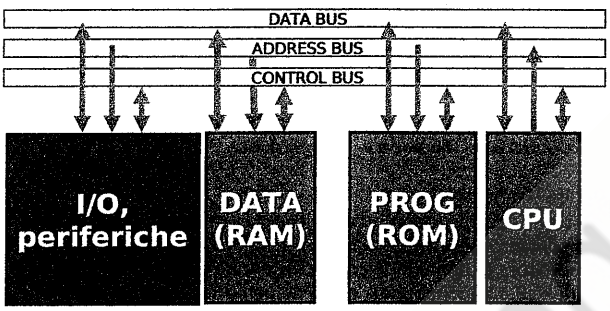
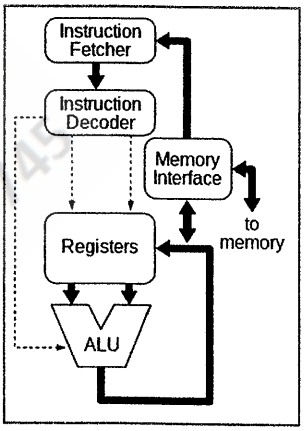
Roberto Roncella mail: roberto.roncella@iet.unipi.it web: roncella.iet.unipi.it tel: 050-2217669 Ricevimento Giovedì 15-18 Venerdi 15-18 dipartimento dell'informazione	Libri: - D.D Givone "Digital principies and design" , McGraw Hill LED 2003, capitoli 3-8 - ibiden appendice A - Milliman A. Grabel, P.Terreni, "Elettronica di Millman", McGraw Hill quarta edizione 2005 Esame Sistemi digitali + Elettronica Analogica —> scritto + orale Test di 5 esercizi da svolgere in 60 minuti, valido per 6 mesi per l'orale. 33 punti, minimo 17,5 per il test.

Come si programma un oggetto elettronico per implementare logiche sofisticate.

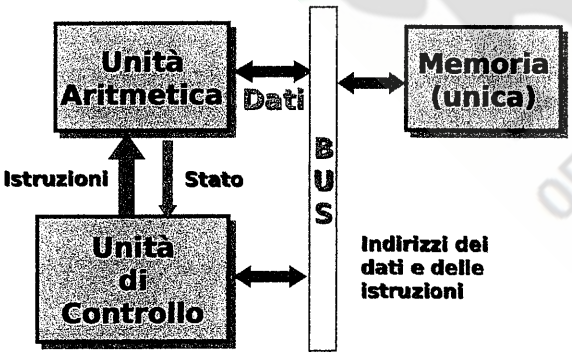


Un microcontrollore è **un chip che implementa tutte le potenzialità di un computer**. All'interno troviamo degli **oggetti in grado di memorizzare informazioni** costituiti o da un'unica memoria in cui sono memorizzate sia le istruzioni sia i dati, oppure da memorie separate o da memorie esterne al microcontrollore. Troviamo poi **un blocco I/O** che contiene tanti **circuiti ausiliari in grado di affrontare problemi specifici** per dialogare con l'esterno. Il blocco funzionale più importante è la **CPU**, più noto come **microprocessore**.
Il microcontrollore è quindi un dispositivo che contiene un microprocessore, una memoria e delle periferiche che gli permettono di interfacciarsi con il mondo esterno e gli permettono di avere tutte le potenzialità di un computer per svolgere attività di controllo e di gestione di una grande gamma di dispositivi.



La **ALU**, l'Unità Aritmetica e Logica, è il cuore della CPU, **l'unico blocco in grado di svolgere operazioni logiche e aritmetiche**. Sono poi presenti dei **registi**, ovvero delle memorie in cui vengono inserite temporaneamente dei dati accessibili alla ALU e su cui poi la ALU va a riscrivere i risultati delle operazioni. La parte superiore del nostro sistema è un interfaccia con le memorie che sa leggere le memorie del circuito. L'**Istruction Fetcher** è una parte fondamentale nella gestione della memoria che **pilota sia la memoria che l'interfaccia, recupera istruzioni, le codifica e le fa eseguire alla ALU.** Il microcontrollore è una macchina dotata di sincronismo, dotata di clock.

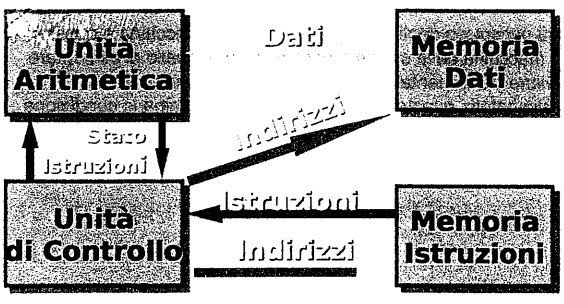
Nel corso degli anni nel realizzar equestri microprocessori ci sono stati varie soluzioni.



Architettura di Von Neumann

Questa architettura è ancora largamente diffusa, ereditata dalle macchine IBM. **La memoria dei dati e la memoria del programma sono in realtà due aree diverse di un unico dispositivo**. Questa soluzione è stata adottata perché prima la memoria era una risorsa preziosa e questo ottimizzava le risorse, fatto che l'ha resa dominante.

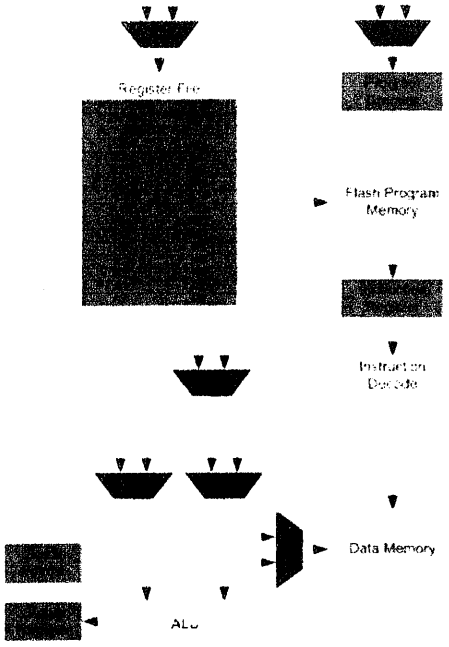
I vantaggi di questa architettura sono che ha una una **complessità ridotta, c'è un unico collegamento tra il processore e la memoria**, cosa molto utile quando prima la memoria e il processore erano due dispositivi fisicamente divisi. Lo svantaggio sta nel fatto che in questo modo **i dati e le istruzioni nella memoria possono essere letti o trascritti una alla volta**: se si legge un'istruzione non si può leggere un dato e se memorizzo un dato non posso leggere istruzioni. Questo limite è diventato intollerabile quando si è creata l'esigenza di usare queste macchine per fare conti su dati che rappresentassero informazioni in tempo reale. Inoltre questa architettura rende il sistema facilmente soggetto ad attacchi virali.



Architettura di Harvard

In questo genere di architettura **il programma e i dati vengono a trovarsi su memorie separate**. La parte di memoria delle istruzioni è in grado di dialogare solo con l'unità di controllo, la memoria dati dialoga soltanto con la ALU. In questo modo **un processore con questa architettura può lavorare con una velocità doppia**. La memoria delle istruzioni contiene il programma da eseguire e in genere non c'è l'esigenza di cambiare continuamente il programma, per questo quasi sempre **le istruzioni sono memorizzare su una memoria non volatile** (ROM) di sola lettura, e per questo motivo il codice che fa lavorare il microcontrollore è detto **Firmware**, inciso in modo permanente nel dispositivo.

L'architettura di Harvard ha maggiori prestazioni poiché può leggere o scrivere istruzioni e dati contemporaneamente e questo si traduce in **un numero superiore di operazioni svolte al secondo**. L'architettura di Harvard diventa particolarmente vantaggiosa quando memoria e CPU sono implementati sullo stesso dispositivo e **possono essere aggiunte più bus dati** in modo da lavorare su più dati alla volta e usare un bus singolo per le istruzioni anche molto potenti.



Schemi a Blocchi

Nello schema a blocchi il produttore racconta la struttura del microcontrollore, indicando al CPU, la memoria dei dati e del programma, le periferiche che svolgono determinate funzioni e i circuiti ausiliari digitali.

I blocchi neri si chiamano **multiplexer** e sono dei selettori che **instradano i dati** che provengono da varie fonti. **Program counter** fa parte dell'interfaccia di memoria, decide l'indirizzo dell'istruzione da eseguire. Questo componente **contiene l'ordine delle operazioni da eseguire, in questo modo la macchina è automatizzata per eseguire una dopo l'altra le istruzioni del programma**. Tuttavia il Program Counter può essere modificato in modo che il processore possa eseguine operazioni anche non in sequenza o possa modificare l'ordine delle operazioni sulla base dell'esito delle istruzioni.

Periferiche di un microcontrollore

A fianco del microprocessore sono presenti varie **periferiche**, blocchi hardware specifici che aiutano a svolgere compiti particolari. Sono presenti blocchi con funzioni analogiche, ma anche blocchi che svolgono compiti digitali o altre funzioni particolari come contare il tempo trascorso tra due eventi. In particolare quest'ultimo blocco è importante perché se si impegnai il microprocessore nella misura di un intervallo, durante questo intervallo esso non può eseguire nessun altra operazione. La presenza di un circuito apposito a cui si delega il compito di misurare intervalli di tempo, non blocca la macchina in questo compito.

Una stessa famiglia di microcontrollori condivide la stessa CPU, ciò che differisce tra i vari dispositivi è il tipo e la quantità di memoria e di periferiche.

Come dialoga la CPU con le periferiche?

Inizialmente si era pensato di prevedere istruzioni apposite con cui il microprocessore poteva dare informazioni alle periferiche,ma questo metodo è stato subito abbandonato perché complicava il linguaggio, perché periferica doveva avere un proprio set di istruzioni. La soluzione standard adesso utilizzata è quella di **far vedere al microprocessore le periferiche come periferiche di memoria**. Leggere i valori riportati da una periferica significa leggere una data locazione di memoria. **Le periferiche vengono mappate come come parte di memoria del microprocessore.**

Tipi di periferiche

- **Pin di I/O**: sono i pin elettrici che il dispositivo ha sulla periferica. In genere la CPU non ha modo di imporre al pin un valore o di leggere direttamente un valore dal pin. Questa periferica permette uno scambio di informazioni digitali sul singolo pin, in particolare permette a uno stesso contatto di essere o di ingresso o di uscita. Questa periferica permette ad esempio di interrompere una fase di flusso del programma a seconda dei valori riportati sul pin.
- **Timer e Contatori**: misurano il tempo e la frequenza e possono generare segnali periodici. In particolare possono **generazione di segnali con ciclo di lavoro variabile** (PWM), ovvero un'onda rettangolare attraverso il quale il microprocessore può ad esempio controllare carichi.

$$\%\ \text{duty cycle} = \frac{T_1}{T}$$

- **Periferiche di conversione** ad esempio le periferiche che convertono un segnale analogico in un segnale digitale (il comparatore analogico) e viceversa.
- **Interfacce di comunicazione standard** utilizzano protocolli di comunicazione per controllare informazioni in uscita e in entrata (RS232, USB).

Circuiti ausiliari

- **Generatore di clock**
- **Supervisore dell'alimentazione**: oggi ha un grandissima importanza gestire l'assorbimento di corrente si cerca di ottimizzare.
- **Circuito di reset**
- **Supervisore della modalità operativa**
- **Watchdog**

Il circuito di WatchDog riguarda l'affidabilità del firmware e in particolare il controllo dei bug. Il più pericoloso degli eventi è il blocco della macchina e questo circuito serve per ridurre l'effetto dannoso di questi avvenimenti. Se prevedo tra i compiti che la macchina deve svolgere periodicamente quello di alimentare un circuito che tende continuamente a scaricarsi, se la macchina si blocca questo circuito non viene più alimentato e a questo punto interviene. Può intervenire in un modo solo resettandola. Se passa un tempo limite senza che il circuito venga alimentato, questo resetta la macchina e elimina la condizione che ha potato la macchina sbloccarsi.

Linguaggio dei microcontrollori

Come si programmano queste macchine?
Il livello più basso, il modo più elementare per riuscire a controllare un microcontrolore è conoscere e programmare con il suo linguaggio macchina, il linguaggio Assembly.

È un linguaggio comprensibile da un umano, che si presenta con una sintassi e un lessico fatto per il progettista che lo usa. La particolarità di questo linguaggio è che **gli elementi del linguaggio corrispondono agli elementi fisici dell'architettura del processore**, le istruzioni sono legate alle istruzioni che il processore può eseguire. **Il linguaggio assembly non viene compilato, ma assemblato, semplicemente tradotto in numero, il rapporto tra linguaggio Assembly e ciò che viene scritto nel microcontrollore è esattamente corrispondente. Con l'assembly si ha anche la percezione del tempo poiché c'è una relazione diretta tra le istruzioni eseguite e il tempo impiegato per eseguirle.**
Ogni famiglia di microcontrollori ha il suo Assembly, è il motivo per cui la conoscenza approfondita dell'Assembly è relativa al tipo di oggetto su cui si lavora.

```
.org 100                    ;scrivi dall'indirizzo 100
MULT:   push    CNT         ;salva il contatore
        clr     OUT1        ;cancella la parte alta del risultato
        mov     OUT0,IN0    ;copia il moltiplicatore nella parte bassa
        ldi     CNT,8       ;inizializza il contatore
        lsr     OUT0        ;shifta il moltiplicatore a destra
M1:     brcc    M2          ;somma solo se C set
        add     OUT1,IN1    ;somma il moltiplicando a OUT1
M2:     ror     OUT1        ;ruota a destra la parte alta del risultato
        ror     OUT0        ;ruota a destra la parte alta del risultato
        dec     CNT         ;decrementa il contatore (non tocca C)
        brne    M1          ;ripeti per 8 volte
        pop     CNT         ;ripristina il contatore
        ret
```

Questo è un classico listato Assembly. C'è una certa rigidità di tipo formale sulla sintassi, legato a un hardware ben preciso con un set ben limitato di istruzione. C'è poi una parte scritta in italiano che sono i commenti. È un linguaggio di tipo imperativo in cui le istruzioni danno dei comandi diretti.
Ci sono delle righe diverse da tutte le altre che iniziano con un identificatore, un due punti e poi l'istruzioni. La normale riga ha il comando operativo e poi degli altri argomenti, ovvero i comandi e i numeri da dare alla ALU.

Elementi del linguaggio

L'ossatura del linguaggio Assembly sono **le istruzioni con i loro operandi**. Ciascuno di questi oggetti corrisponde a un ben preciso codice, esiste un vocabolario dove ogni istruzione è tradotta con un preciso numero con cui si traduce l'istruzione per il microprocessore, la traduzione si può fare anche a mano. **L'identificatore che segue ci dice quale è il contenitore o il valore con cui l'istruzione agisce**.
Qui si usano identificatori qualunque ma i registri all'interno del microcontrollore hanno un nome proprio che può essere usato, nel nostro Assembly hanno un nome proprio che va da R0 a R31.

Quali sono gli operandi e come fa CPU a recuperarli è un tema delicato.

Il nome seguito dai due punti è l'**etichetta**. Sono informazioni ridondanti perché quando scrivo un programma in Assembly ogni volta che scrivo un'istruzione so esattamente che spazio occupa e so esattamente in che indirizzo è localizzata tale istruzione nella memoria dei programmi. Per il programmatore fare questo può essere impegnativo, ma conoscere gli indirizzi è fondamentale. Per evitare di fare i conti degli indirizzi, si mettono dei label, ovvero **un segnaposto che va a assumere il numero dell'indirizzo**.

I commenti sono indispensabili per la comprensione del programma. È indispensabile se si lavora in ambito condiviso ed è indispensabile per lo stesso programmatore per capire cosa fare.

La prima riga inizia con il punto, è il caso tipico del **metacomando, una direttiva che serve per dire qualcosa all'assemblatore**, può fare comodo dire qualcosa a chi tradurrà in modo che la traduzione segua certi criteri. La .ORG da all'assemblatore l'indirizzo di partenza su cui scrivere le istruzioni. Le costanti .EQU possono far comodo se definite all'inizio del codice in modo che tutti i numeri possono essere utilizzati semplicemente richiamando il nome.

Tipologia di memoria

A seconda dell'utilizzo che ne facciamo le memorie assumono diverse configurazioni. **In base al tipo di informazioni che andranno a contenere, esistono leggi diverse che associano ogni sequenza di bit ad un particolare valore** che il programmatore intende gestire. All'interno del microcontrollore troviamo:

- ***Memoria di programma*** dove viene inserita la traduzione del listato Assembly. Ogni istruzione occupa tipicamente una riga, la quale ha un suo indirizzo specifico. Tale memorie si chiamano spesso **memorie ad accesso casuale**, poiché è possibile leggerle e scriverle accedendo ad una qualsiasi di queste righe. L'indirizzo della sezione di memoria di programma che si vuole leggere è data dal **program counter**.
- ***Memoria dei dati interna*** dove sono inserite tutte le informazione su cui si lavora. La memoria dei dati può essere divisa in più blocchi e spesso anche in più oggetti fisicamente divisi. Tra questi oggetti ce ne sono alcuni chiamati chiamati **registi interni** per utilizzi generici, altri sono detti **registi di I/O** nella quale sono contenute informazioni e parametri riguardanti le periferiche. Alcune locazioni dello spazio di I/O può essere usato dal microprocessore come spazio di appoggio per contenere informazioni a breve termine, come il risultato dell'ultima operazione o informazioni più semplici chiamate Flag.
- ***Memoria dati estesa*** comprende tutto il restante spazio di memoria utile per la soluzione dei vari problemi. Tale memoria può essere estesa aggiungendo chip esterni.

Questi contenitori contengono normalmente delle cifre binarie poiché **ogni tipo di informazione può essere rappresentata usando un opportuno numero di bit**. La traduzione di una sequenza di bit in ciò a cui è legata avviene utilizzando apposite **leggi di rappresentazione**, che sono a loro volta contenute all'interno della memoria. **Dire che il microprocessore riconosce tali dati significa che ha delle istruzioni che elaborano direttamente queste grandezze**.
I valori rappresentabili all'interno del microcontrollore sono relativamente limitate. Il microcontrollore riconosce solo determinate tipologie di grandezze:

- ***Valori di tipo istruzione***, un valore numerico associato a un codice mnemonico (sigla listato Assembly) che rappresenta un'istruzione. Nel nostro microcontrollore in genere le istruzioni sono a 16, 32 o 48 bit, memorizzate su registri della lunghezza di 16 bit.
- ***Valori di tipo Byte o Word***: il microprocessore riconosce valori rappresentabili su 8 bit per la maggior parte delle istruzioni. In certi casi riesce a gestire e fare operazioni con grandezze rappresentate su 16 bit. Questi valori posso essere interpretati in modo più semplice, come **cifre binarie che seguono una notazione posizionale**. Sa rappresentare i numeri con segno, un certo numero di numeri relativi, in genere con la tecnica del complemento al due.
- ***Valori di tipo carattere:*** rappresentati utilizzando in genere 8 bit.
- ***Valori di pio indirizzo:*** nella machina ci sarà bisogno di gestire non solo i valori ma anche gli indirizzi, che sono dei numeri.

- ***Valori di tipo salto:*** mi dice di quanto devo modificare un indirizzo per ottenerne un altro.
- ***Valori di tipo flag***: è 1 bit che mi indica la presenza o l'assenza di una data informazione.

Memoria di programma

Address	Value (*n*)
0	Instr #1 (1 W)
1	Instr #2 (1 W)
2	Instr #3 (2 W)
4	Instr #4 (1 W)
5	Instr #5 (3 W)
7	

Contiene tutti i valori di tipo istruzione, in generale contiene **m parole di n bit**. Ogni locazione è individuata da un indirizzo costituito da log2(m) bit.
Il nostro microcontrollore si chiama Xmega256A3BU ha istruzioni la cui dimensione è 16 bit. Il numero 256 indica la memoria di programma, in particolare ci indica che il microcontrollore ha 256Kb di memoria, utile per contenere 128mila istruzioni. In un microcontrollore c'è un vincolo fisico alla dimensione della memoria di programma perché **nasce per eseguire una sola applicazione** ed esegue solo quella, gestisce un unico dispositivo. La struttura di un programma è in genere costituita da un ciclo infinito in cui si leggono dei sensori, si elabora e poi si predispongono delle uscite. Affinché l'apparecchio sia reattivo questo ciclo deve avvenire più o meno velocemente e per far ciò devo elaborare il ciclo in modo che le istruzioni vengano eseguite in un certo tempo. La velocità massima del microcontrollore è bassa, il nostro va a 32Mhz al massimo, velocità che permette di svolgere tutte le operazioni in un tempo accettabile e che permette a sua volta di limitare i consumi energetici.

Codice operativo	Op destinazione
Op sorgente	

Il valore istruzione è un record articolato, costituito da più campi che variano per numero e dimensioni. Non c'è nessuna legge matematica dietro al modo in cui è strutturato il valore istruzione, ma solo la volontà del progettista. Nei 16 o 32 bit le le informazioni relative all'istruzioni sono divise in diversi campi. Ci sarà sempre un campo detto **Codice Operativo**, che comunica al processore che cose deve fare (somma, sottrazione, operazione logica), dice al processore come deve impostare la ALU. **Il resto dell'istruzione serve a spiegare al microprocessore quali sono gli operandi su cui esegue l'istruzione.** Il codice operativo è strettamente legato al codice mnemonico mente gli altri campi sono legati agli operandi. Il problema di individuare gli operandi su cui il codice operativo agisce è un problema detto dell'**idirizzamento**.

Registri interni

Sono contenuti all'interno dell'elaboratore e sono in generale le motorie più accessibili. Nella nostra macchina sono tanti contenitori da 1 byte nominati da un numero compreso tra zero e 28 che segue la lettera R e a cui è possibile assegnare nomi simbolici. La legge di rappresentazione dei dati inediti in questi registri dipende dall'istruzione che agisce su tali registri. **I registri interni sono gli unici che lavorano con la ALU**.
La dimensione del registro è di 8 bit ma la macchina può lavorare anche su **aggregazioni di byte**, in modo che per certi codici operativi si possa gestire dei multipli di byte. Un operazione tipica che richiede più di 8 bit è quella di **puntamento**, un istruzione che mi permette di far riferimento a un valore contenuto nella memoria estesa in un certo indirizzo. Per gestire gli indirizzi della memoria estesa, che in queste macchine può arrivare fino a 64.000, si accorpano 2 byte e si crea un unità a 16bit che conterrà l'indirizzo della cella che mi interessa.

Registri di I/O

I contenitori di I/O contengono tutto quello che serve alla gestione delle periferiche. Se ad esempio è presente una periferica associata ai conduttori esterni, può essere utile avere dei registri che mi indichino se tali pin sono settati come ingressi o uscite, e dalla quale posso eventualmente leggere i valori. Nel manuale del prodotto sono indicati questi registri nella descrizione della periferica specifica.

Memoria dati estesa

La memoria estesa è un insieme di byte con il loro indirizzo che posso utilizzare a piacimento. La ALU può lavorare solo e sempre con i registi interni per cui le uniche operazioni consentite con la memoria dati estesa sono operazioni di trasporto, ovvero operazioni di lettura e di scrittura. Un dato può essere letto dalla memoria estesa e trascritto su un registro interno, dopo di che la ALU lo può utilizzare per svolgere una data operazione il cui risultato viene nuovamente inserito in un registro interno che potrà poi essere copiato nella memoria estesa.
La ALU ha bisogno di contenitori di memoria in prossimità che siano molto veloci. La connessione con al memoria estesa sarà sempre lenta perché mediata da una logica di indirizzamento che richiede tempo e che penalizzerebbe troppo il tempo di esecuzione del programma.

Registri specializzati

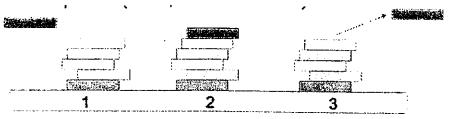
Alcuni registri interni hanno funzioni particolari, possono ad esempio essere legati al meccanismo di funzionamento del processore, ad esempio il *program counter*, oppure dare informazioni sullo svolgimento del programma, il *registro di stato* costituito da una collezione di falg, o ancora permettono di realizzare particolari strutture dati, lo *Stack Pointer.*

- ***Il registro Program Counter*** è un contenitore che **contiene un valore indirizzo che specifica l'indirizzo di un istruzione**, in particolare viene strutturato in modo da indicare l'indirizzo dell'istruzione che la machina sta per eseguire da prelevare dalla memoria di programma. La dimensione del Program Counter è legata alla dimensione della memoria di programma. Il nostro microprocessore ha una memoria di programma da 128Kbyte e il program count ha una dimensione da almeno 17bit. **Questo registro ha la capacità di aggiornarsi automaticamente ad ogni ciclo di esecuzione** dell'istruzione in corso, incrementando il valore dell'indirizzo. **Questo registro ha necessita di essere inizializzato** e tale inizializzazione è gestita a livello hardware, poiché se non venisse inizializzato con un valore preciso il programma inizierebbe da un punto a caso della memoria di programma. In genere viene inizializzato con il valore zero, ma può essere diverso e può essere impostato dal programma. **Nel caso in cui il flusso del programma preveda l'esecuzione di istruzioni più lunghe di un indirizzo, esiste un sistema hardware che provvede ad incrementare l'indirizzo di due o tre unità.** È inoltre possibile modificare il contenuto del Program Counter nel corso del programma tramite operazioni di completa riscrittura in modo da effettuare dei salti nell'esecuzione del programma, in genere questo viene fatto sommando un numero all'indirizzo contenuto.

- ***il Registro di Stato*** contiene una collezione di bit il cui valore individuale è dichiarare l'assenza o la presenza di una data condizione facendo riferimento a l'ultima istruzione eseguita. Mi da informazioni sullo stato della macchina in seguito all'esecuzione di un'istruzione. Si fa riferimento ai bit del registro di stato mediante delle lettere:
  - ***C*** indica il **Carry** ovvero il riporto, una condizione che si genera quando l'esecuzione di un operazione di somma o differenza non è rappresentabile usando le stesse cifre di partenza ed ha senso solo quando si eseguono operazioni tra numeri interi assoluti.
  - **S** indica il **segno**, molto importante se lavoro su numeri relativi.
  - **Z** indica **se il risultato dell'ultima operazione è stato zero**, utile perché dopo un'operazione di differenza mi da un'informazione sull'uguaglianza dei due operandi.

- **P** indica la **parità** ovvero mi dice se il risultato dell'ultima operazione ha un numero pari di uni. Serve come strategia di affidabilità delle comunicazioni perché se consideriamo la possibilità che un operazione di comunicazione abbia comportato un singolo errore, questo ha come effetto di alterare la parità.

- ***Il registro Stack Pointer*** **contiene un indirizzo della memoria dati e viene usato in combinazione di operazioni di scrittura e lettura**. Viene utilizzato per creare nella memoria estesa una particolare struttura dati dinamica, la **Pila** o **stack**. L'accesso alla pila è particolare poiché l'ultimo dato inserito è il primo dato che è possibile estrarre. Per ottenere questo meccanismo il registro è fatto in modo che un'operazione di scrittura provochi un decremento del puntatore e un'operazione di lettura provoca incremento. Quando si inserisce un dato nella pila, il dato viene scritto e successivamente si decrementa il puntatore, quando si preleva un dato, prima si incrementa il puntatore e successivamente si lega il dato. Il **puntatore contiene l'indirizzo della prima locazione libera**. Con questa struttura si ha una parte della memoria dove memorizzare informazioni senza definire a priori una dimensione. Stack -> array di locazioni, Stack Pointer -> registro di puntatore.

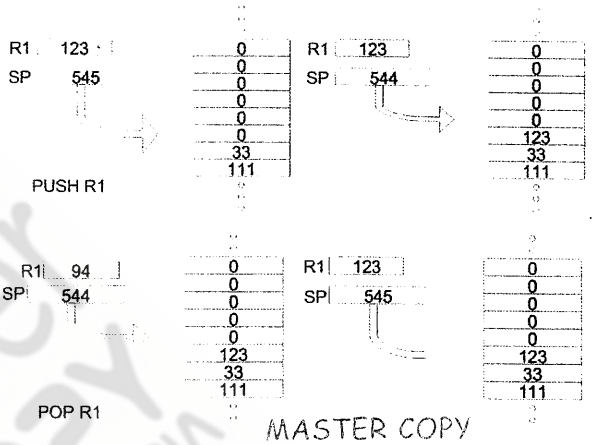


L'inserimento di un oggetto nella pila viene definito tramite un'operazione indicata con PUSH. R1 è un registro che contiene il valore 123, nello Steck Pointer è inserito un indirizzo di partenza della pila. La PUSH R1 prende 123 e lo scrive all'indirizzo 545, dopo di che automaticamente lo Steck Pointer viene decrementato ed è pronto per memorizzare all'indirizzo 544 un altro valore.

L'operazione inversa di estrazione dalla pila si indica con POP. Lo Steck Pointer da 544 viene incrementato e torna al valore 545, dopo di che viene eseguita una lettura della pila e scrittura nel registro R1.

Un'applicazione immediata di questa struttura è quella di servire come appoggio temporaneo dei dati. Si inserisce il contenuto di un registro nella pila, si utilizza il registro per altri scopi, dopo di che si recupera il vecchio valore prelevandolo dalla pila. Viene quindi spesso vista come una zona di memoria di appoggio

Dopo la POP il valore scritto nella pila non viene cancellato ma rimane scritto. L'operazione di cancellazione non è in realtà una vera e propria eliminazione del valore, ma semplicemente una perdita di riferimenti.



Rappresentazione

Per descrivere il funzionamento delle istruzioni e capire che tipo di informazioni abbiamo nelle diverse memorie, è necessario avere presenti le principali leggi di rappresentazione. I bit della rappresentazione sono numerati, il bit numero 7 è il più significativo (MSB), mentre il bit numero 0 è quello meno significativo (LSB). Questa nomenclatura prelude alla **notazione posizionale**.

$x = 128b_7 + 64b_6 + 32b_5 + 16b_4 + 8b_3 + 4b_2 + 2b_1 + b_0$

Qualunque numero naturale può essere sempre rappresentato in forma binaria in modo univoco avendo a disposizione un adeguato numero di bit. La maggior parte dei numeri seguono questa legge di rappresentazione che può essere usata ~~direttamente~~ DIRETTAMENTE per cambiare base da 2 a 10.

Per gli interi assoluti questo non è l'unico modo per attuare una codifica. Una codifica diversa è ad esempio la codifica detta BCD. Questa codifica mantiene l'ossatura del numero decimale e utilizza 4 bit per codificare individualmente ogni cifra. Quindi con 8 bit si rappresenta due cifre decimali da 0 a 99.

$x = 10(8b_3 + 4b_2 + 2b_1 + b_0) + 8b_3 + 4b_2 + 2b_1 + b_0$

Questo modo di rappresentare l'informazione è utile da usare quando può far comodo avere una forte analogia con un'interfaccia utente letta dall'uomo, è ad esempio molto usata nelle calcolatrici. Rispetto alla normale codifica binaria è marginale perché poco efficiente, poiché con un byte si rappresentano solo 100 cifre, che vuol dire perdere più della metà della capacità di rappresentazione del byte, 100 su 256.

Normalmente i numeri vengono rappresentati in modulo e segno, ma questo metodo non è ~~efficiente~~ EFFICIENTE da utilizzare all'interno della macchina. Nel caso dei numeri negativi questi vengono rappresentati con la codifica del complemento al due, che consiste nel considerare il bit più significativo di peso negativo.

$x = -128b_7 + 64b_6 + 32b_5 + 16b_4 + 8b_3 + 4b_2 + 2b_1 + b_0$

Siccome 128 è maggiore di qualsiasi numero rappresentabile con i restati bit, la presenza di un uno al posto sette ci fornisce un'informazione riguardante il segno del numero rappresentato.
Un primo vantaggio di queste rappresentazione è che è una rappresentazione senza sprechi, poiché oltre a poter rappresentare tutti i numeri compresi tra -128 e +127 con una combinazione univoca, esiste una rappresentazione unica anche per per lo zero, mentre nell'altro modo avevamo il più zero e il meno zero. **Il grossissimo vantaggio è che le operazioni di somma e differenza si eseguono allo stesso modo con lo stesso hardware**.

Normalmente il contenitore a 8 bit funziona bene per tutti gli alfabeti, con 256 codici si rappresentano facilmente tutti i numeri le lettere e i simboli. Per molti anni la rappresentazione più diffusa è stata la codifica ASCII. Oggi 8 bit non sono più sufficienti, per rappresentare tutti i simboli e le codifiche siamo passati a 16 bit. Maiuscole e minuscole differiscono di 32 e quindi è facile convertirle, basta semplicemente cambiare il bit di peso 32. I numeri sono tutti di seguito, vanno da 30 per lo zero a 39 per il nove, in esadecimale.

0x33 = 3 0x34 = 4 0x37 = 7

Per vedere la codifica ASCII dei numeri basta perdere la cifra meno significativa.
Per alcune operazione si lavora anche accoppiando due byte e considerando contenitori da 16 bit. I numeri senza segno vengono rappresentati allo stesso modo, partendo dal bit meno significativo in posizione zero, fino a quello più significativo in posizione quindici. **Il valore a 16 bit è quello tipico dell'indirizzo dei dati**. Se il numero è dotto di segno avrò sempre un complemento al due, il bit 15 avrà peso negativo.