ICL51

manuale di riferimento

release 4.0

Detail®

ELETTRONICA INDUSTRIALE

ICL51 - release 4.0
Febbraio 1997

Il costruttore si riserva di modificare, senza alcun preavviso, le caratteristiche del software riportato in questo manuale.

Ogni cura è stata posta nella raccolta e nella verifica della documentazione contenuta in questo manuale, tuttavia la proprietà del marchio ***Detail***® non può assumersi alcuna responsabilità derivante dall'utilizzo della stessa.

Sommario

Introduzione

Aggiornamenti della release 4.0

La release 4.0 si propone principalmente l'obiettivo di ottimizzare il pacchetto software ICL51. A questo scopo è stata completamente rivista l'impostazione globale del sistema di sviluppo tenendo soprattutto in considerazione gli aspetti di modularità ed espandibilità dello stesso.

L'introduzione continua di nuove Logiche ha richiesto lo studio di un sistema di sviluppo con un' impostazione del tutto generica ed automaticamente adattabile e riconfigurabile: con la release 4.0 basta aggiungere nella directory del software i files relativi alla nuova Logica ed il sistema sarà immediatamente pronto a lavorare anche con questa.

Il software ICL51 release 4.0 si presenta al Programmatore in modo molto più pratico e diretto; i menu sono stati ridotti e semplificati e le fasi di messa a punto del programma vengono suggerite automaticamente da un diagramma di flusso. Tutto questo per non costringere il Programmatore a girare continuamente in una "giungla" di menu e sottomenu per svolgere anche le più elementari operazioni. Semplicità ed immediatezza d'uso sono alla base delle scelte dell'impostazione di tutto il pacchetto software: il Programmatore dovrà solo concentrarsi sul programma della macchina da automatizzare e non sul software di programmazione.

Le diverse fasi di sviluppo del programma come l'editing, la compilazione, il trasferimento ed il monitor vengono eseguite singolarmemte da appositi programmi ottimizzati per ogni diversa Logica. In questo modo il menu principale, richiamato dal programma ICL51, funziona solo da coordinatore dei vari programmi caricando nella memoria del PC solo lo stretto necessario per la fase in corso. E' per questo che la nuova release 4.0 occuppa una quantità di memoria RAM esigua rispetto alle precedenti versioni, escludendo in questo modo ogni possibile situazione di saturazione della memoria del PC disponibile all'esecuzione dei programmi.

La release 4.0 è stata particolarmente potenziata per quanto riguarda il programma di monitor. La lista delle variabili monitorate è diventata un'unica lista continua di 512 variabili alle quali è possibile associare anche un campo alfanumerico descrittivo. La gestione più sofisticata della connessione tra il PC e la Logica ha permesso di eliminare ogni operazione manuale di ripristino in caso di mancata comunicazione.

Non trascuriamo anche il fatto che il nuovo software ICL51 nasce in un periodo di forte esplosione del mondo Internet. La nuova struttura del pacchetto consente un facile aggiornamento dello stesso mediante la rete Internet; inoltre questo manuale è stato completamente riscritto sfruttando le potenzialità dei collegamenti ipertestuali ed è disponibile come file di tipo .PDF, divenuto ormai uno standard nella diffusione della documentazione.

Elementi generali

Il linguaggio ICL51 è stato appositamente studiato per consentire la programmazione ad alto livello di un sistema elettronico complesso, realizzato dalla connessione, mediante una rete multipunto, di un certo numero di schede a microprocessore della famiglia INTEL® 80C51.

Lo scopo di tale sistema elettronico è il controllo delle funzioni di una macchina o di un impianto automatico, indipendentemente dalle dimensioni di questi. Il termine ICL51 deriva da Industrial Control Language, mentre 51 indica che il software è indirizzato a tutti i microprocessori della famiglia INTEL® 80C51, compresi tutti i suoi derivati.

La caratteristica di base del sistema elettronico, sul quale opera tale linguaggio, è quella di essere composto da una o più schede elettroniche tra loro collegate secondo una struttura di tipo MASTER/SLAVE. Il sistema minimo sarà dunque costituito da una sola scheda, che assume necessariamente la funzione di MASTER. Il sistema massimo è invece costituito da 32 schede, delle quali una opera come MASTER e le rimanenti 31 operano come SLAVES.

Ciascuna scheda, sia MASTER che SLAVE, deve avere a bordo almeno un microprocessore della famiglia 80C51. Il linguaggio ICL51 permette, tramite l'apposita compilazione, di generare il codice macchina che gira sulla scheda MASTER del sistema; le eventuali schede SLAVES hanno invece un proprio definito programma a bordo, che consente di elaborare in loco le proprie risorse e di comunicare con il MASTER. Esiste tuttavia la possibilità da parte di alcune schede di operare come SLAVE di espansione intelligente, ossia capace di eseguire localmente un proprio programma d'automazione, secondo il linguaggio ICL51, in modo parallelo a quello del MASTER. Questa modalità di funzionamento permette di realizzare sistemi molto complessi composti da un MASTER principale che controlla un insieme di SLAVES tra i quali alcuni eseguono parallelamente un proprio programma locale (sulle proprie risorse) con le stesse potenzialità di programmazione del MASTER.

Il MASTER si occupa quindi della vera e propria gestione dell'intero processo automatico, secondo il programma scritto nel linguaggio ICL51, elaborando sia le proprie risorse, sia quelle degli SLAVES. In ogni momento il MASTER ha una dettagliata visione di tutte le risorse dell'intero sistema, mentre ogni SLAVES ha visibilità solo delle proprie risorse.

Il software residente sul MASTER si occupa principalmente di quattro cose:

1) Gestione delle proprie risorse
2) Comunicazione con gli SLAVES per le loro risorse
3) Comunicazione con il COMPUTER collegato
4) Esecuzione del programma utente compilato da ICL51

Mentre le parti 1, 2, 3 fanno parte del sistema operativo del MASTER e quindi sempre fisse, la parte 4 è variabile da applicazione ad applicazione, in quanto rappresenta la parte di codice oggetto generata dalla compilazione del linguaggio sorgente ICL51, descrivente il ciclo di funzionamento della macchina.

Struttura del linguaggio ICL51

Il linguaggio ICL51 permette di descrivere il funzionamento di una macchina automatica semplicemente editando un FILE sorgente, secondo le regole sintattiche stabilite. Il FILE sorgente (.PRG), una volta editato, verrà sottoposto ad un processo di compilazione il quale genererà il FILE oggetto (.OBJ), successivamente caricato sulla scheda MASTER.

Questo linguaggio è del tipo LISTA ISTRUZIONI, ossia una sequenza di righe nel file sorgente, ciascuna delle quali esprime un'istruzione del linguaggio completa di operandi.

Un'istruzione sarà dunque costituita da una sola riga, divisa in un certo numero di campi. Il primo campo è sempre la parola chiave dell'istruzione, mentre gli altri campi rappresentano l'argomento dell'istruzione ossia la lista degli eventuali operandi richiesti dalla stessa; un eventuale ultimo campo, preceduto dal carattere ' (apice), viene considerato commento, ossia un testo scritto come promemoria dell'istruzione:

OPERAZIONE [OPERANDO1] [OPERANDO2] [OPERANDO3] ['COMMENTO]

I vari campi di una riga istruzione devono essere separati almeno uno spazio (BLANK) oppure almeno un tabulatore (TAB).

Il file sorgente può contenere, in qualunque punto, delle righe vuote (CR), oppure delle righe di solo testo, allo scopo di COMMENTO, facendole precedere dal carattere apice.

I commenti, preceduti dal carattere apice, possono essere alla destra di un'istruzione, oppure occupare da soli un'intera riga del file di programma. Questi commenti vengono completamente ignorati dal compilatore e quindi rimangono solo dei promemoria del file sorgente editato.

Esiste un altro tipo di commento utilizzabile all'interno del file di programma; a differenza del precedente questo tipo di commento viene elaborato dal compilatore e fatto diventare parte integrante del codice caricato sulla scheda MASTER. Questi commenti non possono essere presenti alla destra di un'istruzione, ma devono costituire da soli una riga del file programma, come se fossero delle istruzioni eseguibili; inoltre per far sì che tali commenti siano distinguibili dai precedenti, occorre farli precedere dal carattere " (doppio apice).

I commenti, preceduti dal carattere doppio apice, vengono elaborati dal compilatore mediante un algoritmo di codifica segreta e quindi collocati in un'area della memoria di programma del MASTER. La memoria totale disponibile per tali commenti è di 8176 bytes, per cui occorre fare attenzione nell'uso di tali commenti; in ogni caso il compilatore avviserà il Programmatore quando lo spazio di memoria commenti verrà esaurito.

Tutti i commenti del tipo doppio apice vengono archiviati nella memoria programma, finché vi è spazio sufficiente; il loro recupero può avvenire semplicemente caricando il programma dalla scheda MASTER al Computer e decodificando tali informazioni. Il risultato è la creazione di un file .TXT in formato ASCII e quindi stampabile; si consideri che questo file conterrà tutte le righe commento del tipo doppio apice consecutivamente e nello stesso ordine presente sul listato sorgente. L'operazione di recupero sarà possibile solo conoscendo la password dichiarata nel programma sorgente e quindi le informazioni rimarranno riservate al solo Programmatore.

Consideriamo ora un'ulteriore possibilità offerta dal compilatore, ossia quella di associare, ad ogni operando, una stringa mnemonica di lunghezza massima 32 caratteri. Tale associazione la si effettua editando nel file sorgente una riga del tipo:

```
LABEL = OPERANDO ['COMMENTO]
```

Con LABEL si intende un'etichetta (stringa max. 32 caratteri alfanumerici) che in genere ricorda il significato di quel operando; ponendo tale associazione in una riga del programma, tutte le istruzioni nelle righe successive, potranno avere, come operandi, indifferentemente la loro dicitura originaria oppure la LABEL mnemonica. In genere è conveniente porre tali associazioni in testa al programma, in modo da farle valere per tutte le istruzioni successive. Si tenga conto che molti programmi EDITOR di file permettono la sostituzione automatica di una certa stringa con un'altra; sfruttando tale funzione, è possibile sostituire, nel campo operando di un'istruzione, la LABEL con l'identificatore originario o viceversa. Il compilatore, in ogni caso, riuscirà ad elaborare l'istruzione in modo del tutto equivalente.

E' possibile anche un ulteriore tipo di LABEL da utilizzare con le istruzioni di salto condizionato e di richiamo di subroutines: questo tipo di LABEL individua il punto esatto del listato al quale saltare con le istruzioni GOTO e GOSUB. Essa è costituita da una stringa (max 32 caratteri alfanumerici) seguita, senza spazi intermedi, dal carattere : (due punti) e deve occupare da sola un'intera riga del file sorgente. Ad esempio:

```
GOTO            PUNTO_DOVE_SALTARE

.............
.............

PUNTO_DOVE_SALTARE:
```

A conclusione ricordiamo che l'ultima istruzione della lista deve essere l'istruzione END di fine programma, che consente al sistema operativo della logica di sapere che è terminata la scansione del programma utente e quindi permettere l'esecuzione delle altre fasi descritte nell'introduzione. Nel caso siano presenti subroutines, possono comparire ulteriori istruzioni END poichè la lista istruzioni di ogni subroutine deve essere terminata con una propria istruzione END.

Un esame dettagliato di tutte le istruzioni del linguaggio ICL51 e della loro sintassi sarà effettuato nel seguito.

Le risorse del sistema

Le risorse esterne del sistema

Per RISORSA del sistema si intende un qualunque elemento messo a disposizione sia dall'hardware che dal software e che può utilmente essere sfruttato dal programma utente allo scopo di gestire il processo automatico.

Definiamo risorsa ESTERNA qualunque risorsa del sistema complessivo (MASTER + SLAVES) che sia in qualche modo riferita ad un MORSETTO delle schede, oppure abbia una qualche funzione di comunicazione con il mondo esterno, mediante dispositivi a bordo delle schede. Sono risorse esterne, ad esempio, gli I/O digitali ed analogici sia del MASTER che degli SLAVES, i caratteri di un display, i tasti di un terminale, ecc.

Una risorsa INTERNA è invece messa a disposizione solo ed esclusivamente dal MASTER e non ha alcun riferimento a ciò che è esterno alla scheda del MASTER stesso; sono risorse interne le memorie di lavoro, i contatori, i timers e tutti quei dispositivi realizzati dal sistema operativo della Logica. Si consideri che generalmente le risorse interne sono ottenute mediante simulazione software da parte del sistema operativo del MASTER e che quindi non hanno un diretto riferimento con dispositivi hardware a bordo della scheda.

La suddivisione in risorse esterne e risorse interne ci permette di spezzare in due l'analisi preliminare del linguaggio di programmazione ICL51.

Nel paragrafo precedente abbiamo parlato di istruzione e di relativi operandi. Gli operandi non sono altro che delle risorse, poiché sono proprio queste a dover essere elaborate dalle istruzioni allo scopo di realizzare un controllo automatico della macchina.

Concentrando l'attenzione sulle sole risorse esterne, possiamo immaginare che tutte queste siano disponibili nella memoria RAM del MASTER, sottoforma di operandi elaborabili dalle istruzioni del programma utente.

Poiché si vuole prescindere, per generalità del sistema, da quanti SLAVES saranno presenti e dalla loro natura e funzione svolta, si considera che la parte di RAM del MASTER, abbinata alle risorse esterne, sia equamente suddivisa in 32 aree di dimensione fissa (128 bytes), ciascuna assegnata ad ogni scheda del sistema.

Tali aree numerate da 0 a 31 (dove la 0 è per le risorse esterne del MASTER e le 1÷31 per le risorse esterne degli SLAVES) consentono al programma utente di avere accesso, in ogni momento, a tutte le risorse esterne dell'intero sistema, così come se fossero fisicamente presenti sull'unica scheda del MASTER.

Ciascuna di tali aree è rappresentativa di ogni scheda presente nel sistema distribuito; dei 128 bytes otto sono riservati al sistema operativo, mentre la maggior parte (120 bytes) sono potenzialmente disponibili per le risorse esterne della relativa scheda.

Ad esempio se il MASTER è una scheda con le seguenti risorse esterne:

24 Ingressi digitali ON/OFF (3 bytes)
16 Uscite digitali ON/OFF (2 bytes)
4 Ingressi analogici ad 8 BITS (4 bytes)

l'area numero 0 utilizzerà 3+2+4=9 bytes per rappresentare le risorse esterne del MASTER.
Se lo SLAVE numero 3 è una scheda con le seguenti risorse esterne:

8 Ingressi digitali ON/OFF (1 bytes)
8 Uscite digitali ON/OFF (1 bytes)
4 Uscite analogiche ad 8 BITS (4 bytes)

l'area numero 3 utilizzerà 1+1+4=6 bytes per rappresentare le risorse esterne di tale scheda periferica.

Qualunque dei 128 bytes di ognuna delle 32 aree di tale parte di RAM del MASTER, può essere un valido operando per quelle istruzioni che richiedono, come tale, uno o più bytes, sia in lettura che in scrittura (istruzioni su bytes); per di più, ciascuno degli 8 bits di ognuno dei 128 bytes costituenti ogni area associata ad una scheda, possono essere utilizzati, sia in lettura che in scrittura, da tutte quelle istruzioni che agiscono sui singoli bits (istruzioni booleane).

Questa impostazione fornisce al sistema una grande potenzialità ed apertura verso futuri sviluppi di schede SLAVES completamente nuove, sia come funzioni che come dimensioni, senza precludere la strada allo studio di schede SLAVES progettate appositamente per una certa applicazione, per la quale non è possibile utilizzare le schede periferiche standard fino a quel punto sviluppate.

Resta ora da individuare il modo più semplice possibile per identificare univocamente, mediante un codifica, uno qualunque dei 32*128=4096 bytes oppure uno qualunque dei 4096*8=32768 bits delle risorse esterne possibili del sistema complessivo.

La soluzione più semplice è quella di numerare le 32 aree, corrispondenti ciascuna ad una possibile scheda presente nel sistema, con un numero da 0 a 31; fissata la scheda, si possono poi numerare i 128 bytes di ogni area con un numero da 0 a 127; infine si possono numerare gli 8 bits di ogni byte partendo da 0 per il LSB (bit meno significativo) ed arrivando a 7 per il MSB (bit più significativo).

Separando infine i tre numeri identificatori con il carattere . (punto), si vengono ad introdurre gli operandi per le risorse esterne del sistema:

Scheda.Byte.Bit (Scheda=0÷31, Byte=0÷127, Bit=0÷7)

Ad esempio consideriamo alcuni operandi di tipo byte e di tipo bit di un sistema costituito da un MASTER (con 3*8=24 ingressi digitali allocati ai bytes di indirizzo 0÷2, più 2*8=16 uscite digitali allocate ai bytes 8÷9, più 4 ingressi analogici a 8 bits allocati ai bytes 4÷7) ed uno SLAVE di indirizzo 2 (con 2*8=16 ingressi digitali allocati ai bytes 0÷1, più 1*8=8 uscite digitali allocate al byte 8):

0.0 canale ingressi (byte) numero 0 del MASTER
0.8 canale uscita (byte) numero 8 del MASTER
2.1 canale ingresso (byte) numero 1 dello SLAVE 2
2.8 canale uscita (byte) numero 8 dello SLAVE 2
0.4 primo ingresso analogico (byte) del MASTER
0.0.5 ingresso digitale 5 del canale 0 del MASTER
0.9.6 uscita digitale 6 del canale 9 del MASTER
2.1.3 ingresso digitale 3 del canale 1 dello SLAVE 2
2.8.4 uscita digitale 4 del canale 8 dello SLAVE 2

Abbiamo in questo modo illustrato il primo passo dal quale nasce, in modo del tutto naturale, la definizione della sintassi del linguaggio di programmazione ICL51; infatti con tali identificatori simbolici delle risorse esterne si sono introdotti i primi simboli del linguaggio.

Diamo ancora delle indicazioni fondamentali, per chi si avvicina all'uso di tale linguaggio, sempre sull'argomento delle risorse esterne.

Qualunque sia l'operando (di tipo byte o di tipo bit) l'identificatore usato nel linguaggio deve avere, come primo numero, il numero di scheda. Su ogni scheda del sistema è possibile definire un indirizzo mediante un certo numero di selettori di tipo Dip-Switch, Jumper oppure mediante selezione software. La scheda MASTER sarà sempre configurata come scheda 0 mentre tutti gli SLAVES avranno invece un numero identificativo compreso tra 1 e 31. E' ovviamente vietato configurare due SLAVES con lo stesso numero di scheda, mentre è del tutto arbitrario scegliere per ciascuno SLAVE un numero da 1 a 31.

A questo proposito dobbiamo fare un'ulteriore precisazione: l'aggiornamento delle risorse degli SLAVES avviene in modo dinamico rispetto all'indirizzo massimo di scheda effettivamente utilizzato in una qualche istruzione del programma. In pratica se nel definire i numeri di scheda degli elementi del sistema complessivo, si utilizza la progressione numerica crescente dall'indirizzo 0 del MASTER, si ottengono considerevoli incrementi di velocità di esecuzione del programma macchina. Questo perché il compilatore automaticamente comunica al sistema operativo del MASTER l'indirizzo massimo di scheda incontrato nel programma utente; in questo modo il MASTER non tenterà assolutamente di comunicare con le schede di indirizzo superiore.

Si consiglia quindi di scegliere gli indirizzi di schede SLAVES consecutivamente a partire dal numero 1, ponendo cura a non lasciare spazi vuoti nella numerazione; ciò consentirà al sistema operativo di raggiungere le massime prestazioni in termini di velocità.

Il secondo numero dell'identificatore di un qualunque operando (sia byte che bit) rappresentante una risorsa esterna, deve essere sempre il numero di byte dell'area costituita dai 128 bytes corrispondenti ad ogni scheda. Le specifiche funzioni della scheda (sia MASTER che SLAVE) definiranno in quali numeri di bytes trovare le risorse offerte. Per ogni scheda fare riferimento alle relative documentazioni per individuare i numeri di indirizzo dei bytes corrispondenti alle varie risorse, sia di ingresso (lettura del byte) sia di uscita (scrittura del byte).

Ciò che è definito a priori è che per ogni singola area di scheda tutti i bytes corrispondenti a risorse in ingresso nel sistema sono contigui, così come sono contigui tutti i bytes corrispondenti a risorse in uscita dal sistema. Entrambi i blocchi dei bytes di ingresso e di uscita sono ovviamente non sovrapposti e possono essere separati da bytes non usati, purché tali blocchi siano compresi entro l'area dei bytes 0÷119. I rimanenti 8 bytes (120÷127) sono riservati al sistema operativo allo scopo di memorizzare alcune informazioni, come l'identificazione dello SLAVE trovato su quell'indirizzo, oppure i punti di partenza e le lunghezze dei blocchi di ingresso ed uscita delle risorse esterne.

Tutti i bytes (compresi in 0÷119) delle aree di scheda, non utilizzati dalle risorse esterne, sono disponibili come memorie interne di tipo non ritentivo e cioè azzerate ad ogni accensione del sistema; fare tuttavia attenzione nell'uso di istruzioni di scrittura sui bytes riservati al sistema operativo (120÷127), poiché è possibile creare dei problemi a questo stesso. L'utilizzo degli 8 bytes riservati al sistema operativo consente, per esempio, di sospendere momentaneamente, sotto certe condizioni, la comunicazione con lo SLAVE (ponendo a 0 il numero di bytes trasmessi e ricevuti), abbassando così il tempo di esecuzione del programma.

Il terzo ed eventuale numero nell'identificatore di una risorsa esterna, individua uno degli 8 bits del canale (ossia del byte). Sarà il compilatore ad accorgersi, segnalando errore, se una certa istruzione richiede, come operandi, degli identificatori di tipo byte oppure degli identificatori di tipo bit.

Quello che va sottolineato è che tutti i bits e tutti i bytes di tutte le 32 aree delle schede, possono essere sia scritti che letti dal programma utente sempre nello stesso identico modo, qualunque sia la natura delle schede. E' compito del sistema operativo del MASTER di trascrivere, ad ogni ciclo di scansione, tutti i bytes di uscita della RAM del MASTER verso le proprie risorse di uscita e verso quelle degli SLAVES; inoltre, sempre in modo del tutto invisibile al programma utente, il sistema operativo del MASTER effettuerà una lettura di tutte le proprie risorse di ingresso e di quelle degli SLAVES, trascrivendole negli appositi bytes di ingresso della RAM utente.

I bytes riservati delle risorse esterne

Tutte le risorse esterne del sistema vengono memorizzate nella memoria RAM dati del MASTER; ciascuna scheda, per le proprie risorse, occupa un'area di 128 bytes contigui. A seconda della natura della scheda considerata, tale area sarà occupata in parte dai bytes delle risorse di ingresso del sistema (per esempio i bytes corrispondenti alle porte di ingressi digitali provenienti dai sensori della macchina) ed in parte dai bytes delle risorse di uscita del sistema (per esempio i bytes corrispondenti alle porte di uscite digitali che alimentano gli attuatori della macchina).

I primi 120 bytes associati ad ogni scheda (dal byte 0 al byte 119) sono utilizzati per le operazioni di INPUT/OUTPUT delle risorse esterne (secondo una mappatura caratteristica di ogni tipo di scheda e per la quale rimandiamo alle specifiche di questa), mentre gli ultimi 8 (dal byte 120 al byte 127) sono riservati al sistema operativo; questo non significa che, per una programmazione avanzata ed esperta, non sia possibile leggere e scrivere tali bytes (con le ovvie precauzioni).

Lasciando la spiegazione dettagliata dei primi 120 bytes alla documentazione specifica di ogni singola scheda, analizziamo nel dettaglio il significato dei bytes riservati al sistema operativo. La Tabella 1 riassume tali informazioni:

BYTES RISERVATI DELLE RISORSE ESTERNE		
BYTE RISERVATO	DESCRIZIONE	VALORE
*.120	Numero totale di bytes di uscita della scheda SLAVE	0 - 120
*.121	Numero totale di bytes di ingresso della scheda SLAVE	0 - 120
*.122	Indirizzo iniziale area bytes di uscita della scheda SLAVE	0 - 119
*.123	Indirizzo iniziale area bytes di ingresso della scheda SLAVE	0 - 119
*.124	Byte 0 di codice di identificazione della scheda SLAVE	0 - 255
*.125	Byte 1 di codice di identificazione della scheda SLAVE	0 - 255
*.126	Byte 2 di codice di identificazione della scheda SLAVE	0 - 255
*.127	Byte 3 di codice di identificazione della scheda SLAVE	0 - 255

Tabella 1. Significato dei bytes riservati

Questi bytes, utilizzati dal sistema operativo del MASTER per le operazioni di comunicazione con gli SLAVES, possono essere modificati dal programma utente con le normali istruzioni previste dal linguaggio.

All'accensione del sistema complessivo, il MASTER effettua un "appello" di tutti gli SLAVES dall'indirizzo 1 all'indirizzo 31, indipendentemente dalla presenza fisica di questi. Con tale appello il MASTER si preoccupa di stabilire la configurazione del sistema, in modo da poter procedere ad una efficiente gestione di tutte le risorse esterne.

Le risorse esterne vengono comunicate tra il MASTER e gli SLAVES tramite la linea che collega in rete tutte le schede. E' ovvio che il modo più rapido di eseguire le operazioni di aggiornamento dei valori correnti delle risorse, è quello di trasmettere il numero di bytes minimo necessario; gran parte degli SLAVES disporranno di risorse corrispondenti a pochi bytes, per cui non è assolutamente necessario comunicare tutti i possibili 120 bytes dell'area associata ad ogni scheda. Inoltre è completamente inutile comunicare con quegli indirizzi di SLAVES, sui quali non è presente fisicamente alcuna scheda.

Per questo, all'accensione, il MASTER prova a comunicare con tutte le possibili 31 schede SLAVES; ciascuna scheda SLAVE, trovata presente sulla rete, fornirà al MASTER gli 8 bytes contenenti tutte le informazioni dell'area riservata al sistema operativo. Facendo riferimento alla Tabella 1, ogni SLAVE presente trasmetterà al MASTER i valori da memorizzare nei bytes allocati nelle posizioni 120÷127.

Saranno questi i valori di default che caratterizzeranno tutti gli aggiornamenti successivi delle risorse degli SLAVES. Questo presettaggio viene effettuato automaticamente dal sistema all'accensione; come già detto una programmazione utente esperta, può variare tali parametri di comunicazione, per esempio per interrompere, sotto certe condizioni, l'aggiornamento delle risorse di alcuni SLAVES, allo scopo di diminuire il tempo di scansione del programma.

La lettura da parte del MASTER dei bytes 124 e 127 permette di avere, nel programma utente, una chiara configurazione della situazione hardware, il che permette di realizzare funzioni di controllo e allarme sullo stato dell'impianto.

Tra questi ultimi quattro byte riservati, uno di essi riveste un ruolo particolare; il byte di indirizzo 127 contiene infatti, nella posizione di peso 2^0 (LSB), il bit di errore comunicazione del relativo SLAVE. Normalmente il bit *.127.0 ha valore "0" logico; questo corrisponde ad una corretta comunicazione in rete tra MASTER e lo stesso SLAVE.

Nel caso il sistema operativo dovesse ad un tratto riscontrare un errore di comunicazione, tale bit verrà automaticamente portato allo stato "1" logico, corrispondente alla segnalazione di allarme di comunicazione per il particolare SLAVE. Il bit rimane memorizzato in tale stato anche se la comunicazione dovesse riprendere la sua normale attività; dovrà essere il programma utente ad utilizzare tale bit, segnalando l'anomalia (quando richiesto) e successivamente a resettarlo. Quest'ultima operazione sarà possibile ovviamente solo se la comunicazione ha già precedentemente ripreso le sue funzioni regolari.

Le risorse interne del sistema

Le risorse interne del sistema risiedono tutte sul MASTER e sono praticamente costituite da una collezione di dispositivi che richiamano dei particolari componenti fisici presenti in un impianto d'automazione, ma che non sono fisicamente presenti sul MASTER. Questi sono realizzati mediante simulazione software e per questo sono spesso anche molto potenti.

Risorse interne sono per esempio le memorie di lavoro del programma utente, sia considerate come bytes che come bits; sono risorse interne anche i contatori, i timers, i generatori di impulsi su fronte di variazione, ecc.

Come per le risorse esterne, è necessario, anche per le risorse interne, individuare una terminologia simbolica per poterle classificare tutte in modo univoco.

Questa volta non è necessario specificare il numero di scheda, dato che le risorse interne risiedono tutte nel MASTER e quindi si considera inutile il numero 0 identificatore di scheda.

Per evitare confusione all'interno dell'insieme dei vari dispositivi di risorse interne, conviene usare, come primo campo dell'identificatore, un CARATTERE (od una STRINGA DI CARATTERI), caratteristico del tipo di risorsa:

M	per la memoria non ritentiva (azzerata al POWER ON)
H	per la memoria ritentiva (tamponata da batteria)
C	per i contatori a 16 bits
P	per i generatori impulso su fronte OFF/ON e ON/OFF
T	per gli oscillatori periodici di riferimento
F	per i flag speciali
K	per le costanti di tipo byte
SXS	per il contatore dei cicli scansione per secondo
W	per l'orologio/calendario in tempo reale
X	per la memoria ritentiva estesa (disponibile solo su alcune Logiche)

Per analogia alle risorse esterne, successivamente a tale carattere, conviene esprimere, separato dal carattere . (punto) un secondo campo, poiché il sistema metterà a disposizione del Programmatore un certo numero di dispositivi per ogni tipo.

Ad esempio:

M.0	byte di memoria non ritentiva numero 0
M.25	byte di memoria non ritentiva numero 25
M.1023	byte di memoria non ritentiva numero 1023
H.0	byte di memoria ritentiva numero 0
H.89	byte di memoria ritentiva numero 89
H.1023	byte di memoria ritentiva numero 1023

C.0	contatore dispositivo numero 0
C.127	contatore dispositivo numero 127
P.0	generatore impulso dispositivo numero 0
P.127	generatore impulso dispositivo numero 127
W.HOUR	byte valore attuale ore del giorno
W.DAY	byte valore attuale giorno della settimana
W.YEAR	byte valore attuale dell'anno
X.0	byte di memoria ritentiva estesa numero 0
X.24567	byte di memoria ritentiva estesa numero 24567

Infine, a seconda del tipo di dispositivo, un eventuale successivo campo potrà individuare una particolare risorsa. Ad esempio, per individuare il singolo bit all'interno di un byte di tipo M, H ed X basta aggiungere un ulteriore campo numerico da 0 a 7 (separandolo con il carattere punto), seguendo quindi la stessa procedura adoperata per identificare il singolo bit all'interno di un byte di una risorsa esterna. Per individuare i bytes o i bits, specifici delle altre risorse, si aggiunge un campo identificatore opportuno; un'analisi dettagliata verrà svolta nel seguito.

Analisi delle risorse interne

Le risorse interne sono caratteristiche del solo MASTER e sono costituite dall'insieme dei dispositivi o componenti che il software mette a disposizione del Programmatore. Questi dispositivi non sono quasi mai fisicamente individuabili a bordo della scheda MASTER, perché sono realizzati mediante simulazione SOFTWARE da parte del sistema operativo.

In precedenza abbiamo visto come classificare tali dispositivi mediante i primi due campi dell'identificatore. Per ciascuno dei dispositivi delle risorse interne, si può individuare un insieme di operandi di tipo byte e di tipo bit, che ne rappresentano le particolari funzioni; ciascun tipo di risorsa interna sarà quindi rappresentata da un identificatore costituito da due o tre campi, separati dal punto, studiati nel modo più opportuno per ogni singolo tipo di dispositivo.

Per le memorie di tipo M, H ed X (disponibili solo su alcune Logiche) una rappresentazione a due campi esprime sempre un operando di tipo byte, mentre una rappresentazione a tre campi esprime sempre un operando di tipo bit: per tali risorse il metodo di identificazione è lo stesso utilizzato per le risorse esterne.

Si tenga presente che i campi numerici degli identificatori non devono necessariamente avere sempre lo stesso numero di cifre, facendoli precedere da un opportuno numero zeri; questo significa, per esempio, che per il byte di memoria numero 3 possiamo scrivere indifferentemente M.3, M.03, M.003 o M.0003.

Il dispositivo CONTATORE

Per quanto riguarda i dispositivi di tipo CONTATORE (primo campo C), il secondo campo rappresenta il numero di dispositivo considerato; il terzo campo, sempre presente, permette di indirizzare i singoli bits o i singoli bytes che caratterizzano un dispositivo contatore. I possibili 128 contatori (a 16 bits) sono ciascuno implementato mediante l'uso di 5 bytes di memoria, dei quali il primo (byte CB da "control byte") rappresenta il byte di CONTROLLO del contatore, mentre gli altri 4, a coppie di due, rappresentano il valore CORRENTE di conteggio (CL, CH) ed il valore FINALE di conteggio (FL, FH).

Indicando genericamente con * il numero di dispositivo considerato, riassumiamo il significato dei singoli bytes del dispositivo contatore:

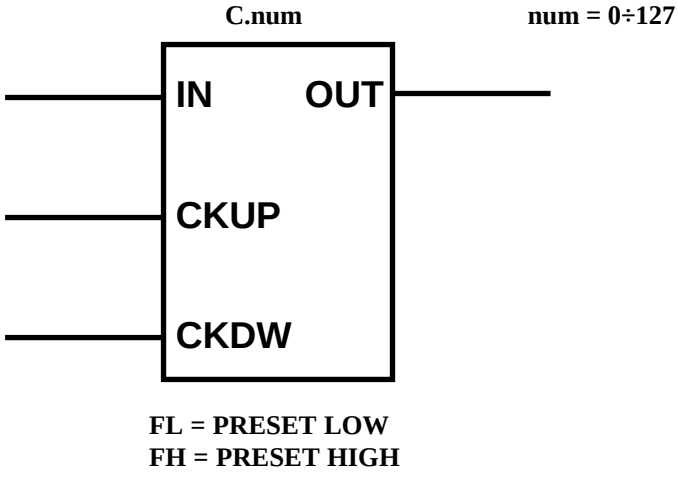
C.*.CB byte di controllo del contatore
C.*.CL byte LOW del valore corrente di conteggio
C.*.CH byte HIGH del valore corrente di conteggio
C.*.FL byte LOW del valore finale di preset del contatore
C.*.FH byte HIGH del valore finale di preset del contatore

All'interno del byte di controllo C.*.CB il significato dei singoli bits è il seguente:

C.*.IN bit di abilitazione del conteggio
C.*.OUT bit di fine conteggio (CORRENTE = FINALE)
C.*.CKUP bit di clock up del contatore
C.*.CKDW bit di clock down del contatore

Il contatore, quando trova l'ingresso C.*.IN alimentato, inizia a contare il numero di impulsi che si presentano in C.*.CKUP e in C.*.CKDW, rispettivamente incrementando e decrementando il registro contatore a 16 bits del valore corrente (CL, CH). Quando questo raggiunge il valore finale presettato nel registro a 16 bits (FL, FH), l'uscita C.*.OUT diventa ON; solo riportando l'ingresso C.*.IN in condizioni OFF, il contatore viene resettato e l'uscita ritorna anch'essa OFF.

Il dispositivo contatore può essere rappresentato come un componente con 3 ingressi (IN, CKUP, CKDW) ed 1 uscita (OUT):



I morsetti di ingresso vanno alimentati con delle normali istruzioni di uscita su bit (OUT, OUTNOT, SET, RES, CPL); se il conteggio è solo UP o solo DOWN, si può lasciare non definito rispettivamente l'ingresso CKDW oppure CKUP.

Il morsetto di uscita OUT (fine conteggio) può essere utilizzato per alimentare le altre parti della rete, utilizzandolo come operatore delle normali istruzioni booleane di lettura del bit (LD, LDNOT, AND, ANDNOT, OR, ORNOT).

Il valore di preset (FH per il byte High, FL per il byte Low) deve essere invece definito dal programma utente mediante una qualunque istruzione che forza il valore di un byte o di una coppia di bytes (ad esempio MOV1, MOV2).

Il contatore permette inoltre di realizzare un dispositivo TIMER, semplicemente connettendo l'ingresso CKUP ad uno dei bits di oscillazione periodica di periodo prestabilito. L'ingresso IN permette di alimentare il TIMER con funzione "RITARDATA ALL'ECCITAZIONE", mentre l'uscita OUT costituisce il segnale di fine tempo (l'ingresso CKDW va lasciato sconnesso).

Per far ciò, tra le risorse interne del sistema, viene messo a disposizione del Programmatore un insieme di 6 bits di oscillazioni periodiche fisse, facenti parte dello stesso byte (identificato con il carattere T):

T.50	bit oscillazione periodo 50 ms
T.100	bit oscillazione periodo 100 ms
T.200	bit oscillazione periodo 200 ms
T.500	bit oscillazione periodo 500 ms
T.1000	bit oscillazione periodo 1000 ms
T.2000	bit oscillazione periodo 2000 ms

Queste 6 diverse BASI TEMPI consentono di realizzare TIMERS con tempi che vanno da 0" a 2184.5 h.

I bits di oscillazione a periodi fissi possono essere sfruttati anche per far lampeggiare delle uscite del sistema, come per esempio nel caso di segnalatori luminosi; per far ciò basta connettere in serie (mediante l'istruzione AND) uno di tali bit con il ramo che alimenta tale uscita.

Si consiglia, per gli identificatori dei contatori, di utilizzare numeri crescenti a partire dal numero 0, evitando salti nella numerazione; questo consente di ottenere dal sistema operativo le massime prestazioni in termini di velocità di esecuzione del programma utente. Il compilatore infatti comunica al sistema operativo il numero massimo di identificatore riscontrato nel programma; in questo modo tutti i dispositivi con numero superiore non vengono gestiti, con ovvio vantaggio per il tempo di esecuzione del ciclo macchina. Si consideri che tale suggerimento ha il solo scopo di abituare il Programmatore ad ottenere il massimo delle prestazioni, se ciò non comporta limitazioni al suo modo di lavorare, anche senza tale regola il sistema garantisce dei tempi di aggiornamento dei contatori molto rapidi.

Nel linguaggio ICL51 sono presenti anche due particolari istruzioni che permettono di utilizzare in modo semplice ed immediato i dispositivi contatori; si tratta delle istruzioni CNT e TIM rispettivamente per gestire tali dispositivi come veri e propri contatori e per utilizzarli come temporizzatori con base tempi fissa a 100 ms. Per maggiori dettagli rimandiamo alla descrizione specifica delle istruzioni CNT e TIM.

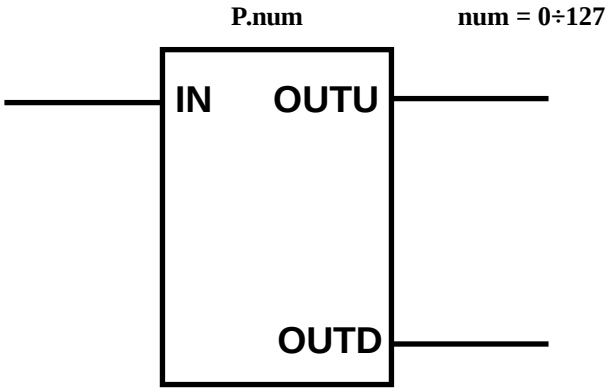
Il dispositivo GENERATORE D'IMPULSO

I dispositivi di generazione d'impulso in corrispondenza della rilevazione di un fronte di salita o di discesa del segnale di ingresso, sono altri utili strumenti per la programmazione.

Mentre il primo campo è espresso dal carattere P, il secondo campo dell'identificatore rappresenta il numero del dispositivo generatore (il numero massimo di dispositivi è 128); il terzo campo individua il bit di ingresso o i bits di uscita.

I loro identificatori si riducono quindi a tre soli bits (uno per l'ingresso P.*.IN e due per l'uscita P.*.OUTU e P.*.OUTD) con i quali è possibile rilevare il passaggio di un segnale dal valore OFF al valore ON (sull'uscita OUTU) oppure dal valore ON al valore OFF (sull'uscita OUTD). Quando il bit di ingresso subisce la transizione, sull'appropriato bit di uscita si presenta un impulso ON per tutta la durata del ciclo di scansione successivo.

Il generatore d'impulso su variazione d'ingresso lo si può rappresentare con il seguente componente:



L'ingresso IN viene alimentato mediante una normale istruzione di uscita (OUT, OUTNOT, SET, RES, CPL); le uscite OUTU e OUTD possono essere lette dalle normali operazioni booleane (LD, LDNOT, AND, ANDNOT, OR, ORNOT).

Ogni qual volta sull'ingresso IN si presenta un fronte di variazione dallo stato OFF allo stato ON, sull'uscita OUTU si presenta un impulso ON della durata di un ciclo di scansione del programma; se invece sull'ingresso si verifica una transizione dallo stato ON allo stato OFF, l'impulso ON per un ciclo di scansione lo si ha sull'uscita OUTD. In questo modo lo stesso dispositivo può servire, sia alla rilevazione dei fronti di salita, che alla rilevazione dei fronti di discesa di un certo segnale.

Si consiglia, per gli identificatori dei generatori d'impulso, di utilizzare numeri crescenti a partire dal numero 0, evitando salti nella numerazione; questo consente di ottenere dal sistema operativo le massime prestazioni in termini di velocità di esecuzione del programma utente. Il compilatore infatti comunica al sistema operativo il numero massimo di identificatore riscontrato nel programma; in questo modo tutti i dispositivi con numero superiore non vengono gestiti, con ovvio vantaggio per il tempo di esecuzione del ciclo macchina. Si consideri che tale suggerimento ha il solo scopo di abituare il Programmatore ad ottenere il massimo delle prestazioni, se ciò non comporta limitazioni al suo modo di lavorare, anche senza tale regola il sistema garantisce dei tempi di aggiornamento dei generatori d'impulso molto rapidi.

I bits speciali di FLAG

Esistono altre risorse interne con caratteristiche abbastanza particolari; ci si riferisce alle risorse di tipo F.* costituite da un singolo byte F i cui bits assumono una funzione particolare.

Le risorse di tipo F.* corrispondono ad un insieme di operandi di tipo bit con funzioni speciali (flag segnalatori).

Le risorse sono le seguenti:

F.0	bit sempre OFF
F.1	bit sempre ON
F.P	impulso ON durante il primo ciclo scansione
F.<	bit ON se la comparazione risulta <
F.=	bit ON se la comparazione risulta =
F.>	bit ON se la comparazione risulta >
F.C	bit di riporto (CARRY)
F.E	bit di errore (ERROR)

I bits F.0 ed F.1 hanno un significato di costante booleana; in particolare F.1 può essere caricata da una istruzione LD a monte di operazioni su bytes che richiedano l'alimentazione del ramo precedente, se si vuole che vengano eseguite in tutti i cicli di scansione (e non solo sotto certe condizioni, come sarebbe di regola per non aumentare eccessivamente il tempo di scansione).

Il bit F.P corrisponde ad un impulso ON durante tutto il primo ciclo di scansione: questo bit permette di inizializzare il programma ogni volta che si fornisce l'alimentazione alla macchina.

I bits F.<, F.= ed F.> permettono di verificare il risultato di un'istruzione di comparazione tra bytes; questi bits vanno letti subito dopo l'istruzione di comparazione (CMP1, CMP2, CMP4, CMP, ?, CMPBLK), per conoscere il risultato di questa: infatti la successiva istruzione di comparazione prenderà a sua volta il controllo su tali bits.

Il bit F.C corrisponde all'eventuale riporto dopo una operazione di somma o sottrazione. Inoltre è utilizzato come ingresso e come uscita per l'operazione SFR di shift, realizzando, in questo modo, SHIFT REGISTER di lunghezza qualunque (multipla di un byte).

Il bit F.E verifica con lo stato ON la presenza di un overflow dopo un'istruzione di moltiplicazione, oppure la presenza del divisore nullo prima di una istruzione di divisione; tale flag viene inoltre attivato ad "1" logico se vengono riscontrati errori durante il calcolo in un'espressione matematica con la tecnica della notazione polacca inversa.

Le COSTANTI a 1/2/4 bytes

Le risorse di tipo K.* non sono altro che degli operandi costanti; un classico uso è quello di presettaggio dei bytes del valore finale dei contatori. Il secondo campo dell'identificatore rappresenta direttamente il valore della costante di tipo intero, espresso nelle tre possibili basi di rappresentazione decimale, esadecimale e binaria.

Per costanti espresse in base decimale, occorre far seguire il simbolo K. semplicemente dal numero. Per le costanti espresse in base binaria occorre far seguire il valore binario (caratteri 0 ed 1) dal carattere B attaccandolo alla cifra meno significativa. Infine per le costanti espresse in base esadecimale (caratteri 0-9, A, B, C, D, E, F) si utilizza come suffisso il carattere H.

Per esprimere numeri interi negativi occorre far precedere il valore decimale dal carattere - (meno); la costante decimale negativa verrà automaticamente convertita dal compilatore secondo la rappresentazione binaria in complemento a due. Per esprimere le costanti negative direttamente in forma binaria occorre fornire le cifre binarie 0 ed 1 secondo la rappresentazione in complemento a due del numero.

Alcuni esempi di operandi costanti sono i seguenti:

K.0	costante decimale 0
K.-128	costante decimale -128
K.125	costante decimale 125
K.-31627	costante decimale -31627
K.2312634	costante decimale 2312634
K.-452312634	costante decimale -452312634
K.10010011B	costante binaria 10010011
K.001111101010B	costante binaria 001111101010
K.3EFH	costante esadecimale 3EF
K.E34FA4C2H	costante esadecimale E34FA4C2

Il valore costante espresso nel secondo campo, indipendentemente dalla base di rappresentazione, non deve superare i limiti consentiti dal tipo di istruzione utilizzata.

Tutte le istruzioni di manipolazione dei bytes sono suddivise in 3 categorie, a seconda della dimensione delle grandezze che trattano. Ad esempio per il trasferimento dei bytes sono presenti 3 istruzioni: la MOV1 consente di operare trasferimenti da un singolo byte su un'altro byte, la MOV2 opera su grandezze composte da due bytes consecutivi, la MOV4 opera su grandezze composte da quattro bytes consecutivi (nelle operazioni che richiedono grandezze a più di un byte, gli operandi espressi dopo l'istruzione rappresentano il byte di peso inferiore delle variabili).

Le istruzioni con operandi costanti non possono discriminare se questi sono intesi dal Programmatore come assoluti senza segno oppure come rappresentazione in complemento a due di una costante negativa e conseguentemente i campi di validità sono i seguenti:

1 BYTE: -128.....0.....127.....255
2 BYTES: -32768.....0.....32767.....65535
4 BYTES: -2147483648.....0.....2147483647.....4294967295

I precedenti campi di validità sono espressi in base decimale; in rappresentazione binaria le costanti possono avere rispettivamente al massimo 8, 16, 32 cifre binarie, mentre in rappresentazione esadecimale 2, 4, 8 cifre esadecimali.

In ogni caso una violazione di questi limiti, a seconda delle dimensioni dell'istruzione, viene segnalata automaticamente dal compilatore.

L' indicatore SXS delle scansioni per secondo

La risorsa interna di tipo SXS consiste in una variabile a due bytes di analogo nome, il cui valore viene aggiornato dal sistema operativo a scadenze di 1 secondo. Tale variabile mantiene, per tale periodo di tempo, la copia del precedente valore finale di conteggio di un contatore incrementato di una unità ad ogni ciclo di scansione.

L'utilizzo di tale variabile è generalmente limitato ad una funzione esclusivamente informativa e di controllo della velocità di elaborazione del programma sviluppato; tanto più elevato è tale valore, tanto più il programma è ottimizzato ai sensi della velocità di risposta.

Per visualizzare tale valore si può far uso della funzione di MONITOR on-line dell'ambiente di sviluppo, richiamando il monitoring di 2 bytes da SXS; si consideri, a tale proposito, che il collegamento on-line del Computer alla scheda MASTER, a causa dello scambio di dati tra le due unità, rende più lento il processo di esecuzione del programma utente, diminuendo il valore di SXS. In questo modo la lettura di tale valore, costituisce una stima particolarmente pessimistica della velocità di scansione del programma: sicuramente il numero dei cicli di scansione al secondo, durante il normale funzionamento del MASTER senza Computer connesso, sarà superiore.

A questo punto verrà da chiedersi a cosa serve tale SXS se, per leggerne il suo valore, occorre alterarlo: tale variabile può essere correttamente visualizzata mediante l'invio del suo valore a dei canali di I/O del sistema, come per esempio un terminale con DISPLAY connesso in rete. In questo modo è possibile visualizzare tale valore, senza neccessariamente connettersi con il Computer.

Un uso di tale variabile all'interno del programma utente, potrebbe essere quello di controllo del tempo massimo di scansione dello stesso, generando degli allarmi, in caso di superamento di certi limiti stabiliti, prima ancora che intervenga il circuito di WATCH-DOG hardware della scheda MASTER.

Un completo OROLOGIO/CALENDARIO

Le risorse interne di tipo W mettono a disposizione del Programmatore un completo e preciso orologio/calendario in tempo reale, controllato al quarzo.

Questo è l'unico esempio di risorsa interna legata alla presenza di dispositivi hardware specifici sulla scheda MASTER e non realizzata mediante simulazione software.

Si precisa che tale tipo di risorsa è un' OPZIONE della scheda MASTER, poiché essa può essere facilmente installata in un secondo tempo (nel caso non sia stata richiesta una scheda MASTER con tale opzione già installata). La presenza di tale risorsa determina un costo supplementare della scheda MASTER, poiché comporta la sostituzione della memoria RAM, di tipo ZERO-POWER normale, con una ZERO-POWER TIMEKEEPER, il cui costo è superiore.

Le risorse di tipo W si possono riassumere in un insieme composto da 8 bytes della memoria RAM utente. Per identificare questi bytes si introducono, dopo il solito punto separatore, i seguenti campi:

W.CB	byte di controllo dell'orologio/calendario
W.SEC	byte valore decimale dei secondi (0-59)
W.MIN	byte valore decimale dei minuti (0-59)
W.HOUR	byte valore decimale delle ore (0-23)
W.DAY	byte valore decimale giorno della settimana (1-7)
W.DATE	byte valore decimale giorno del mese (1-31)
W.MTH	byte valore decimale del mese (1-12)
W.YEAR	byte valore decimale dell'anno (0-99)

Il significato dei bytes da W.SEC ad W.YEAR è evidente: essi contengono il valore corrente dell'ora e della data in notazione decimale. Si vuole precisare che il MSB del byte dei secondi costituisce il bit di STOP dell'oscillatore quarzato dell'orologio; occorre dunque fare attenzione a non presettare tale byte con valori al di fuori del campo lecito (0-59), altrimenti è possibile che tale bit sia forzato al valore logico 1 con il conseguente arresto dell'aggiornamento dell'orologio/calendario.

Analizziamo in dettaglio il byte di controllo W.CB; tale byte utilizza solo uno dei suoi bits (bit di peso 2^0), per svolgere le funzioni di presettaggio dell'ora e della data:

W.CB.ADJ	bit di adjustment dell'orologio/calendario

Tale bit permette di presettare il valore di tutti 7 bytes dell'orologio/calendario. Il sistema operativo, ad ogni ciclo di scansione, controlla il valore di tale bit; in base al valore di questo, si comporta di conseguenza.

Se il valore di W.CB.ADJ è "0" logico, l'orologio/calendario è in modalità normale di lavoro, corrispondente alla possibilità di lettura, da parte del programma utente, dell'attuale ora e data; in questo caso il sistema provvede all'aggiornamento di tale area di memoria con il valore corrente presente nell'orologio/calendario (contenuto nel dispositivo RAM TIMEKEEPER).

Se il valore di W.CB.ADJ è “1” logico, l’orologio/calendario è in modalità di aggiustamento dell’ora e data. Il sistema trovando tale bit ON provvede a sospendere l’aggiornamento dell’area di RAM utente, poiché si aspetta che sia l’utente stesso a forzare dei valori in tali bytes allo scopo di presettare l’orologio/calendario. Quando il bit di aggiustamento viene riportato OFF, il sistema operativo della logica preleva il valore forzato nella RAM utente e lo trasferisce nel dispositivo RAM TIMEKEEPER, ritornando nella normale modalità di lettura dell’ora/data.

Normalmente il bit W.CB.ADJ deve quindi essere tenuto allo stato OFF, come del resto si trova ogni volta che la scheda MASTER viene alimentata.

La gestione del preset di un nuovo orario è lasciata dunque al Programmatore mediante l’uso del bit W.CB.ADJ; se il sistema complessivo non prevede particolari schede SLAVES, come un terminale con display (il quale, con opportune parti di programma utente, potrebbe realizzare l’interfaccia uomo/macchina con l’orologio/calendario), l’unico modo per impostare l’orario, è quello di far uso dell’ambiente di sviluppo su Personal Computer, il quale mette a disposizione un’apposita utility di preset dell’orologio/calendario.

Le istruzioni del linguaggio

Nozioni preliminari

Nei successivi paragrafi verrà effettuata una analisi dettagliata delle istruzioni disponibili nel linguaggio.

Come abbiamo già accennato un programma è costituito dalla sequenza di un certo numero di istruzioni; ciascuna istruzione occupa una riga del file sorgente editato e genererà con il processo di compilazione una parte di codice oggetto eseguibile dalla scheda MASTER. Gran parte delle istruzioni, per essere complete, necessitano di un certo numero di operandi: descrivendo le risorse del sistema abbiamo già introdotto gli operandi e abbiamo definito un modo per identificarli.

Escludendone alcune di uso particolare o speciale, la maggior parte delle istruzioni può essere suddivisa in due principali categorie: le istruzioni booleane (quelle che operano sul singolo bit) e quelle di manipolazione dei bytes.

Le istruzioni booleane sono molto utilizzate in quanto consentono di descrivere una rete elettromeccanica equivalente che corrisponde alle esigenze di funzionamento della macchina automatica. Queste istruzioni realizzano operazioni booleane sulle risorse (sia esterne che interne) del sistema. Il metodo di calcolo del valore di nodo della rete, si avvale di un registro accumulatore, invisibile al Programmatore, che permette la momentanea memorizzazione dei valori intermedi del calcolo.

Il registro accumulatore va visto come una lista di singoli bits di tipo LIFO (Last In First Out), nella quale entrano i valori temporanei del calcolo della rete ogni qual volta si utilizza l'istruzione LD oppure LDNOT. La profondità massima della lista è di 8 bits, il che consente di tenere in memoria il risultato intermedio di 8 rami della rete, prima di eventuali serie e paralleli tra i rami stessi mediante le istruzioni ANDLD ed ORLD. Queste istruzioni realizzano rispettivamente la funzione serie e parallelo di due risultati intermedi (tra la prima e la seconda posizione ad uscire dalla lista) e, dopo aver posizionato il risultato nella stessa prima posizione della lista, fanno scorrere di un posto tutti gli eventuali altri risultati intermedi (dalla terza alla seconda, dalla quarta alla terza, ecc.).

Il bit, primo ad uscire della lista LIFO, viene utilizzato come alimentazione delle istruzioni di uscita; tale bit non viene alterato dalle operazioni di uscita, per cui è possibile alimentare, con lo stesso ramo calcolato, un numero qualunque di uscite e di dispositivi.

Le istruzioni di manipolazione di bytes lavorano invece su grandezze composte da 1, 2 e 4 bytes; esse consentono di effettuare calcoli su variabili e parametri del sistema. Mediante le istruzioni di comparazione è possibile controllare i valori assunti da tali grandezze e conseguentemente far procedere il programma. Un uso frequente di tali istruzioni è quello di presettaggio dei valori finali dei contatori e dei timers e di elaborazione di grandezze analogiche del sistema.

Le variabili a bytes possono essere di tre tipi a seconda della loro dimensione; le variabili ad 1 byte sono quelle più piccole dimensionalmente e sono indicate dal nome o della Label del byte stesso. Le variabili a 2 e 4 bytes sono invece ottenute dalla concatenazione di 2 e 4 bytes consecutivi e la loro individuazione avviene mediante il nome o la Label del primo (indirizzo minore) dei bytes che è anche il byte di peso inferiore della variabile. In Tabella 2 sono riportati i campi di variazione delle variabili nelle tre diverse dimensioni.

TIPO	CAMPO DI VARIAZIONE	
	INTERO POSITIVO	INTERO CON SEGNO
1 BYTE	0.....255	-128.....+127
2 BYTES	0.....65535	-32768.....+32767
4 BYTES	0.....4294967295	-2147483648...+2147483647

Tabella 2. Campi di variazione delle variabili a 1/2/4 bytes

Tra le istruzioni a bytes alcune meritano una particolare attenzione. Si tratta delle istruzioni RCL1, RCL2, RCL4, ADD, SUB, MUL, DIV, CMP, STO1, STO2 e STO4 per la valutazione di espressioni matematiche su variabili ad 1, 2, 4 bytes (anche in modo misto) descritte secondo la "Notazione Polacca Inversa". Queste istruzioni costituiscono un'efficiente alternativa alle classiche istruzioni matematiche dei linguaggi d'automazione. Possiamo anticipare che questa tecnica di valutazione delle espessioni è molto simile a quella appena descritta per la valutazione delle espressioni booelane sui singoli bits; anche in questo caso è presente una lista LIFO nella quale sono temporaneamente archiviate variabili (in formato 32 bits con segno) come risultati intermedi dell'espressione (fino ad un massimo di 4 livelli). Rimandiamo all'apposito paragrafo l'analisi dettagliata di questa tecnica di valutazione delle espressioni matematiche.

In generale si consideri che le istruzioni di uscita (ossia quelle che forzano con la scrittura i bits o i bytes), non possono essere utilizzate su tutti gli operandi; sono infatti esclusi tutti gli operandi di tipo "costante", gli operandi speciali (oscillazioni, alcuni flags) e le uscite già pilotate dai dispositivi interni. Sarà comunque il compilatore ad effettuare un'analisi della validità di applicazione delle istruzioni ed a segnalare al Programmatore le eventuali incompatibilità di certe istruzioni con alcuni tipi di operandi.

Le istruzioni di uso più frequente sono state dotate di un mnemonico duale abbreviato, per una più rapida stesura del programma sorgente; questo mnemonico alternativo verrà riportato alla voce "forma abbreviata" della specifica istruzione.

LD

Carica il valore ON/OFF del contatto normalmente aperto

Sintassi

LD OperandoBit ‘commento

Argomento

OperandoBit è l’indirizzo o la relativa label di qualsiasi bit della memoria ram.

Descrizione

LD carica nel registro accumulatore a bits il valore ON/OFF del contatto normalmente aperto indicato dall’argomento; la lista a bits del registro accumulatore sale automaticamente di un livello. Questa è in genere la prima dell’elenco istruzioni che descrivono un ramo della rete elettromeccanica.

Esempio

Il seguente esempio illustra il caricamento del bit 0.0.0 e la copiatura sul bit 0.8.0:

```
LD      0.0.0
OUT     0.8.0
```

Forma abbreviata

L OperandoBit ‘commento

Vedi anche

LDNOT

LDNOT

Carica il valore ON/OFF del contatto normalmente chiuso

Sintassi

LDNOT OperandoBit ‘commento

Argomento

OperandoBit è l’indirizzo o la relativa label di qualsiasi bit della memoria ram.

Descrizione

LD carica nel registro accumulatore a bits il valore ON/OFF del contatto normalmente chiuso indicato dall’argomento; la lista a bits del registro accumulatore sale automaticamente di un livello. Questa è in genere la prima dell’elenco istruzioni che descrivono un ramo della rete elettromeccanica.

Esempio

Il seguente esempio illustra il caricamento negato del bit 0.0.0 e la copiatura sul bit 0.8.0:

```
LDNOT   0.0.0
OUT     0.8.0
```

Forma abbreviata

LN OperandoBit ‘commento

Vedi anche

LD

AND

Serie con il valore ON/OFF del contatto normalmente aperto

Sintassi

AND OperandoBit ‘commento

Argomento

OperandoBit è l’indirizzo o la relativa label di qualsiasi bit della memoria ram.

Descrizione

AND esegue la serie tra l’ultimo bit inserito nel registro accumulatore a bits ed il valore ON/OFF del contatto normalmente aperto indicato dall’argomento. Il risultato della serie sostituisce il valore del bit più esterno della lista nel registro accumulatore.

Esempio

Il seguente esempio esegue la serie del bit 0.0.0 con il bit 0.0.1. Il risultato viene poi copiato sul bit bit 0.8.0:

```
LD      0.0.0
AND     0.0.1
OUT     0.8.0
```

Forma abbreviata

A OperandoBit ‘commento

Vedi anche

ANDNOT

ANDNOT

Serie con il valore ON/OFF del contatto normalmente chiuso

Sintassi

ANDNOT OperandoBit ‘commento

Argomento

OperandoBit è l’indirizzo o la relativa label di qualsiasi bit della memoria ram.

Descrizione

ANDNOT esegue la serie tra l’ultimo bit inserito nel registro accumulatore a bits ed il valore ON/OFF del contatto normalmente chiuso indicato dall’argomento. Il risultato della serie sostituisce il valore del bit più esterno della lista nel registro accumulatore.

Esempio

Il seguente esempio esegue la serie del bit 0.0.0 con il bit 0.0.1 negato. Il risultato viene poi copiato sul bit bit 0.8.0:

```
LD      0.0.0
ANDNOT  0.0.1
OUT     0.8.0
```

Forma abbreviata

AN OperandoBit ‘commento

Vedi anche

AND

OR

Parallelo con il valore ON/OFF del contatto normalmente aperto

Sintassi

OR OperandoBit ‘commento

Argomento

OperandoBit è l’indirizzo o la relativa label di qualsiasi bit della memoria ram.

Descrizione

OR esegue il parallelo tra l’ultimo bit inserito nel registro accumulatore a bits ed il valore ON/OFF del contatto normalmente aperto indicato dall’argomento. Il risultato del parallelo sostituisce il valore del bit più esterno della lista nel registro accumulatore.

Esempio

Il seguente esempio esegue il parallelo tra il bit 0.0.0 ed il bit 0.0.1. Il risultato viene poi copiato sul bit bit 0.8.0:

```
LD      0.0.0
OR      0.0.1
OUT     0.8.0
```

Forma abbreviata

O OperandoBit ‘commento

Vedi anche

ORNOT

ORNOT

Parallelo con il valore ON/OFF del contatto normalmente chiuso

Sintassi

```
ORNOT    OperandoBit          ‘commento
```

Argomento

OperandoBit è l’indirizzo o la relativa label di qualsiasi bit della memoria ram.

Descrizione

ORNOT esegue il parallelo tra l’ultimo bit inserito nel registro accumulatore a bits ed il valore ON/OFF del contatto normalmente chiuso indicato dall’argomento. Il risultato del parallelo sostituisce il valore del bit più esterno della lista nel registro accumulatore.

Esempio

Il seguente esempio esegue il parallelo tra il bit 0.0.0 ed il bit 0.0.1 negato. Il risultato viene poi copiato sul bit bit 0.8.0:

```
LD       0.0.0
ORNOT    0.0.1
OUT      0.8.0
```

Forma abbreviata

```
ON       OperandoBit          ‘commento
```

Vedi anche

OR

ANDLD

Serie tra due risultati intermedi

Sintassi

```
ANDLD                    ‘commento
```

Argomento

Nessuno.

Descrizione

ANDLD esegue la serie tra gli ultimi due bits introdotti nel registro accumulatore a bits. La lista a bits presente nel registo accumulatore scende automaticamente di un livello ed il risultato della serie è posizionato nel bit più esterno della lista nel registro accumulatore.

Esempio

Il seguente esempio esegue il parallelo tra il bit 0.0.0 ed il bit 0.0.1 ed il parallelo tra il bit 0.0.2 ed il bit 0.0.3. I risultati dei due paralleli sono quindi messi in serie tra loro ed il relativo risultato viene poi copiato sul bit bit 0.8.0:

```
LD      0.0.0       ‘primo parallelo
OR      0.0.1
LD      0.0.2       ‘secondo parallelo
OR      0.0.3
ANDLD               ‘serie dei paralleli
OUT     0.8.0
```

Forma abbreviata

```
AL                       ‘commento
```

Vedi anche

ORLD

ORLD

Parallelo tra due risultati intermedi

Sintassi

```
ORLD                    ‘commento
```

Argomento

Nessuno.

Descrizione

ORLD esegue il parallelo tra gli ultimi due bits introdotti nel registro accumulatore a bits. La lista a bits presente nel registo accumulatore scende automaticamente di un livello ed il risultato del parallelo è posizionato nel bit più esterno della lista nel registro accumulatore.

Esempio

Il seguente esempio esegue la serie tra il bit 0.0.0 ed il bit 0.0.1 e la serie tra il bit 0.0.2 ed il bit 0.0.3. I risultati delle due serie sono quindi messi in parallelo tra loro ed il relativo risultato viene poi copiato sul bit bit 0.8.0:

```
LD      0.0.0       ‘prima serie
AND     0.0.1
LD      0.0.2       ‘seconda serie
AND     0.0.3
ORLD                ‘parallelo delle serie
OUT     0.8.0
```

Forma abbreviata

```
OL                  ‘commento
```

Vedi anche

ANDLD

OUT

Alimentazione della bobina del relè

Sintassi

OUT OperandoBit ‘commento

Argomento

OperandoBit è l’indirizzo o la relativa label di qualsiasi bit della memoria ram.

Descrizione

OUT copia il valore ON/OFF dell’ultimo bit inserito nel registro accumulatore sul bit specificato dall’argomento.
La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio copia il bit 0.0.0 su entrambi i bits 0.8.0 e 0.8.1:

```
LD      0.0.0
OUT     0.8.0
OUT     0.8.1
```

Forma abbreviata

= OperandoBit ‘commento

Vedi anche

OUTNOT

OUTNOT

Alimentazione negata della bobina del relè

Sintassi

OUTNOT OperandoBit ‘commento

Argomento

OperandoBit è l’indirizzo o la relativa label di qualsiasi bit della memoria ram.

Descrizione

OUTNOT copia in modo negato il valore ON/OFF dell’ultimo bit inserito nel registro accumulatore sul bit specificato dall’argomento.
La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio copia in modo negato il bit 0.0.0 su entrambi i bits 0.8.0 e 0.8.1:

```
LD      0.0.0
OUTNOT  0.8.0
OUTNOT  0.8.1
```

Forma abbreviata

=N OperandoBit ‘commento

Vedi anche

OUT

SET

Alimentazione con autoritenuta della bobina del relè

Sintassi

SET OperandoBit ‘commento

Argomento

OperandoBit è l’indirizzo o la relativa label di qualsiasi bit della memoria ram.

Descrizione

SET forza il valore logico “1” sul bit specificato dall’argomento solo se l’ultimo bit inserito nel registro accumulatore è anch’esso ad “1”. L’istruzione SET è spesso abbinata all’istruzione RES per realizzare la funzione di alimentazione relè con autoritenuta e sgancio (blocco funzione set/reset); in questo caso l’istruzione che si presenta per seconda nel listato avrà priorità sull’altra.
La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio setta il bit 0.8.0 se il bit 0.0.0 è ON:

```
LD      0.0.0
SET     0.8.0
```

Forma abbreviata

S OperandoBit ‘commento

Vedi anche

RES, CPL

RES

Sgancio dell' autoritenuta della bobina del relè

Sintassi

RES OperandoBit 'commento

Argomento

OperandoBit è l'indirizzo o la relativa label di qualsiasi bit della memoria ram.

Descrizione

RES forza il valore logico "0" sul bit specificato dall'argomento solo se l'ultimo bit inserito nel registro accumulatore è ad "1". L'istruzione RES è spesso abbinata all' istruzione SET per realizzare la funzione di alimentazione relè con autoritenuta e sgancio (blocco funzione set/reset); in questo caso l'istruzione che si presenta per seconda nel listato avrà priorità sull'altra.
La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio resetta il bit 0.8.0 se il bit 0.0.1 è ON:

```
LD      0.0.1
RES     0.8.0
```

Forma abbreviata

R OperandoBit 'commento

Vedi anche

SET, CPL

CPL

Complementazione dell'alimentazione della bobina del relè

Sintassi

```
CPL     OperandoBit     'commento
```

Argomento

OperandoBit è l'indirizzo o la relativa label di qualsiasi bit della memoria ram.

Descrizione

CPL complementa (inversione di stato) il valore logico del bit specificato dall'argomento solo se l'ultimo bit inserito nel registro accumulatore è ad "1". L'istruzione CPL corrisponde alla funzione di relè bistabile il quanto il bit indicato cambia di stato ogni qual volta viene alimentata l'istruzione; in questo caso occorre alimentare la CPL con un generatore d'impulso di durata un ciclo di scansione, altrimenti il bit indicato cambierà alternativamente di stato ad ogni ciclo.
La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio complementa il bit 0.8.0 ogni volta che il bit 0.0.0 ha un fronte di salita:

```
LD      0.0.0       'alimentazione del generatore d'impulso
OUT     P.0.IN

LD      P.0.OUTU    'l'impulso su fronte di salita alimenta la CPL
CPL     0.8.0
```

Forma abbreviata

```
C       OperandoBit     'commento
```

Vedi anche

SET, RES

JMP

Salto alla successiva JME se il ramo calcolato è ON

Sintassi

```
JMP                    ‘commento
```

Argomento

Nessuno.

Descrizione

JMP salta alla prossima istruzione JME se l’ultimo bit inserito nel registro accumulatore è ad “1”. Tale istruzione consente di scavalcare intere parti di programma sotto certe condizioni, per esempio per dividere il programma complessivo in sottoparti da eseguire alternativamente oppure per velocizzare il ciclo di scansione con l’esclusione di parti di programma non attive nella fase corrente.

Esempio

Il seguente esempio esclude dalla scansione del listato la parte di programma indicata per tutto il tempo per il quale il bit 0.0.0 vale “1”:

```
LD      0.0.0          ‘condizione di abilitazione del salto
JMP

LD      T.500          ‘parte di programma esclusa dal salto
OUT     0.8.0

JME                    ‘posizione di arrivo del salto
```

Vedi anche

JME

JME

Punto di arrivo dell'istruzione di salto condizionato JMP

Sintassi

```
JME                    'commento
```

Argomento

Nessuno.

Descrizione

JME costituisce la label di arrivo del salto della precedente istruzione JMP. La coppia di istruzioni JMP/JME consente di scavalcare intere parti di programma sotto certe condizioni. Si tenga presente che le istruzioni JMP e JME possono essere utilizzate in coppia in numero qualsiasi all'interno di un programma, ma non si possono nidificare tali salti: un' istruzione JMP richiede successivamente la sua JME, prima di iniziare un altro blocco di salto.

Esempio

Il seguente esempio esclude dalla scansione del listato la parte di programma indicata per tutto il tempo per il quale il bit 0.0.0 vale "1":

```
LD      0.0.0          'condizione di abilitazione del salto
JMP

LD      T.500          'parte di programma esclusa dal salto
OUT     0.8.0

JME                    'posizione di arrivo del salto
```

Vedi anche

JMP

GOTO

Salto alla Label se il ramo calcolato è ON

Sintassi

```
GOTO    Label                ‘commento
```

Argomento

Label è una stringa di testo con 32 caratteri massimi e senza spazi intermedi.

Descrizione

GOTO salta alla Label indicata dall’argomento se l’ultimo bit inserito nel registro accumulatore è ad “1”. L’istruzione GOTO è molto simile all’istruzione JMP; l’unica differenza è che la JMP non richiede la specificazione di una particolare Label alla quale saltare, poichè il salto avviene sempre alla successiva istruzione JME, mentre l’istruzione GOTO richiede sempre come argomento il nome della Label alla quale saltare. La Label di arrivo del salto può essere posizionata in un punto qualsiasi del listato (anche precedente alla relativa GOTO) purchè sempre succeduta dal carattere “:” (due punti) ed unica istruzione della linea. Non esistono per la GOTO limiti di nidificazione dei salti in quanto la GOTO salta sempre esattamenente nel punto indicato.

Esempio

Il seguente esempio esclude dalla scansione del listato la parte di programma indicata con il tratteggio per tutto il tempo per il quale il bit 0.0.0 vale “1”:

```
LD      0.0.0              ‘condizione di abilitazione del salto
GOTO    Punto_dove_saltare

----------------------
----------------------

Punto_dove_saltare:        ‘Label di arrivo del salto
```

Vedi anche

JMP

GOSUB

Chiamata di una subroutine se il ramo calcolato è ON

Sintassi

```
GOSUB    Label                ‘commento
```

Argomento

Label è una stringa di testo con 32 caratteri massimi e senza spazi intermedi.

Descrizione

GOSUB esegue la subroutine indicata dalla Label dell’argomento se l’ultimo bit inserito nel registro accumulatore è ad “1”. La Label di inizio del listato della subroutine deve essere posizionata dopo l’istruzione END del programma principale, in una riga non occupata da un’istruzione e deve essere sempre terminata con il carattere “:” (due punti). Successivamente alla Label deve seguire il listato delle istruzioni della subroutine terminato da una nuova istruzione END oppure dall’istruzione RET. All’interno della subroutine si possono utilizzare tutte le istruzioni del linguaggio; è possibile inserire anche le GOTO e le GOSUB e posizionare le relative nuove etichette. Si possono avere fino a 16 livelli di nidificazione delle chiamate alle subroutine.

Esempio

Il seguente esempio esegue il listato della subroutine “Gestione_Allarme_0” per tutto il tempo per il quale il bit 0.0.0 vale “1”, mentre esegue la subroutine “Gestione_Allarme_1” per tutto il tempo per il quale il bit 0.0.1 vale “1”:

```
----------------------
listato programma principale
----------------------

LD       0.0.0              ‘condizione di abilitazione della subroutine
GOSUB    Gestione_Allarme_0

LD       0.0.1              ‘condizione di abilitazione della subroutine
GOSUB    Gestione_Allarme_1

----------------------
listato programma principale
----------------------
END
```

```
Gestione_Allarme_0:          'Label di inizio della subroutine

LD          M.0.0.0          'abilitazione di un'ulteriore subroutine nidificata
GOSUB       Verifica_Allarme
------------------------
listato subroutine
------------------------
END
```

```
Gestione_Allarme_1:          'Label di inizio della subroutine

LD          M.0.0.1          'abilitazione di un'ulteriore subroutine nidificata
GOSUB       Verifica_Allarme
------------------------
listato subroutine
------------------------
END
```

```
Verifica_Allarme:            'Label di inizio della subroutine nidificata
------------------------
listato della subroutine nidificata
------------------------
END
```

Vedi anche

END, RET

NOP

Istruzione di non operazione

Sintassi

NOP ‘commento

Argomento

Nessuno.

Descrizione

NOP non è una vera e propria istruzione in quanto questa non esegue nulla all’interno del programma. Essa corrisponde ad un’istruzione fittizia che può essere inserita, per esempio, temporaneamente nel programma come promemoria di un’istruzione cancellata. Tale istruzione introduce un ritardo fisso al ciclo di scansione pari ad 1 µs.

Esempio

Il seguente esempio mostra l’inserimento dell’istruzione NOP tra altre due istruzioni:

```
LD      0.0.0
NOP
OUT     0.8.0
```

END

Fine del listato programma

Sintassi

END ‘commento

Argomento

Nessuno.

Descrizione

END è l’ultima istruzione del listato programma. Questa consente al sistema operativo di aggiornare le risorse sia interne che esterne e di ricominciare di nuovo un’altra scansione del listato programma.
Se si utilizzano subroutines questa istruzione deve essere utilizzata anche per terminare le singole parti di listato che le descrivono; le subroutines vanno posizionate al termina del listato programma principale dopo la relativa istruzione END. Ciascuna subroutine deve essere iniziata da una Label (con il carattere “:” terminale) e conclusa con una propria istruzione END.

Esempio

Il seguente esempio illustra l’utilizzo dell’istruzione END nel caso di subroutines:

```
-------------------------
listato programma principale
-------------------------
END

Subroutine1:
-------------------------
listato subroutine 1
-------------------------
END

Subroutine2:
-------------------------
listato subroutine 2
-------------------------
END
```

Vedi anche

GOSUB, RET

RET

Fine del listato programma di una subroutine

Sintassi

RET ‘commento

Argomento

Nessuno.

Descrizione

RET è l’ultima istruzione del listato programma di una subroutine. In verità l’istruzione RET è perfettamente equivalente e sostituibile all’istruzione END, per cui potrebbe essere utilizzata anche per terminare il programma principale. Si consideri che anche il programma principale può essere assimilato ad una subroutine chiamata questa volta dal sistema operativo ad ogni ciclo di scansione. Tuttavia per correttezza formale si consiglia di utilizzare l’istruzione RET solo per le subroutine per differenziarle dal programma principale.

Esempio

Il seguente esempio illustra l’utilizzo dell’istruzione RET:

```
-------------------------
listato programma principale
-------------------------
END

Subroutine1:
-------------------------
listato subroutine 1
-------------------------
RET

Subroutine2:
-------------------------
listato subroutine 2
-------------------------
RET
```

Vedi anche

GOSUB, END

TIM

Temporizzatore con base tempi 0.1”

Sintassi

TIM C.*.IN Costante ‘commento

Argomento

C.*.IN è il bit di start del generico contatore (* = 0÷127).
Costante è una costante numerica di dimensione massima 2 bytes (K.0 ÷ K.65535).

Descrizione

Il dispositivo TIM realizza temporizzatori con funzione ritardata all’eccitazione e tempi compresi tra 0 e 6553.5 secondi.

TIM costituisce una forma semplificata e rapida per l’utilizzo dei dispositivi contatori in veste di dispositivi temporizzatori. Per utilizzare un contatore come temporizzatore occorre connettere l’ingresso di clock UP ad uno dei bits di oscillazione periodica (bits di tipo T.); inoltre occorre precaricare il valore finale del contatore con la costante di tempo voluta.
L’istruzione TIM svolge tutte queste funzioni automaticamente trasformando in questo modo un dispositivo contatore in un temporizzatore. L’istruzione TIM collega automaticamente al suo interno il valore dell’ultimo bit inserito nel registro accumulatore al bit di start del contatore, il bit di oscillazione periodica T.100 (decimi di secondo) al bit di clock UP e precarica il valore finale del contatore con la costante di tempo fornita (numero di decimi di secondo richiesti). Il bit di fine tempo del temporizzatore coincide quindi con il bit di fine conteggio del contatore.
La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: si consiglia, ove possibile, di utilizzare l’istruzione TIM anzichè gestire per esteso il contatore; in questo modo si ottiene un risparmio di memoria ed anche una maggiore velocità di elaborazione.

Esempio

Il seguente esempio compara l'utilizzo esteso del contatore per realizzare un temporizzatore con la sua gestione più compatta mediante l'istruzione TIM:

```
LD      0.0.0              'segnale di abilitazione temporizzazione
OUT     C.0.IN             'abilitazione del dispositivo contatore
MOV2    C.0.FL    K.30     'preset del valore finale di comparazione
LD      T.100              'bit di oscillazione periodica a 0.1"
OUT     C.0.CKUP           'alimentazione clock incremento conteggio

LD      C.0.OUT            'carica il valore di esito comparazione
OUT     0.8.0              'se fine tempo alimenta il bit 0.8.0
```

utilizzando l'istruzione TIM la precedente parte di programma si può così riscrivere:

```
LD      0.0.0              'segnale di abilitazione temporizzazione
TIM     C.0.IN    K.30     'abilitazione del dispositivo temporizzatore

LD      C.0.OUT            'carica l'uscita del temporizzatore
OUT     0.8.0              'se fine tempo alimenta il bit 0.8.0
```

Vedi anche

CNT

CNT

Contatore ad incremento

Sintassi

CNT C.*.IN OperandoBit Costante ‘commento

Argomento

C.*.IN è il bit di start del generico contatore (* = 0÷127).
OperandoBit è l’indirizzo o la relativa label del bit da utilizzare come clock Up.
Costante è una costante numerica di dimensione massima 2 bytes (K.0 ÷ K.65535).

Descrizione

Il dispositivo CNT realizza contatori a solo incremento valore.

CNT costituisce una forma semplificata e rapida per l’utilizzo dei dispositivi contatori quando si richiedono solo conteggi in avanti del valore corrente. L’istruzione CNT collega automaticamente al suo interno il valore dell’ultimo bit inserito nel registro accumulatore al bit di start del contatore, il bit indicato nell’argomento al clock Up e precarica il valore finale del contatore con la costante di conteggio fornita.
Nel caso sia necessario decrementare il contatore è possibile utilizzare a parte il clock Down di questo, lasciato sconnesso ed inutilizzato dall’istruzione CNT.
La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: si consiglia, ove possibile, di utilizzare l’istruzione CNT anzichè gestire per esteso il contatore; in questo modo si ottiene un risparmio di memoria ed anche una maggiore velocità di elaborazione.

Esempio

Il seguente esempio compara l'utilizzo esteso di un dispositivo contatore con la sua gestione più compatta mediante l'istruzione CNT:

```
LD      0.0.0                   'segnale di abilitazione contatore
OUT     C.0.IN                  'abilitazione del dispositivo contatore
MOV2    C.0.FL      K.30        'preset del valore finale di comparazione
LD      0.0.1                   'bit da conteggiare sul fronte
OUT     C.0.CKUP                'alimentazione clock incremento conteggio

LD      C.0.OUT                 'carica il valore di esito comparazione
OUT     0.8.0                   'se fine conteggio alimenta il bit 0.8.0
```

utilizzando l'istruzione CNT la precedente parte di programma si può così riscrivere:

```
LD      0.0.0                   'segnale di abilitazione contatore
CNT     C.0.IN      0.0.1       K.30    'abilitazione del dispositivo contatore

LD      C.0.OUT                 'carica il valore di esito comparazione
OUT     0.8.0                   'se fine conteggio alimenta il bit 0.8.0
```

Vedi anche

TIM

SFR

Shift a sinistra di un bit all'interno di un byte

Sintassi

SFR OperandoByte 'commento

Argomento

OperandoByte è l'indirizzo o la relativa label di qualsiasi byte della memoria ram.

Descrizione

SFR esegue lo shift a sinistra di una posizione dei bits del byte specificato dall'argomento solo se l'ultimo bit inserito nel registro accumulatore è ad "1". L'operazione di shift è effettuata attraverso il bit di riporto F.C in quanto lo spostamento a sinistra è accompagnato dall'entrata nel bit meno significativo del valore del flag F.C e dall'uscita del bit più significativo sullo stesso bit F.C. Questo consente la connessione in cascata di più istruzioni SFR realizzando shifts di catene anche molto lunghe (multiple intere di un byte). Il bit di flag F.C assume il significato di "ingresso dato" dello SHIF-REGISTER; l'alimentazione dell'istruzione SFR corrisponde al segnale di "clock " mentre per il "reset" occorre forzare la costante 0 sul byte. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: l'istruzione SFR esegue lo shift di una posizione ad ogni ciclo di scansione per il quale essa è alimentata. Per effettuare un solo shift in corrispondenza di una certa condizione utilizzare un dispositivo generatore d'impulso per alimentare l'istruzione.

Esempio

Il seguente esempio illustra l'utilizzo dell'istruzione SFR per realizzare un dispositivo shif-register a 16 bis:

```
Dato_Shift      =    0.0.0      'segnale di ingresso dato nello shift-register
Clock_Shift     =    0.0.1      'segnale di clock per l'avanzamento di una posizione
Reset_Shift     =    0.0.2      'segnale di reset a "0" dello shift-register

LD      Dato_Shift              'prepara il dato di ingresso nel flag di riporto
OUT     F.C

LD      Clock_Shift             'generatore d'impulso per avanzamento sui fronti di salita
OUT     P.0.IN

LD      P.0.OUTU                'alimentazione con un impulso dell'istruzione di shift
SFR     M.100
SFR     M.101

LD      Reset_Shift             'reset dello shift mediante forzatura a "0"
MOV2    M.100   K.0
```

ANDB

Prodotto logico dei singoli bits di due bytes

Sintassi

```
ANDB      OperandoByte1  OperandoByte2  OperandoByte3      ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.
OperandoByte3 è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.

Descrizione

ANDB esegue il prodotto logico (AND) tra i singoli bits di pari peso dei bytes OperandoByte2 ed OperandoByte3; il byte risultante è posizionato nell’indirizzo di destinazione OperandoByte1. L’istruzione ANDB viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio esegue il prodotto bit per bit tra il byte M.101 ed M.102 e posiziona il risultato nel byte M.100. Viene eseguito anche, come esempio, il prodotto bit per bit tra la variabile M.201 ed una costante espressa in binario; in questo modo si possono realizzare delle istruzioni di RES multiplo su più bits dello stesso byte secondo una maschera predefinita:

```
LD      0.0.0                              ‘condizione di abilitazione
ANDB    M.100   M.101   M.102              ‘istruzione su entrambi valori variabili
ANDB    M.200   M.201   K.11101011B        ‘istruzione tra variabile e costante
```

Vedi anche

ORB, XORB, CPLB

ORB

Somma logica dei singoli bits di due bytes

Sintassi

```
ORB     OperandoByte1  OperandoByte2  OperandoByte3     ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.
OperandoByte3 è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.

Descrizione

ORB esegue la somma logica (OR) tra i singoli bits di pari peso dei bytes OperandoByte2 ed OperandoByte3; il byte risultante è posizionato nell’indirizzo di destinazione OperandoByte1. L’istruzione ORB viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio esegue la somma bit per bit tra il byte M.101 ed M.102 e posiziona il risultato nel byte M.100. Viene eseguita anche, come esempio, la somma bit per bit tra la variabile M.201 ed una costante espressa in binario; in questo modo si possono realizzare delle istruzioni di SET multiplo su più bits dello stesso byte secondo una maschera predefinita:

```
LD      0.0.0                          ‘condizione di abilitazione
ORB     M.100   M.101   M.102          ‘istruzione su entrambi valori variabili
ORB     M.200   M.201   K.00010100B    ‘istruzione tra variabile e costante
```

Vedi anche

ANDB, XORB, CPLB

XORB

Somma logica esclusiva dei singoli bits di due bytes

Sintassi

```
XORB    OperandoByte1  OperandoByte2  OperandoByte3      ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.
OperandoByte3 è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.

Descrizione

XORB esegue la somma logica esclusiva (XOR) tra i singoli bits di pari peso dei bytes OperandoByte2 ed OperandoByte3; il byte risultante è posizionato nell’indirizzo di destinazione OperandoByte1.
L’istruzione XORB viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio esegue la somma esclusiva bit per bit tra il byte M.101 ed M.102 e posiziona il risultato nel byte M.100. Viene eseguita anche, come esempio, la somma esclusiva bit per bit tra la variabile M.201 ed una costante espressa in binario; in questo modo si possono realizzare delle istruzioni di CPL multiplo su più bits dello stesso byte secondo una maschera predefinita:

```
LD      0.0.0                              ‘condizione di abilitazione
XORB    M.100   M.101   M.102              ‘istruzione su entrambi valori variabili
XORB    M.200   M.201   K.00010100B        ‘istruzione tra variabile e costante
```

Vedi anche

ANDB, ORB, CPLB

CPLB

Complementazione di tutti i bits di un byte

Sintassi

```
CPLB      OperandoByte          ‘commento
```

Argomento

OperandoByte è l’indirizzo o la relativa label di qualsiasi byte della memoria ram.

Descrizione

CPLB esegue la complementazione di tutti i singoli bits del byte indicato dall’argomento OperandoByte.
L’istruzione CPLB viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: l’istruzione CPLB esegue una complementazione del byte ad ogni ciclo di scansione per il quale essa è alimentata. Per effettuare una sola complementazione in corrispondenza di una certa condizione utilizzare un dispositivo generatore d’impulso per alimentare l’istruzione.

Esempio

Il seguente esempio esegue la complementazione di tutti i bits del byte M.100 ogni qual volta si verifica un fronte di salita del bit 0.0.0:

```
LD        0.0.0          ‘segnale il cui fronte esegue la complementazione
OUT       P.0.IN         ‘alimentazione del generatore d’impulso

LD        P.0.OUTU       ‘impulso su fronte di salita per un ciclo di scansione
CPLB      M.100          ‘alimentazione dell’istruzione di complemetazione
```

Vedi anche

ANDB, ORB, XORB

MOV1

Muove una variabile o una costante ad 1 byte su una variabile ad 1 byte

Sintassi

```
MOV1    OperandoByte1   OperandoByte2        ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.

Descrizione

MOV1 copia il valore della variabile o della costante ad 1 byte indicata da OperandoByte2 sulla variabile ad 1 byte indicata OperandoByte1.
L’istruzione MOV1 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio muove nel byte M.100 la costante 123 e successivamente copia nel byte M.200 il valore del byte M.300:

```
LD      0.0.0                   ‘condizione di abilitazione
MOV1    M.100   K.123           ‘forza su M.100 la costante 123
MOV1    M.200   M.300           ‘copia in M.200 il valore di M.300
```

Vedi anche

MOV2, MOV4, MOVBLK

MOV2

Muove una variabile o una costante a 2 bytes su una variabile a 2 bytes

Sintassi

```
MOV2    OperandoByte1  OperandoByte2      ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 2 bytes.

Descrizione

MOV2 copia il valore della variabile o della costante a 2 bytes indicata da OperandoByte2 sulla variabile a 2 bytes indicata OperandoByte1.
L’istruzione MOV2 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio muove nella variabile a 2 bytes M.100 la costante 12345 e successivamente copia nella variabile a 2 bytes M.200 il valore della variabile a 2 bytes M.300:

```
LD      0.0.0                   ‘condizione di abilitazione
MOV2    M.100    K.12345        ‘M.101÷100 <--- 12345
MOV2    M.200    M.300          ‘M.201÷200 <--- M.301÷300
```

Vedi anche

MOV1, MOV4, MOVBLK

MOV4

Muove una variabile o una costante a 4 bytes su una variabile a 4 bytes

Sintassi

```
MOV4    OperandoByte1  OperandoByte2      ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 4 bytes.

Descrizione

MOV4 copia il valore della variabile o della costante a 4 bytes indicata da OperandoByte2 sulla variabile a 4 bytes indicata OperandoByte1.
L’istruzione MOV4 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio muove nella variabile a 4 bytes M.100 la costante 123456789 e successivamente copia nella variabile a 4 bytes M.200 il valore della variabile a 4 bytes M.300:

```
LD      0.0.0                   ‘condizione di abilitazione
MOV4    M.100   K.123456789     ‘M.103÷100 <--- 123456789
MOV4    M.200   M.300           ‘M.203÷200 <--- M.303÷300
```

Vedi anche

MOV1, MOV2, MOVBLK

CMP1

Compara il valore di due variabili o costanti ad 1 byte

Sintassi

```
CMP1      OperandoByte1   OperandoByte2      ‘commento
```

Argomento

OperandoByte1 è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.

Descrizione

CMP1 compara il valore della variabile o della costante ad 1 byte indicata da OperandoByte1 con il valore della variabile o della costante ad 1 byte indicata OperandoByte2. Il risultato del confronto è disponibile, immediatamente dopo tale istruzione, nei bits speciali di Flag (F.<, F.=, F.>), che possono essere verificati da qualsiasi istruzione booleana.
L’istruzione CMP1 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: il valore forzato nei flags di risultato rimarrà inalterato lungo il corso del listato in scansione fino all’eventuale successiva prossima istruzione di comparazione.

Esempio

Il seguente esempio compara il valore nel byte M.100 con il valore nel byte M.200:

```
LD        0.0.0                    ‘condizione di abilitazione
CMP1      M.100     M.200          ‘confronta M.100 con M.200
```

RISULTATO	F.<	F.=	F.>
M.100 < M.200	ON	OFF	OFF
M.100 = M.200	OFF	ON	OFF
M.100 > M.200	OFF	OFF	ON

Vedi anche

CMP2, CMP4

CMP2

Compara il valore di due variabili o costanti a 2 bytes

Sintassi

CMP2 OperandoByte1 OperandoByte2 ‘commento

Argomento

OperandoByte1 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 2 bytes.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 2 bytes.

Descrizione

CMP2 compara il valore della variabile o della costante a 2 bytes indicata da OperandoByte1 con il valore della variabile o della costante a 2 bytes indicata OperandoByte2. Il risultato del confronto è disponibile, immediatamente dopo tale istruzione, nei bits speciali di Flag (F.<, F.=, F.>), che possono essere verificati da qualsiasi istruzione booleana.
L’istruzione CMP2 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: il valore forzato nei flags di risultato rimarrà inalterato lungo il corso del listato in scansione fino all’ eventuale successiva prossima istruzione di comparazione.

Esempio

Il seguente esempio compara il valore nella variabile M.100 con il valore nella variabile M.200:

```
LD      0.0.0                   ‘condizione di abilitazione
CMP2    M.100     M.200         ‘confronta M.101÷100 con M.201÷200
```

RISULTATO	F.<	F.=	F.>
M.101,M.100 < M.201,M.200	ON	OFF	OFF
M.101,M.100 = M.201,M.200	OFF	ON	OFF
M.101,M.100 > M.201,M.200	OFF	OFF	ON

Vedi anche

CMP1, CMP4

CMP4

Compara il valore di due variabili o costanti a 4 bytes

Sintassi

CMP4 OperandoByte1 OperandoByte2 ‘commento

Argomento

OperandoByte1 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 4 bytes.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 4 bytes.

Descrizione

CMP4 compara il valore della variabile o della costante a 4 bytes indicata da OperandoByte1 con il valore della variabile o della costante a 4 bytes indicata OperandoByte2. Il risultato del confronto è disponibile, immediatamente dopo tale istruzione, nei bits speciali di Flag (F.<, F.=, F.>), che possono essere verificati da qualsiasi istruzione booleana.
L’istruzione CMP4 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: il valore forzato nei flags di risultato rimarrà inalterato lungo il corso del listato in scansione fino all’ eventuale successiva prossima istruzione di comparazione.

Esempio

Il seguente esempio compara il valore nella variabile M.100 con il valore nella variabile M.200:

```
LD      0.0.0                   ‘condizione di abilitazione
CMP4    M.100   M.200           ‘confronta M.103÷100 con M.203÷200
```

RISULTATO	F.<	F.=	F.>
M.103,,,M.100 < M.203,,,M.200	ON	OFF	OFF
M.103,,,M.100 = M.203,,,M.200	OFF	ON	OFF
M.103,,,M.100 > M.203,,,M.200	OFF	OFF	ON

Vedi anche

CMP1, CMP2

ADD1

Somma binaria di due variabili o costanti ad 1 byte su una variabile ad 1 byte

Sintassi

```
ADD1      OperandoByte1  OperandoByte2  OperandoByte3      ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.
OperandoByte3 è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.

Descrizione

ADD1 somma il valore della variabile o della costante ad 1 byte indicata da OperandoByte2 con il valore della variabile o della costante ad 1 byte indicata OperandoByte3 e posiziona il risultato sulla variabile ad 1 byte indicata da OperandoByte1. Il valore del bit di riporto oltre il byte viene posizionato nel flag di carry F.C.
L’istruzione ADD1 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni di valutazione matematica possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio somma il valore del byte M.200 con la costante 123 e posiziona il risultato nel byte M.100:

```
LD      0.0.0                          ‘condizione di abilitazione
ADD1    M.100   M.200   K.123          ‘M.100 <--- M.200 + 123
```

Vedi anche

ADD2, ADD4

ADD2

Somma binaria di due variabili o costanti a 2 bytes su una variabile a 2 bytes

Sintassi

```
ADD2    OperandoByte1  OperandoByte2  OperandoByte3    ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 2 bytes.
OperandoByte3 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 2 bytes.

Descrizione

ADD2 somma il valore della variabile o della costante a 2 bytes indicata da OperandoByte2 con il valore della variabile o della costante a 2 bytes indicata OperandoByte3 e posiziona il risultato sulla variabile a 2 bytes indicata da OperandoByte1. Il valore del bit di riporto oltre i due bytes viene posizionato nel flag di carry F.C.
L’istruzione ADD2 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni di valutazione matematica possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio somma il valore della variabile a 2 bytes M.200 con la costante 12345 e posiziona il risultato nella variabile a 2 bytes M.100:

```
LD      0.0.0                          ‘condizione di abilitazione
ADD2    M.100   M.200   K.12345       ‘M.101÷100 <--- M.201÷200 + 12345
```

Vedi anche

ADD1, ADD4

ADD4

Somma binaria di due variabili o costanti a 4 bytes su una variabile a 4 bytes

Sintassi

```
ADD4    OperandoByte1  OperandoByte2  OperandoByte3    ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 4 bytes.
OperandoByte3 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 4 bytes.

Descrizione

ADD4 somma il valore della variabile o della costante a 4 bytes indicata da OperandoByte2 con il valore della variabile o della costante a 4 bytes indicata OperandoByte3 e posiziona il risultato sulla variabile a 4 bytes indicata da OperandoByte1. Il valore del bit di riporto oltre i quattro bytes viene posizionato nel flag di carry F.C.
L’istruzione ADD4 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni di valutazione matematica possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio somma il valore della variabile a 4 bytes M.200 con la costante 123456789 e posiziona il risultato nella variabile a 4 bytes M.100:

```
LD      0.0.0                          ‘condizione di abilitazione
ADD4    M.100   M.200   K.123456789    ‘M.103÷100 <--- M203÷200 + 123456789
```

Vedi anche

ADD1, ADD2

SUB1

Sottrazione binaria tra due variabili o costanti ad 1 byte su una variabile ad 1 byte

Sintassi

```
SUB1      OperandoByte1  OperandoByte2  OperandoByte3      ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.
OperandoByte3 è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.

Descrizione

SUB1 sottrae al valore della variabile o della costante ad 1 byte indicata da OperandoByte2 il valore della variabile o della costante ad 1 byte indicata OperandoByte3 e posiziona il risultato sulla variabile ad 1 byte indicata da OperandoByte1. Il valore del bit di riporto (borrow) oltre il byte viene posizionato nel flag di carry F.C.
L’istruzione SUB1 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni di valutazione matematica possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio sottrae al valore del byte M.200 la costante 123 e posiziona il risultato nel byte M.100:

```
LD      0.0.0                              ‘condizione di abilitazione
SUB1    M.100   M.200   K.123              ‘M.100 <--- M.200 - 123
```

Vedi anche

SUB2, SUB4

SUB2

Sottrazione binaria di due variabili o costanti a 2 bytes su una variabile a 2 bytes

Sintassi

```
SUB2    OperandoByte1  OperandoByte2  OperandoByte3    ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 2 bytes.
OperandoByte3 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 2 bytes.

Descrizione

SUB2 sottrae al valore della variabile o della costante a 2 bytes indicata da OperandoByte2 il valore della variabile o della costante a 2 bytes indicata OperandoByte3 e posiziona il risultato sulla variabile a 2 bytes indicata da OperandoByte1. Il valore del bit di riporto (borrow) oltre i due bytes viene posizionato nel flag di carry F.C.
L’istruzione SUB2 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni di valutazione matematica possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio sottrae al valore della variabile a 2 bytes M.200 la costante 12345 e posiziona il risultato nella variabile a 2 bytes M.100:

```
LD      0.0.0                          ‘condizione di abilitazione
SUB2    M.100   M.200   K.12345        ‘M.101÷100 <--- M.201÷200 - 12345
```

Vedi anche

SUB1, SUB4

SUB4

Sottrazione binaria di due variabili o costanti a 4 bytes su una variabile a 4 bytes

Sintassi

```
SUB4    OperandoByte1  OperandoByte2  OperandoByte3    ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 4 bytes.
OperandoByte3 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 4 bytes.

Descrizione

SUB4 sottrae al valore della variabile o della costante a 4 bytes indicata da OperandoByte2 il valore della variabile o della costante a 4 bytes indicata OperandoByte3 e posiziona il risultato sulla variabile a 4 bytes indicata da OperandoByte1. Il valore del bit di riporto (borrow) oltre i quattro bytes viene posizionato nel flag di carry F.C.
L’istruzione SUB4 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni di valutazione matematica possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio sottrae al valore della variabile a 4 bytes M.200 la costante 123456789 e posiziona il risultato nella variabile a 4 bytes M.100:

```
LD      0.0.0                          ‘condizione di abilitazione
SUB4    M.100   M.200   K.123456789    ‘M.103÷100 <--- M203÷200 - 123456789
```

Vedi anche

SUB1, SUB2

MUL1

Moltiplicazione di due variabili o costanti ad 1 byte su una variabile a 2 bytes

Sintassi

MUL1 OperandoByte1 OperandoByte2 OperandoByte3 ‘commento

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.
OperandoByte3 è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.

Descrizione

MUL1 moltiplica il valore della variabile o della costante ad 1 byte indicata da OperandoByte2 con il valore della variabile o della costante ad 1 byte indicata OperandoByte3 e posiziona il risultato sulla variabile a 2 bytes indicata da OperandoByte1.
La moltiplicazione effettuata dalla MUL1 è di tipo binario senza segno; il risultato ha sempre dimensione doppia di quella delle variabili moltiplicate, poichè un “8 bits” per un “8 bits” può richiedere fino a 16 bits per il risultato. Tuttavia l’eventuale necessità del byte aggiuntivo per la rappresentazione del risultato viene segnalata dal bit di flag F.E (owerflow oltre 1 byte).
L’istruzione MUL1 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni di valutazione matematica possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio moltiplica il valore del byte M.200 con la costante 123 e posiziona il risultato nella variabile a 2 bytes M.100:

```
LD      0.0.0                       ‘condizione di abilitazione
MUL1    M.100   M.200   K.123       ‘M.101÷100 <--- M.200 * 123
```

Vedi anche

MUL2, MUL4

MUL2

Moltiplicazione di due variabili o costanti a 2 bytes su una variabile a 4 bytes

Sintassi

```
MUL2    OperandoByte1  OperandoByte2  OperandoByte3    ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 2 bytes.
OperandoByte3 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 2 bytes.

Descrizione

MUL2 moltiplica il valore della variabile o della costante a 2 bytes indicata da OperandoByte2 con il valore della variabile o della costante a 2 bytes indicata OperandoByte3 e posiziona il risultato sulla variabile a 4 bytes indicata da OperandoByte1.
La moltiplicazione effettuata dalla MUL2 è di tipo binario senza segno; il risultato ha sempre dimensione doppia di quella delle variabili moltiplicate, poichè un “16 bits” per un “16 bits” può richiedere fino a 32 bits per il risultato. Tuttavia l’eventuale necessità dei 2 bytes aggiuntivi per la rappresentazione del risultato viene segnalata dal bit di flag F.E (owerflow oltre 2 byte).
L’istruzione MUL2 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni di valutazione matematica possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio moltiplica il valore della variabile a 2 bytes M.200 con la costante 12345 e posiziona il risultato nella variabile a 4 bytes M.100:

```
LD      0.0.0                          ‘condizione di abilitazione
MUL2    M.100   M.200   K.12345        ‘M.103÷100 <--- M.201÷200 * 12345
```

Vedi anche

MUL1, MUL4

MUL4

Moltiplicazione di due variabili o costanti a 4 bytes su una variabile a 8 bytes

Sintassi

```
MUL4    OperandoByte1  OperandoByte2  OperandoByte3      ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 4 bytes.
OperandoByte3 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 4 bytes.

Descrizione

MUL4 moltiplica il valore della variabile o della costante a 4 bytes indicata da OperandoByte2 con il valore della variabile o della costante a 4 bytes indicata OperandoByte3 e posiziona il risultato sulla variabile a 8 bytes indicata da OperandoByte1.
La moltiplicazione effettuata dalla MUL4 è di tipo binario senza segno; il risultato ha sempre dimensione doppia di quella delle variabili moltiplicate, poichè un “32 bits” per un “32 bits” può richiedere fino a 64 bits per il risultato. Tuttavia l’eventuale necessità dei 4 bytes aggiuntivi per la rappresentazione del risultato viene segnalata dal bit di flag F.E (owerflow oltre 4 byte).
L’istruzione MUL4 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni di valutazione matematica possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio moltiplica il valore della variabile a 4 bytes M.200 con la costante 123456789 e posiziona il risultato nella variabile a 8 bytes M.100:

```
LD      0.0.0                          ‘condizione di abilitazione
MUL4    M.100   M.200   K.123456789    ‘M.107÷100 <--- M.203÷200 * 123456789
```

Vedi anche

MUL1, MUL2

DIV1

Divisione di due variabili o costanti ad 1 byte su una variabile a 2 bytes

Sintassi

```
DIV1    OperandoByte1  OperandoByte2  OperandoByte3    'commento
```

Argomento

OperandoByte1 (destinazione) è l'indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l'indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.
OperandoByte3 è l'indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.

Descrizione

DIV1 divide il valore della variabile o della costante ad 1 byte indicata da OperandoByte2 con il valore della variabile o della costante ad 1 byte indicata OperandoByte3 e posiziona il quoziente sulla variabile ad 1 byte indicata da OperandoByte1 ed il resto sulla variabile a 1 byte di indirizzo successivo.
La divisione effettuata dalla DIV1 è di tipo binario senza segno; il risultato ha sempre dimensione doppia di quella delle variabili divise, poichè un "8 bits" diviso un "8 bits" può richiedere fino ad 8 bits per il quoziente ed 8 bits per il resto.
Il riscontro del divisore nullo forza ad "1" il flag di errore F.E ed annulla l'esecuzione.
L'istruzione DIV1 viene eseguita solo se l'ultimo bit inserito nel registro accumulatore è al valore logico "1". La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni di valutazione matematica possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio divide il valore del byte M.200 con la costante 123 e posiziona il quoziente nel byte M.100 ed il resto nel byte M.101:

```
LD      0.0.0                          'condizione di abilitazione
DIV1    M.100   M.200   K.123         'M.100 <--- Q(M.200 / 123)
                                       'M.101 <--- R(M.200 / 123)
```

Vedi anche

DIV2, DIV4

DIV2

Divisione di due variabili o costanti a 2 bytes su una variabile a 4 bytes

Sintassi

DIV2 OperandoByte1 OperandoByte2 OperandoByte3 ‘commento

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 2 bytes.
OperandoByte3 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 2 bytes.

Descrizione

DIV2 divide il valore della variabile o della costante a 2 bytes indicata da OperandoByte2 con il valore della variabile o della costante a 2 bytes indicata OperandoByte3 e posiziona il quoziente sulla variabile a 2 bytes indicata da OperandoByte1 ed il resto sulla variabile a 2 bytes di indirizzo successivo.
La divisione effettuata dalla DIV2 è di tipo binario senza segno; il risultato ha sempre dimensione doppia di quella delle variabili divise, poichè un “16 bits” diviso un “16 bits” può richiedere fino a 16 bits per il quoziente e 16 bits per il resto.
Il riscontro del divisore nullo forza ad “1” il flag di errore F.E ed annulla l’esecuzione.
L’istruzione DIV2 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni di valutazione matematica possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio divide il valore della variabile a 2 bytes M.200 con la costante 12345 e posiziona il quoziente nella variabile a 2 bytes M.100 ed il resto nella variabile a 2 bytes M.102:

```
LD      0.0.0                       ‘condizione di abilitazione
DIV2    M.100   M.200   K.12345     ‘M.101÷100 <--- Q(M.201÷200 / 12345)
                                    ‘M.103÷102 <--- R(M.201÷200 / 12345)
```

Vedi anche

DIV1, DIV4

DIV4

Divisione di due variabili o costanti a 4 bytes su una variabile a 8 bytes

Sintassi

```
DIV4    OperandoByte1  OperandoByte2  OperandoByte3    ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 4 bytes.
OperandoByte3 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 4 bytes.

Descrizione

DIV4 divide il valore della variabile o della costante a 4 bytes indicata da OperandoByte2 con il valore della variabile o della costante a 4 bytes indicata OperandoByte3 e posiziona il quoziente sulla variabile a 4 bytes indicata da OperandoByte1 ed il resto sulla variabile a 4 bytes di indirizzo successivo.
La divisione effettuata dalla DIV4 è di tipo binario senza segno; il risultato ha sempre dimensione doppia di quella delle variabili divise, poichè un “32 bits” diviso un “32 bits” può richiedere fino a 32 bits per il quoziente e 32 bits per il resto.
Il riscontro del divisore nullo forza ad “1” il flag di errore F.E ed annulla l’esecuzione.
L’istruzione DIV4 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni di valutazione matematica possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio divide il valore della variabile a 4 bytes M.200 con la costante 123456789 e posiziona il quoziente nella variabile a 4 bytes M.100 ed il resto nella variabile a 4 bytes M.104:

```
LD      0.0.0                                ‘condizione di abilitazione
DIV4    M.100   M.200   K.123456789         ‘M.103÷100 <--- Q(M.203÷200 / 123456789)
                                             ‘M.107÷104 <--- R(M.203÷200 / 123456789)
```

Vedi anche

DIV1, DIV2

INC1

Incrementa di uno la variabile ad 1 byte

Sintassi

```
INC1        OperandoByte            ‘commento
```

Argomento

OperandoByte è l’indirizzo o la relativa label di qualsiasi byte della memoria ram.

Descrizione

INC1 incrementa di una unità il valore presente nella variabile ad 1 byte indicata dall’argomento OperandoByte.
Il valore del bit di riporto oltre il byte viene posizionato nel flag di carry F.C.
L’istruzione INC1 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: l’istruzione INC1 esegue un incremento della variabile ad ogni ciclo di scansione per il quale essa è alimentata. Per effettuare un solo incremento in corrispondenza di una certa condizione utilizzare un dispositivo generatore d’impulso per alimentare l’istruzione.

Esempio

Il seguente esempio incrementa di una unità il valore della variabile ad 1 byte M.100 ogni qual volta si verifica un fronte di salita del bit 0.0.0:

```
LD          0.0.0           ‘segnale il cui fronte esegue l’incremento
OUT         P.0.IN          ‘alimentazione del generatore d’impulso

LD          P.0.OUTU        ‘impulso su fronte di salita per un ciclo di scansione
INC1        M.100           ‘M.100 <--- M.100 + 1
```

Vedi anche

INC2, INC4

INC2

Incrementa di uno la variabile a 2 bytes

Sintassi

```
INC2        OperandoByte            ‘commento
```

Argomento

OperandoByte è l’indirizzo o la relativa label di qualsiasi byte della memoria ram.

Descrizione

INC2 incrementa di una unità il valore presente nella variabile a 2 bytes indicata dall’argomento OperandoByte.
Il valore del bit di riporto oltre i due bytes viene posizionato nel flag di carry F.C.
L’istruzione INC2 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: l’istruzione INC2 esegue un incremento della variabile ad ogni ciclo di scansione per il quale essa è alimentata. Per effettuare un solo incremento in corrispondenza di una certa condizione utilizzare un dispositivo generatore d’impulso per alimentare l’istruzione.

Esempio

Il seguente esempio incrementa di una unità il valore della variabile a 2 bytes M.100 ogni qual volta si verifica un fronte di salita del bit 0.0.0:

```
LD          0.0.0           ‘segnale il cui fronte esegue l’incremento
OUT         P.0.IN          ‘alimentazione del generatore d’impulso

LD          P.0.OUTU        ‘impulso su fronte di salita per un ciclo di scansione
INC2        M.100           ‘M.101÷100 <--- M.101÷100 + 1
```

Vedi anche

INC1, INC4

INC4

Incrementa di uno la variabile a 4 bytes

Sintassi

```
INC4        OperandoByte            ‘commento
```

Argomento

OperandoByte è l’indirizzo o la relativa label di qualsiasi byte della memoria ram.

Descrizione

INC4 incrementa di una unità il valore presente nella variabile a 4 bytes indicata dall’argomento OperandoByte.
Il valore del bit di riporto oltre i quattro bytes viene posizionato nel flag di carry F.C.
L’istruzione INC4 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: l’istruzione INC4 esegue un incremento della variabile ad ogni ciclo di scansione per il quale essa è alimentata. Per effettuare un solo incremento in corrispondenza di una certa condizione utilizzare un dispositivo generatore d’impulso per alimentare l’istruzione.

Esempio

Il seguente esempio incrementa di una unità il valore della variabile a 4 bytes M.100 ogni qual volta si verifica un fronte di salita del bit 0.0.0:

```
LD          0.0.0               ‘segnale il cui fronte esegue l’incremento
OUT         P.0.IN              ‘alimentazione del generatore d’impulso

LD          P.0.OUTU            ‘impulso su fronte di salita per un ciclo di scansione
INC4        M.100               ‘M.103÷100 <--- M.103÷100 + 1
```

Vedi anche

INC1, INC2

DEC1

Decrementa di uno la variabile ad 1 byte

Sintassi

```
DEC1        OperandoByte        ‘commento
```

Argomento

OperandoByte è l’indirizzo o la relativa label di qualsiasi byte della memoria ram.

Descrizione

DEC1 decrementa di una unità il valore presente nella variabile ad 1 byte indicata dall’argomento OperandoByte.
Il valore del bit di riporto (borrow) oltre il byte viene posizionato nel flag di carry F.C.
L’istruzione DEC1 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: l’istruzione DEC1 esegue un decremento della variabile ad ogni ciclo di scansione per il quale essa è alimentata. Per effettuare un solo decremento in corrispondenza di una certa condizione utilizzare un dispositivo generatore d’impulso per alimentare l’istruzione.

Esempio

Il seguente esempio decrementa di una unità il valore della variabile ad 1 byte M.100 ogni qual volta si verifica un fronte di salita del bit 0.0.0:

```
LD      0.0.0       ‘segnale il cui fronte esegue il decremento
OUT     P.0.IN      ‘alimentazione del generatore d’impulso

LD      P.0.OUTU    ‘impulso su fronte di salita per un ciclo di scansione
DEC1    M.100       ‘M.100 <--- M.100 - 1
```

Vedi anche

DEC2, DEC4

DEC2

Decrementa di uno la variabile a 2 bytes

Sintassi

```
DEC2        OperandoByte        ‘commento
```

Argomento

OperandoByte è l’indirizzo o la relativa label di qualsiasi byte della memoria ram.

Descrizione

DEC2 decrementa di una unità il valore presente nella variabile a 2 bytes indicata dall’argomento OperandoByte.
Il valore del bit di riporto (borrow) oltre i due bytes viene posizionato nel flag di carry F.C.
L’istruzione DEC2 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: l’istruzione DEC2 esegue un decremento della variabile ad ogni ciclo di scansione per il quale essa è alimentata. Per effettuare un solo decremento in corrispondenza di una certa condizione utilizzare un dispositivo generatore d’impulso per alimentare l’istruzione.

Esempio

Il seguente esempio decrementa di una unità il valore della variabile a 2 bytes M.100 ogni qual volta si verifica un fronte di salita del bit 0.0.0:

```
LD      0.0.0       ‘segnale il cui fronte esegue il decremento
OUT     P.0.IN      ‘alimentazione del generatore d’impulso

LD      P.0.OUTU    ‘impulso su fronte di salita per un ciclo di scansione
DEC2    M.100       ‘M.101÷100 <--- M.101÷100 - 1
```

Vedi anche

DEC1, DEC4

DEC4

Decrementa di uno la variabile a 4 bytes

Sintassi

DEC4 OperandoByte ‘commento

Argomento

OperandoByte è l’indirizzo o la relativa label di qualsiasi byte della memoria ram.

Descrizione

DEC4 decrementa di una unità il valore presente nella variabile a 4 bytes indicata dall’argomento OperandoByte.
Il valore del bit di riporto (borrow) oltre i quattro bytes viene posizionato nel flag di carry F.C.
L’istruzione DEC4 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: l’istruzione DEC4 esegue un decremento della variabile ad ogni ciclo di scansione per il quale essa è alimentata. Per effettuare un solo decremento in corrispondenza di una certa condizione utilizzare un dispositivo generatore d’impulso per alimentare l’istruzione.

Esempio

Il seguente esempio decrementa di una unità il valore della variabile a 4 bytes M.100 ogni qual volta si verifica un fronte di salita del bit 0.0.0:

```
LD      0.0.0       ‘segnale il cui fronte esegue il decremento
OUT     P.0.IN      ‘alimentazione del generatore d’impulso

LD      P.0.OUTU    ‘impulso su fronte di salita per un ciclo di scansione
DEC4    M.100       ‘M.103÷100 <--- M.103÷100 - 1
```

Vedi anche

DEC1, DEC2

ABS1

Valore assoluto di una variabile o costante ad 1 byte su una variabile ad 1 byte

Sintassi

```
ABS1      OperandoByte1  OperandoByte2      ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.

Descrizione

ABS1 determina il valore assoluto della variabile o della costante ad 1 byte indicata da OperandoByte2 e lo scrive sulla variabile ad 1 byte indicata OperandoByte1.
Se la variabile operata è negativa tale istruzione equivale ad un’ inversione di segno del valore; in questo caso il bit di flag F.C viene complementato per segnalare l’avvenuto cambio di segno.
Il flag F.C può essere utilizzato per calcolare il segno del risultato di un calcolo con moltiplicazioni o divisioni di tipo senza segno. Inizializzando a “0” logico il flag F.C ed applicando preventivamente il valore assoluto agli operandi, si ottiene in F.C il valore “1” in presenza di un numero dispari di operandi negativi: tale flag potrà quindi abilitare l’inversione di segno del risultato.
L’istruzione ABS1 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio posiziona nella variabile ad 1 byte M.100 il valore assoluto della variabile ad 1 byte M.200, complementando il flag F.C se il valore in M.200 è negativo:

```
LD        0.0.0                    ‘condizione di abilitazione
ABS1      M.100     M.200          ‘M.100 <--- |M.200|
```

Vedi anche

ABS2, ABS4, NEG1

ABS2

Valore assoluto di una variabile o costante a 2 bytes su una variabile a 2 bytes

Sintassi

```
ABS2      OperandoByte1   OperandoByte2      ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 2 bytes.

Descrizione

ABS2 determina il valore assoluto della variabile o della costante a 2 bytes indicata da OperandoByte2 e lo scrive sulla variabile a 2 bytes indicata OperandoByte1.
Se la variabile operata è negativa tale istruzione equivale ad un’ inversione di segno del valore; in questo caso il bit di flag F.C viene complementato per segnalare l’avvenuto cambio di segno.
Il flag F.C può essere utilizzato per calcolare il segno del risultato di un calcolo con moltiplicazioni o divisioni di tipo senza segno. Inizializzando a “0” logico il flag F.C ed applicando preventivamente il valore assoluto agli operandi, si ottiene in F.C il valore “1” in presenza di un numero dispari di operandi negativi: tale flag potrà quindi abilitare l’inversione di segno del risultato.
L’istruzione ABS2 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio posiziona nella variabile a 2 bytes M.100 il valore assoluto della variabile a 2 bytes M.200, complementando il flag F.C se il valore in M.200 è negativo:

```
LD        0.0.0                        ‘condizione di abilitazione
ABS2      M.100     M.200              ‘M.101÷100 <--- |M.201÷200|
```

Vedi anche

ABS1, ABS4, NEG2

ABS4

Valore assoluto di una variabile o costante a 4 bytes su una variabile a 4 bytes

Sintassi

```
ABS4      OperandoByte1   OperandoByte2      ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 4 bytes.

Descrizione

ABS4 determina il valore assoluto della variabile o della costante a 4 bytes indicata da OperandoByte2 e lo scrive sulla variabile a 4 bytes indicata OperandoByte1.
Se la variabile operata è negativa tale istruzione equivale ad un’ inversione di segno del valore; in questo caso il bit di flag F.C viene complementato per segnalare l’avvenuto cambio di segno.
Il flag F.C può essere utilizzato per calcolare il segno del risultato di un calcolo con moltiplicazioni o divisioni di tipo senza segno. Inizializzando a “0” logico il flag F.C ed applicando preventivamente il valore assoluto agli operandi, si ottiene in F.C il valore “1” in presenza di un numero dispari di operandi negativi: tale flag potrà quindi abilitare l’inversione di segno del risultato.
L’istruzione ABS4 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio posiziona nella variabile a 4 bytes M.100 il valore assoluto della variabile a 4 bytes M.200, complementando il flag F.C se il valore in M.200 è negativo:

```
LD        0.0.0                    ‘condizione di abilitazione
ABS4      M.100     M.200          ‘M.103÷100 <--- |M.203÷200|
```

Vedi anche

ABS1, ABS2, NEG4

NEG1

Inversione di segno di una variabile ad 1 byte

Sintassi

```
NEG1        OperandoByte        ‘commento
```

Argomento

OperandoByte è l’indirizzo o la relativa label di qualsiasi byte della memoria ram.

Descrizione

NEG1 inverte il segno al valore presente nella variabile ad 1 byte indicata dall’argomento OperandoByte.
Tale istruzione consente di “aggiustare” il segno del risultato di un’operazione matematica, come la moltiplicazione e la divisione senza segno, dopo aver eseguito l’operazione sui valori assoluti degli operandi.
L’istruzione NEG1 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: l’istruzione NEG1 esegue un cambio di segno della variabile ad ogni ciclo di scansione per il quale essa è alimentata. Per effettuare una sola inversione in corrispondenza di una certa condizione utilizzare un dispositivo generatore d’impulso per alimentare l’istruzione.

Esempio

Il seguente esempio nega il valore della variabile ad 1 byte M.100 ogni qual volta si verifica un fronte di salita del bit 0.0.0:

```
LD      0.0.0       ‘segnale il cui fronte esegue la negazione
OUT     P.0.IN      ‘alimentazione del generatore d’impulso

LD      P.0.OUTU    ‘impulso su fronte di salita per un ciclo di scansione
NEG1    M.100       ‘M.100 <---  - M.100
```

Vedi anche

NEG2, NEG4, ABS1

NEG2

Inversione di segno di una variabile a 2 bytes

Sintassi

```
NEG2        OperandoByte        ‘commento
```

Argomento

OperandoByte è l’indirizzo o la relativa label di qualsiasi byte della memoria ram.

Descrizione

NEG2 inverte il segno al valore presente nella variabile a 2 bytes indicata dall’argomento OperandoByte.
Tale istruzione consente di “aggiustare” il segno del risultato di un’operazione matematica, come la moltiplicazione e la divisione senza segno, dopo aver eseguito l’operazione sui valori assoluti degli operandi.
L’istruzione NEG2 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: l’istruzione NEG2 esegue un cambio di segno della variabile ad ogni ciclo di scansione per il quale essa è alimentata. Per effettuare una sola inversione in corrispondenza di una certa condizione utilizzare un dispositivo generatore d’impulso per alimentare l’istruzione.

Esempio

Il seguente esempio nega il valore della variabile a 2 bytes M.100 ogni qual volta si verifica un fronte di salita del bit 0.0.0:

```
LD      0.0.0       ‘segnale il cui fronte esegue la negazione
OUT     P.0.IN      ‘alimentazione del generatore d’impulso

LD      P.0.OUTU    ‘impulso su fronte di salita per un ciclo di scansione
NEG2    M.100       ‘M.101÷100 <---  - M.101÷100
```

Vedi anche

NEG1, NEG4, ABS2

NEG4

Inversione di segno di una variabile a 4 bytes

Sintassi

NEG4	OperandoByte	‘commento

Argomento

OperandoByte è l’indirizzo o la relativa label di qualsiasi byte della memoria ram.

Descrizione

NEG4 inverte il segno al valore presente nella variabile a 4 bytes indicata dall’argomento OperandoByte.
Tale istruzione consente di “aggiustare” il segno del risultato di un’operazione matematica, come la moltiplicazione e la divisione senza segno, dopo aver eseguito l’operazione sui valori assoluti degli operandi.
L’istruzione NEG4 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: l’istruzione NEG4 esegue un cambio di segno della variabile ad ogni ciclo di scansione per il quale essa è alimentata. Per effettuare una sola inversione in corrispondenza di una certa condizione utilizzare un dispositivo generatore d’impulso per alimentare l’istruzione.

Esempio

Il seguente esempio nega il valore della variabile a 4 bytes M.100 ogni qual volta si verifica un fronte di salita del bit 0.0.0:

```
LD      0.0.0       ‘segnale il cui fronte esegue la negazione
OUT     P.0.IN      ‘alimentazione del generatore d’impulso

LD      P.0.OUTU    ‘impulso su fronte di salita per un ciclo di scansione
NEG4    M.100       ‘M.103÷100 <---  - M.103÷100
```

Vedi anche

NEG1, NEG2, ABS4

BINBCD1

Conversione da binario a BCD di una variabile ad 1 byte su variabile ad 1 byte

Sintassi

```
BINBCD1  OperandoByte1   OperandoByte2        ‘commento
```

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.

Descrizione

BINBCD1 converte il valore binario senza segno della variabile o costante ad 1 byte indicata da OperandoByte2 nel formato BCD a due cifre posizionandolo sulla variabile ad 1 byte indicata OperandoByte1.
Essendo il risultato della BINBCD1 espresso mediante due cifre decimali, il campo di validità del valore binario da convertire è 0÷99.
L’istruzione BINBCD1 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni di conversione possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio converte il valore binario della variabile ad 1 byte M.200 e posiziona il risultato in BCD a 2 cifre nella variabile ad 1 byte M.100:

```
LD        0.0.0                    ‘condizione di abilitazione
MOV1      M.200     K.53           ‘valore 53 in M.200
BINBCD1   M.100     M.200          ‘M.100 <--- 0101 0011B
```

Vedi anche

BINBCD2, BINBCD4

BINBCD2

Conversione da binario a BCD di una variabile a 2 bytes su variabile a 2 bytes

Sintassi

BINBCD2 OperandoByte1 OperandoByte2 ‘commento

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 2 bytes.

Descrizione

BINBCD2 converte il valore binario senza segno della variabile o costante a 2 bytes indicata da OperandoByte2 nel formato BCD a quattro cifre posizionandolo sulla variabile a 2 bytes indicata OperandoByte1.
Essendo il risultato della BINBCD2 espresso mediante quattro cifre decimali, il campo di validità del valore binario da convertire è 0÷9999.
L’istruzione BINBCD2 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni di conversione possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio converte il valore binario della variabile a 2 byte M.200 e posiziona il risultato in BCD a quattro cifre nella variabile a 2 byte M.100:

```
LD        0.0.0                 ‘condizione di abilitazione
MOV2      M.200     K.3567      ‘valore 3567 in M.201÷200
BINBCD2   M.100     M.200       ‘M.101÷100 <--- 0011 0101 0110 0111B
```

Vedi anche

BINBCD1, BINBCD4

BINBCD4

Conversione da binario a BCD di una variabile a 4 bytes su variabile a 4 bytes

Sintassi

BINBCD4 OperandoByte1 OperandoByte2 ‘commento

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 4 bytes.

Descrizione

BINBCD4 converte il valore binario senza segno della variabile o costante a 4 bytes indicata da OperandoByte2 nel formato BCD a otto cifre posizionandolo sulla variabile a 4 bytes indicata OperandoByte1.
Essendo il risultato della BINBCD4 espresso mediante otto cifre decimali, il campo di validità del valore binario da convertire è 0÷99999999.
L’istruzione BINBCD4 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni di conversione possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio converte il valore binario della variabile a 4 byte M.200 e posiziona il risultato in BCD ad otto cifre nella variabile a 4 byte M.100:

```
LD        0.0.0                 ‘condizione di abilitazione
MOV4      M.200     K.85463567  ‘valore 85463567 in M.203÷200
BINBCD4   M.100     M.200       ‘M.103÷102 <--- 1000 0101 0100 0110B
                                ‘M.101÷100 <--- 0011 0101 0110 0111B
```

Vedi anche

BINBCD1, BINBCD2

BCDBIN1

Conversione da BCD a binario di una variabile ad 1 byte su variabile ad 1 byte

Sintassi

BCDBIN1 OperandoByte1 OperandoByte2 ‘commento

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.

Descrizione

BCDBIN1 converte il valore BCD a due cifre della variabile o costante ad 1 byte indicata da OperandoByte2 nel formato binario senza segno posizionandolo sulla variabile ad 1 byte indicata OperandoByte1.
Il valore del byte, al fine di essere correttamente convertito, deve avere una valida rappresentazione consistente nella condizione che ogni suo nibble sia nel campo 0÷9.
L’istruzione BCDBIN1 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni di conversione possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio converte il valore BCD a due cifre della variabile ad 1 byte M.200 e posiziona il risultato in binario nella variabile ad 1 byte M.100:

```
LD        0.0.0                 ‘condizione di abilitazione
MOV1      M.200     K.53        ‘valore 53 in M.200
BCDBIN1   M.100     M.200       ‘M.100 <--- 35
```

Vedi anche

BCDBIN2, BCDBIN4

BCDBIN2

Conversione da BCD a binario di una variabile a 2 bytes su variabile a 2 bytes

Sintassi

BCDBIN2 OperandoByte1 OperandoByte2 ‘commento

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 2 bytes.

Descrizione

BCDBIN2 converte il valore BCD a quattro cifre della variabile o costante a 2 bytes indicata da OperandoByte2 nel formato binario senza segno posizionandolo sulla variabile a 2 bytes indicata OperandoByte1.
I valori dei bytes, al fine di essere correttamente convertiti, devono avere una valida rappresentazione consistente nella condizione che ogni loro nibble sia nel campo 0÷9.
L’istruzione BCDBIN2 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni di conversione possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio converte il valore BCD a quattro cifre della variabile a 2 bytes M.200 e posiziona il risultato in binario nella variabile a 2 bytes M.100:

```
LD        0.0.0                 ‘condizione di abilitazione
MOV2      M.200     K.13671     ‘valore 13671 in M.201÷200
BCDBIN2   M.100     M.200       ‘M.101÷100 <--- 3567
```

Vedi anche

BCDBIN1, BCDBIN4

BCDBIN4

Conversione da BCD a binario di una variabile a 4 bytes su variabile a 4 bytes

Sintassi

BCDBIN4 OperandoByte1 OperandoByte2 ‘commento

Argomento

OperandoByte1 (destinazione) è l’indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l’indirizzo o label di qualsiasi byte oppure un valore costante a 4 bytes.

Descrizione

BCDBIN4 converte il valore BCD ad otto cifre della variabile o costante a 4 bytes indicata da OperandoByte2 nel formato binario senza segno posizionandolo sulla variabile a 4 bytes indicata OperandoByte1.
I valori dei bytes, al fine di essere correttamente convertiti, devono avere una valida rappresentazione consistente nella condizione che ogni loro nibble sia nel campo 0÷9.
L’istruzione BCDBIN4 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni di conversione possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio converte il valore BCD ad otto cifre della variabile a 4 bytes M.200 e posiziona il risultato in binario nella variabile a 4 bytes M.100:

```
LD        0.0.0                    ‘condizione di abilitazione
MOV4      M.200     K.87453567H    ‘valore 87453567H in M.203÷200
BCDBIN4   M.100     M.200          ‘M.103÷100 <--- 87453567
```

Vedi anche

BCDBIN1, BCDBIN2

SWAP

Scambio nibbles di una variabile ad 1 byte

Sintassi

```
SWAP      OperandoByte          ‘commento
```

Argomento

OperandoByte è l’indirizzo o la relativa label di qualsiasi byte della memoria ram.

Descrizione

SWAP esegue lo scambio reciproco tra i due nibbles del byte indicato OperandoByte.
L’istruzione SWAP viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: l’istruzione SWAP esegue un scambio di nibbles del byte ad ogni ciclo di scansione per il quale essa è alimentata. Per effettuare un solo scambio in corrispondenza di una certa condizione utilizzare un dispositivo generatore d’impulso per alimentare l’istruzione.

Esempio

Il seguente esempio scambia tra loro i due nibbles del byte M.100 ogni qual volta si verifica un fronte di salita del bit 0.0.0:

```
LD        0.0.0          ‘segnale il cui fronte esegue lo scambio
OUT       P.0.IN         ‘alimentazione del generatore d’impulso

LD        P.0.OUTU       ‘impulso su fronte di salita per un ciclo di scansione
SWAP      M.100          ‘M.100.7 <---> M.100.3
                         ‘M.100.6 <---> M.100.2
                         ‘M.100.5 <---> M.100.1
                         ‘M.100.4 <---> M.100.0
```

RCL1

Carica una variabile o costante ad 1 byte nella catasta operativa

Sintassi

RCL1 OperandoByte ‘commento

Argomento

OperandoByte è l’indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.

Descrizione

RCL1 carica il valore della variabile o costante ad 1 byte indicata da OperandoByte nella prima posizione (livello 0) della catasta operativa; i valori già presenti nella catasta vengono preventivamente spostati verso i livelli crescenti di una posizione, con conseguente perdita del valore presente nel livello 3. Il valore della nuova variabile caricata è inteso ad 1 byte con segno per cui viene automaticamente convertito nel formato a 4 bytes con segno.
L’istruzione RCL1 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio carica nella catasta operativa il valore della variabile ad 1 byte con segno M.100 se il bit 0.0.0 è ON:

```
LD      0.0.0
RCL1    M.100
```

L’effetto sulla catasta è il seguente:

```
CATASTA(3) <--- CATASTA(2)
CATASTA(2) <--- CATASTA(1)
CATASTA(1) <--- CATASTA(0)
CATASTA(0) <--- M.100
```

Forma abbreviata

R1 OperandoByte ‘commento

Vedi anche

RCL2, RCL4, STO1, STO2, STO4, ADD, SUB, MUL, DIV, CMP

RCL2

Carica una variabile o costante a 2 bytes nella catasta operativa

Sintassi

RCL2 OperandoByte ‘commento

Argomento

OperandoByte è l’indirizzo o label di qualsiasi byte oppure un valore costante a 2 bytes.

Descrizione

RCL2 carica il valore della variabile o costante a 2 bytes indicata da OperandoByte nella prima posizione (livello 0) della catasta operativa; i valori già presenti nella catasta vengono preventivamente spostati verso i livelli crescenti di una posizione, con conseguente perdita del valore presente nel livello 3. Il valore della nuova variabile caricata è inteso a 2 bytes con segno per cui viene automaticamente convertito nel formato a 4 bytes con segno.
L’istruzione RCL2 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio carica nella catasta operativa il valore della variabile a 2 bytes con segno M.100 se il bit 0.0.0 è ON:

```
LD      0.0.0
RCL2    M.100
```

L’effetto sulla catasta è il seguente:

```
CATASTA(3) <--- CATASTA(2)
CATASTA(2) <--- CATASTA(1)
CATASTA(1) <--- CATASTA(0)
CATASTA(0) <--- M.101÷100
```

Forma abbreviata

R2 OperandoByte ‘commento

Vedi anche

RCL1, RCL4, STO1, STO2, STO4, ADD, SUB, MUL, DIV, CMP

RCL4

Carica una variabile o costante a 4 bytes nella catasta operativa

Sintassi

RCL4 OperandoByte ‘commento

Argomento

OperandoByte è l’indirizzo o label di qualsiasi byte oppure un valore costante a 4 bytes.

Descrizione

RCL4 carica il valore della variabile o costante a 4 bytes indicata da OperandoByte nella prima posizione (livello 0) della catasta operativa; i valori già presenti nella catasta vengono preventivamente spostati verso i livelli crescenti di una posizione, con conseguente perdita del valore presente nel livello 3. Il valore della nuova variabile caricata è inteso a 4 bytes con segno e in questo caso non viene convertito in quanto è già nel formato richiesto dalla catasta. L’istruzione RCL4 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio carica nella catasta operativa il valore della variabile a 4 bytes con segno M.100 se il bit 0.0.0 è ON:

```
LD      0.0.0
RCL4    M.100
```

L’effetto sulla catasta è il seguente:

```
CATASTA(3) <--- CATASTA(2)
CATASTA(2) <--- CATASTA(1)
CATASTA(1) <--- CATASTA(0)
CATASTA(0) <--- M.103÷100
```

Forma abbreviata

R4 OperandoByte ‘commento

Vedi anche

RCL1, RCL2, STO1, STO2, STO4, ADD, SUB, MUL, DIV, CMP

STO1

Copia il valore del livello 0 della catasta operativa su una variabile ad 1 byte

Sintassi

STO1 OperandoByte ‘commento

Argomento

OperandoByte è l’indirizzo o la relativa label di qualsiasi byte della memoria ram.

Descrizione

STO1 copia il valore della prima posizione (livello 0) della catasta operativa sulla variabile ad 1 byte indicata da OperandoByte. Il valore presente nel livello 0 della catasta viene preventivamente convertito dal formato a 4 bytes con segno al formato ad 1 byte con segno, prima di essere copiato nella memoria; tuttavia tutti i valori presenti nei quattro livelli della catasta non vengono modificati o spostati di posizione. Se il valore presente nella catasta non può essere convertito nel formato ad 1 bytes con segno viene restituito il flag di errore F.E allo stato logico “1”.
L’istruzione STO1 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio copia il valore presente nella catasta operativa sulla variabile ad 1 byte con segno M.100 se il bit 0.0.0 è ON:

```
LD      0.0.0
STO1    M.100
```

L’effetto sulla catasta è il seguente: M.100 <--- CATASTA(0)

Forma abbreviata

S1 OperandoByte ‘commento

Vedi anche

RCL1, RCL2, RCL4, STO2, STO4, ADD, SUB, MUL, DIV, CMP

STO2

Copia il valore del livello 0 della catasta operativa su una variabile a 2 bytes

Sintassi

STO2 OperandoByte ‘commento

Argomento

OperandoByte è l’indirizzo o la relativa label di qualsiasi byte della memoria ram.

Descrizione

STO2 copia il valore della prima posizione (livello 0) della catasta operativa sulla variabile a 2 bytes indicata da OperandoByte. Il valore presente nel livello 0 della catasta viene preventivamente convertito dal formato a 4 bytes con segno al formato a 2 bytes con segno, prima di essere copiato nella memoria; tuttavia tutti i valori presenti nei quattro livelli della catasta non vengono modificati o spostati di posizione. Se il valore presente nella catasta non può essere convertito nel formato a 2 bytes con segno viene restituito il flag di errore F.E allo stato logico “1”.
L’istruzione STO2 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio copia il valore presente nella catasta operativa sulla variabile a 2 bytes con segno M.100 se il bit 0.0.0 è ON:

```
LD      0.0.0
STO2    M.100
```

L’effetto sulla catasta è il seguente: M.101÷100 <--- CATASTA(0)

Forma abbreviata

S2 OperandoByte ‘commento

Vedi anche

RCL1, RCL2, RCL4, STO1, STO4, ADD, SUB, MUL, DIV, CMP

STO4

Copia il valore del livello 0 della catasta operativa su una variabile a 4 bytes

Sintassi

STO4 OperandoByte ‘commento

Argomento

OperandoByte è l’indirizzo o la relativa label di qualsiasi byte della memoria ram.

Descrizione

STO4 copia il valore della prima posizione (livello 0) della catasta operativa sulla variabile a 4 bytes indicata da OperandoByte. Il valore presente nel livello 0 della catasta non viene convertito in quanto è già nel formato a 4 bytes con segno; tutti i valori presenti nei quattro livelli della catasta non vengono modificati o spostati di posizione.
L’istruzione STO4 viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio copia il valore presente nella catasta operativa sulla variabile a 4 bytes con segno M.100 se il bit 0.0.0 è ON:

```
LD      0.0.0
STO4    M.100
```

L’effetto sulla catasta è il seguente: M.103÷100 <--- CATASTA(0)

Forma abbreviata

S4 OperandoByte ‘commento

Vedi anche

RCL1, RCL2, RCL4, STO1, STO2, ADD, SUB, MUL, DIV, CMP

ADD

Somma il valore del livello 1 con il valore del livello 0 della catasta operativa

Sintassi

ADD ‘commento

Argomento

Nessuno.

Descrizione

ADD somma il valore nella seconda posizione (livello 1) con il valore nella prima posizione (livello 0) della catasta operativa, posizionando il risultato nel livello 0. I valori già presenti negli altri livelli della catasta vengono successivamente spostati verso i livelli decrescenti di una posizione. Se l’operazione di somma a 4 bytes con segno determina un overflow viene restituito il flag di errore F.E allo stato logico “1”.
L’istruzione ADD viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni matematiche eseguite sulla catasta operativa possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio somma i valori presenti nei livelli 1 e 0 della catasta se il bit 0.0.0 è ON:

```
LD      0.0.0
ADD
```

L’effetto sulla catasta è il seguente:

```
CATASTA(0) <--- CATASTA(1) + CATASTA(0)
CATASTA(1) <--- CATASTA(2)
CATASTA(2) <--- CATASTA(3)
```

Forma abbreviata

\+ ‘commento

Vedi anche

RCL1, RCL2, RCL4, STO1, STO2, STO4, SUB, MUL, DIV, CMP

SUB

Sottrae al valore del livello 1 il valore del livello 0 della catasta operativa

Sintassi

SUB ‘commento

Argomento

Nessuno.

Descrizione

SUB sottrae al valore nella seconda posizione (livello 1) il valore nella prima posizione (livello 0) della catasta operativa, posizionando il risultato nel livello 0. I valori già presenti negli altri livelli della catasta vengono successivamente spostati verso i livelli decrescenti di una posizione.
Se l’operazione di sottrazione a 4 bytes con segno determina un overflow (borrow) viene restituito il flag di errore F.E allo stato logico “1”.
L’istruzione SUB viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni matematiche eseguite sulla catasta operativa possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio sottrae i valori presenti nei livelli 1 e 0 della catasta se il bit 0.0.0 è ON:

```
LD      0.0.0
SUB
```

L’effetto sulla catasta è il seguente:

```
CATASTA(0) <--- CATASTA(1) - CATASTA(0)
CATASTA(1) <--- CATASTA(2)
CATASTA(2) <--- CATASTA(3)
```

Forma abbreviata

- ‘commento

Vedi anche

RCL1, RCL2, RCL4, STO1, STO2, STO4, ADD, MUL, DIV, CMP

MUL

Moltiplica il valore del livello 1 con il valore del livello 0 della catasta operativa

Sintassi

MUL ‘commento

Argomento

Nessuno.

Descrizione

MUL moltiplica il valore nella seconda posizione (livello 1) con il valore nella prima posizione (livello 0) della catasta operativa, posizionando il risultato nel livello 0. I valori già presenti negli altri livelli della catasta vengono successivamente spostati verso i livelli decrescenti di una posizione. Se l’operazione di moltiplicazione a 4 bytes con segno determina un overflow viene restituito il flag di errore F.E allo stato logico “1”.
L’istruzione MUL viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni matematiche eseguite sulla catasta operativa possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio moltiplica i valori presenti nei livelli 1 e 0 della catasta se il bit 0.0.0 è ON:

```
LD      0.0.0
MUL
```

L’effetto sulla catasta è il seguente:

```
CATASTA(0) <--- CATASTA(1) * CATASTA(0)
CATASTA(1) <--- CATASTA(2)
CATASTA(2) <--- CATASTA(3)
```

Forma abbreviata

* ‘commento

Vedi anche

RCL1, RCL2, RCL4, STO1, STO2, STO4, ADD, SUB, DIV, CMP

DIV

Divide il valore del livello 1 per il valore del livello 0 della catasta operativa

Sintassi

DIV ‘commento

Argomento

Nessuno.

Descrizione

DIV divide il valore nella seconda posizione (livello 1) per il valore nella prima posizione (livello 0) della catasta operativa, posizionando il risultato (quoziente intero con segno) nel livello 0. I valori già presenti negli altri livelli della catasta vengono successivamente spostati verso i livelli decrescenti di una posizione. Se nel livello 0 è presente un divisore nullo l’operazione non viene eseguita e viene restituito il flag di errore F.E allo stato logico “1”.
L’istruzione DIV viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Attenzione: le istruzioni matematiche eseguite sulla catasta operativa possono richiedere tempi di esecuzione elevati. Verificare nelle relative tabelle il tempo richiesto alla loro esecuzione e, nel caso sia opportuno, alimentare le istruzioni solo per un ciclo di scansione oppure solo in stati del programma non critici per tempo di ciclo.

Esempio

Il seguente esempio divide i valori presenti nei livelli 1 e 0 della catasta se il bit 0.0.0 è ON:

```
LD      0.0.0
DIV
```

L’effetto sulla catasta è:

```
CATASTA(0) <--- INT(CATASTA(1) / CATASTA(0))
CATASTA(1) <--- CATASTA(2)
CATASTA(2) <--- CATASTA(3)
```

Forma abbreviata

/ ‘commento

Vedi anche

RCL1, RCL2, RCL4, STO1, STO2, STO4, ADD, SUB, MUL, CMP

CMP

Compara il valore del livello 1 con il valore del livello 0 della catasta operativa

Sintassi

CMP ‘commento

Argomento

Nessuno.

Descrizione

CMP compara il valore nella seconda posizione (livello 1) con il valore nella prima posizione (livello 0) della catasta operativa. Il risultato del confronto è disponibile, immediatamente dopo tale istruzione, nei bits speciali di Flag (F.<, F.=, F.>), che possono essere verificati da qualsiasi istruzione booleana. Tutti i valori presenti nei quattro livelli della catasta non vengono modificati o spostati di posizione.
L’istruzione CMP viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio compara i valori presenti nei livelli 1 e 0 della catasta se il bit 0.0.0 è ON:

```
LD      0.0.0
CMP
```

RISULTATO	F.<	F.=	F.>
CATASTA(1) < CATASTA(0)	ON	OFF	OFF
CATASTA(1) = CATASTA(0)	OFF	ON	OFF
CATASTA(1) > CATASTA(0)	OFF	OFF	ON

Forma abbreviata

? ‘commento

Vedi anche

RCL1, RCL2, RCL4, STO1, STO2, STO4, ADD, SUB, MUL, DIV

MOVADD

Muove l'indirizzo assoluto di una variabile su una variabile a 2 bytes

Sintassi

MOVADD OperandoByte1 OperandoByte2 'commento

Argomento

OperandoByte1 (puntatore) è l'indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l'indirizzo o label di qualsiasi byte della memoria.

Descrizione

MOVADD scrive il valore dell'indirizzo assoluto di RAM della variabile indicata da OperandoByte2 sulla variabile a 2 bytes indicata OperandoByte1 che assume le funzioni di puntatore di OperandoByte2.
Per identificare una qualsiasi variabile mediante il suo indirizzo assoluto occorre utilizzare una variabile puntatore a due bytes contenente il valore stesso dell'indirizzo a 16 bits; se la variabile da puntare è a 2 o 4 bytes il valore del puntatore corrisponde all'indirizzo assoluto del byte normalmente utilizzato per identificare la variabile (bytes meno significativo).
L'istruzione MOVADD risulta indispensabile nell'utilizzo degli operandi indiretti. Il vantaggio più evidente offerto dagli operandi espressi in modo indiretto è che variando il valore del puntatore è possibile applicare le stesse operazioni ad insiemi di variabili, anziché ad un determinato nome di variabile. La MOVADD permette di inizializzare il puntatore ad una specifica variabile dell'insieme; successivi incrementi o decrementi del valore del puntatore permetteranno di operare con la stessa parte di programma su tutte le altre variabili.
L'istruzione MOVADD viene eseguita solo se l'ultimo bit inserito nel registro accumulatore è al valore logico "1". La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio muove nella variabile a 2 bytes M.100 il puntatore della variabile H.0:

```
LD       F.1                    'condizione sempre abilitata
MOVADD   M.100    H.0           'M101÷100 <--- 37888
```

MOVASC

Muove una stringa di testo su un'area della memoria

Sintassi

MOVASC OperandoByte |Stringa| 'commento

Argomento

OperandoByte (destinatario) è l'indirizzo o label di qualsiasi byte della memoria.
Stringa è una sequenza di caratteri ASCII di lunghezza massima 100 caratteri.

Descrizione

MOVASC riempie l'area di memoria RAM, ad iniziare dal byte indicato da OperandoByte, con la sequenza dei codici ASCII dei caratteri corrispondenti alla Stringa fornita.
La stringa di testo deve essere delimitata tra due caratteri del tipo "doppio tratto verticale" e può contenere fino a 100 caratteri ad eccezione di tale delimitatore.
All'interno della stringa i tre caratteri @ (chiocciola), \ (barra rovesciata) ed ^ (apice) assumono un significato particolare; infatti questi caratteri sono automaticamente sostituiti dal compilatore rispettivamente con i codici ASCII 13, 10, 12 corrispondenti ai caratteri RITORNO CARRELLO, SALTO RIGA e SALTO PAGINA. Ciò permette di posizionare all'interno dell'area di stringa questi tre caratteri speciali, spesso necessari nella comunicazione verso l'esterno delle stringhe stesse. Se si desidera memorizzare nei bytes della RAM i veri caratteri @, \, ^ senza che vengano sostituiti dai codici 13, 10, 12, occorre muovere in ciascun byte, mediante l'istruzione MOV1, il valore decimale 64, 92, 94 corrispondente al vero codice ASCII del carattere voluto.
L'istruzione MOVASC viene eseguita solo se l'ultimo bit inserito nel registro accumulatore è al valore logico "1". La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio muove nell'area di ram, con inizio dal byte M.100, la sequenza di codici ASCII corrispondenti alla stringa indicata:

```
LD      F.1
MOVASC  M.100     |ESEMPIO DI STRINGA@|
```

l'effetto di tale istruzione è:

M.100÷118 <--- 69,115,101,109,112,105,111,32,100,105,32,115,116,114,105,110,103,97,13

MOVBLK

Muove un'area della memoria su un'altra area di memoria

Sintassi

MOVBLK OperandoByte1 OperandoByte2 OperandoByte3 'commento

Argomento

OperandoByte1 (destinazione) è l'indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 (sorgente) è l'indirizzo o label di qualsiasi byte della memoria.
OperandoByte3 è l'indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.

Descrizione

MOVBLK copia l'area di memoria RAM iniziata da OperandoByte2 sull'area di memoria RAM iniziata da OperandoByte1; il numero di bytes trasferiti è pari al valore di OperandoByte3. L'operazione di copia multipla equivale alla ripetizione dell'istruzione MOV1 per il numero di bytes indicato. Il valore di OperandoByte3 può essere costante oppure variabile purchè di dimensione ad 1 byte ed in ogni caso il suo valore deve essere compreso nel campo 0÷255.
L'istruzione MOVBLK viene eseguita solo se l'ultimo bit inserito nel registro accumulatore è al valore logico "1". La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio muove nell'area di ram, con inizio dal byte M.100, l'area di ram con inizio da M.200 per un totale di 18 bytes:

```
LD       0.0.0                          'condizione di abilitazione
MOVBLK   M.100    M.200    K.18         'M100÷117 <--- M200÷217
```

Vedi anche

MOV1, MOV2, MOV4

CMPBLK

Compara un'area della memoria con un'altra area di memoria

Sintassi

CMPBLK OperandoByte1 OperandoByte2 OperandoByte3 'commento

Argomento

OperandoByte1 è l'indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l'indirizzo o label di qualsiasi byte della memoria.
OperandoByte3 è l'indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.

Descrizione

CMPBLK compara l'area di memoria RAM iniziata da OperandoByte1 con l'area di memoria iniziata da OperandoByte2; il numero di bytes comparati è pari al valore di OperandoByte3. Il valore di OperandoByte3 può essere costante oppure variabile purchè di dimensione ad 1 byte ed in ogni caso il suo valore deve essere compreso nel campo 0÷255.
I bytes delle due aree di memoria vengono comparati a due a due per tutta l'estensione delle aree ad iniziare dai bytes di indirizzo inferiore. Se tutti i bytes di pari posizione delle due aree sono uguali, il flag di uguaglianza F.= viene forzato ON, altrimenti il confronto è sospeso alla prima diversità; in tal caso il flag F.< oppure F.> è forzato ON se l'esito della comparazione della coppia corrente di bytes è rispettivamente minore o maggiore.
L'istruzione CMPBLK viene eseguita solo se l'ultimo bit inserito nel registro accumulatore è al valore logico "1". La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

L'esempio compara l'area di 18 bytes con inizio da M.100 con l'area con inizio da M.200:

```
LD      0.0.0                       'condizione di abilitazione
CMPLK   M.100   M.200   K.18        'compara M100÷117 con M200÷217
```

RISULTATO	F.<	F.=	F.>
M.100...M.117 < M.200...M.217	ON	OFF	OFF
M.100...M.117 = M.200...M.217	OFF	ON	OFF
M.100...M.117 > M.200...M.217	OFF	OFF	ON

RESMEM

Azzera un'area della memoria

Sintassi

RESMEM OperandoByte1 OperandoByte2 'commento

Argomento

OperandoByte1 è l'indirizzo o label di qualsiasi byte della memoria.
OperandoByte2 è l'indirizzo o label di qualsiasi byte oppure un valore costante ad 1 byte.

Descrizione

RESMEM azzera l'area di memoria RAM iniziata da OperandoByte1; il numero di bytes azzerati è pari al valore di OperandoByte2.
Il valore di OperandoByte2 può essere costante oppure variabile purchè di dimensione ad 1 byte ed in ogni caso il suo valore deve essere compreso nel campo 0÷255.
L'istruzione RESMEM viene eseguita solo se l'ultimo bit inserito nel registro accumulatore è al valore logico "1". La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio muove il valore costante 0 in tutti i 18 bytes dell'area di ram con inizio dal byte M.100:

```
LD        0.0.0                    'condizione di abilitazione
RESMEM    M.100      K.18          'M100÷117 <--- 0,,,0
```

IOREFR

Aggiorna l'area dati corrispondente alle risorse esterne del MASTER

Sintassi

```
IOREFR                                  'commento
```

Argomento

Nessuno.

Descrizione

IOREFR permette di aggiornare l'area dati corrispondente all'immagine di processo degli I/O locali del MASTER.
Tale istruzione è automaticamente eseguita ad ogni ciclo di scansione dal sistema operativo; per questo motivo il suo utilizzo è limitato solo ai casi nei quali occorre aggiornare gli I/O locali più volte per ogni ciclo di scansione, in punti intermedi al programma utente. Si consideri che l'aggiornamento forzato degli I/O avviene solo ed esclusivamente per le risorse esterne locali della sola scheda MASTER, senza coinvolgere la comunicazione in rete con gli SLAVES.
L'istruzione IOREFR viene eseguita solo se l'ultimo bit inserito nel registro accumulatore è al valore logico "1".
Attenzione: la lista a bits del registro accumulatore può venire modificata dall'esecuzione di tale istruzione.

Esempio

Il seguente esempio aggiorna gli I/O locali del MASTER se il bit M.0.0 è ON:

```
LD      M.0.0                   'condizione di abilitazione
IOREFR                          'aggiornamento forzato degli I/O
```

RESWD

Reset del temporizzatore di sorveglianza del sistema (watch-dog)

Sintassi

```
RESWD                                   ‘commento
```

Argomento

Nessuno.

Descrizione

RESWD permette di riazzerare il valore corrente totalizzato dal timer di watch-dog della scheda MASTER.
Il timer di watch-dog è un dispositivo hardware incorporato nella Logica che tiene sotto controllo il corretto funzionamento della stessa. Un contatore viene incrementato ad intervalli fissi in modo automatico e non disabilitabile, mentre il sistema operativo della Logica provvede, in condizioni di normale scansione, a resettarlo. Nel caso che ciò non avvenga, il contatore, raggiunto il suo fondoscala, resetterà in modo hardware la Logica, ripristinandone le funzioni come dopo una mancanza di alimentazione.
Il reset forzato del circuito di watch-dog all’interno del listato programma può essere necessario in programmi con cicli di scansione molto lunghi. Tuttavia si consiglia di non arrivare mai a tali limiti strutturando con molta cura i programmi ed ottimizzando l’utilizzo delle istruzioni.
L’istruzione RESWD viene eseguita solo se l’ultimo bit inserito nel registro accumulatore è al valore logico “1”. La lista a bits del registro accumulatore non viene modificata e quindi è possibile alimentare più istruzioni con lo stesso ramo calcolato.

Esempio

Il seguente esempio resetta il valore raggiunto dal temporizzatore di watch-dog del MASTER se il bit M.0.0 è ON:

```
LD          M.0.0               ‘condizione di abilitazione
RESWD                           ‘reset forzato del circuito di watch-dog
```

INCLUDE

Include nel programma la lista istruzioni di un altro file

Sintassi

INCLUDE NomeFile ‘commento

Argomento

NomeFile è il nome di un file senza estensione .PRG e contenente istruzioni di programma.

Descrizione

INCLUDE permette di inserire in un particolare punto del listato programma l’elenco delle istruzioni contenute in un secondo file presente nella directory di compilazione.
Il nome del file secondario deve essere dichiarato mediante gli 8 caratteri massimi consentiti e senza estensione .PRG. Il compilatore in presenza di tale istruzione sospende momentaneamente la lettura delle istruzioni dal file principale ed inizia a compilare tutte le istruzioni trovate nel file secondario; quando tutte le righe del file sono terminate, la compilazione riprende nel file principale dalla riga successiva a quella dell’istruzione INCLUDE. In questo modo è possibile frazionare il file sorgente del programma in più files, posizionando nei files secondari, per esempio, parti di programma comuni o ricorrenti o addirittura collezioni di subroutines di libreria.
Il compilatore è in grado di gestire un numero qualunque di istruzioni di inclusione purché nessun file sorgente secondario contenga a sua volta istruzioni di inclusione. Eventuali errori di sintassi riscontrati nelle istruzioni presenti in un file incluso vengono riportati nel file principale di errore .ERR con l’indicazione del file che contiene l’errore e della riga relativa a tale file.
L’istruzione INCLUDE non deve essere alimentata in nessun modo per la sua attivazione; essa è sempre eseguita in quanto consente di inserire intere liste di istruzioni nel programma.

Esempio

Il seguente esempio mostra come inserire nel programma tutte le istruzioni presenti nel file sorgente AUXPROG.PRG:

```
INCLUDE     AUXPROG          ‘inclusione delle istruzioni di AUXPROG
```

PASSW

Dichiara la password dello specifico programma

Sintassi

PASSW Stringa ‘commento

Argomento

Stringa è una qualunque sequenza di caratteri ASCII con il limite massimo di 8 caratteri.

Descrizione

PASSW permette di dichiarare, all’interno del listato programma, la parola chiave mediante la quale è possibile successivamente leggere il programma stesso dalla memoria del MASTER. L’istruzione PASSW può comparire in un qualunque punto del listato programma; il compilatore esegue tale istruzione memorizzando la stringa fornita secondo una codifica segreta ed in punti nascosti del codice oggetto. A questo punto qualunque operazione di trasferimento del codice oggetto dal MASTER al Computer non è consentita, a memo che non si specifichi, all’interno dell’ambiente di programmazione su Computer, una password uguale a quella memorizzata nel programma.
Si consideri che in assenza di tale istruzione, il compilatore provvederà automaticamente a dichiarare una password di default corrispondente alla stringa PASSWORD; in questo caso occorre selezionare nell’ambiente di programmazione su Computer una password uguale alla stringa PASSWORD.
L’istruzione PASSW non deve essere alimentata in nessun modo per la sua attivazione.

Esempio

Il seguente esempio mostra come dichiarare nel programma la password “ELISA”:

```
PASSW    ELISA    ‘dichiarazione della password
```

Operandi indiretti per le istruzioni su bytes

Tutte le istruzioni che operano su variabili di tipo bytes possono utilizzare operandi di tipo indiretto.

Un operando si dice di tipo indiretto quando il valore elaborato o modificato dall'istruzione non è quello direttamente contenuto nei bytes corrispondenti al suo identificatore ma è contenuto in un'altra area di memoria il cui indirizzo assoluto è memorizzato nei due bytes associati all'identificatore dell'operando.

Gli operandi indiretti sono di estrema importanza per eseguire delle operazioni ripetitive su un certo insieme di dati; utilizzati in un ciclo permettono di parametrizzare tutte le operazioni evitando di ripetere più volte la stessa parte di programma per ogni dato.

Un operando diventa indiretto facendo precedere il suo identificatore dal carattere @ (chiocciola) senza spazi intermedi. Ovviamente il simbolo utilizzato per l'identificatore dell'operando deve essere considerato come variabile a 2 bytes, tale da poter contenere l'indirizzo assoluto che punta all'effettivo operando.

In ciascuna istruzione possono comparire uno o più operandi indiretti. Ad esempio:

```
LD      M.0.0
MOVADD  M.100   M.110
MOVADD  M.200   M.210
MOVADD  M.300   M.310
MUL4    @M.100  @M.200  @M.300
```

rappresenta un'istruzione di moltiplicazione nella quale sia le variabili da moltiplicare che la variabile nella quale posizionare il prodotto, sono individuate da altrettanti puntatori (variabili a 2 bytes M.100, M.200, M.300). E' ovvio che, prima di utilizzare un'istruzione con uno o più operandi indiretti, occorre precaricare, negli operandi puntatori, gli indirizzi assoluti dei veri operandi (variabili a 4 bytes M.110, M.210, M.310).

L'istruzione MOVADD permette di inizializzare i puntatori; eventuali incrementi o decrementi dei valori dei puntatori permettono di effettuare le stesse operazioni su insiemi di dati.

Utilizzando operandi indiretti su istruzioni di trasferimento bytes è possibile generare sequenze di uscita mediante valori precaricati in apposite tabelle. Ad esempio:

```
LD      M.0.0
MOV1    M.100   K.11001010B
MOV1    M.101   K.00011010B
MOV1    M.102   K.11010101B
MOV1    M.103   K.01010111B
MOVADD  M.200   M.100
ADD2    M.200   M.200   M.300
MOV1    0.8     @M.200
```

il valore trasferito sul byte di uscita 0.8 corrisponde alla riga della tabella il cui numero è contenuto nella variabile M.300 (indirizzo di riga).

Valutazione delle espressioni matematiche

Il linguaggio ICL51 prevede un set di istruzioni che consentono una rapida e semplice valutazione delle espressioni, senza ricorrere alle classiche istruzioni matematiche dei controllori programmabili.

Si tratta di una tecnica di descrizione delle espressioni matematiche nota come “NOTAZIONE POLACCA INVERSA”. Questa tecnica è molto efficace in quanto consente di valutare espressioni molto complesse senza far uso di parentesi di raggruppamento; possono essere valutate espressioni frazionarie con più termini al numeratore e denominatore senza preoccuparsi di memorizzare temporaneamente alcun risultato intermedio.

La tecnica della notazione polacca inversa fa riferimento alla presenza di una CATASTA OPERATIVA; essa non è altro che una lista di tipo LIFO (Last In First Out) di variabili con formato 32 bits con segno. La catasta consente di memorizzare automaticamente ed in modo completamente trasparente al Programmatore fino a 4 risultati intermedi in altrettanti registri ciascuno con dimensione 4 bytes:

CATASTA(3)	livello 3
CATASTA(2)	livello 2
CATASTA(1)	livello 1
CATASTA(0)	livello 0

Occorre immaginare che le variabili entrano nella lista (mediante le istruzioni RCL1, RCL2, RCL4) dal basso, ossia dal livello 0, facendo scorrere i valori già presenti di una posizione verso l’alto:

CATASTA(3) <— CATASTA(2)
CATASTA(2) <— CATASTA(1)
CATASTA(1) <— CATASTA(0)
CATASTA(0) <— NUOVO VALORE

Prima di eseguire una qualsiasi istruzione matematica occorre dunque caricare nella catasta gli operandi di ingresso dell’istruzione stessa.

Ad esempio per calcolare la somma, indicata con S, di due variabili a 4 bytes con segno, indicate con A e B, occorre utilizzare due volte l’istruzione RCL4:

RCL4	A	CATASTA(3) <— CATASTA(2) CATASTA(2) <— CATASTA(1) CATASTA(1) <— CATASTA(0) CATASTA(0) <— A
RCL4	B	CATASTA(3) <— CATASTA(2) CATASTA(2) <— CATASTA(1) CATASTA(1) <— A CATASTA(0) <— B

A questo punto è possibile richiamare l'istruzione di somma ADD (mnemonico abbreviato +) con la quale viene calcolata la somma A+B; il risultato viene posizionato ancora in catasta, nella posizione di livello 0, ed automaticamente i valori presenti nelle posizioni superiori a quelle occupate dagli operandi elaborati dall'istruzione vengono fatti scendere di un livello:

ADD	CATASTA(2) <— CATASTA(3) CATASTA(1) <— CATASTA(2) CATASTA(0) <— A + B

Una volta calcolata la somma occorre trasferire il risultato dalla catasta alla variabile nella quale si desidera tale somma:

STO4	S	S <— CATASTA(0)

Il sistema della catasta consente di valutare espressioni molto complesse per le quali i risultati intermedi vengono automaticamente archiviati e recuperati al momento opportuno. Al primo approccio tale sistema può sembrare complicato; tuttavia, una volta appreso il meccanismo, ci si accorgerà delle enormi possibilità offerte nella valutazione di espressioni anche molto elaborate. Un Programmatore di Logiche non troverà difficoltà ad apprendere tale metodo poichè esso non è altro che l'estensione alle variabili numeriche della tecnica utilizzata per la descrizione dei rami elettromeccanici mediante le istruzioni booleane; per esempio l'istruzione LD corrisponde alla RCL1/2/4, l'istruzione OUT corrisponde alla STO1/2/4, mentre le istruzioni ANDLD, ORLD operano in modo equivalente alle istruzioni ADD, SUB, MUL, DIV.

Facciamo un'altro esempio appena più complicato. Valutiamo l'espressione:

R = A(B - C) + D

RCL4	B	CATASTA(0) <— B
RCL4	C	CATASTA(1) <— B CATASTA(0) <— C
SUB		CATASTA(0) <— B - C
RCL4	A	CATASTA(1) <— B - C CATASTA(0) <— A
MUL		CATASTA(0) <— A(B - C)
RCL4	D	CATASTA(1) <— A(B - C) CATASTA(0) <— D
ADD		CATASTA(0) <— A(B - C) + D
STO4	R	R <— CATASTA(0)

Tutte le istruzioni matematiche disponibili per l'elaborazione di dati in catasta lavorano su variabili con formato 32 bits con segno; questo permette di avere la massima precisione possibile dei risultati in quanto è possibile lavorare con grandezze comprese tra -2147483648 e +2147483647.

Per evitare il troncamento della parte decimale con l'istruzione DIV occorre premoltiplicare il dividendo per un'opportuna potenza del 10 prima di effettuare la divisione; il risultato di tipo intero ovviamente risulterà moltiplicato per la stesso fattore (rappresentazione in virgola fissa).

La valutazione delle espressioni matematiche è possibile anche per variabili di dimensioni diverse da 4 bytes con segno; possono infatti essere indifferentemente mescolate variabili ad 1 byte con segno, 2 bytes con segno, 4 bytes con segno. A questo scopo le istruzioni di richiamo e di archiviazione sono state previste in tre forme: RCL1 e STO1 interfacciano la catasta con le variabili ad 1 byte con segno, RCL2 e STO2 con le variabili a 2 bytes con segno, RCL4 e STO4 con le variabili a 4 bytes con segno.

Si ricordi che un insieme successivo di istruzioni corrispondenti alla valutazione completa di un'espressione dovrebbe essere alimentato solo per un solo ciclo di scansione, allo scopo di non allungare troppo il tempo di ciclo; il bit di alimentazione può essere unico per tutta la sequenza in quanto queste istruzioni non alterano il contenuto del registro accumulatore di valutazione del ramo elettromeccanico.

Facciamo infine un'ultimo esempio per meglio comprendere il funzionamento della tecnica di valutazione espressioni. Quando il bit M.0.0 è attivato ON, si vuole valutare una disequazione e, solo nel caso essa sia verificata, eseguire la subroutine MINORE:

$$\frac{\dfrac{A(B - 24C) - D}{256 + E} + 455}{\dfrac{F - G}{H} + 3455} < \frac{I - L}{M - 223}$$

Il listato corrispondente alla disequazione è il seguente:

```
LD        M.0.0
RCL4      B
RCL4      K.24
RCL4      C
*
-
RCL4      A
*
RCL4      D
-
RCL4      K.256
RCL4      E
+
/
RCL4      K.455
+
RCL4      F
RCL4      G
-
RCL4      H
/
RCL4      K.3455
+
/
RCL4      I
RCL4      L
-
RCL4      M
RCL4      K.223
-
/
?
AND       F.<
GOSUB     MINORE
```

Ottimizzare le prestazioni del programma

Il presente paragrafo illustra alcuni semplici accorgimenti di programmazione suggeriti al fine di ottenere le massime prestazioni dal sistema; in particolare ci riferiamo a prestazioni in termini di velocità di elaborazione del programma utente e di occupazione di memoria programma.

Notevoli incrementi di prestazioni, nel calcolo delle reti booleane, si ottengono utilizzando, per istruzioni successive, bits appartenenti allo stesso byte. Per fare un esempio, il seguente listato:

```
LD      M.0.0
AND     M.1.0
AND     M.2.0
AND     M.3.0
AND     M.4.0
AND     M.5.0
AND     M.6.0
OUT     M.7.0
```

comporta un'occupazione in termini di memoria di 49 bytes ed un incremento del tempo di ciclo di 49 µs. Operando una diversa scelta delle memorie, che vari il meno possibile l'indirizzo di byte, il ramo potrebbe essere così riscritto:

```
LD      M.0.0
AND     M.0.1
AND     M.0.2
AND     M.0.3
AND     M.0.4
AND     M.0.5
AND     M.0.6
OUT     M.0.7
```

Questa parte di programma ha un'occupazione di memoria di 21 bytes e comporta un incremento del tempo di ciclo di 21µs.

Se consideriamo le medie per singola istruzione di tali valori, nel primo caso abbiamo 6.125 bytes/istruzione e 6.125µs/istruzione, mentre nel secondo 2.625 bytes/istruzione e 2.625µs/istruzione; l'incremento di prestazioni, con la seconda scelta, è del 233%.

Ovviamente non sempre è possibile obbligare il listato programma a tali scelte. In generale tuttavia, l'abitudine, fin dall'inizio della stesura dello stesso, a prendere di volta in volta bits successivi di memoria, porterà statisticamente ad una percentuale elevata di istruzioni adiacenti che operano su bits appartenenti allo stesso byte: l'incremento di prestazioni sarà sicuramente ancora evidente.

Un'altra tecnica di programmazione che permette di aumentare le prestazioni del sistema è quella di utilizzare per i numeri identificatori delle schede SLAVES, dei dispositivi CONTATORI e dei dispositivi GENERATORI D'IMPULSO, numeri più bassi possibile senza buchi intermedi di numerazione. Questo perché il compilatore individua, per ognuno dei tre insiemi in questione, il numero massimo di dispositivo richiamato nel listato programma e comunica al sistema operativo della Logica tali informazioni; durante l'aggiornamento delle risorse il sistema operativo automaticamente esclude l'elaborazione per tutti i dispositivi con numero identificatore superiore a questi massimi. E' buona regola quindi compattare il più possibile i numeri identificatori delle schede SLAVES, dei contatori e dei generatori d'impulso, verso i numeri più bassi possibile.

Per quanto riguarda l'eventuale utilizzo di istruzioni di tipo matematico, funzioni di conversione e altre funzioni di elaborazione su bytes o aree di bytes, è necessario prevedere un dispositivo generatore d'impulso per la loro alimentazione o un bit di memoria che resti abilitato per un solo ciclo. Infatti queste istruzioni richiedono tempi di elaborazione considerevolmente più lunghi di quelli delle istruzioni booleane; per questo è conveniente alimentarle per un solo ciclo di scansione. Si consideri che tali istruzioni sono tutte di tipo finale e quindi da abilitare con un'opportuno bit; esse non alterano il valore del bit presente nell'accumulatore e lo stesso bit può essere utilizzato anche per abilitare una successione di più istruzioni.

Si raccomanda inoltre di fare ampio uso di istruzioni di salto blocchi (JMP e JME) e di salto a Label (GOTO) allo scopo di escludere dall'elaborazione ciclica le parti di programma momentaneamente non necessarie. Per esempio, nel caso il programma preveda più modalità operative di lavorazione selezionabili dall'operatore, non è necessario elaborare le istruzioni di tutte le modalità, ma è conveniente scavalcare con un'istruzione di salto le parti non selezionate.

Anche un uso metodico di subroutines all'interno del programma consente di migliorare notevolmente le prestazioni dello stesso; inoltre l'utilizzo di istruzioni indicizzate permette di eseguire operazioni su insiemi di dati in modo sicuramente più efficiente della ripetizione delle stesse istruzioni per ogni singolo dato.

Tabelle riassuntive del linguaggio ICL51

Nelle successive pagine vengono riportate le tabelle riassuntive del linguaggio ICL51.

Nella Tabella 3 sono elencate tutte le possibili risorse esterne ed interne del sistema, con le relative terminologie di identificazione.

La Tabella 4 riportata la “mappatura” della memoria RAM dati del MASTER, nella quale sono collocate tutte le risorse di cui sopra.

Nella Tabella 5 è riportato un elenco completo di tutte le istruzioni riconosciute dal linguaggio, con la sintassi di utilizzo ed una sommaria descrizione. Per ciascuna istruzione vengono forniti i campi di validità degli operandi con riferimento ai gruppi illustrati nella Tabella 6.

Infine la Tabella 7 offre un quadro sintetico delle prestazioni delle singole istruzioni; sono riportati i bytes occupati nel codice programma ed il tempo di esecuzione di ciascuna istruzione. Queste prestazioni dipendono dalla particolare situazione di utilizzo delle istruzioni stesse e per questo occorre verificare le corrispondenti note.

Si tenga presente che le istruzioni elencate sono solo quelle base del linguaggio e quindi riconosciute sempre dal compilatore; sono cioè escluse tutte le eventuali istruzioni aggiunte al compilatore mediante la creazione di routines esterne in linguaggio assembler.

OPERANDO = CAMPO1.[CAMPO2].[CAMPO3]

CAMPO1	CAMPO2	CAMPO3
RISORSE ESTERNE		
0 – 31 NUMERO DI SCHEDA	0 – 127 NUMERO DI BYTE	0 – 7 NUMERO DI BIT
RISORSE INTERNE		
M MEMORIA NON RITENTIVA	0 – 1023 NUMERO DI BYTE	0 – 7 NUMERO DI BIT
H MEMORIA RITENTIVA	0 – 1023 NUMERO DI BYTE	0 – 7 NUMERO DI BIT
C CONTATORE 16 BIT	0 – 127 NUMERO DI DISPOSITIVO	IN BIT DI START CONTEGGIO OUT BIT DI FINE CONTEGGIO CKUP BIT DI CONTEGGIO UP CKDW BIT DI CONTEGGIO DOWN CB BYTE DI CONTROLLO CL BYTE LOW VALORE CORRENTE CH BYTE HIGH VALORE CORRENTE FL BYTE LOW VALORE FINALE FH BYTE HIGH VALORE FINALE TIPO DI BIT / BYTE
P GENERATORE IMPULSO	0 – 127 NUMERO DI DISPOSITIVO	IN BIT DI INGRESSO FRONTE OUTU BIT DI USCITA FRONTE UP OUTD BIT DI USCITA FRONTE DOWN TIPO DI BIT

Tabella 3.a. Le risorse del sistema

CAMPO1	CAMPO2	CAMPO3
RISORSE INTERNE		
T TEMPO DI RIFERIMENTO	50 PERIODO 50 ms 100 PERIODO 100 ms 200 PERIODO 200 ms 500 PERIODO 500 ms 1000 PERIODO 1 s 2000 PERIODO 2 s TIPO DI BIT	
F FLAG SPECIALI	0 BIT SEMPRE OFF 1 BIT SEMPRE ON P IMPULSO POWER ON < RISULTATO < = RISULTATO = > RISULTATO > C BIT DI CARRY E BIT DI ERRORE TIPO DI BIT	
K COSTANTE	$-2^{31} - 2^{32}-1$ DEC. 0H – F...FH ESA. 0B – 1...1B BIN. VALORE DI COSTANTE	
SXS CONTATORE SCAN X SEC		
W OROLOGIO/CALENDARIO	CB BYTE CONTROLLO SEC SECONDI MIN MINUTI HOUR ORE DAY GIORNO SETTIM. DATE GIORNO MESE MTH MESE YEAR ANNO TIPO DI BYTE	ADJ ADJUSTMENT TIPO DI BIT
X MEMORIA RITENTIVA ESTESA	0 – 24567 NUMERO DI BYTE	0 – 7 NUMERO DI BIT

ATTENZIONE: LA MEMORIA RITENTIVA ESTESA E' DISPONIBILE SOLO SU ALCUNE LOGICHE

Tabella 3.b. Le risorse del sistema

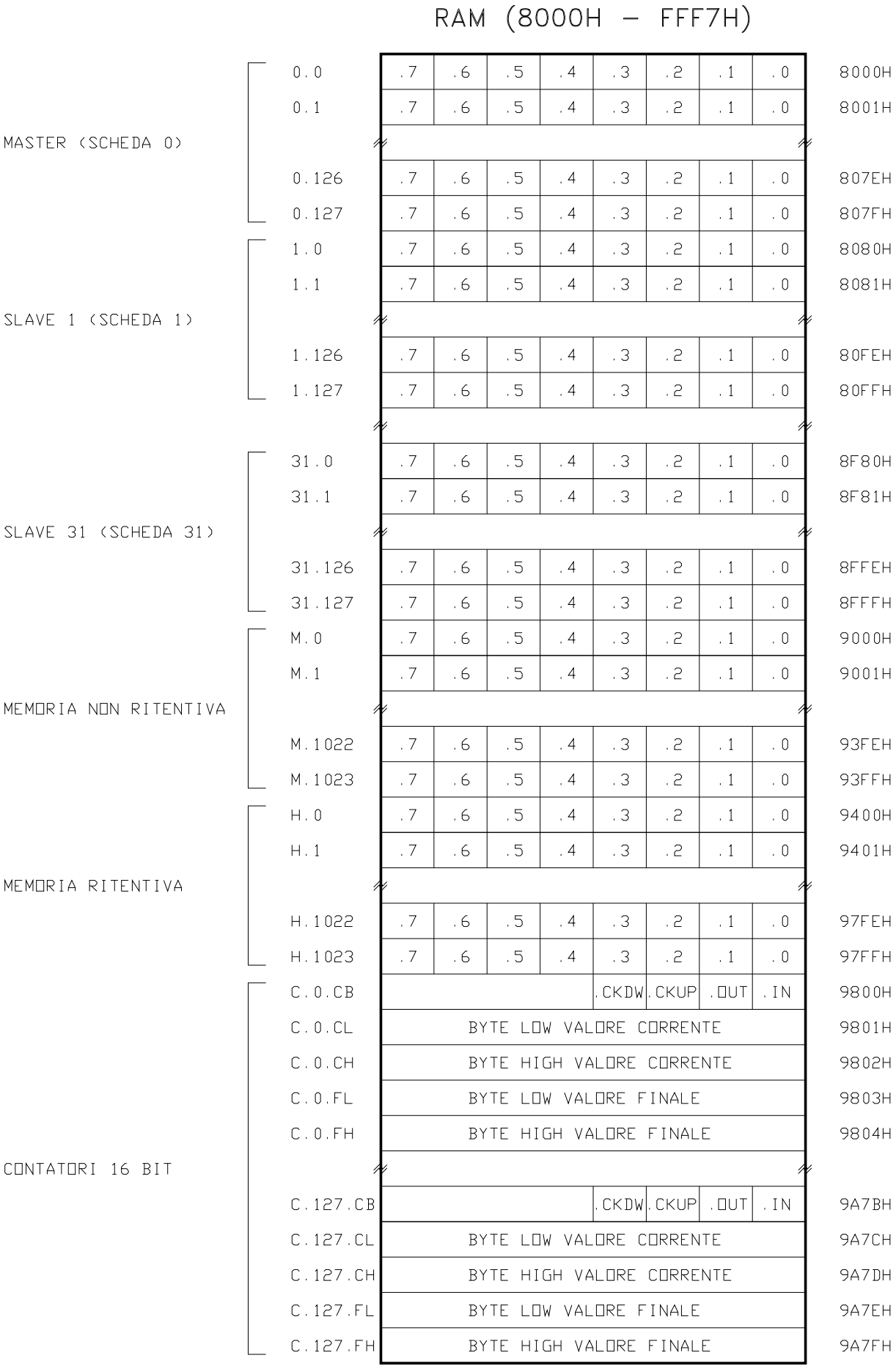


Tabella 4.a. Mappatura della RAM dati

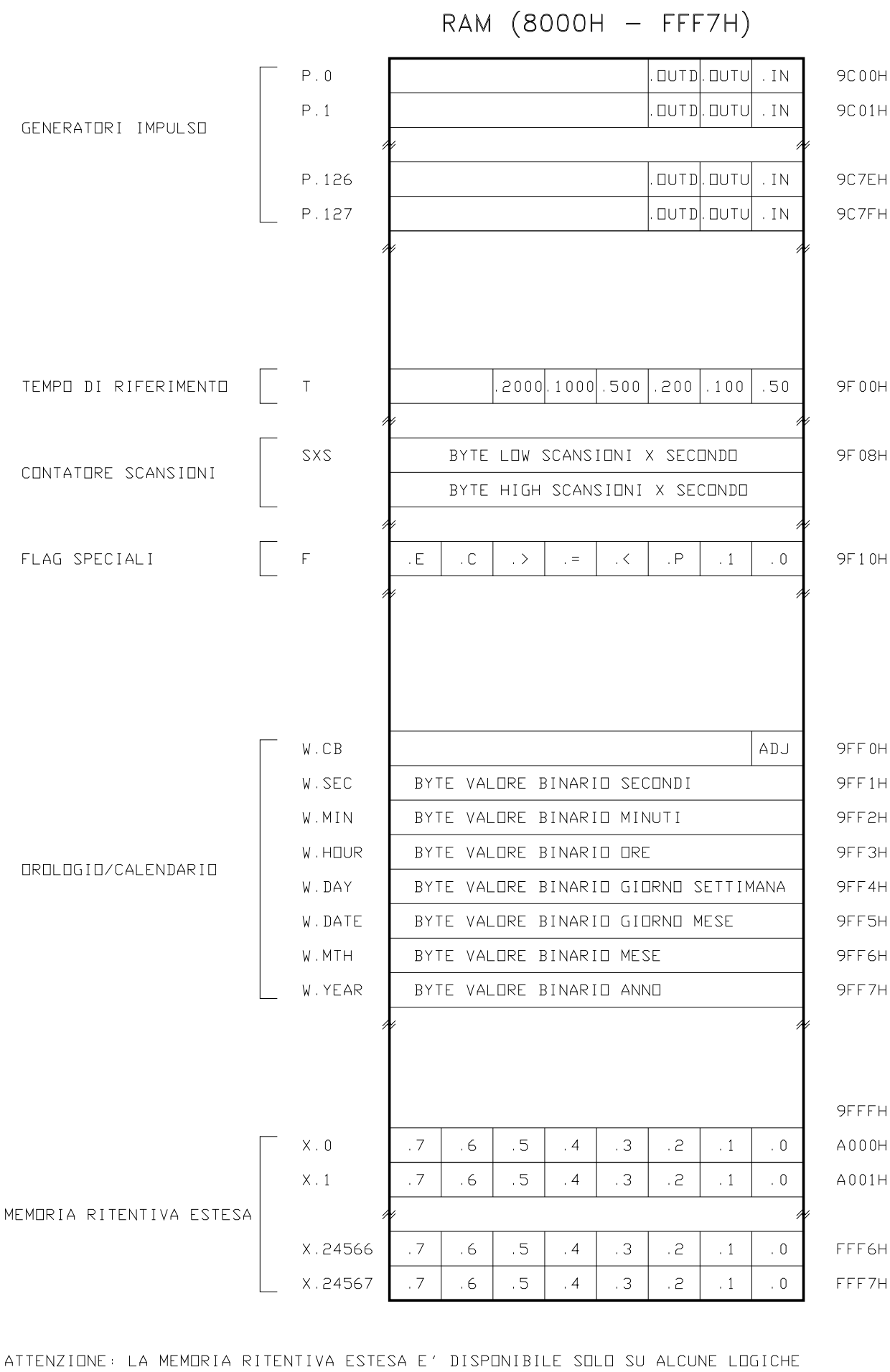


Tabella 4.b. Mappatura della RAM dati

COMPOSIZIONE DEL LISTATO PROGRAMMA:

PROGRAMMA = RIGA 1
RIGA 2
RIGA 3
.....
.....
RIGA n

RIGA = [“COMMENTO_MEMORIA]

RIGA = [LABEL_OPERANDO = OPERANDO] [‘COMMENTO]

RIGA = [LABEL_SALTO:] [‘COMMENTO]

RIGA = [ISTRUZIONE] [‘COMMENTO]

ISTRUZIONE = OPERAZIONE [OPERANDO 1] [OPERANDO 2] [OPERANDO 3]

OPERAZIONE	OPERANDO 1 (GRUPPI)	OPERANDO 2	OPERANDO 3	FUNZIONE
LD	BIT (qualunque)			Caricamento del primo contatto N.O. del ramo
LDNOT	BIT (qualunque)			Caricamento del primo contatto N.C. del ramo
AND	BIT (qualunque)			Serie del ramo con un contatto N.O.
ANDNOT	BIT (qualunque)			Serie del ramo con un contatto N.C.
OR	BIT (qualunque)			Parallelo del ramo con un contatto N.O.
ORNOT	BIT (qualunque)			Parallelo del ramo con un contatto N.C.
ANDLD				Serie di due rami
ORLD				Parallelo di due rami
OUT	BIT (1,2,3,4)			Alimentazione della bobina di un relè
OUTNOT	BIT (1,2,3,4)			Alimentazione inversa della bobina di un relè

Tabella 5.a. Quadro riassuntivo delle istruzioni

OPERAZIONE	OPERANDO 1 (GRUPPI)	OPERANDO 2 (GRUPPI)	OPERANDO 3 (GRUPPI)	FUNZIONE
SET	BIT (1,2,3,4)			Alimentazione con autoritenuta della bobina di un relè (*)
RES	BIT (1,2,3,4)			Caduta della autoritenuta della bobina di un relè (*)
CPL	BIT (1,2,3,4)			Inversione dello stato logico della bobina di un relè (*)
JMP				Salto alla prossima JME (*)
JME				Fine salto dalla JMP
GOTO	LABEL (max 32 caratt.)			Salto alla LABEL: (*)
GOSUB	LABEL (max 32 caratt.)			Esecuzione subroutine compresa tra LABEL: ed END (*)
NOP				Istruzione nulla
END				Fine della lista istruzioni del programma o subroutine
TIM	C.n.IN (n=0-127)	BYTE (5,6,7,8,11)		Gestione temporizzatore con base tempi 0.1"
CNT	C.n.IN (n=0-127)	BIT (qualunque)	BYTE (5,6,7,8,11)	Gestione contatore UP
SFR	BYTE (5,6,7,8,9)			Shift a sinistra del byte; entrata del F.C a destra con uscita a sinistra sul F.C (*)

*** = istruzione eseguita solo se il ramo precedente é ON (alimenta l'istruzione).**

Tabella 5.b. Quadro riassuntivo delle istruzioni

OPERAZIONE	OPERANDO 1 (GRUPPI)	OPERANDO 2 (GRUPPI)	OPERANDO 3 (GRUPPI)	FUNZIONE
ANDB	BYTE1 (5,6,7,8,9)	BYTE2 (5,6,7,8,9,10)	BYTE3 (5,6,7,8,9,10)	BYTE1 <-- BYTE2 AND BYTE3 Prodotto logico dei singoli bits delle variabili a 1 byte (*)
ORB	BYTE1 (5,6,7,8,9)	BYTE2 (5,6,7,8,9,10)	BYTE3 (5,6,7,8,9,10)	BYTE1 <-- BYTE2 OR BYTE3 Somma logica dei singoli bits delle variabili a 1 byte (*)
XORB	BYTE1 (5,6,7,8,9)	BYTE2 (5,6,7,8,9,10)	BYTE3 (5,6,7,8,9,10)	BYTE1 <-- BYTE2 XOR BYTE3 Somma esclusiva dei singoli bits delle variabili a 1 byte (*)
CPLB	BYTE (5,6,7,8,9)			BYTE <-- NOT BYTE Complementazione dei singoli bits delle variabili a 1 byte (*)
MOV1	BYTE1 (5,6,7,8,9)	BYTE2 (5,6,7,8,9,10)		BYTE1 <-- BYTE2 Copia di una variabile a 1 byte su un'altra (*)
MOV2	BYTE1 (5,6,7,8)	BYTE2 (5,6,7,8,11)		BYTE1 <-- BYTE2 Copia di una variabile a 2 bytes su un'altra (*)
MOV4	BYTE1 (5,6,7)	BYTE2 (5,6,7,12)		BYTE1 <-- BYTE2 Copia di una variabile a 4 bytes su un'altra (*)
CMP1	BYTE1 (5,6,7,8,9,10)	BYTE2 (5,6,7,8,9,10)		BYTE1 = BYTE2 ? Comparazione tra due variabili a 1 byte; risultato in F.< F.= F.> (*)
CMP2	BYTE1 (5,6,7,8,11)	BYTE2 (5,6,7,8,11)		BYTE1 = BYTE2 ? Comparazione tra due variabili a 2 bytes; risultato in F.< F.= F.> (*)
CMP4	BYTE1 (5,6,7,12)	BYTE2 (5,6,7,12)		BYTE1 = BYTE2 ? Comparazione tra due variabili a 4 bytes; risultato in F.< F.= F.> (*)

*** = istruzione eseguita solo se il ramo precedente é ON (alimenta l'istruzione).**

Tabella 5.c. Quadro riassuntivo delle istruzioni

OPERAZIONE	OPERANDO 1 (GRUPPI)	OPERANDO 2 (GRUPPI)	OPERANDO 3 (GRUPPI)	FUNZIONE
ADD1	BYTE1 (5,6,7,8,9)	BYTE2 (5,6,7,8,9,10)	BYTE3 (5,6,7,8,9,10)	BYTE1 <-- BYTE2 + BYTE3 Addizione di variabili a 1 byte; overflow in F.C (*)
ADD2	BYTE1 (5,6,7,8)	BYTE2 (5,6,7,8,11)	BYTE3 (5,6,7,8,11)	BYTE1 <-- BYTE2 + BYTE3 Addizione di variabili a 2 bytes; overflow in F.C (*)
ADD4	BYTE1 (5,6,7)	BYTE2 (5,6,7,12)	BYTE3 (5,6,7,12)	BYTE1 <-- BYTE2 + BYTE3 Addizione di variabili a 4 bytes; overflow in F.C (*)
SUB1	BYTE1 (5,6,7,8,9)	BYTE2 (5,6,7,8,9,10)	BYTE3 (5,6,7,8,9,10)	BYTE1 <-- BYTE2 - BYTE3 Sottrazione di variabili a 1 byte; borrow in F.C (*)
SUB2	BYTE1 (5,6,7,8)	BYTE2 (5,6,7,8,11)	BYTE3 (5,6,7,8,11)	BYTE1 <-- BYTE2 - BYTE3 Sottrazione di variabili a 2 bytes; borrow in F.C (*)
SUB4	BYTE1 (5,6,7)	BYTE2 (5,6,7,12)	BYTE3 (5,6,7,12)	BYTE1 <-- BYTE2 - BYTE3 Sottrazione di variabili a 4 bytes; borrow in F.C (*)
MUL1	BYTE1 (5,6,7,8,9) (2 bytes)	BYTE2 (5,6,7,8,9,10)	BYTE3 (5,6,7,8,9,10)	BYTE1 <-- BYTE2 * BYTE3 Moltiplicazione di variabili a 1 byte; overflow in F.E (*)
MUL2	BYTE1 (5,6,7,8) (4 bytes)	BYTE2 (5,6,7,8,11)	BYTE3 (5,6,7,8,11)	BYTE1 <-- BYTE2 * BYTE3 Moltiplicazione di variabili a 2 bytes; overflow in F.E (*)
MUL4	BYTE1 (5,6,7) (8 bytes)	BYTE2 (5,6,7,12)	BYTE3 (5,6,7,12)	BYTE1 <-- BYTE2 * BYTE3 Moltiplicazione di variabili a 4 bytes; overflow in F.E (*)

*** = istruzione eseguita solo se il ramo precedente é ON (alimenta l'istruzione).**

Tabella 5.d. Quadro riassuntivo delle istruzioni

OPERAZIONE	OPERANDO 1 (GRUPPI)	OPERANDO 2 (GRUPPI)	OPERANDO 3 (GRUPPI)	FUNZIONE
DIV1	BYTE1 (5,6,7,8,9) (2 bytes)	BYTE2 (5,6,7,8,9,10)	BYTE3 (5,6,7,8,9,10)	BYTE1 <-- BYTE2 / BYTE3 Divisione di variabili a 1 byte; divisore = 0 in F.E (*)
DIV2	BYTE1 (5,6,7,8) (4 bytes)	BYTE2 (5,6,7,8,11)	BYTE3 (5,6,7,8,11)	BYTE1 <-- BYTE2 / BYTE3 Divisione di variabili a 2 bytes; divisore = 0 in F.E (*)
DIV4	BYTE1 (5,6,7) (8 bytes)	BYTE2 (5,6,7,12)	BYTE3 (5,6,7,12)	BYTE1 <-- BYTE2 / BYTE3 Divisione di variabili a 4 bytes; divisore = 0 in F.E (*)
INC1	BYTE (5,6,7,8,9)			BYTE <-- BYTE + 1 Incremento di variabili a 1 byte; overflow in F.C (*)
INC2	BYTE (5,6,7,8)			BYTE <-- BYTE + 1 Incremento di variabili a 2 bytes; overflow in F.C (*)
INC4	BYTE (5,6,7)			BYTE <-- BYTE + 1 Incremento di variabili a 4 bytes; overflow in F.C (*)
DEC1	BYTE (5,6,7,8,9)			BYTE <-- BYTE - 1 Decremento di variabili a 1 byte; borrow in F.C (*)
DEC2	BYTE (5,6,7,8)			BYTE <-- BYTE - 1 Decremento di variabili a 2 bytes; borrow in F.C (*)
DEC4	BYTE (5,6,7)			BYTE <-- BYTE - 1 Decremento di variabili a 4 bytes; borrow in F.C (*)

*** = istruzione eseguita solo se il ramo precedente é ON (alimenta l'istruzione).**

Tabella 5.e. Quadro riassuntivo delle istruzioni

OPERAZIONE	OPERANDO 1 (GRUPPI)	OPERANDO 2 (GRUPPI)	OPERANDO 3 (GRUPPI)	FUNZIONE
ABS1	BYTE1 (5,6,7,8,9)	BYTE2 (5,6,7,8,9,10)		BYTE1 <-- |BYTE2| Valore assoluto variabile 1 byte; negativa complementa F.C (*)
ABS2	BYTE1 (5,6,7,8)	BYTE2 (5,6,7,8,11)		BYTE1 <-- |BYTE2| Valore assoluto variabile 2 bytes; negativa complementa F.C (*)
ABS4	BYTE1 (5,6,7)	BYTE2 (5,6,7,12)		BYTE1 <-- |BYTE2| Valore assoluto variabile 4 bytes; negativa complementa F.C (*)
NEG1	BYTE (5,6,7,8,9)			BYTE <-- -BYTE Negazione (complemento a 2) di variabile a 1 byte; (*)
NEG2	BYTE (5,6,7,8)			BYTE <-- -BYTE Negazione (complemento a 2) di variabile a 2 bytes; (*)
NEG4	BYTE (5,6,7)			BYTE <-- -BYTE Negazione (complemento a 2) di variabile a 4 bytes; (*)
BINBCD1	BYTE1 (5,6,7,8,9)	BYTE2 (5,6,7,8,9,10)		BYTE1 (BCD) <-- BYTE2 (BIN) Conversione BIN-BCD a 1 byte (*)
BINBCD2	BYTE1 (5,6,7,8)	BYTE2 (5,6,7,8,11)		BYTE1 (BCD) <-- BYTE2 (BIN) Conversione BIN-BCD a 2 bytes (*)
BINBCD4	BYTE1 (5,6,7)	BYTE2 (5,6,7,12)		BYTE1 (BCD) <-- BYTE2 (BIN) Conversione BIN-BCD a 4 bytes (*)

*** = istruzione eseguita solo se il ramo precedente é ON (alimenta l'istruzione).**

Tabella 5.f. Quadro riassuntivo delle istruzioni

OPERAZIONE	OPERANDO 1 (GRUPPI)	OPERANDO 2 (GRUPPI)	OPERANDO 3 (GRUPPI)	FUNZIONE
BCDBIN1	BYTE1 (5,6,7,8,9)	BYTE2 (5,6,7,8,9,10)		BYTE1 (BIN) <-- BYTE2 (BCD) Conversione BCD-BIN a 1 byte (*)
BCDBIN2	BYTE1 (5,6,7,8)	BYTE2 (5,6,7,8,11)		BYTE1 (BIN) <-- BYTE2 (BCD) Conversione BCD-BIN a 2 bytes (*)
BCDBIN4	BYTE1 (5,6,7)	BYTE2 (5,6,7,12)		BYTE1 (BIN) <-- BYTE2 (BCD) Conversione BCD-BIN a 4 bytes (*)
SWAP	BYTE (5,6,7,8,9)			BYTE.7-4 <---> BYTE.3-0 Scambio in un byte del nibble alto con il nibble basso (*)
RCL1	BYTE (5,6,7,8,9,10)			CATASTA(0) <-- BYTE Carica in catasta una variabile ad 8 bit con segno (*)
RCL2	BYTE (5,6,7,8,11)			CATASTA(0) <-- BYTE Carica in catasta una variabile a 16 bit con segno (*)
RCL4	BYTE (5,6,7,12)			CATASTA(0) <-- BYTE Carica in catasta una variabile a 32 bit con segno (*)
STO1	BYTE (5,6,7,8,9)			BYTE <-- CATASTA(0) Copia catasta(0) su variabile ad 8 bit con segno; errore in F.E (*)
STO2	BYTE (5,6,7,8)			BYTE <-- CATASTA(0) Copia catasta(0) su variabile a 16 bit con segno; errore in F.E (*)
STO4	BYTE (5,6,7)			BYTE <-- CATASTA(0) Copia catasta(0) su variabile a 32 bit con segno; errore in F.E (*)

*** = istruzione eseguita solo se il ramo precedente é ON (alimenta l'istruzione).**

Tabella 5.g. Quadro riassuntivo delle istruzioni

OPERAZIONE	OPERANDO 1 (GRUPPI)	OPERANDO 2 (GRUPPI)	OPERANDO 3 (GRUPPI)	FUNZIONE
ADD				CAT(0) <-- CAT(1) + CAT(0) Somma in catasta a 32 bit con segno; overflow in F.E (*)
SUB				CAT(0) <-- CAT(1) - CAT(0) Sottrazione in catasta a 32 bit con segno; borrow in F.E (*)
MUL				CAT(0) <-- CAT(1) * CAT(0) Moltiplicazione in catasta a 32 bit con segno; overflow in F.E (*)
DIV				CAT(0) <-- CAT(1) / CAT(0) Divisione in catasta a 32 bit con segno; divisore = 0 in F.E (*)
CMP				CAT(1) = CAT(0) ? Comparazione catasta a 32 bit con segno; risultato in F.< F.= F.> (*)
MOVADD	BYTE1 (5,6,7,8) (2 bytes)	BYTE2 (5,6,7,8,9)		BYTE1 <-- ADDRESS(BYTE2) Caricamento indirizzo assoluto su variabile a 2 bytes (*)
MOVASC	BYTE (5,6,7,8,9)	|STRINGA| (max 100 caratt.)		BYTE1 <-- ASC(STRINGA) Riempimento di un'area memoria con i codici ASCII di una stringa (*)
MOVBLK	BYTE1 (5,6,7,8,9)	BYTE2 (5,6,7,8,9)	BYTE3 (5,6,7,8,9,10) (max 255)	BYTE1..... <-- BYTE2..... Copia di un'area di BYTE3 bytes su altra area (*)
RESMEM	BYTE1 (5,6,7,8,9)	BYTE2 (5,6,7,8,9,10) (max 255)		BYTE1..... <-- K.0 Azzeramento di un'area di BYTE2 bytes (*)

*** = istruzione eseguita solo se il ramo precedente é ON (alimenta l'istruzione).**

Tabella 5.h. Quadro riassuntivo delle istruzioni

OPERAZIONE	OPERANDO 1 (GRUPPI)	OPERANDO 2 (GRUPPI)	OPERANDO 3 (GRUPPI)	FUNZIONE
IOREFR				Aggiornamento forzato delle risorse esterne locali (I/O MASTER) (*)
RESWD				Reset forzato del timer di Watch-Dog (*)
INCLUDE	STRINGA (max 8 caratteri)			Inclusione di righe programma da file esterno
PASSW	STRINGA (max 8 caratteri)			Dichiarazione della password di programma

*** = istruzione eseguita solo se il ramo precedente é ON (alimenta l'istruzione).**

Tabella 5.i. Quadro riassuntivo delle istruzioni

GRUPPO	TIPO	ELENCO
1	BIT	0.0.0 - 31.127.7
2	BIT	M.0.0 - M.1023.7
3	BIT	H.0.0 - H.1023.7 X.0.0 - X.24567.7
4	BIT	F.C F.E F.< F.= F.> C.*.IN C.*.CKUP C.*.CKDW P.*.IN
5	BYTE	0.0 - 31.127
6	BYTE	M.0 - M.1023
7	BYTE	H.0 - H.1023 X.0 - X.24567
8	BYTE	C.*.FL C.*.CL SXS
9	BYTE	C.*.FH C.*.CH W
10	BYTE	COSTANTE MAX 1 BYTE
11	BYTE	COSTANTE MAX 2 BYTES
12	BYTE	COSTANTE MAX 4 BYTES

Tabella 6. Gruppi di validità degli operandi

ISTRUZIONE	BYTES	NOTE	TEMPO (µS)	NOTE
LD	6/2/8/4	A	5/1/7/3	A
LDNOT	7/3/9/5	A	6/2/8/4	A
AND	6/2	B	6/2	B
ANDNOT	6/2	B	6/2	B
OR	6/2	B	6/2	B
ORNOT	6/2	B	6/2	B
ANDLD	2		2	
ORLD	2		2	
OUT	7/3	B	8/4	B
OUTNOT	9/5	B	9/5	B
SET	9/5	B	2-9/2-5	B,C
RES	9/5	B	2-9/2-5	B,C
CPL	9/5	B	2-9/2-5	B,C
JMP	5		2-4	C
JME	0		0	
GOTO	5		2-4	C
GOSUB	5		2-4	C
NOP	1		1	
END	1		2	

Tabella 7.a. Prestazioni delle istruzioni

ISTRUZIONE	BYTES	NOTE	TEMPO (µS)	NOTE
TIM	45		37-127-110	D
CNT	45		37-127-110	D
SFR	8		2-38	C
ANDB	17		2-19	C
ORB	17		2-19	C
XORB	17		2-19	C
CPLB	17		2-19	C
MOV1	11		2-13	C
MOV2	14		2-20	C
MOV4	20		2-34	C
CMP1	14		2-29	C
CMP2	21		2-36	C
CMP4	35		2-59	C
ADD1	17		2-29	C
ADD2	24		2-42	C
ADD4	38		2-68	C
SUB1	17		2-30	C
SUB2	24		2-43	C
SUB4	38		2-69	C
MUL1	17		2-42	C
MUL2	24		2-110	C
MUL4	38		2-342	C
DIV1	17		2-45	C
DIV2	24		2-240	C
DIV4	38		2-773	C

Tabella 7.b. Prestazioni delle istruzioni

ISTRUZIONE	BYTES	NOTE	TEMPO (µS)	NOTE
INC1	8		2-24	C
INC2	8		2-30	C
INC4	8		2-45	C
DEC1	8		2-24	C
DEC2	8		2-32	C
DEC4	8		2-47	C
ABS1	11		2-17-26	E
ABS2	14		2-28-43	E
ABS4	20		2-49-67	E
NEG1	8		2-14	C
NEG2	8		2-24	C
NEG4	8		2-40	C
BINBCD1	11		2-89	C
BINBCD2	14		2-248	C
BINBCD4	20		2-743	C
BCDBIN1	11		2-27	C
BCDBIN2	14		2-60	C
BCDBIN4	20		2-182	C
SWAP	8		2-9	C
RCL1	11		2-45	C
RCL2	15		2-49	C
RCL4	23		2-57	C
STO1	8		2-24	C
STO2	8		2-27	C
STO4	8		2-25	C
ADD	5		2-38	C
SUB	5		2-39	C
MUL	5		2-185	C
DIV	5		2-895	C
CMP	5		2-22-34	F

Tabella 7.c. Prestazioni delle istruzioni

ISTRUZIONE	BYTES	NOTE	TEMPO (µS)	NOTE
MOVADD	12		2-11	C
MOVASC	18+Ncar		2 - [8+10Ncar]	C
MOVBLK	17		2 - [20+27Nbyte]	C
CMPBLK	17		2 - [28+30NByte]	C
RESMEM	11		2 - [14+8Nbyte]	C
IOREFR	5		2- vedere manuale logica	C
RESWD	5		2-12	C
INCLUDE	0		0	
PASSW	0		0	

Tabella 7.d. Prestazioni delle istruzioni

NOTE:

A: PRIMA_DEL_RAMO E NUOVO_BYTE /
PRIMA_DEL_RAMO E STESSO_BYTE /
SUCCESSIVA_DEL_RAMO E NUOVO_BYTE /
SUCCESSIVA_DEL_RAMO E STESSO_BYTE

B: NUOVO_BYTE / STESSO_BYTE

C: NON_ESEGUITA - ESEGUITA

D: IN=0 - IN=1_CONTEGGIO - IN=1_FINE_CONTEGGIO

E: NON_ESEGUITA - POSITIVO - NEGATIVO

F: NON_ESEGUITA - SEGNO_DIVERSO - STESSO_SEGNO

Le istruzioni esterne

Aggiungere istruzioni personalizzate

Il linguaggio ICL51 offre al Programmatore la possibilità di crearsi nuove istruzioni personalizzate, da aggiungere a quelle base già disponibili, così come se fosse stata realizzata una versione del software perfettamente rispondente alle sue esigenze specifiche.

Si premette che per aggiungere nuove istruzioni occorre una certa esperienza nella programmazione in LINGUAGGIO ASSEMBLER dei microprocessori della famiglia INTEL® 80C51. Inoltre occorre dotarsi di un programma ASSEMBLATORE di tipo commerciale per tale linguaggio, capace di generare dei files eseguibili in FORMATO BINARIO.

La tecnica utilizzata per fare in modo che il compilatore ICL51 riconosca valide anche le nuove istruzioni, consiste nel creare, per ogni nuova istruzione, un file oggetto il cui nome (8 caratteri massimi), coincida con il nome dell'istruzione e l'estensione sia necessariamente .IOF (Istruction Object File).

I files così creati vanno poi collocati nella directory corrente di lavoro; quando il compilatore incontra nel listato programma un'istruzione che non sia riconosciuta come base, prima di dichiarare errore, ricerca nella directory corrente un file con il nome uguale a quello dell'istruzione ed estensione .IOF. Se viene trovato tale file, il codice memorizzato in questo viene utilizzato come codice di compilazione dell'istruzione.

Al pari delle istruzioni base, le nuove istruzioni creabili possono avere degli opportuni operandi, passati con la stessa identica sintassi. Questo consente di trattare le istruzioni personalizzate come se fossero di base: nessuno potrebbe sospettare che le nuove istruzioni siano state appositamente realizzate ed aggiunte esternamente al compilatore.

Passaggio degli operandi all'istruzione esterna

Il compilatore gestisce sostanzialmente 5 diverse modalità di passaggio dei parametri alla routine esterna assembler: queste modalità ricalcano ciò che succede per gran parte delle istruzioni base.

In particolare, indicando genericamente con NOME il nome scelto per la nuova istruzione, esse sono:

```
NOME1    byte1    byte2    byte3

NOME2    byte1    byte2    byte3

NOME4    byte1    byte2    byte3

NOME     byte

NOME
```

Le prime tre permettono di trasferire all'istruzione esterna due operandi (byte2 e byte3) che possono essere rispettivamente di dimensione 1, 2, 4 bytes e di tipo sia variabile che costante.

Il valore effettivo di queste due variabili, che in genere costituiscono i due operandi di ingresso di un'istruzione, vengono precaricati in alcuni registri interni del microprocessore, prima della chiamata al codice esterno; il compilatore richiede obbligatoriamente che il nome dell'istruzione termini con il carattere 1 oppure 2 oppure 4, in modo da distinguere se passare degli operandi ad 1, 2, 4 bytes. I registri di passaggio di tali operandi sono i seguenti:

```
01CH <— byte2(0)
01DH <— byte2(1)      (solo NOME2 e NOME4)
01EH <— byte2(2)      (solo NOME4)
01FH <— byte2(3)      (solo NOME4)

018H <— byte3(0)
019H <— byte3(1)      (solo NOME2 e NOME4)
01AH <— byte3(2)      (solo NOME4)
01BH <— byte3(3)      (solo NOME4)
```

Per quanto riguarda l'operando byte1, in genere operando di uscita dell'istruzione, viene passato esclusivamente l'indirizzo assoluto della sua locazione nella RAM dati (o del byte di peso inferiore per NOME2 e NOME4); questo indirizzo viene precaricato nel registro puntatore a 16 bit noto come DPTR (riferirsi alla documentazione specializzata dei microprocessori della famiglia 80C51).

La quarta modalità di trasferimento parametri consente di passare l'indirizzo assoluto dell'operando byte nel registro puntatore DPTR, allo stesso modo delle precedenti modalità; inoltre, nel caso l'operando fosse di tipo bit, oltre al caricamento dell'indirizzo assoluto di byte nel DPTR, viene caricato nel registro B la maschera (un solo bit a 1) che individua la posizione nel byte del bit in questione.

Infine la quinta modalità non consente il passaggio di parametri, ma solo di eseguire il codice presente nel file esterno.

Per concludere la descrizione del passaggio parametri possiamo dire, a titolo di esempio, che le prime tre modalità equivalgono a ciò che avviene per un'istruzione rispettivamente del tipo ADD1, ADD2, ADD4, la quarta modalità a ciò che avviene per l'istruzione SFR ed infine la quinta per l'istruzione ANDLD.

Riportiamo infine gli indirizzi assoluti della memoria interna del microprocessore utilizzata per la catasta operativa e gestita dalle istruzioni di valutazione delle equazioni secondo la notazione polacca inversa:

004H <— byte(0)	CATASTA(3)
005H <— byte(1)	
006H <— byte(2)	
007H <— byte(3)	
008H <— byte(0)	CATASTA(2)
009H <— byte(1)	
00AH <— byte(2)	
00BH <— byte(3)	
00CH <— byte(0)	CATASTA(1)
00DH <— byte(1)	
00EH <— byte(2)	
00FH <— byte(3)	
010H <— byte(0)	CATASTA(0)
011H <— byte(1)	
012H <— byte(2)	
013H <— byte(3)	

Come creare un'istruzione esterna

Per creare un'istruzione esterna occorre, come già detto in precedenza, editare un file in linguaggio assembler 80C51. Successivamente occorre assemblare tale sorgente per generare un file binario contente il linguaggio macchina del microprocessore e cambiare l'estensione di tale file in .IOF.

Quando il compilatore ICL51 incontrerà un'istruzione definita esternamente, provvederà in modo automatico a generare il codice corrispondente al precaricamento degli operandi dell'istruzione stessa e successivamente accoderà a tale codice il contenuto del file .IOF.

Le istruzioni esterne che richiedono passaggio di parametri dovranno elaborare i valori passati nei registri e posizionare il risultato nella RAM dati utilizzando per esempio l'indirizzo presente nel DPTR.

Il codice scritto in assembler viene radicalmente copiato nel codice totale di programma utente (file .OBJ), per cui è indispensabile che esso termini sempre la sua esecuzione senza loop infiniti, altrimenti il circuito di watch-dog della Logica interverrà forzando un reset.

La struttura del programma assembler deve quindi essere "aperta", ossia deve consentire il proseguimento dell'esecuzione di tutte le altre istruzioni del programma utente. Inoltre il codice generato non deve in alcun modo essere legato a particolari indirizzi assoluti propri dello stesso programma: gli eventuali salti di programma devono essere di tipo relativo poiché non è prevedibile a quale indirizzo assoluto il compilatore collocherà il codice. Si consiglia dunque di iniziare il programma assembler con la direttiva:

```
ORG  0000H
```

Facciamo ora alcune considerazioni su un altro punto chiave della programmazione assembler. I rami della rete elettromeccanica sono calcolati mediante l'ausilio di una pila di bits di tipo LIFO (Last In First Out): in questa pila vengono appoggiati temporaneamente tutti i bits intermedi del calcolo booleano della rete.

In dettaglio la pila è costituita da 8 bits; il bit di livello 0, corrispondente alla posizione di ingresso e uscita della pila, coincide con lo stesso bit di carry C del microprocessore.

La gestione di questa pila è fatta dallo stesso compilatore in modo automatico; in particolare un puntatore di posizione, inizializzato sul bit 71H, permette di archiviare temporaneamente il valore calcolato nel carry C ogni volta che una nuova istruzione LD o LDNOT viene incontrata.

Se il ramo contiene una sola istruzione LD o LDNOT il solo carry C è sufficiente al calcolo, altrimenti i risultati intermedi sono archiviati successivamente dal bit 71H al bit 77H della memoria interna del microprocessore. Le istruzioni di ANDLD e ORLD fanno scendere di un livello la pila realizzando l'operazione tra il bit di carry C e l'ultimo risultato intermedio archiviato nel bit 71H.

Nel caso l'esecuzione dell'istruzione esterna necessiti del bit di carry C per sue esigenze, esso può essere temporaneamente salvato nel bit 70H e recuperato al termine del codice dell'istruzione; la pila deve infatti rimanere inalterata dopo l'esecuzione dell'istruzione.

L'interesse principale è per il bit di livello 0 cioè il carry C del microprocessore, in quanto contiene il valore ON/OFF del punto corrente di descrizione della rete; in genere molte istruzioni sono eseguite solo in presenza del valore logico "1" in tale bit, mentre sono scavalcate in caso contrario. Quando si richiede tale caratteristica occorrerà racchiudere il listato assembler tra le seguenti due righe di programma:

```
JNC         Istruction_End

--------------------------------

(listato dell'istruzione)

--------------------------------

Istruction_End:
```

In questo modo se il valore contenuto nel carry C è lo "0" logico, il blocco di istruzioni assembler vengono scavalcate e l'istruzione esterna non esegue nulla.

Inoltre dato che il ramo elettromeccanico precedentemente calcolato può alimentare più istruzioni di uscita, occorre non alterare il valore del carry C, in modo che, dopo l'esecuzione dell'istruzione esterna, altre istruzioni possano utilizzare il suo contenuto. Si consiglia a questo proposito di salvare il contenuto del carry C nel bit di memoria interna 70H:

```
MOV         70H,C

-----------------------------------------------------------

(listato dell'istruzione con disponibilità del bit C)

-----------------------------------------------------------

MOV         C,70H
```

Il programma assembler può far riferimento a qualunque area di RAM interna del microprocessore ed esterna (RAM dati), purché non si eseguano delle istruzioni di scrittura che alterino, in modo spesso fatale, l'esecuzione del sistema operativo. Per semplificare le cose consigliamo di utilizzare, per il proprio programma assembler, solo i seguenti registri interni del microprocessore:

```
ACC
PSW
B
REGISTRI BANCO 0
REGISTRI BANCO 1
REGISTRI BANCO 2
REGISTRI BANCO 3
DPTR
STACK           (massima profondità utilizzabile: 32 bytes)
```

L’istruzione esterna MUX

Con il seguente esempio viene creata un’istruzione esterna che permette di realizzare un dispositivo MULTIPLEXER ad 8 uscite. L’istruzione MUX necessita di un solo operando costituito dall’identificatore del byte che si vuole convertire; dopo la conversione il risultato è riposizionato nello stesso byte. La conversione è effettuata solo se il ramo che alimenta l’istruzione MUX è ON; in questo caso la conversione è la seguente:

BYTE (prima della MUX)	BYTE (dopo la MUX)	
0	00000001B	
1	00000010B	
2	00000100B	
3	00001000B	
4	00010000B	
5	00100000B	
6	01000000B	
7	10000000B	
8÷255	00000000B	(F.E = 1)

L’istruzione lavora solo su variabili ad 1 byte per cui il campo di validità dell’operando prima della conversione è 0-7; in tutti gli altri casi si ha una situazione di errore segnalata forzando ad ON il bit di flag F.E.

Questo esempio, opportunamente modificato, costituisce la base per lo sviluppo di qualsiasi altra operazione di conversione del valore di un byte secondo i valori memorizzati in una tabella.

Occorre ricordare che il codice oggetto corrispondente ad un’istruzione esterna (file con estensione .IOF) viene integralmente inserito nel codice oggetto (file .OBJ) del programma utente ogni qual volta si richiama l’istruzione stessa.

Fare quindi attenzione a non creare file .IOF eccessivamente lunghi e, nel caso ciò non fosse possibile, a non richiamare troppo frequentemente l’istruzione nel file di programma. Un metodo per ovviare a tale inconveniente è quello di definire una subroutine contenente la chiamata all’istruzione esterna; in questo modo il codice dell’istruzione esterna verrà incorporato solo una volta nel file .OBJ del programma utente.

Infine fare attenzione ad eliminare l’eventuale parte terminale del file binario creato dall’assemblatore utilizzato; spesso infatti i codici binari generati hanno dimensioni multiple di certe quantità (ad esempio di 128 byte) e riempiono la parte finale con codici operativi corrispondenti ad istruzioni NOP. La parte terminale, se non eliminata, occuperà inutilmente la memoria programma e rallenterà l’esecuzione dell’istruzione.

Riportiamo di seguito il listato assembler dell’istruzione esterna MUX:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                           ;;
;; istruzione:     MUX                                                       ;;
;;                                                                           ;;
;; operandi:       byte          (byte con dimensione 1)                     ;;
;;                                                                           ;;
;; sintassi:       MUX   byte   ‘commento                                    ;;
;;                                                                           ;;
;; funzione:       quando il ramo precedente è ON converte il valore del     ;;
;;                 byte nella posizione del bit; se il byte da convertire    ;;
;;                 non vale 0,1,2,3,4,5,6,7, dopo l’istruzione, il byte      ;;
;;                 vale 0 ed il flag di errore F.E è ON                      ;;
;;                                                                           ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


EXT_FLAG          =     09F10H                  ;puntatore byte dei FLAG

                  ORG   0000H


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


                  JNC   MUX_END                 ;salta se ramo precedente OFF
                  MOV   70H,C                   ;salva temporaneamente il carry
                  MOVX  A,@DPTR                 ;carica in ACC il byte
                  CJNE  A,#8,TRY_LESS           ;verifica se byte < 8
                  SJMP  MUX_ERROR
TRY_LESS:         JNC   MUX_ERROR
                  ADD   A,#2
                  MOVC  A,@A+PC                 ;carica in ACC il byte tabella
                  SJMP  MUX_OK
                  DB    00000001B               ;valore per byte = 0
                  DB    00000010B               ;valore per byte = 1
                  DB    00000100B               ;valore per byte = 2
                  DB    00001000B               ;valore per byte = 3
                  DB    00010000B               ;valore per byte = 4
                  DB    00100000B               ;valore per byte = 5
                  DB    01000000B               ;valore per byte = 6
                  DB    10000000B               ;valore per byte = 7

MUX_ERROR:        MOV   A,#0
                  MOVX  @DPTR,A
                  MOV   DPTR,#EXT_FLAG          ;puntatore byte dei FLAG
                  MOVX  A,@DPTR
                  SETB  ACC.7                   ;set del bit F.E
                  MOVX  @DPTR,A
                  MOV   C,70H                   ;recupera il carry
                  SJMP  MUX_END

MUX_OK:           MOVX  @DPTR,A
                  MOV   C,70H                   ;recupera il carry
MUX_END:


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


                  END
```

L’istruzione esterna SQR

Il seguente esempio illustra come creare nuove istruzioni di tipo matematico da utilizzare sulla catasta operativa. In particolare viene sviluppata un’istruzione di RADICE QUADRATA che opera sul valore presente nel livello 0 della catasta e lascia il risultato nella stessa posizione.

L’algoritmo utilizzato per la valutazione della radice quadrata è molto semplice ed è schematizzato dal diagramma di flusso in Figura 1.

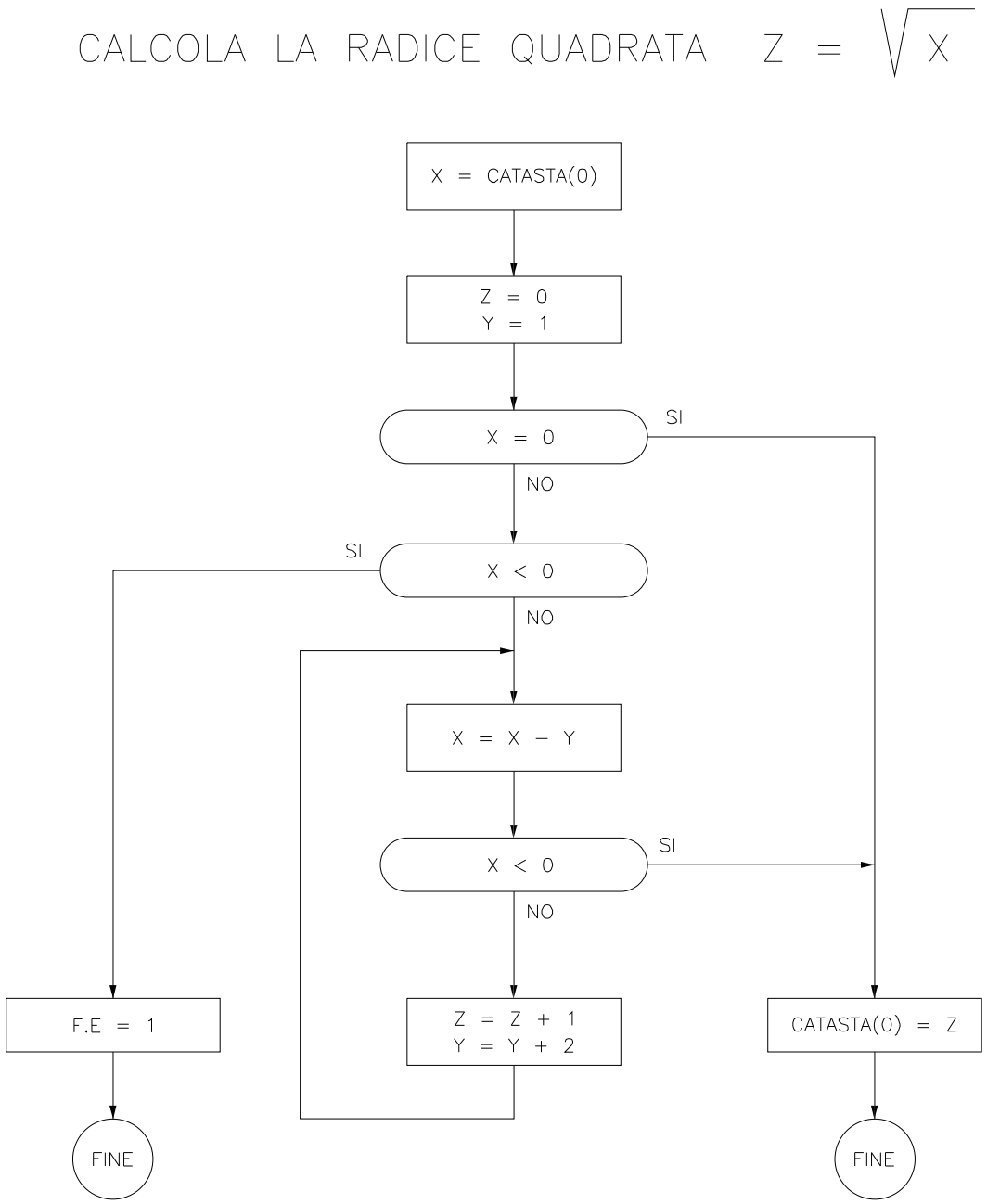


Figura 1. Diagramma di flusso per il calcolo della radice quadrata

Per semplificare la corrispondenza tra il diagramma di flusso riportato ed il listato assembler che implementa l'istruzione SQR, si è fatto riferimento a dei nomi simbolici (X, Y, Z) per le variabili utilizzate nell'algoritmo. La variabile X costituisce il valore di partenza del quale si vuol conoscere la radice quadrata e coincide con il livello 0 della catasta operativa; la variabile Y rappresenta un contatore necessario internamente all'algoritmo ed infine Z rappresenta il valore ricercato della radice. Al termine, se il calcolo della radice ha dato buon esito, il valore di Z viene sovrascritto al valore del livello 0 della catasta.

Il listato assembler corrispondente all'istruzione SQR è il seguente:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;; istruzione:    SQR                                                         ;;
;;                                                                            ;;
;; sintassi:      SQR      'commento                                          ;;
;;                                                                            ;;
;; funzione:      calcola la radice quadrata intera del valore presente       ;;
;;                nella catasta. Il risultato sostituisce il valore senza     ;;
;;                alterare la posizione della catasta.                        ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


BIT_STACK         =     02EH                   ;ram byte per bit stack
EXT_FLAG          =     09F10H                 ;flag utente di stato
RESET_WD          =     0FC0H                  ;routine di reset watchdog


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


                  ORG   0000H

                  MOV   BIT_STACK.0,C          ;salvataggio del bit stack

                  MOV   014H,#0                ;azzeramento risultato Z = 0
                  MOV   015H,#0
                  MOV   016H,#0
                  MOV   017H,#0
                  MOV   018H,#1                ;contatore dispari Y = 1
                  MOV   019H,#0
                  MOV   01AH,#0
                  MOV   01BH,#0

                  MOV   A,010H                 ;verifica se X = 0
                  ORL   A,011H
                  ORL   A,012H
                  ORL   A,013H
                  JZ    SQR_END

                  MOV   A,013H                 ;verifica se X < 0
                  JB   ACC.7,SQR_NEG
```

```
SQR_LOOP:       CLR   C                     ;calcola X = X - Y
                MOV   A,010H
                SUBB  A,018H
                MOV   010H,A
                MOV   A,011H
                SUBB  A,019H
                MOV   011H,A
                MOV   A,012H
                SUBB  A,01AH
                MOV   012H,A
                MOV   A,013H
                SUBB  A,01BH
                MOV   013H,A

                JB    ACC.7,SQR_END         ;verifica se X < 0

                MOV   A,014H                ;incrementa Z = Z + 1
                ADD   A,#1
                MOV   014H,A
                MOV   A,015H
                ADDC  A,#0
                MOV   015H,A
                MOV   A,016H
                ADDC  A,#0
                MOV   016H,A
                MOV   A,017H
                ADDC  A,#0
                MOV   017H,A

                MOV   A,018H                ;incrementa Y = Y + 2
                ADD   A,#2
                MOV   018H,A
                MOV   A,019H
                ADDC  A,#0
                MOV   019H,A
                MOV   A,01AH
                ADDC  A,#0
                MOV   01AH,A
                MOV   A,01BH
                ADDC  A,#0
                MOV   01BH,A

                LCALL RESET_WD              ;reset del watchdog
                SJMP  SQR_LOOP

SQR_NEG:        MOV   DPTR,#EXT_FLAG        ;radice di valore negativo
                MOVX  A,@DPTR
                SETB  ACC.7                 ;setta bit di errore
                MOVX  @DPTR,A
                SJMP  SQR_EXIT

SQR_END:        MOV   010H,014H             ;trasferimento risultato Z
                MOV   011H,015H
                MOV   012H,016H
                MOV   013H,017H

SQR_EXIT:       MOV   C,BIT_STACK.0


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

                END
```

L’ambiente di sviluppo

La programmazione mediante PC

Il software ICL51 opera su Personal Computer IBM® (oppure compatibile) con sistema operativo MS-DOS; esso consente tutte le operazioni di sviluppo del programma utente nel linguaggio ICL51 e permette un'immediata utilizzazione delle Logiche sulla macchina o impianto da automatizzare.

Per la programmazione occorre dunque dotarsi di un comune Personal Computer con le seguenti caratteristiche:

- 640 Kbytes di memoria RAM
- 1 interfaccia seriale RS232
- 1 driver per floppy-disk da 1.4 MBytes
- Hard-disk con almeno 5 Mbytes disponibili

Si riscontra immediatamente che i requisiti richiesti al Computer di programmazione sono estremamente limitati ed ampiamente soddisfatti da qualsiasi elaboratore in commercio.

La scrittura del programma d'automazione della macchina avviene utilizzando un qualunque Editor di testi, purché questo sia in grado di elaborare files di testo "puro" ossia senza caratteri speciali e di controllo tipici dello specifico Editor. La scelta di tale Editor è lasciata al Programmatore, consentendogli di utilizzare quello con il quale ha maggior pratica; tuttavia, in dotazione con il software ICL51 release 4.0, viene fornito l'editor EDIT51.EXE, particolarmente adatto a tale compito. In ogni caso l'Editor esterno può essere richiamato dall'interno dell'ambiente di sviluppo ICL51.

Tramite l'Editor si dovrà preparare un file di testo contenente tutte le righe del programma, in linguaggio ICL51, secondo la sintassi descritta nelle apposite sezioni di questo manuale.

Successivamente un programma compilatore consentirà di tradurre le informazioni contenute nel listato sorgente, così editato, in informazioni comprensibili dal microprocessore della logica. Questo avviene con la creazione, da parte del compilatore, di un file oggetto il quale costituisce una versione equivalente del programma editato ma in linguaggio assembler, utilizzabile direttamente dal microprocessore.

Le funzioni di trasferimento da e verso la Logica, sia del programma che dei dati dello stesso, sono possibili connettendo la scheda MASTER all'interfaccia seriale del Personal Computer.

Il programma ICL51 consente inoltre il debugging del programma macchina mediante la sua esecuzione in tempo reale: durante l'esecuzione è possibile monitorare e forzare tutte le variabili interne della Logica.

Un semplice diagramma di flusso delle operazioni, presentato dal menu principale del software ICL51, guiderà anche il Programmatore meno esperto durante tutte le fasi di sviluppo del programma di automazione. L'estrema semplicità ed immediatezza del sistema consentiranno al Programmatore di concentrarsi esclusivamente sul programma da sviluppare e non sulla "giungla" di menu e sottomenu del software di sviluppo.

Installazione del software ICL51

Il pacchetto software ICL51 viene normalmente fornito mediante floppy-disks contenenti tutti i files necessari al suo funzionamento ed una vasta serie di programmi applicativi e dimostrativi. Aggiornamenti futuri sono possibili mediante dischetti contenenti i nuovi files da sovrascrivere sui vecchi o da aggiungere a questi. La diffusione del software ICL51 non è soggetta ad alcuna licenza d'uso; tra l'altro il pacchetto software ICL51, i suoi aggiornamenti, programmi ausiliari e tutta la documentazione in formato .PDF sono prelevabili in ogni momento tramite la rete INTERNET.

Per iniziare a lavorare occorre per prima cosa installare il programma ICL51 sul Personal Computer che si intende utilizzare per la programmazione delle Logiche. Piena libertà nell'installazione e nell'impostazione delle directory è lasciata all'utilizzatore del pacchetto; di seguito diamo solo alcune indicazioni ed informazioni utili in questa fase.

Creare una directory, per esempio col nome ICL51_40, e copiare tutti i files del dischetto in tale directory; supponendo di lavorare sul disco fisso C: e di aver inserito il floppy-disk del programma ICL51 nel drive A:, le operazioni sono le seguenti:

```
C:\

MD ICL51_40

CD ICL51_40

COPY A:*.* C:
```

Attenzione: normalmente, a causa delle elevate dimensioni richieste dall'insieme di tutti i files, i dischetti forniti per il pacchetto software o per i suoi aggiornamenti contengono delle versioni compresse dei dati. Decomprimere tale files per ottenerli nella loro forma originaria ed utilizzabile.

Si consiglia inoltre di creare una subdirectory di ICL51_40 per archiviare i vari programmi utente creati, allo scopo di tenerli separati dei programmi eseguibili dell'ambiente di sviluppo.

Per richiamare il programma ICL51 da un'eventuale altra directory, aggiungere nel file AUTOEXEC.BAT la voce C:\ICL51_40 al comando PATH.

Effettuare infine una copia di tutti i files di configurazione *.CFG nella directory corrente di lavoro. Questi files consentono di memorizzare tutte le configurazioni scelte per operare con il sistema; le configurazioni possono essere anche differenziate per ogni particolare programma utente, predisponendo delle copie dei files di configurazione in ogni subdirectory di lavoro e modificandone opportunamente le opzioni in funzione del particolare programma da sviluppare.

La release 4.0 del software ICL51 è stata completamente ristrutturata allo scopo di rendere semplice ed automatico il suo aggiornamento in base allo sviluppo successivo di nuove Logiche. Infatti alcuni dei files sono di tipo generale dell'ambiente e validi per tutte le Logiche, mentre altri sono stati duplicati in modo specifico per ogni Logica. In questo modo per aggiornare il software ad un nuovo prodotto è necessario solamente aggiungere nella directory ICL51_40 i nuovi files relativi a questo ed automaticamente verrà riconosciuto dall'ambiente di programmazione.

Per maggior comprensione di come è strutturato il software ICL51 release 4.0 riportiamo un sommario elenco di tutti i files che lo compongono. Con la parola generica LOGICA si vuole indicare una o più delle Logiche che si intende utilizzare e delle quali è necessaria la presenza dei relativi files; con la parola NOMEFILE si intende il nome di uno o più programmi utente di esempio forniti a corredo del software:

ICL51.EXE	programma eseguibile da richiamare per l'ambiente (menu principale)
CFG51.EXE	programma di configurazione generale (richiamato dal menu principale)
ICL51.CFG	file di configurazione generale dell'ambiente
VIEW51.EXE	programma di visualizzazione dei files (richiamato dal menu principale)
EDIT51.EXE	programma editor di files (richiamato dal menu principale)
ICL51.HLP	file di aiuto visualizzabile dall'interno dell'ambiente

C_LOGICA.EXE	programma di compilazione (richiamato dal menu principale)
T_LOGICA.EXE	programma di trasferimento (richiamato dal menu principale)
M_LOGICA.EXE	programma di monitor (richiamato dal menu principale)
M_LOGICA.CFG	file di configurazione del monitor

Attenzione: questi ultimi 4 files sono necessari per l'utilizzo di una specifica Logica. Normalmente il pacchetto software ICL51 viene fornito comprensivo dei files di tutte le Logiche commerciali.

NOMEFILE.PRG	file sorgente del programma utente specifico dell'applicazione
NOMEFILE.OBJ	file oggetto del programma utente (da trasferire sulla logica)
NOMEFILE.ERR	file di report degli eventuali errori di compilazione del programma utente
NOMEFILE.TXT	file di testo archiviato nel programma utente dopo il suo recupero
NOMEFILE.HLD	file di dati memorie ritentive di tipo H
NOMEFILE.XMF	file di dati memorie ritentive di tipo X (disponibili solo su alcune Logiche)

Ricordiamo infine che i files con estensione .IOF corrispondono ad istruzioni esterne; in particolare con il software ICL51 release 4.0 sono distribuiti i due esempi (MUX ed SQR) riportati nel presente manuale.

L’ambiente di sviluppo ICL51

L’intero ambiente di sviluppo del programma utente viene gestito dall’eseguibile ICL51.EXE. Per iniziare a lavorare con il programma ICL51 occorre lanciare il menu principale digitando dalla riga comando del DOS:

ICL51

La schermata principale dell’ambiente ICL51 è molto semplice ed intuitiva. Uno schema grafico a cinque stati riassume le principali fasi da seguire durante lo sviluppo della propria applicazione; queste fasi sono richiamabili sia dal tasto funzione indicato, che digitando il tasto INVIO dopo essersi posizionati mediante i tasti direzione (Freccie) sulla casella voluta. Lo stesso grafo si posiziona automaticamente sulla casella più opportuna a seconda della fase precedentemente svolta ed in base all’esito di questa.

Altre funzioni ausiliarie, richiamabili dai tasti funzione, sono disponibili dal questo menu principale ed una finestra di stato riporta le condizioni di configurazione generale dell’ambiente.

Le funzioni disponibili dal menu principale sono le seguenti:

Edit (F1)

Richiama l’Editor di testi selezionato per la stesura del listato programma sorgente .PRG. Questa è la prima fase nello sviluppo di un programma applicativo; il programma di editor selezionato viene eseguito passandogli come parametro il nome del programma sorgente.

Compile (F2)

Compila il programma sorgente .PRG generando il file oggetto .OBJ ed il file di errore .ERR. Questa fase si avvale dello specifico compilatore della Logica selezionata e trasforma il listato programma, scritto in linguaggio ICL51, in un programma equivalente ma comprensibile dal microprocessore della Logica.

View Error (F3)

Richiama il programma di visualizzazione testi selezionato passandogli come parametro il nome del file di errore .ERR del programma corrente. La visione di tale file è indispensabile nel caso che il processo di compilazione, al suo termine, segnali la presenza di errori.

Transfer (F4)

Richiama lo specifico programma di traferimento della Logica selezionata. Questo programma si presenta a sua volta come un nuovo menu che offre varie opzioni di trasferimento da e verso la Logica. Un'analisi più approfondita di questo menu verrà svolta nell'apposito paragrafo.

Monitor (F5)

Richiama lo specifico programma di monitor della Logica selezionata. Sullo schermo del monitor viene visualizzato il listato del programma corrente e nella metà sottostante dello schermo è possibile aprire la finestra di monitor. Questa finestra permette di visualizzare e forzare lo stato di tutte le variabili interne alla memoria RAM della Logica (risorse) allo scopo di testare l'esecuzione del programma. Il programma di monitor sarà maggiormente approfondito nel seguito.

Dos Shell (F6)

Permette di uscire momentaneamente dall'ambiente ICL51 e di eseguire qualsiasi comando o programma DOS. Per rientrare nell'ambiente, terminando la shell, occorre inviare il comando EXIT tramite la tastiera.

Configure (F7)

Richiama il programma di configurazione CFG51.EXE nel caso occorra cambiare una o più delle impostazioni generali dell'ambiente ICL51. Le informazioni di configurazione generali sono archiviate nel file ICL51.CFG.

Help (F8)

Visualizza, mediante il programma di visualizzazione selezionato, il file ICL51.HLP contenente il testo di aiuto all'utilizzo dell'ambiente.

Esc

Il tasto Escape consente di lasciare il menu principale dell'ambiente di programmazione terminando il programma ICL51.

La configurazione del software

Prima di iniziare a lavorare per la prima volta ad un nuovo progetto occorre configurare opportunamente l'ambiente di sviluppo ICL51.

Dal menu principale digitare il tasto funzione F7 per accedere al programma di configurazione; temporaneamente viene lanciato il programma CFG51.EXE il quale consente di visualizzare e modificare le impostazioni presenti nel file di configurazione generale dell'ambiente (ICL51.CFG).

Sono presentate otto distinte opzioni di configurazione. Per modificare la configurazione posizionarsi mediante i tasti direzione (Frecce) sull'opzione prescelta e digitare il tasto INVIO. Alcune delle opzioni possono assumere solo pochi valori alternativi; in questo caso la ripetuta pressione del tasto INVIO permetterà di cambiare il valore dell'opzione. Negli altri casi è invece necessario fornire un nome digitandolo per intero o selezionandolo dalla lista presentata; con il tasto Esc si può annullare la selezione. Digitando Esc dal menu di configurazione si termina questo programma e si ritorna al menu principale dell'ambiente.

Riportiamo nel seguito una descrizione più specifica e dettagliata delle possibili opzioni di configurazione e di come operare per la loro modifica:

Logic type

Seleziona il tipo di Logica MASTER scelta per l'applicazione. Digitando il tasto INVIO viene presentata una lista di tutte le Logiche disponibili; per la disponibilità della Logica viene testata la presenza, nella directory corrente di lavoro, almeno del file eseguibile di compilazione (C_LOGICA.EXE). Il nome della Logica viene estratto dallo stesso nome del programma compilatore eliminando il prefisso "C_" e verrà successivamente utilizzato per richiamare anche gli altri programmi specifici della Logica (trasferimento "T_" e monitor "M_").
Per selezionare la Logica posizionarsi su quella prescelta e confermare con INVIO; il tasto Esc chiude la finestra di elenco Logiche ed annulla la selezione.

File

Indica il nome del file di progetto da utilizzare correntemente per tutte le operazioni svolte dall'ambiente. Il nome specificato (max 8 caratteri) verrà corredato automaticamente della corretta estensione richiesta nelle varie fasi di sviluppo del programma.
Per selezionare il nome del programma utente digitare INVIO su tale opzione. A questo punto è possibile digitare per esteso il nome del file oppure richiamare uno o più programmi, già presenti nella directory corrente di lavoro, digitando un nome comprensivo dei caratteri jolly ? (punto interrogativo) ed * (asterisco). Il carattere ? sostituisce un solo carattere generico del nome di ricerca, mentre il carattere * equivale ad una parte del nome costituita da generici caratteri. Tutti i nomi rispondenti alle caratteristiche di ricerca saranno elencati in una finestra; a questo punto, posizionandosi con i tasti direzione e confermando con INVIO, si selezionerà il voluto nome di programma.

Logic at

Seleziona il numero di porta seriale RS232 da utilizzare per tutte le operazioni di comunicazione tra il Personal Computer e la Logica utilizzata.
La successiva pressione del tasto INVIO cambia il valore della corrente selezione; le seriali utilizzabili presentate sono la COM1, COM2, COM3 e COM4.

Printer at

Seleziona il numero di porta parallela CENTRONICS da utilizzare per tutte le operazioni di stampa eseguite dell'interno dell'ambiente di programmazione.
La pressione ripetuta del tasto INVIO seleziona una delle possibili interfacce parallele (LPT1, LPT2 e LPT3).

Automatic

Permette di abilitare o meno la funzione di richiamo automatico della compilazione, trasferimento (download) e monitor all'uscita dal programma di Editor del file sorgente.
Con il tasto INVIO è possibile cambiare da "Yes" a "No" questa funzione. Selezionando "N" al termine del programma di Editor si ritorna al menù principale dell'ambiente e tutte le successive fasi devono essere richiamate manualmente. Con l'opzione su "Y" invece, al termine del programma di Editor, viene attivato automaticamente il compilatore e, se non sono stati riscontrati errori in questa fase, il file oggetto corrispondente verrà trasferito sulla logica; infine verrà richiamato automaticamente il menu di monitor. Questa automatizzazione delle operazioni è molto utile per accelerare la fase di messa a punto del programma quando si richiedono continue modifiche e prove dello stesso.

Editor

Consente di selezionare il nome del programma di Editor da utilizzare per la stesura del listato sorgente. Digitando INVIO è possibile introdurre il nome (senza estensione) del programma eseguibile da utilizzare; si sottolinea che tale Editor di testi deve essere disponibile nella directory corrente di lavoro o almeno reso visibile tramite il comando PATH del file autoexec.bat.
Il programma di Editor viene lanciato dal menu principale mediante una riga di comando DOS costituita dal nome dell'Editor seguito dal nome del programma applicativo con estensione .PRG. Nel caso l'Editor che si desidera utilizzare non preveda il passaggio nella riga comando del nome del file da editare, occorrerà creare un file fittizio di tipo batch che opera in modo intermedio tra la chiamata effettuata dall'ambiente di programmazione e il programma esterno di editing.
In dotazione con il pacchetto software ICL51 release 4.0 viene fornito il programma EDIT51, particolarmente studiato per la stesura del listato sorgente. Il suddetto programma è molto semplice e pratico da utilizzare; mediante il tasto funzione F1 è possibile visualizzare dal suo interno un elenco dei comandi e delle funzioni disponibili.

Viewer

Consente di selezionare il nome del programma di visualizzazione dei files di testo da utilizzare nei vari punti dell'ambiente dove si richiede tale funzione.
Digitando il tasto INVIO è possibile introdurre un nome di un programma di visualizzazione esterno; il passaggio del nome del file da visualizzare avviene con la stessa tecnica adottata per il programma di Editor.
Si consiglia di utilizzare il programma VIEW51 fornito con il pacchetto software ICL51 release 4.0 in quanto esso dispone, in modo molto semplice ed immediato, di tutte le possibili funzioni richieste da un programma di questo genere.

Password

L'indicazione della Password dell'ambiente di programmazione è indispensabile solo per le fasi di recupero delle informazioni dalla Logica al Personal Computer.
La Password è costituita da una stringa (al massimo di 8 caratteri) e viene comparata con quella archiviata nella memoria programma della Logica (tramite l'istruzione PASSW del programma sorgente). Quando le due Password non coincidono vengono annullate alcune operazioni come l'UpLoad ed il Compare.
Digitando il tasto INVIO è possibile digitare la Password di ambiente. Si ricorda che nel caso nel listato sorgente non sia stata ridefinita la Password di Logica, mediante l'istruzione PASSW, il compilatore per default inserirà nel file oggetto la Password "PASSWORD"; in questo caso occorrerà verificare che anche la Password di ambiente sia la stringa "PASSWORD".

La scrittura del programma

La scrittura del file di testo contenente la lista delle istruzioni in linguaggio ICL51 è la prima fase richiesta nello sviluppo di un proprio programma applicativo.

Per la stesura di questo file di testo, denominato programma sorgente e con estensione .PRG, è necessario utilizzare un Editor di testi qualsiasi. Esistono molti programmi commerciali per la stesura dei files di testo; alcuni di questi sono stati espressamente realizzati per la scrittura di programmi, indipendentemente dal linguaggio. La libertà di utilizzare un Editor di testi con il quale si ha già una certa pratica consente al Programmatore di iniziare a lavorare immediatamente e con la massima operatività.

Tuttavia, per completezza del pacchetto software ICL51 release 4.0, viene fornito un programma di Editor di testi chiamato EDIT51: questo Editor è stato sviluppato per consentire la programmazione delle Logiche a tutti coloro che non dispongono di un proprio Editor preferenziale.

Il file di testo deve contenere tutte le istruzioni richieste dal programma in svolgimento secondo le regole sintattiche descritte nelle apposite parti di questo manuale. Istruzioni non riconosciute o applicate in maniera errata saranno individuate e listate dal compilatore.

La compilazione ed il file di errore

La compilazione è la fase successiva alla stesura del file di testo sorgente del programma. Questa fase è necessaria ogni qual volta si desidera provare il programma sulla Logica dopo che sono state apportate delle modifiche nel file sorgente. Infatti occorre riaggiornare il contenuto del file oggetto .OBJ in funzione delle nuove informazioni disponibili nel file sorgente .PRG.

Il compilatore analizza riga per riga le istruzioni del file sorgente e genera il corrispondente codice assembler comprensibile al microprocessore della Logica. Durante la verifica delle linee di programma tutti gli errori di utilizzo delle istruzioni vengono annotati nel file di errore .ERR per una successiva consultazione.

Il file di errore .ERR viene sempre generato dal compilatore in quanto contiene anche altri generi di informazioni come l'occupazione percentuale di memoria del programma corrente; questo dato è molto importante in quanto consente di tenere sotto controllo la disponibilità della memoria programma ad accettare l'aggiunta di nuove istruzioni.

Il Menu di trasferimento

Attivando il menu di trasferimento vengono presentate le varie opzioni di comando disponibili. In questo menu sono riunite tutte le funzioni di trasferimento da e verso la Logica sia del programma macchina (memorizzato nella memoria FLASH-EPROM) che dei dati e parametri di lavoro (memorizzati nella RAM).

Tutte le operazioni di trasferimento avvengono mediante la connessione seriale RS232 tra il Personal Computer e la Logica. Verificare prima di procedere che la connessione sia stata correttamente realizzata e che le opzioni di configurazione (numero di seriale COM utilizzata) siano opportunamente selezionate.

I tasti funzione della tastiera del Personal Computer vengono ridefiniti per le funzioni del menu di trasferimento secondo quanto segue:

DownLoad (F1)

Attiva il trasferimento del programma dal Personal Computer alla Logica selezionata.
Il file oggetto .OBJ viene programmato in modo permanente a bordo della memoria FLASH-EPROM della Logica dopo la preventiva cancellazione dell'eventuale programma preesistente. Eventuali problemi riscontrati durante la programmazione della memoria FLASH-EPROM saranno tempestivamente segnalati dal software.
A questo proposito analizziamo una situazione particolare che si può verificare durante la programmazione.
La Logica, alla sua accensione, esamina la presenza di un programma caricato nella memoria FLASH-EPROM ed in questo caso entra automaticamente nello stato di esecuzione del programma (RUN).
Nel caso invece di memoria FLASH-EPROM cancellata, viene automaticamente inizializzato lo stato di fermo (STOP) in attesa del DownLoad di un programma.
Esiste un terzo caso possibile che è quello di un programma alterato, per un motivo qualunque, oppure con errati salti o loop infiniti causati da un'errata stesura funzionale del programma utente. In questi casi si avrà un intervento continuo del circuito di watch-dog della scheda con conseguente impossibilità di riaggiornare il programma. Per uscire da tale situazione occorre forzare in STOP all'accensione la Logica, mediante l'apposito jumper o dip-switch oppure mediante la funzione F9 (per le Logiche che lo prevedono) di questo stesso menu. Successivamente all'entrata della Logica nello stato di STOP è possibile eseguire il DownLoad di un programma corretto.

UpLoad (F2)

Attiva il trasferimento del programma dalla Logica selezionata al Personal Computer.
Con questo comando è possibile leggere il programma presente nella memoria FLASH-EPROM della Logica archiviandolo sul file .OBJ. L'eventuale area di testi commento (commenti del programma sorgente originario preceduti dal carattere " doppio apice) verrà archiviata sul file .TXT per una successiva consultazione.
Questa operazione di trasferimento è possibile solo se la Password memorizzata sulla Logica coincide con quella impostata nell'ambiente.

Si vuole sottolineare che il file di programma recuperato dalla Logica è il solo file oggetto .OBJ; il file sorgente .PRG non può essere recuperato poichè le sue dimensioni in bytes sono spesso molto elevate. Si pensi infatti a tutte le Label alfanumeriche a 32 catratteri o i commenti alle istuzioni che possono essere presenti in un file sorgente; sarebbe stato necessario limitare le dimensioni del file sorgente e renderlo meno generoso nella sua forma per poterlo archiviare tutto nella memoria della Logica.

View Text (F3)

Richiama il visualizzatore di file di testo selezionato passondogli il file .TXT.
Il file di testo .TXT viene creato automaticamente dalla funzione di UpLoad se, nel file .OBJ recuperato dalla memoria della Logica, è stato trovato almeno un commento del tipo archiviabile.
Nel listato sorgente .PRG è possibile infatti digitare intere linee di testo di commento facendole iniziare con il carattere " (doppio apice); queste linee di testo vengono incluse, con una codifica segreta, nel file .OBJ da trasferire sulla memoria della Logica.
Lo scopo di tali commenti è quello di poter disporre di un "block-notes" a bordo di ogni singola Logica. Ben 8176 caratteri di testo sono disponibili per poter descrivere eventuali caratteristiche o modifiche del programma della macchina oppure per archiviare tutta una serie di informazioni sulla stessa come se questa disponesse di un "Libro di bordo" leggibile solo da chi conosce la Password.

Compare (F4)

Attiva il confronto tra il file .OBJ presente sul Personal Computer ed il contenuto della memoria FLASH-EPROM della Logica.
Durante la funzione di programmazione, mediante il comando DownLoad, viene fatta automaticamente una verifica dell'avvenuta scrittura della memoria FLASH-EPROM per cui il comando di Compare non è strettamente necessario.
Il comando Compare può essere utile per verificare se il programma memorizzto sulla Logica corrisponde ad un determinato programma di cui si dispone del file .OBJ.

Backup H/X bytes (F5)

Consente di leggere i valori correnti delle memorie ritentive di tipo H dalla Logica e di archiviarle sul file con estensione .HLD.
Per le Logiche che dispongono di memorie ritentive di tipo X, è possibile eseguire un'operazione equivalente con archiviazione sul file .XMF. Solo in questo caso è presente l'opzione /X nella voce del menu e la prima finestra di dialogo presentata richiederà di specificare se si desidera leggere le memorie di tipo H oppure di tipo X.
Il comando di backup necessita dell'introduzione del numero di byte iniziale e del numero di byte finale dell'area di memoria che si intende leggere. Nel caso delle memorie di tipo H il campo di validità degli estremi è 0÷1023 mentre per le memorie di tipo X è 0÷24567; inoltre il numero di byte finale non deve essere inferiore a quello iniziale.

Lo scopo del comando di backup è prevalentemente quello di recuperare i valori dei parametri di lavoro della macchina per poi scaricarli su altre Logiche; infatti, quando una Logica è nuova, la memoria RAM ritentiva non può ovviamente già contenere i parametri di lavoro. Archiviando in un file le informazioni contenute in una Logica di riferimento già configurata e successivamente scaricando tale file con il comando di restore su di una seconda Logica, è possibile predisporre questa a funzionare secondo i corretti parametri.
Il formato dei files .HLD e .XMF è di tipo testo e può essere visualizzato, stampato ed anche elaborato da programmi specifici esterni all'ambiente ICL51.

Restore H/X bytes (F6)

Consente di scrivere i dati precedentemente archiviati nel file .HLD sulle memorie ritentive di tipo H dalla Logica.
Per le Logiche che dispongono di memorie ritentive di tipo X, è possibile eseguire un'operazione equivalente prelevando i dati dal file .XMF. Solo in questo caso è presente l'opzione /X nella voce del menu e la prima finestra di dialogo presentata richiederà di specificare se si desidera scrivere le memorie di tipo H oppure di tipo X.
Il comando di restore necessita dell'introduzione del numero di byte iniziale e del numero di byte finale dell'area di memoria che si intende scrivere. Nel caso delle memorie di tipo H il campo di validità degli estremi è 0÷1023 mentre per le memorie di tipo X è 0÷24567; inoltre il numero di byte finale non deve essere inferiore a quello iniziale.
Nel caso venga specificata un'area non coincidente al contenuto del file .HLD (oppure del file .XMF per le memorie X), i bytes da riempire non trovati nel file di backup saranno forzati con il valore decimale zero. Questo suggerisce un trucco per riempire con estrema semplicità tutta o parte della RAM ritentiva della Logica con il valore zero; basta infatti creare un file .HLD (oppure .XMF per le memorie X) contenente la lettura di un solo byte coincidente con l'estremo iniziale e con valore decimale zero. Successivamente effettuare il restore specificando questo estremo iniziale ed un voluto estremo finale: il primo byte verrà effettivamente letto dal file, mentre tutti i successivi, non trovati nel file stesso, saranno automaticamente forzati al valore zero.

View Holding file (F7)

Richiama il visualizzatore di file selezionato passandogli come parametro il file .HLD.
Con questo comando è quindi possibile visualizzare, analizzare e stampare il file di backup.

Update Watch (F8)

Il comando di Update Watch esegue una forzatura dell'orario e della data del Computer nell'orologio/calendario della Logica (quando questo optional è installato).
Questa operazione può essere effettuata sia con la Logica in RUN che in STOP; l'unica differenza è che nel caso di Logica in RUN, tale comando non potrà esplicare le sue funzioni nel caso siano attive nel programma utente delle istruzioni di scrittura sull'orologio/calendario. Questo perché si verrebbe ad avere una sovrapposizione tra le operazioni effettuate dal programma in esecuzione sulla Logica e le operazioni di trasferimento del Computer.

Prima di effettuare l'aggiornamento dell'orologio/calendario viene visualizzato l'orario e la data del Computer; nel caso non fossero corretti, prima del trasferimento nella Logica, aggiornarli mediante i comandi TIME e DATE del DOS (a questo proposito basta uscire temporaneamente dall'ambiente di sviluppo mediante il comando Dos Shell).
La conferma dell'aggiornamento dell'orologio/calendario inizia le operazioni di trasferimento dei valori nei bytes di tipo W della Logica.
Questa operazione può essere eseguita in linea di produzione della macchina allo scopo di presettare correttamente l'orologio/calendario. Successivi aggiustamenti dell'orario e data potranno essere effettuati sulla Logica di nuovo con tale comando o, se non si dispone di un Computer, mediante un pannello terminale programmato a tale scopo.

Stop Logic (F9)

Questo comando consente di forzare la Logica nello stato di STOP all'accensione. In alcune Logiche questa funzione è ottenuta mediante la chiusura di un determinato dip-switch o jumper e quindi la presente voce di menu non è disponibile. Nelle Logiche di ultima generazione l'operazione è invece ottenuta senza dover effettuare alcuna configurazione hardware sulla scheda.
Le fasi da seguire per forzare la Logica nello stato di STOP all'accensione sono le seguenti.
Togliere alimentazione alla Logica e successivamente attivare il comando con F9. Alimentare la Logica ed attendere che termini le sue operazioni iniziali (normalmente 2 secondi sono più che sufficienti). A questo punto disattivare il comando digitando di nuovo il tasto funzione F9. La Logica si trova ora in STOP ed è disponibile alla programmazione (DownLoad); si ricorda che la funzione di STOP all'accensione ha prevalentemente lo scopo di uscire da situazioni di stallo nelle quali il programma memorizzato, per un qualche motivo, non è corretto ma la Logica all'accensione entra ugualmente in RUN, causando un intervento ciclico del circuito di watch-dog.

Il monitor del programma

Il programma di monitor consente di visualizzare e di modificare il contenuto della memoria RAM della Logica durante l'esecuzione in tempo reale del programma utente.

Attivando tale funzione viene richiamato sullo schermo del Personal Computer il listato del programma utente correntemente selezionato. Questo listato farà sempre da sfondo per tutte le operazioni di monitor; nel caso di attivazione della finestra di monitor l'area di schermo utilizzata per il listato verrà ristretta alla metà soprastante del video, mentre nella metà sottostante verrà visualizzata la finestra di monitor.

Per scorrere l'intero listato programma sullo schermo utilizzare i seguenti tasti:

• Freccia su	sposta il listato di una linea verso l'alto
• Freccia giù	sposta il listato di una linea verso il basso
• Pagina su	sposta il listato di una pagina verso l'alto
• Pagina giù	sposta il listato di una pagina verso il basso
• Home	ritorna all'inizio del listato
• Fine	va alla fine del listato

Per attivare la finestra di monitor digitare il tasto funzione F1=Watch. Se è stata selezionata l'opzione di configurazione Automatic, la finestra di monitor viene automaticamente richiamata e la Logica è forzata subito nello stato di RUN (Start del programma).

La finestra di monitor permette di visualizzare contemporaneamente 8 variabili in altrettante righe dello schermo. Ciascuna riga riporta diverse informazioni della variabile:

- Il campo T evidenzia il tipo della variabile con e senza segno (U=unsigned, S=signed) necessario all'interpretazione corretta da parte del monitor della convenzione utilizzata per la variabile (intero senza segno oppure intero con segno mediante notazione in complemento a 2).

- Il campo N specifica il numero di bytes (1, 2 o 4) della variabile; anche questa scelta deve essere comunicata al programma di monitor per una visualizzazione completa del valore a 8, 16, 32 bits.

- Il campo Byte indica il nome della variabile mediante il mnemonico dell'operando secondo quanto stabilito dal linguaggio ICL51.

- Il campo Description consente di associare alla variabile una stringa di testo descrittiva (al massimo di 16 caratteri). Se nel file di configurazione M_LOGICA.CFG non viene trovata alcuna descrizione (stringa nulla) il sistema automaticamente andrà a verificare nel listato sorgente del programma utente la presenza di un'assegnazione di Label alfanumerica all'operando in questione ed utilizzerà tale Label come descrizione del monitor.

- Il campo Decimal visualizza il valore decimale correntemente monitorato della variabile indicata. Tale valore rappresenta il valore con o senza segno della variabile ad 1, 2 o 4 bytes.

- Il campo di visualizzazione dei singoli bits della variabile permette di analizzare lo stato di tutti i relè o flags associati a tale area di memoria.

Il sistema prevede la predisposizione nel file di configurazione M_LOGICA.CFG di un elenco di 512 variabili che possono essere richiamate al volo nella finestra di monitor. Questa finestra di ampiezza 8 variabili può essere spostata avanti/indietro sulla lista continua di tutte le variabili memorizzate nel file di configurazione.

Una in particolare delle 8 variabili visualizzate nella finestra è evidenziata sullo schermo: questa è la variabile attualmente puntata e su essa saranno effettuate tutte le operazioni di modifica offerte dal monitor. Sulla variabile puntatata è anche evidenziato uno dei suoi bits per consentire le operazioni di modifica sul singolo bit.

Per spostare la visualizzazione della finestra di monitor nella lista di tutte le 512 variabili utilizzare i seguenti tasti:

• Ctrl + Freccia su	punta alla variabile precedente
• Ctrl + Freccia giù	punta alla variabile successiva
• Ctrl + Pagina su	sposta la finestra sulle 8 variabili precedenti
• Ctrl + Pagina giù	sposta la finestra sulle 8 variabili successive
• Ctrl + Home	sposta la finestra alle prime 8 variabili
• Ctrl + Fine	sposta la finestra alle ultime 8 variabili
• Ctrl + Freccia sinistra	sposta il puntatore bit di una posizione a sinistra
• Ctrl + Freccia destra	sposta il puntatore bit di una posizione a destra

Il menu di monitor consente di richiamare diverse funzioni delle quali alcune agiscono sulla specifica variabile puntata o sul suo bit puntato. Per questo le azioni svolte dai suddetti tasti sono molto importanti; prima di agire su una determinata variabile o un suo bit occorre accertarsi di aver posizionato correttamente il puntatore.

Le funzioni disponibili nel programma di monitor sono le seguenti:

Start (F1)

Forza la Logica nello stato di RUN avviando la scansione del programma utente. La Logica rimane in esecuzione programma fino al successivo comando di Stop.

Stop (F2)

Forza la Logica nello stato di STOP fermando la scansione del programma utente.
In questo stato la finestra di monitor visualizzerà i valori iniziali presenti nella RAM dati prima dell'entrata nello stato di RUN (come all'accensione). La Logica rimane in fermo programma fino al successivo comando di Start.

Change (F3)

Consente di selezionare una diversa variabile nella posizione correntemente puntata.
Attivando tale comando una finestra di dialogo richiederà l'introduzione dei campi T, N, Byte ed eventualmente Description.
Con il tasto INVIO è possibile confermare i valori dei singoli campi mentre con il tasto Esc si annulla il comando.

La nuova variabile selezionata verrà aggiornata nel file di configurazione del monitor M_LOGICA.CFG alla posizione puntata.

Force (F4)

Consente di forzare il valore della variabile correntemente puntata. La finestra di dialogo richiederà l'introduzione di un valore decimale, binario (con carattere terminale B) oppure esadecimale (con carattere terminale H).
Il valore da forzare deve rientrare nel campo di validità della variabile puntata; in caso di errore tuttavia il software provvederà a segnalarlo.
Si consideri che l'operazione di forzatura di un valore nella RAM dati è possibile solo se non sono presenti nel programma istruzioni che fanno altrettanto sulle stesse variabili: in questo caso prevale l'azione svolta dal programma della macchina. Analoga situazione si verifica per tutte le risorse già forzate dal sistema operativo della Logica come i bytes di ingresso.

Set (F5)

Forza allo stato logico "1" il bit attualmente puntato della variabile selezionata.
La forzatura di un bit della memoria dati è possibile solo se non esistono istruzioni nel programma della Logica che lo forzano a loro volta o se il bit non viene definito nel suo valore dal sistema operativo (es: bit di ingresso).

Res (F6)

Forza allo stato logico "0" il bit attualmente puntato della variabile selezionata.
La forzatura di un bit della memoria dati è possibile solo se non esistono istruzioni nel programma della Logica che lo forzano a loro volta o se il bit non viene definito nel suo valore dal sistema operativo (es: bit di ingresso).

Help (F7)

Visualizza un sommario elenco delle funzioni svolte dai tasti di posizionamento sul listato e sulla finestra di monitor.

Search (F8)

Consente la ricerca in avanti di una stringa di testo all'interno del listato programma visualizzato.

Esc

Il tasto Escape consente di lasciare il menu di monitor e di ritornare al menu principale dell'ambiente ICL51.

Protocollo di comunicazione

Informazioni generali

L'interfaccia RS232 della scheda MASTER consente tutte le operazioni di comunicazione con un Personal Computer o con qualsiasi dispositivo in grado di gestire un'interfaccia di questo tipo.

In entrambi i casi è necessario seguire alcune regole per controllare il flusso di informazioni scambiate tra la Logica e il dispositivo esterno: queste regole costituiscono il protocollo di comunicazione della scheda MASTER.

Nel seguito illustreremo queste regole facendo riferimento per semplicità ad un collegamento tra Logica MASTER e Personal Computer, riportando parti esemplificative di programma scritte in linguaggio BASIC.

Lo scopo di tale sezione è quella di mettere il Programmatore nelle condizioni di far colloquiare la scheda MASTER con un Computer, rendendo accessibili in lettura e scrittura sia il programma di funzionamento della Logica sia tutti i dati di lavoro memorizzati nella memoria RAM.

Con le informazioni che riporteremo è possibile scrivere dei programmi con linguaggi ad alto livello, operanti sul Computer, che supervisionano il funzionamento del programma su MASTER, consentendo di acquisire dati in tempo reale durante il funzionamento della macchina; inoltre è possibile comandare la macchina stessa forzando delle informazioni dal Computer alla Logica.

Queste operazioni di gestione mediante Computer possono essere temporanee, collegando cioè provvisoriamente al MASTER un Computer solo quando è necessario, oppure fisse, lasciando il Computer sempre connesso alla Logica. Quest'ultimo caso si riferisce all'utilizzo di un Computer Industriale come parte integrante della macchina e le cui funzioni possono essere sia di supervisione del processo automatico sia di controllo dello stesso. Caso limite di questa applicazione è il controllo del processo completamente effettuato dal Computer, utilizzando la logica esclusivamente come interfaccia di INPUT/OUTPUT verso la macchina.

Apertura del canale di comunicazione

Prima di ogni operazione di comunicazione con la Logica è necessario aprire, nel programma su Computer, il canale di comunicazione seriale RS232 dello stesso.

Supponendo di aver connesso la Logica alla porta seriale COM1 del Computer, l'istruzione BASIC che consente di fare questo è la seguente:

```
OPEN "COM1:9600,N,8,1" FOR RANDOM AS #1
```

Questa istruzione apre il file di comunicazione numero 1 sulla porta di comunicazione seriale COM1. La scelta del numero di file è arbitraria e non ha nessun riferimento con il numero di porta seriale; tutte le operazioni di accesso alla COM1 dovranno nel seguito essere ovviamente effettuate su tale numero di file.

Dall'istruzione di apertura si deduce che i parametri di comunicazione seriale sono:

Baud rate:	9600
Parità:	NESSUNA
Numero bits:	8
Bit di stop:	1

Questi parametri sono definiti all'interno del sistema operativo della Logica e devono essere rigorosamente rispettati.

Terminata la comunicazione occorre eseguire sul Computer l'istruzione di chiusura del canale di comunicazione:

```
CLOSE #1
```

L'utilizzo delle istruzioni di apertura e chiusura del canale di comunicazione RS232 è lasciato completamente all'arbitrio del Programmatore; niente vieta la continua apertura e chiusura del canale a seconda che ci sia necessità o meno di comunicare con la Logica.

Gestione degli errori di comunicazione

Si consiglia di inserire nel programma operante su Computer una routine di gestione degli errori di comunicazione; la mancanza di una tale routine non pregiudica il colloquio con la Logica ne il suo funzionamento, ma può causare un blocco del programma su Computer o l'uscita a sistema operativo dello stesso.

In BASIC la gestione degli errori può essere effettuata ponendo in testa al programma la seguente istruzione:

```
ON ERROR GOTO GestioneErrore
```

Al termine del programma principale inserire invece la routine di gestione dell'eventuale errore; di seguito riportiamo un esempio di tale routine:

```
GestioneErrore:

    SELECT CASE ERR
    CASE 24
        Errore$ = "Errore di timeout"
    CASE 52, 57
        Errore$ = "Errore di comunicazione"
    CASE 68
        Errore$ = "Porta non disponibile"
    CASE 69
        Errore$ = "Overflow nel buffer"
    ------------------------------------------------
    ------------------------------------------------
    END SELECT

    BEEP
    PRINT Errore$
    SLEEP
    RESUME NEXT
```

Queste righe di programma sono state riportate esclusivamente a titolo di esempio; sarà compito del Programmatore approfondire l'argomento facendo riferimento alle appropriate sezioni del manuale del linguaggio di programmazione utilizzato.

Il tipo di gestione degli errori dipende fortemente dall'intera struttura del programma sviluppato; infatti la rivelazione degli errori dipende dal tipo e dal modo di utilizzo delle istruzioni adoperate nella comunicazione. Il ritorno dopo un errore (es: RESUME NEXT) dipende da come è stato strutturato il programma e dalle specifiche dello stesso.

In alcuni casi è conveniente gestire l'errore di TIMEOUT di comunicazione anche mediante il TIMER di Watchdog del BASIC:

```
ON TIMER(3) GOSUB TimeOut

-----------------------------------------------
-----------------------------------------------

Errore = FALSE
TIMER ON

DO UNTIL LOC(1) = 4
        IF Errore THEN
                EXIT LOOP
        END IF
LOOP

TIMER OFF

IF NOT Errore THEN
        StringaRicevuta$ = INPUT$(4, #1)
END IF

-----------------------------------------------
-----------------------------------------------



TimeOut:

        Errore = TRUE
        BEEP
        PRINT "Errore di timeout"
        SLEEP

RETURN
```

In genere nel programma di comunicazione gli errori si possono verificare durante un'apertura del canale di comunicazione; oppure si possono incontrare delle situazioni di attesa (mediante "polling") di dati in ricezione dalla Logica che non terminano mai a causa di un imprevisto sulla linea di comunicazione. In questo caso l'utilizzo dell'istruzione TIMER, come nell'esempio precedente, riesce a terminare ugualmente la ricezione continuando l'esecuzione del programma.

In alternativa alla gestione "polling" delle ricezioni dalla Logica, si può utilizzare in BASIC il gestore degli eventi ON COM(1) per eseguire le istruzioni di risposta alla seriale non appena arrivano i dati al Computer.

I comandi di comunicazione

Tutte le operazioni di comunicazione tra Computer e Logica MASTER avvengono mediante l'invio da parte del Computer di un BYTE DI COMANDO alla Logica stessa. E' quindi il Computer ad avere il controllo della situazione della comunicazione; la Logica si limita semplicemente ad eseguire i comandi inviati dal Computer.

Ovviamente la Logica deve continuare il suo compito di elaborazione del programma macchina caricato sulla memoria; per capire il meccanismo della comunicazione su RS232 diciamo che ogni volta che il Computer invia alla Logica un pacchetto di bytes costituito in testa dal byte di comando e successivamente dai parametri del comando stesso, la Logica esegue una routine di INTERRUPT che salva in un buffer interno i bytes del pacchetto.

Quando la Logica termina la scansione del programma macchina, essa esegue il comando del Computer; al termine di questo riprende la scansione del programma o nel caso di certi comandi rimane in una condizione di STOP.

In questo modo tutte le operazioni di comunicazione con il Computer sono sincronizzate con la scansione del programma macchina della Logica e non può essere eseguita più di una operazione di comunicazione per ogni scansione. Inoltre occorre considerare la velocità di elaborazione del programma sul Computer; se la velocità di controllo del Computer è inferiore a quella di esecuzione di un ciclo di scansione della Logica, è ovvio che un'operazione di comunicazione potrà avvenire ogni due o più cicli di scansione. Questo tuttavia non porta alcun ritardo apprezzabile per una supervisione della Logica mediante il Computer, ma neanche per il controllo attivo da parte del programma su Computer delle funzioni della macchina.

Per tutti quei comandi che non prevedono risposta da parte della Logica, occorre far attenzione a non inviare consecutivamente due pacchetti distinti, poiché la Logica non avrebbe tempo di elaborarli. Per alcuni tipi di comando è consigliabile quindi introdurre un ritardo software sul Computer che consenta alla Logica di elaborare il pacchetto appena inviato; la necessità o meno di tale ritardo dipende dal tipo di operazione richiesta come si vedrà nel seguito.

Una routine BASIC che ritarda di un numero multiplo di 55ms è la seguente:

```
SUB AttesaNx55ms (N)

        FOR I= 0 TO N
              SOUND 32767, 1
        NEXT I

END SUB
```

Tale routine sfrutta l'istruzione di suono con frequenza oltre l'udibile e durata 1/18.2=0.05494 secondi e determina una attesa del programma pari a N volte 55ms.

Prima di procedere ad un'analisi dettagliata dei comandi di comunicazione, riportiamo nella Tabella 8 la mappatura della memoria RAM dati; l'utilizzo di tale tabella è fondamentale nelle operazioni di comunicazione in quanto tutti i comandi di lettura e scrittura dati fanno riferimento agli indirizzi assoluti delle variabili.

La memoria RAM è suddivisa in alcune aree assegnate ciascuna ad un particolare tipo di variabile o di risorsa del programma. Per esempio la prima area di 4096 bytes è utilizzata come memoria di appoggio per tutti gli I/O sia del MASTER che di tutti i 31 SLAVES; le successive due aree sono utilizzate per i 1024 bytes della memoria non ritentiva e per i 1024 bytes della memoria ritentiva. Segue l'area utilizzata per implementare i contatori software, l'area per i generatori di impulso su fronte ed infine l'area occupata da un insieme vario di risorse di piccola dimensione.

Queste tabelle permettono di individuare, per ogni variabile del programma, il corrispondente indirizzo esadecimale; tale indirizzo, composto da quattro cifre esadecimali, andrà separato in due parti (LOW ed HIGH) ciascuna a due cifre. Ognuna di queste parti costituisce il valore di un byte da inviare in modo opportuno alla Logica, per specificare quale variabile si intende leggere o scrivere.

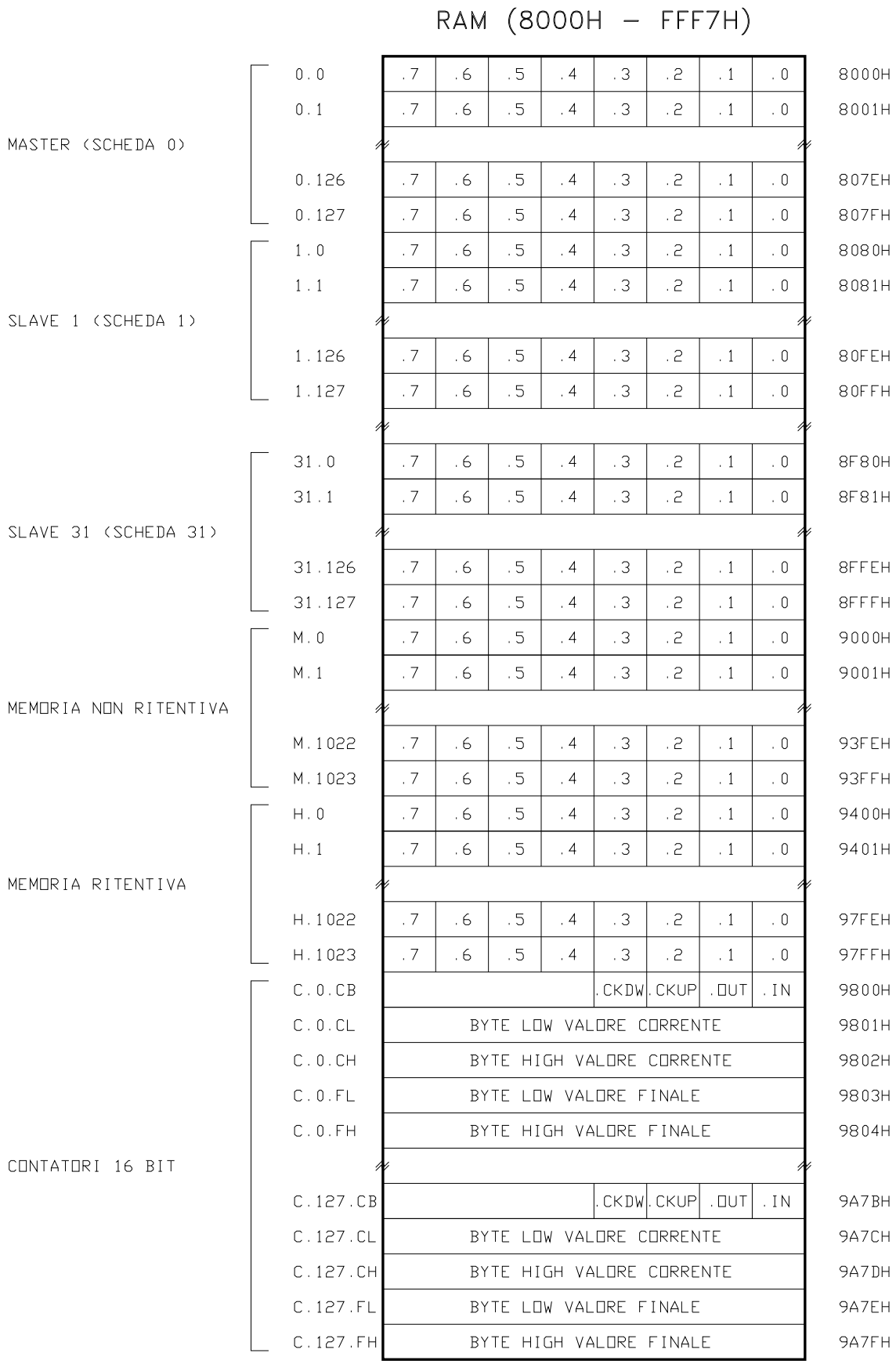


Tabella 8.a. Mappatura della RAM dati

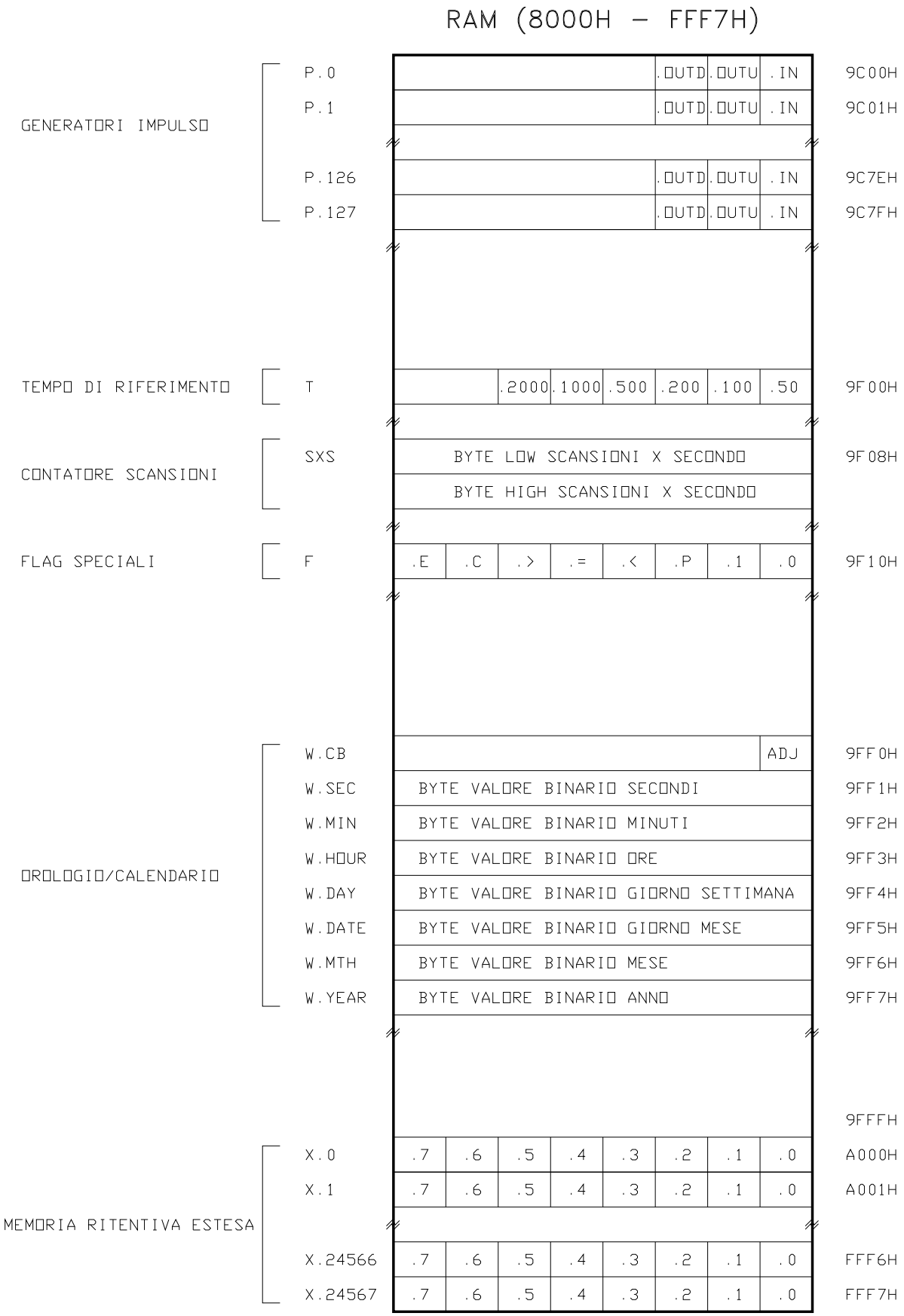


Tabella 8.b. Mappatura della RAM dati

STOP

Arresta l'esecuzione del programma utente

Codifica comando

(1)

Risposta dalla Logica

Nessuna.

Descrizione

Il comando STOP consente al PC di fermare la scansione del programma utente ponendo la Logica in uno stato di inattività. La memoria RAM viene azzerata ad esclusione dei bytes ritentivi; viene costantemente eseguito l'appello di tutti gli SLAVES con relativo aggiornamento dei bytes riservati. La Logica rimane in tale stato fino al successivo invio del comando RUN.

Esempio

In BASIC la seguente istruzione forza la Logica MASTER nello stato di STOP:

```
PRINT #1, CHR$(1);
```

Vedi anche

RUN

RUN

Inizia l'esecuzione del programma utente

Codifica comando

(10)

Risposta dalla Logica

Nessuna.

Descrizione

Il comando RUN consente al PC di riavviare la scansione del programma utente. All'accensione dell'alimentazione, se viene trovato presente un programma utente nella memoria FLASH-EPROM, la Logica entra automaticamente nello stato di RUN; è possibile tuttavia forzare, all'accensione, la Logica nello stato di STOP mediante l'inserimento dell'apposito switch o mediante il comando F9 del menu di trasferimento (a seconda del modello di Logica).

Attenzione: normalmente dopo un comando di RUN seguono altri pacchetti di comando, poiché ciò che più interessa è la comunicazione in fase di esecuzione del programma. Prima di inviare altri pacchetti di comando, si consiglia di attendere alcune decine di millisecondi per consentire alla Logica di iniziare completamente la sua attività.

Esempio

In BASIC la seguente istruzione forza la Logica MASTER nello stato di RUN:

```
PRINT #1, CHR$(10);
```

Vedi anche

STOP

STATUS

Legge il byte di comando correntemente presente sulla Logica

Codifica comando

(250)

Risposta dalla Logica

(Byte)

1 byte corrispondente allo stato corrente del byte di comando sulla Logica.

Descrizione

Il comando STATUS legge il valore presente nel primo byte del buffer di ricezione del pacchetto di comando. In questo modo è possibile conoscere il byte di testa, corrispondente al codice comando, dell'ultimo pacchetto inviato.
Lo scopo di tale comando è quello di verificare se la Logica si trova nello stato di STOP o nello stato di RUN quando questa condizione non è nota.

Esempio

In BASIC la seguente parte di programma richiede alla Logica MASTER il valore del byte di comando precedentemente inviato:

```
PRINT #1, CHR$(250);

DO UNTIL LOC(1) = 1: LOOP

Stato$ = INPUT$(1, #1)
```

MONITOR1

Legge il valore corrente di una variabile ad 1 byte

Codifica comando

(200) + (Address_Low) + (Address_High)

Risposta dalla Logica

(Byte)

1 byte corrispondente al valore corrente della variabile ad 1 byte.

Descrizione

Il comando MONITOR1 consente di leggere il valore corrente della variabile ad 1 byte con indirizzo assoluto di RAM indicato dalla word composta dai due bytes Address_High e Address_Low.
In risposta tale pacchetto comando di 3 bytes, la Logica invia 1 byte corrispondente al valore corrente della variabile ad 1 byte richiesta; successivamente alla ricezione della risposta è possibile inviare un'altro pacchetto di comando.

Esempio

In BASIC la seguente parte di programma legge il valore della variabile ad 1 byte H.1022:

```
PRINT #1, CHR$(200) + CHR$(254) + CHR$(151);

DO UNTIL LOC(1) = 1: LOOP

Valore = ASC(INPUT$(1, #1))
```

Vedi anche

MONITOR2, MONITOR4

MONITOR2

Legge il valore corrente di una variabile a 2 bytes

Codifica comando

(201) + (Address_Low) + (Address_High)

Risposta dalla Logica

(Byte_0) + (Byte_1)

2 bytes corrispondenti al valore corrente della variabile a 2 bytes.

Descrizione

Il comando MONITOR2 consente di leggere il valore corrente della variabile a 2 bytes con indirizzo assoluto di RAM indicato dalla word composta dai due bytes Address_High e Address_Low.
In risposta tale pacchetto comando di 3 bytes, la Logica invia i 2 bytes corrispondenti al byte basso ed al byte alto del valore corrente della variabile a 2 bytes richiesta; successivamente alla ricezione della risposta è possibile inviare un'altro pacchetto di comando.

Esempio

In BASIC la seguente parte di programma legge il valore della variabile a 2 bytes H.0:

```
PRINT #1, CHR$(201) + CHR$(0) + CHR$(148);

DO UNTIL LOC(1) = 2: LOOP

ValoreLow = ASC(INPUT$(1, #1))
ValoreHigh = ASC(INPUT$(1, #1))

Valore = 256 * ValoreHigh + ValoreLow
```

Vedi anche

MONITOR1, MONITOR4

MONITOR4

Legge il valore corrente di una variabile a 4 bytes

Codifica comando

(202) + (Address_Low) + (Address_High)

Risposta dalla Logica

(Byte_0) + (Byte_1) + (Byte_2) + (Byte_3)

4 bytes corrispondenti al valore corrente della variabile a 4 bytes.

Descrizione

Il comando MONITOR4 consente di leggere il valore corrente della variabile a 4 bytes con indirizzo assoluto di RAM indicato dalla word composta dai due bytes Address_High e Address_Low.
In risposta tale pacchetto comando di 3 bytes, la Logica invia i 4 bytes, ad iniziare dal byte più basso fino al byte più alto, del valore corrente della variabile a 4 bytes richiesta; successivamente alla ricezione della risposta è possibile inviare un'altro pacchetto di comando.

Esempio

In BASIC la seguente parte di programma legge il valore della variabile a 4 bytes H.0:

```
PRINT #1, CHR$(202) + CHR$(0) + CHR$(148);

DO UNTIL LOC(1) = 4: LOOP

Valore0 = ASC(INPUT$(1, #1))
Valore1 = ASC(INPUT$(1, #1))
Valore2 = ASC(INPUT$(1, #1))
Valore3 = ASC(INPUT$(1, #1))

Valore = 16777216 * Valore3 + 65536 * Valore2 + 256 * Valore1 + Valore0
```

Vedi anche

MONITOR1, MONITOR2

FORCE1

Scrive un valore su una variabile ad 1 byte

Codifica comando

(210) + (Address_Low) + (Address_High) + (Byte)

Risposta dalla Logica

Nessuna.

Descrizione

Il comando FORCE1 consente di scrivere un determinato valore sulla variabile ad 1 byte con indirizzo assoluto di RAM indicato dalla word composta dai due bytes Address_High e Address_Low.
La logica non risponde in alcun modo a tale pacchetto comando di 4 bytes; si consiglia di inserire un ritardo fisso dopo l'invio del pacchetto comando (pari almeno al tempo richiesto da un ciclo di scansione) allo scopo di consentire alla Logica l'esecuzione del comando.

Esempio

In BASIC la seguente parte di programma scrive il valore 78 sulla variabile ad 1 byte H.1022:

```
PRINT #1, CHR$(210) + CHR$(254) + CHR$(151) + CHR$(78);
```

Vedi anche

FORCE2, FORCE4

FORCE2

Scrive un valore su una variabile a 2 bytes

Codifica comando

(211) + (Address_Low) + (Address_High) + (Byte_0) + (Byte_1)

Risposta dalla Logica

Nessuna.

Descrizione

Il comando FORCE2 consente di scrivere un determinato valore sulla variabile a 2 bytes con indirizzo assoluto di RAM indicato dalla word composta dai due bytes Address_High e Address_Low.
La logica non risponde in alcun modo a tale pacchetto comando di 5 bytes; si consiglia di inserire un ritardo fisso dopo l'invio del pacchetto comando (pari almeno al tempo richiesto da un ciclo di scansione) allo scopo di consentire alla Logica l'esecuzione del comando.

Esempio

In BASIC la seguente parte di programma scrive il valore 32578=256*127+66 sulla variabile a 2 bytes H.0:

```
PRINT #1, CHR$(211) + CHR$(0) + CHR$(148) + CHR$(66) + CHR$(127);
```

Vedi anche

FORCE1, FORCE4

FORCE4

Scrive un valore su una variabile a 4 bytes

Codifica comando

(212) + (Address_Low) + (Address_High) + (Byte_0) + (Byte_1) + (Byte_2) + (Byte_3)

Risposta dalla Logica

Nessuna.

Descrizione

Il comando FORCE4 consente di scrivere un determinato valore sulla variabile a 4 bytes con indirizzo assoluto di RAM indicato dalla word composta dai due bytes Address_High e Address_Low.
La logica non risponde in alcun modo a tale pacchetto comando di 7 bytes; si consiglia di inserire un ritardo fisso dopo l'invio del pacchetto comando (pari almeno al tempo richiesto da un ciclo di scansione) allo scopo di consentire alla Logica l'esecuzione del comando.

Esempio

In BASIC la seguente parte di programma scrive il valore 2355455890 (scomposto nei suoi 4 bytes corrisponde, iniziando dal byte alto, a 140, 101, 103, 146) sulla variabile a 4 bytes H.0:

```
PRINT #1, CHR$(212) + CHR$(0) + CHR$(148) + CHR$(146) + CHR$(103) + CHR$(101) + CHR$(140);
```

Vedi anche

FORCE1, FORCE2

RESBIT

Forza al valore “0” logico uno o più bits di un byte

Codifica comando

(220) + (Mask) + (Address_Low) + (Address_High)

Risposta dalla Logica

Nessuna.

Descrizione

Il comando RESBIT consente di forzare a “0” uno più bits appartenenti al byte con indirizzo assoluto di RAM indicato dalla word composta dai due bytes Address_High e Address_Low. Per indicare quali bits del byte vanno resettati occorre fornire nel pacchetto di comando il valore del byte di maschera (Mask); questo byte deve avere al valore logico “1” tutti e solo i bits corrispondenti a quelli da resettare.
La logica non risponde in alcun modo a tale pacchetto comando di 4 bytes; si consiglia di inserire un ritardo fisso dopo l’invio del pacchetto comando (pari almeno al tempo richiesto da un ciclo di scansione) allo scopo di consentire alla Logica l’esecuzione del comando.

Esempio

In BASIC la seguente parte di programma scrive il valore logico “0” sui bits 0 e 5 del byte H.1022 ossia resetta i bits H.1022.0 e H.1022.5:

```
PRINT #1, CHR$(220) + CHR$(33) + CHR$(254) + CHR$(151);
```

Vedi anche

SETBIT

SETBIT

Forza al valore “1” logico uno o più bits di un byte

Codifica comando

(221) + (Mask) + (Address_Low) + (Address_High)

Risposta dalla Logica

Nessuna.

Descrizione

Il comando SETBIT consente di forzare a “1” uno più bits appartenenti al byte con indirizzo assoluto di RAM indicato dalla word composta dai due bytes Address_High e Address_Low. Per indicare quali bits del byte vanno settati occorre fornire nel pacchetto di comando il valore del byte di maschera (Mask); questo byte deve avere al valore logico “1” tutti e solo i bits corrispondenti a quelli da settare.
La logica non risponde in alcun modo a tale pacchetto comando di 4 bytes; si consiglia di inserire un ritardo fisso dopo l’invio del pacchetto comando (pari almeno al tempo richiesto da un ciclo di scansione) allo scopo di consentire alla Logica l’esecuzione del comando.

Esempio

In BASIC la seguente parte di programma scrive il valore logico “1” sui bits 1 e 4 del byte H.1022 ossia setta i bits H.1022.1 e H.1022.4:

```
PRINT #1, CHR$(221) + CHR$(18) + CHR$(254) + CHR$(151);
```

Vedi anche

RESBIT

BACKUP

Legge i valori dei bytes di un'area di memoria RAM

Codifica comando

(120) + (Address_Low) + (Address_High) + (Number_Low) + (Number_High)

Risposta dalla Logica

(Byte_0) + (Byte_1) + ... + (Byte_N-2) + (Byte_N-1)

N bytes corrispondenti al valore corrente dell'area di RAM con inizio dall'indirizzo Address.

Descrizione

Il comando BACKUP consente di leggere il valore corrente di tutti i bytes di un'area di memoria RAM ad iniziare dall'indirizzo assoluto indicato dalla word composta dai due bytes Address_High e Address_Low, per un totale di Number bytes.
In risposta tale pacchetto comando di 5 bytes, la Logica invia tutti i bytes corrispondenti al valore corrente dell'area di memoria richiesta; al termine di un comando di questo tipo, la Logica entra automaticamente nello stato di STOP.

Esempio

In BASIC la seguente parte di programma legge l'area di memoria dal byte H.0 al byte H.399 (in totale 400=256*1+144 bytes):

```
PRINT #1, CHR$(120) + CHR$(0) + CHR$(148) + CHR$(144)+ CHR$(1);

DO UNTIL LOC(1) = 400: LOOP

FOR I = 1 TO 400
      Valore(I) = ASC(INPUT$(1, #1))
NEXT I
```

Vedi anche

RESTORE

RESTORE

Scrive una sequenza di valori su un'area di memoria RAM

Codifica comando

(130) + (Address_Low) + (Address_High) + (Number_Low) + (Number_High)

Risposta dalla Logica

(130)

Conferma della disponibilità da parte della Logica a ricevere la sequenza di valori.

Invio dei valori

(Byte_0) + (Byte_1) + ... + (Byte_N-2) + (Byte_N-1)

N bytes da scrivere nell'area di RAM con inizio dall'indirizzo Address.

Descrizione

Il comando RESTORE consente di scrivere dei valori su tutti i bytes di un'area di memoria RAM ad iniziare dall'indirizzo assoluto indicato dalla word composta dai due bytes Address_High e Address_Low, per un totale di Number bytes.
In risposta tale pacchetto comando di 5 bytes, la Logica invia un byte di valore 130 (eco del comando) a conferma della sua disponibilità a ricevere i dati. A questo punto il PC può inviare consecutivamente la sequenza dei valori; al termine di un comando di questo tipo, la Logica entra automaticamente nello stato di STOP.

Esempio

In BASIC la seguente parte di programma scrive sull'area di memoria dal byte H.0 al byte H.399 (in totale 400=256*1+144 bytes) una sequenza di valori:

```
PRINT #1, CHR$(130) + CHR$(0) + CHR$(148) + CHR$(144)+ CHR$(1);

DO UNTIL INPUT$(LOC(1), #1) = CHR$(130): LOOP

FOR I = 1 TO 400
      PRINT #1, CHR$(Valore(I));
NEXT I
```

Vedi anche

BACKUP

UPLOAD

Legge i valori dei bytes di un'area di memoria programma FLASH-EPROM

Codifica comando

(110) + (Start_Page) + (Pages_Number)

Risposta dalla Logica

(Byte_0) + (Byte_1) + ... + (Byte_N-2) + (Byte_N-1) N = 256 * Pages_Number

N bytes corrispondenti all'area di FLASH-EPROM con inizio dalla pagina Start_Page.

Descrizione

Il comando UPLOAD consente di leggere i valori dei bytes di un'area di memoria programma FLASH-EPROM ad iniziare dalla pagina Start_Page, per un totale di Pages_Number pagine. Per pagina si intende un blocco contiguo di 256 bytes e l'indirizzo di pagina è un numero nel campo 0÷255; con il comando UPLOAD è quindi possibile leggere delle aree che iniziano e terminano esattamente su multipli interi di blocchi di 256 bytes.
In risposta tale pacchetto comando di 3 bytes, la Logica invia tutti i bytes corrispondenti all'area di memoria programma richiesta; il numero di bytes ricevuti è pari ad un multiplo intero (valore di Pages_Number) di 256. Al termine di un comando di questo tipo, la Logica entra automaticamente nello stato di STOP.

Esempio

In BASIC la seguente parte di listato legge l'area di memoria programma dalla pagina 4 alla pagina 23 (in totale 20 pagine = 5120 bytes):

```
PRINT #1, CHR$(110) + CHR$(4) + CHR$(20);

Area$ = STRING$(5120, CHR$(0))

NumeroRx = 0

DO WHILE NumeroRx < 5120
      DimBuffer = LOC(1)
      MID$(Area$, 1 + NumeroRx) = INPUT$(DimBuffer, #1)
      NumeroRx = NumeroRx + DimBuffer
LOOP
```

DOWNLOAD

Programma la memoria FLASH-EPROM

Codifica comando

(100) + (Start_Page) + (Pages_Number)

Risposta dalla Logica

(100)	se la cancellazione è stata effettuata correttamente.
(15)	se la cancellazione non è stata effettuata correttamente.

Invio dei valori

(Byte_0) + (Byte_1) + ... + (Byte_N-2) + (Byte_N-1) N = 256 * Pages_Number

N bytes corrispondenti all'area di FLASH-EPROM con inizio dalla pagina Start_Page.

Risposta dalla Logica

(0)	se la programmazione è stata effettuata correttamente.
(255)	se la programmazione non è stata effettuata correttamente.

Descrizione

Il comando DOWNLOAD consente di programmare l'intera memoria FLASH-EPROM con il codice del programma utente. La programmazione della memoria FLASH-EPROM avviene normalmente tramite la funzione di DownLoad dell'ambiente di sviluppo; tuttavia questo comando è stato ugualmente incluso nell'elenco dei comandi di protocollo al fine di chiarire maggiormente come avviene la programmazione della memoria FLASH-EPROM.

Prima di ogni programmazione, la memoria deve essere totalmente cancellata; questo avviene elettricamente sulla scheda della Logica senza che sia necessario estrarre il dispositivo dal suo zoccolo. Successivamente alla totale cancellazione di tutte le locazioni, occorre programmare tutti i bytes necessari poichè non è possibile programmarne solo una parte alla volta.
La programmazione di ogni byte della memoria richiede un tempo inferiore a quello necessario alla comunicazione di un byte mediante RS232 a 9600 Baud; per questo la programmazione dei singoli bytes avviene contemporaneamente al trasferimento sulla seriale.

Le fasi della programmazione sono le seguenti. Il PC invia un pacchetto comando composto da tre bytes nel quale sono indicati la pagina iniziale ed il numero di pagine da programmare.

Alla ricezione del pacchetto da parte della Logica questa provvede all'immediata cancellazione dell'intera memoria FLASH-EPROM; terminata questa operazione invia al PC un byte di conferma della cancellazione. Questo byte vale 100 se l'operazione di cancellazione ha avuto buon esito, oppure vale 15 se non è stato possibile cancellare la memoria per un difetto di questa. In tal caso il led di RUN presente sulla scheda della Logica lampeggia con il codice di errore 2 lampeggi/pausa per un totale di 10 segnalazioni; una simile situazione richiede la sostituzione del dispositivo di memoria.

Se la cancellazione è stata effettuata correttamente (risposta con il byte di valore 100), si può procedere a trasferire dal PC alla Logica tutti i bytes delle pagine da programmare. Durante la programmazione il led di RUN lampeggia, cambiando di stato ad ogni pagina programmata nella memoria.
Nella fase di programmazione la Logica invia al PC un secondo byte di conferma il cui valore dipende dall'esito dell'operazione; in particolare il byte vale 0 se la programmazione è stata effettuata tutta corretamente, oppure vale 255 se è stato riscontrato un errore nel programmare una o più celle di memoria. In tal caso sulla Logica viene segnalato tale errore mediante il codice di errore 3 lampeggi/pausa per un totale di 10 segnalazioni.
Più precisamente, se viene riscontrato un errore di programmazione, questo viene subito segnalato con l'invio del byte 255, senza attendere la fine della programmazione, mentre in caso di programmazione effettuata l'invio del byte 0 avviene al termine della stessa.

Esempio

In BASIC la seguente parte di listato esegue la programmazione delle prime 64 pagine della memoria FLASH-EPROM:

```
PRINT #1, CHR$(100) + CHR$(0) + CHR$(64)

DO
        SELECT CASE INPUT$(LOC(1), #1)
        CASE CHR$(100)
                EXIT DO
        CASE CHR$(15)
                BEEP
                PRINT "Errore di cancellazione"
                SLEEP
                RETURN
        END SELECT
LOOP

FOR I = 0 TO 63
        PRINT #1, MID$(Codice$, 1 + I * 256, 256);
        IF INPUT$(LOC(1), #1) = CHR$(255) THEN
                BEEP
                PRINT "Errore di programmazione"
                SLEEP
                RETURN
        END IF
NEXT I
```

```
DO UNTIL LOC(1) = 1: LOOP

IF INPUT$(1, #1) = CHR$(0) THEN
        PRINT “Programmazione effettuata”
END IF
```