

Fondamenti di Informatica

Parte 2

CCL - Amb-Ter

Facoltà di Ingegneria
Università degli Studi di Firenze

Ver.: 1.4

Data: 19/3/2000

Questo documento fa parte di una serie di dispense prodotte dallo sforzo volontario di molti ma che sono state riviste e corrette dal docente solo in parte. Nonostante la loro incompiutezza ho deciso di distribuirle per dare comunque una traccia sugli argomenti del corso in mancanza di un libro di riferimento. Non rappresentano pertanto ne' il libro ne' il programma di riferimento del corso, a questo fine fanno fede le lezioni stesse che in molte parti differiranno da queste dispense.

Si prega pertanto di segnalare ogni mancanza e correzione inviando una mail al Prof. P. Nesi al seguente indirizzo di posta elettronica con soggetto "dispense": nesi@dsi.unifi.it, con la speranza di arrivare a produrre una versione rivista in breve tempo, anche con il Vostro aiuto, grazie!

INDICE

1	Cenni Storici.....	4
2	Architettura dell'elaboratore.....	6
2.1	La Macchina di Von Neumann (1947).....	6
	L'elaboratore.....	8
2.3	La Memoria di base e del BUS di Sistema.....	8
2.3.1	Rappresentazione logica della memoria.....	9
2.3.2	Rappresentazione fisica della memoria.....	10
2.3.3	Banchi di memoria, composizione di memorie.....	12
2.3.4	Ciclo di lettura.....	13
2.3.5	Ciclo di scrittura.....	13
2.4	Le memorie di massa.....	14
2.4.1	Floppy Disk.....	14
2.4.2	Disco Rigido, Hard Disk.....	15
2.4.3	Compact Disk.....	15
2.4.4	Memoria a Nastro.....	16
2.5	Il Calcolatore e il Microprocessore.....	16
2.6	Caratteristiche dei Microprocessori.....	17
2.6.1	L'evoluzione dei Microprocessori.....	19
2.6.2	I Microcontrollori.....	19
3	La struttura interna del microprocessore.....	20
3.1	ALU.....	21
3.2	Registri interni.....	22
3.3	L'Unità di Controllo.....	25
3.4	Le Prestazioni dei Microprocessori.....	25
4	Il Linguaggio Assembly.....	27
4.1	Tipi di Istruzioni.....	28
4.2	Le Istruzioni.....	29
4.2.1	Istruzione LOAD.....	29
4.2.2	Istruzione SAVE.....	31
4.2.3	Istruzione ADD.....	31
4.2.4	Istruzione SUB.....	32
4.2.5	Istruzione MUL.....	33
4.2.6	Istruzione DIV.....	34
4.2.7	Istruzione SQR.....	34
4.2.8	Istruzione MOVE.....	35
4.2.9	Istruzione NOT.....	35
4.2.10	Istruzione READ.....	36
4.2.11	Istruzione WRITE.....	37
4.2.12	Istruzione AND.....	37
4.2.13	Istruzione OR.....	38
4.2.14	Istruzione SHL.....	38

4.2.15 Istruzione SHR.....	39
4.3 Le Istruzioni di Salto.....	39
4.3.1 Salto incondizionato.....	39
4.3.2 Salti condizionati.....	40
4.4 Le Istruzioni per la gestione del Contesto.....	41
4.5 La soluzione di un Equazione di secondo grado.....	41
4.6 La somma degli elementi di un vettore	45
4.7 Alcuni Esercizi da fare	46
4.8 Ordine di esecuzione delle istruzioni.....	46
4.9 Modalità di Indirizzamento	48
4.9.1 Indirizzamento Diretto	48
4.9.2 Indirizzamento Indiretto.....	49
4.9.3 Indirizzamento Indicizzato.....	49
4.9.4 Indirizzamento Basato.....	50
5 I Moduli e il Linker	51
6 Hardware e Software.....	52
7 EBNF, Forma di Backus-Naur Estesa.....	54
7.1 Il metalinguaggio EBNF.....	55
7.2 SIMBOLI TERMINALI DELL'EBNF.....	56
7.3 SIMBOLI TERMINALI O IDENTIFICATORI (VOCABOLARIO) DEL PASCAL.....	56
7.3.1 Esempio, l' Assembly.....	57
7.3.2 Esempio: il Pascal.....	58

1 Cenni Storici

1642 -- Macchina calcolatrice di Pascal

La macchina calcolatrice di Pascal aveva le dimensioni di una scatola per scarpe e presentava una interfaccia utente con delle “rotelline” con riportati i numeri da 1 a 9. Queste rotelline funzionavano grosso modo come il contachilometri di un’automobile: quando la prima di esse arrivava al numero 9 ritorna al numero 0 facendo passare quella alla sua sinistra al numero successivo, per esempio se era zero diventa 1 in modo da formare il 10. La novità rivoluzionaria di questa macchina stava nell’introduzione del principio del “riporto automatico“. Venne definita novità rivoluzionaria in quanto tutti i successivi progetti relativi alla costruzione di macchine addizionatrici e calcolatrici avrebbero preso le mosse dal principio del riporto automatico. L’unico limite dell’invenzione di Pascal era che la macchina permetteva soltanto di eseguire addizioni e sottrazioni.

1671 -- Modifiche apportate alla macchina calcolatrice da Leibniz

Leibniz perfezionò il principio della” calcolatrice” costruita da Pascal inventando una macchina in grado di eseguire anche moltiplicazioni e divisioni, che venne chiamata: la macchina moltiplicatrice. Erano macchine non programmabili ma dotate di programmi definiti dalla meccanica stessa della macchina.

1801 -- Jacquart inventa la scheda perforata

Jacquart inventò un telaio per la tessitura automatica che funzionava per mezzo di una serie di schede perforate dove, la posizione dei fori guidava il filo a formare un certo disegno nell’ordito. Questa invenzione oltre ad apportare un miglioramento per la tessitura costituì anche una pietra miliare nell’evoluzione dei computer. Le schede erano la prima forma di programmazione di una macchina automatica.

1823 -- Macchina di Babbage

Babbage progettò una nuova macchina calcolatrice automatica sfruttando il principio sia della macchina di Pascal sia del telaio di Jacquard. La macchina di Babbage utilizzava due serie di schede perforate; una di serie di queste costituiva il programma, cioè l’insieme delle istruzioni da dare alla macchina, l’altra serie di schede rappresentava i dati, cioè i valori sui quali effettuare i calcoli da eseguire in base alla prima serie di schede. Babbage lavorò al progetto per circa 40 anni, ma non riuscì mai a costruire un prototipo funzionante della sua calcolatrice. Forse pensò di aver fallito, ma in realtà il suo lavoro fu così importante che ancora oggi egli viene considerato il “Padre dei computer“, tanto che le sue idee sono alla base della moderna programmazione.

1947 -- Macchina di Von Neumann Princeton

Von Neumann progettò una macchina dalla cui evoluzione è derivata la struttura delle macchine attuali.

I vantaggi della macchina di Von Neumann

- Flessibilità codice e dati nella stessa memoria (nella macchina di Babbage vi erano due insiemi di schede)
- Facilità nel cercare il nuovo codice per l'arrivo delle informazioni in modo sequenziale (non parallelo) cioè un dato dopo l'altro e non tutti insieme
- Il codice del programma può mutare durante l'esecuzione. Automodifica del codice

SVANTAGGI della macchina di Von Neumann

- Non e' possibile caricare codice e dati assieme
- Si possono provocare sovrascritture del codice manipolando questo come se fossero dati.

1951 -- Ultime modifiche alla macchina (20kparole, 40bit)

Venne realizzato un calcolatore detto UNIVAC che era molto più sofisticato e lavorava sia con parole che con numeri. Era però ancora così ingombrante e costoso che poche compagnie potevano permetterselo.

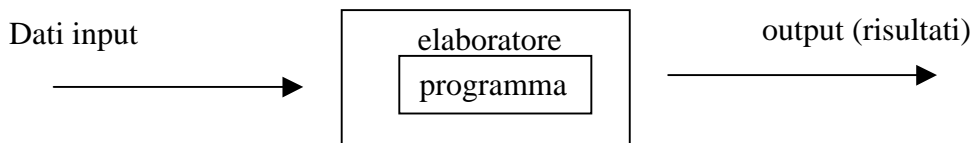
Fondamentalmente i computer moderni lavorano come l'UNIVAC.

Quello che li differenziava era la tecnologia di costruzione i cui progressi hanno reso possibile la realizzazione di macchine più piccole e molto più veloci.

2 Architettura dell'elaboratore

Un elaboratore elettronico è una macchina a componenti elettronici capace di elaborare dati in maniera automatica secondo un preciso algoritmo: tale algoritmo si dice "programma".

Una volta che il programma è allocato all'interno del computer, questo acquisisce dati dall'esterno, li elabora automaticamente e fornisce i risultati ottenuti.



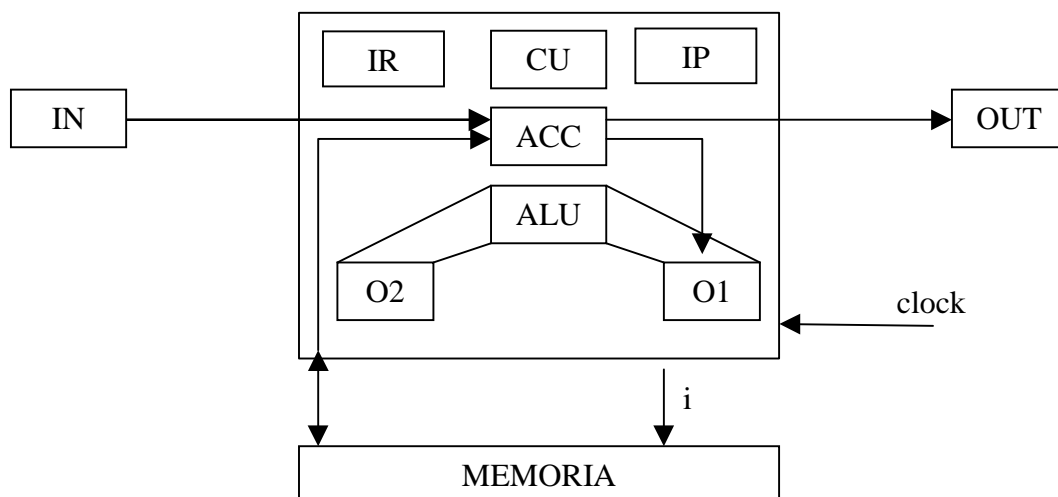
L'elaboratore lavora sempre attraverso informazioni espresse in forma binaria, quindi sia i dati che le istruzioni (programma) devono essere codificati in binario.

2.1 La Macchina di Von Neumann (1947)

La maggior parte degli elaboratori attuali si rifanno alla macchina di Von Neumann questa si basa sulle seguenti ipotesi:

- un sistema di ingresso sequenziale
- un sistema di uscita sequenziale
- una memoria in cui vi sono sia dati che le istruzioni (programma)

La macchina di Von Neumann ha il seguente schema :



- C U = Control Unit
- ALU = Aritmetic Logic Unit
- CPU = Control Processing Unit
- I = indice che permette di identificare la singola cella della memoria

- IP = registro interno alla CPU che permette di puntare all'istruzione successiva (instruction pointer)
- IR = registro interno alla CPU che contiene l'istruzione successiva a quella che stiamo eseguendo, cioè l'istruzione corretta. Tale istruzione viene contenuta in codice macchina (instruction register). Il codice macchina è la rappresentazione binaria dell'istruzione. Per esempio un'istruzione di ADD potrebbe avere 01010010.

Tutte le operazioni sono effettuate in riferimento al registro accumulatore.

Tutti i risultati delle operazioni sono copiati direttamente in un registro OI copia del registro accumulatore pertanto le operazioni sono tutte del tipo:

Acc = Acc <operazione> OI

Oltre alle operazioni aritmetiche le operazioni di base della macchina sono:

Read : lettura dall'ingresso

Acc <- IN (shift)
OI <- Acc

Write : scrittura sull'uscita

Out <- Acc (shift)

Load <i> : lettura della i-esima cella della memoria

Acc <- M(i)
OI <- Acc

Save <i> : scrittura della i-esima cella della memoria

M(i) <- Acc

Add <i> : somma del contenuto del registro accumulatore con il contenuto della cella i-esima della memoria e risultato nel registro accumulatore.

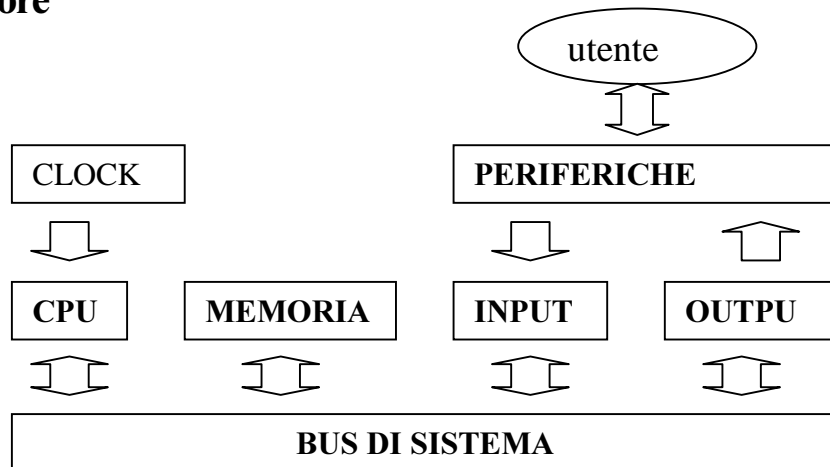
Cioè $Acc = Acc + OI$

OI <- M(i)
ALU(somma)
OI <- Acc

Le microistruzioni vengono eseguite in fasi diverse e sono scandite da colpi di clock. Per esempio: 7 colpi di Clock per ogni ADD uno per ogni operazione interna della CPU.

Vi sono inoltre altre istruzioni per effettuare operazioni matematiche più complesse, per esempio la negazione (per la rappresentazione in Complemento a 2), lo shift, le operazioni di and, etc.

2.2 L'elaboratore



L'elaboratore è formato da 5 sottosistemi:

- La **CPU (Central Process Unit)** chiamata anche microprocessore, è ciò che provvede all'esecuzione del programma e che controlla e coordina le altre unità del computer. La CPU è composta da CU (Control Unit) e dall'ALU (Arithmetic Logic Unit).
 - **Control Unit:** unità di controllo per la traduzione delle istruzioni da linguaggio macchina alle relative azioni che devono essere svolte per eseguire le istruzioni del programma in esecuzione.
 - **ALU:** l'unità in cui vengono effettuate le operazioni sia algebriche che logiche. Questa comprende l'implementazione in termini di componenti elettronica degli algoritmi per effettuare le operazioni logiche e/o algebriche.
- la **memoria centrale**, dove sono collocati il programma ed i vari dati utili all'elaboratore. Esiste anche la memoria di massa ma viene vista dalla CPU come un dispositivo di Ingresso/uscita.
- le unità di **ingresso**: tastiera, mouse, trackball, ecc.
- le unità di **uscita**: monitor, stampante, ecc.
- Il **BUS di sistema** collega tra loro le varie unità. Questo BUS si divide in BUS dati, indirizzi e controlli. Nei calcolatori più moderni vi sono diversi tipi di BUS che tipicamente sono organizzati gerarchicamente da quello più veloce a quello più lento partendo dalla CPU verso le periferiche.

2.3 La Memoria di base e del BUS di Sistema

Le memorie centrali presenti in un microcalcolatore si distinguono in memoria a sola lettura (ROM, Read Only Memory) e in memoria a lettura e scrittura (RAM, Random Access Memory). Le memorie ROM contengono almeno le istruzioni di base (firmware) per consentire l'accensione della macchina e il collegamento della stessa con i terminali di ingresso /uscita, mentre le memorie RAM sono indispensabili per conservare i dati intermedi delle elaborazioni, i risultati dei calcoli, ecc. Il Firmware mette a disposizione una serie di programmi (sequenze di istruzioni) che permettono di inizializzare le periferiche: la tastiera, il video, i lettori per dischetti, etc.

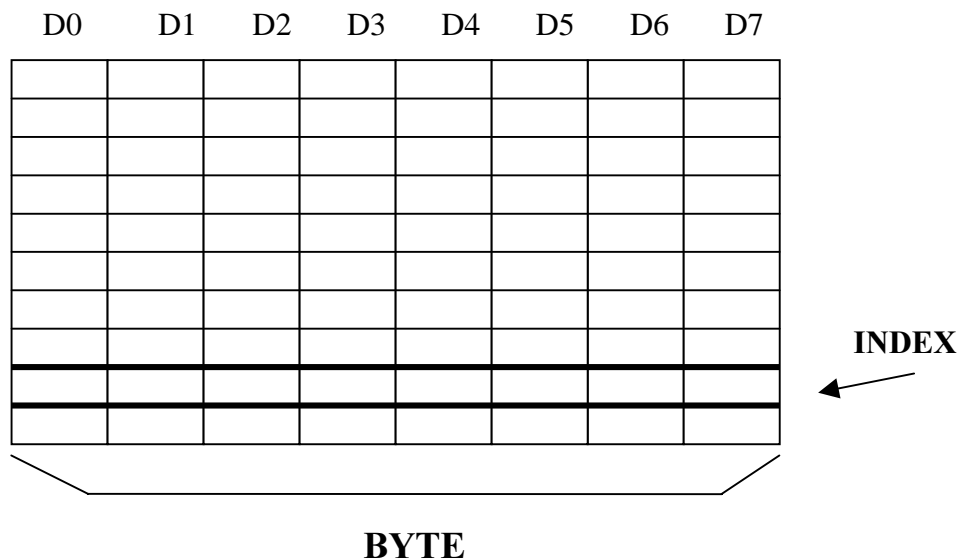
La memoria centrale detta anche di base è ormai solo a stato solido, cioè costruita con dispositivi a stato solido. Della memoria se ne può fare una rappresentazione logica e fisica.

2.3.1 Rappresentazione logica della memoria

La memoria si può schematizzare come un vettore composto da celle in cui vengono immagazzinate informazioni. Ogni cella ha un indice che costituisce il suo "indirizzo" all'interno della memoria, così, conoscendo l'indice l'elaboratore trova l'esatta posizione della cella nella memoria (per esempio 53) e acquisisce i dati e le informazioni (in forma binaria) che vi sono contenute. La numerazione delle celle può essere consecutiva o meno.

La dimensione di una cella è data dal numero di bit che essa contiene: può essere di un bit, ma anche di 8, 16, 32, 64, a seconda di come è organizzata la memoria.

Per esempio, se può realizzare una memoria di 1024 byte con 8 banchi di memoria da 1024 bit in parallelo, in pratica si collegano tra loro più memorie (come se fossero vettori di memoria) e si operano operazioni di lettura e scrittura di tali vettori di memoria in parallelo cioè leggendo i bit degli stessi banchi nella stessa posizione indicata dall'indice allo stesso momento. In questo modo si possono realizzare memorie che contengono dati da 8, 16, 32 e 64 bit. Per esempio nella figura è riportata una memoria da N byte ottenuta con 8 memorie da N bit.



Tipi di memoria:

- RAM** (Random access memory)
- ROM** (Read only memory)
- PROM** (Programmable ROM)
- EPROM** (Elettrically PROM)
- EEPROM** (Elettrically Erasable PROM)

RAM: memoria ad accesso casuale. Nelle celle di questo tipo di memorie si può sia scrivere che leggere. La memoria RAM è una memoria volatile, cioè si perde l'informazione ogni volta che viene a mancare l'alimentazione. Si chiama ad accesso casuale poiché è possibile accedere ad ogni singola cella in modo diretto e pertanto la modalità casuale è possibile. Questa modalità è da vedere contrapposta alla modalità sequenziale per la quale per accedere ad una cella si deve prima avere fatto accesso alla precedente, etc. Partendo dalla prima cella della memoria.

ROM: in questa memoria si può solamente leggere il contenuto delle celle. I dati immagazzinati nella memoria ROM permangono sempre anche se viene a mancare l'alimentazione. Tali memorie sono

prodotte direttamente in fonderia.

PROM: come la ROM ma può essere programmata dall'utente programmatore una sola volta. La programmazione di queste memorie avviene tramite un dispositivo elettronico detto programmatore che produce particolari tensioni di alimentazione e sequenze di operazioni al fine di provocare delle "lesioni" permanenti nella matrice che rappresenta la memoria.

EPROM: questa memoria può essere scritta e cancellata, ma solo globalmente sottoponendola a raggi UV per un certo numero di minuti. Dopo la cancellazione può essere riscritta. Questa operazione può essere ripetuta un numero limitato di volte. Si noti che per la cancellazione è necessario smontare la EPROM dal circuito sul quale è installata.

EEPROM: come la EPROM ma le operazioni di cancellazione possono essere più selettive e vengono effettuate tramite segnali elettrici. Non è necessario rimuovere la EEPROM dal circuito per effettuare le cancellazioni. Dopo la cancellazione può essere riscritta. Questa operazione può essere effettuata un numero limitato di volte.

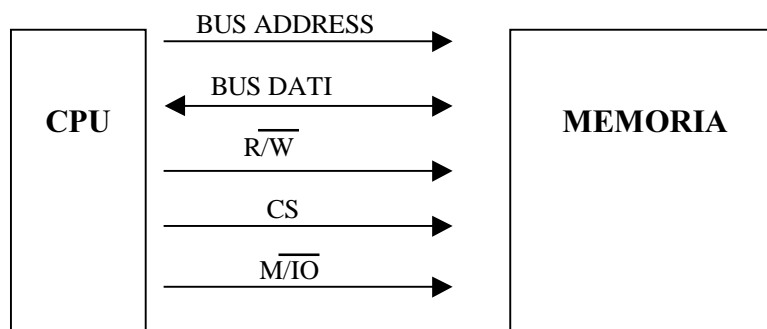
RAM, ROM, EPROM, PROM, EEPROM sono memorie ad accesso diretto in cui, cioè, si accede direttamente in base all'indirizzo proprio di ogni cella. Nei calcolatori vi è di solito la memoria RAM e EPROM; quest'ultima può essere utilizzata come memoria ROM.

2.3.2 Rappresentazione fisica della memoria

Fisicamente si può vedere la memoria come un unico sistema: la CPU invia degli indirizzi alla memoria ed ad altre periferiche attraverso un "BUS ADDRESS". L'indirizzo prodotto dalla CPU identifica una cella di memoria (o di una periferica come vedremo in seguito) e questa può essere scritta o letta. La specifica operazione che la CPU intende effettuare è regolata dai segnali di controllo R/W (read/write), CS, e M/I/O (memoria o ingresso/uscita). Nel caso specifico saranno operazioni di lettura/Read quando il segnale R/W è vero e di scrittura/WRITE quando è falso.

Quando le informazioni sono lette dalla memoria queste arrivano alla CPU attraverso il BUS DATI. Lo stesso BUS viene utilizzato per inviare dei dati dalla CPU alla memoria stessa. Il segnale di controllo CS (chip select) identifica il singolo blocco/chip di memoria qualora ve ne sia più di uno. Il segnale CS viene anche detto COMANDO.

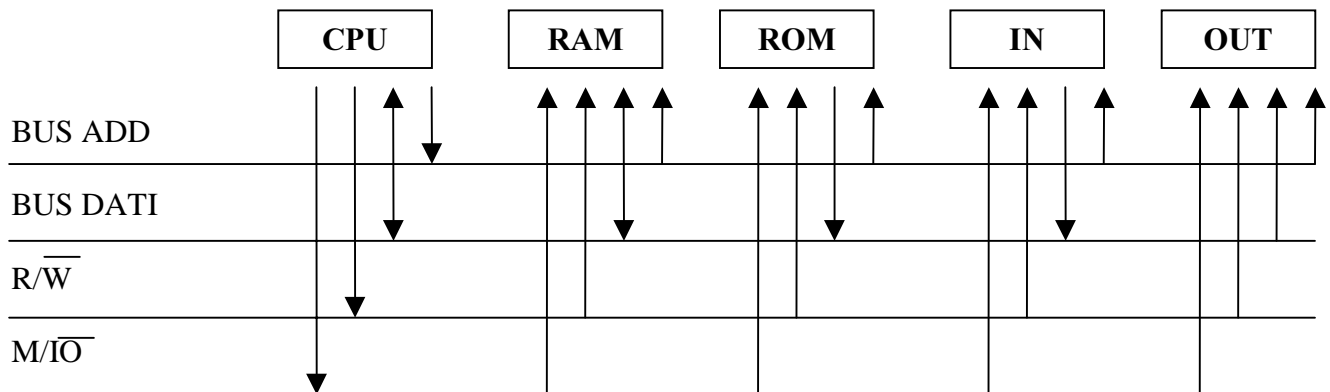
Tali informazioni non sono altro che il contenuto delle celle corrispondenti agli indirizzi mandati dalla CPU. In schema:



Se la memoria ha 1024 celle ognuna con capacità 16 bit (una word) il BUS Address deve essere composto da 10 fili/linee conduttori cioè si ha la necessità di inviare un pacchetto parallelo completo per ogni ciclo macchina. Sul BUS dati passano informazioni a 16 bit, quindi esso è composto da 16 fili/linee elettriche: su ogni filo/linea passa un'informazione in forma binaria 1/O.

Memoria totale = 1024 celle da 16 bit = 16384 bit = 2048 Byte

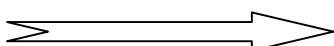
Vediamo più in dettaglio com'è composto il BUS di SISTEMA:

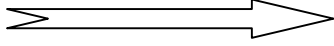


Tutte le operazioni sono regolate dalla CPU. Tuttavia esistono dei meccanismi (ad esempio di interruzione) che permettono al sistema di ingresso di attirare l'attenzione della CPU che, a tal sollecitazione, può effettuare operazioni specifiche, per esempio di lettura dati dalla periferica che ne ha attirato l'attenzione.

La CPU inoltre opera su un segnale di selezione (M/IO, memory or input/output). Quando il segnale è vero si effettuano delle operazioni in memoria mentre quando il segnale è falso si hanno operazioni con le periferiche di ingresso/uscita.

Con R/W, READ/WRITE; la CPU sceglie se leggere o scrivere sulla memoria o dall'ingresso/uscita.

(alta tensione) memory output  read from memory

(bassa tensione) memory input  write on memory

Il tutto secondo la seguente tabella:

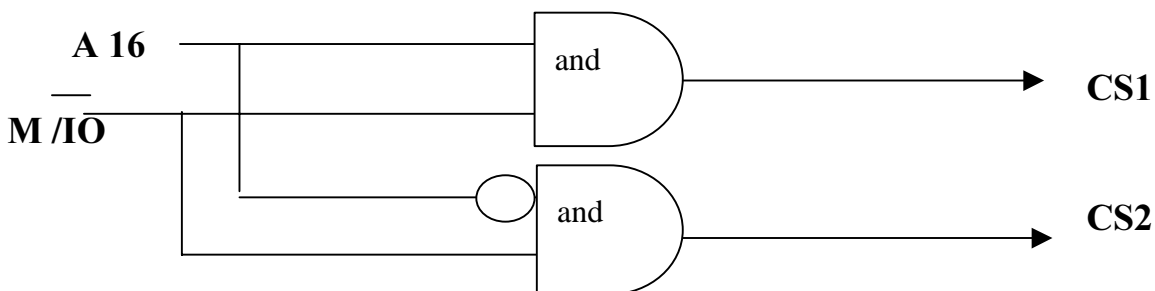
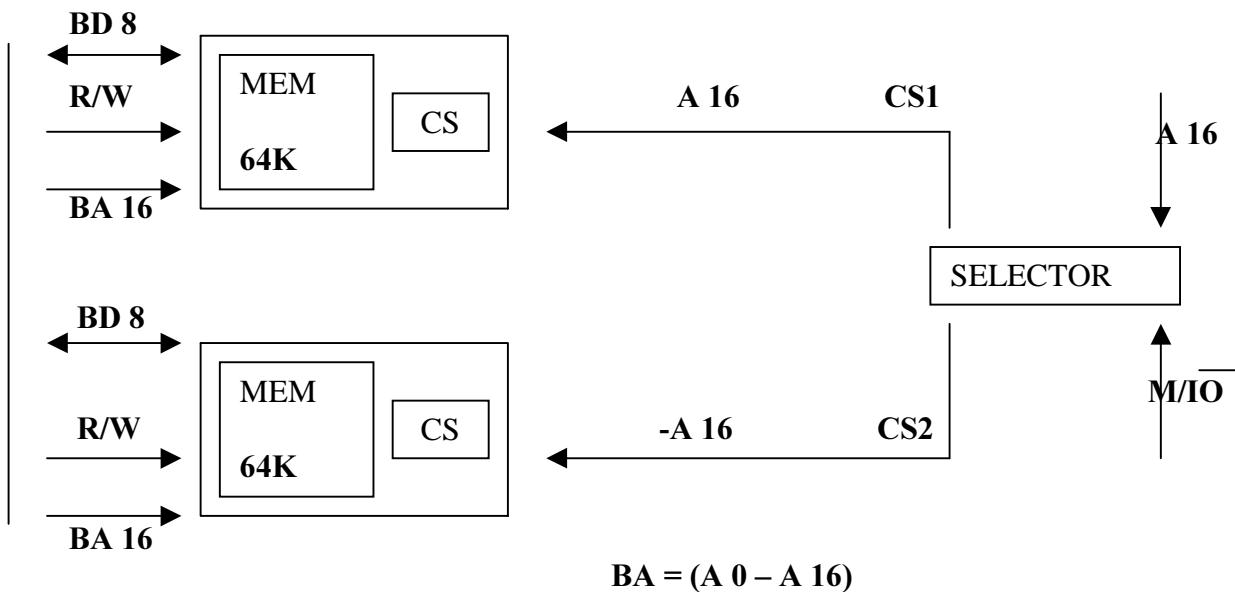
M/~IO	R/~W		INSTRUCTION
0	0	IN	READ <i>
0	1	OUT	WRITE <i>
1	0	MR	LOAD <i>
1	1	MW	SAVE <i>

Con <i> è possibile distinguere fra le diverse celle di memoria o fra i diversi dispositivi/canali di ingresso/uscita.

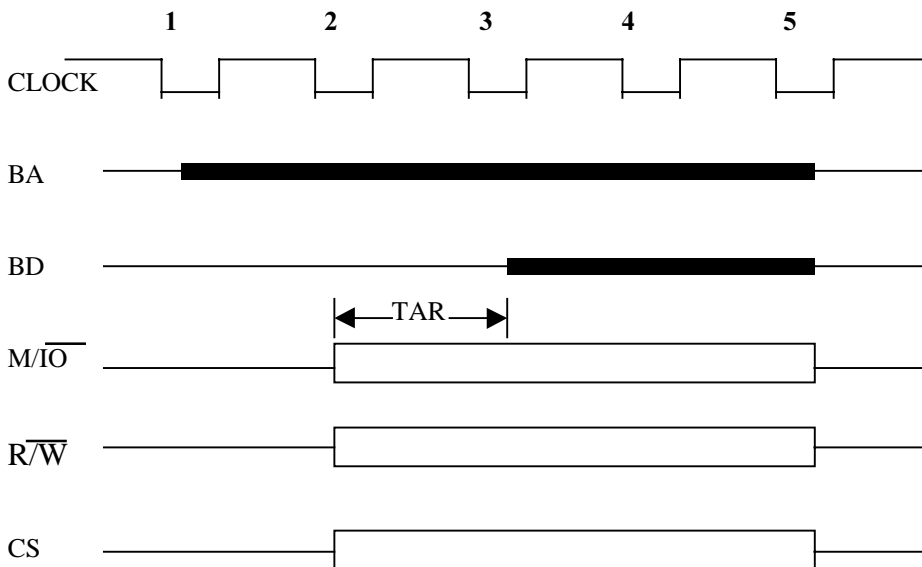
2.3.3 Banche di memoria, composizione di memorie

In quanto le memorie sono organizzate a banche, è indispensabile, per il computer, conoscere in quale zona della memoria andare a leggere o scrivere. Così, tramite l'indirizzo della cella viene prodotto un comando di selezione (selettore), il **CHIP SELECT** (CS) che identifica la zona di memoria dove si trova tale cella. CS e R/W vanno a formare il CONTROL BUS. Le informazioni che devono essere immesse, o prelevate dalla memoria, passano poi sul BUS DATI. IL CS viene generato a partire dai segnali M/IO ed in base all'indirizzo.

In figura e' riportato l'esempio di composizione di una memoria da 128 Kbytes tramite il collegamento di due blocchi da 64 Kbytes cadauno. In un blocco troviamo le codifiche in binario delle informazioni con il primo bit uguale a zero, nell'altro con il primo bit uguale a uno. Grazie a quella differenza il selettore attiva il CS dell'uno o dell'altro.



2.3.4 Ciclo di lettura



All'interno dell'elaboratore le varie operazioni vengono temporizzate dal CLOCK che scandisce la base dei tempi ad una frequenza che può variare da un tipo di elaboratore all'altro, ma che resta costante all'interno del medesimo calcolatore.

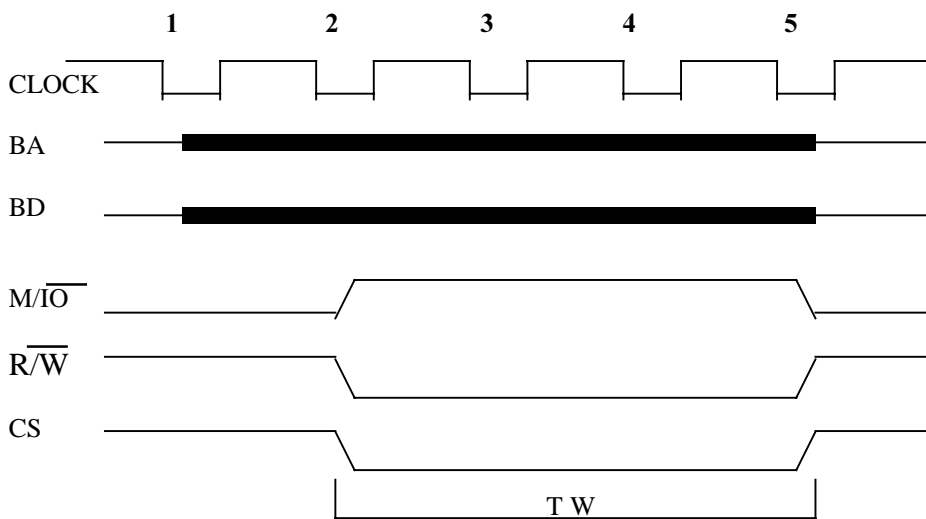
Il BA presenta un indirizzo. Dopo vengono attivati, contemporaneamente, il CS, per andare a cercare la giusta zona di memoria, e il R/W per attivare, in questo specifico esempio, la modalità di lettura. Adesso posso comunicare il dato.

Il tempo intercorso tra la presentazione dell'indirizzo e quella del dato viene definito TEMPO DI ACCESSO ALLA MEMORIA (TAR) che è, per la memoria centrale, nell'ordine di decine di nanosecondi. Il tempo di accesso di lettura e quello di scrittura sono usualmente diversi. Il secondo è tipicamente più lungo.

2.3.5 Ciclo di scrittura

Nella figura seguente è riportato il ciclo di scrittura di una CPU sulla memoria. Si noti come in questo caso il segnale di R/~W sia attivo basso e il tempo di scrittura TW. Tale tempo parte dal momento in cui viene fatta la selezione della cella con CS (se M/IO e R/W sono impostati correttamente) al momento nel quale la cella è effettivamente scritta. Il ciclo macchina deve durante un tempo sufficientemente lungo affinché questo accada.

In alcuni casi parla di funzionamento con degli stati di WAIT. Questi non sono altro che colpi di clock aggiuntivi che vengono inseriti nel ciclo macchina (prolungando la durata dei segnali di controllo) di un certo numero di colpi di clock per dare tempo alla memoria di fissare al suo interno il dato presente sul BD.



2.4 Le memorie di massa

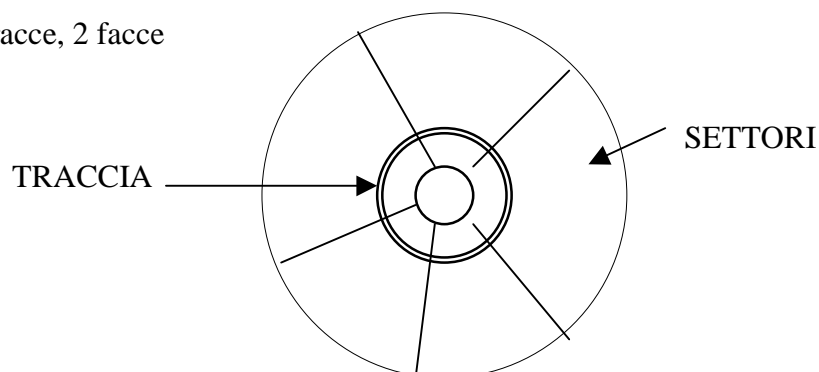
La memoria RAM è una memoria volatile, che si cancella quando spengo il calcolatore, perciò si utilizzano supporti di memoria sui quali io posso riversare e conservare il contenuto della memoria che altrimenti andrebbe ogni volta perduto. Questi supporti, chiamati memoria di massa, sono i floppy-disk, l'hard-disk, il cd-rom ed i nastri.

2.4.1 Floppy Disk

I floppy disk sono dischi flessibili di materiale plastico ricoperti di materiale magnetico come i nastri delle vostre musicassette. A scopo protettivo, sono contenuti in una custodia di materiale meccanicamente più resistente. Grazie a tale materiale vengono memorizzate le informazioni, per mezzo di magnetizzazioni positive e negative. Ognuno dei due lati del floppy-disk è diviso in settori, divisi a loro volta in tracce. I settori sono degli spicchi mentre le tracce sono concentriche. I dischi magnetici sono letti per tracce concentriche a differenza dei dischi di vinile che vengono letti seguendo una spirale.

La lettura dei dischi magnetici viene effettuata per mezzo di una testina di lettura che si muove in modo radiale al disco mentre il disco ruota a velocità costante. Pertanto, la lettura/scrittura delle singole tracce avviene in un tempo costante indipendentemente dalla distanza dal centro del disco.

Es. 20 settori, 80 tracce, 2 facce



Il tempo di accesso ad una traccia è in funzione della sua posizione sulla superficie del dischetto ed è mediamente di circa 25 millisecondi (l'accesso alla memoria RAM è di circa 30 nanosecondi).

Si ha un accesso diretto (quando questa passa sotto la testina) alla traccia e sequenziale all'interno della traccia. Una traccia tipicamente contiene 512 o 1024 byte. Il disco ha un foro per indicare quale è il settore zero.

Formati standard dei dischi magnetici, misurati in pollici, sono di 3"1/2; 5"1/4 (obsoleto); 8" (obsoleto).

I primi dischi da 8" contenevano circa 160 Kbyte e lavoravano solo su una delle due facce.

Attualmente il formato più diffuso è il 3"1/2 che ha una capacità di circa 1440 Kbyte.

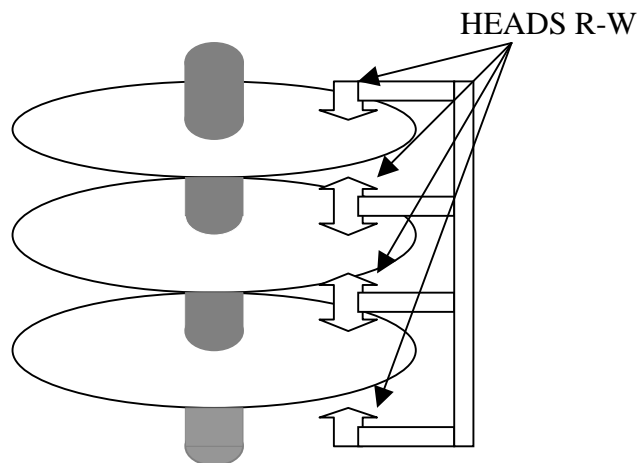
Vi sono anche quelli da 2 Mbyte, ma i più diffusi oggi sono quelli da 100-120 Mbyte e anche da 200 Mbyte.

2.4.2 Disco Rigido, Hard Disk

L'unità a disco fisso, il Disco Rigido, Hard Disk, si trova all'interno del computer ed è composto da dischi di materiale metallico cioè rigidi, generalmente d'alluminio, raccolti in una pila e sigillati all'interno di un contenitore dotato di un dispositivo di rotazione e di testine di lettura/scrittura.

Nell'hard disk, per trovare un'informazione ho bisogno della traccia, del settore, del lato/side e del disco sulla quale essa si trova. La sua capacità va da qualche decina a parecchie migliaia di Mbyte. Poiché i dischi sono di materiale rigido, essi sopportano velocità più alte ed il tempo di accesso per leggere un'informazione sull'hard disk è di circa 10 millisecondi. Le dimensioni tipiche sono di 10 Gbyte, cioè 10.000 Mbyte.

Head, traccia(cilindro), settore,



2.4.3 Compact Disk

Il CD è un disco di materiale plastico rivestito di materiale riflettente metallico e segnato da "microsolchi" grazie ai quali un dispositivo ottico (laser) legge le informazioni. Il CD ROM ha una vita più lunga dei supporti magnetici (nell'ordine di decenni contro pochi anni) ed un'elevata capacità (650 – 700 Mb), quelli da 72-74 minuti. Il CD si legge per piste concentriche come il floppy.

Il tempo di accesso è in funzione del driver in dotazione ma è dell'ordine di 2-10 volte quello di un'Hard

Disk. La velocità di lettura si misura con multipli della velocità di lettura dei CD Audio (170 kbytes/sec = 1x) . Oggi sono in commercio modelli in grado di leggere fino a 50 x.

Recentemente sono comparsi i CDR (CD Recordable). Questi sono dei CD che possono essere scritti (masterizzati) dal singolo utente con un apposito driver nel computer diverso da quello per la lettura dei CD. Tali dischi possono essere scritti anche in fasi successive fino al loro riempimento. Una volta scritti non possono essere cancellati. Le dimensioni in termini di Mbyte sono quelle di un CD.

Ancora più recenti sono i CDRW (CD ReWritable). Questi sono dei CD che possono essere scritti (masterizzati) e cancellati dal singolo utente con un apposito driver nel computer diverso da quello per la lettura dei CD e la scrittura del CDR. Tali dischi possono essere scritti anche in fasi successive fino al loro riempimento. Una volta scritti possono essere cancellati, con un'elevata risoluzione. Anche il singolo file può essere cancellato. Driver che possono essere utilizzati per produrre CDRW possono essere utilizzati anche per leggere CD e leggere e scrivere CDR. Le dimensioni in termini di Mbyte sono quelle di un CD.

L'ultima novità sono i DVD. Questi hanno dimensione in termini di Mbyte circa 10 volte maggiori. Per leggere e scrivere tali supporti sono necessari appositi driver. La scrittura è ancora un'operazione molto costosa.

2.4.4 Memoria a Nastro

Le memorie a nastro sono supporti magnetici solitamente utilizzati per copie di riserva (backup) in quanto il loro uso è molto più scomodo dei supporti di memoria precedentemente elencati; infatti il nastro non è una memoria ad accesso diretto ma ad accesso sequenziale: devo cioè svolgerlo ed avvolgerlo fino a trovare il dato che mi interessa (quindi il tempo di accesso può essere elevatissimo). Nei floppy disk, hard disk e CD ROM, invece, posso andare direttamente al dato a me utile (accesso diretto). La capacità dei nastri va da 60 Mbyte a 40 Gbyte circa.

2.5 Il Calcolatore e il Microprocessore

I calcolatori sono sistemi elettronici programmabili basati su circuiti integrati, collegati, con opportune interfacce, a organi di I/O. Il funzionamento è temporizzato da un Clock centrale, che scandisce i passi delle funzioni, comunque complesse, cui il microcalcolatore deve abitualmente assolvere, e che possono essere di tipo generale, come calcolo numerico, elaborazioni logiche, ecc.

I microprocessori fanno parte di un tipo di circuiti, generalmente indicati come processori o circuiti programmabili, nei quali la relazione di trasferimento ingresso-uscita, può essere variata senza intervenire sull'hardware del sistema. Per esempio, con modifiche dei collegamenti esterni: si passa da una funzione all'altra, semplicemente inviando istruzioni diverse al processore, secondo determinate procedure. Il microprocessore, usato per svolgere le funzioni tipiche di un'unità centrale (CPU, Central Processing Unit) di un sistema di calcolo, è un primo esempio di processore: Altri processori molto comuni sono: le porte parallele e seriali utilizzate per gestire le comunicazioni tra il microcalcolatore e il mondo esterno.

Un microprocessore può quindi definirsi come un dispositivo elettronico ad elevata integrazione, che può svolgere operazioni aritmetiche, logiche e di controllo, su dati generati dal microprocessore stesso o forniti dall'esterno. Il microprocessore per funzionare, deve essere inserito in una struttura in grado di

utilizzarne le potenzialità, fornendo al dispositivo, attraverso programmi definiti dall'utente, le informazioni necessarie, relative al tipo di operazione da eseguire ed i dati su cui operare. Il sistema così configurato prende il nome di microcalcolatore.

All'architettura tipica del microcalcolatore e' quella proposta da Von Neumann dal quale prende il nome "ARCHITETTURA DI VON NEUMANN". La sua caratteristica principale consiste nel memorizzare in un unico dispositivo, la memoria centrale, sia i codici operativi delle istruzioni sia i Dati.

Il calcolatore esegue sequenzialmente una serie di operazioni, che costituiscono il Programma. Il programma, dal punto di vista fisico, si presenta come una sequenza ordinata di numeri binari corrispondenti ai codici delle istruzioni e ai dati, collocati in memoria e richiamati dalla CPU al momento opportuno. La CPU è il cosiddetto "cuore del sistema". Oltre ad eseguire le varie operazioni previste dal programma, gestisce il flusso dei dati, sia all'interno del microcalcolatore sia da e verso l'esterno. All'interno del microprocessore si possono sempre distinguere due blocchi funzionali, il primo dedicato all'esecuzione delle operazioni logico-aritmetiche (ALU Aritmetic and Logic Unit) ed il secondo che determina l'operazione da eseguire, a partire dall'istruzione e dai dati presenti in ingresso (CU, Control Unit).

Le porte di input/output sono processori dedicati, indirizzati e controllati dalla CPU, che gestiscono, a loro volta, i collegamenti con il mondo esterno.

I microprocessori, e di conseguenza i microcalcolatori, si distinguono tra loro per la lunghezza di parola (word) corrispondente al numero di bit trattati in parallelo dal microprocessore stesso. (i termini seriale e parallelo fanno riferimento a come è trasmessa l'informazione, e precisamente, nel caso seriale viene trasmesso un bit per volta, mentre nel caso parallelo i bit trasmessi contemporaneamente sono in numero maggiore di uno).

Tale lunghezza di parola spesso corrisponde alla dimensione dei BUS dei dati del microcalcolatore.

Si definisce come bus un certo numero di conduttori su cui transitano informazioni dello stesso tipo, per esempio i bit di dati. Essi servono quindi per collegare la CPU con memorie e dispositivi di input/output. Accanto al bus dei dati viene realizzato il bus di controllo cioè una serie di conduttori che in linea di principio portano segnali di controllo: partono da un dispositivo del sistema (controller) e raggiungono tutti i dispositivi presenti sul bus (abilita la memoria a operare in lettura o scrittura, abilita i circuiti di interfaccia per consentire le operazioni di I/O). Attualmente esistono e sono correntemente usati microprocessori a 8, 16, 32 e 64 bit per cui la distinzione classica tra microcalcolatori da una parte, e minicalcolatori e mainframe dall'altra, basata sulla lunghezza di parola (micro corrisponde a 8 bit, mini a 16 e mainframe da 32 bit in poi) in generale non vale più e si deve pensare a definizioni basate sulle possibilità di accesso(singolo utente o multiutente) ed eventualmente sulla potenza di calcolo o sul costo.

2.6 Caratteristiche dei Microprocessori

I microprocessori si possono classificare e valutare sulla base di alcune caratteristiche legate all'hardware, i microprocessori si distinguono per:

- la lunghezza di parola
- velocità
- caratteristiche elettriche
- architettura.

La lunghezza di parola è definita come il numero di bit che il microprocessore tratta in parallelo. I microprocessori usati fino a pochi anni fa erano quelli a 8 bit: i campi di impiego erano microcalcolatori di impiego generale (basati su 8088, 8086 della Intel) (i PC IBM-compatibili). A partire dal 1984-1985 sono stati commercializzati prima microprocessori a 16 bit poi microprocessori a 32 bit. In un microprocessore, oltre alla lunghezza di parola di un dato, corrispondente alla precisione ottenibile nelle operazioni aritmetiche in precisione semplice, è importante il numero di bit corrispondenti agli indirizzi (address bus). Quest'ultima caratteristica definisce la dimensione massima della memoria indirizzabile direttamente dal microprocessore stesso (con microprocessori 80286, 80386 della Intel).

In questo caso, l'Intel e Motorola hanno fatto da padroni del mercato presentando microprocessori sempre più potenti e compatibili verso il basso. Questo significa che programmi in grado di lavorare con microprocessori vecchi della stessa famiglia possono lavorare (essere eseguiti) anche da quelli nuovi. Questo ha permesso l'evoluzione dei microprocessori senza bruciare il mercato. Pesante per esempio che una cosa analoga nel mondo delle auto significherebbe poter utilizzare sulle macchine di ultimo modello di una casa costruttrice gli accessori di quella prodotta 10 anni prima.

Tipicamente nei processori a 8 bit l'address bus è a 16 bit e consente di indirizzare 65536 locazioni di memoria cioè 64 kbyte, mentre nei microprocessori a 16 e a 32 bit il BA è rispettivamente di 20/24 bit (da 1 a 16 Mbyte indirizzabili) e di 32 bit (4 Gigabyte di memoria di base indirizzabile).

La velocità di un microprocessore corrisponde al tempo necessario per compiere certe operazioni. Per cercare di dare una valutazione quantitativa, si parte dalla constatazione che il microprocessore è una macchina sincrona, legata alla frequenza di Clock, per cui ogni istruzione richiede un tempo che non può essere qualsiasi, ma deve essere un multiplo intero del periodo del Clock. In particolare ogni istruzione viene realizzata come un insieme di operazioni elementari, tipicamente di lettura e scrittura della memoria o di dispositivi esterni, ognuno dei quali prevede un numero ben preciso di periodi o colpi di Clock.

In questo esempio si può vedere che per effettuare una lettura della memoria sono necessari tre colpi di Clock. Il fronte di salita di T_1 determina la presenza di un indirizzo valido sul bus degli indirizzi, che individua la locazione della memoria da leggere e rende valido il segnale di R/W (read/write), il quale a sua volta, avvisa la cella che la CPU intende fare una lettura. La memoria a questo punto, in relazione al suo tempo d'accesso, rende disponibile in un certo istante interno a T_2 , dati validi sul bus dei dati e infine la CPU li legge sul fronte di salita di T_3 e subito dopo rende non valido il segnale di RD. La temporizzazione dei microprocessori prevede un livello intermedio tra il periodo di Clock e il tempo richiesto per l'esecuzione di un'istruzione, il cosiddetto ciclo macchina. I cicli macchina corrispondono alle operazioni elementari che il processore può compiere; essi sono in numero limitato e le loro istruzioni sono realizzate combinando un numero variabile di cicli di Memoria. Nel caso dell'esempio precedente la CPU vuole leggere il contenuto di una certa locazione di memoria: tenuto conto che il microcalcolatore è un sistema sincrono, per cui le varie operazioni sono legate al Clock, la prima operazione è quella di fornire l'indirizzo di questa locazione in corrispondenza al fronte di salita del primo periodo di Clock. Ciò avviene con il primo tempo di ritardo da considerare, legato alle caratteristiche del microprocessore, tra il fronte di salita di T_1 e l'istante in cui gli indirizzi si sono stabilizzati e possono essere considerati validi sul bus. Successivamente occorre avvertire che il ciclo previsto è di lettura e ciò è fatto tramite il segnale RD che diventa valido con un ulteriore intervallo rispetto agli indirizzi, sempre nell'ambito di T_1 , da questo momento parte la considerazione del tempo di accesso della memoria considerata, cioè di un parametro non dipendente dalla CPU, che permette di determinare l'istante nel quale la memoria stessa mette sul bus dati le informazioni richieste dalla CPU. Una volta individuato questo istante, occorre verificare che sia compatibile con il tempo di "setup" della CPU, cioè con l'esigenza della CPU di avere presenti in ingresso i dati in anticipo rispetto al fronte di campionamento degli stessi. La CPU quindi acquisisce i dati sul

fronte di salita di T3 e con un certo ritardo rende non valido il segnale RD, concludendo il ciclo.

2.6.1 L'evoluzione dei Microprocessori.

Il mondo dei microprocessori si sta gradualmente diversificando. Negli ultimi anni si è assistito alla comparsa di microprocessori dedicati ad applicazioni industriali quali i microcontrollori discussi nei precedenti paragrafi. Più recentemente sono comparsi anche i DSP (Digital Signal Processor) per applicazioni di elaborazione del segnale mono e bi-dimensionale. Di recente sono comparsi anche microprocessori che possono essere collegati fra loro al fine di realizzare architetture multimicroprocessore. Dove i vari microprocessori non colloquiano fra loro tramite il BUS di sistema ma attraverso dei canali ad alta velocità realizzati appositamente per tale scopo. Esempi di questo tipo di microprocessori sono: i Transputer della Inmos/SGS che hanno 4 canali di comunicazione verso altri microprocessori dello stesso tipo, NCUBE della NCUBE che hanno 10 canali di comunicazione. Recentemente alcuni costruttori di microprocessori RISC hanno offerto la possibilità di realizzare architetture multi-RISC predisponendo i loro microprocessori per la connessione diretta con altri microprocessori RISC. Per esempio l'i860 della INTEL e lo Sparc della SUN Microsystems.

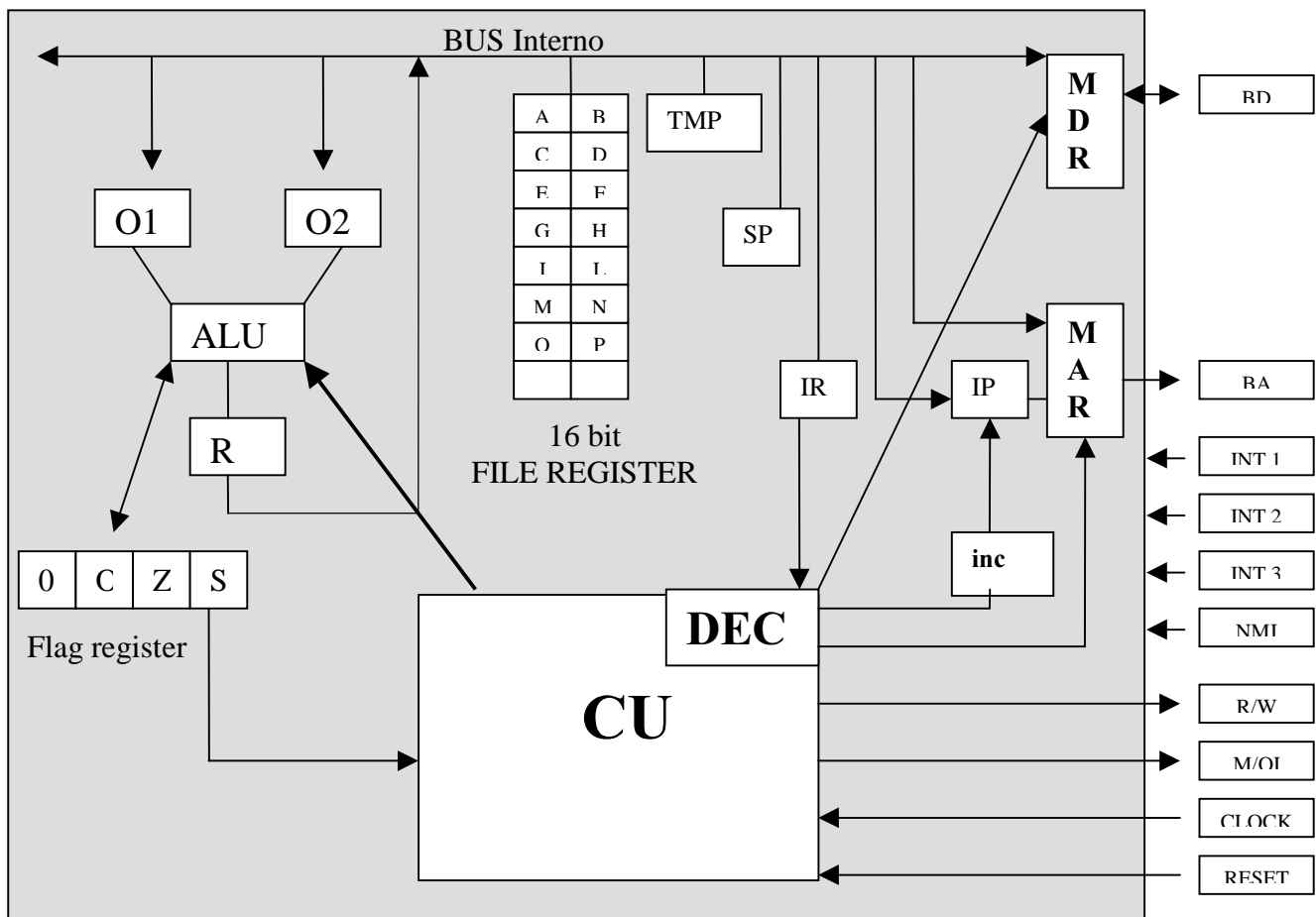
Parallelamente alla tendenza che ha portato a realizzare microprocessori specializzati per campi di applicazione specifici, come l'analisi del segnale e l'automazione, hanno subito uno sviluppo anche le architetture dei microprocessori di uso generale. Questi vengono utilizzati massimamente per la realizzazione di elaboratori elettronici.

2.6.2 I Microcontrollori

In molte applicazioni industriali il microprocessore è sempre affiancato da un certo numero di periferiche: porte di ingresso, porte di uscita, memorie, porte seriali, convertitori analogico-digitali, temporizzatori, contatori di eventi, controllori di interruzione, DMA, etc. Al fine di ridurre le dimensioni dei sistemi a microprocessore utilizzati in applicazioni industriali, e grazie alle attuali capacità d'integrazione, i costruttori sono giunti alla realizzazione di particolari tipi di microprocessori che vengono usualmente chiamati microcontrollori o single-chip. Questi circuiti integrati hanno al loro interno un microprocessore tradizionale ed un numero considerevole di altri dispositivi periferici che in elaboratori tradizionali sono posti in chip addizionali. L'utilizzo di tali microcontrollori per la realizzazione di un elaboratore per uso industriale conferisce maggiore compattezza al sistema e quindi anche maggiore affidabilità.

Usualmente le ditte costruttrici di microcontrollori: HITACHI, INTEL, Motorola, NATIONAL, Siemens, SGS, etc., non propongono una sola versione di un dato microcontrollore ma svariate. Queste differiscono fra loro per il numero e il tipo di periferiche che sono integrate all'interno del chip a partire da una struttura di base detta modello-base o centro (core). Pertanto quando si parla di microcontrollori è più corretto parlare in termini di "famiglia di microcontrollori" quando non si intende specificare la versione in modo preciso. Di particolare interesse sono i microcontrollori che sono forniti di un particolare tipo di memoria PROM che può essere programmata una sola volta dall'utente. Le versioni dei microcontrollori con questo tipo di memoria vengono dette OTP (One Time Programmable). I microcontrollori OTP sono particolarmente adatti per la realizzazione di controllori per applicazioni militari e di sicurezza.

3 La struttura interna del microprocessore



L'architettura di un microprocessore e' composta da svariati componenti alcuni dei quali sono stati gia' presentati in precedenza:

- CU = Control Unit, Unita' di controllo che riceve dall'esterno il segnale di Clock, interpreta l'istruzione corrente (contenuta nel registro IR) e comanda tutte le componenti interne della CPU. Nella CU e' contenuto anche un DEC (decoder) che permette di decodificare l'istruzione contenuta nel registro IR. La decodifica permette di passare da una serie di bit (rappresentazione binaria dell'istruzione) in una sequenza di operazioni interne per la CU scandite dal Clock.
- DEC = decoder delle istruzioni, si veda CU
- ALU = Arithmetic Logic Unit. Unita' interna per effettuare le operazioni aritmetiche e logiche. Ogni operazione viene effettuata leggendo i registri di ingresso/operandi O1, O2 e producendo un risultato. Questa operazione puo' modificare anche il valore dei registri a singolo bit contenuti nel FRAG REGISTER.
- CPU = Control Processing Unit, questo e' sostanzialmente il microprocessore stesso anche se in certi casi il microprocessore non contiene solo la CU e la ALU ma anche moltissime periferiche interne.
- IP = registro interno alla CPU che permette di puntare all'istruzione successiva (instruction pointer)

- IR = registro interno alla CPU che contiene l'istruzione successiva a quella che stiamo eseguendo, cioè l'istruzione corretta. Tale istruzione viene contenuta in codice macchina (instruction register). Il codice macchina è la rappresentazione binaria dell'istruzione. Per esempio un'istruzione di ADD potrebbe avere 01010010.
- BUS interno. BUS interno alla CPU che collega le sue varie componenti. In questo caso il BUS è ha numero di BIT pari ai BUS dati ed indirizzi, rispettivamente BD e BA.

Si notino i segnali di RESET, INT1, INT2, INT3, NMI che servono per influenzare dall'esterno il funzionamento della CPU:

- RESET: permette di far ripartire il microprocessore dalla prima istruzione che esegue al momento in cui gli viene data l'alimentazione. Questa istruzione è prevista essere ad una locazione fissa prossima alla fine della memoria. In questo caso la memoria è di 2^{16} locazioni.
- INT1, INT2, INT3 sono segnali esterni di interruzione che impongono al microprocessore di eseguire delle istruzioni che sono posizionate nella memoria a posizioni predefinite. Queste sono tipicamente verso la parte bassa della memoria. Se il microprocessore sta' effettuando delle operazioni, queste vengono interrotte per eseguire le operazioni assegnate a tali segnali di interruzione. Una volta completata l'esecuzione delle istruzioni associate all'interruzione eseguita il microprocessore ritorna a processare le istruzioni precedenti all'interruzione stessa. Queste operazioni di interruzione possono essere disabilitate per mezzo di specifiche istruzioni.
- NMI, segnale esterno di interruzione che non può essere disabilitato in alcun modo. Anche in questo caso, tale segnale provoca l'attivazione di un segmento di istruzioni locato ad una posizione predeterminata dal costruttore del microprocessore. Una volta completata l'esecuzione delle istruzioni associate all'interruzione eseguita il microprocessore ritorna a processare le istruzioni precedenti all'interruzione stessa.

Con le interruzioni, INT e NMI, si ha un vero e proprio cambio di contesto. Per contesto della CPU si intende lo stato della CPU, inteso come valore dei registri flag, dei registri di uso generale (file register), dell'IP e del IR.

Di seguito sono riportati maggiori dettagli sui registri interni al microprocessore.

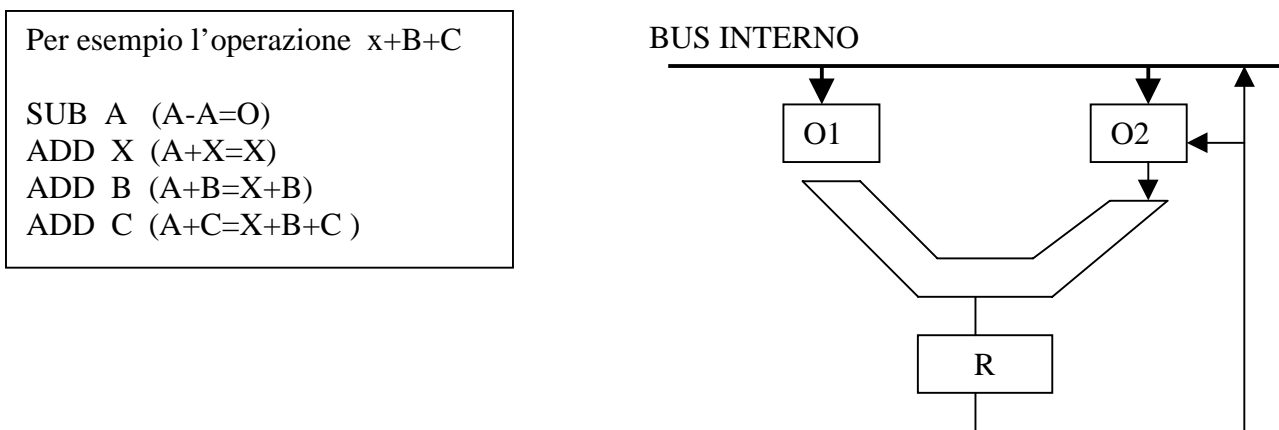
3.1 ALU

L'unità aritmetico logica è la parte del microprocessore che esegue le operazioni logiche e aritmetiche richieste dalle varie istruzioni, che possono agire su una o due operandi. Tali operandi devono essere posti in appositi registri (O1, O2) prima di effettuare l'operazione da parte della ALU.

Nella struttura con accumulatore, un'istruzione tipica è quella che somma un dato prelevato dalla memoria al contenuto di un particolare registro interno, l'accumulatore, memorizzando il risultato nell'accumulatore stesso. Supponendo di aver memorizzato in precedenza nell'accumulatore il primo addendo della somma, la sequenza di esecuzione dell'istruzione di somma in un microprocessore tipico prevede innanzitutto una lettura in memoria per caricare il codice operativo dell'istruzione, cioè la sequenza di bit, che decodificata e interpretata dall'unità di controllo, fa sì che all'interno del microprocessore vengono generati i segnali necessari all'esecuzione della particolare istruzione, in questo caso la somma. Dopo il caricamento del codice operativo in questo caso si ha un ulteriore accesso alla memoria per leggere il secondo addendo, che viene memorizzato in un registro interno e sommato al

contenuto dell'accumulatore: il risultato finale è infine memorizzato nell'accumulatore e resta a disposizione per ulteriori elaborazioni. E' chiaro che la ALU lavora a stretto contatto con l'unità di controllo (CU). La CU definisce l'operazione da eseguire tra tutte quelle possibili, mandando gli opportuni segnali in ingresso all'ALU che, a questo punto, esegue l'operazione agendo sull'operando contenuto nell'accumulatore (in un altro registro se l'operazione riguarda due operandi). Il risultato dell'operazione è infine immagazzinato nell'accumulatore, che così ha un ruolo del tutto particolare nel microprocessore tanto che spesso la sua architettura è detta "a singolo bus basata sull'accumulatore".

Con questo tipo di ALU le fasi per fare un'operazione sono più numerose.



Una versione più evoluta della ALU è quella che invece dell'accumulatore dispone di due registri che contengono i dati su cui operare e di un altro nel quale riporta il risultato dell'operazione; quest'ultimo, a sua volta è collegato al BUS DATI in modo che il dato che è contenuto in esso sia disponibile per qualsiasi operazione successiva (vedi schema architettura interna della CPU). Le operazioni previste dall'ALU sono tipicamente delle normali funzioni logiche (OR, AND, EX-OR, ecc.), la somma e la differenza aritmetica e le operazioni di rotazione e di shift di dati contenuti in un registro del microprocessore. Altre operazioni, come la moltiplicazione e la divisione aritmetica, che, specie nei microprocessori a 16 o a 32 bit, vengono definite con un'unica istruzione, sono eseguite di fatto con algoritmi programmati all'interno della CPU attraverso somme e differenze ripetute e quindi in tempi decisamente maggiori di quelli richiesti dalle operazioni aritmetiche più semplici.

L'unità logico aritmetica è tipicamente un blocco combinatorio, all'interno di una macchina sequenziale. Una volta che sia stata predisposta all'esecuzione di una certa operazione, questa viene eseguita immediatamente in un certo tempo legato solo ai ritardi introdotti dai circuiti logici interni del microprocessore. Il controllo del flusso dei dati è affidato all'unità di controllo che deve temporizzare correttamente il caricamento dei vari registri implicati nell'operazione.

3.2 Registri interni

Il numero e le istruzioni dei registri interni dei microprocessori (sono chiamati con le lettere dell'alfabeto) sono un'altra delle caratteristiche che differenziano i microprocessori gli uni dagli altri. Le situazioni che possono essere affrontate e gli obiettivi che possono essere perseguiti con migliore efficacia disponendo di opportuni registri interni sono numerosi: un primo esempio può essere la disponibilità di un numero maggiore di registri di uso generale, che consente di immagazzinare i dati all'interno della macchina,

riducendo la necessità di accesso alla memoria e conseguentemente i tempi di esecuzione dei programmi, in quanto la lettura e la scrittura di un registro interno è almeno dieci volte più veloce di un'analoga operazione sulla memoria esterna. I registri di uso generale e il bus interno hanno una dimensione pari alla lunghezza di una parola e il loro accesso è regolato dall'unità di controllo. Questi registri contengono i dati su cui deve operare l'ALU e i risultati intermedi e finali delle operazioni. Quando necessario, tramite istruzioni di programma, si effettua il trasferimento delle informazioni nei due sensi tra questi registri, la memoria RAM esterna e gli organi di I/O. Gli altri registri sono i cosiddetti registri dedicati, ognuno dei quali svolge una determinata funzione.

O1, O2, registri operandi

Registri di servizio temporanei che immagazzinano gli operandi, o l'unico operando, secondo le istruzioni su cui deve agire l'ALU.

R, registro risultato

Registro Contiene il risultato dell'operazione effettuata dalla ALU

IP, Instruction Pointer:

Il registro avente dimensioni pari a quelle del BA conserva istante per istante l'indirizzo della locazione di memoria corrispondente alla successiva istruzione del programma da eseguire, salvo che per le istruzioni di salto, è incrementato di uno dalla CU ogni volta che il microprocessore effettua una lettura di un dato del programma della memoria esterna. L'incremento viene guidato dalla CU attraverso il modulo di incremento, [inc].

MAR, MEMORY ADDRESS REGISTER

Registro interno alla CPU che si affaccia direttamente sul BA e contiene di volta in volta l'indirizzo di memoria cui fa riferimento in quel momento il microprocessore per il trasferimento dei dati, operazioni di lettura e scrittura con la memoria o con l'ingresso/uscita. Può quindi corrispondere al IP, per esempio quando viene caricato il codice dell'istruzione successiva, IR.

IR, INSTRUCTION REGISTER

Registro dell'istruzione, immagazzina l'istruzione corrente in formato binario, cioè il suo codice macchina. Il contenuto dell'IR è quindi utilizzato dall'unità di controllo per definire la sequenza dei segnali necessari per l'esecuzione della specifica istruzione. Esso contiene, quindi, l'istruzione che deve essere eseguita.

MDR, MEMORY DATA REGISTER

E' un registro di appoggio nel quale vengono caricate le parole lette dalla memoria oppure le parole pronte per essere scritte in una locazione di memoria: le parole lette possono essere decodificate, se corrispondenti al codice operativo dell'istruzione, o trasferite in un registro interno per la successiva elaborazione.

FLAG REGISTER

Il Flag Register e' un registro composto da bit singoli. Questi sono detti flag cioè bit che segnalano situazioni particolari cui è giunto il microprocessore nel corso dei calcoli. Esempi tipici corrispondenti ai flag più comunemente usati sono:

- **C (carry)**: segnala se in un'operazione aritmetica c'è stato un carry se il suo valore e' 1
- **Z (zero)**: segnala se in un'operazione l'accumulatore è andato a zero, se il suo valore e' 1.

$$(10000000) \text{ AND } (01111111) = 00000000 \quad z = 1$$

- **O** (overflow), che segnala che vi e' stato un overflow, cioè un riporto fuori dal bit di segno, se il suo valore e' 1.
- **S** (sign): segnala il valore del bit più significativo dell'accumulatore e quindi il segno. Se 1 e' negativo.

Si ricorda che un carry in operazioni con numeri rappresentati in complemento a 2 e' un riporto sul bit di segno mentre un overflow e' un riporto fuori dal bit di segno.

$$\begin{array}{r} \text{Riporto sul bit di segno :} \\ 0 \quad 1 \quad + \\ 0 \quad 1 \quad = \\ \hline 1 \quad 0 \end{array}$$

$$\begin{array}{r} \text{Riporto fuori dal bit} \\ \text{Di segno} \\ 1 \quad 0 \quad + \\ 1 \quad 1 \quad = \\ \hline 1 \quad 0 \quad 1 \end{array}$$

Le modifiche del FLAG REGISTER si hanno se per esempio si esegue l'istruzione ADD (SOMMA) che viene fatta nella ALU; Tutte le istruzioni aritmetiche possono dare luogo a variazioni nei flag come verra' mostrato in seguito.

Istruzioni come letture e scritture con la memoria o con l'Ingresso/uscita (READ, WRITE, LOAD, SAVE) non coinvolgono la ALU pertanto i bit di flag rimangono immutati.

FILE REGISTER

Un gruppo di 26 registri che si possono identificare con le lettere dell'alfabeto. Ogni registro e' a 16 bit. Con questi registri si possono fare operazioni con la ALU. Tali registri possono contenere anche indirizzi di memoria come si vede dalla presenza di un collegamento fra il BUS interno e il registro IP.

TMP:

Registro temporaneo su cui viene appoggiato un valore contenuto in una cella per poi prelevarlo e portarlo in un'altra cella (nel linguaggio ASSEMBLY viene usato per eseguire l'istruzione MOVE). Per esempio se si desidera copiare il valore del registro A del file register sul registro N si opera prima una copia in TMP da A ed in seguito una seconda copia da TMP a N.

SP, STACK POINTER

Registro che contiene un indirizzo di memoria che contiene le informazioni necessarie a ripristinare un precedente contesto. Il contesto viene salvato tutte le volte che si attiva un'interruzione e viene ricaricato quando questa termina. I meccanismi di salvataggio e caricamento del contesto vengono utilizzati anche quando si mettono in esecuzione delle procedure, come vedremo in seguito. Esempio tipico è quando il programma principale chiama un sottoprogramma alla fine della cui esecuzione deve essere ripresa l'esecuzione del programma principale, per cui occorre conservare il valore dell'IP al momento della chiamata al sottoprogramma e, eventualmente, il contenuto dei registri interni .

Infine in certi microprocessori sono previsti alcuni registri detti registri indice utilizzati per immagazzinare gli indirizzi di riferimento di aree di memoria, a partire dai quali è possibile assegnare in maniera semplificata gli indirizzi effettivi nei quali il programma deve andare a cercare i dati su cui

operare. Se si pensa di utilizzare un microprocessore a 8 bit con un bus degli indirizzi a 16 bit esso può gestire una mappa di memoria con indirizzi da (0000)E a (FFFF)E.

3.3 L'Unita' di Controllo

L' UNITA' DI CONTROLLO (CU) di un microprocessore ha il compito di riconoscere le istruzioni che di volta in volta devono essere eseguite e in generale la sequenza di segnali richiesta per l'esecuzione dell'istruzione stessa. La CU decodifica per ogni istruzione il contenuto dell'IR e, a seconda del risultato, si sincronizza con il clock per generare i segnali necessari per eseguire le varie operazioni sia all'interno della CPU che all'esterno. Così è la CU che genera i segnali esterni RD e WR per leggere e scrivere la memoria esterna e gli analoghi segnali interni necessari per caricare nei vari registri di lavoro i dati e/o per spostarsi da una locazione all'altra. Inoltre è la CU che incrementa il registro IP e provvede a definire caso per caso il contenuto del registro indirizzi, cioè la locazione di memoria cui deve fare riferimento la CPU per scrivere il risultato di un'operazione o per leggere l'informazione richiesta dall'istruzione in corso di esecuzione.

CLOCK: regola la temporizzazione di tutte le operazioni effettuate dalla CU. Esso è un circuito che genera degli impulsi a intervalli regolari a una certa frequenza espressa in MHZ. Si possono raggiungere i 600 MHZ (circa). Questi segnali abilitano le varie parti del sistema a operare così da coordinare nel tempo il loro errato.

3.4 Le Prestazioni dei Microprocessori

Le prestazioni di un elaboratore hanno una grande importanza nel determinare il valore effettivo della macchina considerata. Per avere validi parametri di riferimento occorre introdurre il concetto di temporizzazione. Tutte le operazioni effettuate dall'unità di controllo sono regolate da un circuito chiamato orologio (*clock*) che genera impulsi ad intervalli regolari operando ad una certa frequenza espressa in MegaHertz (Milioni di cicli). Oggi i personal computer ad esempio lavorano da 50 a 600 MHz; un processore che lavora a 50 MHz, avrà circa 50 colpi di clock al secondo. In alcuni casi il Clock viene prodotto ad una frequenza e questa viene moltiplicata o divisa per 2, 4, o 8 per essere utilizzata per temporizzare le operazioni interne, con la memoria e/o con le periferiche attraverso il BUS di sistema.

Introduciamo a questo punto uno dei parametri più usati per determinare le prestazioni di un elaboratore il *MIPS* acronimo di "Millions of Instruction per second", cioè milioni di istruzioni eseguite in un secondo. Si è stimato che l'esecuzione di un'istruzione macchina può richiedere al massimo una decina di colpi di clock. Con questo dato in mano, è facile dire ad esempio che una macchina tarata a 50 MHz lavorerà sull'ordine dei 5 MIPS.

A fianco del MIPS si posiziona un altro parametro di riferimento che però lavora in funzione delle operazioni in virgola mobile, le cosiddette floating point. Si ha infatti il *MFLOPS*, acronimo di "Millions of Floating Point Operations per second", cioè milioni di operazioni in virgola mobile al secondo. Molte volte questo tipo di operazioni è implementato o con opportune procedure in linguaggio macchina, o con dispositivi hardware quali ad esempio i coprocessori matematici

Naturalmente esistono altri parametri di riferimento per verificare la velocità operativa di un calcolatore, alcuni addirittura più precisi, come ad esempio lo SPEC, etc. Resta il fatto però che i primi due menzionati rimangono i più diffusi

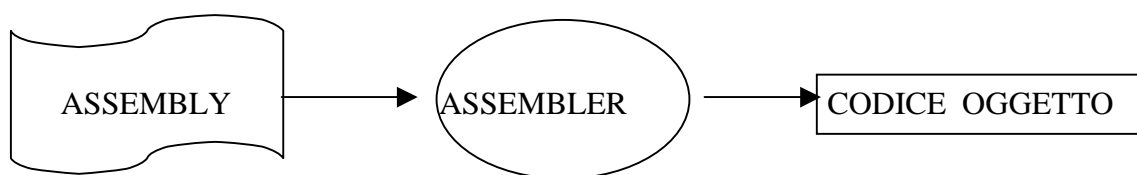
Per misurare i valori effettivi di un calcolatore, è necessario eseguire un programma che rientra nella categoria dei *BENCHMARK*, strumenti ideati appositamente per questa funzione. Molte volte un buon Benchmark dà una quantità di informazioni sul proprio sistema estremamente esauriente, che certo non si limita a MIPS e MFLOPS.

A puro titolo informativo, se un personal computer è capace di raggiungere mediamente i 10 MIPS e 2 MFLOPS, una workstation può elevare queste prestazioni a dieci volte tanto, questi dati sono relativi al 1998.

4 Il Linguaggio Assembly

Ogni microprocessore e' in grado di eseguire un certo numero di istruzioni, cioè può eseguire un numero più o meno grande di operazioni elementari. Un programma è costituito da una sequenza di tali istruzioni che permette al microprocessore di assolvere ad un determinato compito di calcolo e/o di controllo. Le istruzioni che il microprocessore deve leggere ed eseguire sono naturalmente immagazzinate nella memoria in forma di codice binario ovvero sono espresse in quello che si chiama "**LINGUAGGIOMACCHINA**".

Ogni istruzione costituita da un certo numero di byte e il programma nel suo insieme è una successione di byte che va ad occupare una certa porzione di memoria. Redigere un programma direttamente in codice binario non è per nulla agevole ed è suscettibile di errori; anche la possibilità di usare al posto del codice binario il corrispondente codice esadecimale non è di molto aiuto. L'uso del linguaggio assembly consente di superare le difficoltà dette con l'adozione di una forma simbolica (codice mnemonico) che richiama con una notazione sintetica il modo di operare di ogni istruzione. Tale linguaggio è pertanto già orientato verso l'uomo, che lo può usare più agevolmente del linguaggio macchina e tuttavia di quest'ultimo conserva tutti i vantaggi di sintesi e di capacità di esecuzione, in quanto a ogni istruzione in linguaggio Assembly corrisponde una sola istruzione in linguaggio macchina. E' il caso di sottolineare che, al contrario, i linguaggi evoluti (il linguaggio Assembly è a basso livello in quanto dipende dalle caratteristiche dell'hardware) suddetta corrispondenza non esiste. La stesura di un linguaggio complesso è tuttavia più facile in un linguaggio evoluto che in linguaggio Assembly. Questa proprietà si paga con una minore velocità di esecuzione e di un'occupazione e gestione della memoria meno economica. La corrispondenza uno a uno fra istruzione di linguaggio Assembly e istruzione in linguaggio macchina vieta la possibilità di un unico linguaggio Assembly che, pertanto, è diverso da microprocessore a microprocessore. Per essere eseguito, un programma in linguaggio Assembly deve essere tradotto in linguaggio macchina: per un linguaggio assembler si devono seguire generalmente quattro passi i quali sono comunque suscettibili di diversa attuazione a seconda del tipo di software disponibile.



Il codice sorgente di un programma in Assembly è un file di testo, cioè un insieme di caratteri ASCII. Successivamente **l'Assembler (assemblatore)** si occupa di tradurre il file sorgente in un file oggetto, ovvero in un file espresso/codificato in linguaggio macchina. Il file prodotto dall'assemblatore viene poi trattato dal programma detto Linker che fornisce un file effettivamente eseguibile dalla CPU. L'ultimo passo del processo consiste nel caricare e far eseguire il file creato dal linker.

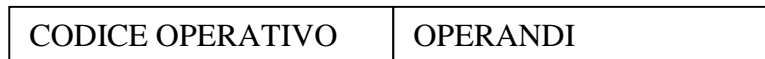
LE PRINCIPALI ISTRUZIONI DEL LINGUAGGIO ASSEMBLY SONO :

LOAD, SAVE,
READ, WRITE
SQR

AND, OR, NOT
SHL, SHR
MOVE
MUL, DIV, ADD, SUB
JMP, JNZ
IE, ID, SAVECNT, LOADCNT

Un **linguaggio macchina** è una forma di codice direttamente eseguibile dall'hardware dell'elaboratore. Il linguaggio macchina è composto da tante istruzioni macchina che sono codificate in forma binaria. Queste istruzioni essendo correlate direttamente all'architettura variano a seconda dell'elaboratore. Affinché si possa eseguire un programma per la risoluzione di un dato algoritmo, le istruzioni devono essere allocate in celle contigue, in base ad indirizzi crescenti. Si può tenere conto di eventuali organizzazioni non lineari tramite delle istruzioni di controllo che verranno mostrate in seguito.

Un'istruzione può essere rappresentata graficamente secondo questo schema generale:



Il **codice operativo** specifica l'operazione da compiere. Se si hanno N bit allora è possibile identificare 2^N diverse istruzioni.

Gli **operandi** specificano le locazioni delle celle di memoria cui ciascuna operazione si riferisce. Il codice operativo è sempre presente; mentre gli operandi possono mancare. La lunghezza di un'istruzione varia a seconda del tipo di elaboratore. La lunghezza può variare anche da istruzione a istruzione.

4.1 Tipi di Istruzioni

Le istruzioni a seconda della funzione che svolgono si classificano in quattro gruppi.

Istruzioni aritmetico-logiche Fanno parte di questo gruppo le istruzioni aritmetiche, logiche o in generale tutte le istruzioni che effettuano manipolazioni sui dati. Esse devono specificare i dati su cui devono essere compiute le operazioni e dove depositare il risultato.

Istruzioni di salto (JMP) Servono per alterare l'esecuzione sequenziale di un programma. Il salto può essere incondizionato o condizionato. Nel primo caso è specificato l'indirizzo di memoria in cui si trova la successiva istruzione da eseguire. Nel secondo caso è specificato anche una condizione necessaria perché il salto avvenga.

Istruzioni di ingresso-uscita (READ,WRITE) Adempiono la funzione di trasferimento dati all'interno o all'esterno dell'elaboratore. Esse indicano sia l'unità periferica che deve essere utilizzata sia dove si trova il dato che deve essere emesso all'esterno (oppure dove deve essere depositato il dato immesso all'interno).

Istruzioni per il controllo del contesto: questo tipo di istruzioni permettono di gestire le interruzioni e di salvare e ripristinare il contesto del microprocessore.

UN PROGRAMMA in linguaggio macchina utilizza codici operativi e indirizzi codificati in forma binaria. Queste caratteristiche rendono molto complicata la scrittura, la lettura, e la comprensione di un programma in linguaggio macchina.

Per venire incontro a questi problemi, esistono linguaggi di programmazione **ad ALTO LIVELLO** che consentono la descrizione degli algoritmi in modo indipendente da specifiche caratteristiche hardware. Tramite i linguaggi ad alto livello il programmatore astrae da una serie di dettagli e si concentra sulla produzione di programmi leggibili nonostante il codice eseguito sia pressoché sempre scritto in linguaggio macchina. Spesso è opportuno programmare in un linguaggio che sia molto vicino al linguaggio macchina poiché i programmi prodotti traducendo linguaggi di alto livello possono essere poco efficienti rispetto ai programmi scritti direttamente in linguaggio macchina.

Per questo motivo ogni processore dispone di un linguaggio assembler o Assembly che viene anche detto linguaggio di **basso livello**. Il linguaggio assembler rispecchia l'architettura hardware dell'elaboratore a cui si riferisce, quindi la descrizione di un algoritmo deve essere fatta in riferimento alle modalità operative dell'elaboratore.

Il codice operativo può essere indicato con un nome mnemonico e alle voci di memoria possono essere associati dei nomi da usare all'interno delle istruzioni al posto dei loro indirizzi. All'interno di un programma in Assembly è possibile inserire commenti e frasi relative alla struttura del programma stesso.

Un programma scritto in linguaggio assembler, prima di poter essere eseguito deve essere tradotto in linguaggio macchina: questa funzione è svolta da un traduttore chiamato **assemblatore**. Esso esamina tutte le frasi del programma in Assembly (programma sorgente) e produce un programma in linguaggio macchina (programma oggetto). L'assemblatore o un suo strumento di supporto stabilisce l'assegnazione di memoria al programma, associando ad ogni etichetta l'indirizzo della voce di memoria corrispondente. Tali indirizzamenti sono in effetti relativi alla posizione di memoria della prima istruzione del programma. Al momento dell'esecuzione, quando il programma viene caricato in memoria viene effettuata la vera e propria assegnazione delle istruzioni alla memoria.

4.2 Le Istruzioni

Il funzionamento dell'elaboratore consiste nella lettura ed esecuzione delle istruzioni dei programmi e avviene ripetendo una sequenza di operazioni nell'unità centrale. L'esecuzione di ogni istruzione richiede lo svolgimento di tre fasi: l'acquisizione dalla memoria centrale, l'interpretazione e l'esecuzione vera e propria. La fase di acquisizione richiede l'esecuzione di alcune microistruzioni ciascuna delle quali corrisponde a un trasferimento di dati fra i registri dell'unità centrale oppure tra le unità funzionali collegate dal bus.

4.2.1 Istruzione LOAD

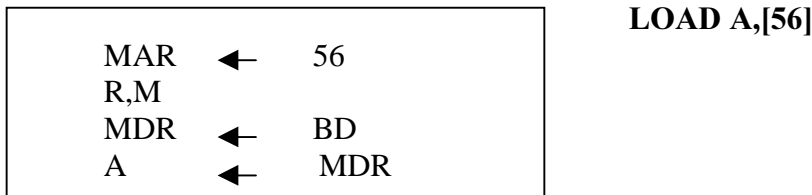
LOAD (caricamento) : operazione di trasferimento di un dato dalla memoria

Per esempio. LOAD A, [B] ← indirizzo di memoria
 Nome registro

LOAD A,B,C sintatticamente errato
 LOAD [A],B sintatticamente errato
 LOAD B, [A] SINTATTICAMENTE CORRETTO

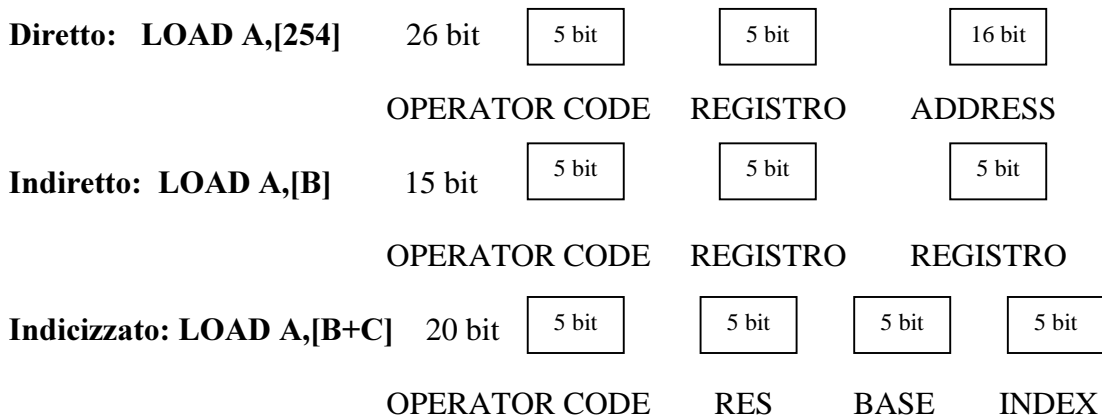
l'operazione LOAD vuole un registro/virgola/indirizzo.

Micro istruzioni dell'operatore "LOAD"

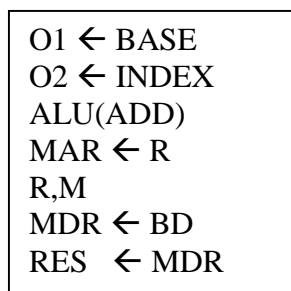


L'indirizzo di memoria 56 viene caricato nel MAR (registro indirizzi) e passando per il BA (BUS ADDRESS) viene definita l'operazione di lettura in memoria. Dal BD (BUS DATI) si trasferisce l'informazione al registro interno MDR ; da MDR i dati vengono messi nel registro A.

Codice per l'istruzione LOAD :



• **ESEMPIO :**



RES = registro destinazione

L'istruzione LOAD non modifica i FLAG.

4.2.2 Istruzione SAVE

SAVE: operazione di salvataggio in memoria di un dato da un registro.

ES: SAVE [A] , C

 ↑ ↑

 indirizzo di memoria nome registro

Microistruzioni dell'operatore SAVE

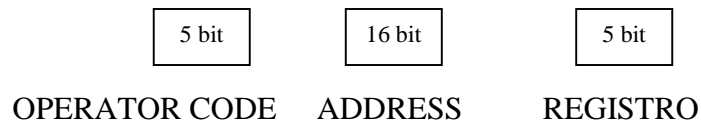
SAVE [A] , C

MAR ← A
MDR ← C
W,M

L'operatore utilizza 5 bit per il OPERATOR CODE, 5 bit per l'identificazione del registro C, e 16 bit per l'indirizzo A, per un totale di **26 bit**.

Codice per l'istruzione SAVE :

SAVE [A] , C



L'istruzione LOAD non modifica i FLAG.

4.2.3 Istruzione ADD

ADD: operazione di addizione fra due membri eseguita dalla ALU (Aritmetic Logic Unit).

ES: ADD C , A , B che significa C = A + B

 ↑ ↑

Registri conteneti gli addendi
indirizzo di memoria destinato al risultato

Microistruzioni dell'operatore ADD

ADD C , A , B

O1	←	A
O2	←	B
ADD (ALU)		
C	←	R

L'operatore utilizza 5 bit per il OPERATOR CODE, 5 bit per l'identificazione del registro A, 5 bit per il registro B, ed altri 5 bit per il registro C, per un totale di **20 bit**.

ES: calcolo della somma: $M[VC]=M[VA]+M[VB]$

Operando da memoria a memoria. Prendendo i dati dalla memoria e riportando il risultato in memoria.

LOAD A,[VA]; carica da VA al registro A
LOAD B,[VB]; carica da VB al registro B
ADD C,A,B; somma A e B e mette il risultato in C
SAVE [VC],C; salva nella cella VC il contenuto del registro C

Codice per l'istruzione ADD :



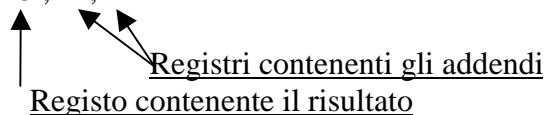
L'istruzione ADD modifica i FLAG: C, Z, O, S.

4.2.4 Istruzione SUB

SUB : operazione di sottrazione fra due membri ; viene eseguita dalla ALU.

$$C = A - B$$

Es: SUB C , A , B



Microistruzioni dell'operatore SUB

SUB C , A , B

O1	←	A
O2	←	B
SUB (ALU -)		
C	←	R

Dal registro A in O1; dal registro B in O2; viene eseguita la sottrazione tra O1 e O2, il risultato R viene memorizzato nel registro C.

L'operatore utilizza 5 bit per il OPERATOR CODE, 5 bit per l'identificazione del registro A, altri 5 bit per il B, ed altri 5 bit per C, per un totale di **20 bit**.

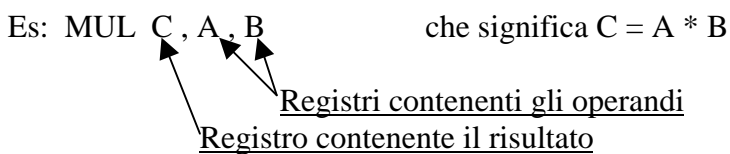
Codice per l'istruzione SUB :



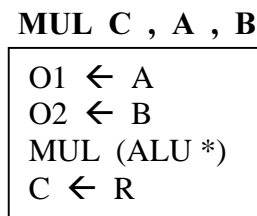
L'istruzione SUB modifica i FLAG: C, Z, O, S.

4.2.5 Istruzione MUL

MUL : operazione di moltiplicazione tra due membri; viene eseguita dalla ALU.



Microistruzioni dell 'operatore MUL



Dal registro A in O1; dal registro B in O2; viene eseguita la moltiplicazione ed il risultato viene memorizzato nel registro C.

L'operatore utilizza 5 bit per il OPERATOR CODE, 5 bit per l'identificazione del registro A, altri 5 bit per il B, ed altri 5 bit per C, per un totale di **20 bit**.

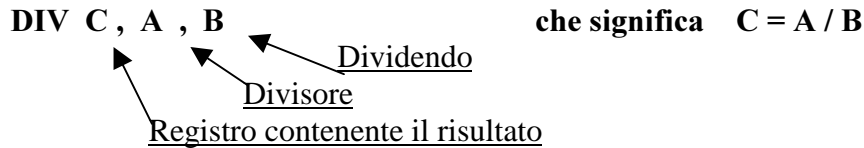
Codice per l'istruzione MUL :



L'istruzione modifica i FLAG: C, Z, O, S.

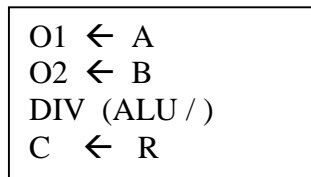
4.2.6 Istruzione DIV

DIV: operatore di divisione tra due membri eseguita dalla ALU.



Microistruzioni dell'operatore DIV

DIV C , A , B



Vengono copiate le informazioni dai registri A , B rispettivamente in O1 e O2 ; viene eseguita la divisione ed il risultato è copiato nel registro C.

L'operatore utilizza 5 bit per il OPERATOR CODE, 5 bit per l'identificazione del registro A, altri 5 bit per il B, ed altri 5 bit per C, per un totale di **20 bit**.

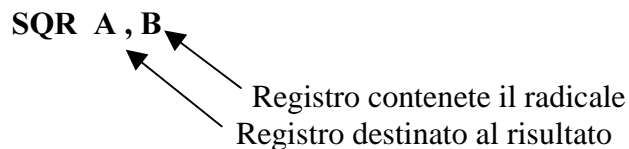
Codice per l'istruzione DIV :



L'istruzione modifica i FLAG: C, Z, O, S.

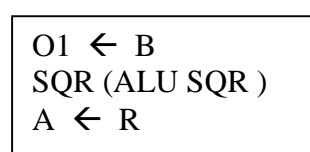
4.2.7 Istruzione SQR

SQR : operazione di estrazione della radice quadrata.



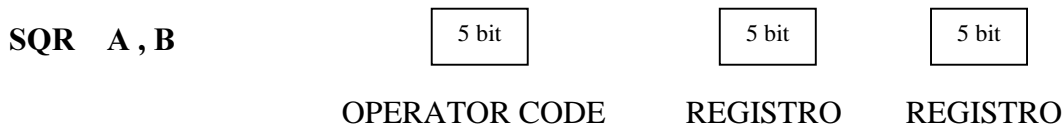
Microistruzioni dell'operazione SQR:

SQR A , B



L'operatore utilizza 5 bit per il OPERATOR CODE, 5 bit per l'identificazione del registro A, ed altri 5 bit per il registro di destinazione, per un totale di **15 bit**.

Codice per l'istruzione SQR :

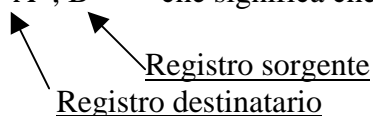


L'istruzione modifica i FLAG: C, Z, O, S.

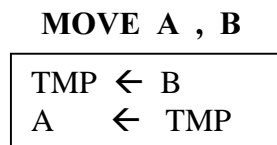
4.2.8 Istruzione MOVE

MOVE : operazione di trasferimento del contenuto di un registro in un altro.

Es: MOVE A , B che significa che copio il contenuto di B in A



Micro istruzioni dell'operazione MOVE



L'operatore utilizza 5 bit per il OPERATOR CODE, 5 bit per l'identificazione del registro A, ed altri 5 bit per il B, per un totale di **15 bit**.

Codice per l'istruzione MOVE :



L'istruzione non modifica i FLAG.

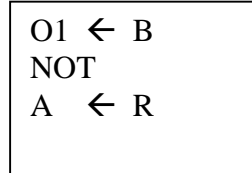
4.2.9 Istruzione NOT

NOT : istruzione che agisce su un solo parametro ,serve a negare il contenuto di un registro.

Es. NOT A , B ; nega il contenuto del registro B e lo memorizza nel registro A.

Microistruzioni dell'operatore NOT

NOT A , B



Dal registro B l'informazione viene copiata in O1, negata ed il risultato viene memorizzato in A. L'operatore utilizza 5 bit per il OPERATOR CODE, 5 bit per l'identificazione del registro A, ed altri 5 bit per il B, per un totale di **15 bit**.

Codice per l'istruzione NOT :



L'istruzione modifica i FLAG: Z, S.

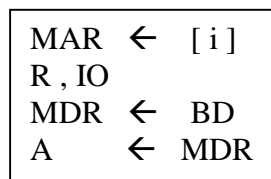
4.2.10 Istruzione READ

READ: operazione equivalente a LOAD, che agisce però operando in Input/Output anziché in memoria.

Es. READ A , [i] ; il dato contenuto in 'i' viene copiato nel registro A

Microistruzioni dell'operatore READ

READ A , [32]



L'indirizzo di memoria 32 viene caricato nel registro indirizzi MAR si definisce l'operazione di lettura, dal bus dati (BD) l'informazione è portata all'MDR, dall'MDR al registro interno A. L'operatore utilizza 5 bit per il OPERATOR CODE, 5 bit per l'identificazione del registro A, e 16 bit per l'indirizzo [i], per un totale di **26 bit**.

Codice per l'istruzione READ :



L'istruzione non modifica i FLAG.

4.2.11 Istruzione WRITE

WRITE : istruzione equivalente a SAVE che però opera in Input/Output anziché in memoria.

Es: WRITE [i] , B ; il dato contenuto nel registro B viene salvato in [i].

Microistruzioni dell'operatore WRITE

WRITE [32] , A

MAR	←	[i]
MDR	←	B
W , IO		
BD	←	MDR

L'indirizzo [i] è caricato nel MAR, è definita l'operazione di scrittura; il dato da B è portato sull'MDR, dall'MDR al bus dati e quindi in IO.

L'operatore utilizza 5 bit per il OPERATOR CODE, 5 bit per l'identificazione del registro B, e 16 bit per l'indirizzo [i], per un totale di **26 bit**.

Codice per l'istruzione WRITE :

WRITE [C] , A

5 bit

16 bit

5 bit

OPERATOR CODE

ADDRESS

REGISTRO

Anche per questi valgono i modi di indirizzamento dell'istruzione LOAD.

L'istruzione non modifica i FLAG.

4.2.12 Istruzione AND

AND : operatore logico AND.

Es. AND C , A , B che significa $C = A \text{ and } B$

Microistruzioni dell'operazione AND

AND C , A , B

O1	←	A
O2	←	B
AND		
C	←	R

L'operatore utilizza 5 bit per il OPERATOR CODE, 5 bit per l'identificazione del registro A, 5 bit per il B, ed altri 5 bit per C, per un totale di **20 bit**.

Codice per l'istruzione AND :



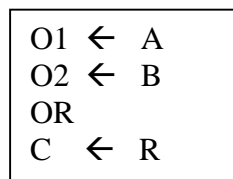
L'istruzione modifica i FLAG: Z, S.

4.2.13 Istruzione OR

OR : operazione logica OR. C = A or B

Microistruzioni dell'operazione OR

OR C , A , B



Codice per l'istruzione OR:



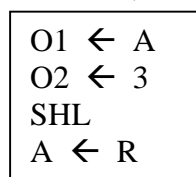
L'istruzione modifica i FLAG: Z, S.

4.2.14 Istruzione SHL

SHL : (shift left) spostato a sinistra di N posizioni/bit.

Microistruzioni dell'operazione SHL

SHL A , 3



L'operatore utilizza 5 bit per il OPERATOR CODE, 5 bit per l'identificazione del registro A, e 4 bit per

l'intero , per un totale di **14 bit**.

Codice per l'istruzione SHL :



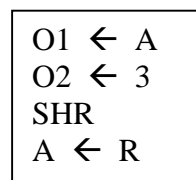
L'istruzione modifica i FLAG: Z, S.

4.2.15 Istruzione SHR

SHR : (shift right) spostato a destra di N posizioni.

Microistruzioni dell'operazione SHR

SHR A , 3



L'operatore utilizza 5 bit per il OPERATOR CODE, 5 bit per l'identificazione del registro A, e 4 bit per l'intero , per un totale di **14 bit**.

Codice per l'istruzione SHR :



L'istruzione modifica i FLAG: Z, S.

4.3 Le Istruzioni di Salto

Sono un particolare tipo di comandi che permettono al microprocessore di cambiare l'Instruction Pointer . Con un'istruzione di salto una parte di codice può essere eseguita prima di un'altra alterando la normale successione delle istruzioni .

Le istruzioni di salto più frequentemente usate sono: JMP, JZ, JNZ, JO, JNO, JC, JNC, JS, JNS..

4.3.1 Salto incondizionato

JMP : quando durante l'esecuzione del codice , si arriva ad un JMP l'istruzione che sarà eseguita è quella indicata dall'etichetta (LABEL) del jump.

Es. `JMP L1` ; il programma esegue l'istruzione identificata da Label 1

Es: Parte di codice

```
LOAD ...
SAVE ...
ADD ...
MUL ...
JMP L1 ; da qui salta alla label L1 ,cioè l'istruzione eseguita sarà SUB
ADD     ....
MUL     ....
NOT     ....
L1 : SUB ....
LOAD   ....
```

Microistruzioni dell'operazione JMP

JMP L1

IP ← L1

Viene caricato nell'IP (Instruction Pointer) l'indirizzo di SUB (1) invece che ADD (2) .

L'operatore utilizza 5 bit per il OPERATOR CODE, 16 bit per l'identificazione della Label (è un indirizzo), per un totale di **21 bit**.

Codice per l'istruzione JMP:

JMP L1

5 bit

16 bit

OPERATOR CODE

ADDRESS

L'istruzione non modifica i FLAG.

4.3.2 Salti condizionati

Sono istruzioni di salto la cui attivazione dipende da un condizione logica. Le condizioni logiche sono quelle evidenziate dallo stato dei bit del FLAG register.

- **LZ**: salta se il risultato dell'operazione precedente è zero
- **JNZ** : salta se il risultato dell'operazione precedente non è zero
- **JO** : salta se si verifica un overflow.
- **JNO** : salta se non si verifica un overflow.
- **JC** : salta se si verifica un carry
- **JNC** : salta se non si verifica un carry.
- **JS** : salta se si ha segno negativo
- **JNS** : salta se non si ha un segno negativo

L'istruzione di JMP ha bisogno delle LABEL cioè di etichette che hanno la funzione di indicare l'indirizzo dell'istruzione successiva da svolgere. Se scrivo JMP L1 prima di tutto cambia l'Instruction Pointer (IP) che fino ad ora conteneva l'indirizzo dell'istruzione che compare di seguito al JMP ma che dovrà contenere l'indirizzo dell'istruzione che compare di fianco all'etichetta.

Oppure sono possibili anche salti con distanza fissa cioè si hanno delle istruzioni del tipo JMP

Le istruzioni di JMP sono necessarie per poter alterare l'esecuzione di un programma. Esse si suddividono in istruzioni di salto incondizionato o di salto condizionato. Nel caso di SALTO INCONDIZIONATO l'istruzione specifica l'indirizzo di memoria in cui si trova la successiva istruzione da eseguire : ecco perché bisogna scrivere nel registro contatore di programma (PC) l'indirizzo specificato dall'istruzione. Nel caso di SALTO CONDIZIONATO l'istruzione deve anche specificare la condizione che deve verificarsi affinché il salto abbia luogo. Tutte le volte che un'istruzione fa riferimento ad informazioni contenute nella memoria centrale, essa deve specificare l'indirizzo di memoria in cui queste sono contenute.

JNZ L1

IF NOT Z THEN IP ← L1

Queste istruzioni non modificano i FLAG.

4.4 Le Istruzioni per la gestione del Contesto

Queste istruzioni permettono di controllare i cambi di contesto:

- **ID** disattiva la reattività della CPU agli stimoli esterni che possono arrivare dall'attivazione dei segnali INT1, INT2, INT3, INT4
- **IE** Attiva la reattività della CPU agli stimoli esterni che possono arrivare dall'attivazione dei segnali INT1, INT2, INT3, INT4
- **SAVECNT**. Permette di salvare il contesto della CPU in memoria. Valore dei registri del file register, del flag register dell'IP e del IR. Tutte le volte che si opera una SAVECNT nello SP viene posto l'indirizzo dove tali informazioni sono state messe in memoria.
- **LOADCNT**. Permette di caricare l'ultimo contesto della CPU dalla memoria. Valore dei registri del file register, del flag register dell'IP e del IR. Tutte le volte che si opera una LOADCNT si utilizza lo SP come indirizzo per sapere da dove si deve iniziare a caricare le informazioni che erano state messe in memoria.

Si possono salvare contesti in modo annidato, cioè è possibile ricevere interruzioni anche durante l'esecuzione di interruzioni. In tale caso il contesto viene accodato la precedente.

4.5 La soluzione di un'Equazione di secondo grado

Equazioni di 2° grado

$$Ax^2 + bx + c = 0$$

$$\Delta = b^2 - 4ac$$

$$x = \frac{-b + \text{sqr}(\Delta)}{2a}$$

$$x = \frac{-b - \text{sqr}(\Delta)}{2a}$$

A pagina seguente e' riportata la realizzazione in linguaggio ASSEMBLY dell'algoritmo esposto in precedenza.

```

a   Read a , [ key ]
    Read b , [ key ]
    Read c , [key ]
b   Mul: e , b, b ;
    Mul: f, a, c ;
    Shl f, 2 ;

```

```

    Sub d, e, f, ;
    JS NONSOLUZIONI

```

```

c   Sqr d , d ;
    Shl a, 1 ;

```

```

    JZ SINGLESOLUTION

```

```

d   Sub b, o, b ;
    Sub g, b, d ;
    Div g, g, a ;
    Add h, b, d ;
    Div h,h, a ;

```

; ORA devo decidere in che sequenza produrre il risultato

```

Write [ CRT ], 2
Write [ CRT ], g
Write [ CRT ], h
JMP END

```

SINGLESOLUTION;

```

Add b, o, b ;
JZ NONSOLUZIONI
Div g , c , b ;
Write [ CRT ], 1
Write [ CRT ], g
JMP END

```

NONSOLUZIONI:

```

Write [ CRT ], 0
END;

```

$$e = b^2$$

$$f = ac$$

$f = 4ac$ (per velocizzare il programma utilizzo SHL invece di utilizzare MUL)

$$d = e - f = b^2 - 4ac$$

(etichetta dei non reali)

$$d = \text{sqr} (d)$$

2a verifico che sia non nullo

calcolo il complementare a due di $b = -b$

$$g = -b - \text{sqr} (\Delta)$$

$$g = x1$$

$$h = x2$$

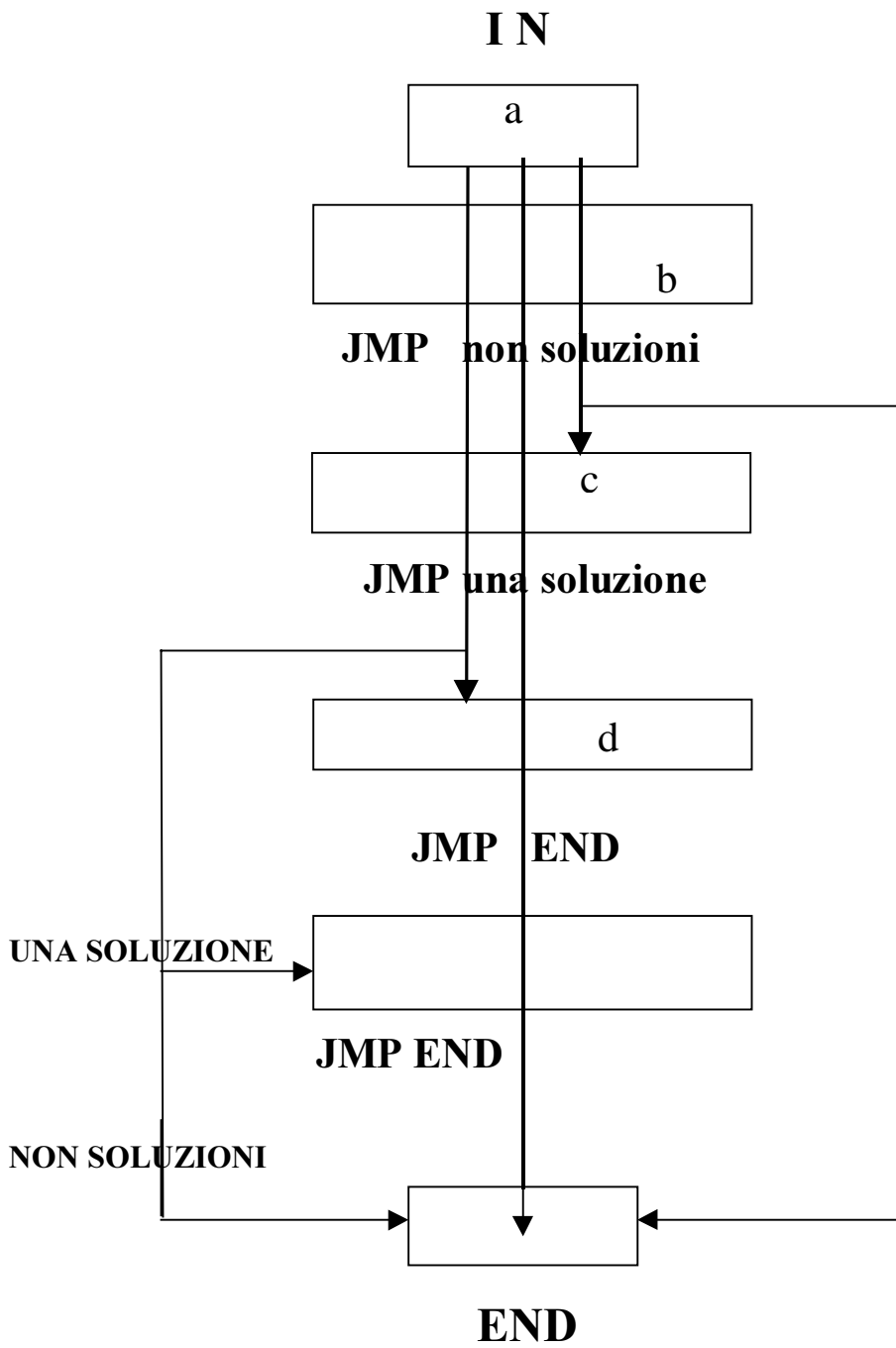
indica che ho due soluzioni

scrive la prima soluzione

scrive la seconda soluzione

$$X = \frac{-B}{C} = \frac{B}{-C}$$

2b



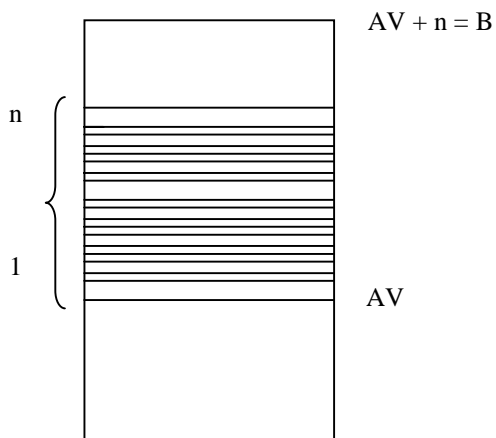
4.6 La somma degli elementi di un vettore

Fare la somma di tutti gli elementi di un vettore

$$S = \sum_{i=1}^n v[i]$$

CODIFICA IN ASSEMBLY

Facciamo l'ipotesi che il vettore sia locato nella posizione della memoria



Devo tenere conto di due casi :

1. CASO GENERALE

$$S_t = S_{t-1} + V_t$$

Ora l'operazione di somma la posso descrivere come operazione iterativa.

$$S_0 = 0 \quad \text{PASSO ZERO}$$

$$S_1 = S_0 + V_1$$

$$S_2 = S_1 + V_2$$

|
 |
 |
 |
 |

$$S_n = S_{n-1} + V_n$$

HO UN CICLO CHE CHIAMO LOOP

$$S_t = S_{t+1} + V_t$$

$$t = t + 1$$

Load v, [t]

Load n, [t]

Move s, 0

```

Move t, AV
Add b, AV, n
Load v, [ t ]
Add s, s, v
Add t, t, 1
Sub d, b, t
JNZ LOOP
HALT

```

} LOOP

il loop termina quando t è arrivato ad AV+n, cioè quando $t+AV+n=0$

4.7 Alcuni Esercizi da fare

- 1) Calcolare la somma degli elementi di una matrice ipotizzando che sia memorizzata per colonne.
- 2) Fare la somma degli elementi diagonali di una matrice.
- 3) Fare la somma degli elementi di una colonna o di una riga (n).
- 4) Trovare il massimo numero fra gli elementi di un vettore e di una matrice.
- 5) Trovare il minimo numero fra gli elementi di un vettore e di una matrice.

4.8 Ordine di esecuzione delle istruzioni

I processi di esecuzione delle istruzioni e l'ordine con cui queste vengono eseguite sono regolati principalmente da due registri

Uno di questi è l'*Instruction Register (IR)* un registro allacciato al bus interno che contiene una copia dell'istruzione che deve essere eseguita dall'elaboratore. E' quindi leggendo il contenuto dell'IR che il calcolatore sa quali circuiti attivare. In effetti il registro è collegato ad una sorta di decodificatore dell'istruzione che è capace di emettere segnali che attivano i dispositivi interessati e dirigono le informazioni da un dispositivo ad un altro.

Ma chi è che posiziona l'istruzione giusta nell'Instruction Register?

A questo ci pensa un altro registro chiamato *Instruction Pointer (IP)* che contiene l'indirizzo di memoria dell'istruzione successiva a quella in esecuzione.

Ad ogni ciclo di lettura dell'IR, l'IP viene prima vagliato per avere l'indirizzo corrente e poi incrementato per avere al ciclo macchina successivo l'istruzione successiva. Naturalmente ci sono delle eccezioni a questo procedimento: ad esempio se ci troviamo di fronte ad un'istruzione di salto, il contenuto dell'IP verrà probabilmente alterato; l'istruzione che verrà eseguita dopo non si troverà nella cella di memoria successiva alla precedente e quindi il registro deve contenere l'indirizzo dell'istruzione a cui si vuole saltare.

Durante questo ciclo, che si ripete in modo continuativo durante tutta la fase di elaborazione, l'unità centrale passa attraverso tre fasi

- 1) Lettura dell'istruzione da eseguire dalla memoria centrale ed incremento dell'IP . Questo primo passo è detto di FETCH.
- 2) Decodifica dell'istruzione letta; viene qui interpretato il significato dell'operazione.
- 3) Fase di esecuzione in cui i circuiti designati vengono attivati per eseguire le operazioni volute E' in questo passo che il contenuto di IP può essere alterato con un'istruzione di salto.

Ogni ciclo di questo genere viene a posizionarsi fra un'istruzione ed la successiva. Se io ho ad esempio un listato Assembly:

```
LOAD B , [Y]
ADD C , B , A
SAVE [ Z ] , C
```

Verrà eseguito un ciclo di LOAD con le proprie microistruzioni, prima però dell'esecuzione di ADD troverà posto un altro ciclo macchina composto dalle tre fasi sopra descritte.

Le microistruzioni necessarie all'adempimento dei passi sono:

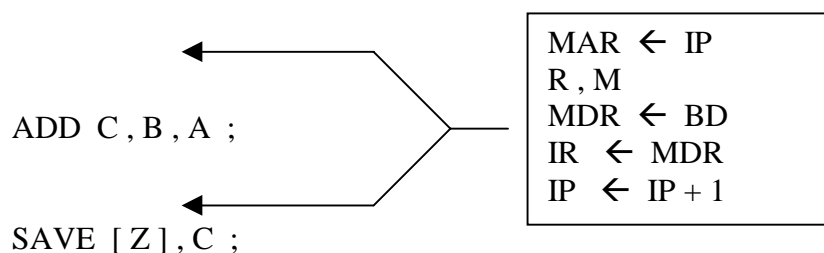
MAR	←	IP
R	,	M
MDR	←	BD
IR	←	MDR
IP	←	IP + 1

L'indirizzo dell'istruzione da eseguire è copiato nel registro interno MAR ; si definisce l'operazione di lettura in memoria ; dal bus dati arrivano le informazioni che vengono copiate nell'MDR, dall'MDR direttamente nell'IR da dove si avrà l'esecuzione dell'istruzione. L'IP intanto viene incrementato di uno e verrà a contenere l'indirizzo dell'istruzione successiva:

(Si analizza un ciclo dove non è presente un'istruzione di salto).

Ecco quindi che l'esempio può essere riscritto:

LOAD B , [Y] ; (con le relative microistruzioni)

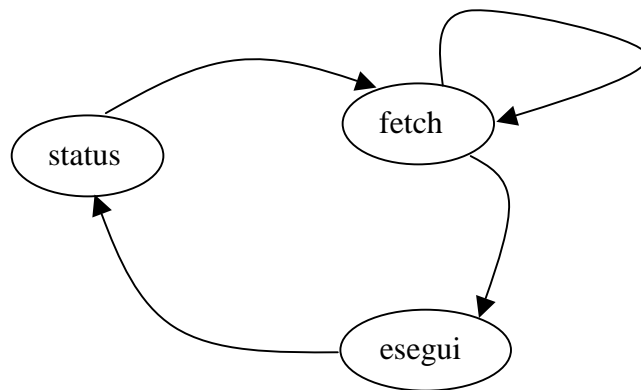


Naturalmente la temporizzazione di tutte le operazioni ed istruzioni è controllata dal clock (cfr 1.7.7).

Le istruzioni possono essere anche piu' lunghe del numero di bit del bus dati pertanto e' necessario fare delle letture multiple e non solo una semplice lettura in memoria. In tali casi si può gestire anche linguaggi con istruzioni di diversa lunghezza visto che tanto la prima parola che leggo contiene OPERATOR CODE e questo e' sufficiente per sapere di che istruzione si tratta e quindi di quante parole e' composta.

Quando si devono caricare istruzioni che sono lunghe diversi byte si devono fare diversi accessi alla

memoria fino a caricare tutta l'istruzione. In tali casi l'IP viene incrementato di un'unità ad ogni byte letto dalla memoria, se la memoria è organizzata a byte. Se invece è organizzata a word, 2 byte, allora l'incremento deve essere di due byte.



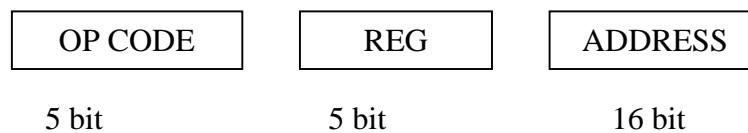
4.9 Modalità di Indirizzamento

Le istruzioni possono far riferimento al loro interno ad una cella di memoria dove è specificato l'indirizzo. Nella maggior parte dei casi l'indirizzo di una cella è codificato direttamente all'interno dell'istruzione. Esistono però dei casi in cui il riferimento di una cella è fatto in modi diversi chiamati *modi di indirizzamento*.

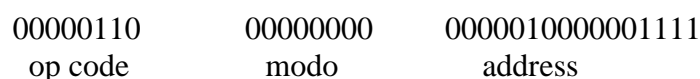
Elenchiamo qui di seguito i più diffusi (considerando solo le istruzioni ad un indirizzo).

4.9.1 Indirizzamento Diretto

L'indirizzo della cella a cui ci si riferisce è direttamente codificato all'interno dell'istruzione. Se ad esempio il formato di un'istruzione ad un indirizzo è il seguente:



e il modo di indirizzamento diretto si codifica con 00000000, allora l'istruzione assume la forma:



In questo caso abbiamo una somma codificata con 00000110 del contenuto del registro di accumulazione con il contenuto della cella 1039.

Sebbene sia il modo di indirizzamento più semplice presenta almeno due inconvenienti : in primo luogo

poiché un indirizzo assoluto deve essere ricalcolato ad ogni spostamento l'esecuzione del programma diventa più laboriosa ; in secondo luogo al crescere della memoria anche i campi di indirizzo diventano molto lunghi.

Si noti infatti che la grandezza della memoria centrale indirizzabile da un'istruzione è proporzionale al numero di bit riservati per la codifica dell'indirizzo all'interno dell'istruzione stessa.

Es. (Assembly) LOAD B , [5]

4.9.2 Indirizzamento Indiretto

All'interno dell'istruzione è codificato un indirizzo che indica una cella che contiene l'indirizzo dell'operando.

Riferendosi all'esempio precedente ,codificando il modo di indirizzamento indiretto con 00 0010 e assumendo che la cella di indirizzo 1039 contenga un ipotetico valore 5000, allora l'istruzione

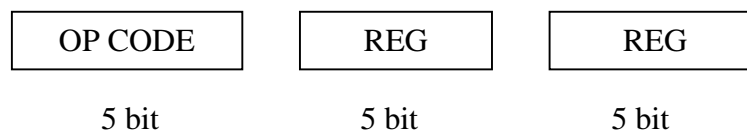
00000110	00000010	0000010000001111
op. code	modo	address

Somma al contenuto del registro di accumulazione il contenuto dell'indirizzo 5000.

Questa operazione, sebbene abbia indiscussi vantaggi ,richiede però un accesso alla memoria in più rispetto all'indirizzamento diretto.

Nel nostro caso la cella 1039 è detta puntatore perché il suo indirizzo è codificato all'interno dell'istruzione e contiene l'indirizzo dell'operando. Notiamo infine che la stessa cella di memoria contiene l'indirizzo assoluto dell'operando; esistono modi di indirizzamento in cui l'indirizzo assoluto deve essere calcolato dalla macchina.

E' il caso ad esempio dell'indirizzamento indicizzato.



Es. (Assembly) LOAD B [A]

4.9.3 Indirizzamento Indicizzato

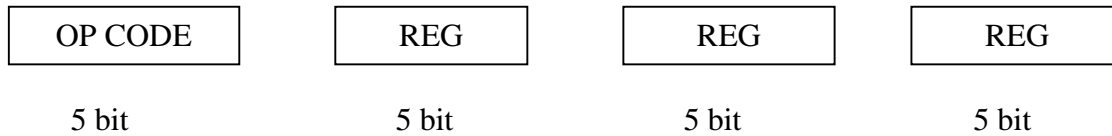
L'indirizzo della cella cui vogliamo riferirci si ottiene sommando in modo algebrico il contenuto di un registro particolare chiamato registro indice, all'indirizzo. Un registro indice è uno dei registri presenti - nella CPU. Codificando il modo di indirizzamento indicizzato con 00001 ,si ha ad esempio:

00000110	010	00001	0000010000001111
op. code	index reg.	modo	address

Ciò indica una somma del contenuto del registro di accumulazione con il contenuto della cella che indirizzo 1039 più il contenuto del registro indice 2. Ad esempio se il registro indice contiene il valore

300, la somma sarà eseguita con il contenuto della cella 1039+300.

Il modo di indirizzamento con indice risulta particolarmente utile per lavorare su un insieme di dati organizzati come un vettore. Ogni elemento del vettore si può ottenere modificando con opportune istruzioni il contenuto del registro indice:



Es. (Assembly) `LOAD B , [3 + C]`

4.9.4 Indirizzamento Basato

All'interno dell'istruzione è codificato al posto dell'indirizzo un valore che indica la distanza fra la cella cui si vuole fare riferimento e la cella che indica l'istruzione stessa

Al momento dell'esecuzione dell'istruzione l'indirizzo della cella si ottiene sommando un valore codificato all'interno dell'istruzione al contenuto del PC.

Indicando il modo di indirizzamento relativo con 00000011, consideriamo l'istruzione

00000110	00000011	0000010000001111
op. code	modo	distanza

Se essa ad esempio è allocata nella cella di indirizzo 2030 l'indirizzo dell'operando è calcolato sommando 1039 (distanza) a 2031 (contenuto di PC incrementato dopo il prelievo dalla memoria centrale dell'istruzione da eseguire) Il registro di accumulazione sarà sommato alla cella di indirizzo 1039+2031

Es. (Assembly) `LOAD B , [A+C]`

5 I Moduli e il Linker

Costituiscono un importante aiuto per la razionalizzazione del lavoro di programmazione . La filosofia che si cela dietro ad essi si basa sulla suddivisione di un programma in tanti blocchi singoli se possibile indipendenti l'uno dall'altro ; sono queste parti che prendono il nome di moduli.

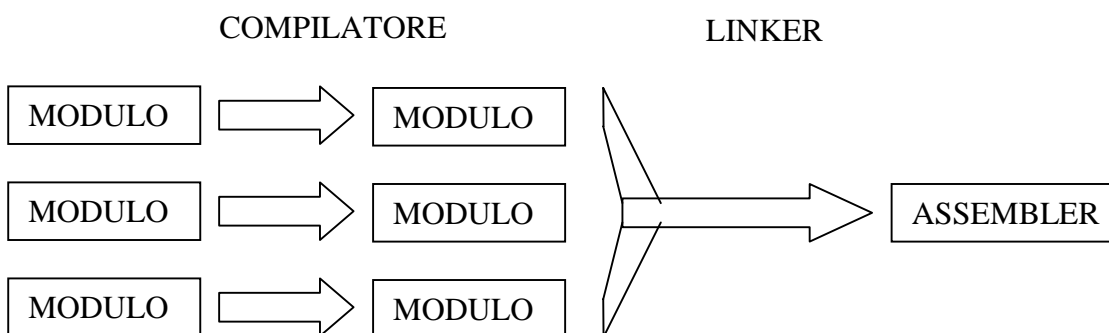
Nell'assembler di alcuni processori si possono trovare anche tipi diversi di moduli ,magari di testo e di codice. Ciò non pregiudica però il fatto che ogni modulo ,per garantire un regolare funzionamento del programma deve essere collegato agli altri moduli oggetto prodotti separatamente da assembleri e compilatori.

Lo strumento che assolve a questo compito, è comunemente chiamato *linker* (collegatore).

In effetti i moduli fanno riferimento l'uno all'altro, tramite specifiche istruzioni di chiamata e di ritorno, che il traduttore non può soddisfare, in quanto esamina solo un programma sorgente per volta. A questo compito provvede il linker, che costruisce un unico modulo oggetto collegando i singoli moduli prodotti dal traduttore e calcolando gli indirizzi da porre nelle istruzioni, completandole.

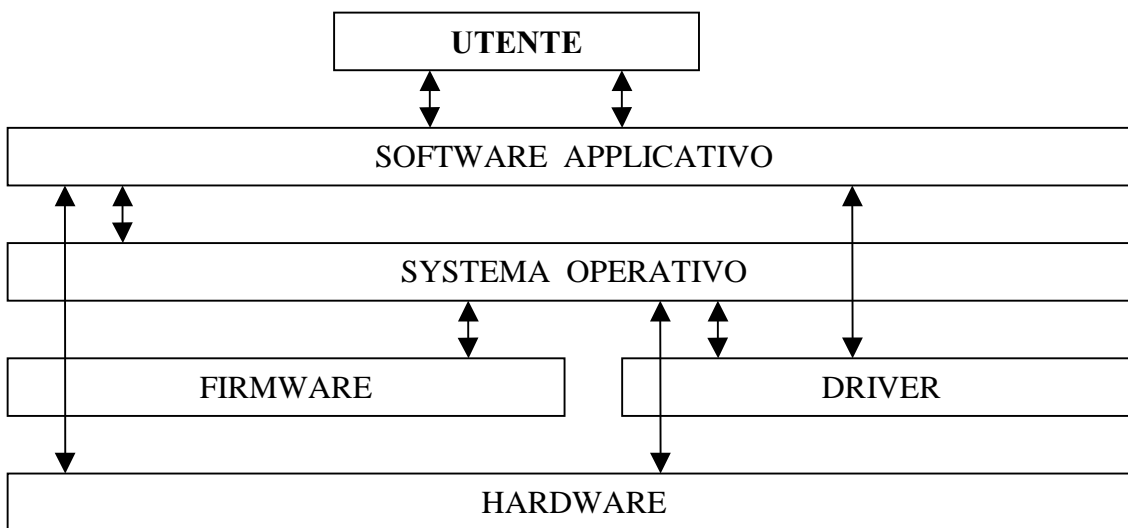
Oltre all'usuale lavoro di collegamento di programmi e sottoprogrammi, il linker può fare riferimento alle cosiddette "librerie", insiemi di sottoprogrammi appositamente scritti dal produttore del linguaggio per risolvere problemi comuni a tutte le applicazioni, oppure scritti da programmatori, e resi disponibili, per risolvere particolari problemi applicativi. Un utente può quindi utilizzare a piacimento un sottoprogramma di una libreria, senza doverlo riscrivere.

Le librerie sono uno strumento molto utile alla programmazione, in quanto consentono ad uno sviluppatore di guadagnare tempo prezioso nella stesura di un programma. Naturalmente l'operazione di collegamento, non è sempre ad esito scontato. Alcune volte il linker può rilevare errori, in tal caso l'operazione di Linking si blocca ed un messaggio viene mandato all'utente.



6 Hardware e Software

L'elaboratore è composto dall'hardware, cioè quei dispositivi fisici che lo costituiscono, e dal software ,quelle procedure e istruzioni che ne dirigono le operazioni.



Il software è' parte integrante del calcolatore poiché ne permette una maggiore o minore adattabilità alle richieste dell'utente stesso.

La struttura software è costituita da diverse categorie di programmi:

B.I.O.S. (basic input output system)

O.S.(operating system): KERNEL, UTILITY

SOFTWARE APPLICATIVO.

IL SISTEMA OPERATIVO (OPERATING SYSTEM) è un insieme di programmi, che agisce da intermediario tra il calcolatore e l'utente, cosicché questi non debba interagire direttamente con l'hardware. Il Sistema Operativo, infatti, rende possibile l'esecuzione del software applicativo in modo trasparente all'utente. Questi infatti, consegna i dati richiesti al sistema operativo, che scende all'hardware, senza che l'utente ne conosca i complicati meccanismi.

Esempi di sistemi operativi sono: MS DOS, WINDOWS, OS2, UNIX VMX..

Una particolare categoria di software è il BIOS (o firmware), un insieme di programmi, inseriti nell'hardware alla costruzione e copiati in memoria centrale all'avviamento (bootstrap). Sono utilizzati dal sistema operativo. per accedere alle risorse hardware del sistema. Grazie a questi programmi il sistema operativo può non considerare la struttura interna dell'elaboratore. All'acquisto di un personal computer si ha l'hardware ed il BIOS su cui si può implementare il sistema operativo più adatto a gestire, col software applicativo, le applicazioni più varie.

Architetture

SIMD, SID, MIMD

Tipi	UT	MT	MP
Personal	1	No/Si	No/Si
WorkStation	10	Si	Si
Mini	100	Si	Si
MainFrame	1000	Si	Si

7 EBNF, Forma di Backus-Naur Estesa

Un programma è costruito come una sequenza di simboli o caratteri che formano il “testo”.

Come per i linguaggi naturali, ogni linguaggio di programmazione ha associato un insieme di regole rigidamente definito che descrive le modalità di costruzione di un programma valido per il linguaggio considerato.

Queste regole servono in primo luogo per garantire al programmatore la correttezza e l’effetto del suo programma, e poi per permetterne la comprensione da parte del calcolatore e di chiunque lo legga.

Le regole del linguaggio sono costituite da due parti note come *sintassi* (regole di scrittura) e *semantica* (significato del testo).

SINTASSI: è l’insieme di regole che definiscono le relazioni possibili tra i singoli costrutti del linguaggio. In altre parole le regole sintattiche definiscono il modo in cui le “parole” (o vocabolario) del linguaggio si possono combinare per formare delle “frasi”. In particolare la sintassi dei linguaggi di programmazione è la forma in cui devono essere scritti i programmi. Ad esempio nella lingua italiana possiamo legare il soggetto, il verbo e il complemento secondo certe regole di sintassi.

SEMANTICA: è il significato che viene dato alle sequenze e combinazioni di costrutti offerti da un linguaggio e, in particolare nel caso dei linguaggi di programmazione, è il significato di un programma scritto in quel linguaggio. Di solito le regole semantiche sono stabilite in maniera meno formale rispetto alle regole sintattiche.

Vediamo degli esempi riguardo alle definizioni sopra riportate.

- Esempio

Prendiamo in esame le due frasi seguenti:

1) *Il gatto è verde*

2) *Il verde è gatto*

La prima frase è sintatticamente giusta, ma semanticamente sbagliata; invece la seconda frase è anche sintatticamente sbagliata. NOTA: è molto importante conoscere il contesto in cui si opera: per esempio in un romanzo di fantascienza la prima frase può risultare semanticamente corretta, mentre nella normalità non lo è.

- Esempio

Anche il linguaggio assembly ha una sua sintassi e una sua semantica; per esempio:

LOAD B, [A]

è sintatticamente corretto, dato che l'istruzione LOAD richiede il *registro* (B) su cui caricare il contenuto della cella avente un certo *indirizzo* ([A]); in questo esempio specifico si ha il caso di indirizzamento indiretto: il contenuto della cella di indirizzo contenuto nel (o puntato dal) registro A, viene caricato nel registro B);

LOAD A,B,C

questo è sintatticamente errato, dato che si hanno tre registri;

LOAD [A], B

questo è sintatticamente errato, dato che si ha prima l'indirizzo e poi il registro;

Abbiamo dunque visto i concetti di sintassi e semantica.

7.1 Il metalinguaggio EBNF

In informatica si usano strumenti formali per descrivere la sintassi di un linguaggio: essi si chiamano *metalinguaggi* e sono dei linguaggi che servono per descrivere il linguaggio di programmazione che ci interessa. Il formalismo che useremo è il **BNF** (Backus - Naur Form), che assume la notazione **EBNF** (Extended Backus - Naur Form) nella sua versione più aggiornata. In generale nell'EBNF i costrutti sintattici sono definiti da termini, alcuni dei quali racchiusi tra parentesi angolari <>, che richiamano alla mente la natura o il significato del costrutto e sono poi utilizzati nella descrizione del significato.

Ad esempio la forma di un programma Pascal è definita dalla seguente regola EBNF:

programma := < intestazione > < corpo > ' . '

Questa regola va letta: "Un programma è composto da una *intestazione*, seguita da un *corpo* seguita da un punto ' . ' ". Poi altre regole definiscono le forme consentite di un' *intestazione* e di un *corpo*:

corpo := ' Begin ' { < istruzione > } ' end '

istruzione := < assegnazione > | < selezione semplice > | < iterazione for > | < iterazione while > |
< iterazione repeat > | < case > | < sottoprogramma procedure > | < sottoprogramma function > |

Secondo l'EBNF i linguaggi di programmazione sono composti da due tipi di elementi:

1) *simboli terminali*: sono le 'parole o i simboli chiave' del linguaggio

- 2) *simboli non terminali*: sono le parole o i simboli non chiave del linguaggio e che quindi hanno bisogno di essere definiti e descritti in termini di EBNF; i simboli non terminali verranno indicati tra le parentesi < >. Riportiamo di seguito i simboli terminali propri dell'EBNF e successivamente i simboli terminali o identificatori propri del linguaggio Pascal.

7.2 SIMBOLI TERMINALI DELL'EBNF

SIMBOLO TERMINALE DELL'EBNF	SIGNIFICATO DEL SIMBOLO TERMINALE
:=	definizione
{ }	ripetizione del costrutto tra parentesi zero o più volte
[]	opzionalità del costrutto tra parentesi
	oppure
< >	delimitazione di un costrutto sintattico non terminale

7.3 SIMBOLI TERMINALI O IDENTIFICATORI (VOCABOLARIO) DEL PASCAL

Qualsiasi linguaggio, sia esso parlato o di programmazione, fa uso di un vocabolario. L'italiano, per esempio, nella sua forma scritta, consiste di parole, numeri, e simboli di punteggiatura, cioè il lessico. Il vocabolario base del linguaggio di programmazione Pascal è costituito da *lettere*, *cifre*, *simboli speciali*. Le frasi del linguaggio sono quindi costruite a partire da questo vocabolario in accordo con la sintassi del Pascal. Esaminiamo adesso queste tre categorie sintattiche.

lettera := A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X |
Y | Z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

cifra := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

simbolo speciale := + | - | * | / | = | < > | < > | < = | > = | (|) | [|] | { | } | : = | . | , | ; | : | ' | ^ | div | mod | nil
| in | or | and | not | if | then | else | case | of | repeat | until | while | do | for | to | downto | begin | end | with |

goto | const | var | type | array | record | set | file | function | procedure | label | packed | program | forward | integer | real | boolean | . . .

NOTA: per differenziare i simboli terminali dell'EBNF dai simboli terminali del linguaggio Pascal, questi ultimi saranno compresi tra apici ' '. Perciò:

– simboli terminali dell'EBNF ⇒ senza ' '

– simboli terminali del linguaggio Pascal ⇒ con ' '

NOTA: pertanto tutte le lettere, cifre e simboli speciali sopra elencati sono da intendersi compresi tra apici ' '.

Esempi

Assegnazione : = < variabile > ' : = ' < espressione > ' ; '

Selezione semplice : = ' if ' < condizione > ' then ' < istruzione > [' else ' < istruzione >] ' ; '

Vediamo adesso alcuni esempi che possano chiarire ulteriormente i concetti finora espressi.

7.3.1 Esempio, l'Assembly

Si abbia l'istruzione

LOAD A, [53]

Dove A = nome della variabile nel file register, [53] = indirizzo della cella di memoria. Questa istruzione di indirizzamento (di tipo immediato) contiene:

- 1) simboli terminali, ossia le parole chiave del linguaggio - come LOAD, A, la virgola, la parentesi quadra –
- 2) simboli non terminali, come il 53 (mentre le cifre 5 e 3 da sole sarebbero state simboli terminali)

Vediamo adesso come, secondo EBNF, possiamo definire l'istruzione LOAD.

Iniziamo innanzitutto col definire un programma in linguaggio Assembly:

Prog : = { < istruzione > } cioè un programma è definito come un insieme di istruzioni

E' da notare come la parola istruzione sia un simbolo non terminale, pertanto deve essere racchiusa tra le parentesi < > e dovrà essere descritta in termini di EBNF.

Istruzione : = < LOAD > | < SAVE > | < AND > | < OR > | < NOT > | < MOVE > | < JMP > |

In questo modo abbiamo elencato le varie istruzioni che ci sono in Assembly, ma esse non sono simboli terminali e perciò le devo ancora definire.

Andiamo in particolare a definire l'istruzione LOAD, che era quella con cui abbiamo iniziato l'esempio:

LOAD : = LOAD < Registro > ' , ' '[' < Registro > | < Indirizzo > '] '

Bisogna notare che in questa scrittura le parole Registro e Indirizzo sono simboli non terminali (e perciò racchiusi tra <>) mentre i simboli ' , ' , '[' , '] ' sono terminali.

Descriviamo adesso i due simboli non terminali:

Registro : = A | B | C | | Z dove A, B, ..., Z sono i nomi dei registri

Indirizzo : = 0 | 1 | 2 | | $2^{16} - 1$

oppure in alternativa

Indirizzo : = < cifra > | < indirizzo > < cifra >

cifra : = 0 | 1 | 2 | | 9

A questo punto abbiamo completamente descritto l'istruzione LOAD.

In questo modo si può descrivere tutta la sintassi del linguaggio.

Infatti si possono descrivere tutte le altre istruzioni; vediamo per esempio ADD:

ADD : = ADD < Registro > , < Registro > , < Registro >

7.3.2 Esempio: il Pascal

Riportiamo adesso, a titolo di esempio, alcune istruzioni e costrutti propri del linguaggio Pascal usando il formalismo di Backus - Naur esteso:

selezione semplice : = ' if ' < condizione > ' then ' < istruzione > [else < istruzione >]

condizione booleana : = false | true | < variabile booleana >

condizione : = <condizione booleana > | < condizione > |
 < condizione > < binary boolean operator > < condizione > |
 < unary boolean operator > < condizione > |
 < numerical expression > < comparator > < numerical expression >

binary boolean operator : = and | or

unary boolean operator : = not

comparator : = < | > | <= | >= | <> | ==

sequenza : = 'begin' { < istruzione > } 'end' ';' ;

iterazione for : = 'for' < variabile > ':=' < valore iniziale > 'to' < valore finale > 'do' < istruzione >

iterazione while : = 'while' < condizione > 'do' < istruzione >

iterazione repeat : = 'repeat' { < istruzione > } 'until' < condizione >

case : = 'case' < variabile > 'of' { < casi > } 'else' < istruzione > ';' 'end' ';' ;

sottoprogramma procedure : = 'procedure' < nome sottoprogramma > ['('
{ < parametri > } ')'] ';' [< intestazione >] 'begin' [{ < istruzione > }]
'end' ';' ;

sottoprogramma function : = 'function' < nome sottoprogramma > ['('
{ < parametri > } ')'] ':' < tipo > ';' [< intestazione >] 'begin' [{ < istruzione > }]
'end'