

Istituto Professionale di Stato per l'Industria e l'Artigianato  
MORETTO

Via Luigi Apollonio, 21 BRESCIA

# CONTROLLO INDUSTRIALE A MICROPROCESSORE

Realizzato da:

FERRARI DAVIDE

FILIPPINI FABIO

MARCOLINI MAURIZIO

Classe 5AI TIEE Tecnici delle Industrie Elettriche Elettroniche

Anno scolastico 1997-98

INDICE :

<i>IL MICROPROCESSORE.....</i>	<i>II</i>
<i>STORIA DEL MICROPROCESSORE.....</i>	<i>II</i>
<i>LE ORIGINI DEL PC.....</i>	<i>VI</i>
<i>COS' E' LA PROGRAMMAZIONE.....</i>	<i>VII</i>
<i>DIAGRAMMA DI FLUSSO.....</i>	<i>VIII</i>
<i>STRUTTURA E FUNZIONAMENTO DI UN CALCOLATORE.....</i>	<i>VIII</i>
<i>LA MEMORIA.....</i>	<i>X</i>
<i>MAPPE DI MEMORIA.....</i>	<i>X</i>
<i>INPUT/OUTPUT.....</i>	<i>XI</i>
<i>ARCHITETTURA INTERNA DEL MICRO 6502.....</i>	<i>XIII</i>
<i>ARCHITETTURA DI UN SISTEMA A MICRO.....</i>	<i>XVI</i>
<i>LINGUAGGIO ASSEMBLY.....</i>	<i>XVII</i>
<i>STRUTTURA DI UNA ISTRUZIONE E</i>	
<i>MODI DI INDIRIZZAMENTO.....</i>	<i>XVII</i>
<i>RAPPRESENTAZIONE DI DATI NUMERICI.....</i>	<i>XX</i>
<i>LA SUBROUTINE.....</i>	<i>XXII</i>
<i>GLI INTERRUPT.....</i>	<i>XXII</i>
<i>IL SINGLE-CHIP R6501.....</i>	<i>XXIII</i>
<i>DESCRIZIONE DELLA PROVA D'ESAME.....</i>	<i>XXIV</i>
<i>DESCRIZIONE DEL PROGRAMMA PER CONTROLLO</i>	
<i>INDUSTRIALE A MICRO.....</i>	<i>X</i>

## IL MICROPROCESSORE

Il microprocessore è una unità centrale di calcolo fabbricata su un unico chip che, corredato da opportuni circuiti e programmato con istruzioni idonee, coordina un complesso di operazioni da svolgere in sequenza, qualunque esso sia e, allo stesso tempo, è il cuore ed il cervello del personal computer; esso determina la velocità e la potenza elaborativa dell'intero sistema. Per quanto riguarda le tecnologie costruttive, attualmente molti microprocessori sono realizzati in tecnologia CMOS, che consente consumi elettrici più ridotti. I circuiti interni del microprocessore sono progettati, al fine di consentire un'elaborazione rapida e di ridurre il riscaldamento, per supportare correnti poco elevate. Di conseguenza, ogni livello di segnale uscente deve passare attraverso un *buffer* di segnale, destinato ad elevare il valore della corrente. Deve la sua nascita alla felice intuizione del suo creatore Hoff (*Intel*) e dell'italiano Federico Faggin, che hanno portato all'implementazione di una intera CPU in un singolo circuito integrato (Intel 4004, nel 1971). In linea generale, un microprocessore contiene al suo interno una unità aritmetico-logica (ALU) e i blocchi di controllo del flusso di dati e di istruzioni di programma, sia interno sia verso il bus di collegamento con la memoria e i dispositivi di I/O. L'unità di I/O può essere piuttosto semplice e contenere solo pochi buffer oppure può coinvolgere funzioni complesse. Negli ultimi microprocessori Intel, l'unità contiene una memoria cache ed una logica a doppio clock per consentire, alle alte velocità operative del processore, di connettersi in modo appropriato con le memorie esterne. Esistono microprocessori che ospitano al loro interno unità di memoria e porte di ingresso/uscita, formando un blocco autonomo e funzionalmente completo; essi vengono detti *single-chip* e costituiscono soluzioni compatte e versatili di uso frequente nel campo della strumentazione di misura e di controllo.

## STORIA DEL MICROPROCESSORE

Il microprocessore è, allo stesso tempo, il cuore ed il cervello del personal computer; esso determina la velocità e la potenza elaborativa dell'intero sistema.

Ogni microprocessore agisce sulla base di un set di istruzioni. Nei laboratori dell'IBM di Yorktown, *John Cocke* scoprì, però, che molti dei lavori computerizzati si fondano su un numero ridotto di passi. Ad esempio, su un microprocessore dotato di un set di 200 istruzioni, i 2/3 dei lavori elaborativi sono basati su una decina circa delle istruzioni totali. Egli ritenne perciò possibile creare dei dispositivi gestiti da un set ridotto di istruzioni ed

introdusse un progetto basato su una architettura RISC (*Reduced Instruction Set Computer*), antitetica rispetto a quella basata su un vasto set di istruzioni, denominata CISC (*Complex Instruction Set Computer*).

Cocke ricevette, per i suoi lavori, il più alto riconoscimento della *Association for Computing Machines*, cioè il *Turing Award*, dal nome del pioniere della scienza dell'elaborazione, *Alan Turing*, il quale fornì un test, detto *test di Turing*, per definire l'intelligenza artificiale. Il risultato delle ricerche effettuato in questo campo fu sintetizzato nella regola dell'80/20, che stabilisce che circa il 20% delle istruzioni totali consente di svolgere l'80% dei lavori possibili.

L'architettura di tipo RISC ha pertanto incontrato un crescente favore nel progetto dei microprocessori ad alta velocità. Va, però, ricordato che non esistono linee di demarcazione così nette tra le complessità connesse alle due diverse tecnologie, RISC e CISC. Ad esempio, se si considerano i microprocessori M/2000 (prodotti dalla MIPS *Computer System*) e il classico 386 Intel, si può notare come, pur essendo il primo un RISC ed il secondo un CISC, non sussista una così grande diversità a livello del set di istruzioni (115 nel primo, 144 nel secondo). Nella filosofia RISC, infatti, la caratteristica fondamentale resta l'ottimizzazione del progetto relativo al compilatore, che si fonda sull'uso di tecniche di memorizzazione di tipo *cache* e *pipeline*. Inoltre, l'attuale tendenza è quella di rendere minimo il numero di cicli macchina necessari per eseguire una tipica istruzione. L'architettura RISC richiede mediamente meno di 1,5 cicli per istruzione, ma anche i CISC più recenti presentano un'efficienza di questo ordine, come il 486 Intel che richiede circa 1,2 cicli per istruzione.

Per quanto riguarda le tecnologie costruttive, attualmente molti microprocessori sono realizzati in tecnologia CMOS, che consente consumi elettrici più ridotti. I circuiti interni del microprocessore sono progettati, al fine di consentire un'elaborazione rapida e di ridurre il riscaldamento, per supportare correnti poco elevate. Di conseguenza, ogni livello di segnale uscente deve passare attraverso un *buffer* di segnale, destinato ad elevare il valore della corrente.

L'unità di I/O può essere piuttosto semplice e contenere solo pochi buffer oppure può coinvolgere funzioni complesse. Negli ultimi microprocessori Intel, l'unità contiene una memoria cache ed una logica a doppio clock per consentire, alle alte velocità operative del processore, di connettersi in modo appropriato con le memorie esterne.

Il primo vero microprocessore *general purpose* fu il 4004 Intel, seguito dall'8008 ad 8 bit. Nel '74, l'Intel realizzò una versione più innovativa, l'8080, progettato per dati delle

dimensioni di un byte e con un set di istruzioni più ricco. Nel '78 l'Intel diede un ulteriore impulso grazie al microprocessore 8086, in cui le dimensioni dei registri erano raddoppiate (16 bit) e che permetteva prestazioni dieci volte superiori. La memoria fu segmentata in 16 segmenti da 64 kbyte.

Un anno dopo, l'Intel presentò l'8088, identico all'8086, ma con un bus dati ridotto ad 8 bit per sfruttare facilmente l'hardware di supporto ad 8 bit normalmente disponibile. Questo microprocessore divenne la base di un'intera generazione di piccoli computer, in quanto l'idea di creare dei compatibili nacque proprio con gli 8088.

Il tipico microprocessore era realizzato secondo la tecnologia NMOS, ma, quanto ad assorbimento di energia, la tecnologia CMOS risulta migliore e pertanto i computer portatili erano spesso costruiti sull'equivalente CMOS dei microprocessori più diffusi. In commercio, si trovavano le versioni 80C88, 80C86. Col crescere dell'industria del piccolo computer, l'Intel non esitò a creare un chip più completo, un 8086 che contenesse la maggior parte dei suoi circuiti di supporto in un unico substrato. Immesso nel mercato nel 1982 con il nome di 80186, questo chip è servito da base a molti compatibili e ad almeno una scheda turbo.

Esistono due chip direttamente intercambiabili con l'8088 e l'8086 e sono rispettivamente il V20 e il V30 della NEC (*Nippon Electric Company*).

Il progetto di ciascun circuito integrato ha alle spalle anni di ricerca e sviluppo e, quindi, i produttori si difendono con tutti i sistemi legali possibili, quali brevetti, copyright e segretezza. L'Intel, comunque, autorizza ufficialmente la AMD (*Advanced Micro Devices*) e l'IBM a produrre molti dei suoi chip. La NEC, invece, non era autorizzata ad utilizzare i progetti Intel, che perciò intentò causa contro la NEC. L'introduzione del personal computer AT IBM nel 1984 focalizzò l'attenzione su un altro microprocessore Intel, l'80286, che utilizzava un bus dati a 16 bit reali con registri interni a 16 bit, velocità più elevate e 24 linee di indirizzo (16 Mbyte indirizzabili). L'80286 permetteva, inoltre, l'impiego della *memoria virtuale*, costituita da informazioni immagazzinate su una memoria di massa e trasferite nella memoria fisica quando necessario (1 Gbyte di memoria totale, di cui 16 Mbyte di memoria fisica e 1008 di memoria virtuale).

Per conservare la compatibilità con i precedenti chip, gli ingegneri dell'Intel dotarono l'80286 di due modi operativi: il *Real mode* ed il *Protected mode*. Quest'ultimo non si guadagnò molto facilmente la simpatia dei programmatori; trascorsero quasi tre anni tra l'introduzione dell'AT e quella di un sistema operativo IBM dotato di Protected mode, l'OS/2.

A differenza dell'80286, la generazione successiva di CPU Intel aprì le braccia al DOS ed alla libreria di software da 50000 miliardi di lire costruitagli attorno. L'80386, introdotto nel 1985, era più veloce, più potente e più versatile; gli furono raddoppiate le dimensioni dei registri e dei bus dati portandole a 32 bit reali. Il chip è in grado di gestire fino a 16 Tbyte di memoria virtuale, che può essere vista ed indirizzata come un'unica sezione contigua, ma che può anche essere segmentata, ed incorpora 16 byte di memoria cache a *pre-fetch*.

Diversamente dall'80286, il 386 può commutare tra le modalità Real mode e Protected mode senza dover essere resettato. Esiste, poi, un nuovo modo (*modo 8086 virtuale*) nel quale il chip simula contemporaneamente un numero pressoché illimitato di 8086, in una gestione di tipo *multitasking*.

Quando venne immesso sul mercato, però, il 386 costava oltre 800000 lire contro le 20000 lire dell'8086, ma, dal momento che l'Intel aveva deciso di non autorizzare altre società (tranne l'IBM) a produrlo, poté praticamente stabilirne il prezzo, imponendo qualunque cosa il mercato fosse in grado di sopportare. Per gli utenti interessati soprattutto al modo virtuale, cioè all'uso del software standard a 16 bit come il DOS e l'OS/2, l'Intel creò l'80386SX, un 80386DX ridotto che perdeva in potenza, ma non in caratteristiche. Invece di essere interfacciato con un bus di memoria a 32 bit, l'SX è progettato per un bus a 16 bit.

Della famiglia 386, ricordiamo, inoltre, il 386SL, per applicazioni di bassa potenza, e il 386SLC, che è un prodotto IBM.

Introdotta sul mercato nel 1989, il microprocessore 486 Intel rappresenta un miglioramento del 386; infatti, i programmi funzionano come se il 486 fosse un 386 più veloce.

Le differenze maggiori sono rappresentate da un progetto estremamente lineare e dalla presenza, nel chip stesso, del *coprocessore* e della cache interna (8k). La linearità del progetto hardware rende il 486 più veloce del 386 a parità di clock. Grazie al suo progetto interno, infatti, molte istruzioni del 486 possono avvenire in un solo ciclo di clock. I soli dispositivi in grado di competere con i 486 sono le stazioni di lavoro RISC, valide però per applicazioni particolari.

In base alle velocità operative, il 486 può dividersi in 486DX e in 486SX; quest'ultimo è privo del coprocessore numerico interno. Uno dei vantaggi offerti dalla famiglia 486 è la possibilità di variare la CPU 486 preesistente con qualunque versione più aggiornata facente parte della stessa linea, inserendo eventualmente un chip *overdrive*, cioè una nuova CPU ad alta velocità, senza asportare il processore esistente. L'*overdrive* velocizza

il processo di elaborazione, ma mantiene inalterata la velocità con cui la scheda madre comunica con l'esterno.

L'erede del 486 è rappresentato dal *Pentium* (il 586), un microprocessore a 64 bit dotato di una cache interna più ampia; infatti, la *cache on board* del Pentium è doppia rispetto a quella dei 486.

L'elemento più appariscente del Pentium è quello dell'architettura superscalare, basata su due pipeline per istruzioni, ognuna indipendente dall'altra. Queste pipeline permettono al Pentium di eseguire due istruzioni su interi in un unico ciclo di clock, raddoppiando teoricamente la potenza del chip, rispetto ad un 486, già a parità di frequenza di funzionamento. Come le pipeline del 486, anche quelle del Pentium sono in grado di eseguire istruzioni in cinque diversi stadi: *pre-fetch*, *decodifica dell'istruzione*, *generazione dell'indirizzo*, *esecuzione*, *write-back*. Ogni pipeline dispone di una sua ALU, una circuiteria per la generazione dell'indirizzo e un'interfaccia indipendente verso la cache per i dati. Aumentano anche di 3,5 volte le prestazioni dell'unità in *virgola mobile*, completamente riprogettata rispetto a quella del 486.

## LE ORIGINI DEL PC

Il PC IBM fu la prima macchina ad avvalersi del nome di *Personal Computer*. Esso (prodotto nell'81 e con una versione definitiva attorno al 1984) ha costituito il primo sistema standardizzato, ma era stato sviluppato per essere destinato ad un mercato limitato. La creazione di uno strumento di tipo *general purpose*, però, costituì la reale spinta propulsiva per la diffusione del PC. Nei primi anni '80, la maggior parte dei piccoli elaboratori disponibili era raggruppabile in tre grosse tipologie: *Apple*, *Tandy/Radio Shack*, *CP/M*.

L'Apple II era il predecessore ed anche il concorrente più importante del PC IBM. Era basato su un solo microprocessore (il 6502, ad 8 bit), aveva un bus di espansione dedicato e memorizzava i dati in audio cassette. In seguito, fu possibile la memorizzazione su disco gestita dal DOS Apple.

Il gruppo Radio Shack altro non era se non un grande magazzino dell'elettronica che vendeva di tutto, compresi piccoli elaboratori basati su progetti propri (come il TSR-80, computer da tavolo con microprocessore Z80 della *Zilog*).

Il CP/M (*Control Program for Microcomputer*) era un sistema operativo che collegava i microprocessori 8080 e Z80 con unità floppy e che consentiva di gestire numerose attività, dalla elaborazione dei testi alla contabilità.

La crescita del mercato dei piccoli computer incentivò l'IBM (*International Business Machines*) a creare il PC, basato su un microprocessore Intel 8088. Oltre al 6502 e allo Z80, infatti, in quel periodo era presente sul mercato l'Intel 8088, che aveva registri interni più ampi ed un maggior campo di indirizzamento della memoria. Per quanto riguarda la RAM, si decise di aggiungere il bit di parità e di fornire strutture per espansione fino a 512 k. Alcune locazioni furono utilizzate per la memoria video, altre per i programmi residenti nella ROM (il BIOS). Per la memoria di massa, furono sfruttati gli stessi supporti usati dagli altri costruttori. Furono, inoltre, previsti degli slot di espansione, in grado di accogliere accessori aggiuntivi. Si decise poi di adottare il BASIC come linguaggio di programmazione. La tastiera fu separata dal corpo del computer e collegata tramite un cavo.

Il Personal Computer ebbe un notevole successo sul mercato, tanto che la stessa IBM non fu in grado di costruirne altri abbastanza rapidamente. Decine di produttori cominciarono a creare le proprie versioni del PC, compatibili con l'originale IBM.

## *COS' E' LA PROGRAMMAZIONE*

Quando noi abbiamo un problema dobbiamo escogitare un modo per poterlo risolvere. Questa soluzione è espressa come una procedura di fasi successive chiamata *algoritmo*. Un algoritmo è quindi una specificazione fase per fase della soluzione da dare al problema. Una volta che la soluzione del problema è stata espressa sotto forma di un algoritmo occorre che questo venga eseguito dal calcolatore . il problema a questo punto sta nel fatto che il calcolatore non capisce il linguaggio comunemente impiegato, ma solo una parte del linguaggio ben definita è utilizzabile . questa parte è chiamata *linguaggio di programmazione*. La conversione di un algoritmo in una sequenza di istruzioni in un linguaggio di programmazione è detta *programmazione*, o per essere più specifici *codifica*. La programmazione effettiva comprende non solo la comprensione delle tecniche di realizzazione possibili per gli algoritmi convenzionali ma anche l'abilità di impiego di tutte le caratteristiche hardware del calcolatore come i registri interni, la memoria ed i dispositivi periferici, più un impiego costruttivo di opportune strutture di dati. La programmazione richiede anche disciplina di documentazione in modo da poter essere compresa anche da persone che non siano il programmatore. La documentazione deve essere interna ed esterna al programma. La documentazione esterna consiste nei documenti di progetto che sono separati dal programma: spiegazioni scritte, manuali e diagrammi di flusso. La

documentazione interna del programma fa riferimento ai commenti introdotti nel corpo del programma, allo scopo di spiegare il suo modo di operare.

## DIAGRAMMI DI FLUSSO

Esiste quasi sempre una fase intermedia fra l'algoritmo ed il programma che utilizza i *diagrammi di flusso*. Un diagramma di flusso è una rappresentazione simbolica dell'algoritmo, espressa come sequenza di blocchi contenenti le fasi dell'algoritmo. Questi blocchi sono dei rettangoli se utilizzati per comandi, oppure dei rombi se sono per un test

## STRUTTURA E FUNZIONAMENTO DI UN CALCOLATORE

Un calcolatore è una macchina per elaborare informazioni che opera obbedendo ad un programma. La maggior parte dei calcolatori individua le seguenti unità fondamentali:

- una unità centrale di elaborazione (CPU);
- una memoria;
- una unità di ingresso/uscita delle informazioni (I/O).

L'elaborazione delle informazioni viene svolta dalla CPU (*Central Processing Unit*) che è divisa in due parti: una unità aritmetico-logica (ALU) ed una unità di controllo (CU).

La memoria è indispensabile per immagazzinare programma e dati durante l'elaborazione. Le istruzioni necessarie al funzionamento base della macchina (*firmware*) sono immagazzinate in forma residente in dispositivi di memoria non volatili.

L'unità di I/O consente al calcolatore di comunicare con il mondo esterno.

L'elaborazione delle informazioni consiste nell'operare sui dati iniziali seguendo un processo strutturato a passi successivi. La successione di passi effettuata viene detta *algoritmo*. L'algoritmo viene tradotto in un programma operativo una volta che si ha a disposizione un certo particolare linguaggio. Le istruzioni devono essere tradotte poi in *linguaggio macchina assoluto*. Il linguaggio che esprime queste operazioni in chiaro e non in codice è detto linguaggio assembleatore (*assembly*). Ogni tipo di macchina avrà un proprio linguaggio assoluto ed un proprio linguaggio assembleatore; ad ogni istruzione in assembly corrisponde una istruzione in linguaggio assoluto. Il linguaggio assembleatore è detto anche a basso livello o orientato alla macchina.

La programmazione, però, può anche essere eseguita utilizzando un linguaggio ad alto livello o orientato ai problemi, la cui caratteristica è quella di essere completamente indipendente dalla CPU e di non richiedere una conoscenza specifica dell'architettura del sistema. Esso è legato al tipo di applicazione verso la quale è principalmente indirizzato e

contiene strutture tipiche e dedicate. Tra i linguaggi ad alto livello ricordiamo: il Fortran, il Cobol, il Pascal, il Basic, il C, ecc....

Il programma scritto in un determinato linguaggio deve essere tradotto in linguaggio macchina, utilizzando un apposito programma traduttore. Questo può essere un *interprete* o un *compilatore*.

Nel caso dell'interprete, il programma sorgente viene tradotto istruzione per istruzione e per ciascuna riga di programma viene eseguito il controllo della sintassi, la traduzione in linguaggio macchina e la sua esecuzione.

Nel caso del compilatore, invece, c'è una corrispondenza tra l'intero programma e il set di istruzioni. Il compilatore fa un'analisi sintattica di tutto il programma sorgente fornendo eventuali segnalazioni di errore; se il controllo sintattico dà esito positivo, viene effettuata la traduzione in linguaggio macchina. Il programma oggetto prodotto dal compilatore non è ancora eseguibile perché privo delle informazioni necessarie per eseguire operazioni matematiche più o meno complesse o operazioni di I/O. Si rende perciò necessaria un'ultima operazione, denominata di *Link* delle librerie. Il programma oggetto prodotto dal compilatore, "linkato" con le routine di libreria, è in grado di funzionare autonomamente.

\$FFFF
<b>ROM1</b>
\$E000
\$DFFF
<b>RAM3</b>
\$C000
\$BFFF
<b>ROM0</b>
\$A000
\$9FFF
<b>P.I.A.</b>
\$8000
\$7FFF
<b>EPROM</b>
\$6000
\$5FFF
<b>RAM2</b>
\$4000
\$3FFF
<b>RAM1</b>
\$2000
\$1FFF
<b>RAM0</b>
\$0000

## LA MEMORIA

Per memorizzare le informazioni, i computer utilizzano due tipi di supporti, la memoria centrale o primaria e la memoria di massa o secondaria. La memoria centrale (RAM) è quella immediatamente accessibile al microprocessore, mentre i dati contenuti nella memoria di massa, per essere utilizzati, devono essere prima trasferiti nella memoria centrale. Ricordiamo i vari tipi di memorie: RAM statiche e dinamiche, ROM mask, PROM, EPROM, EEPROM, RAM flash (particolare tipo di EEPROM nelle quali la cancellazione e la riprogrammazione vengono eseguite tramite le normali tensioni interne del PC).

Una caratteristica fondamentale delle memorie è rappresentata dalla velocità.

La memoria è solitamente connessa al bus locale del microprocessore, gira alla velocità della CPU e comunica con un bus delle stesse dimensioni del bus dati del microprocessore.

La gestione della memoria è determinata dal microprocessore. Per molti anni vi è stato un solo tipo di memoria utilizzata per immagazzinare i programmi, in quanto si era in presenza di un solo sistema operativo, il DOS.

### *MAPPE DI MEMORIA*

Il disegno mostra l'organizzazione dei dispositivi di memoria e di I/O in un generico sistema a microprocessore. In questo caso, i dispositivi presenti sono 4 blocchi: RAM(Ram0,1,2,3), una memoria EPROM, due blocchi ROM(Rom0,1) e un dispositivo di

P.I.A.. Quest'ultimo rappresenta un interfaccia standard di input e di output.

La P.I.A. (adattatore di interfaccia periferica) consente di gestire uscite ed ingressi tra l'esterno e il DATA-BUS rispondendo ad un indirizzo e comportandosi così come una memoria leggibile e scrivibile. Ogni dispositivo vede assegnata un area di memoria di 8K, e viene selezionato grazie all'indirizzamento che provvede ad abilitare il C/S(chip/select) di ognuno di essi. La tabella rappresenta la mappa di memoria del sistema ed è un modo standard per indicare i vari dispositivi e l'area di memoria che essi

occupano. In questo caso, tutti i dispositivi occupano lo stesso spazio di memoria e la mappa rappresenta, in modo completo, un sistema con ADDRESS-BUS a 16 bit da \$0000 a \$ffff pari a 65536 byte o locazione di memoria. Per esempio, la RAM0 risponde agli indirizzi compresi tra \$0000 e \$1ffff, la RAM1 risponde agli indirizzi compresi tra \$2000 e \$3fff. Una mappa di memoria non è necessariamente simmetrica vale a dire che quasi mai i dispositivi hanno la stessa occupazione di memoria, come indicato nell'esempio seguente:

\$ffff	
<b>LIBERO</b>	
\$8400	\$83ff
<b>RAM1k</b>	
\$7fff	\$8000
<b>RAM8k</b>	
\$6000	\$5fff
<b>EPROM16k</b>	
\$1fff	\$2000
<b>RAM8k</b>	
\$0000	

## INPUT/OUTPUT

Una delle funzioni di maggiore importanza per un sistema a microprocessore è INPUT/OUTPUT, ovvero la gestione da parte del microprocessore o della CPU di alcune linee di collegamento esterne. In generale il collegamento con un dispositivo di I/O richiede l'utilizzo di una serie di dispositivi intermedi, le interfaccia, che a loro volta, per funzionare hanno bisogno di speciali controlli, detti device controller. E' infine necessario, da parte dell'elaboratore, una strategia di controllo che va sotto il nome di scheduling. Una interfaccia è un dispositivo che può variare in complessità da pochi registri fino ad arrivare alle dimensioni di schede logiche di elevata complessità. I device controller sono necessari maggiormente nel caso in cui il dispositivo di I/O abbia al suo interno delle parti meccaniche. In generale il controller contiene al suo interno un'unità del tutto equivalente al microprocessore, in grado di ricevere istruzioni dal sistema principale e di eseguirle. Una volta che il sistema di I/O è collegato all'elaboratore deve essere definita una procedura di comunicazione, che va sotto il nome di scheduling, che viene gestita direttamente dal microprocessore. Tra queste si ricordano quella di : POLLING, INTERRUPT (analizzata successivamente) e quella di DMA

## FASI DI ESECUZIONE DI UNA ISTRUZIONE

Il calcolatore esegue il ciclo operativo secondo una sequenza caratteristica, svolgendo le seguenti operazioni:

- caricamento dell'istruzione da un registro di memoria (fase di *fetch*);
- decodifica dell'istruzione (fase di *decode*);
- esecuzione dell'istruzione (fase di *execute*).

Nella fase di fetch, l'istruzione viene prelevata e caricata in un opportuno registro dell'unità di controllo, denominato *Instruction Register*. Un'istruzione è composta di due parti: il *Codice Operativo* e l'*Indirizzo Operativo*. Il Codice Operativo indica quale operazione deve effettuare la CPU, mentre l'Indirizzo Operativo indica su quali dati questa operazione deve essere effettuata.

L'interpretazione dell'istruzione caricata nell'*Instruction Register* avviene nella fase di decode.

Nella fase di execute, infine, le operazioni prescritte dall'istruzione vengono portate a compimento attivando opportunamente i circuiti della ALU. Per poter eseguire queste fasi è fondamentale la presenza di un registro che prende il nome di *Program Counter*. Esso contiene sempre l'indirizzo dell'istruzione successiva a quella in fase di esecuzione e viene incrementato durante la fase di execute.

Questa procedura operativa è detta SISD (*Single Instruction Single Data*) e corrisponde all'architettura di von Neumann.

Per accelerare i calcoli sono state ideate architetture diverse da quella di von Neumann. Esiste un'architettura di tipo SIMD (*Single Instruction Multiple Data*), in cui una singola istruzione agisce su più dati, eseguendo l'operazione in un unico passo. Sono pertanto necessarie più unità aritmetico-logiche con registri in cui le operazioni vengono eseguite in parallelo.

Un altro tipo di architettura è quella di tipo MISD (*Multiple Instruction Single Data*), in cui si utilizzano istruzioni multiple su dato singolo per elaborarlo in modi diversi.

La macchina più complessa ha architettura MIMD (*Multiple Instruction Multiple Data*), nella quale in parallelo si fanno operazioni su dati di tipo diverso. In questo caso, è necessario avere compilatori e sistemi operativi adatti per queste macchine.

Uno dei parametri più importanti per valutare le prestazioni di un calcolatore è il numero di istruzioni eseguite nell'unità di tempo, cioè ad ogni ciclo macchina. Esso si misura in milioni di istruzioni al secondo (MIPS, *Mega Instruction Per Second*). In realtà, i MIPS sono una misura piuttosto labile della potenza elaborativa di un calcolatore. Infatti, ad essa contribuiscono anche il tempo di acquisizione dei dati, i tempi di trasferimento in memoria, i tempi di presentazione.

## **ARCHITETTURA INTERNA DEL MICROPROCESSORE 6502**

Il microprocessore 6502 appartiene alla serie 6500 della *Rockwell*. Le sue principali caratteristiche tecniche sono:



I registri *Indice X* e *Y* sono due registri ad 8 bit utilizzati per registrarvi dati su cui opererà il programma. Risultano utili nel modo di indirizzamento indicizzato e sono importanti per recuperare in modo efficiente i dati quando sono immagazzinati in tabelle.

Lo *Stack Pointer* (SP) è un registro puntatore alla sommità dell'area dello *Stack*, una zona di memoria organizzata secondo una struttura di tipo L.I.F.O. (*Last In First Out*). Si tratta cioè di una struttura cronologica: l'ultimo dato che entra è il primo ad uscire. Lo *Stack* è accessibile solo con le istruzioni di PUSH e PULL ed è disponibile per le Subroutine, gli Interrupt e l'immagazzinamento temporaneo dei dati. Nel 6502, lo *Stack Pointer* è ristretto a 8 bit, ma comprende un nono bit posto sempre a 1. L'area riservata allo *Stack* va, quindi, dall'indirizzo 256 all'indirizzo 511 (da \$100 a \$1FF). Lo *Stack Pointer* viene inizializzato a \$FF e funziona decrementandosi in fase di immagazzinamento dei dati nello *Stack*.

Il *registro di Stato* o *registro dei Flag* (P) è un registro a 8 bit, dei quali però solo 7 sono utilizzati, che indica in ogni momento lo stato del microprocessore, cioè l'esito dell'ultima operazione effettuata. Il registro P è rappresentato in figura 2 .



Figura 2

Ciascun bit è detto *flag* ed evidenzia una particolare condizione. Il bit 0 è il flag di *Carry* (riporto): quando la somma fra due dati ad 8 bit genera un riporto, il flag di *Carry* si setta. Ciò si verifica anche quando si ha un prestito nell'operazione di sottrazione. Il flag di *Carry* è usato in modo estensivo durante le operazioni aritmetiche.

Il bit 1 è il flag di *Zero*: quando il risultato dell'ultima operazione eseguita è nullo, questo flag si setta. Il flag di *Zero* viene influenzato anche dalle operazioni di confronto.

Il bit 2 è il flag di *Interrupt Mask* (interruzione mascherabile): intervenendo su questo flag, il programmatore istruisce la CPU al fine di prendere in considerazione o meno le eventuali richieste di interrupt inviate dall'unità di I/O attraverso la IRQ. Se questo flag è a livello alto, l'interruzione viene mascherata, ossia la richiesta viene ignorata e la CPU continua a svolgere il suo lavoro; se,

invece, il bit è a livello basso, l'interruzione non viene mascherata e, in caso di interrupt, la CPU sospende le operazioni in corso e svolge l'elaborazione richiesta. Il bit 3 è il flag *Decimal*: quando è posto a livello alto, il processore opera in modo BCD, altrimenti opera in modo binario.

Il bit 4 è il flag di *Break*: esso è posto automaticamente a livello alto dal microprocessore se un interrupt è causato da un comando BRK e permette, quindi, di distinguere un break programmato da un interrupt hardware.

Il bit 5 non è utilizzato.

Il bit 6 è il flag di *Overflow* (traboccamento): si porta a livello alto quando si ottiene un risultato che è al di fuori del campo dell'operazione eseguita. Ad esempio, l'Overflow interviene quando una somma fra due numeri positivi dà come risultato un numero negativo.

Il bit 7 è il flag *Negative*: si porta a livello alto quando l'ultimo risultato elaborato è negativo, ossia il bit più significativo del dato ottenuto è a livello alto.

L'*Address Buffer* e il *Data Buffer* sono i buffer che consentono l'interfacciamento del microprocessore con i bus degli indirizzi e dei dati.

L'*Instruction Register* (Registro Istruzioni) consente la memorizzazione delle istruzioni in fase di caricamento, istruzioni successivamente decodificate grazie all'*Instruction Decoder*.

Il clock di sistema è un circuito temporizzatore che regola tutte le sequenze operative. Nel 6502 vengono definite due fasi che regolano rispettivamente le operazioni sull'Address-bus e sul Data-bus; la frequenza di lavoro utilizzata è di 1MHz e, generalmente, l'oscillatore utilizza un quarzo.

## ARCHITETTURA DI UN SISTEMA A MICROPROCESSORI

Come già detto, l'architettura più semplice di un sistema a microprocessore è quella di von Neumann., rappresentata nello schema a blocchi di figura 2.

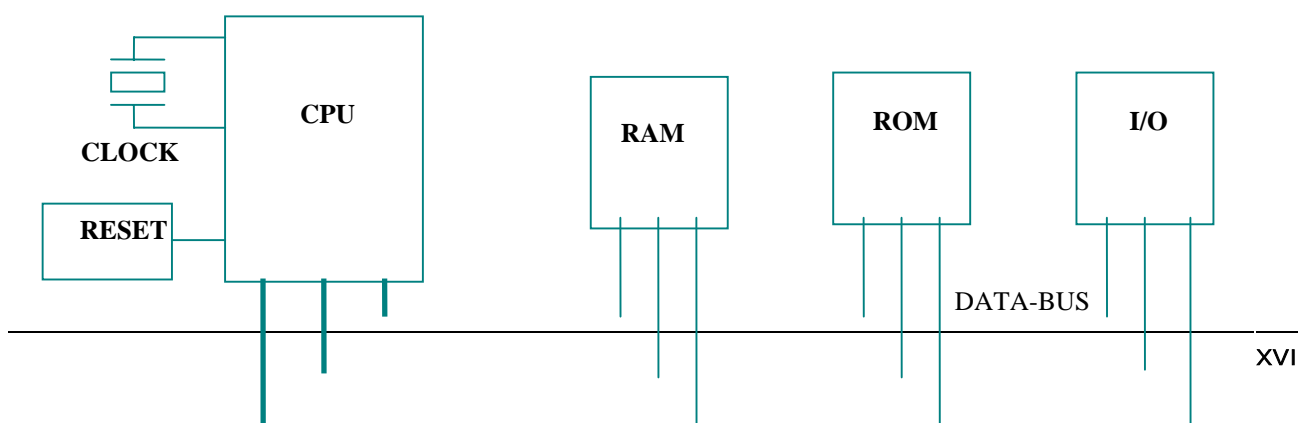




Figura 3

In tale struttura, la CPU è connessa alla memoria e alle unità di I/O tramite tre bus: il DATA-BUS (bus dei dati), l'ADDRESS-BUS (bus degli indirizzi), il CONTROL-BUS (bus di controllo).

Il *Data-bus* permette lo scambio di dati fra CPU e memoria ROM, CPU e memoria RAM, CPU e unità di I/O, ed è un bus bidirezionale. La direzione di movimento dei dati, infatti, dipende dal tipo di operazione: se si tratta di una operazione di lettura (Read), il dato entra nella CPU, mentre se si tratta di un'operazione di scrittura (Write), il dato esce dalla CPU.

L'*Address-bus* è il bus degli indirizzi ed è unidirezionale. L'indirizzo generato dalla CPU specifica la locazione di memoria selezionata o l'unità di I/O interessata allo scambio dei dati. La capacità di indirizzamento dipende dal numero di linee del bus: ad esempio, una CPU con 16 bit di indirizzamento ha una capacità di indirizzamento di 64k; una CPU con 20 bit può indirizzare 1M; una CPU con 32 bit può indirizzare 4096M.

Il *Control-bus* raccoglie i vari segnali necessari alla gestione del colloquio tra i vari blocchi. La complessità di tale bus dipende dall'architettura del sistema e dal microprocessore. Segnali del bus di controllo sono, ad esempio, quello di Read/Write e quello di richiesta di interruzione IRQ (*Interrupt ReQuest*).

Nello schema di figura sono rappresentati anche i circuiti di clock e di reset. Il generatore di clock è un oscillatore al quarzo che genera i segnali per la temporizzazione del sistema. La circuiteria di reset o restart è costituita da un circuito monostabile che fornisce il segnale di avviamento dell'intero sistema.

## IL LINGUAGGIO ASSEMBLY

Il nome di questo linguaggio di programmazione deriva dal fatto che le istruzioni macchina sono direttamente assemblate a partire da istruzioni assembly per mezzo di un particolare programma traduttore, l'assembler, o assemblatore. Il miglioramento sostanziale portato dall'assembly è dato da uso di nomi convenzionali per ogni codice di istruzione. Il codice dell'istruzione viene denominato codice mnemonico perché è fatto in modo da descrivere il compito svolto dall'istruzione. Tutti i microprocessori possiedono un loro set di istruzioni in codice mnemonico che può essere utilizzato direttamente. Oltre a definire codici mnemonici per le istruzioni, il costruttore assegna anche dei nomi convenzionali ai registri,

cioè a quelle locazioni di memoria particolari dedicate a scopi precisi, si tratta ad esempio dell'Accumulatore ,denominato solitamente A, e dei registri di servizio. il linguaggio Assembly ha dei piccoli problemi: se il programma è lungo c'è un notevole dispendio di tempo e il pericolo di commettere errori nella battitura delle istruzioni; la necessità di avere un programma assemblatore che traduca quello scritto in codice mnemonico in linguaggio macchina(cioè quello che il microprocessore capisce); inoltre il fatto che ogni microprocessore ha un proprio set di istruzioni per cui i programmi non possono essere utilizzati su macchine diverse.

## STRUTTURA DI UNA ISTRUZIONE E MODI DI

### INDIRIZZAMENTO

Le istruzioni in assembly possono essere distinte in cinque categorie:

- istruzioni di trasferimento dati;
- istruzioni di elaborazione dati;
- istruzioni di test e diramazione;
- istruzioni di ingresso/uscita;
- istruzioni di controllo.

Le istruzioni di trasferimento dati sono quelle che interessano l'accumulatore e i registri indice (LDA, STA, TAX, TAY, TXA, TYA, LDX, LDY, STX, STY) e quelle che salvano i dati nello Stack (PHA, PHP, PLA, PLP).

Le operazioni aritmetiche sono quelle di ADC e SBC. Le operazioni di incremento e decremento sono disponibili sulla memoria e sui registri indice, ma non sull'accumulatore, e sono: INC, DEC, INX, INY, DEX, DEY.

Le operazioni logiche sono quelle di AND, ORA, EOR. Le istruzioni di shift e di rotazione sono ASL, LSR, ROL, ROR. Le istruzioni di confronto sono CMP, CPX, CPY.

Per test e diramazione sono disponibili le istruzioni di salto condizionato (branch) che testano i bit del registro di stato. Esse sono: BCC, BCS, BEQ, BNE, BMI, BPL, BVC, BVS. I salti incondizionati sono effettuati con le istruzioni JMP, JSR, RTS.

Le istruzioni che operano sul registro di stato sono CLC, SEC, CLD, SED, CLI, SEI, CLV.

Ulteriori istruzioni sono BRK, RTI, BIT, NOP.

In linguaggio macchina, invece, un'istruzione è formata da 1, 2 o 3 byte. Il primo byte rappresenta il Codice Operativo ed indica l'operazione che deve essere eseguita. Esso può essere seguito da uno o due byte, ma questo dipende dal modo di indirizzamento, che

fa riferimento alle specifiche utilizzate per stabilire l'operando su cui interviene l'istruzione stessa.

1) Modo di Indirizzamento Implicito.

In questo caso, l'istruzione è costituita dal solo codice operativo e non è necessario alcun dato. In linguaggio assembly, è rappresentata dal solo simbolo mnemonico (ad esempio, CLC, TAX, INY, ecc.), mentre in linguaggio macchina viene tradotta con un solo byte (ad esempio, CLC è tradotta in \$18).

2) Modo di Indirizzamento di Accumulatore.

Questo modo di indirizzamento coinvolge implicitamente l'Accumulatore e l'istruzione viene tradotta in linguaggio macchina con un solo byte. Ad esempio, ROR A viene tradotta con \$6A.

3) Modo di Indirizzamento Immediato.

Con questo modo di indirizzamento, l'istruzione è seguita direttamente dal dato e, quindi, in linguaggio macchina è rappresentata da due byte. Ad esempio, LDA #15 viene tradotta con \$A9 \$0F. Questo modo di indirizzamento è facilmente riconoscibile per la presenza del simbolo #.

4) Modo di Indirizzamento Assoluto.

Questo modo di indirizzamento prevede che, dopo il codice operativo dell'istruzione, siano presenti due byte indicanti l'indirizzo della locazione di memoria in cui è contenuto il dato da elaborare. In linguaggio macchina viene indicato prima il byte meno significativo e poi quello più significativo. Ad esempio, l'istruzione LDA \$300 viene tradotta con \$AD \$00 \$03.

5) Modo di Indirizzamento in Pagina Zero.

Viene utilizzato quando la locazione di memoria si trova in pagina zero, come avviene per le istruzioni eseguite sui PORT di ingresso/uscita. Ad esempio, STA PORTD viene tradotta con \$85 \$03, in quanto l'indirizzo del PORTD è \$03. In linguaggio macchina, quindi, l'istruzione viene tradotta con due soli byte.

6) Modo di Indirizzamento Indicizzato.

L'indirizzamento indicizzato può essere assoluto o in pagina zero e può essere realizzato con il registro X o con il registro Y. In assembly, l'indicizzazione è espressa, ad esempio, nel modo seguente:

STA \$1000,Y  
LDA \$10,X

In linguaggio macchina, si ha:

\$99 \$00 \$10

## \$B5 \$10

Come si vede, il codice operativo è seguito dall'indirizzo base a cui deve essere aggiunto il contenuto del registro indice utilizzato per ottenere l'esatto indirizzo della locazione di memoria in cui si trova l'operando. Nell'esempio precedente, l'istruzione STA \$1000,Y indica che il contenuto dell'Accumulatore verrà conservato nella locazione di indirizzo \$1000+Y.

In totale abbiamo 4 indirizzamenti indicizzati, ma quello in pagina zero con Y è valido solo per le istruzioni LDX e STX.

### 7) Modo di Indirizzamento Relativo.

Viene usato nelle istruzioni di salto condizionato (branch). In assembly, al simbolo mnemonico dell'istruzione viene fatta seguire l'etichetta dell'istruzione alla quale saltare (ad esempio, BEQ LOOP). In linguaggio macchina, il codice operativo dell'istruzione è seguito da un byte di offset che indica il numero di byte da saltare in avanti o indietro. In quest'ultimo caso, tale numero è rappresentato in complemento a due. Ad esempio, un salto in avanti di 5 byte sarà rappresentato da \$05, mentre un salto indietro di 5 byte sarà rappresentato da \$FB.

### 8) Modo di Indirizzamento Indiretto Indicizzato con X (Preindicizzazione).

Al codice operativo segue un byte che deve essere sommato al contenuto del registro X per ottenere un indirizzo in pagina zero. La locazione di memoria ottenuta e quella immediatamente successiva conterranno rispettivamente la parte bassa e la parte alta dell'indirizzo di memoria effettivo dell'operando. Ad esempio:

```
LDX #2  
LDA ($75,X)
```

Queste due istruzioni consentono di ottenere le locazioni \$77 e \$78. Supponendo che queste locazioni contengano \$00 e \$C0, l'indirizzo completo è \$C000. Quindi, al termine di queste due istruzioni l'accumulatore conterrà il dato memorizzato nell'indirizzo \$C000.

### 9) Modo di Indirizzamento Indiretto Indicizzato con Y (Postindicizzazione).

In questo tipo di indirizzamento, invece, viene prima caricato l'indirizzo indiretto e, successivamente, a questo viene aggiunto il contenuto di Y. Ad esempio:

```
LDY #3  
LDA ($75),Y
```

Supponendo che le locazioni \$75 e \$76 contengano rispettivamente \$00 e \$C0, si ottiene l'indirizzo \$C000 a cui va sommato Y. Pertanto, l'indirizzo effettivo sarà \$C003.

### 10) Modo di Indirizzamento Indiretto Assoluto.

Viene usato solo per l'istruzione di salto incondizionato JMP. Al codice operativo seguono due byte che identificano l'indirizzo di una locazione di memoria che, con quella immediatamente successiva, ci darà l'indirizzo effettivo del salto.

### RAPPRESENTAZIONE DI DATI NUMERICI

La rappresentazione dei numeri deve essere distinta poiché è necessario rappresentare i numeri interi, quelli con segno e quelli decimali.

La rappresentazione di numeri interi viene eseguita impiegando una rappresentazione binaria diretta.

In una rappresentazione binaria con segno, invece, il bit più significativo è impiegato per indicare il segno del numero. Tradizionalmente, 0 è impiegato per denotare un numero positivo, mentre 1 è impiegato per denotare un numero negativo. In questo caso, è possibile rappresentare numeri da -128 a +127, utilizzando la rappresentazione in complemento a due.

Le operazioni aritmetiche richiedono spesso la manipolazione del bit di carry e del bit di overflow. In particolare, l'overflow si verificherà nelle situazioni:

- somma di numeri positivi troppo grandi;
- somma di numeri negativi troppo grandi in valore assoluto;
- sottrazione di un numero positivo molto grande da un numero negativo molto grande in valore assoluto;
- sottrazione di un numero negativo molto grande in valore assoluto da un numero positivo molto grande.

Vediamo alcuni esempi:

00000110 +       (+6)

00001000       (+8)

---

00001110       (+14)

Il risultato è corretto (V=0, C=0).

01111111 +       (+127)

00000001       (+1)

---

10000000       (-128)

---

Si verifica un overflow (V=1, C=0).

00000100 + (+4)

11111110 (-2)

---

00000010 (+2)

Il risultato è corretto (V=0, C=1 ignorato).

00000010 + (+2)

11111100 (-4)

---

11111110 (-2)

Il risultato è corretto (V=0, C=0).

11111110 + (-2)

11111100 (-4)

---

11111010 (-6)

Il risultato è corretto (V=0, C=1 ignorato).

10000001 + (-127)

11000010 (-62)

---

01000011 (+67)

Si verifica un underflow (V=1, C=1).

## LE SUBROUTINE

Quando in un programma si ha la necessità di ripetere più di una volta uno stesso gruppo di istruzioni, è utile creare delle subroutine. Una subroutine è un sottoprogramma che può

essere richiamato da un qualsiasi numero di punti del programma principale con una istruzione di JSR (*Jump to SubRoutine*).

Quando il microprocessore incontra tale istruzione, effettua un salto all'indirizzo specificato, ma salva nello stack l'indirizzo contenuto nel Program Counter, cioè l'indirizzo di ritorno. Infatti, al termine del sottoprogramma, il microprocessore incontra l'istruzione RTS (*ReTurn from Subroutine*) e, quindi, recupera dallo stack l'indirizzo di ritorno, cioè quello dell'istruzione successiva a quella che ha determinato il salto.

### GLI INTERRUPT

L'interrupt è una tecnica di I/O molto usata. Possono essere richiamati durante l'esecuzione di un programma tramite un segnale hardware applicato agli ingressi IRQ e NMI del microprocessore. L'esecuzione di un interrupt comporta il salvataggio nello stack dell'indirizzo contenuto nel Program Counter e del registro di stato.

L'interrupt può essere *mascherabile* o *non mascherabile*.

Nel primo caso, il microprocessore può o meno rispondere in base allo stato del flag I del registro di stato. Se tale bit è settato, il microprocessore non risponde ad eventuali richieste di IRQ (*Interrupt ReQuest*) ma termina l'operazione che stava svolgendo salvando la richiesta in attesa di poterla eseguire. Il flag I viene automaticamente posto a 1 dalle chiamate accettate; esso, inoltre, può essere settato o resettato dalle istruzioni SEI e CLI.

Invece, in caso di interruzione non mascherabile (NMI) il microprocessore deve sempre rispondere.

In risposta ad una chiamata di interrupt, il microprocessore legge le locazioni del vettore di interrupt che contiene l'indirizzo dei programmi di interrupt. Al termine, l'istruzione RTI (*ReTurn from Interrupt*) comporta l'automatico recupero del registro di stato dallo stack. L'esecuzione di un interrupt non altera il programma principale che prosegue poi normalmente.

Una interruzione IRQ può essere richiesta anche via software con l'istruzione BRK. In questo caso, il flag di Break (B) del registro di stato riflette l'avvenuta interruzione del programma.

### IL SINGLE-CHIP R6501

Il *single-chip* 6501 della Rockwell è una CPU 6502 avanzata, compatibile con tutti i membri della famiglia R6502. Tra le sue caratteristiche, ricordiamo quattro nuove istruzioni per la manipolazione del singolo bit, un oscillatore di clock interno, 192 byte di RAM statica ed una circuiteria di interfaccia versatile. Quest'ultima include 4 Port di I/O (32 linee

bidirezionali), due timer/counter programmabili a 16 bit, un canale seriale *full-duplex*, 10 interrupt e un bus espandibile a 64 kbyte di memoria esterna. La versione R6501Q ha un cristallo a 4 MHz con operazioni a 1 MHz, mentre la versione R6501AQ ha un cristallo a 4 MHz con operazioni a 2 MHz. Il chip ha 64 pin ed è costruito in tecnologia NMOS-3 silicon gate.

Funzionalmente, R6501Q consiste di una CPU, una RAM, 4 porte parallele di I/O a 8 bit, una porta seriale, due circuiti counter/latch, un registro di controllo di modo, un registro di interrupt flag ed un registro di interrupt enable.

Lo *Stack Pointer* è posizionato in pagina zero alle locazioni di memoria da \$FF a \$40.

Le quattro istruzioni aggiunte al set riguardano la manipolazione di un singolo bit di un indirizzo in pagina zero (memoria o Port di I/O). Esse sono: SMB (*Set Memory Bit*), RMB (*Reset Memory Bit*), BBS (*Branch on Bit Set*), BBR (*Branch on Bit Reset*).

La RAM consiste di 192 byte di memoria con indirizzi in pagina zero assegnati da \$40 a \$FF.

Il registro MCR (*Mode Control Register*) contiene i bit di controllo per i Port di I/O e i bit di selezione dei due *Counter A* e *B*. La sua configurazione, insieme a quella del registro SCCR (*Serial Communications Control Register*), determina la configurazione base del R6501Q in ogni applicazione. L'inizializzazione di questo registro, di indirizzo \$14, è una delle prime azioni di ogni programma.

Una richiesta di interrupt IRQ può essere abilitata o disabilitata mediante il registro IER (*Interrupt Enable Register*). Il registro IFR (*Interrupt Flag Register*) contiene invece le informazioni che indicano quale I/O o Counter necessita di attenzione. Il contenuto dell'IFR può essere esaminato leggendo all'indirizzo \$11 e può essere azzerato eseguendo istruzioni RMB all'indirizzo \$10. Ogni bit dell'IFR ha un corrispondente bit nell'IER all'indirizzo \$12.

Le 32 linee di I/O sono raggruppate in quattro Port di input/output denominati PORTA (\$0), PORTB (\$1), PORTC (\$2) e PORTD (\$3). Possono essere programmati per realizzare più funzioni, mediante i registri MCR e SCCR.

## **DESCRIZIONE DELLA PROVA D'ESAME**

Il sistema di automazione industriale costruito consiste nella selezione di pezzi uscenti da una macchina industriale. Il pezzo uscente scorre sopra un nastro trasportatore, attraverso una fotocellula che ne rileva la lunghezza e il numero totale. A questo punto a seconda delle lunghezze il pezzo viene in uno dei tre contenitori predisposti per il raccoglimento e

che si trovano antistante il nastro trasportatore. I selettori di pezzi vengono pilotati tramite il programma contenuto nel 6501 stilato in "assembly" Si ha il primo contenitore con pezzi di una lunghezza compresa tra un minimo "A" ed un massimo "A"; il secondo , analogamente al primo, che contiene pezzi compresi tra un minimo "B", leggermente superiore al massimo "A", e un massimo "B"; il terzo cassone contiene i pezzi di scarto, ovvero i pezzi più corti del minimo di "A" e più lunghi del massimo "B". i valori di massimo e di minimo dei vari cassoni è fornito dal programma stilato in "pascal". Quando uno dei tre cassoni è pieno, il cui valore è sempre fornito dal programma precedente, viene mandato un segnale, visualizzato tramite un led per ogni contenitore, che fornisce il comando di cambio cassone, il quale viene fatto automaticamente. Ci sono inoltre tre pulsanti :

- start
- stop
- azzeramento totale

Il compito del programmatore è quello di programmare la scheda 6502 in modo che abbia i seguenti ingressi e le seguenti uscite e che quindi faccia funzionare tutto il sistema senza bisogno di operazioni manuali.

#### Ingressi

- Fotocellula
- Pulsante di start
- Pulsante di stop
- Pulsante di azzeramento del conteggio
- Ingresso dei parametri di lavoro provenienti dal programma scritto in "pascal"

#### Uscite

- Comando di scarica pezzi nella cassa "A"
- Comando di scarica pezzi nella cassa "B"
- Comando di scarica pezzi nella cassa "SCARTI"
- Cambio scatola "A"
- Cambio scatola "B"
- Cambio scatola "SCARTI"

Mentre il programma in "Pascal" ci fornisce i parametri fondamentali di lavoro, che entrano in ingresso al 6502, che sono:

- Lunghezza minima "A"
- Lunghezza massima "A"

- Lunghezza minima “B”
- Lunghezza massima “B”
- Totale massimo di pezzi contenuto nei cassoni “A”, “B”, “SCARTO”.

## DESCRIZIONE DEL PROGRAMMA PER CONTROLLO

### INDUSTRIALE A MICROPROCESSORE

Il programma prevede inizialmente la dichiarazione dei collegamenti hardware. Questa operazione serve per indicare come sono utilizzati i PORT, vale a dire come ingresso o come uscita. Nel nostro caso il PORT\_B è utilizzato come ingresso, il PORT\_D come uscita e il PORT\_C sempre come uscita ma connesso alla circuito stampato di visualizzazione.

In seguito sono elencate le variabili di programma che impostano la locazione di memoria dove sono memorizzati i dati utilizzati.

Dopo l'inizializzazione della CPU, immettiamo le lunghezze massime e minime, dei pezzi che transitano sul nastro trasportatore, che hanno il compito di indicare al sistema la posizione dei pezzi nelle scatole. Questa operazione è eseguita attraverso due istruzioni che sono :

- LDA #(lunghezza) : carica in accumulatore la lunghezza
- STA (locazione) : scarica il contenuto dell'accumulatore nella locazione di memoria

Queste istruzioni sono ripetute per tutte le impostazioni dei parametri.

Il programma è articolato da un corpo centrale e da quattro sub-routine.

Il corpo centrale inizia con l'interrogazione del pulsante di start: se è resettato continua a leggerlo fino a quando non è azionato in altre parole passa a livello alto. In seguito è controllato il pulsante di stop attraverso una sub-routine , perché questo deve poter bloccare il tutto in qualsiasi istante. Per questa sub-routine è richiamata più volte nello svolgimento. Eseguiti questi controlli principali è guardata la fotocellula che se è settata è ricontrollata fino a quando non è resettata. A questo punto è eseguito un ritardo contato fino a quando la fotocellula non ritorna a livello alto. Finito il conteggio il programma confronta i risultati del contatore con le lunghezze inserite nella fase iniziale. A seconda dei vari risultati il pezzo deve andare in una determinata scatola. Questo passaggio fa scattare la fotocellula messa prima del cassone per poter contare il numero di pezzi contenuti. Questo conteggio è confrontato con il numero totale dei pezzi inserito inizialmente, che ogni scatola deve contenere. Se i due parametri sono uguali attraverso un segnale luminoso è segnalato il cambio scatola effettuato automaticamente, che

determina un tempo di blocco del sistema , questo si resetta automaticamente finito l'intervallo.

La sub-routine del pulsante di stop è utilizzata anche per la visualizzazione dei dati poiché viene richiamata spesso evitando così un comando aggiuntivo. Controllando l'interruttore 1 del PORT\_B, con l'istruzione BBR, si controlla se è resettato o è settato e in funzione di questa il programma si blocca oppure continua indisturbato.

La sub-routine che gestisce il ritardo utilizza due registri indice X,Y. Il registro X viene azzerato, mentre il registro Y viene caricato con un valore che influirà sul tempo totale di ritardo. Per poter aumentare il tempo di ritardo vengono inserite delle istruzioni ( NOP ) che occupano due cicli macchina. Decrementando il registro X otteniamo un ciclo di 256 volte, che viene ripetuto per il valore del registro Y. Confrontando il valore di Y, ad ogni decremento, con 0 si può sapere quando sono uguali per cui si può ritornare alla parte centrale.

La sub-routine che gestisce il ritardo del cambio scatola utilizza il sotto programma del ritardo ripetendolo per un numero di volte prefissato nelle sub-routine. il risultato viene visualizzato sul display, però questa operazione altera l'Accumulatore dove all'inizio del sotto programma è stato fissato il parametro di ripetizione. Per ovviare a questo problema utilizziamo le istruzioni PHA (salva contenuto Accumulatore nello Stack) e PLA (carica il contenuto dello Stack in Accumulatore).

La sub-routine che gestisce la visualizzazione su display utilizza i selettori connessi ai bit 6 e 7 del PORT\_B. Nel programma è spiegato come stabilità la selezione dei pulsanti. Inizialmente viene controllata la combinazione dei selettori 6 e 7 e viene quindi scelto quale tipo di visualizzazione si vuole. Ad esempio nella combinazione 0 0 viene caricato in Accumulatore il numero dei pezzi A che poi viene convertito attraverso la routine successiva, e viene invertito l'ordine dei bit come da esempio :

```
0111 0000   diventa
0000 0111
```

Questa operazione viene eseguita per adattare la visualizzazione al circuito stampato inerente.

La sub-routine di cui si è accennato in precedenza contiene la tabella che permette di convertire la cifra binaria in BCD per permetterne la visualizzazione, contiene le seguenti istruzioni :

```
TAX   (trasferisce il valore dell'accumulatore nel registro x)
LDA BCD_TAB,X (carica il valore della tabella alla posizione x)
```

## RTS (esce dalla sub-routine)

```
; *****
; *
; * ----- *
; * CONTROLLO INDUSTRIALE A MICROPROCESSORE *
; * ----- *
; *
; * Ipsia "Moretto" - Brescia *
; *
; * Maturita' TIEE A. S. 1997/8 *
; *
; * Gruppo Lavoro: Ferrari *
; *                 Filippini *
; *                 Marcolini *
; *
; *****

; Collegamenti hardware:

; PORTB_0 = (Input) Pulsante START.
; PORTB_1 = (Input) Pulsante STOP.
; PORTB_2 = (Input) Fotocellula.
; PORTB_3 = (Input) Fotocellula scatola A
; PORTB_4 = (Input) Fotocellula scatola B
; PORTB_5 = (Input) Fotocellula scatola scarti
; PORTB_6 = (Input) Selettore display dati bit 0
; PORTB_7 = (Input) Selettore display dati bit 1

; PORTC_0 = (Output) Display data
; PORTC_1 = (Output) Display clock
; PORTC_2 = (Output) Display blinking command
; PORTC_3 = (Output) Display latch enable

; PORTD_0 = (Output) Scarico scatola A.
; PORTD_1 = (Output) Scarico scatola B.
; PORTD_2 = (Output) Scarico scatola scarti.
; PORTD_3 = (Output) Richiesta cambio contenitore A.
; PORTD_4 = (Output) Richiesta cambio contenitore B.
; PORTD_5 = (Output) Richiesta cambio contenitore scarti.

; Dichiarazioni iniziali di programma.

PROG          .EQU $0200      ; Start programma.

              ; Variabili "globali" di programma.

KEYB_TASTO    .EQU $0050      ; Tasto premuto (ASCII) su tastiera.

SOGLIA1       .EQU $0051      ; Soglia conteggio uno.
SOGLIA2       .EQU $0052      ; Soglia conteggio due.
```

; ----- VARIABILI DI PROGRAMMA -----

LUN\_MIN\_A .EQU \$0060 ; Lunghezza minima pezzi A.  
LUN\_MAX\_A .EQU \$0061 ; Lunghezza massima pezzi A.  
LUN\_MIN\_B .EQU \$0062 ; Lunghezza minima pezzi B.  
LUN\_MAX\_B .EQU \$0063 ; Lunghezza massima pezzi B.  
  
NR\_TOT\_A .EQU \$0064 ; Numero totale pezzi A.  
NR\_TOT\_B .EQU \$0065 ; Numero totale pezzi B.  
NR\_TOT\_SC .EQU \$0066 ; Numero totale pezzi scartati.  
  
CAP\_A .EQU \$0067 ; Capacita' contenitore pezzi A.  
CAP\_B .EQU \$0068 ; Capacita' contenitore pezzi B.  
CAP\_SC .EQU \$0069 ; Capacita' contenitore pezzi scartati.  
  
DEL\_CNT .EQU \$006A ; Contatore uso misura lunghezza pezzi.  
  
BCD\_MEM .EQU \$006B ; Locaz. temporanea uso conv bin --> BCD.

; ----- REGISTRI 6501Q -----

PORT\_A .EQU \$0000 ; Porte di ...  
PORT\_B .EQU \$0001  
PORT\_C .EQU \$0002  
PORT\_D .EQU \$0003 ; ... I/O 6501Q.  
  
FFC .EQU \$0010 ; Registri speciali...  
MCR .EQU \$0014 ; ... interni 6501Q.  
  
IFR .EQU \$0011 ; Interrupt flag register.  
IER .EQU \$0012 ; Interrupt enable register.  
  
SCCR .EQU \$0015 ; Serial communication control register.  
SCDR .EQU \$0017 ; Serial communication data register.  
  
TAL .EQU \$0018 ; Timer A ...  
TAH .EQU \$0019  
CAL .EQU \$001A ; ... 6501Q.  
  
TBL .EQU \$001C ; Timer B ...  
TBH .EQU \$001D  
CBL .EQU \$001E ; ... 6501Q.

; \*\*\*\*\* PROGRAMMA \*\*\*\*\*

.ORG PROG ; Origine del codice macchina.  
  
MAIN SEI ; Inizializza ...  
CLD  
LDX #\$FF  
TXS ; ... CPU.  
  
LDA #\$20 ; Polarizza PORT\_D ...  
STA MCR ; ... come uscita.  
  
LDA #5 ; Lunghezza minima...  
STA LUN\_MIN\_A ; ... pezzi A.  
LDA #10 ; Lunghezza massima...  
STA LUN\_MAX\_A ; ... pezzi A.  
  
LDA #15 ; Lunghezza minima...  
STA LUN\_MIN\_B ; ... pezzi B.  
LDA #20 ; Lunghezza massima...  
STA LUN\_MAX\_B ; ... pezzi B.  
  
LDA #10 ; Capacita' massima...  
STA CAP\_A ; ...scatola A.

```

STA CAP_B           ; ...scatola B.
LDA #15            ; Capacita' massima...
STA CAP_SC         ; ...scatola scarti.

LDA #0             ; Azzeramento...
STA PORT_D        ; ...uscita.

STA NR_TOT_A      ; Azzerata ....
STA NR_TOT_B
STA NR_TOT_SC     ; ... contatori pezzi.

CICLO   JSR DISPLAY           ; Visualizzazione dati.
        BBR 0,PORT_B,CICLO   ; Controlla l'ingresso 0.
CICLO1  JSR CONTROL_S        ; Salta a CONTROL_S.
        BCS CICLO           ; Salta a CICLO se CARRY=1
        LDA #0              ; Azzerata ...
        STA DEL_CNT         ; ... il contatore DEL_CNT.
        BBR 2,PORT_B,CICLO1 ; Fotocellula OFF ?
FOTO    JSR CONTROL_S        ; Salta a CONTROL_S.
        BCS CICLO           ; Salta a CICLO se CARRY=1
        INC DEL_CNT         ; Incrementa il contatore DEL_CNT.
        JSR DELAY_C         ; Esegui subroutine DELAY_C.
        BBR 2,PORT_B,FOTO    ; Fotocellula ancora ON ?
        LDA DEL_CNT         ; Trasferisci DEL_CNT in A ...
        CMP LUN_MIN_A       ; ... confrontandolo con LUN_MIN_A.
        BCS CICLO4         ; Salta a CICLO4 se CARRY=1
        JSR CONTROL_S       ; Salta a CONTROL_S.
CICLO2  SMB 2,PORT_D
CICLO3  JSR DISPLAY           ; Visualizzazione dati.
        BBR 5,PORT_B,CICLO3 ; Controlla l'ingresso 5.
        RMB 2,PORT_D
        INC NR_TOT_SC       ; Incrementa numero di scarti.
        LDA NR_TOT_SC       ; Trasferisci NR_TOT_SC nell'A...
        CMP CAP_SC         ; ...confrontandolo con CAP_SC
        BCC CICLO1
        SMB 5,PORT_D
        JSR IMPULSO_T       ; Salta a IMPULSO_T.
        RMB 5,PORT_D
        LDA #0              ; Azzerata ...
        STA NR_TOT_SC       ; ... contatori pezzi.
        JMP CICLO1         ; Salta a CICLO1.
        JSR CONTROL_S       ; Salta a CONTROL_S.
CICLO4  CMP LUN_MAX_A       ; Confronta A con LUN_MAX_A.
        BCS CICLO7         ; Salta a CICLO7 se CARRY=1.
CICLO5  SMB 0,PORT_D
CICLO6  JSR DISPLAY           ; Visualizzazione dati.
        BBR 3,PORT_B,CICLO6 ; Controlla l'ingresso 3.
        RMB 0,PORT_D
        INC NR_TOT_A        ; Incrementa numero di pezzi.
        LDA NR_TOT_A        ; Trasferisci NR_TOT_A nell'A...
        CMP CAP_A          ; ...confrontandolo con CAP_A.
        BCC CICLO1
        SMB 3,PORT_D
        JSR IMPULSO_T       ; Salta a IMPULSO_T.
        RMB 3,PORT_D
        LDA #0              ; Azzerata ...
        STA NR_TOT_A        ; ... contatori pezzi.
        JMP CICLO1         ; Salta a CICLO1.
        JSR CONTROL_S       ; Salta a CONTROL_S.
CICLO7  BEQ CICLO5
        CMP LUN_MIN_B       ; Controlla A con LUN_MIN_B.
        BCC CICLO2         ; Salta a CICLO2 se CARRY=0
        CMP LUN_MAX_B       ; Confronta A con LUN_MAX_B.
        BCC CICLO8         ; Salta a CICLO8 se CARRY=0
        BNE CICLO2
        JSR CONTROL_S       ; Salta a CONTROL_S.
CICLO8  SMB 1,PORT_D
CICLO9  JSR DISPLAY           ; Visualizzazione dati.
        BBR 4,PORT_B,CICLO9 ; Controlla l'ingresso 4.
        RMB 1,PORT_D
        INC NR_TOT_B        ; Incrementa numero di pezzi.

```

```

        LDA NR_TOT_B          ; Trasferisci NR_TOT_B nell'A...
        CMP CAP_B            ; ...confrontandolo con CAP_B.
        BCS CICLO10
        JMP CICLO1
CICLO10 SMB 4,PORT_D
        JSR IMPULSO_T        ; Salta a IMPULSO_T.
        RMB 4,PORT_D
        LDA #0              ; Azzerata ...
        STA NR_TOT_B        ; ... contatori pezzi.
        JMP CICLO1         ; Salta a CICLO1.

        JSR CONTROL_S       ; Salta a CONTROL_S.
;-----
; Questa sub-routine gestisce la visualizzazione dati ed il controllo
; del pulsante di STOP.

CONTROL_S JSR DISPLAY        ; Visualizza dati.
          CLC                ; Azzerata il CARRY.
          BBR 1,PORTB,CK_END
          LDA #0             ; Carica 0 ...
          STA PORT_D         ; ...nel PORTD.
          SEC
CK_END    RTS                ; Ritorna al programma.

;-----
; Questa sub-routine gestisce il ritardo.

DELAY_C   LDX #0            ; Carica 0 in X.
          LDY #100          ; Carica 80 in Y.
DELAY_C0  NOP
          NOP
          NOP
          DEX                ; Decrementa X.
          BNE DELAY_C0      ; Se non zero torna a DELAY_C0.
          DEY                ; Decrementa Y.
          BNE DELAY_C0      ; Se non zero torna a DELAY_C0.
          RTS                ; Ritorna al programma.

;-----
; Questa sub-routine gestisce il ritardo del cambio delle scatole.

IMPULSO_T LDA #9           ; Carica 9 in accumulatore.
IT0       PHA              ; Salva A in stack.
          JSR DELAY_C       ; Salta a DELAY_C.
          JSR DISPLAY       ; Visualizzazione dati.
          PLA              ; Recupera A da stack.
          SEC              ; Decrementa ...
          SBC #1           ; ... counter.
          BNE IT0          ; Confronta con IT0.
          RTS              ; Ritorna al programma.

;-----
; Questa sub-routine gestisce la visualizzazione dati su display, in base
; ai selettori connessi ai bit 6 (Sel 0) e 7 (Sel 1) del port B.
; La selezione e' cosi' stabilita:
;
; Sel 1   Sel 0
;
;   0       0   = Numero totale pezzi A
;   0       1   = Numero totale pezzi B
;   1       0   = Numero totale scarti
;   1       1   = Lunghezza corrente pezzo (da fotocellula).
;
; Quando il sistema visualizza gli scarti i display lampeggiano.

DISPLAY  LDA PORT_B        ; Input selettori.
          AND #$C0         ; Isola bit 6 e 7.
          CMP #$0         ; Combinaz. selett. 00 ?
          BNE DISP_0      ; Se no continua.

```

```

DISP_0      LDA NR_TOT A           ; Selezione. visualizz. pezzi A.
            JMP DISP_NOBL         ; Salta a visualizzaz. valore.
            CMP #$40              ; Combinaz. selett. 01 ?
            BNE DISP_1           ; Se no continua.
            LDA NR_TOT B           ; Selezione. visualizz. pezzi B.
            JMP DISP_NOBL         ; Salta a visualizzaz. valore.
DISP_1      CMP #$80              ; Combinaz. selett. 10 ?
            BNE DISP_2           ; Se no continua.
            LDA NR_TOT_SC         ; Selezione. visualizz. scarti.
            RMB 2,PORT_C          ; Display blinking ON.
            JMP DISP_3            ; Salta a visualizzaz. valore.
DISP_2      LDA DEL_CNT           ; Selezione. visualizz. lungh.

DISP_NOBL   SMB 2,PORT_C          ; Display blinking OFF.
            JMP DISP_3            ; A visualizzaz.

DISP_3      SMB 3,PORT_C          ; LE\ = ON.
            JSR BIN_BCD           ; Conversione binario --> BCD.

```

```

; Inversione nibble per adattarsi alla
; scheda hardware di visualizzazione.

```

```

            STA BCD_MEM           ; Copia di A.
            LSR A                 ; High nibble A ...
            LSR A
            LSR A
            LSR A                 ; ... in low nibble.
            ASL BCD_MEM           ; Low nibble mem...
            ASL BCD_MEM
            ASL BCD_MEM
            ASL BCD_MEM           ; ... in high nibble.
            ORA BCD_MEM           ; Ricomposizione byte.

DISP_4      LDX #8                ; Numero di shift da eseguire.
            RMB 1,PORT_C          ; Clock = OFF.
            ROL A                 ; Shift bit 7 di A in C.
            BCS DISP_5           ; Bit attivo ?.
            RMB 0,PORT_C         ; Dato = 0.
            JMP DISP_6            ; Continua.
DISP_5      SMB 0,PORT_C         ; Dato = 1.
DISP_6      NOP                  ; Ritardo ...
            NOP
            NOP
            NOP
            NOP                   ; ... stabilizzazione out.
            SMB 1,PORT_C         ; Clock = ON.
            NOP                  ; Ritardo ...
            NOP
            NOP
            NOP                   ; ... stabilizzazione out.
            DEX                  ; Counter iterazioni.
            BNE DISP_4           ; Finito ?
            RMB 3,PORT_C         ; LE\ = OFF.
            RTS

```

```

;-----
; Questa sub-routine converte in BCD il dato binario presente in
; Accumulatore.

```

```

BIN_BCD     TAX                  ; Trasferisci A in X.
            LDA BCD_TAB,X        ; Valore da tabella.
            RTS

```

```

BCD_TAB
.BYTE $00,$01,$02,$03,$04,$05,$06,$07,$08,$09
.BYTE $10,$11,$12,$13,$14,$15,$16,$17,$18,$19
.BYTE $20,$21,$22,$23,$24,$25,$26,$27,$28,$29
.BYTE $30,$31,$32,$33,$34,$35,$36,$37,$38,$39
.BYTE $40,$41,$42,$43,$44,$45,$46,$47,$48,$49
.BYTE $50,$51,$52,$53,$54,$55,$56,$57,$58,$59

```

.BYTE \$60,\$61,\$62,\$63,\$64,\$65,\$66,\$67,\$68,\$69  
.BYTE \$70,\$71,\$72,\$73,\$74,\$75,\$76,\$77,\$78,\$79  
.BYTE \$80,\$81,\$82,\$83,\$84,\$85,\$86,\$87,\$88,\$89  
.BYTE \$90,\$91,\$92,\$93,\$94,\$95,\$96,\$97,\$98,\$99

.END