengono utilizzati come area per depositare temporaneamente i risultati intermedi e gli operandi delle operazioni logiche ed aritmetiche compiute dalla CPU.
Essi possono a loro volta essere suddivisi in coppie di registri a 8 bit indirizzabili
separatamente e, in particolare, AH, BH, CH, DH rappresentano il byte più significativo *(High)* e AL, BL, CL, DL quello meno significativo *(Low).*
Oltre che come registri generali, i registri di transito vengono utilizzati all'interno del microprocessore per particolari funzioni:

AX - *registro accumulatore,* interviene in tutte le operazioni di I/O ed è il registro più utilizzato nelle operazioni di calcolo;
BX - *registro base,* è usato negli indirizzamenti indiretti da solo o in combinazione con i registri indici;
CX - *registro contatore,* è usato come contatore di ripetizione per il controllo delle iterazioni e dei movimenti di dati ripetuti;
DX - *registro dati,* è utilizzato per immagazzinare dati e come riferimento in diretto nell'i/O.

I registri di segmento

Come abbiamo visto, l'indirizzo completo di una locazione di memoria viene formato a partire dall'indirizzo di un segmento e da quello di spiazzamento allo interno del segmento stesso.
In particolare, in ogni momento sono attivi in memoria 4 segmenti a cui sono associati i seguenti 4 registri di segmento CS, DS, SS, ES dove:

CS - individua il segmento destinato a contenere il codice del programma *(code seginent);*
DS - individua il segmento destinato a contenere i dati del programma *(data segrnent);*
SS - individua il segmento dedicato allo stack, una zona di lavoro privilegiata di cui parleremo approfonditamente in seguito *(stack segment);*
ES - individua un segmento ausiliario utilizzato per completare il segmento destinato, in modo da poter usare per questi più di 64Kb di memoria *(extra segment).*

I registri puntatori

Così come i registri di segmento vengono usati per individuare i segmenti da 64Kb all'interno della memoria, i 5 registri puntatori servono per individuare, attraverso l'indirizzo di spiazzamento, una locazione all'interno di uno specifico segmento. Questi 5 registri sono:
IP - *(Istruction Pointer)* rappresenta il program counter dell'8086 e fornisce l'indirizzo di riferimento, all'interno del segmento individuato da CS, dell'istruzione successiva da eseguire;
SP - *(Stack Pointer)* rappresenta il puntatore alla testa dello stack nel segmento individuato da SS,
BP -*(Base Pointer)* rappresenta spesso Io spiazzamento di una locazione all'interno del segmento di stack e viene usato per le operazioni di trasferimento dati all'interno di tale segmento;
SI - *(Source index)* viene usato come registro di spiazzamento negli indirizzamenti all'interno del segmento dati individuato da DS;
DI- *(Destination Index)* viene usato come registro di spiazzamento negli indirizzamenti all'interno del segmento dati individuato da DS o del segmento extra individuato da ES.

Il registro dei flag

Il quattordicesimo registro dell'8086, detto registro di stato o *registro dei flag, è* formato da un insieme di bit di controllo *(flag)* che vengono attivati ed esaminati come entità indipendenti.

Dei 16 bit che compongono il registro solo 9 risultano significativi e tra questi 6 rappresentano *flag di stato* (ZF, CF PF, SF, OF, AF) e indicano le condizioni derivate dal risultato di istruzioni di elaborazione, mentre 3, detti *flag di controllo* (TF,IF,DF), possono essere modificati da programma per condizionare il comportamento della CPU.

OF - Indica se il risultato dell'ultima operazione aritmetica ha generato un riporto. Questo flag è condizionato dalle operazioni che utilizzano la rappresentazione in complemento a 2.

OF=O assenza di riporto *(overflow);*
OF=1 presenza di riporto

DF - Indica, nelle operazioni di manipolazione delle stringhe, la direzione degli incrementi / decrementi.

DF=O incremento
DF=1 decremento.

IF - Determina se le interruzioni sono abilitate o no.

IF=O disabilitate
IF=1 abilitate

TP - Permette di porre la CPU in una modalità che permette l'esecuzione delle istruzioni un passo alla volta *(single step)* facilitando le operazioni di verifica di un programma *(debugging).*

TF=O non attivo
TF=1 attivo

SF - Indica se il risultato dell'ultima operazione aritmetico-logica eseguita segno positivo o negativo.

SF = 0 risultato positivo
SF = 1 risultato negativo

ZF - Indica se il risultato dell'ultima operazione aritmetico-logica eseguita è zero o no.

ZF=O risultato non zero
ZF=l risultato zero.

AF - È utilizzato dalle operazioni che operano con l'aritmetica BCD e segnala l'eventuale riporto fra il 3° e il 4° bit

AF=0 risultato senza riporto
AF=1 risultato con riporto

PF - Indica se l'ultima operazione aritmetico-logica eseguita ha dato un risultato con parità pari (numero di i presenti nel risultato è pari)

PF=0 risultato con parità dispari
PF=1 risultato con parità pari

CF - Indica se il risultato dell'ultima operazione aritmetico-logica eseguita ha dato riporto sul bit più significativo.

CF=0 risultato senza riporto
CF=1 risultato con riporto

FIag di carry e di overflow

A seguito di istruzioni aritmetiche in complemento a due il flag di overflow indica il verificarsi di un overflow, assume il valore 1 quando il risultato eccede la capacità di rappresentazione in complemento a due di un registro.
I flag di carry ed overflow hanno significati simili che possono confondersi, ma in realtà sono due flag ben distinti.

Gli esempi che seguono, relativi ad operazioni su numeri ad 8 bit, ne chiariscono le differenze.

a) Somma di due numeri positivi in complemento a due

```
   01001000 +      (72)
   01000000 =      (64)
   ------------
(0) 10001000       (136)
```

C=0 perché non c'è riporto nel risultato
O=1 perché il risultato della somma dei due numeri positivi è in complemento a due negativo e quindi scorretto (somma>127)

b) Somma di due numeri negativi in complemento a due

```
   11110101 +    (-11)
   11111000 =     (-8)
   ------------
(1) 11101101     (-19)
```

C=1 perché c'è riporto
O=0 perché nella somma di due numeri negativi si è ottenuto un risultato negativo corretto (somma - 128)

c) Somma di due numeri negativi in complemento a due

```
   10001100 +   (-116)
   10010010 =   (-110)
   ------------
(1) 00011110    (-226,+30)
```

C=1 perché c'è riporto
O=1 perché nella somma di due numeri negativi si è ottenuto un risultato positivo (somma -128)

É facile convincersi inoltre che nella somma di due numeri positivi piccoli (somma < 127) non ha luogo né overflow, né carry.
In definitiva si verifica un overflow quando la somma di due numeri positivi produce un risultato negativo e, viceversa, quando la somma di due numeri negativi produce un risultato positivo;pertanto l'overflow segnala una situazione che deve essere corretta.
Occorre osservare che il microprocessore non avverte la differenza tra numeri in binario e in complemento a due ed è solo il programmatore che sa in quale forma sta trattando i suoi dati, e deve preoccuparsi quindi di verificare il flag di carry o quello di overflow a seconda dei casi.
Pertanto se nelle somme esemplificate i numeri si considerano senza segno, il flag di overflow può essere trascurato.

5 - Hardware del microprocessore INTEL 8086

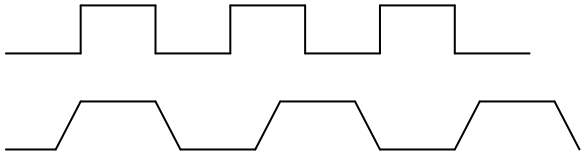
Perché studiare questa parte?

1. Capire l'interazione fra μP (Cpu) da una parte e memoria (o I/O) dall'altra (anche durante le fasi di fetch e di execute)
2. Capire come avviene la sincronizzazione dei segnali di controllo e del bus dati/indirizzi fra Cpu e memoria (o I/O)
3. Capire cosa succede a livello materiale/fisico/elettronico fra μP(Cpu) e Memoria (o I/O) durante un ciclo di lettura/scrittura in memoria (o I/O)

5a. Richiami sulle onde periodiche

Onde periodiche : sono onde che si ripetono uguali a se stesse ad intervalli di tempo regolari.

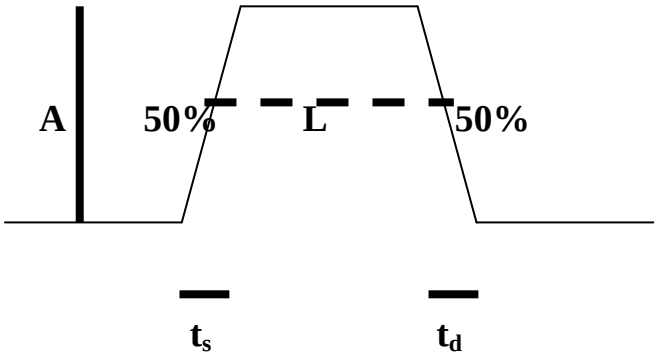
Es.



onda quadra ideale

onda quadra più reale

Onde quadre più reali :



t_s = tempo di salita (o fronte di salita)
t_d = tempo di discesa (o fronte di discesa)

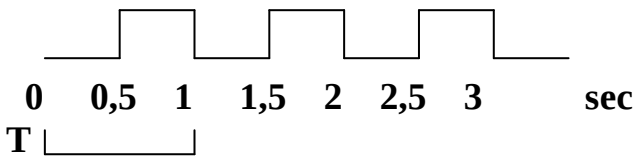
A = ampiezza
L = larghezza dell'impulso

Periodo (T) : è il tempo che un'onda periodica impiega a compiere un ciclo completo. Si misura in secondi

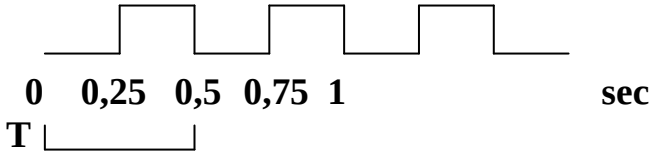
Frequenza (f) : è il nomero di cicli completi che un'onda periodica compie nell'unità di tempo (1 secondo) . Si misura in Hertz (1 Hz = 1 ciclo/1 sec)

$$f = \frac{1}{T}$$

Es.



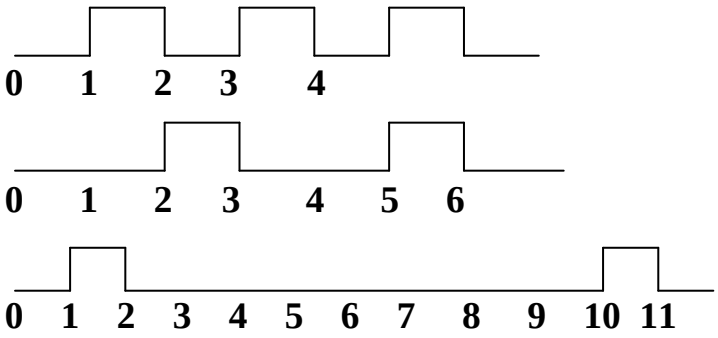
f = 1 Hz = 1 ciclo/sec



f = 1/0,5 Hz = 2 Hz = = 2 cicli/sec

Duty-cycle (d.c.) : ci sono vari tipi di onde quadre

Es.



d.c. = (1/2)*100% = 50%

d.c. = (1/3)*100% = 33%

d.c. = (1/10)*100% = 10%

Per distinguerle è necessario introdurre un nuovo parametro il duty-cycle

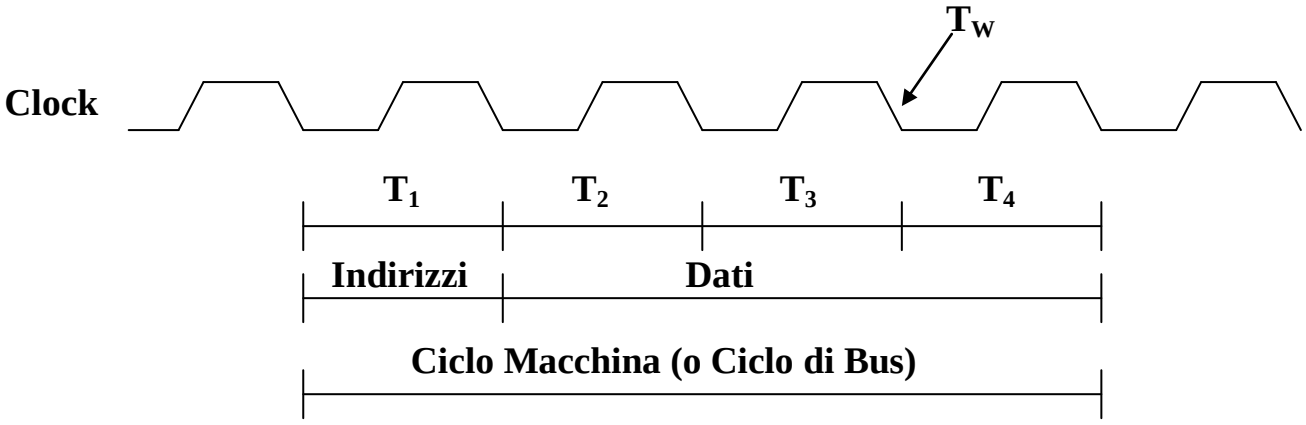
$$\text{Duty-cycle} = \frac{\text{Larghezza impulso}}{\text{Periodo}} * 100\%$$

Quindi il duty-cycle dà informazioni sul rapporto fra la larghezza d'impulso dell'onda ed il periodo dell'onda stessa.
In poche parole ci dice quanto è grande la larghezza d'impulso rispetto al periodo dell'onda.

5b. Il clock ed i cicli macchina

Un sistema a microprocessore,essendo formato da svariate e molteplici componenti,deve avere un segnale di sincronizzazione in grado di ordinare le varie operazioni svolte dalle suddette componenti (un po' come avviene in un complesso quando tutti i musicisti devono andare allo stesso tempo per poter udire una melodia sensata).
Tale segnale è il cosiddetto CLOCK generato dal μP che quindi si dice macchina sincrona.

Ogni operazione elementare svolta dal μP (in generale cicli di lettura/scrittura in memoria o in I/O) è formata da 4 cicli di clock T_1,T_2,T_3,T_4 che definiscono il cosiddetto Ciclo Macchina (o Ciclo di Bus).
Il numero di cicli di clock del ciclo macchina è aumentabile, aggiungendo indefiniti *cicli di attesa* T_W fra T_3 e T_4 ,nel caso il μP colloqui con periferiche lente.



Come indicato dalla figura, sul bus dati/indirizzi multiplexato,il μP porrà dapprima gli indirizzi validi per l'accesso in memoria durante il ciclo T_1 , mentre negli altri cicli saranno presenti i dati

5c. I segnali esterni

La figura in fondo a questa pagina mostra la configurazione dei piedini dell'8086.
Come appare dalla figura, molti piedìni hanno due definizioni, in corrispondenza di quello che è stato definito *multiplessaggio* di alcuni segnali.
In alcuni casi, (dati-stato/indirizzi), le due definizioni appaiono necessariamente nell'ambito della stessa applicazione e pertanto devono essere previsti dispositivi hardware per demultiplessare i segna1i in altri casi la funzione di alcuni piedini, e precisamente quelli da 24 a 31,è definita univocamente dal valore del segnale applicato al piedino *33 MN/MX*, che determina se il microprocessore deve operare nel modo minimo o massimo: poiché ciò implica, come si vedrà in seguito, anche differenze a livello di sistema hardware, tale scelta resta normalmente definitiva per tutte le applicazioni che girano su quel particolare microcalcolatore.
Il funzionamento dell'8086 è sostanzialmente basato su una serie di accessi ripetuti alla memoria separati da cicli di attesa *(Idle cycles, T_1).*
Gli accessi in memoria avvengono secondo un ciclo tipico detto *ciclo di bus* costituito da quattro stati elementari,corrispondenti a periodi di clock, detti T_1,T_2,T_3 e T_4 con il possibile inserimento dì un numero indefinito di stati T_w tra T_3 e T_4 se ciò risulta necessario per la sincronizzazione con dispositivi lenti.
La descrizione dei segnali che segue fa riferimento al generico ciclo di bus, mentre in un paragrafo successivo saranno esaminate le temporizzazioni corrispondenti ai possibili diversi cicli (lettura, scrittura, interrupt, ecc.).
E' infine da tener presente che talvolta nella descrizione si resterà nel generico per non complicare troppo questo primo approccio all' hardware dell'8086: ulteriori approfondimenti su singoli punti saranno fatti nel seguito, quando necessario.

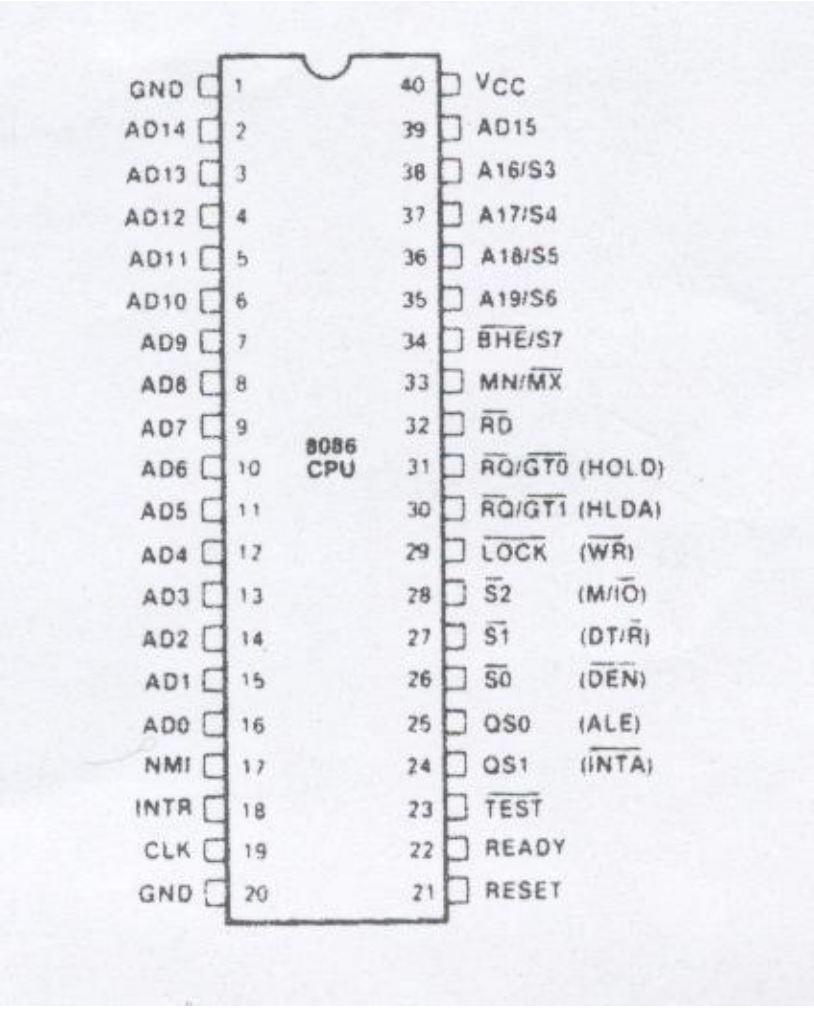


Figura - La piedinatura del chip del μP 8086

AD_{15}-ADo (Piedini 2-16 e 39, I/O)
Bus indirizzi e dati: durante T_1 su queste linee sono presenti i segnali di indirizzo per la memoria e l'I/O, mentre durante gli altri periodi del ciclo di bus le stesse linee sono dedicate ai dati. A_0 viene utilizzato insieme a *BHE** per definire se il ciclo di bus interessa il byte alto, il byte basso o la parola *(vedi BHE).* Queste linee sono attive alte e vanno in alta impedenza *(tristate)* durante il riconoscimento dell'interrupt e dell'hold.

A_{19}-A_{16}/S_6-S_3 (Piedini 35-38, O)
Indirizzi-stato: durante *T_1* su queste linee sono presenti i quattro bit più significativi dell'indirizzo, mentre durante gli altri stati *T* queste linee forniscono informazioni di stato e precisamente:
- S3 e S4 individuano il segmento utilizzato nel ciclo di bus;
- S5 è uguale al flag di interrupt;
- *S_6* = 0 indica il collegamento dell'8086 al bus.

Queste linee vanno in alta impedenza durante il riconoscimento di hold.

*BHE** /S7 (Piedino 34, O)
Bus High Enable/Status: durante *T_1* è disponibile il segnale *BHE*, durante gli altri stati *T* il segnale S7. *BHE* abilita il passaggio dei dati sulla metà più significativa del bus *(*D_{15}-D_0*)* insieme ad *A_0 è* utilizzato per determinare il trasferimento tra CPU e memoria di un singolo byte o di tutta una parola a 16 bit. Anche questo piedino entra in uno stato di alta impedenza nel riconoscimento di hold. S_7 è disponibile durante T_2,T_3 e T_4:diventa 0FF se è in corso un DMA e va in basso durante T_1, durante il primo ciclo di riconoscimento di interrupt.

*RD** (Piedino 32, O)
Read: indica che il processore sta effettuando un ciclo di lettura in memoria o in I/O. E attivo basso durante T_2, T_3 e T_4 di ogni ciclo di lettura e passa in alta impedenza nel riconoscimento di hold.

READY (Piedino 22, I)
Ready, pronto: segnale generato dalla memoria o dal dispositivo di 1/0 indirizzato per indicare che ha completato l'operazione richiesta dalla CPU. Il segnale è attivo alto e deve essere sincronizzato, per esempio, dal generatore di clock 8284 per dare un segnale valido per la CPU.

TEST*(Piedino 23, I)
Tesi: segnale d'ingresso che viene esaminato dall'istruzione *WAIT*. Se *TEST* è basso, il microprocessore continua nell'esecuzione del programma, altrimenti si arresta in un stato di attesa e vi resta finché *TEST* non ritorna basso.

INTR (Piedino 18, I)
Interrupt Request: segnale di ingresso che viene testato durante l'ultimo stato *T* di ogni istruzione per verificare se vi è una richiesta pendente di interrupt e se quindi il microprocessore deve entrare in un ciclo di riconoscimento di interrupt *(Interrupt Acknowledge).* E attivo alto e può essere mascherato internamente resettando via software il bit di abilitazione dell'interruzione.

NMI (Piedino 17, I)
Non Maskable interrupt: segnale di ingresso valido sul fronte, indica la richiesta di un interrupt non mascherabile che deve essere onorato al termine dell'istruzione in corso di esecuzione

RESET(Piedino 21, I)
Reset: fa terminare immediatamente alla CPU l'attività in corso. Il segnale è attivo alto e deve essere tale per almeno quattro periodi di clock; quando ritorna basso, la CPU riprende il suo funzionamento.

CLK(Piedino 19, I)
Clock : dà la temporizzazione di base al microprocessore e al controllore di bus. Per un funzionamento ottimizzato è richiesto un duty-cycle del 33%.

V_{cc} (Piedino 40) e GND (Piedini I e 20)
Alimentazione (5 V) e Terra.

MN/MX* (Piedino 33, I)
Minimo/Massimo : indica in quale modo deve operare il microprocessore, definendo le funzioni dei piedini da 24 a 31.

Modo minimo (MN/MX*= V_{cc})

M/IO* (Piedino 28, 0)
Linea di status : diventa valido nello stato T_4 precedente il ciclo di bus e rimane valido fino allo stato T_4 del ciclo considerato e indica se il processore sta effettuando un ciclo di memoria o di I/O. Entra in alta impedenza nel ciclo di riconoscimento di hold.

WR*(Piedino 29, O)
Write : indica che il microprocessore sta effettuando un ciclo di scrittura in memoria o in I/O. Attivo basso, è valido negli stati T_2,T_3 e T_W dei cicli di scrittura e passa in alta impedenza nel ciclo di riconoscimento di hold.

INTA* (Piedino 24, O)
Interrupt Acknowledge : attivo basso durante T_2,T_3 e T_W di ogni ciclo di riconoscimento di interrupt, è usato come segnale di sincronismo per la lettura.

ALE(Piedino 25, O)
Address Latch Enable : generato dal processore per memorizzare gli indirizzi, è attivo alto durante T_1. Questo piedino non va mai in alta impedenza.

DT/R* (Piedino 27, O)
Data Transmit/Receive : segnale usato per controllare la direzione del trasferimento dei dati in connessione all'impiego di un tranceìver (trasmettitore-ricevitore) tipo 8286 o 8287. Va in alta impedenza nel ciclo di riconoscimento di hold.

DEN* (Piedino 26, O)
Data Enable : attivo basso in tutti i cicli di accesso in memoria e in I/O Nei cicli di riconoscimento di interrupt (INTA), è usato come output enable nei sistemi che utilizzano i tranceiver 8286 o 8287.

HOLD, HLDA (Piedini 31 e 32, I e O)
Hold, Hold Acknowledge : attivi alti, HOLD segnala che un altro master richiede il controllo del bus e HLDA diventa attivo a metà dello stato T_1 corrispondente al ciclo durante il quale il processore rilascia all'altro master il controllo del bus. Il microprocessore riacquista il controllo del bus non appena HOLD ritorna basso.

Modo Massimo (MN/MX*=V_{ss})

S_2*, S_1*,S_0* (Piedini 26-28, O)
Status : questi segnali sono usati dal bus controller 8288 per generare i segnali di controllo per l'accesso in memoria o ai dispositivi di I/O.
Sono attivi durante T_4, per indicare l'inizio di un ciclo di bus, e durante T_1 e T_2, mentre il ritorno allo stato passivo (S_0= S_1 = S_2 =1) durante T_3 o Tw è usato per indicare la fine di un ciclo di bus. Questi piedini vanno in alta impedenza durante il ciclo di riconoscimento di hold.

RQ*/GT_0*, RQ*/GT_1* (piedini 30 e 31, I/O)
Request/Grant : ciascuna linea è utilizzabile, separatamente, per ricevere la richiesta di bus da parte di un altro master, per trasmettere un segnale di accettazione della richiesta verso il master richiedente e per ricevere la segnalazione da parte dell'altro master della fine dell'esigenza di controllo del bus. Ognuno dei segnali è realizzato tramite un impulso basso di larghezza pari a quella del clock e la linea a maggiore priorità è quella con
indice O. La CPU risponde affermativamente alla richiesta di bus alla fine del ciclo di memoria in corso se sono verificate le seguenti condizioni:

a) la richiesta è avvenuta entro T_2
b) il ciclo in corso non è la lettura di un byte di indirizzo dispari o l'inizio di INTA;
c) l'istruzione in corso non è locked.

LOCK* (Piedino 29, O)
Lock : il piedino, attivo basso, impedisce che altri master possano prendere il controllo de1 bus. Il segnale LOCK* è attivato con l'inserimento, via software, del corrispondente prefisso su un'istruzione e resta valido fino al termine dell'istruzione successiva.

QS_0 , QS_1 (Piedini 24 e 25, O)
Queue Status: indica all'esterno lo stato della coda dei byte di istruzione.

5d. I sistemi basati sull'8086

La realizzazione di sistemi basati sull'8086 richiede al progettista di prendere in considerazione una serie di problematiche strettamente legate all'architettura del microprocessore e ai segnali esterni.

Il multiplessaggio dati-indirizzi

Un primo problema è rappresentato dall'esistenza di segnali multiplessati corrispondenti a dati e indirizzi: la CPU mette sul bus AD gli indirizzi durante lo stato T_1 utilizza il bus per il trasferimento dei dati e per la presentazione di alcuni bit di status durante gli altri stati T e genera un segnale valido, ALE, quando gli indirizzi sono stabili.
In ogni sistema è quindi necessario provvedere innanzitutto a demultiplessare questi segnali e a questo proposito l'intel prevede, come latch bistabili a 8 bit non invertenti o invertenti, l'8282 e l'8283. Lo schema tipico è mostrato in fig.1 : gli ingressi sono collegati ai segnali AD e BHE* che vengono memorizzati in corrispondenza del fronte di discesa di ALE. Come risultato si ha un bus degli indirizzi e un bus multiplessato dati-status-indirizzi che girano per tutto il sistema e questa soluzione è quella che meglio risponde alle specifiche richieste di un sistema multiprocessore: in alternativa, si può pensare di far girare per tutto il sistema solo il bus multiplessato e di effettuare il demultiplessaggio localmente secondo lo schema di fig. 2.
Anche la separazione dei dati dagli indirizzi può avvenire sia centralmente, in corrispondenza della CPU, creando un bus dati separato dal bus indirizzi, sia localmente; in corrispondenza di ogni periferica.
La prima soluzione può essere realizzata con l'impiego di tranceiver tipo 8286/8287 collegati come previsto in fig. 3. I segnali utilizzati per comandare la memorizzazione dei dati sono DEN*e DT/R* previsti nel modo di funzionamento minimo, per cui questa soluzione è possibile nella forma indicata solo in questo modo.

Riassumendo :

Nel μP 8086 vi sono segnali diversi che viaggiano sullo stesso canale fisico(multiplexaggio),nella fattispecie si intendono i segnali degli indirizzi e quelli dei dati,quindi bisogna trovare un modo per capire quando passa un segnale od un altro,in questo caso il problema è risolto,come gia accennato,facendo prima passare gli indirizzi durante T_1 e poi i dati,non rimane quindi che dividerli (demultiplexarli) tramite dei circuiti apposta.
In particolare gli indirizzi saranno memorizzati in dispositivi chiamati latch ed i dati saranno lasciati transitare invece dai tranceiver.

Vi sono quindi 2 tipi di demultiplexaggio,centrale cioè all'uscita dal μP o periferico cioè all'ingresso dei dispositivi da collegare.
Il demultiplexaggio dati/indirizzi quindi potrà utilizzare o una di queste due tecniche (fig. 2 o 3) o entrambe contemporaneamente (fig. 1)

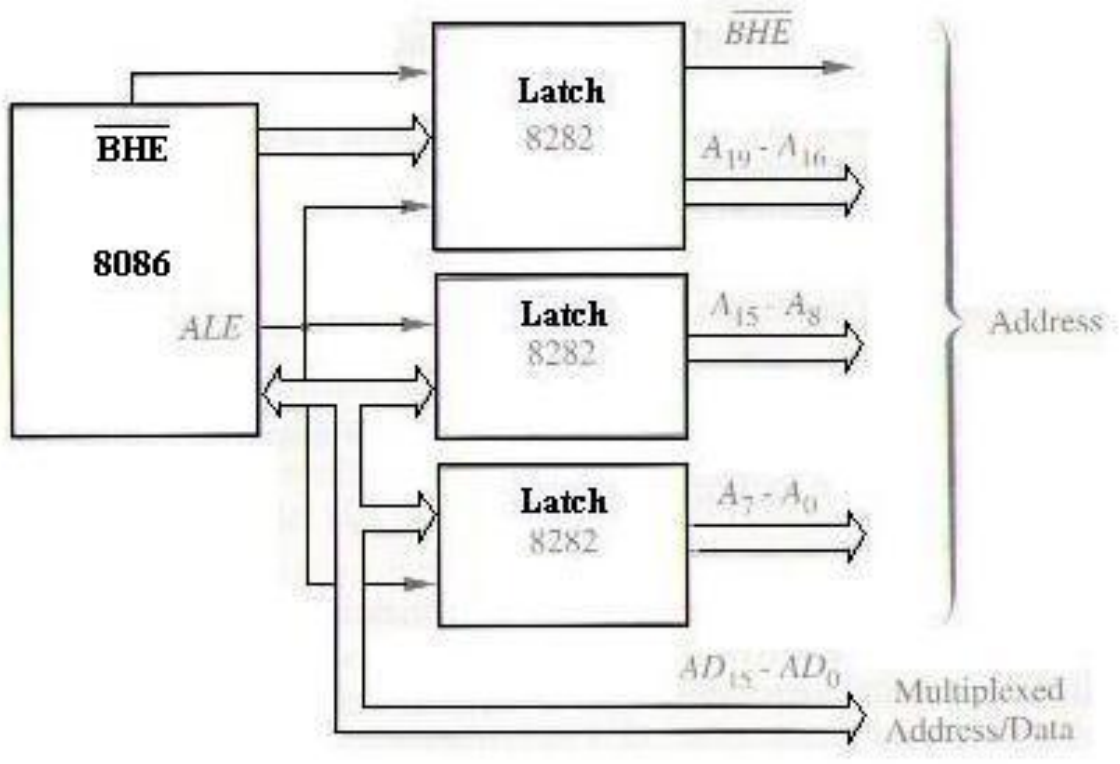


Fig. 1 – Sistema con demultiplexaggio centrale per gli indirizzi e periferico per i dati

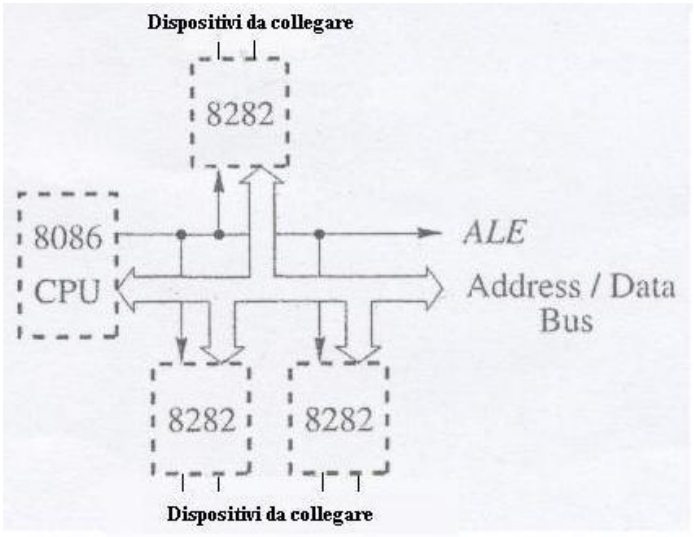


Fig. 2 – Sistema con demultiplexaggio periferico per dati e indirizzi

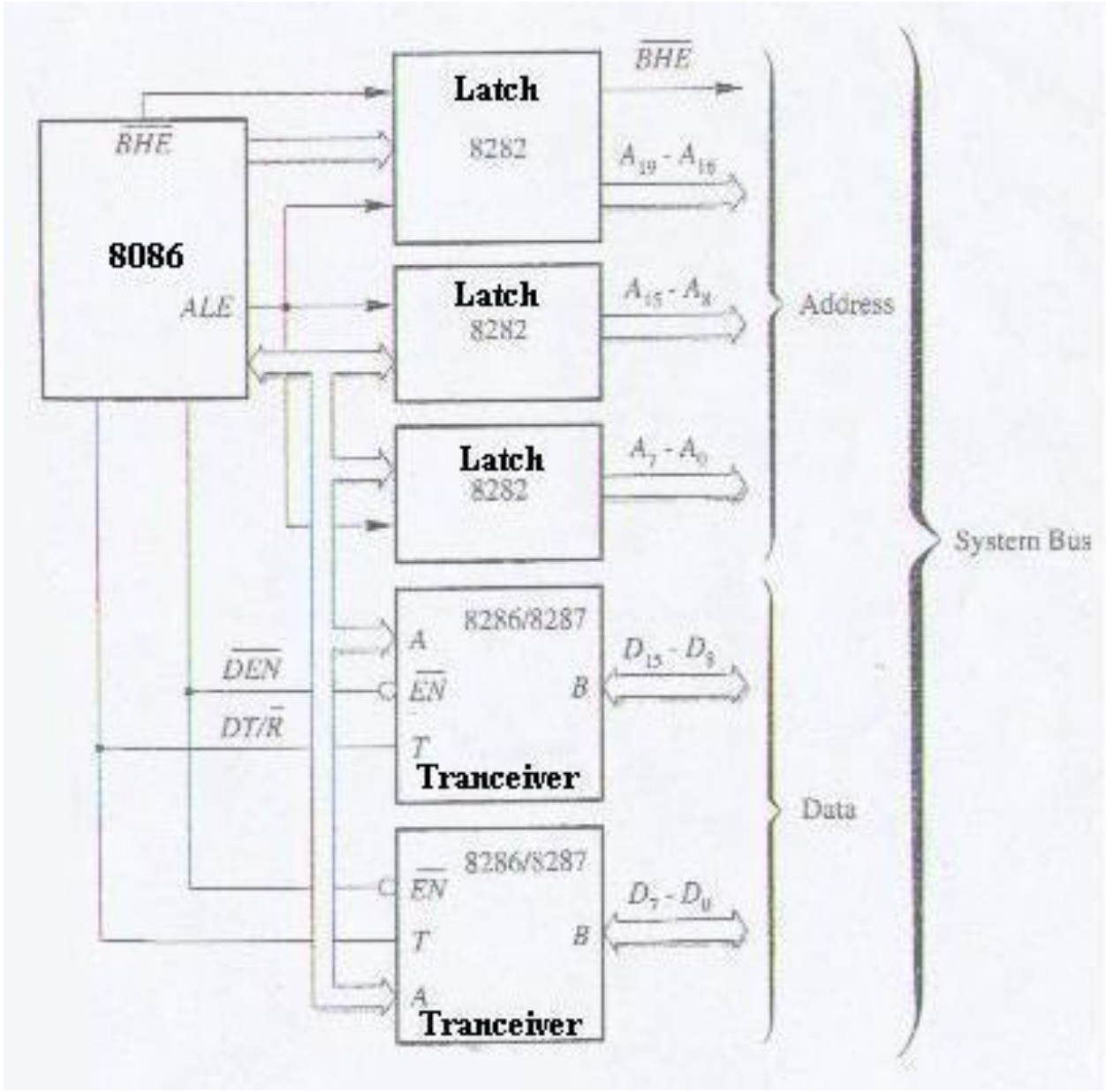


Fig. 3 – Sistema con demultiplexaggio centrale completo per dati e indirizzi

Volendo ora invece riassumere il funzionamento dei precedenti schemi e prendendo ad esempio la fig. 3 (per gli altri il funzionamento è analogo,basterà eliminare il passaggio n°3 fra quelli che seguono) :

1. Il μP genera gli indirizzi
2. ALE li memorizza nei latch sul suo fronte di discesa
3. DEN* e DT/R* abilitano i tranceiver ed indicano la direzione dei dati : da o verso il μP
4. Transitano i dati,da o verso il μP

Si inserisce per completezza il collegamento μP-memoria basato sull'8086 nel modo minimo (come si può notare,lo schema comprende nel blocco centrale, gli schemi precedenti)

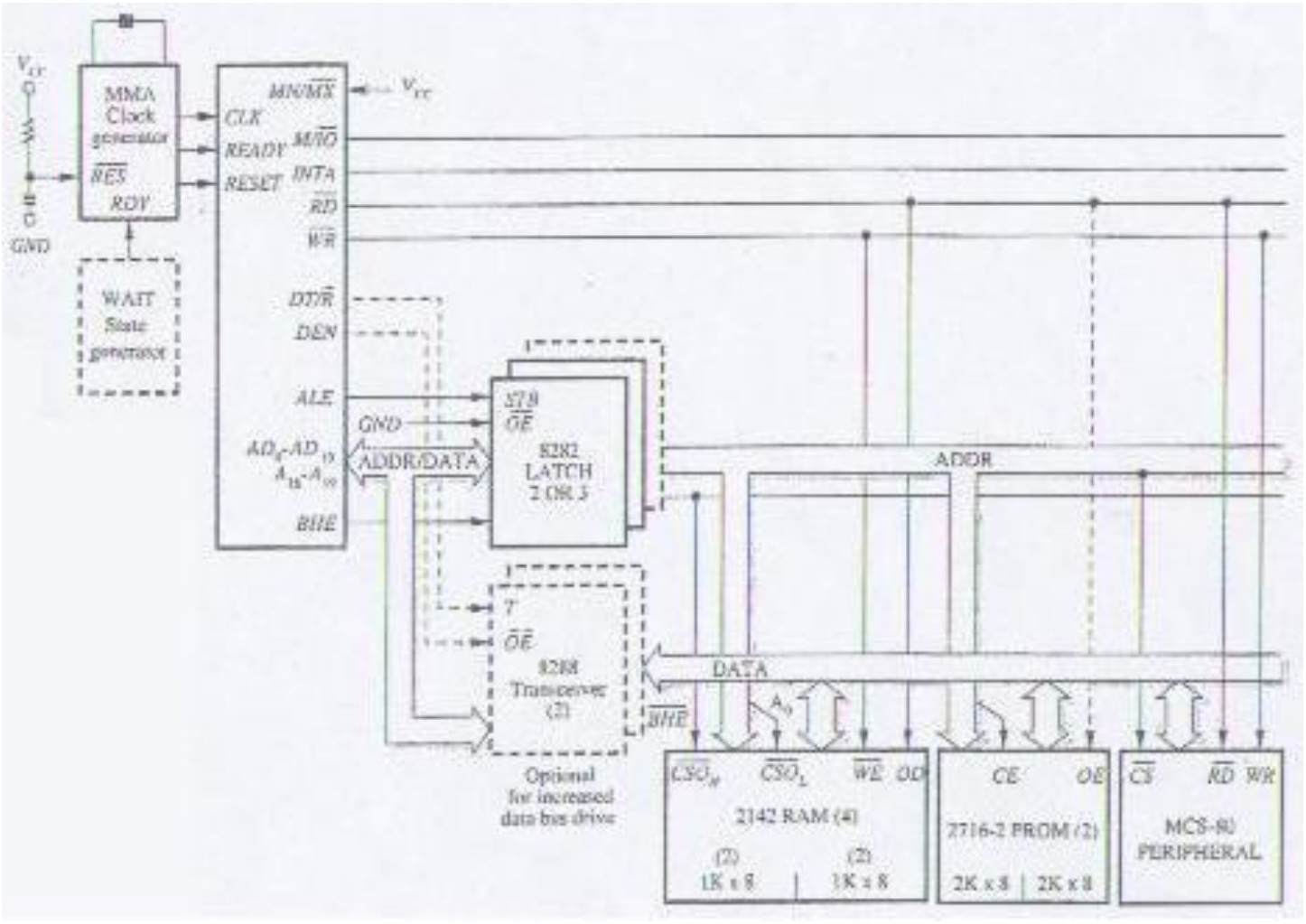
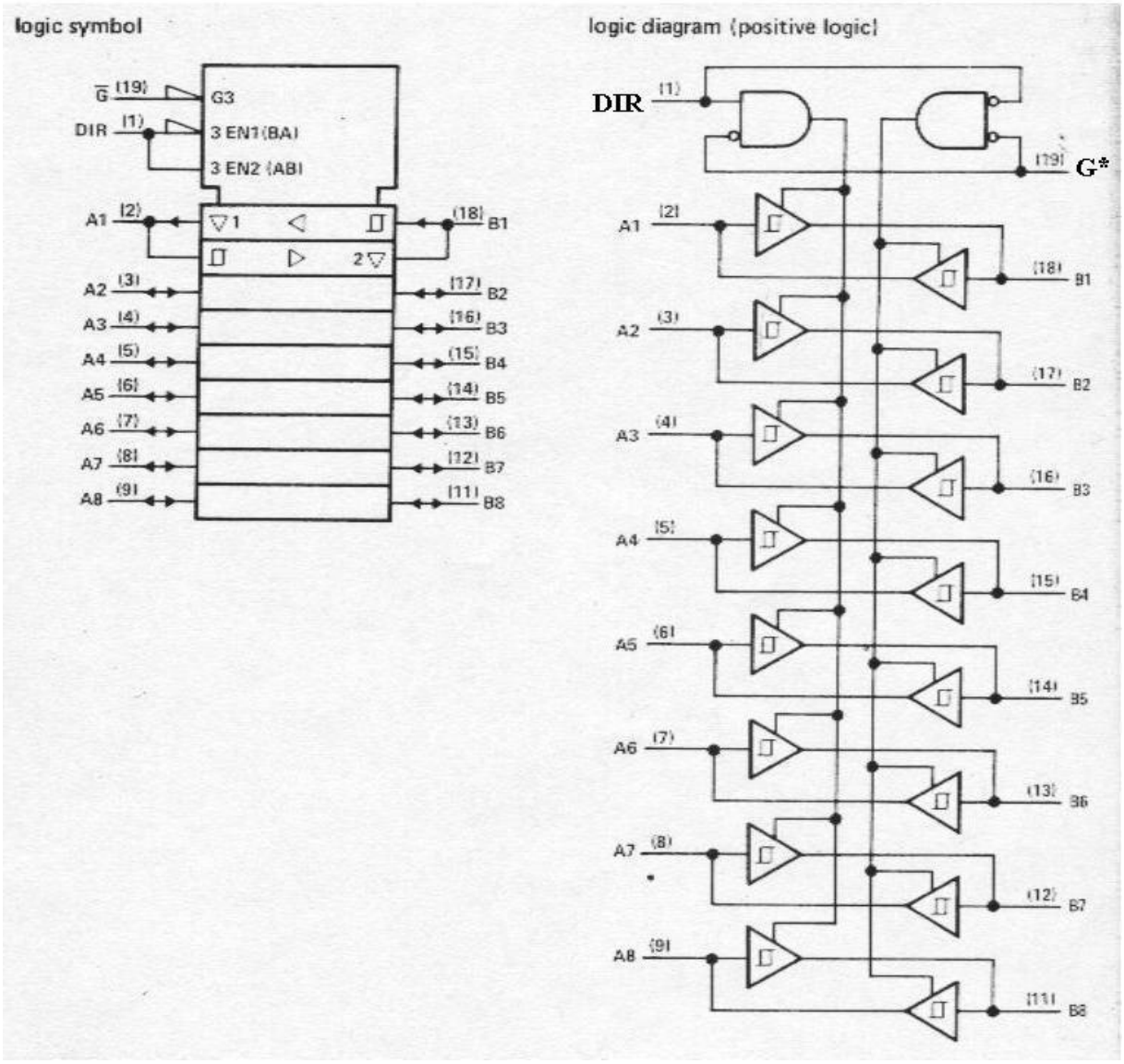


Fig. 4 – Sistema basato sull'8086 nel modo minimo

Struttura di un tranceiver



Il diagramma logico,parte destra,presenta i segnali DIR e G* collegati a due porte AND (qui DIR=DT/R* e G*=DEN*),le cui uscite sono collegate a due blocchi di buffer tristate (i triangoli che assomigliano a delle not).
Un buffer tristate è un dispositivo che può assumere tre stati (3-state) a seconda del comando ricevuto : 0,1,alta impedenza (= segnale non valido per il sistema); cioè se il comando è 0 l'uscita sarà sempre in alta impedenza,mentre se è 1, l'uscita rispecchia l'ingresso (0 se l'ingresso è 0,1 altrimenti)
Quindi immaginando il μP collegato al blocco di sinistra (A_1-A_8) e la memoria (o l'I/O) collegata a quello di destra (B_1-B_8) :

G* = DEN* = 1	che negato all'ingresso delle porte AND le manda tutte e due a 0 bloccando tutti i buffer tristate in alta impedenza,quindi non funziona nessun blocco
G* = DEN* = 0	che negato all'ingresso delle porte AND va a 1 e non influisce sulle porte stesse,quindi tutto dipende da DIR
DIR = DT/R* = 1	uscita porta AND di sinistra a 1 (blocco sinistro attivato) uscita porta AND di destra a 0 (blocco destro non attivo) flusso dati da μP(sinistra) a memoria(destra)
DIR = DT/R* = 0	uscita porta AND di destra a 1 (blocco destro attivato) uscita porta AND di sinistra a 0 (blocco sinistro non attivo) flusso dati da memoria(destra) a μP(sinistra)

Struttura di un Latch

Un latch si può costruire a partire dal circuito di base indicato nella figura e che verrà presentato nel corso di elettronica

5e. Le temporizzazioni dei cicli macchina

Il ciclo macchina tipico dell'8086 consiste di quattro stati elementari, che possono aumentare se la CPU colloquia con periferiche lente con l'inserimento di uno o più stati di attesa.
In relazione a quello che il microprocessore deve fare nel particolare ciclo cambiano i segnali interessati e i rapporti tra gli stessi, intesi come requisiti per il corretto funzionamento del sistema.
Fermo restando che una quantificazione precisa dei vari tempi di presentazione, di ritardo, ecc. è ovviamente legata al clock usato e alla tecnologia, per cui occorre esaminare di volta in volta i data sheet corrispondenti al dispositivo usato, e precisamente le caratteristiche in corrente alternata, qui di seguito saranno illustrate le problematiche generali di temporizzazione relative alla generazione e alla memorizzazione degli indirizzi fisici e ai cicli di lettura e scrittura,

Gli indirizzi

L'8086 genera durante lo stato T_1 i bit di indirizzo e il segnale ALE valido (fig. 5a): poiché è necessario avere disponibile l'indirizzo per tutto il tempo necessario all'esecuzione dell'istruzione, è stato già detto che occorre memorizzare in registri esterni alla CPU i bit relativi ed è stato suggerito l'impiego dei latch 8282 o 8283 prodotti dall'intel appunto per questo scopo.

I cicli di lettura e scrittura

La fase di indirizzamento più sopra descritta è comune a tutti i cicli macchina possibili nell'8086. Saranno esaminati innanzitutto i cicli di lettura e scrittura in memoria o in I/O, rappresentati nelle fig. 5 b) e c).

La lettura di un dato da parte dell' 8086 richiede valori specifici di alcuni segnali di controllo e precisamente deve essere valido RD*, deve andare a 0 il segnale DT/R*, mentre M/IO* deve assumere il valore 1 o 0 secondo che la lettura avvenga da una memoria o da un dispositivo di I/O. I segnali M/IO* e DT/R* vanno al valore richiesto precocemente durante T_1, per cui non intervengono in questa discussione.
Dopo la memorizzazione dell'indirizzo nei registri appositi, il bus locale AD
nei cicli di lettura va in alta impedenza per un certo intervallo, durante il quale diventa valido il segnale RD*, la cui durata determina la fase di lettura effettiva per i dispositivi collegati direttamente sul bus del microprocessore, mentre il segnale DEN*, valido all'interno di RD*, è usato quando i dispositivi da leggere vengono collegati al bus dell'8086 tramite tranceiver.
Considerando memorie direttamente collegate al bus del microprocessore, occorre che la decodifica della memoria preceda o sia almeno contemporanea alla disponibilità dei dati in uscita dalla memoria.
Il segnale RD*, come detto più sopra, diventa valido con un certo ritardo rispetto all'intervallo di tempo in cui è disponibile sul bus AD l'indirizzo e resta valido fino a dopo il fronte di discesa all'inizio di T_4, quando il microprocessore campiona il contenuto del bus e lo acquisisce come dato valido. Successivamente il dato resta valido per un certo intervallo di tempo all'interno di T_4 e quindi il bus passa in alta impedenza prima dello stato T_1 del ciclo macchina successivo: ciò al fine di permettere una chiara individuazione, istante per istante, del contenuto, dati o indirizzi, del bus AD.

Il ciclo di scrittura in memoria è identico a quello di lettura per la parte relativa alla generazione degli indirizzi: dopo gli indirizzi il microprocessore presenta immediatamente sul bus AD i dati da scrivere, senza che il bus stesso vada, per un certo tempo, in alta impedenza, come avviene nel ciclo di lettura. Il segnale WR* diventa valido durante T_2 prima di quando diventava valido RD* e resta valido fino a un istante precedente il,fronte di salita di T_4 : i dati restano validi oltre l'inizio di T_4 e la scrittura in memoria è comandata dal fronte di salita di WR*.
Ovviamente, se necessario, come nel caso della lettura, vengono inseriti tanti stati di attesa, incrementando i valori limite fino a rispettare tutte le relazioni suddette.
In sistemi di una certa complessità può non essere conveniente collegare le memorie direttamente al bus locale: in questo caso, come detto nel precedente paragrafo, vengono usati i tranceiver, che fungono da buffer verso l'esterno per i dati. I sistemi basati sull'8086 nel modo minimo di funzionamento utilizzano, per la gestione dei tranceiver, tipicamente gli Intel 8286 e 8287, i

segnali DEN* e DT/R*, che rispettivamente abilitano all'istante opportuno il trasferimento dei dati tra bus di sistema e bus locale e specificano la direzione del trasferimento, il segnale DT/R* normalmente alto, cioè predisposto per la scrittura: diventa basso all'inizio di ogni ciclo (T_1) che invece prevede una lettura dalla memoria.
Il segnale DEN* sostituisce, praticamente, il segnale RD* nei cicli di lettura mentre gestisce, insieme a WR*, la scrittura in memoria. Nella lettura occorre tenere ben separate la fase di memorizzazione degli indirizzi nei latch dalla fase di lettura effettiva, con i dati contenuti in memoria che sono trasferiti sul bus AD: tenuto conto della generazione precoce del segnale DT/R*, l'8086 garantisce questo obiettivo rendendo valido DEN* all'interno di RD, che già garantiva comunque questa separazione indirizzi-dati.
Nei cicli di scrittura invece, in presenza di un segnale DT/R* che garantisce una direzione di trasferimento dal microprocessore alla memoria sin dall'inizio dello stato T_1, non sorge alcun problema se DEN* diventa valido anche quando non è stato ancora memorizzato l'indirizzo, in quanto comunque non viene modificato il contenuto del bus AD in corrispondenza del fronte attivo del segnale ALE e i dati da scrivere restano quelli presenti sul bus AD in corrispondenza del fronte attivo del segnale WR*.

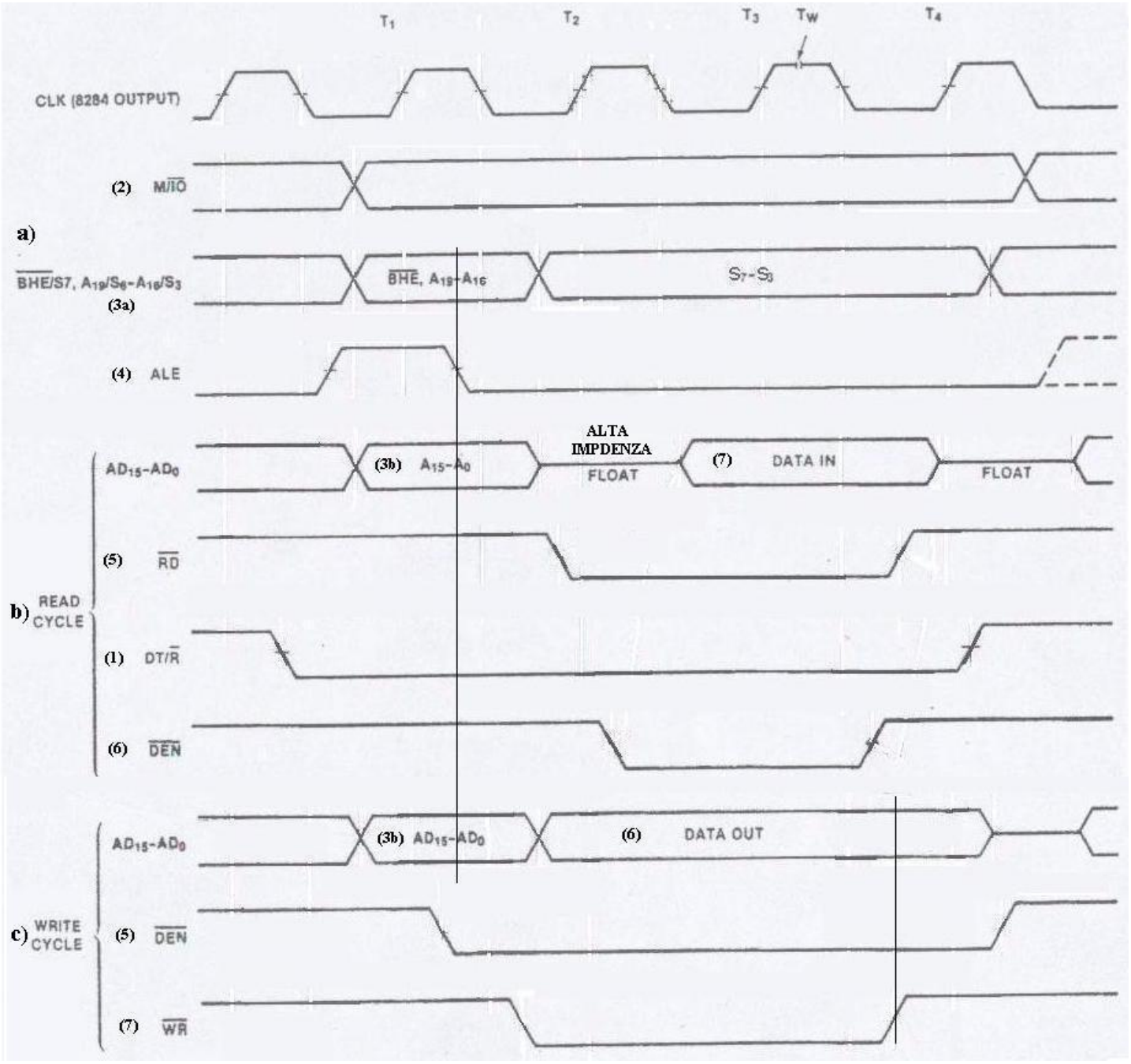


Fig. 5 – Temporizzazioni dei cicli macchina : a)generazione degli indirizzi, b)ciclo di lettura e c)di scrittura

Riassumendo , come potete notare dai numeri fra parentesi indicati a fianco di ogni segnale…

a) generazione degli indirizzi (uguale sia per il ciclo di lettura che di scrittura)

1. DT/R* va a 0 nel caso di una lettura (Ricezione)
   resta a 1 nel caso di una scrittura (Trasmissione,si infatti questo segnale è predisposto per la scrittura quindi è normalmente ad 1 ed è per questo che nel ciclo di scrittura non è indicato)
2. M/IO* va 1 se la lettura/scrittura è in memoria
   va a 0 se la lettura/scrittura è in I/O

3a) e 3b) vengono generati i bit di indirizzo A_{19}-A_{16} e A_{15}-A_0

4. ALE memorizza i bit di indirizzo nei latch sul fronte di discesa (vedere linea verticale grande)

b) ciclo di lettura

5. RD* va a 0 ed indica la lettura.
   Questa transizione avviene pochi istanti dopo che il bus multiplexato dati/indirizzi va in alta impedenza (=segnale non valido per il sistema).Il periodo di alta impedenza permette alla circuiteria di distinguere bene quando passano gli indirizzi e quando i dati sul bus,in questo modo non vi sono rischi di conflitto (scontro) fra i due tipi di segnali : si ricorda che in questo caso gli indirizzi vengono generati dal μP,mentre i dati arrivano al μP dalla memoria(o dall'I/O).
6. DEN* va a 0 ed indica ai tranceiver che possono lasciare transitare i dati in arrivo dalla memoria (o dall'I/O) verso il μP : naturalmente questi dati viaggiano sottoforma di segnali elettrici e paragonando la corrente elettrica ad una corrente d'acqua,si può immaginare DEN* come un rubinetto che si apre e lascia passare questa corrente
7. Arrivano i dati al μP

c) ciclo di scrittura

5. DEN* va a 0 ed indica ai tranceiver che possono lasciare transitare i dati in uscita dal μP verso la memoria (o l'I/O) : se notate bene però,quando ciò avviene sul bus ci sono ancora gli indirizzi,ma non ci sono problemi, perché tanto la scrittura dei dati in memoria (o in I/O) avviene sul fronte di salita di WR* (linea verticale piccola),quando si è sicuri che ci sono invece i dati.
6. Il μP emette i dati
7. WR* sul fronte di salita (linea verticale piccola) memorizza i dati

5f. Il microprocessore INTEL 8088

Il microprocessore 8088 è la versione a 8 bit dell'8086, assolutamente identico come struttura interna, mentre si differenzia, all'esterno, per il bus locale, multiplessato solo relativamente al byte meno significativo degli indirizzi con l'unico byte di dati. L' 8088 è completamente compatibile dal punto di vista del software con l'8086, potendo operare su byte, parole (word) a 16 bit e blocchi, con la differenza che anche gli accessi in memoria relativi a parole di 16 bit in indirizzi pari avvengono in due cicli di bus.
I segnali esterni corrispondono esattamente a quelli dell'8086, con le sole differenze che:
— i piedini 2-8 e 39 ora sono relativi solo al byte più significativo degli indirizzi, che resta disponibile per tutto il ciclo di bus e quindi non deve essere memorizzato in un registro, come continua a essere necessario per il byte meno significativo;
— il piedino 34 non ha più la funzione di abilitare la parte alta del bus dei dati,ma conserva solo una funzione di linea di stato nel modo minimo per individuare il ciclo di bus in esecuzione insieme a DT/R* e M/IO*.

Complessivamente si può dire che l'8088 è un microprocessore a 8 bit che, rispetto agli altri microprocessori a 8 bit, sfrutta un'architettura interna a 16 bit, quindi molto più efficace, un set di istruzioni molto più esteso e potente e ha una capacità di indirizzamento molto più ampia. Può essere collegato facilmente alle normali porte seriali e parallele e, infine, ha una completa compatibilità software con l'8086 e i successivi microprocessori a 16 bit della famiglia Intel.

Utilizzato nel primo personal computer IBM, nella logica di continuità e compatibilità più sopra ricordata, è stato uno dei fattori determinanti del successo dei PC.

Bibliografia di riferimento

- Sistemi 1, Antonio Garavaglia - Franco Petracchi, ed. Zanichelli
- Sistemi a Microprocessore, A.Salsano – A. Ferreri – F.Tota Gramegna, ed. Petrini Editore