1 Computer

In linea di principio un computer è "solo" un enorme automa a stati finiti, con un numero inimmaginabile di possibili stati[1]. A differenza di un automa normale, però, un computer è "programmabile".
Mentre un automa realizzato con le normali porte logiche è da rifare daccapo ogni volta che gli si vuole far fare cose diverse, un computer, basato su un "programma" può cambiare le sue funzioni cambiando solamente il suo programma.
Cambiando programmi si ha di fronte una possibilità di fatto infinita di modificazioni, senza cambiare nulla nella struttura fisica del computer. Questo significa che lo stesso pezzo di silicio può servire per eseguire diversi compiti.

Ogni computer è un sistema discreto e lavora solo con numeri. I suoi ingressi sono costituiti da numeri così come le sue uscite.
Sui numeri che riceve in ingresso il computer è in grado di fare calcoli ed altre "elaborazioni", alcune delle quali anche molto complicate. Sui numeri che sta elaborando esso è in grado anche di fare dei confronti e di "prendere delle decisioni", secondo i risultati dei confronti.
Alcuni numeri "speciali" assumono, per il computer, il significato di "comandi". In questo caso quando il computer riceve un certo numero, esegue un'elaborazione, se riceve un numero diverso esegue un'elaborazione diversa.
Questi "comandi" vengono detti "**istruzioni**".

Un **programma** è un insieme di istruzioni che risolve un singolo problema.

Hardware e software

Molti sapranno già la differenza fra hardware e software, ma, come si dice, "repetita juvant" (giova ripetere (i concetti)).
Hardware ("roba dura", ferraglia) è quella parte di un computer che si può toccare con mano; **software** ("roba morbida") è la componente immateriale, che non si tocca ma è indispensabile per il suo funzionamento, cioè tutti i programmi che, insieme, ci permettono di usare il computer e di fare quello che ci serve.
Di solito la distinzione è estremamente chiara, gli unici dubbi possono venire quando si parla degli strumenti tramite i quali il software può essere "spostato" o memorizzato. In questo caso bisogna distinguere fra il contenuto, il programma, che è software, ed il contenitore (si dice il "supporto"), che è hardware. Per esempio, in un CDROM contenente un gioco elettronico, il disco "è hardware" mentre i numeri che ci sono scritti sopra sono software.
L'insieme di tutti i programmi scritti per un computer è il suo software.
Molti abbreviano la parola Hardware con Hw e Software con Sw.

Dati

I numeri che un computer deve elaborare vengono chiamati "**dati**" (in Inglese: "datum" al singolare, "data" al plurale).
In un senso molto generale il computer è dunque un "elaboratore di dati", cioè un sistema che riceve in ingresso dati, cioè che vengono anche detti Input o Ingressi , li elabora secondo funzioni complesse determinate dai suoi programmi, e mette in uscita altri numeri, che comunicano i **risultati** dell'elaborazione, che vengono anche detti Output o Uscite.

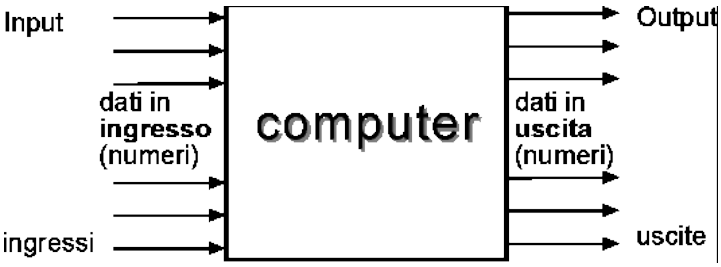

Figura 1: il computer come elaboratore di dati

1.1 L'architettura di Von Neumann

La fondazione teorica dell'informatica è stata posta da Johann Von Neumann (John). Neumann, ungherese, nato a Budapest nel 1903, aveva il titolo nobiliare di "Margattai", che egli stesso cambiò nel tedesco "Von" quando andò a lavorare in Germania. Nel 1928 pose le basi della teoria matematica dei giochi. Lasciò la Germania per gli Stati Uniti con l'avvento del nazismo e lavorò ai progetti militari che, durante la seconda guerra mondiale, portarono allo sviluppo dei primi calcolatori elettronici.
I primi computer sviluppati non erano molto diversi dai computer odierni, che ne condividono tutti i "fondamenti", anche se, come vedremo, molti computer avanzati si discostano da numerosi dei concetti che esporremo inizialmente.
L'architettura di un computer è suddivisa in primo luogo in due grandi sezioni: la CPU e la Memoria (vedi Figura 2).
La sigla CPU è l'abbreviazione di "**C**entral **P**rocessing **U**nit" ed è il cuore del computer. Nella CPU è concentrata tutta la capacità di elaborazione di un computer, come peraltro spiega anche il suo nome, dato che il termine "processing" si tra-

1 Vedi la Parte 1 di questo testo, capitolo "Sistemi discreti"

duce bene in italiano con "elaborazione" o anche "calcolo". La CPU concentra in sé anche le capacità di "controllo" di tutte le altre unità presenti nel computer, che sono da essa coordinate e comandate.
La memoria è un serbatoio di informazioni. I numeri dei quali la CPU ha bisogno per eseguire le sue elaborazioni si trovano nella memoria.
Mentre la CPU è il soggetto "attivo" del computer, la memoria è invece il soggetto "passivo". Se la CPU "fa", la memoria "subisce".
Naturalmente ci deve essere un modo per far passare numeri dalla CPU alla memoria e viceversa. Per effettuare questi trasferimenti c'è un terzo componente, che per ora abbiamo indicato nel disegno solo come "mezzo di trasmissione". La direzione delle frecce nel disegno indica che l'informazione può fluire in entrambi i versi.

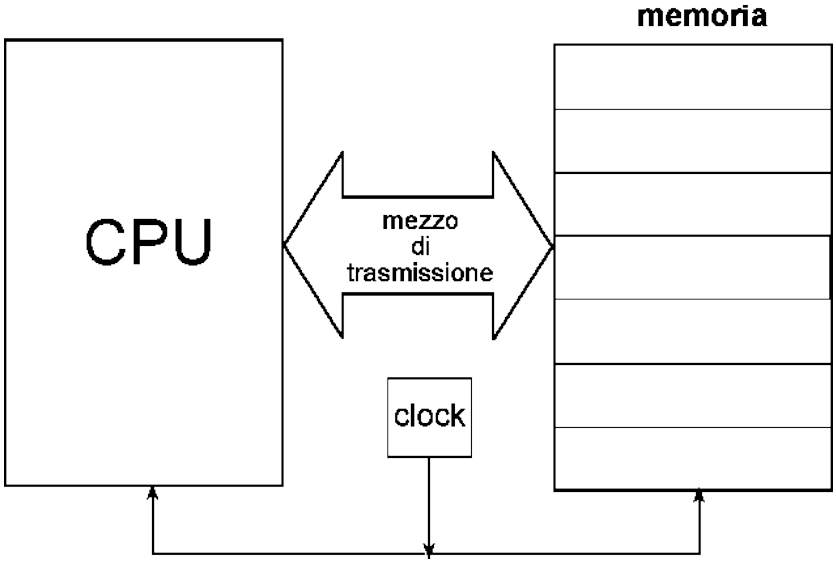

Figura 2: componenti principali di un computer

Dispositivi (device)
Un termine generico, molto usato nell'informatica e nell'elettronica è "dispositivo".
Indichiamo come "**dispositivo**" (**device**) un qualunque sottosistema hardware individuabile in un computer, che ha una funzione singola ben definita.
Vengono chiamati "device" la tastiera, il monitor, i singoli circuiti integrati che compongono un computer, la CPU stessa, vista come componente elettronico ed un'infinità di altri sistemi.

Clock
La tecnologia con cui sono realizzati i computer è quella delle reti logiche sincrone, per cui tutti i dispositivi che sono compresi all'interno di un computer cambiano stato contemporaneamente. Il momento in cui deve avvenire il cambio di stato è stabilito da un segnale che viene distribuito ad ogni dispositivo. Si tratta di un segnale ad onda quadra detto "**clock**" di sistema (system clock), che, da bravo orologio, sincronizza insieme tutti i dispositivi, dettando gli instanti in cui si considera che essi cambiano stato.

Reset
Un computer è un sistema digitale sequenziale complesso e come tutti i sistemi sequenziali è necessario che all'inizio del suo funzionamento esso si ponga in uno stato ben preciso.
Tutti i componenti di un computer, CPU compresa, dispongono di un piedino di "**reset**". Quando il segnale su quel piedino diventa alto il dispositivo si mette nella sua condizione iniziale. All'accensione del computer, o quando l'utente preme il tasto "reset" sul pannello frontale, un segnale di reset verrà propagato a tutti i dispositivi presenti nel computer, che così potrà partire da una condizione nota.

1.1.1 Memoria di lavoro

In un computer esistono diversi tipi di memorie, come avremo modo di vedere in dettaglio. Quella a cui facciamo riferimento in questo contesto è la cosiddetta "**memoria di lavoro**", "memoria **centrale**" o "memoria **principale**", l'unica che è a contatto diretto con la CPU, ossia che può avere con essa una comunicazione senza intermediari.
La **memoria** di lavoro è un insieme di dispositivi elettronici nei quali possono essere mantenuti indefinitamente dei numeri binari.
La CPU può stabilire, secondo i comandi che riceve, se leggere o scrivere nella memoria.
Un'operazione di lettura prenderà dei bit dalla memoria e li trasferirà nella CPU, mentre una scrittura trasferirà un numero dalla CPU alla memoria. Per non sbagliarsi nello stabilire se un trasferimento è di lettura o scrittura basta "mettersi nei panni" della CPU, alla quale tutta la nomenclatura del computer è riferita. Se mi immagino di essere la CPU e voglio ricevere qualcosa dalla memoria, lo "leggo", se voglio modificare la memoria, devo "scrivere".

Una scrittura cancella il contenuto precedente della memoria sulla quale interviene e lo sostituisce con quanto proveniente dalla CPU. Una lettura non modifica in alcun modo la memoria.

Indirizzo

La memoria è divisa in "locazioni" in ciascuna delle quali si può tenere un certo numero di bit.
Una **locazione** è la porzione minima di memoria cui la CPU può avere accesso con una singola operazione.
In tutti i casi significativi la dimensione delle locazioni di memoria è di 8 bit (un Byte)[2]. Dunque la quantità di memoria si misura in Byte.

Le CPU moderne possono avere molte locazioni di memoria, milioni o miliardi. Dobbiamo trovare il modo di permettere alla CPU, che ha il controllo della memoria, di stabilire in ogni istante quale specifica locazione della memoria deve essere utilizzata. Siccome la CPU può trattare solo numeri, dovrà distinguere fra le varie locazioni con un numero, che viene detto "indirizzo".

L'**indirizzo** è il numero che permette alla CPU di indicare su quale locazione di memoria vuole lavorare.

Tramite l'indirizzo le locazioni sono numerate consecutivamente, a partire dalla numero 0.
Ogni locazione di memoria deve avere uno ed un solo indirizzo.
In ogni computer l'hardware deve essere realizzato in modo tale che ogni locazione di memoria, dentro uno qualsiasi dei circuiti integrati (chip) che la compongono, sia raggiungibile usando il suo indirizzo e solo quello.
Ogni CPU ha locazioni di una certa dimensione. Per esempio nell'8086 e nel PowerPC le locazioni sono di 8 bit. Ciò significa che la memoria in questi due tipi di CPU è indirizzabile a Byte. Di conseguenza la memoria di queste CPU si misura in Byte o nei suoi multipli (kByte, MByte, ..).
Un modello che può aiutare per comprendere cos'è la memoria di lavoro è quello di una cassettiera. Se immaginiamo la memoria come una grande cassettiera, una locazione è un cassetto, l'indirizzo è il numero del cassetto, mentre il Byte di memoria che può essere letto o scritto è il contenuto del cassetto.
Chiamiamo "dato o istruzione" questo numero. Come implica la denominazione il contenuto di una locazione di memoria può avere per una CPU i due significati di comando (istruzione) o di numero da elaborare (dato).

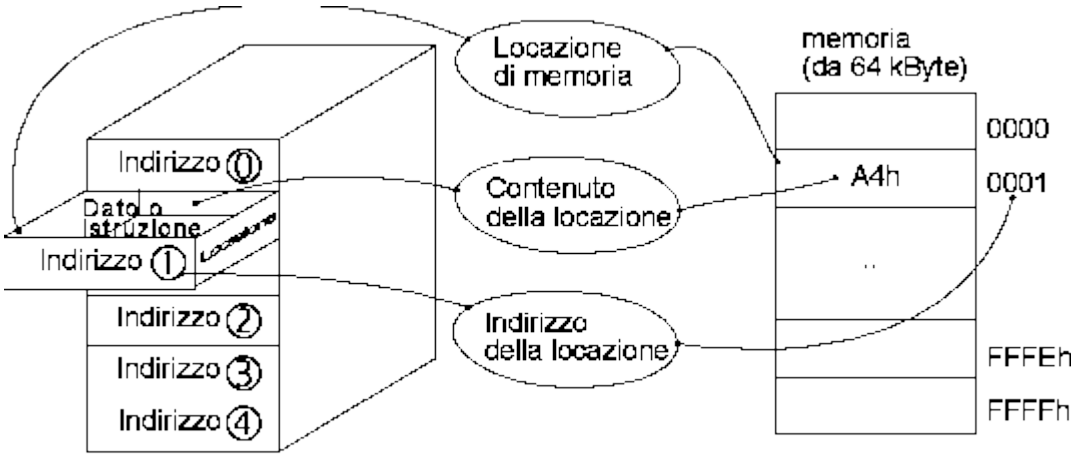

Figura 3: memorie e cassetti[3]

Input/Output

Un computer deve svolgere elaborazioni ma deve anche comunicarne i risultati al resto del mondo. Necessita dunque di dispositivi di ingresso (Input) che prendano dall'esterno del computer i dati da elaborare e di dispositivi di uscita (Output) che restituiscano all'esterno i risultati.
Qualunque forma abbiano gli ingressi che vogliamo far elaborare ad un computer, essi devono essere in qualche modo trasformati in numeri. Analogamente, anche se otteniamo risultati da un computer sotto forme diverse (es. stampa su carta, visualizzazione su monitor ..), il "cuore" del computer ha prodotto quei risultati come numeri, che sono stati successivamente trasformati.
Oltre alla memoria ed alla CPU dobbiamo perciò individuare, all'interno del computer, anche una o più unità di Input / Output (I/O, si legga "ai-ô", all'Inglese, perché tutti fanno così). Le unità I/O sono dispositivi numerici, in grado di co-

2 Esistono alcuni microcontrollori (esempio 8051) che permettono di accedere direttamente a singoli bit dei loro memoria interna (memoria "bit addressable"). Dunque la definizione di "locazione" appena enunciata implicherebbe una dimensione di un bit per le locazioni. Però, dato che le zone di memoria "bit addressable" sono ben determinate e molto piccole, anche in questi microcontrollori si considera che le locazioni siano di 8 bit.
3 In questa ed in tutte le altre figure di questo testo la memoria è indicata con gli indirizzi minori in alto. Questa è una scelta personale dell'Autore, perché gli risulta più naturale; peraltro in altri testi e nella documentazione di alcune case costruttrici si trovano disegni organizzati al contrario.

municare con la CPU, che convertono le informazioni del mondo esterno e si presentano alla CPU come se fossero memorie.
Come indicato nella Figura 4, la CPU non vede direttamente la tastiera o il monitor, ma un dispositivo elettronico, un circuito integrato di I/O, fatto apposta per poter comunicare con la CPU. Il circuito di I/O si comporta, dal punto di vista elettrico, esattamente come una memoria, si "traveste" in qualche modo da memoria. Per questo la comunicazione con i dispositivi è, dal punto di vista della CPU, assolutamente analoga a quella con la memoria.

L'alternativa a questo approccio è che il dispositivo possa comunicare direttamente con la memoria, come fa l'hard disk a destra nella figura, ma questo è un caso particolare che vedremo nel più avanti e che per un bel po' potremo dimenticare.

Nella figura c'è anche disegnato un confine tratteggiato. Quel confine corrisponde grossomodo alla scatola del computer, che racchiude le sue unità più importanti. La scatola del computer viene detta "**Unità Centrale**", e quello che c'è "fuori" come la tastiera o il monitor vengono detti "**periferiche**".

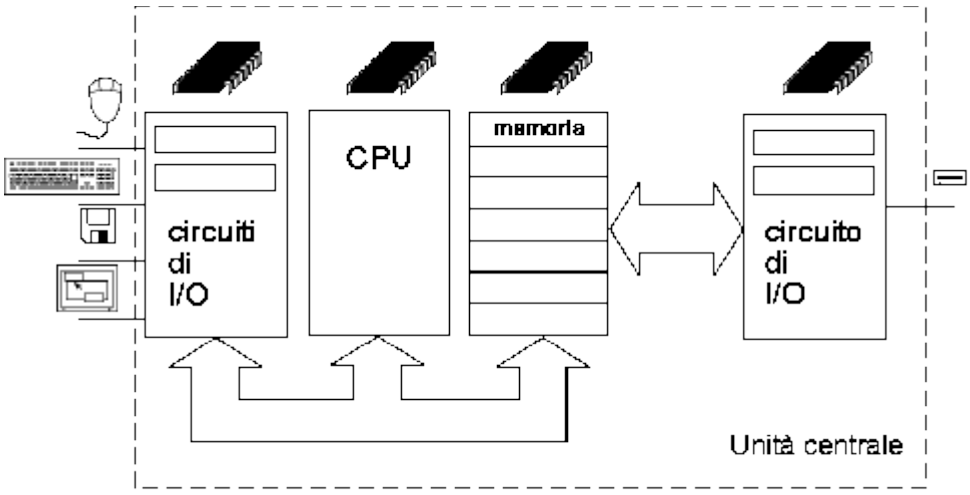

Figura 4: comunicazioni con l'esterno

Interfacce

In gergo tecnico – scientifico ciò che sta al confine fra due sistemi e consente loro di scambiarsi informazioni, energia o materia viene definito "**interfaccia**".
Quando si parla dell'architettura di un computer un'interfaccia è l'insieme dei dispositivi che permettono ad un computer di comunicare con l'esterno.
Una caratteristica importante di ciò che viene chiamato interfaccia è che le modalità dello scambio di informazioni attraverso di essa sono sempre ben definite e spesso codificate in vere e proprie normative, ratificate da organismi internazionali di normalizzazione. Esempi: interfaccia seriale RS-232 (norma emessa da EIA e da ITU), interfaccia parallela Centronics ("standard di fatto" della ditta Centronics, una interfaccia di grande diffusione "copiata" da tutti gli altri).
Si noti che il termine ha preso piede anche nel mondo software, ed indica l'insieme delle convenzioni usate per far comunicare fra loro due "pezzi di software".

1.1.2 Bus

Vediamo ora come è fatto quel "mezzo di trasmissione" che permette ai numeri di viaggiare attraverso al computer. Il mezzo di trasmissione è composto da semplici linee di materiale conduttore, fili, ciascuno dei quali trasmette un bit d'informazione. Il bit viene rappresentato con il valore di una tensione elettrica, diversa secondo il modello di CPU utilizzato. Nella stragrande maggioranza dei casi si considera che il bit sia "0" quando la tensione è vicina allo zero, mentre si considera che la linea del bus assuma il valore "1" quando la tensione è vicina alla tensione di alimentazione del sistema.
L'insieme di queste linee viene detto "bus". In ogni computer vi sono diversi tipi di bus.
Un **bus** è perciò un fascio di fili paralleli che distribuisce le informazioni a tutti i dispositivi hardware che sono contenuti in un computer[4].
Dato che la CPU è il centro di controllo del computer, tutti i principali bus passano per la CPU, ogni filo di questi bus è collegato ad un piedino del microprocessore.

Data Bus

In ogni computer sarà indispensabile avere un bus per trasportare quelle informazioni che devono fluire da e per la memoria di lavoro. Questo bus viene detto "**data bus**" (bus dei dati).

4 "bus" deriva dalla parola latina "omnibus", che significa "a tutti"

Address Bus
Oltre al data bus è comodo avere un altro bus, che in ogni istante tenga traccia di qual è la locazione di memoria il cui contenuto viene trasferito sul data bus. Questo bus contiene solo indirizzi e di conseguenza viene detto "**address bus**". Dato che la CPU ha il controllo di tutto ciò che accade in un computer, decide quale deve essere la locazione indirizzata in ogni istante. Per questa ragione l'address bus può essere scritto solo dalla CPU. Ecco perché nella Figura 5 la freccia va solo in una direzione. Nella stessa figura le frecce grosse rappresentano fasci di fili, quelle piccole fili singoli.
Control "Bus"
Abbiamo già detto che la CPU ha la responsabilità di comandare il funzionamento di ciascuno dei dispositivi presenti nel computer. L'unico mezzo che ha per adempiere a questo compito è di disporre di alcuni fili in cui immettere informazioni binarie di controllo.
Come esempio di uno di questi fili si può prendere quello collegato al piedino R/W **W soprassegnato** (**R**ead o **W**rite negato della CPU. Con questo piedino la CPU può indicare alla memoria se l'operazione in corso è una lettura od una scrittura.
Se R/W **W soprassegnato** è al livello alto il trasferimento deve essere in lettura (Read), in scrittura (Write) se è basso. Ogni CPU ha uno o più piedini dedicati a questo scopo, anche se non è detto che il nome ed il funzionamento nel dettaglio siano esattamente come descritto.
Altre linee di controllo di una CPU, delle quali tratteremo molto più avanti, sono: interrupt, DMA, handshake .. .
Molti chiamano l'insieme di questi fili "control bus"; peraltro questo nome non è del tutto corretto, dato che l'insieme delle linee ci controllo non è un vero bus come gli altri.
Infatti in ogni filo di un "vero" bus passano bit che fanno parte della stessa informazione, mentre in questo caso le linee di controllo hanno ciascuna un significato diverso da tutte le altre, portano informazioni "scorrelate". Inoltre alcune linee del control "bus" potrebbero essere del tutto inutili per certi dispositivi, come evidenziato anche nella Figura 5, ove il circuito n.2 non usa alcuna linea di controllo. Per questo nel prosieguo di questo testo si preferirà parlare di "linee di controllo" piuttosto che di control "bus".

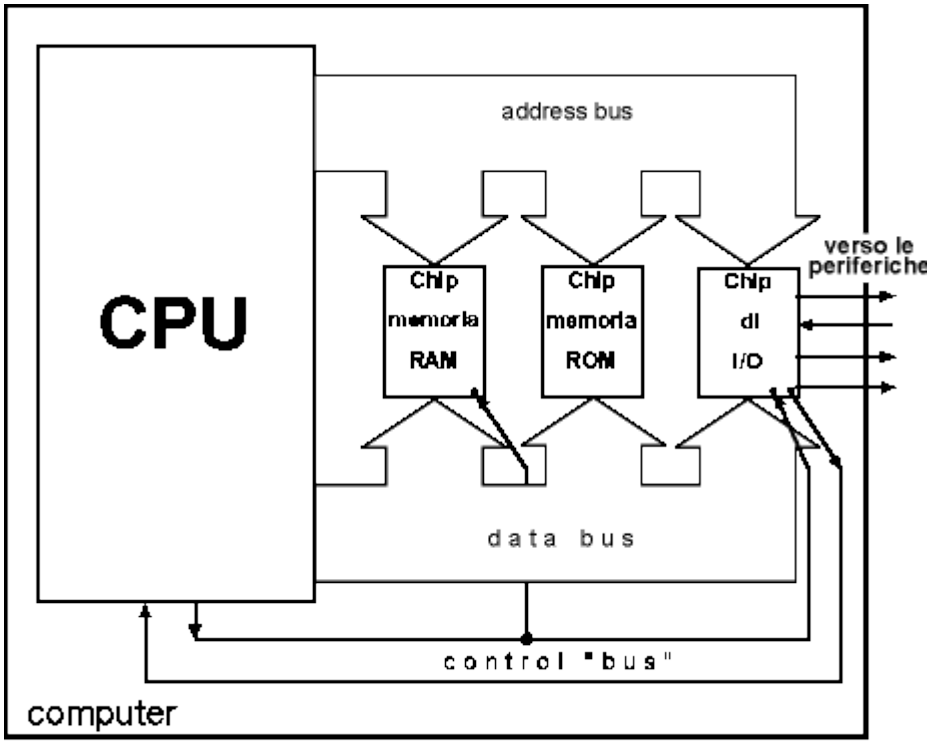

Figura 5: i bus in un computer

La velocità con cui un computer ha accesso alle informazioni dipende anche dalle caratteristiche dei bus.
Nella Figura 6 è disegnato il segnale che percorre una linea di un bus, nel periodo di tempo in cui essa deve cambiare di valore da zero a uno.
Disegnato con tratteggio è il segnale ideale, quello che sarebbe il sogno del progettista elettronico. Si tratta di uno "scatto" istantaneo e "pulito" da un valore all'altro.
Naturalmente non è questo ciò che accade nella vita reale, ove è facile incontrare segnali come quello disegnato a tratto pieno. Il sistema costituito dalla linea di bus e dai circuiti elettronici che la guidano comincia, al tempo zero, un transitorio, durante il quale il segnale subisce variazioni repentine e di grande entità. Poi, dopo un certo periodo che dipende dalla costante di tempo del sistema, il segnale si assesta e rimane vicino al valore ideale.

Il tempo di assestamento (t_{set}) è il tempo dopo il quale si è ragionevolmente certi che il segnale non uscirà più da una certa banda intorno al valore ideale di regime (indicata con due segmenti a tratteggio diverso).
t_{set} dipende da molti parametri, fra i quali il più importante è la lunghezza della linea di bus. Più è lunga la linea, più è lungo il suo tempo di assestamento. Questa è la ragione per cui l'elettronica più veloce di un computer deve essere concentrata in posizioni vicine alla CPU, idealmente al suo interno. Per questo ed altri fattori la memoria, che dista dalla CPU una decina di cm, è molto più lenta nell'acceso che non i registri, che sono all'interno della CPU e non distano fra di loro più di 2 mm.

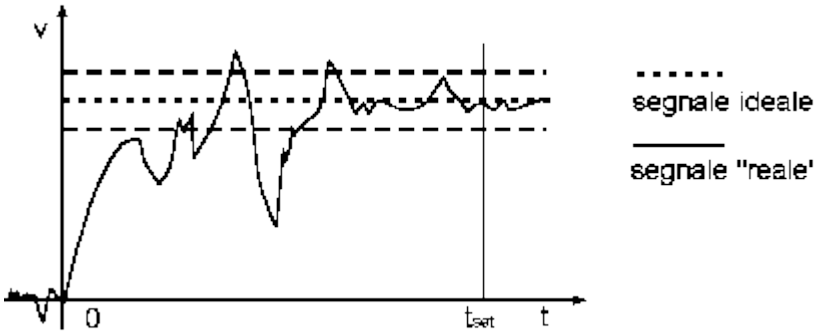

Figura 6: grafico di un segnale ideale e di un segnale reale (realistico) su un bus.

1.1.3 Istruzioni e dati

Come abbiamo già detto il computer, o meglio la sua CPU, deve essere in grado di ricevere ed interpretare dei numeri "speciali" che significano particolari comandi da eseguire. Questi comandi vengono detti "istruzioni di macchina" o "istruzioni".

Un'**istruzione di macchina** è uno dei comandi elementari che una CPU può accettare.

L'istruzione è un comando "atomico" per una CPU: non esistono comandi più "piccoli" di un'istruzione di macchina.
Ogni tipo di CPU ha le sue istruzioni di macchina, diverse da quelle degli altri tipi.
Una CPU può "capire" solo le istruzioni di macchina che le vengono fornite sotto forma di numeri. Data la precedente definizione di codice, si può dire che i numeri che rappresentano le istruzioni sono i "codici" di quelle istruzioni. Per questi numeri si parla perciò di "codice operativo".

Un **codice operativo** (**op code**) è un numero che rappresenta per una CPU un'istruzione da eseguire.

Così come un codice ASCII è un numero che rappresenta una lettera, un codice operativo è un numero che rappresenta un'istruzione di macchina ed indica alla CPU quale deve eseguire.
Un'istruzione è uno dei piccoli "mattoni" che i programmatori usano per costruire tutti i programmi.

Esempio: L'istruzione NOP appartiene al set delle istruzioni della CPU 8086. La funzione dell'istruzione NOP è di non far niente (**N**ot **Op**erate). Il suo codice operativo è 144. Se la CPU riceve il numero 144 come codice della prossima istruzione da eseguire, esegue un'istruzione NOP (cioè non fa niente).

Formato dell'istruzione di macchina

Alcune CPU hanno istruzioni di diverse dimensioni, altre di dimensioni fisse. Per esempio le istruzioni dell'8086 possono avere una lunghezza variabile da uno a sei Byte, mentre le CPU di tipo RISC[5] hanno spesso istruzioni di lunghezza fissa, di 32 o 64 bit.

Ogni istruzione può essere divisa in due parti ("campi"). Una è dedicata al codice operativo, l'altra parte agli operandi. L'opcode indica quale istruzione deve essere eseguita, mentre gli operandi indicano i valori che devono essere usati per l'elaborazione.
Come esempio prendiamo l'istruzione dell'8086 MOX AX, 2, la cui versione in linguaggio macchina abbiamo già visto essere B80200. B8 è il codice operativo dell'istruzione, che dice che si tratta di un trasferimento (MOV) che ha come destinazione il registro AX. 0200 è il numero 2, scritto in un modo che spiegheremo più avanti, cioè l'operando dell'istruzione.

5 per la definizione di RISC vedi più avanti

Il numero degli operandi va da zero a tre, dipendendo dall'istruzione e dal tipo di CPU. Per esempio l'8086 ha istruzioni che accettano fino a due operandi, ma uno solo di essi può essere in memoria. Al contrario le CPU RISC accettano usualmente istruzioni a tre operandi.
Nella Tabella 1 si possono esaminare alcuni codici operativi della CPU 8086.

Istruzione	Op code (esadecimale)
NOP	90
MOV	88 (ed altri)
ADD	00 (ed altri)
JMP	E9 (ed altri)

Tabella 1: Codici operativi di alcune istruzioni 8086

1.1.4 Programmi

L'insieme delle istruzioni che servono a svolgere un ben determinato compito si chiama "**programma**".

Le porzioni di un programma vengono spesso dette "**codice**" (**code**), con nome un po' improprio ma che viene molto usato, anche in questo testo.
Quando un programma è in corso di elaborazione su di un computer si dice che sta "**eseguendo**" (is executing), con termine un po' più colloquiale si dice che sta "**girando**" (is running).
Durante l'esecuzione di un programma la CPU ha bisogno di numeri da elaborare, che potrà prendere dalla memoria. Per esempio le potrebbe essere richiesto di fare la somma fra due numeri. I numeri da sommare non sono il codice di niente, sono solo due semplici numeri, dei quali si vuole sapere la somma, cioè sono quelli che abbiamo definito "dati".
I computer con architettura di tipo Von Neumann conservano nella stessa memoria sia le istruzioni di programma che i dati. Esistono altri tipi di architetture che non rispettano questa condizione, ma i computer alla Von Neumann sono ancora di gran lunga i più diffusi.
Dunque le CPU alla Von Neumann usano la stessa memoria sia per capire quali comandi devono eseguire, sia per ottenere i dati che devono elaborare.
Il problema è: come la fa CPU a distinguere fra dati e istruzioni? La risposta fra un po', per ora è importante sottolineare che è indispensabile distinguere bene, fin da subito, fra istruzioni e dati, perché non capire la differenza porta a fare errori gravi e fondamentali.

Set d'istruzioni

L'insieme di tutte le istruzioni di macchina di una CPU è detto "**set d'istruzioni**" di quella CPU (in Inglese "insieme" si dice appunto "set"). In Appendice A3 si può vedere il set d'istruzioni di una CPU 8086.
Ogni modello di CPU ha il suo set d'istruzioni. CPU con lo stesso set d'istruzioni possono eseguire lo stesso programma senza bisogno di cambiarlo in alcun modo.

Famiglie di CPU e compatibilità binaria

Quando il produttore di una CPU ne realizza una nuova, più aggiornata e più potente, fa in modo che il set d'istruzioni della nuova comprenda anche tutto quello della vecchia. Anche i codici operativi delle vecchie istruzioni rimangono esattamente gli stessi nella nuova CPU. In questo modo tutto il software scritto per la vecchia CPU può girare senza alcuna modifica anche nella nuova.
Questa caratteristica della nuova CPU viene detta "compatibilità binaria" con la vecchia. Spesso i produttori che vogliono catturare una fetta di mercato di una importante CPU concorrente realizzano le loro CPU con lo stesso set d'istruzioni e gli stessi codici operativi, in modo che sia assicurata la compatibilità binaria. In questo caso le CPU vengono dette "compatibili".
Tutte le CPU che sono compatibili a livello binario vengono a far parte della stessa "famiglia" di CPU.
Il set d'istruzioni delle CPU più moderne di una famiglia sarà più esteso di quello delle vecchie, considerandolo come un insieme lo includerà (vedi Figura 7). Il produttore infatti introdurrà nuove istruzioni ad ogni nuova versione, dato che può disporre di tecnologie più moderne, che gli permettono di realizzare un maggior numero di transistor sul chip.
Naturalmente il codice scritto per un nuova CPU non è detto che sia compatibile con una vecchia. Basta che una sola istruzione di un programma sia presente nel set d'istruzioni della CPU nuova, ma non in quello della vecchia, per far sì che il programma non possa più girare sulla vecchia.

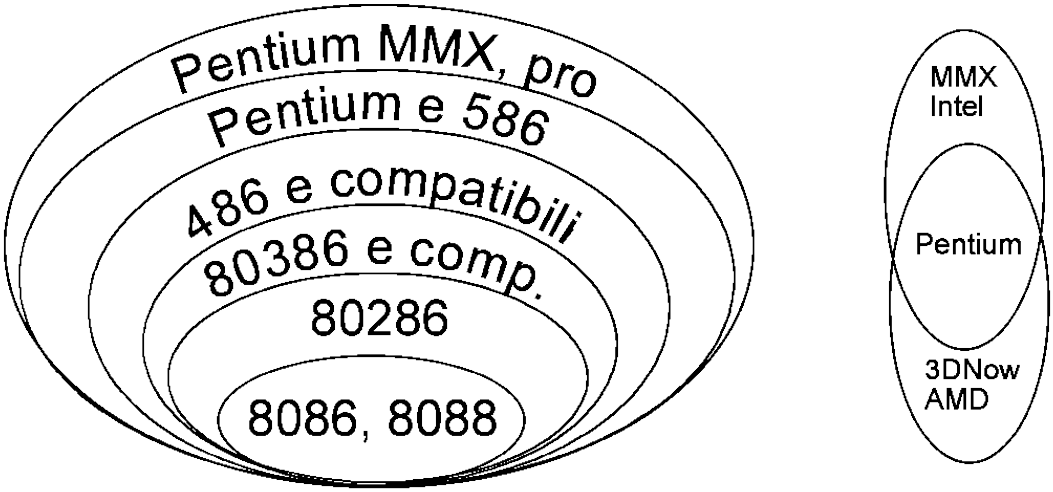

Figura 7: set d'istruzioni della famiglia 80X86

Linguaggio macchina

Un programma è dunque fatto di numeri. Un modo, brutale e folle, di programmare una CPU è quello di mettersi a scrivere in forma numerica i codici operativi dei programmi che si vogliono realizzare. Questo modo di programmare è detto lavorare in "**linguaggio macchina**" (machine code) ed è stato questo il lavoro dei primissimi programmatori. Per fortuna, e per la sanità mentale dell'umanità, tutto ciò non è durato molto. Sin dai primi stadi dello sviluppo dell'informatica il compito di tradurre i programmi in codici operativi è stato affidato ai computer, tramite altri programmi. Alle persone è stato riservato un modo più simbolico ed "umano" di esprimere le istruzioni di macchina. Questo modo è il "linguaggio Assembly" e che avremo modo di vedere con molto dettaglio più avanti.
Come esempio vediamo come sono codificate le istruzioni in linguaggio macchina 8086 che fanno la somma 2 + 2:

```
B8 02 00              (MOV AX, 2)
05 02 00              (ADD AX, 2)
```

Le istruzioni in linguaggio macchina precedenti sono scritte in forma esadecimale; fra parentesi è indicata la forma detta "mnemonica", in Assembly.

> *machine code*
> *The representation of a computer program which is actually read and interpreted by the computer. A program in machine code consists of a sequence of machine instructions (possibly interspersed with data). (..)The collection of all possible instructions for a particular computer is known as its "instruction set".*

da FOLDOC, the Free On-line Dictionary of Computing (in http://wombat.doc.ic.ac.uk/)

L'esecuzione di un programma

Un programma è un insieme di istruzioni in linguaggio macchina. Nei computer alla Von Neumann esso si trova, sotto forma di insieme di codici operativi, in locazioni contigue della memoria di lavoro. All'indirizzo successivo a quello dove finisce un'istruzione inizia l'istruzione che dovrà essere eseguita immediatamente dopo.
Quando il computer viene acceso la CPU prende dalla memoria il codice della prima istruzione che deve eseguire, poi la esegue, prende il codice della seconda, la esegue, poi prosegue con la terza .. e continua così per sempre.
L'ordine con cui le istruzioni di un programma vengono eseguite viene detto "**flusso**" di quel programma.
Di norma il flusso di un programma procede perciò da un'istruzione a quella che si trova subito dopo nella memoria[6]

Fetch e execute

La CPU quindi continua instancabilmente a fare queste due cose, passa continuamente fra questi due stati:
1) Acquisizione della prossima istruzione da eseguire
2) Esecuzione dell'istruzione
La fase 1) viene detta fase di "**fetch**", la 2) fase di "**execute**".
Il verbo to fetch in Inglese significa "andare a prendere" ed è alquanto esplicativo, ancora più significativo è "execute"!
Dunque in ogni istante del suo funzionamento la CPU si trova sempre in uno di questi due stati: fetch o execute.

6 avremo modo di vedere che questo flusso Sequenziale può essere spezzato con le istruzioni di salto.

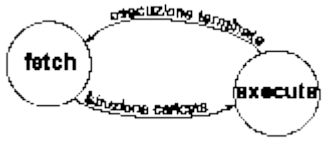

Figura 8: Stati di fetch e execute.

A questo punto ha senso porsi la domanda: "come fa la CPU a procurarsi il codice operativo dell'istruzione da eseguire?".
Il meccanismo è semplice ma "diabolico", come altri che vedremo in seguito. Bisogna che la CPU si ricordi sempre l'indirizzo della locazione alla quale troverà il codice operativo della prossima istruzione da eseguire.
Per sapere dove è la prossima istruzione da eseguire la CPU deve avere al suo interno una sorta di memoria. Questo tipo di "memoria", che non è una locazione di memoria principale, perché è all'interno della CPU, viene detta "**registro**". All'interno di una CPU esistono molti registri, con molti scopi. Uni dei tanti registri serve per memorizzare l'indirizzo della prossima istruzione da eseguire.

Chiameremo "**Program Counter**" (contatore di programma) il registro che "punta" sempre alla locazione in memoria ove si trova la prossima istruzione da eseguire.

All'interno del Program Counter ci sarà in ogni istante l'indirizzo della locazione nella quale la CPU potrà trovare il codice operativo della prossima istruzione da eseguire.
Durante la fase di fetch la CPU eseguirà questi passi:
1) farà un accesso alla memoria, all'indirizzo indicato dal Program Counter
2) leggerà dalla memoria l'op code dell'istruzione da eseguire e lo porterà al suo interno
3) aggiornerà il contenuto del Program Counter, in modo che alla prossima fase di fetch esso punti alla locazione successiva a quella dove finiva l'istruzione appena presa.
Dato che il programma viene memorizzato in locazioni successive, alla fine della fase di fetch il Program Counter punterà alla prossima istruzione da eseguire.
Di quello che accade durante la fase di execute parleremo successivamente, ma quando essa sarà finita la prossima fetch verrà fatta all'indirizzo indicato dal Program Counter, che era stato già sistemato.
Così il programma "va avanti da solo" eseguendo in modo automatico un'istruzione dopo l'altra, nell'ordine con cui sono state scritte in memoria.
La Figura 9 illustra questo funzionamento.
Inizialmente il Program Counter punta alla locazione 70 (punto 1 della figura). Il codice operativo della prima istruzione viene letto dall'indirizzo 70. Si tratta del numero 144 (90h), codice operativo dell'istruzione NOP dell'8086. Dopo aver letto il contenuto della locazione 70 il Program Counter viene automaticamente incrementato e diventa 71 (punto 2 di figura). Poi c'è l'execute della NOP (che non fa niente), infine il nuovo fetch alla locazione 71, dove ci sarà il primo Byte della successiva istruzione, che, per un 8086, è una MOV AX, 2.

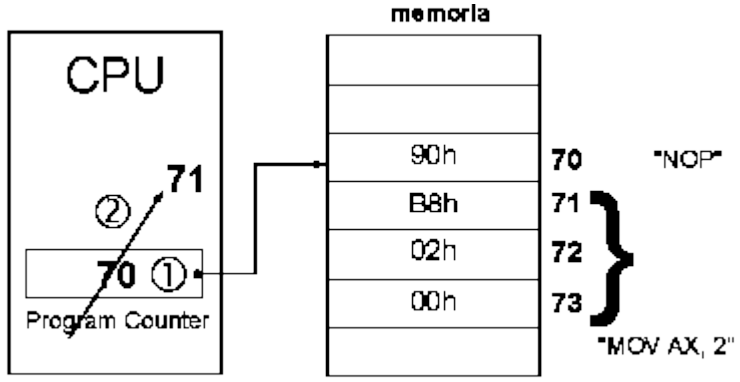

Figura 9: fase di fetch e Program Counter

L'unico dubbio che ci rimane da fugare è questo: "dove comincia tutto?" Quando accendo il computer dove punta il Program Counter?
La risposta è semplice: "ad una locazione fissa, stabilita dal costruttore della CPU".
Per esempio la prima istruzione che un 8086 esegue dopo che ha ricevuto un segnale di reset si trova all'indirizzo FFF0h. La prima cosa che fa un 8086 dopo il reset è caricare il suo Program Counter con l'indirizzo 0FFF0h.
A partire da quella locazione si trova un programma che fa parte del Sistema Operativo (vedi oltre) e che provvede a caricare dall'hard disk tutto il resto del software che permette al computer di funzionare. Naturalmente l'hardware del computer dovrà essere fatto in modo che all'indirizzo FFFF0h corrisponda una locazione di memoria che non si cancella quando il computer viene spento (ROM, vedi in seguito).

Ora possiamo rispondere alla domanda fatta precedentemente: "Come fa la CPU a distinguere fra istruzioni e dati, visto che sono mantenuti nella stessa memoria?".
La risposta a questo punto dovrebbe essere chiara. Dato che le istruzioni vengono caricate solo durante la fase di fetch la distinzione dipende dallo stato della CPU.
Se la CPU è in fase di fetch i numeri che passano sul data bus sono istruzioni, se è in fase di execute sono dati.

La maggior parte delle CPU, 8086 compreso, ha istruzioni di dimensione variabile. La parte iniziale del codice operativo indica alla CU di che tipo è l'operazione. Così la fase di fetch può proseguire e caricare nel registro d'istruzione più di una locazione. Vediamo alcuni esempi per 8086, nei quali si può notare come istruzioni con operandi diversi, in numero e tipo, abbiano lunghezze diverse:

Assembly	Linguaggio macchina	commento
NOP	90	Un'istruzione senza operandi; un Byte di lunghezza
MOV AX, BX	8B C3	Un'istruzione fra due registri, 8B è il codice operativo, C3 indica che l'istruzione lavora su AX e BX. Lunghezza due Byte
MOV AL, 2	B0 02	Un'istruzione con due operandi: un registro da 8 bit ed un numero da trasferire con accesso immediato. L'opcode B0 indica che è una MOV immediata con destinazione AL; poi segue 02, che è l'operando, che può essere un qualsiasi numero da 8 bit. Lunghezza due Byte.
MOV AX, 2	B8 0200	La stessa di prima con trasferimento a 16 bit. L'opcode è B8, diverso dal precedente perché indica un'operazione a 16 bit con risultato in AX. L'operando è sempre 2, ma questa volta prende 2 Byte (0200 è il numero 0002 scritto dal Byte basso a quello alto). Lunghezza tre Byte.
MOV word ptr [BX], 2	C707 0200	Istruzione con indirizzamento tramite BX ed accesso immediato al secondo operando (2).. Lunghezza 4 Byte.
MOV word ptr DS:[0Dh], 2	C706 0D00 0200	Istruzione con un accesso alla memoria, alla locazione 0Dh, ed un operando "fisso" a 16 bit, di valore 2. L'opcode prende due Byte: C706. Poi vengono gli operandi: 000D è l'indirizzo e 0002 (prima la parte bassa). Lunghezza 6 Byte.

1.1.5 CPU

Anche all'interno della CPU si possono individuare unità separate, ciascuna con le sue funzioni e caratteristiche costruttive.
All'interno di una CPU si delinea perciò una vera e propria "architettura", fatta di unità diverse interconnesse, che concorrono a generare un sistema che esternamente si presenta come un unico.
Ogni modello di CPU ha la sua architettura, che determina le sue modalità di funzionamento e le sue prestazioni.
Le CPU odierne hanno architetture di una complessità spaventosa, spesso all'interno hanno caratteristiche "non Von Neumann". La gran parte di esse peraltro si presenta all'esterno come una normale CPU di tipo Von Neumann con istruzioni molto potenti e veloci nell'esecuzione.
A partire da questo punto illustriamo l'architettura di una CPU ideale, nella quale sono inclusi i componenti tipici di tutte le CPU. Si noti che l'architettura che andremo a delineare non è quella di nessuna CPU effettivamente esistente, spesso semplificheremo e taglieremo qualche angolo, l'obiettivo non è saper progettare CPU ma capire come funzionano.

Nella Figura 10 è riportato uno schema dell'architettura della nostra CPU ideale.

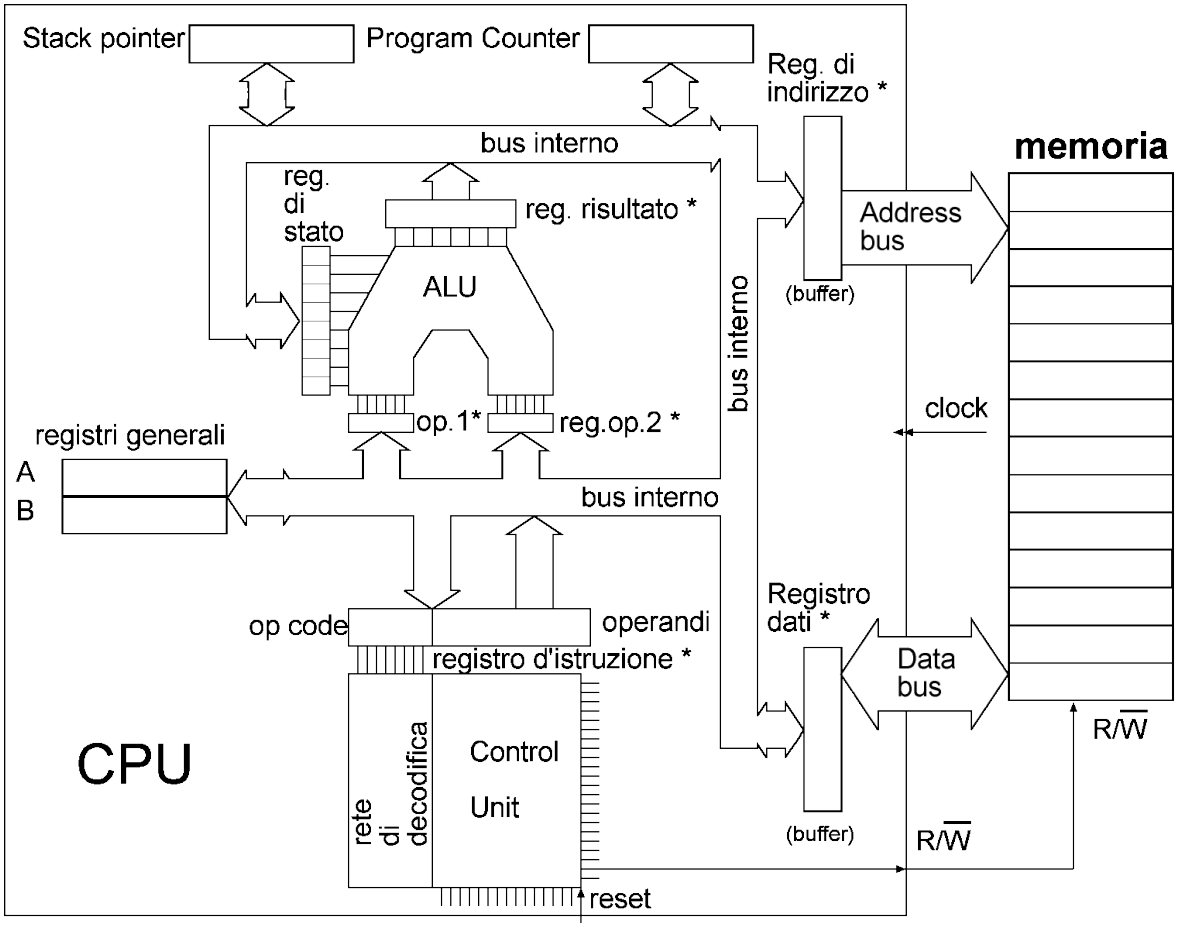

Figura 10: architettura di una CPU tipica

Le funzioni che ogni CPU deve essere in grado di svolgere sono:

- Effettuare trasferimenti di informazioni con la memoria
- Effettuare elaborazioni su quelle informazioni
- Effettuare confronti ed eseguire istruzioni diverse in conseguenza di quei confronti

Per effettuare trasferimenti di informazioni serve qualcosa per memorizzarle prima dell'elaborazione, per le elaborazioni ed i confronti serve una sorta di calcolatrice, per cambiare l'ordine di esecuzioni del programma basta che sia possibile modificare il Program Counter.
Infine ci sarà bisogno di circuiti per coordinare tutte le unità presenti nella CPU.
Di conseguenza i componenti che individuiamo all'interno della CPU sono:

Registri	Posizioni di memoria interne alla CPU
ALU	Unità che fa i calcoli
Control Unit	Unità che comanda ogni fase dell'esecuzione di ogni istruzione
Bus interni	Mezzi di trasmissione, analoghi ai bus esterni, trasferiscono i dati all'interno della CPU.

Registri

Per realizzare le sue funzioni la CPU ha bisogno di mantenere informazioni che non vengono perdute, ovverosia necessita di elementi di memoria al suo interno. Ogni elemento di memoria che risiede all'interno della CPU va sotto il nome di "registro".
La differenza sostanziale fra un registro ed una locazione di memoria è che un registro è all'interno della CPU, una locazione di memoria all'esterno (questa non è una "legge" e si possono trovare molte eccezioni, ma per il momento possiamo senz'altro prenderla per buona).
Il vantaggio di avere la possibilità di memorizzare numeri all'interno della CPU è legato alla velocità di esecuzione dei programmi. Se la CPU deve fare dei calcoli su un numero, ci vorrà molto meno tempo per lavorare su di un registro che su una locazione di memoria. L'accesso alla memoria di lavoro è infatti estremamente più lento dell'accesso ai registri, in quanto richiede una sequenza complessa di operazioni, l'utilizzazione dei bus e l'attesa della risposta dei chip di memoria, che si somma all'attesa perché i segnali sui bus diventino stabili.
Le funzioni assegnate ai registri dai progettisti delle CPU sono molto diverse.

Esistono registri interni alle CPU che non sono accessibili tramite programma. Essi non si possono usare esplicitamente nel programma, ma sono utilizzati dalla CPU durante le fasi di fetch e di execute. Essi non sono disponibili al programmatore alla fine della fase di execute, per cui li chiameremo "registri temporanei" (gli altri saranno registri e basta).
Vediamo allora quali sono i registri più comuni nelle varie CPU, che metteremo nella nostra CPU immaginaria:

Registri generali

Sono i registri in cui si memorizzano i risultati delle operazioni che la CPU sta eseguendo.
Alcuni tipi di CPU hanno registri accessibili individualmente da parte del programmatore, tramite il loro "nome", mentre in altre, in genere le più nuove, sono disponibili "banchi" di registri, che ne contengono diverse decine.

```
Esempio:
Nell'8086 una istruzione come questa:
    MOV AX, 1000
fa il trasferimento del numero 1000 nel registro AX, indicato con il "nome" che il pro-
gettista della CPU gli ha assegnato.
Un'istruzione analoga per il PowerPC potrebbe essere questa:
    lhz r9, 0 (r3)
se nel registro r3 c'è il numero 1000, fa il trasferimento del contenuto della locazione
1000 nel registro "general purpose" numero 9, chiamato per numero, visto che ci sono 32
registri generali.
```

Nella nostra architettura immaginaria mettiamo due soli registri generali, che chiamiamo A e B, con nomi simili a quelli dell'8086 ma non identici, per rimarcare il fatto che non si sta trattando esplicitamente dell'architettura dell'8086.

Registri per uso specifico (speciali)

I registri speciali sono presenti in ogni CPU per scopi particolari. Su ogni famiglia di CPU sono diversi, come sono fra loro diverse le architetture. Alcuni registri sono presenti in tutte le architetture, anche se spesso hanno nomi diversi da quelli che presenteremo di seguito.

Program Counter

Abbiamo già visto che mantiene sempre l'indirizzo della locazione in cui la CPU, all'inizio della fase di fetch, potrà trovare il primo Byte della prossima istruzione da eseguire.

Accumulatore

L'accumulatore è un registro nel quale finiscono i risultati delle operazioni aritmetiche che una CPU svolge. Nei primi microprocessori, che potevano avere pochi transistor al loro interno, le architetture ad accumulatore erano frequenti. In quei casi ogni operazione della ALU si concludeva sempre nello stesso registro, l'accumulatore. Dall'accumulatore il risultato doveva poi essere copiato nella destinazione finale con altre istruzioni.
Non spenderemo di più sul registro accumulatore perché nella famiglia X86 è un concetto presente solo per pochissime istruzioni. Per tutte le altre il risultato può essere messo nel registro che si vuole, ed anche direttamente in memoria. Per questa ragione non abbiamo messo l'accumulatore nello schema generale della nostra CPU.
Altre CPU, come per esempio i microcontrollori della famiglia PIC o quelli della famiglia 6502, su cui era basato il Commodore 64, sono basati su accumulatore.

Stack pointer

In ogni CPU esiste un registro speciale che sovrintende al funzionamento di una particolare area di memoria, detta stack. In questo registro, detto "**stack pointer**" è sempre memorizzato il valore dell'indirizzo dell'ultimo elemento che è stato scritto nello stack. Dello stack si tratterà con grande dettaglio nel capitolo ad esso dedicato.
Un altro registro speciale presente in tutte le CPU è il **registro di stato**, sul quale si tornerà presto.
Ogni tipo di CPU ha anche altri registri speciali, su alcuni di quelli dell'8086 si tornerà nel capitolo dedicato all'architettura di quella CPU.

Registri temporanei

Quando il progettista della CPU ha bisogno di registri nei quali memorizzare temporaneamente dei risultati intermedi e non può fare affidamento sui registri normali, introduce un registro temporaneo. Noi facciamo una cosa analoga. Vediamo di cosa abbiamo bisogno.

Registro d'istruzione

Quando siamo in fase di fetch e prendiamo l'istruzione dalla memoria, dove la mettiamo? Dobbiamo inventarci un registro. Supponiamo che durante tutta la fase di execute l'istruzione di macchina sia memorizzata in un apposito registro, che chiameremo "Registro d'istruzione" (**I**nstruction **R**egister (IR)).

Registro dati

Quando facciamo trasferimenti sul bus dei dati spesso dobbiamo anche svolgere delle operazioni sul numero che abbiamo letto in memoria. Per ricordarci che cosa abbiamo letto, o cosa dobbiamo scrivere, ci inventiamo un registro che abbia ogni suo bit attaccato ad un filo del data bus e che mostri in ogni istante alla CPU il contenuto di quel bus. Chiamiamo questo registro "Registro dati" (**M**emory **D**ata **R**egister, (MDR)).

Registro indirizzi

Mentre facciamo accesso alla memoria è necessario che l'indirizzo su cui si lavora sia sempre mantenuto sull'address bus. Dato che chi stabilisce il numero da mettere sull'address bus è la CPU, per non "dimenticarcelo" dobbiamo inventarci un registro che lo tenga memorizzato. Lo chiamiamo "Registro indirizzi" (**M**emory **A**ddress **R**egister (MAR))

Dal punto di vista elettrico gli ultimi due registri sono un po' diversi dagli altri; siccome sono collegati direttamente ai piedini esterni del microprocessore, devono essere dei "buffer". Hanno cioè transistor più grossi, che sono in grado di erogare correnti maggiori. Così possono far cambiare stato in tempi accettabili le linee dei bus esterni.

Avremo bisogno di altri registri temporanei, ma li introdurremo alla bisogna.

1.1.6 Bus interni

In ogni CPU ci sono diversi bus interni, alcuni dei quali vengono detti "dedicati" (dedicated) perché collegano direttamente fra loro solo due dispositivi. Gli altri sono "condivisi" (shared) e possono collegare molti dispositivi diversi.
Naturalmente i bus dedicati saranno molto più veloci, ma richiedono un tal numero di fili in giro per il circuito che ne consumano molta area; i bus condivisi richiedono meno spazio ma significheranno trasferimenti più lenti.
I progettisti della CPU dovranno fare saggi compromessi fra la velocità e lo spazio impegnato sul chip dai bus.
Per non dover scendere troppo nel dettaglio supponiamo che nella nostra CPU ideale ci sia un solo bus interno condiviso che collega tutte le unità della CPU.

1.1.7 Control Unit

La Control Unit (CU) è la parte della CPU che controlla ogni passo della fase di fetch e dell'esecuzione delle istruzioni.
Non tutte le istruzioni di macchina vengono eseguite dalla CPU in una sola fase. Molte hanno bisogno di proseguire a piccoli passi, in sequenze che possono essere lunghe centinaia di fasi.
La CU è un circuito estremamente complicato che ha come ingressi l'istruzione di macchina in forma binaria ed alcune linee che riportano informazioni sullo stato degli altri dispositivi. Le uscite della CU sono un gran numero di fili che si collegano con tutte le altre unità della CPU. Questi fili costituiscono delle linee di controllo, che accendono o spengono gli altri dispositivi. Attraverso queste linee la CU comanda le altre unità e le porta, un passo alla volta, a realizzare le istruzioni.
Alcuni dei fili che escono dalla CU sono portati anche fuori dalla CPU. Infatti la CU pilota anche le linee di controllo che escono dalla CPU; si può ben dire che abbia il controllo di tutto in computer, non solo della CPU.

Rete di decodifica

Durante la fase di fetch il codice macchina dell'istruzione viene portato nel registro d'istruzione. Poi la CPU dovrà eseguirlo, ma per poterlo fare dovrà prima "capire" di che istruzione si tratta.
Preposta a questo compito c'è un'unità che si può anche considerare parte della CU, detta "Decodificatore d'istruzioni". I circuiti di questa rete sono progettati per riconoscere il codice operativo dell'istruzione e fare in modo che la CU possa eseguire la giusta sequenza in fase di execute, così che la CPU realizzi l'istruzione richiesta.

Anche se non ce ne dovrebbe essere bisogno, si fa notare che questa rete non è un "decoder" nel senso delle reti logiche elettroniche, ma una rete, ben più complessa di un decoder, che svolge la funzione di "capire" un codice operativo (decodificarlo).

Clock, cicli di macchina e CU
Anche la CPU, come gli altri dispositivi del computer, disporrà di un segnale di clock. Oltre che per la sincronizzazione con i dispositivi esterni la CPU usa il segnale di clock per cambiare stato internamente.
Ogni volta che cambia di stato la CPU fa un piccolo passo nell'esecuzione delle sue istruzioni.

Ciascuno dei passi di queste sequenze viene detto "**ciclo di macchina**" (instruction cycle). Un ciclo di macchina dura da uno a pochi periodi di clock, e, dato un modello di CPU, è sempre costante.
La CU può essere realizzata fisicamente con due tecniche: in logica cablata (hardwired), cioè con le tecniche con cui si realizzano elettronicamente gli automi, oppure con tecnica microprogrammata (microprogrammed), con strumenti che ricordano molto da vicino i normali programmi, ma realizzate in hardware all'interno della CPU.

La tecnica microprogrammata è tipica delle architetture tradizionali (CISC), mentre la logica cablata è caratteristica delle architetture RISC[7]

CU in logica cablata (hardwired)
Se la CU è in logica cablata viene progettata come un normale circuito logico sequenziale, eventualmente partendo dall'automa, effettuandone la minimizzazione, poi la sintesi con le porte logiche che sono disponibili sul chip. Il risultato è un circuito estremamente complesso del quale è difficile verificare bene il funzionamento.

CU microprogrammata (microprogrammed)

7 per la definizione di RISC vedi il più avanti.

Se la CU è microprogrammata, essa è una sorta di "micro-CPU" all'interno di una CPU. La sequenza delle linee che la CU deve attivare è scritta in una memoria interna e si configura come un vero programma, con "microistruzioni" scritte in una memoria ROM interna ed esecuzione sequenziale. A differenza delle istruzioni normali il microprogramma esegue molte microistruzioni contemporaneamente, per aumentare l'efficienza del processore.
Le CU microprogrammate sono più semplici da progettare e flessibili da "riparare" e modificare. In linea di principio con una CU microprogrammata si potrebbero realizzare set d'istruzioni diversi sulla stessa CPU, cambiando solo il microprogramma. Nel passato sono state prodotte CPU in cui l'utente poteva cambiare il microprogramma, ma non hanno avuto successo commerciale. Con i microprogrammi è relativamente facile implementare set d'istruzioni complessi, con istruzioni che eseguano sequenze lunghe e complicate, mentre nel caso di CU hardwired ci si deve limitare ad istruzioni più semplici, se non si vuole correre il rischio di errori.
Le CU in logica cablata hanno il grande vantaggio di essere più veloci, tanto che le istruzioni vengono eseguite in pochissimi cicli di macchina, molto spesso in un solo ciclo. Di solito quelle istruzioni devono essere relativamente semplici, anche se negli ultimi tempi, con diversi milioni di transistor a disposizione, i termini di questo discorso sono un po' cambiati.

Sequenze

Dunque una CPU funziona con sequenze di piccoli passi ed esegue le sue istruzioni un po' alla volta. Vediamo, in un certo dettaglio, come la CU dovrebbe realizzare alcune tipiche sequenze della nostra CPU inventata.

Sequenza di accesso alla memoria

Per l'accesso alla memoria è necessario utilizzare un indirizzo. Come vedremo più avanti, questo indirizzo potrebbe essere dentro un registro, essere calcolato in qualche modo od anche essere scritto direttamente nel codice macchina.
Supponendo di avere l'indirizzo, e che sia 228 la CU fa eseguire questa sequenza (vedi Figura 11):
1) la CU controlla le altre unità della CPU in modo che 228 venga scritto nel registro d'indirizzo (MAR)
2) il numero scritto nel registro d'indirizzo compare nelle linee dell'address bus, in forma binaria, un filo del bus per ogni bit. La relativa locazione di memoria è abilitata al funzionamento
3) contemporaneamente a ciò che avviene nel punto 2 la CU alza la linea R/W (**W soprassegnato)**, comunicando alla memoria che il trasferimento sarà in lettura
4) la memoria mette sul Data Bus il contenuto della locazione 228 (602)
5) la CPU attende un certo numero di tempi di clock, per dar modo alla memoria di scrivere sul bus ed alle linee del bus di stabilizzarsi
6) La CU dà l'ordine al registro dati (MDR) di leggere il contenuto del data bus, memorizzandolo per uso futuro
L'accesso alla memoria è finito. Il dato è nel registro dati; da lì verrà successivamente smistato per la sua destinazione finale.
Se l'accesso fosse stato in scrittura, il dato avrebbe fatto il viaggio opposto, dal registro dati alla memoria, mentre la linea di controllo R/W (**W soprassegnato**) sarebbe stata tenuta bassa, ad indicare "scrittura". La sequenza per l'identificazione della locazione tramite l'indirizzo sarebbe stata identica.

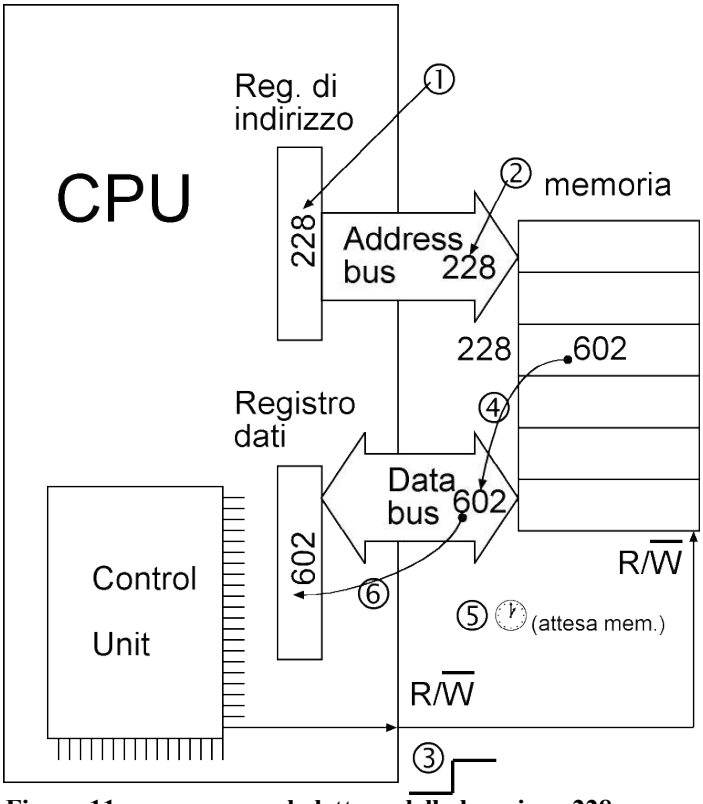

Figura 11: sequenza per la lettura della locazione 228

Sequenza di fetch
La fase di fetch di un'istruzione è una lettura in memoria. L'indirizzo al quale va fatta la lettura è quello contenuto nel Program Counter.
La sequenza è la seguente:
1) la CU ordina al Program Counter di mettere il suo contenuto sul bus interno; 191 sul bus interno
2) la CU ordina al registro indirizzi di leggere il bus interno; 191 sul registro indirizzi
3) ora c'è una sequenza di accesso in memoria in lettura, identica a come descritto precedentemente: 191 sull'address bus
4) contemporaneamente la CU alza la linea R/W **(W soprassegnato)**
5) la memoria prende il contenuto della locazione 191 (144 = 90h, op code di NOP) e lo mette sul Data Bus; 144 sul data bus
6) la CPU attende la stabilizzazione del data bus
7) il registro dati legge il data bus; 144 nel registro dati
8) la CU ordina al registro dati di scrivere il suo contenuto sul bus interno; 144 sul bus interno
9) la CU ordina al registro d'istruzione di leggere il bus interno; 144 nella parte "op code" del registro d'istruzione.
10) il Program Counter viene aumentato di 1, per prepararsi alla prossima fase di fetch
Questo conclude la fase di fetch: l'istruzione da eseguire è felicemente giunta a destinazione nel registro d'istruzione ed il Program Counter è stato già aggiornato.
Il progettista della CPU farà in modo che vengano eseguite contemporaneamente il maggior numero possibile delle operazioni precedenti. In questo modo la CPU impiegherà meno cicli di macchina per eseguire una fase di fetch e perciò sarà più veloce.

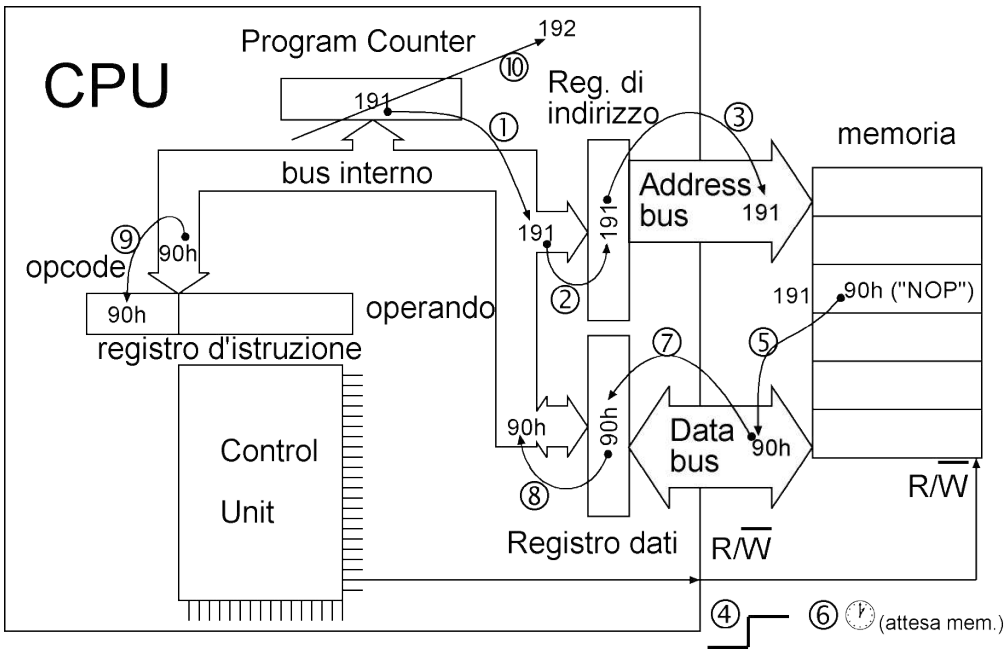

Figura 12: sequenza di fetch dell'istruzione NOP.

Al contrario dell'istruzione NOP, che non deve fare niente, molte istruzioni sono più lunghe di un Byte. Quando ciò accade la CPU effettuerà più accessi alla memoria, uno per ogni Byte dell'istruzione di macchina. Ogni volta aumenterà il Program Counter di uno, per raggiungere in memoria il nuovo Byte da leggere.
Alla fine, quando nel registro istruzione ci saranno sia l'op code che gli eventuali operandi, anche il Program Counter sarà correttamente aggiornato alla locazione seguente a quella dove siamo arrivati con il fetch.
E' importante rimarcare il fatto che durante la fase di fetch sul data bus passano istruzioni (codici operativi), anche se quello si chiama "data" bus, in fondo anche i codici sono "dati", anche se molto particolari!
Non c'è altro modo per leggere la prossima istruzione da eseguire, perché in ogni computer alla Von Neumann l'unico mezzo di comunicazione della CPU con il resto del mondo sono solamente i suoi piedini collegati al bus dati, sia in ingresso che in uscita.

Sequenze di execute

Naturalmente ci sarà una diversa sequenza di execute per ciascuna istruzione, e non tutte saranno così semplici come quella della NOP!
Quello che succede durante la fase di execute è:
1) l'op code dell'istruzione, giunto nel registro d'istruzione durante la fase di fetch, viene "capito" dal decodificatore d'istruzioni
2) comincia la sequenza di esecuzione, diversa in base a quanto decodificato nella fase precedente. La sequenza di esecuzione può essere da molto semplice (come la NOP), a molto complicata (come i calcoli a virgola mobile).
La fase 1) può cominciare anche durante la fase di fetch, in particolare in quelle CPU che hanno una lunghezza variabile delle istruzioni. Infatti la CU, dopo aver letto il primo Byte dell'istruzione (l'op code), saprà quante altre letture in memoria deve ordinare perché il fetch dell'istruzione sia completato.

1.1.8 ALU

L'ALU è la "calcolatrice" della CPU, come implica il suo nome di "**A**rithmetic **L**ogic **U**nit".
Sotto il controllo della CU, che indica le operazioni che deve compiere, l'**ALU** realizza quella parte delle istruzioni che richiede una elaborazione di dati.
In tutte le CPU l'ALU svolge le operazioni aritmetiche di somma e sottrazione fra numeri interi. Nella maggioranza delle CPU svolge anche operazioni di moltiplicazione e divisione fra numeri interi, mentre le operazioni aritmetiche con virgola mobile di solito sono assenti nelle CPU meno potenti, quali i microcontrollori, che devono essere più semplici, dato che devono costare molto poco.
L'ALU realizza anche operazioni logiche, quali la AND e la OR ed è in grado di fare confronti fra numeri. Il risultato dei confronti viene fissato dai flag, in modo che il programmatore, usando le istruzioni che segnalano il valore dei flag, sia in grado di far prendere strade diverse al programma e seconda dei risultati dei confronti.

Al contrario della CU, che è tipicamente una rete sequenziale (non a caso deve produrre le "sequenze" per la realizzazione delle istruzioni), la ALU è una rete logica di tipo combinatorio, almeno idealmente.

Il compito della ALU è infatti quello di effettuare operazioni algebriche, che potrebbero essere eseguite da una ALU ideale in un solo ciclo di macchina.
Una ALU ideale non ha quindi lo stato perché non ha la necessità di memorizzare nulla. Dati i suoi ingressi produce sempre gli stessi risultati, indipendentemente dal passato.
Nelle CPU moderne ci sono molte ALU, che possono mandare avanti più istruzioni in parallelo (su questo argomento si tornerà molto più avanti).

Nella nostra CPU ideale includiamo tre nuovi registri temporanei, che ci sono utili per descrivere il comportamento della ALU nell'economia dell'intera CPU. Due di questi registri sono all'ingresso dell'ALU e contengono gli operandi, prelevati da qualche altra parte della CPU o della memoria. Il terzo contiene il risultato dell'operazione, che viene mantenuto in questo registro in attesa di essere smistato alla sua destinazione finale. Si noti che in questo caso consideriamo che la ALU abbia due ingressi ed una uscita, anche se non è assolutamente detto che sia così nelle architetture reali.

Flag

Nell'America rurale la posta non viene distribuita nella casa ma nella strada più vicina. Così le persone che non vogliono farsi mezzo chilometro qualche volta al giorno per vedere se la posta è arrivata hanno inventato un ingegnoso sistema.
La cassetta della posta vuota ha una bandierina tenuta in questa posizione: .
Quando il postino arriva e mette nuova posta, alza la bandierina così che l'abitante della casa, col cannocchiale, potrà vedere la sua cassetta così: e decidere se è il caso di muoversi.
Questo sistema di comunicazione è basato su un'unica informazione (bandiera), di tipo binario (la bandiera può essere solo su o giù), che trasmette all'utilizzatore (abitante) lo stato del sistema (posta).
Come tutti sanno in Inglese bandiera si dice "flag". In Informatica, per analogia con l'uso "postale" delle bandiere, i bit tramite i quali si indica lo stato di un elemento del sistema vengono detti "flag".

Un **flag** è quindi un bit di informazione tramite il quale la CPU comunica al programmatore il verificarsi di certe condizioni.

In particolare la CPU può comunicare al programmatore cosa è successo durante l'esecuzione di ogni istruzione attraverso una batteria di flag, contenuti in un registro detto "**registro di stato**".
Per esempio se un'operazione di somma ha dato luogo ad un riporto, la CPU lo indicherà mettendo a uno il flag detto di "carry" (riporto).
Altri flag si sistemano in conseguenza di un confronto, per cui siamo in grado di sapere se un numero è maggiore di un altro.
Altri flag abilitano certe funzioni della CPU, per cui se il flag vale 1 (si dice che "è On") la CPU lavora in un certo modo, se è Off (vale zero) lavora diversamente.
Il programmatore ha a disposizione alcune istruzioni per fare in modo che il programma reagisca diversamente se un flag ha un valore piuttosto che un altro. In questo modo i programmi possono prendere decisioni ed eseguire istruzioni diverse secondo quanto è successo in istruzioni precedenti.
Ciascuno dei flag contenuti nel registro di stato ha una "vita" indipendente dagli altri, quindi piuttosto che di registro di stato si dovrebbe parlare di registri indipendenti della dimensione di un bit, uno per ogni flag. La ragione per cui si parla di registro di stato è perché le CPU hanno delle istruzioni che, se serve, salvano in memoria tutti insieme i flag del registro di stato.
Nel nostro disegno della CPU il registro dei flag è vicino alla ALU, questo perché è la ALU che determina il valore della maggior parte dei flag, i più importanti dei quali indicano la condizione del risultato di operazioni aritmetiche o logiche.

Sequenze di execute

Una volta conosciuti tutti i componenti della CPU, con la Figura 10 sott'occhio non è difficile immaginare le sequenze che la CPU deve realizzare per far eseguire le istruzioni.
Per esercizio si può provare a descrivere la fase di execute delle seguenti istruzioni, elencate in ordine di complessità. Il funzionamento di ogni istruzione è spiegato brevemente nel commento alla sua destra.

```
MOV A, 2         ; mette nel registro A il numero 2. Si consideri che l'operando
                 ; 2 sia già caricato nel registro d'istruzione in fase di fetch
MOV A, [2]       ; mette nel registro A il numero letto alla locazione di indirizzo 2
ADD A, 2         ; mette in A il suo valore iniziale, cui viene aggiunto 2
MOV [2], A       ; scrive nella locazione 2 il valore attuale di A
MOV A, [B]       ; legge in A il numero contenuto nella locazione il cui indirizzo
                 ; è contenuto in B
```

Come compendio sulle sequenze della CU, si descrive di seguito la sequenza di execute dell'istruzione ADD [0A0h], 3Ah. Questa istruzione accede all'indirizzo A0h, legge il suo contenuto, lo somma al numero 3A e scrive il risultato della somma in memoria, nuovamente all'indirizzo A0h.
Si suppone che l'istruzione sia codificata in questo modo: 8006 A0 3A (questo è l'effettivo codice macchina 8086 per l'istruzione).
Durante la fase di fetch (punti da [1] a [8] di Figura 13) la CPU accede a tutte le locazioni dell'istruzione e la carica nel registro d'istruzione.
Alla fine della fase di fetch in P.C. punta alla prossima istruzione (punto[8] di Figura 13).
Il punto [9] della sequenza inizia l'execute. Con [9] e [10] l'indirizzo A0h viene messo nel registro d'indirizzo, passando per il bus interno, e "punta" alla locazione di memoria dove leggere il dato (nel punto [11] la CU ordina una lettura). Dopo il tempo di assestamento di bus e memoria il numero (31h) può essere portato nel registro dati ([12] e [13]), poi sul bus interno ([14]).
Ora il numero 31h può essere copiato dal bus interno ad uno dei registri operando ([15]). L'altro operando (di valore 3A), che è dentro il registro d'istruzione, può essere copiato con le operazioni [16] e [15].
A questo punto la CU ordina alla CPU di eseguire una somma ([18]). Quando il risultato (6B) è pronto bisogna scriverlo in memoria. Per questo il numero deve passare sul data bus, mentre l'indirizzo è già pronto nel registro di indirizzo, dato che non è cambiato dalla precedente lettura.
[19] e [20] portano il numero (6B) dal registro di risultato della ALU al data bus. Intanto la CU ordina il trasferimento in scrittura, commutando la linea R/W **soprassegnato** su livello basso.
Dopo l'attesa per l'accesso alla memoria il risultato è stato scritto alla locazione voluta ([21]) e l'esecuzione dell'istruzione è finita.
La CPU comincia la fase di fetch dell'istruzione successiva, all'indirizzo 12A5, come indicato dal P.C. .

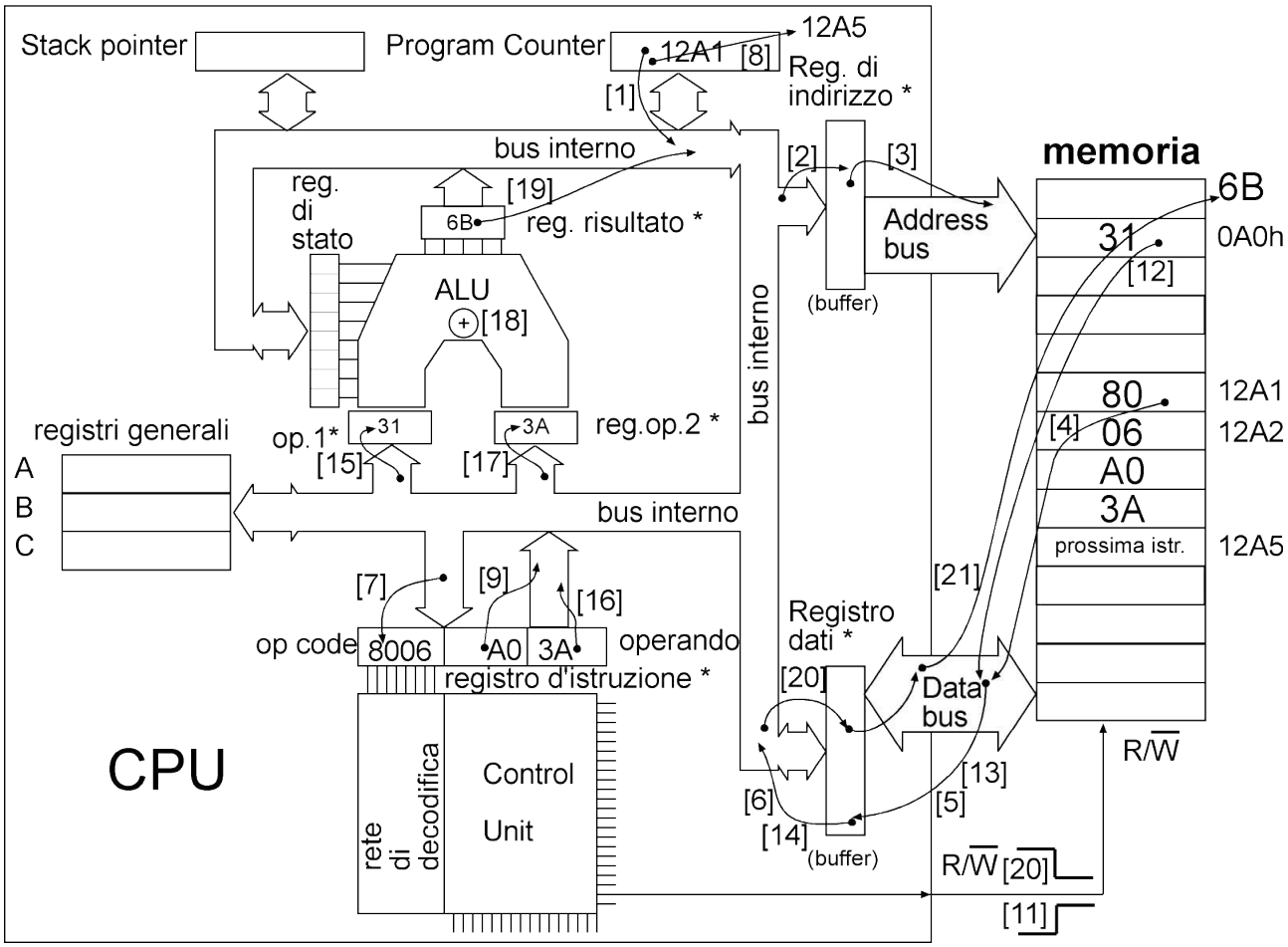

Figura 13: esecuzione dell'istruzione ADD [0A0h], 3Ah

Curiosità

Informatica
Il termine "informatica" è una bella sintesi di "informazione" ed "automatica", inventato in Francia ed usato in tutti i paesi europei, adattato alla lingua locale. Negli USA il termine non si usa :-). Per difficoltà di pronuncia o ritrosia nell'accettare termini stranieri, in USA per indicare l'Informatica si diceva "computer science". Di fronte al fatto che in Informatica c'è molta tecnica e poca scienza, il termine più in voga oggi in America è "**I**nformation **T**echnology" (IT).