SOFTWARE

1. Assembly

1.0 Premessa

Le cose che ho scritto di seguito sono scritte molto meglio nel materiale del corso "Laboratorio di Architetture I", disponibile sul sito *http://twiki.dsi.uniroma1.it/twiki/view* (Twiki).

1.1 Il Linguaggio

L'assembly è IL linguaggio di programmazione a basso livello (più vicino alla macchina).
Viene "costruito" sull'architettura della macchina e per questo ne sfrutta tutte le caratteristiche.

Questo è il motivo per cui viene studiato nel corso di Architettura degli Elaboratori e questo è il motivo per cui, in questo corso, ogni volta che si parla di "programmatore", si intende "programmatore assembly", ovvero che scrive in assembly i suoi programmi.

L'assembly è composto da:

- *istruzioni*: sono istruzioni macchina espresse tramite codice mnemonico.
  Esiste, infatti, una corrispondenza 1:1 fra istruzioni dell'assembly e istruzioni macchina.
  Il compito dell'assembly è pertanto quello di proporre le istruzioni macchina in modo mnemonico.
  Questa corrispondenza 1:1 si riflette anche sul fatto che l'assembly non viene compilato, ma soltanto tradotto da un programma detto *assemblatore* o *assembler*.
  Ogni istruzione, sia essa dell'assembly o macchina, viene pensata avente quattro campi (eventualmente vuoti): etichetta, istruzione, operandi, commento. Il campo "etichetta", in particolare, serve per semplificare i salti relativi.

- *direttive*: non sono istruzioni direttamente eseguite dalla macchina, ma informazioni che vengono fornite all'asseblatore perchè faccia cose che fanno comodo.
  Ad es.: riservare un tot di celle consecutive nella memoria centrale.

1.2 Funzionamento

Quando un file o modulo sorgente scritto in assembly viene tradotto in modulo oggetto, l'assemblatore trasforma il codice mnemonico in stringhe binarie e risolve soltanto i salti interni al modulo.
Il modulo oggetto sarebbe già perfettamente eseguibile se non ci fossero salti e riferimenti irrisolti ad altri moduli oggetto.

Un programma di nome *linker* si occupa di fondere in modo corretto i moduli oggetto che servono. I moduli oggetto fusi, possono derivare anche da altri linguaggi di programmazione, oltre all'assembly. Viene creato così il modulo eseguibile.

In particolare, nei moduli oggetto: gli indirizzi assoluti vengono fatti partire da zero; i salti ed i riferimenti ad etichette esterne non sono risolti.

1.3 L'importanza delle interruzioni software

Se non esistessero, per far funzionare più programmi contemporaneamente bisognerebbe linkarli.

Le istruzioni di trap, in fondo, eseguono una specie di link tramite gli indirizzi assoluti del vettore delle interruzioni: la stessa cosa che fa il linker quando cambia gli indirizzi assoluti dei moduli che unisce.

Se non esistessero, più programmi che usano una stessa libreria dovrebbero essere linkati ognuno alla sua copia della libreria. Tramite trap, invece, viene caricata una sola copia della libreria a cui i vari programmi accedono a turni.

Se non esistessero, non si potrebbe avere la struttura di sistema operativo che si ha attualmente: un nucleo gestisce il colloquio tra applicazioni e periferiche interne ed esterne. Questi colloqui, infatti, nel caso nucleo-applicazioni vengono gestiti tramite trap.

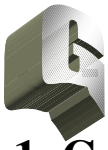
2. Software di base

Il software che deve essere fornito dal produttore della macchina.
Senza questi programmi, la macchina sarebbe inutilizzabile.

Ciò che assolutamente non può mancare nel software di base:

- *sistema operativo*: si occupa di gestire la macchina a basso livello
- *librerie*: moduli di codice già pronto.
  Possono servire, ad esempio, per implementare via software funzioni che la macchina non sa fare direttamente via hardware
- *bootstrep*: programma che avvia la macchina.
  La memoria centrale è composta da registri, i quali altro non sono che catene di flip-flop. I flip-flop, non essendo basati su proprietà magnetiche, perdono i bit che memorizzano se non ricevono corrente. Quando il calcolatore viene spento, quindi, la RAM si azzera. Quando si accende un calcolatore, il PC caricherebbe solo stringhe nulle se non ci fosse un programma in grado di caricare in memoria le prime istruzioni.
  Questo programma è il bootstrep. Esso risiede in una memoria ROM: legge una istruzione alla volta da una periferica di memorizzazione in grado di mantere stringhe binarie anche in assenza di corrente e la carica in memoria.
  Si noti che se la RAM potesse conservare le informazioni anche a macchina spenta, non servirebbe il bootstrep e si avrebbe un avvio istantaneo. Esistono studi per l' implementazione di memorie di questo tipo in futuro
- *Assembly e assembler*
- *Compilatori*
- *File System*: software per la gestione delle memorie di massa.

Per avere una macchina efficiente, sarebbe opportuno scrivere almeno il software di base in assembly.



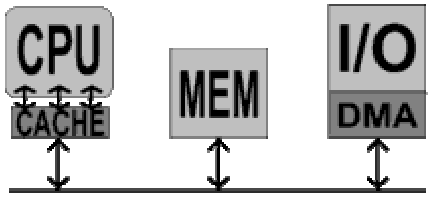
ACCORGIMENTI E TRUCCHI

1. Gerarchia di memoria

1.1 Introduzione

La velocità dei circuiti elettronici dipende dalla velocità di commutazione delle loro porte logiche. Purtroppo, maggiore è la velocità di una porta, più elevato è il suo prezzo.
La memoria centrale è composta da moltissime porte, almeno 4 per ogni bit che può contenere. La CPU è composta da molte meno porte. Questo comporta che la CPU possa essere costruita con porte più veloci di quelle della memoria centrale. Avere la RAM più lenta, però, rallenta di conseguenza la CPU quando questa ha bisogno di recuperare stringhe dalla memoria (molto spesso), vanificando la sua maggiore velocità.
Per evitare tutto ciò è stata introdotta la gerarchia di memoria, un accorgimento esclusivamente pratico.

1.2 La memoria cache



Fra CPU e RAM viene interposta una memoria costruita con la stessa tecnologia della CPU, detta *cache memory* o *memoria nascosta*. Essendo realizzata con porte costose (le stesse della CPU), deve essere in quantità molto minore rispetto alla RAM, per limitare i costi; ciononostante, essa può contenere centinaia di word.
La CPU non colloquia direttamente con la RAM, ma con la cache. Quando la control unit richiede un nuovo indirizzo alla cache, se questa ha la stringa relativa a quell' indirizzo, la manda alla CPU direttamente, altrimenti carica dalla memoria centrale - passando attraverso il bus - non solo quella stringa,

ma tutto un blocco di stringhe limitrofe. Questo trasferimento da memoria centrale a cache avviene in modo parallelo: in una sola volta vengono trasferite più stringhe. Tramite questo accorgimento, la gerarchia di memoria riesce ad essere molto efficiente e permette alla CPU di lavorare quasi sempre alla sua massima velocità, colloquiando solo con la cache e minimizzando gli accessi alla memoria centrale.

Perchè tutto funzioni correttamente, è necessario che la cache lavori con una logica diversa da quella della memoria centrale. La cache è, infatti, una *memoria di tipo associativo*.
Questi tipi di memoria ricevono una maschera (una parte) del contenuto che si cerca e restituiscono tutto il contenuto associato a quella maschera. In particolare, nella cache ogni registro non contiene soltanto il dato vero e proprio, ma anche il suo indirizzo in memoria centrale (che è la maschera). Alla cache arriva una parte della stringa cercata - l' indirizzo - ed essa restituisce tutto il dato - indirizzo + dato vero e proprio.
Ciò è necessario perchè permette di memorizzare dati in qualsiasi ordine e non sequenzialmente (dopo il dato di indirizzo 0000, nella RAM ci deve essere il dato di indirizzo 0001, mentre nella cache ci può essere qualsiasi dato), fattore, questo, che risulta indispensabile proprio per come funziona la cache.
Ovviamente, perché ogni registro della cache contenga indirizzo + dato, è necessario utilizzare registri più capienti di quelli della RAM, aumentando ulteriormente il prezzo.

Ultime osservazioni:

questo meccanismo gerarchico risulta trasparente al programmatore assembly;
programmi con molte istruzioni di salto e in cui tali salti sono particolarmente ampi sono poco efficienti perchè richiedono dati non contenuti nella cache; per questo alcuni costruttori forniscono istruzioni assembly di salto con un offset limitato;
in questo corso non si tratteranno i criteri per la sostituzione dei dati all' interno della cache.

1.3 Livelli di cache

Ci possono essere diversi livelli di cache, via via più lenti, ma anche più capienti e meno costosi. In genere, la cache di primo livello (L1) si trova integrata nel chip della CPU (il core), la cache di secondo livello (L2) si trova sulla scheda della CPU ed una eventuale cache di terzo livello (L3) viene posta nei pressi della CPU.

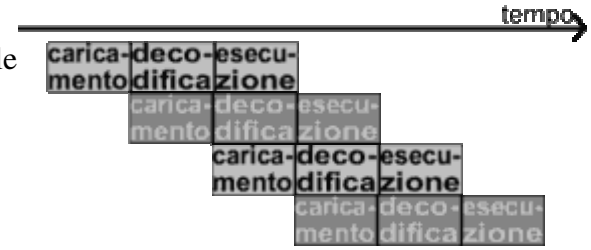
2. Canalizzazione

2.1 Introduzione

Come visto in precedenza, le macchine di Von Neumann hanno bisogno di tre fasi per arrivare a fare quello che è richiesto da una istruzione. Queste fasi sono: caricamento, decodifica, esecuzione.

Ognuna di esse richiede un certo tempo: il caricamento richiede un tempo Δt_1 fisso, la decodifica richiede un tempo Δt_2 fisso se la macchina non è microprogrammata o variabile altrimenti, l'esecuzione richiede un tempo Δt_3 variabile.
Ogni volta che una istruzione deve essere eseguita, quindi, occorre un tempo $\Delta t=\Delta t_1+\Delta t_2+\Delta t_3$.
L' idea alla base della *canalizzazione o pipelining* è di far eseguire queste fasi contemporaneamente.

2.2 Applicazione



Progettando il calcolatore in modo tale che le varie fasi non interferiscano l' una con l' altra, è possibile iniziare il caricamento dell' istruzione successiva mentre si stà ancora decodificando l' istruzione corrente.
Ciò è possibile progettando macchine in cui le tre fasi richiedano lo stesso tempo, altrimenti i tempi non si incastrerebbero bene [si vedano le macchine RISC].
Grazie alla canalizzazione, al tendere ad infinito delle istruzioni eseguite, si riduce il tempo di un terzo (tre volte più veloce).

Ultime osservazioni:
in realtà il tempo diminuisce di uno fratto il numero di fasi, quindi di 1\3 nel nostro caso;
esistono tuttavia calcolatori progettati per avere molte più fasi, così da ottenere una riduzione più marcata di tempo (computer vettoriali, studiati ad Architetture III);
il meccanismo di pipelining è trasparente al programmatore assembly;
quando si hanno istruzioni di salto il metodo perde la sua efficacia perchè l' istruzione successiva già acquisita non serve più; questo problema si avverte ancora di più sui calcolatori con molte fasi (bisogna svuotare la pipeline di tutte le istruzioni che ci si stanno trattando in anticipo); per ovviare a ciò sono stati introdotti algoritmi di *branch prediction*, ovvero di previsione dei salti, in

grado di aumentare di molto l'efficienza del pipelining.

3. Memoria Virtuale

3.1 Introduzione

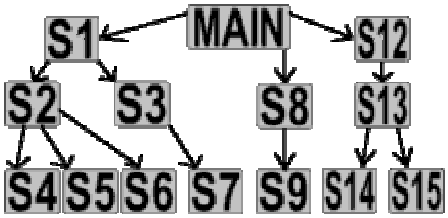
Può capitare di dover eseguire un programma più grande della memoria centrale disponibile. Quando ciò avviene, il programma non può essere caricato nella RAM e quindi non può essere eseguito. In realtà esiste un metodo per risolvere tale problema: la memoria virtuale.

L' idea di base è la seguente: innanzitutto serve una memoria diversa dalla RAM per contenere quella parte di programma che non vi entra - in genere viene utilizzata una periferica, come ad esempio una unità disco; quando il programma deve essere eseguito, si carica nella RAM solo la parte che entra, mantenendo il resto sulla periferica.

Questo principio deve essere implementato in due modi diversi, a seconda che la memoria indirizzabile sia minore o maggiore della memoria fisica. Come già visto, infatti, per indirizzare la memoria si usano stringhe numeriche di *n* bit, dove in genere *n* coincide con la dimensione di una word. In questo modo si possono indirizzare 2^n celle. Esistono tre eventualità: la memoria fisica ha $N=2^n$ celle; la memoria fisica ha $N>2^n$ celle (più celle di quante se ne possono indirizzare e quindi è divisa in pagine); la memoria fisica ha $N<2^n$ celle (meno celle di quante se ne possono indirizzare). Tenendo presente che il primo caso si può trattare sia come il secondo che come il terzo, rimane da vedere come realizzare una memoria virtuale negli ultimi due casi.

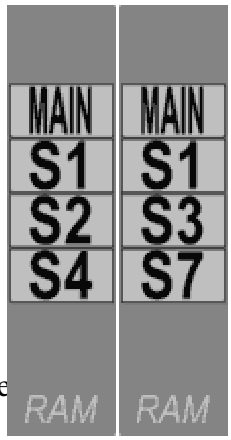
3.2 Overlay

Si supponga che $N>2^n$.



In questo caso, il programma più grande della memoria deve essere spezzato in un main che richiami subroutines.

In memoria viene caricato un ramo alla volta, ad esempio il ramo Main-S1-S2-S4.



Finchè si lavora su quel ramo non si hanno problemi. Se però da quel ramo bisogna passare a S5, per arrivarci occorre togliere dalla RAM S4 e metterci S5. Se invece bisogna andare ad S7, è necessario togliere S4 ed S2 per mettere S3 ed S7. Per andare infine ad S15, sarebbe stato necessario togliere S4, S2, S1 per mettere S12, S13 ed S15.
In questo caso, con togliere e mettere si possono intendere due azioni distinte: togliere e mettere in un' altra pagina o, se tutte le pagine sono occupate, togliere e mettere sulla periferica utilizzata.

Perchè tutto funzioni occorre che:

- il codice contenuto in un ramo non occupi più di una pagina (2^n celle); se ciò non si verifica, basta suddividere ulteriormente in programma in subroutines, finchè tutti i rami entrano;
- le chiamate a subroutines siano ad albero (non ci possono essere chiamate tra fratelli);
- il linker sia scritto in modo da generare un eseguibile per ogni ramo.

Per questo, la memoria virtuale ad overlay (che significa sovrapposizione) permette di eseguire

programmi lunghi a piacere ma non è trasparente al programmatore assembly, perchè questo deve scrivere un opportuno linker e deve spezzare opportunamente i programmi in subroutines.

3.2 Swapping

Si supponga che $N<2^n$.
In questo caso il programmatore assembly non si preoccupa che $N<2^n$, ma sfrutta tutti i 2^n indirizzi possibili, anche se questi non corrispondono a nessuna cella. Questi indirizzi scritti dal programmatore assembly prendono nome di *indirizzi virtuali*.

Si supponga, ad esempio, di trovarsi nel caso in cui un programma di 2MB debba essere caricato in una RAM da 1 MB per essere eseguito.
Come prima cosa, la macchina carica quel che può: il primo MB.
I problemi nascono quando il programmatore effettua un salto ad un' istruzione presente nel secondo MB, oppure quando si arriva all' ultima istruzione del primo MB. Quando ciò avviene, si ha una interruzione interna perchè ci si è riferiti ad un indirizzo inesistente. Si ha quindi un salto al programma di servizio che toglie il MB già caricato, salvandolo su disco, e carica il rimanente MB al suo posto.

Ciò comporta due problemi. Il primo è che gli indirizzi del secondo MB risultano tutti sfasati di un MB. Il secondo è che scaricate tutto quello che c' era in memoria e caricare tutto quello che non entrava è molto inefficiente.
Riguardo alla prima questione, basta una minima modifica harware: bisogna inserire un registro di offset che agisca sul P.C. .
Si ha quindi che:

```
        indirizzo virtuale =
  indirizzo fisico (valore del P.C.) +
offset (numero nel registro di offset)
```

L' offset è di 0 MB quando ci si trova nel primo MB ed è di 1 MB quando viene caricato il secondo pezzo.
Riguardo alla seconda questione, in realtà non viene scaricato e poi caricato tutto il contenuto della RAM, ma "pezzi" di RAM detti *pagine*. Si dice quindi che la memoria è *paginata* e l' operazione che carica e scarica pagine fra disco e memoria centrale è detta swapping.
ATTENZIONE ! Queste pagine non hanno niente a che vedere con le pagine di memoria sinora trattate: dire che una memoria ha una struttura a pagine è completamente diverso dal dire che è paginata; sono due concetti differenti ed indipendenti l' uno dall' altro.

La tecnica dello swapping non è trasparente al programmatore assembly, perchè questo deve scrivere un opportuno programma di servizio dell'interruzione.

MACCHINE CISC E MACCHINE RISC

1. Caratteristiche

RISC: reduced instruction set computer.
CISC: complex instruction set computer.
Le macchine RISC hanno istruzioni più semplici, cioè meno potenti.
Le macchine CISC hanno istruzioni più complesse, cioè più potenti.

La potenza di una istruzione indica "quante cose riesce a fare quella istruzione tutte in una volta".
La complessità ovvero potenza di una istruzione è legata al modo di indirizzamento degli operandi e più in generale a tutto ciò che è di contorno all' istruzione.

Utilizzando l' indirizzamento implicito, le istruzioni sono poco potenti: ne servono molte per fare anche le operazioni più semplici.
Una istruzione di somma, ad esempio, è sempre una istruzione che attiva l' opportuno circuito della CPU, ma se ci si riferisce agli operandi implicitamente, allora è meno potente e più semplice, se al contrario gli operandi sono indicati espicitamente, è più potente e complessa. Nel caso dell' istruzione semplice, bisogna prima portare i due operandi dalla memoria o dai registri in cui sono ai registri predefiniti; bisogna quindi chiamare l' istruzione; bisogna infine spostare il risultato dal registro in cui viene di default messo, alla memoria o al registro in cui si intende conservarlo. Per la somma con indirizzamento esplicito, invece, basta una sola istruzione, nella quale vengono contestualmente indicate le posizioni degli operandi e del risultato.

Le istruzioni delle macchine RISC, avendo una carenza di modi di indirizzamento proprio perché sono poco potenti, operano direttamente sui registri della CPU e non sulla memoria centrale. Vengono a tal proposito fornite solo due istruzioni che accedono alla RAM: *load* e *store*. Con queste si caricano le celle di memoria su cui si vuole operare nei registri della CPU e viceversa. L'istruzione *move*, al contrario, può agire solo sui registri della CPU.

Un' altro degli elementi di contorno che determina la maggiore o minore complessità è il formato delle istruzioni: nelle macchine CISC può essere variabile (come già visto, questo significa che la lunghezza delle istruzione può variare di multipli di word), nelle RISC è fisso ed in genere pari ad una word.

Ovviamente esistono macchine intermedie fra RISC e CISC.

I motivi della distinzione fra macchine RISC e macchine CISC sono storici:

- inizialmente i calcolatori erano RISC perchè più economici e facili da realizzare; allora, infatti, la tecnica costruttiva era ancora molto primitiva e gli elementi utilizzati erano molto costosi e rari;
- si è poi passati a macchine CISC; questo passaggio è stato necessario poichè la tecnica costruttiva era migliorata mentre il costo dei componenti era ancora elevato; si potevano quindi realizzare macchine più complesse, ma con poca memoria; servivano per questo istruzioni molto potenti, così che i programmi occupassero meno spazio possibile;
- con l' economizzazione dei componenti ed ulteriori miglioramenti nella tecnica costruttiva, si è avuta la rinascita di calcolatori RISC; a macchine più semplici, infatti,

corrisponde una maggiore efficienza dovuta proprio alla semplicità architetturale. Tuttavia, le macchine CISC sono ancora presenti in quantità ed attualmente si ha una coesistenza delle due architetture.
Tutti i computer che utilizzano processori Pentium ed AMD sono CISC, tutti i computer che utilizzano processori Motorola (i computer della Apple) sono RISC.

Vantaggi delle macchine RISC: non utilizzano la microprogrammazione (utilizzando istruzioni semplici, il decodificatore non è troppo complesso); permettono una più semplice ed efficace canalizzazione (le tre fasi di caricamento, decodifica, esecuzione richiedono lo stesso tempo, visto anche il fatto che la microprogrammazione non è necessaria e perchè è più facile fare in modo che le tre fasi non interferiscano fra loro); sono più facili da realizzare; spesso si tende a non utilizzare molte delle istruzioni complesse delle macchine CISC; sono più veloci; permettono una portabilità del codice più agevole (essendo più semplici delle CISC, permettono ai vari costruttori di realizzare macchine più simili e con set di istruzioni molto vicini fra loro).

2. Macchina RISC di esempio

Un esempio di macchina RISC si studia nel Laboratorio di questo corso: la macchina MIPS.

3. Macchina CISC di esempio

3.1 Introduzione

Come esempio verrà presa la famiglia di calcolatori PDP 11, prodotta negli anni settanta dalla Digital.

Per approfondire l'argomento si consigliano i seguenti siti:
http://simh.trailing-edge.com/pdp11.html
http://www.conknet.com/~w_kranz/pdp11/pdp11.htm
http://www.pdp11.org/ .

3.2 Caratteristiche generali

Questa famiglia di calcolatori aveva le seguenti caratteristiche:

- clock pari ad un Mhz
- lunghezza della parola corrispondente a 16 bit, con la possibilità di indirizzare anche metà parola: gli indirizzi pari si riferivano all'inizio delle parole, mentre quelli dispari si riferivano alla metà delle parole
- sistema di interconessione a bus - fu la prima macchina ad implementarlo
- memoria centrale di 32 kilo-parole organizzata in 4 pagine per un totale di 256 kbyte di spazio
- una parte di memoria dedicata allo stack
- numeri interi rappresentati in complemento a 2; numeri naturali rappresentati in modo standard binario
- interruzioni/eccezioni vettorizzate - fu la prima macchina ad implementare questa tecnologia.
  Permetteva di gestire più efficientemente le periferiche a quel tempo più utilizzate: i *terminali*. Allora, infatti, c' era un solo grande calcolatore centrale, detto *mainframe*, e tutti gli utenti avevano un proprio terminale che includeva solo un monitor ed una tastiera

collegati al mainframe. Ogni volta che ad es. su un qualsiasi terminale veniva premuto un tasto, partiva una interruzione verso il mainframe. Usando le interruzioni vettorizzate non c' era bisogno di ricercare chi avesse lanciato l' interruzione, evitando di rallentare la macchina come succedeva invece con il polling

- 8 registri da 16 bit ad uso generico nella CPU, anche se il sesto conteneva il puntatore di catasta ed il settimo conteneva l'indirizzo puntato dal P.C.

- un registro di stato detto PSW (processor status word) organizzato come in figura:

-	N Z V C	TRAP (1 bit)	PRIORITÀ (3 bit)	MODO FUNZIONAMENTO (2 bit)	-

con N, Z, V e C risultati logici attuali, TRAP t.c. ogni volta che il programmatore lanciava una eccezione si alzava fino al ritorno dal programma di servizio (serviva per il debugging), PRIORITÀ del processo attuale (da 0 a 7 con 0 il massimo), scritto dal programma in esecuzione, che stabiliva da chi questo poteva accettare interruzioni\eccezioni, MODO DI FUNZIONAMENTO comprendente tre opzioni: sistema operativo, utente e intermedio; il set di istruzioni veniva modificato non permettendo l' accesso ad istruzioni pericolose (come lo spegnimento della macchina) all' utente

- l' I\O tramite l' estenzione all' esterno della memoria centrale

3.2 Modi di indirizzamento

I modi di indarizzamento possibili erano (il numero che li identifica veniva messo nel campo "modo di indirizzamento" delle istruzioni):

- modo 0 - a registro - nel registro c' era il dato - notazione: Rn

- modo 1 - a registro differito - nel registro c' era l' indirizzo del dato - notazione: (Rn)

- modo 2 - ad autoincremento - nel registro c'era un offset e, ogni volta che si accedeva al dato, il contenuto del registro veniva incrementato di una parola o di una mezza parola, a seconda che si operasse su parole o su mezze parole; l'offset andava sommato al contenuto di un certo registro indice - notazione: (Rn)+

- modo 3 - ad autoincremento differito - nel registro c' era l' indirizzo di una cella di memoria in cui si trovava un offset e, ogni volta che si accedeva al dato, il contenuto del registro veniva incrementato di una parola o di una mezza parola a seconda che si operasse su parole o su mezze parole; l'offset andava sommato al contenuto di un certo registro indice - notazione: @(Rn)+

- modo 4 - ad autodecremento - nel registro c'era un offset e, ogni volta che si accedeva al dato, il contenuto del registro veniva decrementato di una parola o di una mezza parola, a seconda che si operasse su parole o su mezze parole; l'offset andava sommato al contenuto di un certo registro indice - notazione: -(Rn)

- modo 5 - ad autodecremento differito - nel registro c' era l' indirizzo di una cella di memoria in cui si trovava un offset e, ogni volta che si accedeva al dato, il contenuto del registro veniva decrementato di una parola o di una mezza parola a seconda che si operasse su parole o su mezze parole - notazione: @-(Rn)

- modo 6 - ad indice - nel registro si trovava un offset da sommare al contenuto di un registro indice - notazione: offset(Rn)

- modo 7 - ad indice differito - nel registro si trovava un indirizzo di memoria in cui c'era un offset da sommare al contenuto di un registro indice - notazione: @offset(Rn)

Sul registro 7 della CPU (P.C.) si potevano utilizzare solo 2,3,6,7. Nel paragrafo 3.8 si vedrà come fosse possibile utilizzare anche gli indirizzamenti immediato ed assoluto grazie a questa caratteristica del registro 7.

3.3 Istruzioni a due operandi d'ingresso

Il formato di queste istruzioni era il seguente:

Se il primo bit era 1, segnalava che gli operandi erano mezze parole, se era zero segnalava che erano word: poiché era possibile indirizzare sia mezze parole che parole, tutte le istruzioni potevano lavorare su entrambe i formati.
I 3 bit seguenti indicavano l' operazione che si voleva eseguire (codice dell' istruzione). Il massimo numero di istruzioni a due operandi d'ingresso possibile era quindi di otto (2^3).
Nei successivi 6 bit si trovava il primo operando. Questi 6 bit venivano divisi in due parti da 3 bit ognuna: nei primi veniva indicato il modo di indizzamento (infatti c' erano otto modi - 2^3) mentre negli ultimi veniva indicato il registro a cui si riferiva il modo di indirizzamento (infatti c' erano otto registri nella CPU - 2^3).
Gli ultimi 6 bit erano analoghi ai precedenti, ma contenevano il secondo operando.
Nel caso di operazioni con un valore di ritorno, il risultato veniva messo in questi ultimi 6 bit, sovrascrivendo il secondo operando.

Le istruzioni avevano i seguenti codici mnemonici:
MOV: spostava una word da un posto ad un' altro
CMP: comparava i due operandi, mettendo il risultato nei bit logici di PSW
BIT: and logico
BIC: mascheramento - azzerava tutti i bit del I operando che corrispondevano agli uno del II operando (che era la maschera)
BIS: or esclusivo (xor)
ADD: somma e sottrazione
Tutte queste istruzioni operavano su parole.
Aggiungendo una B alla fine del nome si ottenevano istruzioni operanti su mezze parole.

3.4 Istruzioni ad un operando d'ingresso

Il formato di queste istruzioni era il seguente:

Se il primo bit era 1, segnalava che l' operando era una mezza parola, se era zero segnalava che era di una word: poiché era possibile indirizzare sia mezze parole che parole, tutte le istruzioni potevano lavorare su entrambe i formati.
I 9 bit seguenti indicavano l' operazione che si voleva eseguire (codice dell' istruzione), che quindi erano molte di più delle precedenti (2^9).
Negli ultimi 6 bit si trovava l' operando. Questi 6 bit venivano divisi in due parti da 3 bit ognuna: nei primi veniva indicato il modo di indizzamento (infatti c' erano otto modi - 2^3) mentre negli ultimi veniva indicato il registro a cui si riferiva il modo di indirizzamento (infatti c' erano otto registri nella CPU - 2^3).
Nel caso di operazioni con un valore di ritorno, il risultato veniva messo in questi ultimi 6 bit, sovrascrivendo l' operando.

Le istruzioni avevano i seguenti codici mnemonici:
JMP: salto incodizionato con un offset di 6 bit
SWAB: scambiava le due mezze parole di una parola fra loro
CLR: azzerava l' operando
COM: ???
INC: incrementava di uno l' operando
DEC: decrementava di uno l' operando
NEG: negava l' operando complementando i suoi bit
ADC: addizionava il riporto di una somma precedente, contenuto nei bit dei risultati logici di PSW; serviva per addizionare stringhe più lunghe di una word
SBC: sottraeva il prestito di una differenza precedente, contenuto nei bit dei risultati logici di PSW; serviva per sottrarre stringhe più lunghe di una word
TST: testava la positività o negatività dell' operando rappresentato in Ca2, controllando il MSB; il risultato veniva scritto nei risultati logici in PSW
ROR: shift logico a destra di una posizione (i bit che uscivano da un lato rientravano dal lato opposto: serviva per fare operazioni bit a bit)
ROL: shift logico a sinistra di una posizione (i bit che uscivano da un lato rientravano dal lato opposto: serviva per fare operazioni bit a bit)
ASR: shift aritmetico a destra di una posizione (i bit che escivano da dx NON rientravano da sn, perchè a sn entrava lo stesso bit che c' era per primo, per mantenere il segno: serviva per dividere per multipli di due l' operando)
ASL: shift aritmetico a sinistra di una posizione (i bit che escivano da sn NON rientravano da dx, perchè a dx entravano zeri, per mantenere il numero: serviva per moltiplicare per multipli di due l' operando)
SXT: estendeva il segno per i numeri in Ca2 (ad es. passando da 8 bit a 16 bit, ripeteva a sinistra il bit più significativo fino a coprire tutti i 16 bit della word)

Aggiungendo una B alla fine del nome si ottenevano istruzioni operanti su mezze parole.

3.5 Istruzioni di salto

Il formato di queste istruzioni era il seguente:

8 bit	8 bit

I primi otto bit rappresentavano il codice dell' istruzione, gli ultimi otto bit rappresentavano l' offset (quindi indirizzamento relativo). Si potevano pertanto effettuare salti a +127 e –128 posizioni dal contenuto del P.C. .

Le istruzioni avevano i seguenti codici mnemonici:
BR: salto incodizionato con un offset di 8 bit
BNE: saltava se nel bit Z dei risultati logici di PSW c' era 0 (non uguale)
BEQ: saltava se nel bit Z dei risultati logici di PSW c' era 1 (uguale)
BGE: saltava se maggiore o uguale
BLT: saltava se minore
BGT: saltava se maggiore
BLE: saltava se minore o uguale
BLP: saltava se positivo
BMI: saltava se negativo
BHI: saltava se maggiore o uguale per naturali non in Ca2
BLOS: saltava se minore o uguale per naturali non in Ca2
???: saltava se overflow (Ca2)
???: saltava se trabocco (naturali non in Ca2)
JSR: saltava a subroutine (salvando P.C. nello stack)

RTS: ritornava da subroutine (ripristinando P.C. dallo stack)

3.6 Istruzioni di trap

Il formato di queste istruzioni era il seguente:

8 bit	8 bit

I primi otto bit rappresentavano il codice dell' istruzione, gli ultimi otto bit rappresentavano l' opportuna locazione del vettore delle interruzioni (che quindi aveva 2^8 posizioni).

Le istruzioni avevano i seguenti codici mnemonici:
EMT: saltava alla specificata locazione del vettore delle interruzioni (salvando registro di stato e P.C.) - veniva raccomandato di non utilizzare questa istruzione perchè riservata alla casa costruttrice per costruire il software di base
TRAP: come la precedente ma utilizzabile liberamente; separando le due istruzioni si evitavano conflitti fra software di base e programmi utente; nulla vietava cmq di utilizzare EMT
BPT: saltava alla posizione 14 del vettore delle interruzioni, in cui si trovava l' indirizzo del programma di servizio scritto dal produttore che dopo ogni istruzione eseguita da parte del programma utente interrompeva la macchina (usato per debugging)
IOT: saltava alla posizione 20 del vettore delle interruzioni, in cui si trovava l' indirizzo del programma di servizio scritto dal produttore che gestiva l' I\O con interruzioni
RTI: ritornava dal programma di servizio (ripristinando registro di stato e P.C.)

3.7 Istruzioni di controllo

HALT: arrestava il calcolatore
HOP: perdeva un ciclo di clock senza far nulla (utile per sincronizzazione / temporizzazione)
WAIT: arrestava la macchina in attesa di una interruzione: appena veniva lanciata una qualsiasi interruzione, il calcolatore ripartiva
RESET: metteva a zero tutti i registri interni ed esterni (periferiche) del calcolatore

3.8 Istruzioni a formato variabile

Non esistevano istruzioni "native" a formato variabile.
Si è detto tuttavia che il settimo registro della CPU conteneva l'indirizzo puntato dal P.C. .
Questo permetteva di creare istruzioni a formato variabile per realizzare gli indirizzamenti assoluto e immediato. Analizzando le istruzioni sinora trattate, infatti, si nota che questi due tipi di indirizzamento non erano possibili, poichè nel campo dell' istruzione contenente gli operandi non ci sarebbe stato sufficiente spazio per mettere direttamente il valore di questi (sarebbero serviti 16 bit per ogni operando).
Ad esempio scrivendo:

```
ADD (R1), (R2)
```

gli operandi non erano contenuti nella word di ADD, dove si trovavano, invece, soltanto i nomi dei registri che li contenevano.
Scrivendo:

```
ADD (R7)+, (R7)+
```

veniva comunicato alla macchina che gli operandi si trovavano nelle due word successive a quella dell' istruzione ADD. Si otteneva, in pratica, una istruzione di tre word di cui una contenente ADD e le altre due contenenti gli operandi.
Ovviamente si poteva utilizzare questo trucco anche su di un solo operando, così da ottenere una istruzione di 2 word:

```
ADD (R1), (R7)+
```

PERIFERICHE

1. Introduzione

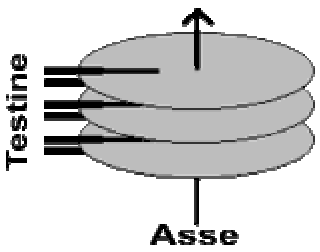
Sono dispositivi collegati al calcolatore tramite il canale di I\O.

2. Unità disco

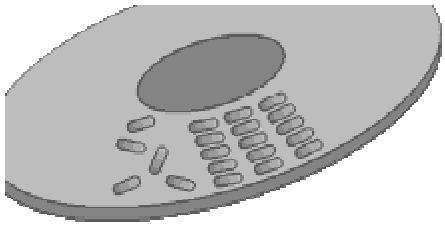
Supporto magnetico in grado di conservare nel tempo stringhe binarie e restituirle su richiesta del calcolatore.
Caratteristiche di questa periferica sono l' elevata capienza, la possibilità di conservare informazioni anche a calcolatore spento e in assenza di corrente, nonchè la latenza molto maggiore rispetto a quella della RAM (per via dell' architettura costruttiva).

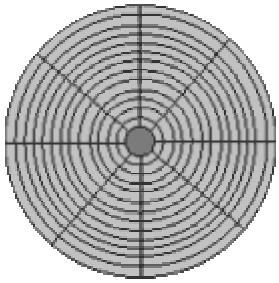
All' interno dei calcolatori, dunque, convivono vari tipi di memoria, che possono essere suddivisi in maniera generale in *memoria volatile* e *memoria permanente*. Questa suddivisione si riflette nello studio della psicologia del pensiero, dove viene teorizzata la presenza di due tipi di memoria anche all' interno del cervello umano. Una memoria a breve termine sarebbe, infatti, responsabile della capacità di tenere a mente un numero di telefono dopo una breve lettura per consentire di digitarlo sulla tastiera del telefono, mentre quella a lungo termine si manifesterebbe nei ricordi e nella capacità di trattenere una grande quantità di informazioni per un lungo periodo di tempo, tipica del cervello umano. Dalla collaborazione tra questi due tipi di memoria, sostengono gli studiosi, si realizzano la comprensione del discorso e molte altre facoltà complesse. Questo tipo di suddivisione funziona perfettamente anche in ambito informatico, dove esistono memorie di massa, che possono conservare le informazioni in modo permanente senza grossi problemi di spazio, e memorie volatili, come registri e RAM, che sono deputate alle funzioni operative ma a cui non è richiesto di conservare i dati anche dopo lo spegnimento del computer.

Questo tipo di unità viene realizzata tramite dischi concentrici, detti *piatti*, in grado di ruotare a velocità variabile attorno ad un asse centrale comune. I piatti vengono ricoperti con un particolare materiale magnetico. Su di esso sono presenti numerosi magneti disposti inizialmente in maniera casuale, ciascuno con un polo positivo "+" ed uno negativo "-". Accoppiando i magneti due a due, a seconda dell'orientamento dei poli è possibile rappresentare i bit 0 ed 1. In particolare con lo schema "+" "-","+" "-" si rappresenta uno zero, mentre con "+" "-", "-" "+" si rappresenta un uno.



Opportune testine (due per ogni disco: una per la faccia superiore, una per quella inferiore) sono in grado sia di allineare i microscopici magneti, che di leggere il loro campo.





Le unità disco sono così organizzate: ogni piatto ha due *facce*, ogni faccia è divisa in cerchi concentrici detti *tracce* o *cilindrici* e ogni traccia è divisa in *settori*. Dunque, per accedere ad una certa stringa memorizzata, si usano indirizzi che indicano la faccia, la traccia e quindi il settore dove esso si trova. Fatto ciò, il circuito che controlla il funzionamento del disco seleziona la testina, la sposta meccanicamente sulla traccia indicata e poi aspetta che il giusto settore passi sotto di essa per iniziare la lettura o scrittura. Per verificare di aver posizionato la testina nella giusta posizione, all' inizio di ogni settore è ripotato il suo indirizzo. Tale associazione settore - indirizzo avviene al momento della *formattazione* del disco.

Trasferire (scrivere o leggere) un dato alla volta, però, è poco efficiente, vista la lentezza di questo tipo di periferiche rispetto alla CPU. Per minimizzare gli accessi, quindi, si effettua il trasferimento di un settore alla volta. Il quantitativo di dati che occupano un settore viene detto *record*. Si può quindi dire che viene trasferito un record alla volta. Si noti che il quantitativo di dati che occupano un settore è costante su tutto il disco, nonostante i settori interni siano più corti di quelli esterni. Questo perchè all' interno i dati sono via via più "fitti".

I dati vengono scritti in modo *non contiguo*. Ciò serve per evitare di sprecare spazio inutilmente quando si libera un settore precedente all' ultimo, ma i dati che si devono scrivere sono più grandi del settore. Questa caratteristica pone un nuovo problema: ritrovare tutti i pezzi di dato che interessano. Vengono utilizzate varie tecniche. Una di queste consiste nell' avere memorizzata all' inizio del disco una tabella con l' indirizzo di tutte le parti di ogni dato. Una seconda tecnica consiste nel mettere alla fine di ogni settore l' indirizzo del settore in cui continua il dato. Di tutto questo si occupa il sistema operativo.
Dovendo ricercare i vari pezzi, comunque, si ha una maggiore perdita di tempo. Per questo esiste la possibilità di riservare zone di disco alla memorizzazione contigua. In queste zone andranno tutti quei dati che necessitano di un accesso particolarmente veloce, come ad esempio gli swap file per la memoria virtuale.

Poichè ogni settore contiene molte word, quando si trasferisce un settore viene generato un notevole traffico di stringhe, quindi le unità disco vengono in genere collegate al calcolatore tramite il metodo del DMA.

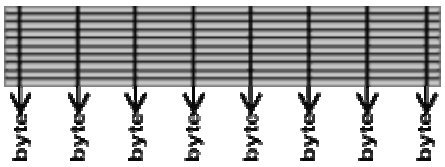
Memoria virtuale: quando il disco è utilizzato per lo swapping, le pagine sono grandi proprio un record, per comodità.

Esistono due classi di unità disco: i *dischi morbidi*, in cui le testine sono a contatto con i piatti, e quelli *rigidi*, in cui le testine volano a pochi micron di distanza dai piatti, sostenute dal cuscino d' aria che si crea grazie alla rotazione degli stessi piatti.
Fra le due tecnologie, la prima è più lenta a causa dell' attrito, ma più resistente, la seconda è più delicata (le testine possono precipitare sul disco - *crash di testine*), ma più veloce (i dischi possono ruotare molto più velocemente).

3. Unità nastro

Supporto magnetico in grado di conservare nel tempo stringhe binarie e restituirle su richiesta del calcolatore.

Vengono realizzati tramite nastri magnetici avvolti in bobine.



Parallele al nastro, scorrono 9 tracce. Sulle prime otto vengono memorizzati i byte, perpendicolarmente al nastro. Sull' ultima, per ogni byte memorizzato, si trova un bit di *parità trasversale*. Dopo un certo numero di byte, per comodità in quantità pari ad un record, si trova un byte di *parità longitudinale*. In questo modo è possibile identificare, come in una matrice, i bit errati e correggerli (basta complementarli).

La *parità* è una tecnica di rilevazione degli errori in cui si stabilisce che il numero di bit di ogni stringa deve essere pari. Per fare ciò, si aggiunge un bit alla stringa; questo prende il valore zero se già c' è un numero pari di bit e prende il valore uno se ce n' è un numero dispari, così da renderli in numero pari. In questo modo, se un numero dispari di bit è errato a causa della smagnetizzazione del nastro, la parità non torna più e l' errore può essere rilevato e corretto. Se però è errato un numero pari di bit, non si riesce a rilevarlo. Per questo si dice che la parità è una tecnica di *protezione leggera*. Esistono, invece, tecniche di protezione dette forti.
C' è da dire, però, che in tutte le tecniche di rilevazione degli errori bisogna aggiungere informazioni ridondanti alle stringhe che si vogliono "proteggere". In genere, più una tecnica è forte e maggiore è l'aumento di informazioni ridondanti. Via via che la protezione aumenta, dunque, lo spazio occupato è sempre maggiore: questo è il prezzo da pagare.
Le tecniche di rilevamento degli errori sono molto usate anche nella trasmissione di dati digitali.

Mentre i dischi vengono utilizzati per l'*achiviazione in linea*, i nastri vengono utilizzati per l'*archiviazione non in linea*.
L' archiviazione non in linea consiste nel memorizzare grandi quantità di dati a cui si accede raramente, ma che interessa conservare per periodi molto lunghi. Vengono per questo utilizzati dispositivi magnetici come i nastri, che hanno il vantaggio di poter essere tolti dal calcolatore e conservati a parte quando non si deve scrivere o leggere da essi. Pur essendo dispositivi magnetici, e quindi soggetti a smagnetizzazione, i nastri dispongono di varie tecniche di rilevazione e correzione degli errori, anche molto più complesse della parità e per questo risultano molto indicati per l'archiviazione non in linea.
Tuttavia, per conservare un archivio non in linea memorizzato su nastro magnetico, bisogna periodicamente eseguire la copia del nastro vecchio su di uno nuovo, così da eliminare gli errori di smagnetizzazione prima che diventino troppi per essere corregibili.
Ulteriore vantaggio dei nastri magnetici è che la scrittura e lettura da questi dispositivi è standardizzata (al contrario delle unità a disco), cioè è indipendente dal calcolatore e dal sistema operativo utilizzato.
L' archiviazione in linea, invece, consiste nella memorizzazione di dati a cui si accede più spesso e che devono comunque essere conservati anche a computer spento.

4. Interfacce di comunicazione

Dispositivi in grado di trasmettere dati da un calcolatore ad un altro.

Attualmente i più diffusi sono i *modem*. Questi sfruttano le linee telefoniche per trasferire word di bit.
Le linee telefoniche sono pensate per trasmettere la voce umana (onde sonore), in particolare quella parte di frequenze che va da 10 a 3000 Hz circa.
Si è stabilito che ad una frequenza di circa 900 Hz corrispondesse il bit 1 e che ad una frequenza di circa 950 Hz corrispondesse il bit zero. L' orecchio umano percepisce i due suoni in questione come fischi, uno più alto e uno più basso.
Applicando due filtri che rilevano queste due frequenze entrando in risonanza con esse, è possibile riconvertire le due frequenze in bit. Questo metodo è detto a *modulazione di frequenza* e significa che le onde che sono trasmesse oscillano tra frequenze ben determinate (900 e 950 Hz

in questo caso).
Un vantaggio di utilizzare le onde sonore è che la trasmissione risulta quasi del tutto indipendente dalla potenza del segnale, poichè i filtri comunque entrano in risonanza ed amplificano le due frequenze in questione, anche se esse arrivano molto deboli.

Come visto nel paragrafo precedente, anche per la trasmissione di dati digitali vengono utilizzate diverse tecniche per individuare e correggere errori che possono avvenire durante la comunicazione (ad es. disturbi sulla linea, ecc.).

-VARICELLA martedì 28 maggio 2002-

5. Schermo

-VARICELLA martedì 28 maggio 2002-

6. Stampante

-VARICELLA martedì 28 maggio 2002-

Terminali video: tastiera/monitor

- L'utente inserisce tramite la tastiera dati che vengono trasmessi al calcolatore e possono essere visualizzati su monitor.
- L'unità di scambio è il carattere, formato di solito da 7 (ASCII) o 8 bit.
- Dato che l'interazione è con operatori umani non serve una grande velocità di trasmissione • viene usato collegamento seriale

- Nel caso di ingresso da tastiera quando l'utente rilascia il tasto premuto viene generato un segnale elettrico interpretato dal trasduttore della tastiera e tradotto in una sequenza di bit di codice ASCII e trasmessa al modulo di I/O del calcolatore.
- Dispositivo trasmittente e ricevente devono utilizzare la stessa informazione di temporizzazione (segnale di clock): trasmissione sincrona (segnale unico) o asincrona (due segnali di clock con frequenze simili e opportuni controlli).

- Trasmissione asincrona con tecnica start-stop:

o trasmissione di caratteri alfanumerici codificati con 8 bit, preceduti da un carattere 0, start, e uno o più bit 1, stop.

o la linea di trasmissione è nello stato 1 quando inattiva (rimane a 1 dopo il bit stop).

Stampato martedì 9 luglio 2002

DSI

La Sapienza

???

Versioni aggiornate e corrette di questi appunti vengono periodicamente rese disponibili presso il sito internet:
http://twiki.dsi.uniroma1.it/twiki/view/Users/AlbertoRocca

• 0.00