ISTITUTO PROFESSIONALE DI STATO PER L'INDUSTRIA L'ARTIGIANATO

63039 SAN BENEDETTO DEL TRONTO (Ascoli Piceno)

Classe 5A-5B T.I.E.N. Anno Scolastico 2003/2004

MICROCONTROLLORI ST6

SISTEMI, AUTOMAZIONE E ORGANIZZAZIONE DELLA PRODUZIONE

1.1 Introduzione

I microcontrollori sostituiscono in misura sempre maggiore i tradizionali circuiti di regolazione, misura, e controllo. Essi sono caratterizzati da una uguale struttura hardware: CPU, ROM, RAM, PORTE.

Tramite programmazione, i micro si trasformano in integrati dedicati o specializzati alla gestione di una sola applicazione, ne consegue che l'attività maggiore di progettazione non è hardware, bensì software.

Infatti, il costo delle ore di lavoro necessarie al programmatore per "specializzare" il micro, ovvero per scrivere il programma, risulta decisamente superiore al costo del micro stesso, ecco quindi l'esigenza di razionalizzare il lavoro seguendo un criterio di programmazione ben preciso.

1. **Fattibilità**: verifichiamo se il set d'istruzioni e le risorse presenti nel micro consentono di soddisfare tutte le richieste dell'applicazione in esame.
2. **Schema elettrico**: stabiliamo i collegamenti tra il micro e il mondo esterno, associamo ad ogni porta la relativa funzione e il modo di funzionamento (ad esempio: ingresso, uscita, ingresso analogico, ecc.)
3. **Tabella di verità:** associamo ad ogni porta il relativo compito, ad esempio: PA0 è un ingresso con resistore pull-up collegato ad un interruttore e risulta attivo quando assume il valore di "0" logico, ecc.
4. **Flow_Chart**: realizziamo lo schema a blocchi (flow-chart) del programma, pianificando la sequenza delle istruzioni in funzione di eventi interni o esterni, si trova l'algoritmo
5. **Programma**: tramite il linguaggio assembler, rispettando il set di istruzioni del micro, traduciamo il flow-chart in programma, creiamo il file sorgente (nome_file.asm)
6. **Assemblaggio**: tramite il software **AST6.EXE** trasformiamo il file sorgente in formato opcode (codice oggetto)
7. **Memorizzazione programma**: tramite lo Starter Kit si memorizza il programma in formato **opcode** nella memoria EPROM /OTP del microcontrollore, ove possibile si effettua l'emulazione, cioè si simula attraverso lo starter kit il funzionamento del software.
8. **Collaudo**: si effettua il collaudo finale del progetto, si inserisce il micro programmato nel circuito progettato, comprensivo di tutte le sue interfacce (clock, reset, alimentazione, I/O) e si verifica se il funzionamento corrisponde alle specifiche richieste dal progetto.

1.2 Algoritmo – Flow-Chart

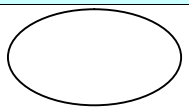
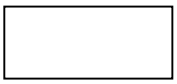
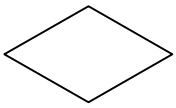
Si definisce **algoritmo** la sequenza di operazioni univocamente interpretabili, eseguite in sequenza, che trasforma i dati iniziali nel risultato richiesto.

L'insieme dei valori permessi ai dati in ingresso si definisce **dominio** dell'algoritmo, mentre l'insieme dei valore che possono assumere i dati in uscita rappresenta il **condominio**.

Un programma prende origine comunque da un algoritmo che viene descritto in modo sintetico utilizzando un tipo di rappresentazione che non dà alito ad ambiguità nelle operazioni da eseguire, la maniera più comune di rappresentare un algoritmo è quella dei diagrammi di flusso o **flow-chart**.

Il flow-chart rappresenta l'algoritmo graficamente mediante una lista di istruzioni inserite in appositi simboli grafici, ogni simbolo viene collegato al successivo con delle linee orientate che danno luogo così alla sequenza delle operazioni da effettuare.

Il flow-chart o diagramma a blocchi da un visione immediata dell'intero algoritmo mettendo in evidenza la sequenza di esecuzione delle varie operazioni.
In tabella sono rappresentati i simboli principali per la stesura di un flow-chart.

Simbolo	Descrizione
	Inizio di un programma o di una subroutine, si scrive all'interno il nome del programma o della subroutine, viene anche utilizzato per indicare fine programma o fine subroutine
	Elaborazione dati, esempio calcolo, oppure manipolazione dati, operazione incondizionata
	Blocco decisione, esso contiene una domanda, in funzione della risposta viene deviato il corso del flow-chart, tale blocco è caratterizzato da una o più linee d'ingresso e da due diverse linee di uscita.
	Chiamata subroutine
	Operazione Input /Output
	Flusso logico

Per realizzare un flow-chart scriviamo una o più azioni/comandi all'interno di figure geometriche (ellissi, rettangoli, rombi, ecc) e colleghiamo le figure tra loro con delle linee, aggiungiamo ad ogni linea di congiunzione una freccia, essa indica la sequenza degli eventi, ovvero la sequenza utilizzata dal micro per processare le istruzioni. (in fig.1, un esempio di flow-chart)

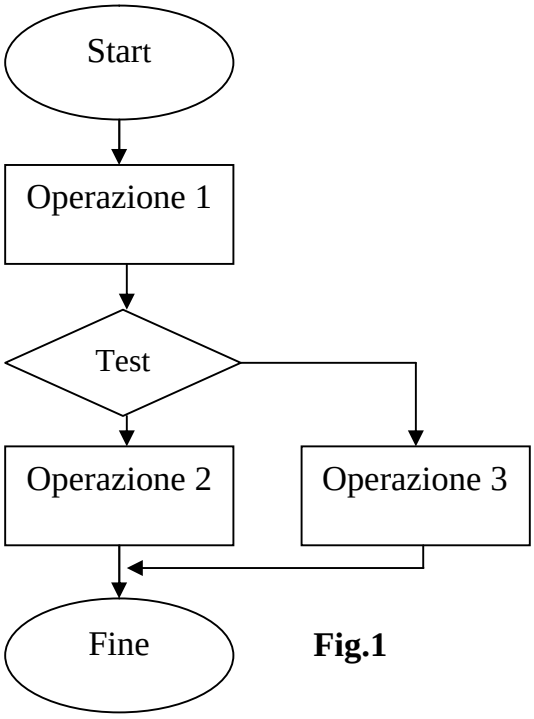


Fig.1

Durante la stesura del flow-chart distinguiamo due diversi blocchi:

1. programma principale detto "***main program***"
2. uno o più programmi secondari denominati "***subroutine***".

Per meglio localizzare eventuali errori si stesura del programma è consigliabile sviluppare e provare le subroutine singolarmente, e solo quando queste ultime funzioneranno correttamente sarà possibile unirle al main program per testare il programma completo.
È buona norma, inoltre, realizzare le subroutine e catalogarle formando una biblioteca di programmi secondari.
Per meglio comprendere questa affermazione facciamo un piccolo esempio, supponiamo di dover realizzare un contapezzi e di avere come uscita un display, dovremo quindi realizzare

una routine che visualizzi un numero sul display. Procedere in modo razionale significa rendere questa routine standard, documentandola correttamente e salvandola nella biblioteca.
Il vantaggio si avrà quando, ad esempio, dovremo realizzare un voltmetro con uscita su display, risulta ovvio che non sarà necessario riscrivere le istruzioni per la visualizzazione, in quanto già presenti nella nostra biblioteca.
Un programmatore esperto non scrive un intero programma per ogni applicazione, ma bensì si limita a collegare tra di loro differenti programmi standard, scrivendo solo la parte del programma (main program) specifica per quell'applicazione.

1.3 Set di istruzioni

Con il termine "set di istruzioni" indichiamo un certo numero di comandi eseguibili dalla CPU.
Ne deriva che ogni set di istruzioni appartiene ad un determinato microcontrollore, nel nostro caso quello relativo ai micro ST626X è composta da 31 diverse istruzioni, ognuna delle quali impartisce un diverso ordine alla CPU.
Ogni istruzione può essere rappresentata in due modi diversi:

1. "**opcode**" e in questo caso risulta comprensibile alla CPU,;
2. "**mnemonica**" e in questo caso è comprensibile dal programmatore.

La rappresentazione mnemonica viene utilizzata dal programmatore per la creazione del file definito "**sorgente**", questo file viene scritto utilizzando un editatore di testi, ad esempio il programma EDIT.COM del DOS, oppure Blocco note di Windows

Il file sorgente ha l'estensione "**.asm**" (ad esempio **prova.asm**) e contiene la sequenza di istruzioni mnemoniche e le direttive di assemblaggio.

Istruzioni e direttive rappresentano il linguaggio di programmazione, esso viene detto anche "**linguaggio assembler".**

Il micro ST6 è caratterizzato da un set di istruzioni (31) estremamente completo ed anche, come vedremo, relativamente facile da comprendere, riportiamo di seguito in ordine alfabetico il set di istruzioni:

Codice mnemonico	Descrizione (inglese)	Descrizione (italiano)
ADD	**ADD**ition	Somma
ADDI	**ADD**ition **I**mmediate	Somma immediata
AND	Logical **AND**	Funzione logica AND
ANDI	Logical **AND I**mmediate	Funzione logica AND immediata
CALL	**CALL** subroutine	Chiama subroutine
CLR	**CL**ea**R**	Azzera byte
COM	**COM**plement	Complemento byte
CP	**C**om**P**are	compara
CPI	**C**om**P**are **I**mmediate	Compara immediato
DEC	**DEC**rement	Decrementa
INC	**INC**rement	Incremento
JP	**J**um**P**	Salta
JRC	**J**ump **R**elative on **C**arry flag	Salta se c'è riporto
JRNC	**J**ump **R**elative on **N**on **C**arry flag	Salta se non c'è riporto
JRNZ	**J**ump **R**elative on **N**on **Z**ero flag	Salta se l'operazione non dà 0
JRR	**J**ump **R**elative if **R**eset	Salta se il bit è 0
JRS	**J**ump **R**elative if **S**et	Salta se il bit è 1
JRZ	**J**ump **R**elative on **Z**ero flag	Salta se l'operazione dà 0
LD	**L**oa**D**	Carica registro/locazione
LDI	**L**oa**D I**mmediate	Carica registro/loc. immediato
NOP	**N**o **OP**eration	Nessuna operazione
RES	**RES**et bit	Resetta bit
RET	**RET**urn from subroutine	Ritorna da una subroutine
RETI	**RET**urn from **I**nterrupt	Ritorna da un interrupt
RLC	**R**otate **L**eft through **C**arry	Ruota a sinistra con riporto
SET	**SET** bit	Setta bit
SLA	**S**hift **L**eft **A**ccumulator	Ruota a sinistra senza riporto
STOP	**STOP** operation	Blocca il funzionamento del micro
SUB	**SUB**traction	Sottrazione
SUBI	**SUB**traction **I**mmediate	Sottrazione immediata
WAIT	**WAIT** processor	Stato di attesa

Ogni istruzioni impartisce un diverso comando alla CPU, il set di istruzioni è suddiviso in sei famiglie:

1. trasferimento e memorizzazione;
2. aritmetica e logica;
3. salto condizionato, e salto incondizionato
4. richiamo ad una subroutine;
5. manipolazione di bit;
6. controllo.

Raggruppando ora le istruzioni a seconda della loro funzione a cui sono destinate, otteniamo la seguente tabella

Tabella riassuntiva delle istruzioni divise per famiglie					
Trasferimento e memorizzazione	Aritmetica e logica	Salto condizionato e salto incondizionato	Richiamo subroutine	Manipolazione di bit	Controllo
LD LDI	ADD CPI ADDI DEC AND INC ANDI RLC CLR SLA COM SUB CP SUBI	JRC JRNC JRR JRS JRZ JRNZ JP	CALL	RES SET	NOP RET RETI STOP WAIT

Per meglio comprendere il significato preciso di ogni comando (descritto in seguito), è necessario innanzitutto capire i vari modi di indirizzamento della memoria che il micro ST6 utilizza.

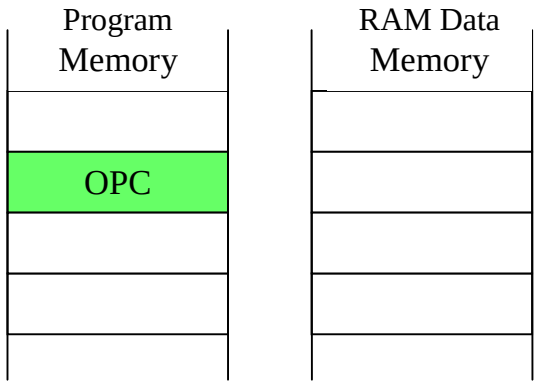
1.4 Modi di indirizzamento

Ogni istruzione agisce diversamente sulla memoria programma poiché occupa un diverso spazio di tale memoria. Le istruzioni mnemoniche vengono convertite dall'assemblatore in numeri esadecimali, questi vengono poi trasferiti permanentemente, mediante programmazione, all'interno della memoria programma.
Ogni istruzione viene quindi scritta all'interno della memoria, e lo spazio necessario per contenerla può variare da uno a tre byte in funzione del tipo di istruzione. Inoltre ogni istruzione accede alle locazioni di memoria, o meglio indirizza le locazioni di memoria, sia essa di programma o di dati, in modo diverso.
Possiamo distinguere all'interno del set di istruzioni nove diversi modi di indirizzamento della memoria, essi sono:

N	**Modi di indirizzamento (Ing)**	**Modi di indirizzamento (Ita)**
1	Inherent	Inerente
2	Direct	Diretto
3	Short Direct	Corto Diretto
4	Indirect	Indiretto
5	Immediate	Immediato
6	Program Counter Relative	Program Counter Relativo
7	Extended	Esteso
8	Bit Direct	Bit Diretto
9	Bit Test e Branch	Testa il Bit e Salta

Analizziamo singolarmente i nove modi di indirizzamento.

1- Inherent (inerente) (***1 byte, 2 cicli***):L'indirizzamento **inherent** è caratterizzato da istruzioni che occupano un solo byte di memoria e il cui unico opcode contiene tutte le informazioni necessarie alla CPU

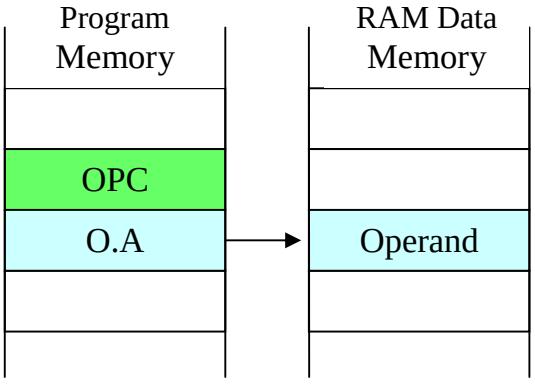


OPC = Opcode

Esempi:

Istruzione	**Commento**
WAIT	Mette in Stanby (basso consumo) il micro, l'oscillatore di clock rimane attivo
STOP	Blocca l'oscillatore di clock mettendo in Stanby tutto il micro
RETI	Esce dalla subroutine di interrupt e ritorna al punto precedente all'evento di interrupt

2- Direct (diretto) (***2 byte, 4 cicli***): Nel modo di indirizzamento **direct** l' indirizzo dell'operando (O.A.) si trova nella memoria programma subito dopo il codice operativo (opcode), l'operando si trova nella memoria dati (RAM Data Memory).



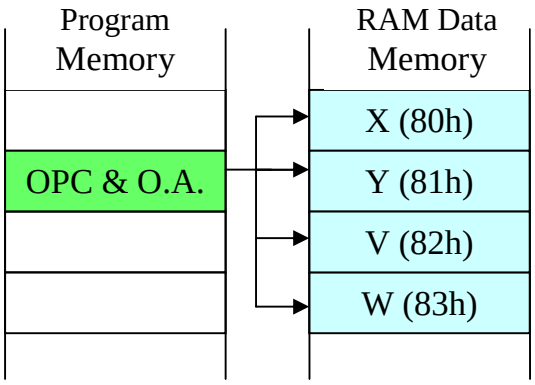
OPC = Opcode
O.A. = Operand Address

Esempi:

Istruzione	**Commento**
LD A,0A3h	Trasferisce il byte contenuto nella locazione 0A3h all'interno del registro accumulatore
SUB A, 11h	Sottrae al byte contenuto nel registro accumulatore il byte contenuto nella locazione 11h

Le istruzioni che causano un trasferimento di un numero da uno spazio di memoria ad un altro spazio di memoria, oppure che eseguono un'operazione matematica tra i numeri contenuti in due spazi di memoria vengono definite ad indirizzamento "direct".
Tutte le istruzioni ad indirizzamento direct occupano due byte di memoria e richiedono per essere eseguite un massimo di quattro cicli macchina.

3- Short Direct (Corto Diretto) (***1 byte, 4 cicli***): Nel modo d'indirizzamento **Short Direct** il dato si trova in uno dei quattro registri del core (X, Y, V, W), l'indirizzo del dato è una parte dell'opcode.



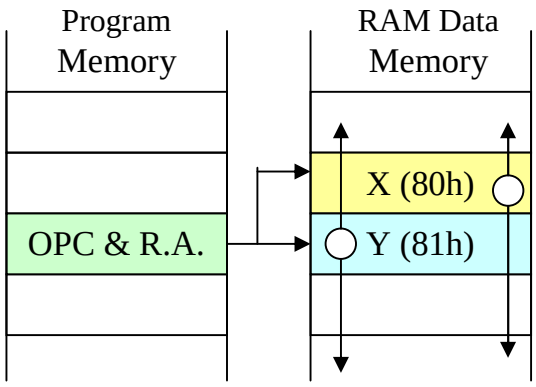
OPC = Opcode
O.A. = Operand Address

Esempi:

Istruzione	**Commento**
LD A,X	Trasferisce il byte contenuto nel registro X (080h) all'interno del registro accumulatore
INC Y	Incrementa di uno il contenuto del registro Y (081h)

Tutte le istruzioni ad indirizzamento short direct occupano un byte di memoria e richiedono per essere eseguite un massimo di quattro cicli macchina.

4 - Indirect (Indiretto) (***1 byte, 4 cicli***): Nel modo d'indirizzamento **Indirect**, le istruzioni utilizzano il registro X o il registro Y per "puntare" alla locazione di memoria, ovvero l'indirizzo della locazione di memoria contenente il dato coincide con il contenuto del registro X o Y.



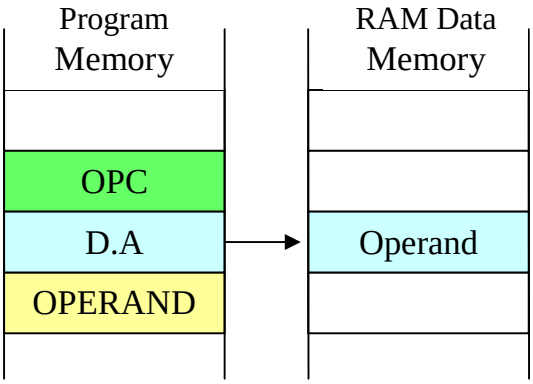
OPC = Opcode
R.A. = Register Address

Esempi:

Istruzione	**Commento**
LD A,(X)	Trasferisce il numero contenuto all'interno della cella di memoria il cui indirizzo è uguale al contenuto del registro X, all'interno del registro accumulatore
INC (Y)	Somma il numero uno al numero contenuto all'interno della cella di memoria il cui indirizzo coincide con il contenuto del registro Y.

Tutte le istruzioni ad indirizzamento indirect occupano un byte di memoria e richiedono per essere eseguite un massimo di quattro cicli macchina.

5 - Immediate (Immediato) (***3 byte, 4 cicli***): Nel modo d'indirizzamento **Immediate** l'operando si trova nella ROM di programma (ultimo byte dell' istruzione), questo modo può essere usato per l' inizializzazione dei registri e delle variabili.
Le istruzioni ad indirizzamento "immediate" sono caratterizzate da un trasferimento immediato di un numero all'interno della memoria, oppure da un'operazione matematica tra un numero e una cella di memoria.



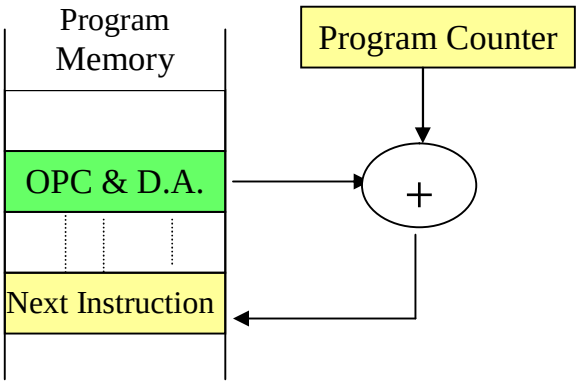
OPC = Opcode
D.A. = Destination Address

Esempi:

Istruzione	Commento
LDI 34h,DFh	Trasferisce all'interno della cella con indirizzo 34h il numero DFh
SUBI A,22h	Sottrae al contenuto del registro accumulatore il numero 22h

Tutte le istruzioni ad indirizzamento immediate occupano tre byte di memoria e richiedono per essere eseguite un massimo di quattro cicli macchina

6 - Program Counter Relative (Program Counter Relativo) (**1** ***byte, 2 cicli***): L'indirizzamento consiste in istruzioni che possono causare un salto della CPU a una locazione con indirizzo superiore, al massimo, di 16 e inferiore, al massimo, di 15 rispetto alla locazione iniziale (opcode+offset: offset=+16, -15)
Questo modo di indirizzamento è usato soltanto con i salti condizionati (JRC, JRNC, JRZ, ecc), se la condizione è vera viene aggiunto il valore dell'offset (+16, -15) al Program Counter, altrimenti salta due istruzioni.. Il valore di offset è dato dagli ultimi 5 bit del opcode



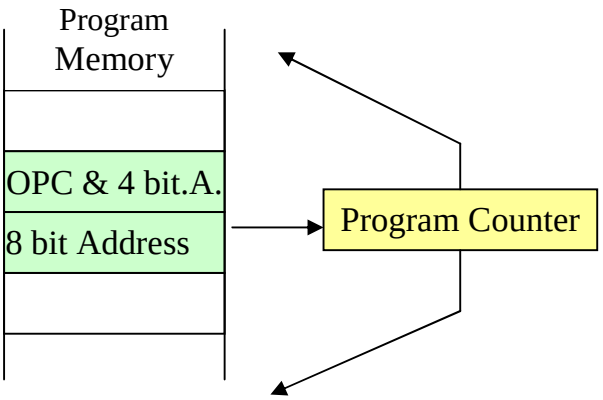
OPC = Opcode
D.A. = Destination Address

Esempi:

Istruzione	Commento
JRC 3	Se il flag di carry è uguale a uno salta alla locazione con indirizzo uguale al contenuto del program counter +3
JRNZ -7	Se il flag di zero è uguale a zero salta alla locazione con indirizzo uguale al contenuto del program counter - 3

L'indirizzo relativo di salto può essere anche un'etichetta, che viene trasformata automaticamente dall' assemblatore. Queste istruzioni occupano un solo byte di memoria e richiedono due cicli macchina.

7 - Extended (Esteso) (***2 byte, 4 cicli***): il modo di indirizzamento **extended** è usato per fare salti lunghi all'interno dello spazio di memoria di programma (4k). I dati richiedono 12 bit e sono forniti dalla metà del byte del opcode e dal secondo byte (8 bit Address)



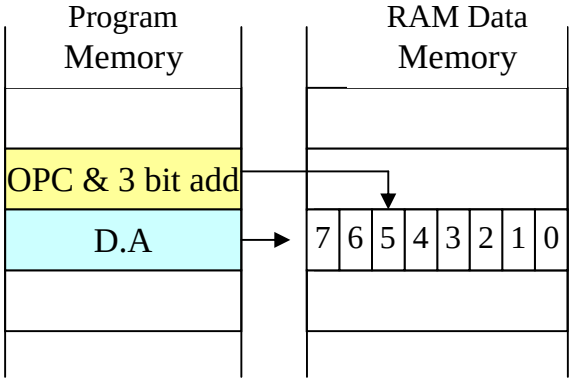
OPC = Opcode

Esempi:

Istruzione	**Commento**
JP 3FAh	Carica il Program Counter con 3FAh e salta all'istruzione memorizzata in 3FAh
CALL tempo	Il Program Counter corrente viene salvato nello stack e successivamente caricato con il valore associato all'etichetta tempo

L'indirizzo assoluto di salto può essere anche un'etichetta, che viene trasformata automaticamente dall' assemblatore. Queste istruzioni occupano due byte di memoria e richiedono quattro cicli macchina

8 - Bit Direct (Bit Diretto) (***2 byte, 4 cicli***): L'indirizzamento **Bit Direct**, indica le istruzioni che agiscono direttamente su un bit di una determinata locazione di memoria, esso permette che settare o resettare un determinato bit di un registro o locazione di memoria RAM.
L' indirizzo del bit è dato nella forma"**n, R**" dove la **n** è il numero del bit ed **R** è l' indirizzo del registro. Il numero del bit è determinato da tre bit del opcode e l' indirizzo del registro è dato dal secondo byte. (Distination Address



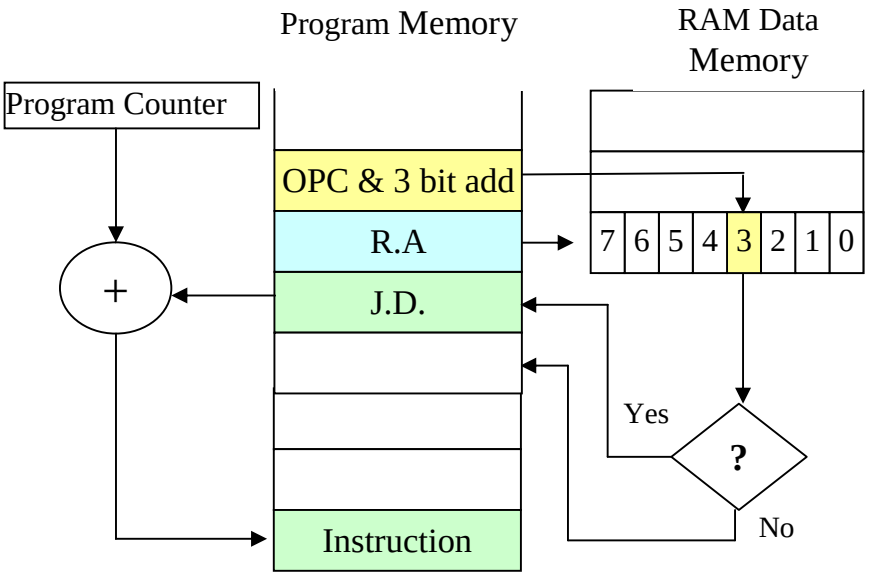
OPC = Opcode
D.A. = Destination Address

Esempi:

Istruzione	**Commento**
SET 4, A	Associa al bit numero 4 del registro accumulatore lo stato "1
RES 6,PORT	Azzera il bit 6 della locazione denominata PORT

L'indirizzo del registro/variabile può essere anche un'etichetta, che viene trasformata automaticamente dall' assemblatore. Queste istruzioni occupano due byte di memoria e richiedono quattro cicli macchina

9 - Bit Test & Branch (Testa il Bit e Salta) (***3 byte, 5 cicli***): L'indirizzamento di memoria definito **"Bit test e Branch",** indica istruzioni che testano lo stato di un bit ed eventualmente causano un "salto" del programma.
Il numero del bit è determinato da tre bit del opcode e l' indirizzo del registro/locazione è dato dal secondo byte. (Relative Address), il terzo byte è lo spostamento di salto (Jump Displacement), con range –126 - + 129.
Lo spostamento (Jump Displacement) ed il secondo byte (Relative Address) possono essere determinati usando delle etichette, che saranno convertite in numero dall' assemblatore.
Lo stato del bit testato viene copiato nel flag di carry



OPC = Opcode
R.A. = Relative Address
J.D. = Jump Displacement

Ogni istruzione richiede tre byte di memoria e cinque cicli macchina.

Esempi:

Istruzione	**Commento**
JRS 3,PORT,LAB1	Testa lo stato del terzo bit del byte di memoria definito con il nome PORT. Se questo bit è a "1" vai all'indirizzo contraddistinto dalla label: LAB1
JRR 1,0Ah,-72	Se il bit 1 della locazione di memoria 0Ah è resettato ("0"), salta a PC=PC-72

Dopo aver compreso i modi di indirizzamento della memoria, riportiamo la tabella, che raggruppa per ogni istruzione 1'indirizzamento supportato.

Addressing modes/Instruction Table

Instruction	Inh	Dir	Sh Dir	Ind	Imm	PCR	Ext	Bit dir	Bit Test	Flags	
										Z	C
ADD		X	X	X						Y	Y
ADDI			X	X	X					Y	Y
AND		X								Y	N
ANDI					X					Y	N
CALL							X			N	N
CLR A		X								Y	Y
CLR		X								N	N
COM	X									Y	Y
CP		X		X						Y	Y
CPI					X					Y	Y
DEC		X	X	X						Y	N
INC		X	X	X						Y	N
JP							X			N	N
JRC, JRNC						X				N	N
JRZ, JRNZ						X				N	N
JRR, JRS									X	N	Y
LD										Y	N
LDI					X					Y	N
NOP						X				N	N
RES, SET								X		N	N
RET	X									N	N
RETI	X									Y	Y
RLC	X									Y	Y
SLA	X									Y	Y
STOP, WAIT	X									N	N
SUB		X		X						Y	Y
SUBI					X					Y	Y

Notes:
Inh=Inherent, Dir=Direct, Sh Dir=Short Direct, Ind=Indirect, Imm=Immediate
PCR=Program Counter Relative, Ext=Extended, Bit Dir=Bit Direct
Bit Test=Bit Test & Branch, Y=Yes, N=No

1.5 Sintassi - Set di istruzioni ST6

Analizziamo ora la corretta sintassi di ogni istruzione e riportiamo dettagliatamente la funzione svolta. Nella trattazione usiamo la seguente simbologia:

- rr : indirizzo di uno spazio di memoria (1 byte) all'interno della memoria dati;
- nn : dato rappresentato da un byte ovvero un numero da 00h a FFh;
- src : source, per indicare un byte sorgente;
- dst : destination, per indicare un byte di destinazione;
- A : accumulatore;
- X : registro X;
- Y : registro Y;
- V . registro V;
- W : registro W.

ADD -- *somma*

Mnemonica : ADD

Funzione : somma

Sintassi : ADD A,src

Descrizione : somma il contenuto del byte sorgente (src) con il contenuto del registro accumulatore (A), il risultato viene scritto nel registro accumulatore, il byte sorgente non viene modificato

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
ADD A,A	5F FF	2	4	yes	yes
ADD A,X	5F 80	2	4	yes	yes
ADD A,Y	5F 81	2	4	yes	yes
ADD A,V	5F 82	2	4	yes	yes
ADD A,W	5F 83	2	4	yes	yes
ADD A,(X)	47	1	4	yes	yes
ADD A,(Y)	4F	1	4	yes	yes
ADD A,rr	5F rr	2	4	yes	yes

rr : indirizzo di uno spazio di memoria (1 byte) all'interno della memoria dati;

Il flag Z e' settato se il risultato e' 0, resettato se diverso da 0
Il flag C e' settato se l'operazione da un riporto, cioè se e' maggiore di 255 (FFh)

Esempio:

Prima A [F6h] X [04h] C [0] Z [0]

ADD A,X

Dopo A [FAh] X [04h] C [0] Z [0]

Modi di indirizzamento: Source : diretto, indiretto
Destinazione : accumulatore

ADDI – *somma immediata*

Mnemonica : ADDI

Funzione : somma immediata

Sintassi : ADDI A,nn

Descrizione : somma un numero (nn) con il contenuto del registro accumulatore (A), il risultato viene scritto nel registro accumulatore

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
ADDI A,nn	57 nn	2	4	yes	yes

nn : dato rappresentato da un byte ovvero un numero da 00h a FFh

Il flag Z e' settato se il risultato e' 0, resettato se diverso da 0
Il flag C e' settato se l'operazione da un riporto, cioè se e' maggiore di 255 (FFh)

Esempio:

Prima A [20h] C [0] Z [0]

ADDI A,22h

Dopo A [42h] C [0] Z [0]

Modi di indirizzamento: Source : immediato
Destinazione : accumulatore

AND – *funzione logica AND*

Mnemonica : AND

Funzione : funzione logica AND

Sintassi : AND A,src

Descrizione : Esegue l'operazione di AND logico tra il byte sorgente (src) e il registro accumulatore (A) il risultato viene scritto nel registro accumulatore, il byte sorgente non viene modificato.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
AND A,A	BF FF	2	4	yes	no
AND A,X	BF 80	2	4	yes	no
AND A,Y	BF 81	2	4	yes	no
AND A,V	BF 82	2	4	yes	no
AND A,W	BF 83	2	4	yes	no
AND A,(X)	A7	1	4	yes	no
AND A,(Y)	AF	1	4	yes	no
AND A,rr	BF rr	2	4	yes	no

rr : indirizzo di uno spazio di memoria (1 byte) all'interno della memoria dati;

Il flag Z e' settato se il risultato e' 0, resettato se diverso da 0
Il flag C non e' influenzato

Esempio:

Prima A 11001100 Y 11110000 C 0 Z 0

AND A,Y

Dopo A 11000000 Y 11110000 C 0 Z 0

Modi di indirizzamento: Source : diretto, indiretto
Destinazione : accumulatore

ANDI – *funzione logica AND immediata*

Mnemonica : ANDI

Funzione : funzione logica AND immediata

Sintassi : ANDI A,nn

Descrizione : Esegue l'operazione di AND logico tra il byte immediato (nn) e il registro accumulatore (A), il risultato viene scritto nel registro accumulatore

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
ANDI A,nn	B7 nn	2	4	yes	No

nn : dato rappresentato da un byte ovvero un numero da 00h a FFh

Il flag Z e' settato se il risultato e' 0, resettato se diverso da 0
Il flag C non e' influenzato

Esempio:

Prima A 00001111 C 0 Z 0

ANDI A,33h → 33h =00110011b

Dopo A 00000011 C 0 Z 0

Modi di indirizzamento: Source : immediato
Destinazione : accumulatore

CALL – *chiama subroutine*

Mnemonica : CALL

Funzione : chiama subroutine

Sintassi : CALL nome subroutine

Descrizione : viene usata per chiamare una subroutine, la CPU abbandona il suo normale corso di istruzioni, salvando la posizione nello stack, e va a gestire la subroutine.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
CALL label	C0001 ab	2	4	no	no

label : etichetta della subroutine da eseguire

I flag Z e C non sono influenzati

Esempio:

Prima Program Counter (PC) 345h

CALL 8DCh

Dopo Program Counter (PC) 8DCh

Il numero 345h viene salvato all'interno dello stack, poi, al Program Counter (PC) viene associato il numero 8DCh, la CPU va ad eseguire l'istruzione presente alla locazione esadecimale 8DC, definita come la prima istruzione della subroutine chiamata.

Modi di indirizzamento: Extended

CLR *–azzera byte*

Mnemonica : CLR

Funzione : azzera byte

Sintassi : CLR dst

Descrizione : resetta l'accumulatore, un registro o una variabile, ha come effetto il trasferimento del numero 00h all'interno del byte di destinazione (dst).

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
CLR A	DF FF	2	4	yes	yes
CLR X	0D 80 00	3	4	no	no
CLR Y	0D 81 00	3	4	no	no
CLR V	0D 82 00	3	4	no	no
CLR W	0D 83 00	3	4	no	no
CLR rr	0D rr 00	3	4	no	no

rr : indirizzo di uno spazio di memoria (1 byte) all'interno della memoria dati;

Il flag Z e' settato
Il flag C e' resettato

Esempio:

Prima A 11001100 C 0 Z 0

CLR A

Dopo A 00000000 C 0 Z 1

Modi di indirizzamento: Direct

COM -- *complemento byte*

Mnemonica : COM

Funzione : complemento byte

Sintassi : COM A

Descrizione : Calcola il complemento del contenuto dell'accumulatore e memorizza il risultato nell'accumulatore

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
COM A	2D	1	4	yes	yes

Il flag Z e' settato se il risultato e' 0, resettato se diverso da 0
Il flag C e' settato se prima della funzione COM il bit 7 di A e' 1

Esempio:

Prima	A	11001100	C	0	Z	0
		COM A				
Dopo	A	00110011	C	1	Z	0

Modi di indirizzamento: Inherent

CP – *compara*

Mnemonica : CP

Funzione : compara

Sintassi : CP A, src

Descrizione : Compara il contenuto di un registro o di una variabile con l'accumulatore

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
CP A,A	3F FF	2	4	yes	yes
CP A,X	3F 80	2	4	yes	yes
CP A,Y	3F 81	2	4	yes	yes
CP A,V	3F 82	2	4	yes	yes
CP A,W	3F 83	2	4	yes	yes
CP A, (X)	27	1	4	yes	yeso
CP A,(Y)	2F	1		yes	yes
CP A,rr	3F rr	2		yes	yes

rr : indirizzo di uno spazio di memoria (1 byte) all'interno della memoria dati;

Il flag **Z** e' settato se i numeri sono uguali, resettato se diversi
Il flag **C** e' settato se l'accumulatore è minore del registro o della variabile, resettato se maggiore o uguale

Esempio: se il registro accumulatore contiene il valore 11111000 (F8h) e la locazione di memoria 34h contiene il valore 11111010 (FAh), l'esecuzione dell'istruzione
CP A,34h determina:

Prima	A	F8h	Loc. 34h	FAh	C	0	Z	0

CP A, 34h

Dopo	A	F8h	Loc. 34h	FAh	C	1	Z	0

Modi di indirizzamento: Diretto, Indiretto

CPI – *compara immediato*

Mnemonica : CPI

Funzione : compara immediato

Sintassi : CPI A,nn

Descrizione : Compara il contenuto dell'accumulatore con un numero

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
CPI A,nn	37 nn	2	4	yes	yes

nn : 1 byte (numero da 0 a 255 (00h – FFh)

Il flag **Z** e' settato se i numeri sono uguali, resettato se diversi
Il flag **C** e' settato se l'accumulatore è minore del numero **nn**, resettato se maggiore o uguale

Esempio: se il registro accumulatore contiene il valore 11111001 (F9h), l'esecuzione dell'istruzione: CPI A,FBh determina:

Prima A [F9h] C [0] Z [0]

CPI A, FBh

Dopo A [F9h] C [1] Z [0]

Modi di indirizzamento: Sorce: Immediato

Destination: Accumulatore

DEC -- *decrementa*

Mnemonica : DEC

Funzione : decrementa

Sintassi : DEC dst

Descrizione : decrementa di 1 il contenuto dell'accumulatore, di un registro o di una variabile.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
DEC A	FF FF	2	4	yes	No
DEC X	1D	1	4	yes	No
DEC Y	5D	1	4	yes	No
DEC V	9D	1	4	yes	No
DEC W	DD	1	4	yes	No
DEC (X)	E7	1	4	yes	No
DEC (Y)	EF	1	4	yes	No
DEC rr	FF rr	2	4	yes	No

rr : indirizzo di uno spazio di memoria (1 byte) all'interno della memoria dati;

Il flag **Z** è settato se il risultato è 0, resettato se diverso da 0
Il flag **C** non è influenzato

Esempio:

Prima A 01h C 0 Z 0

DEC A

Dopo A 00h C 0 Z 1

Modi di indirizzamento: corto diretto, diretto, indiretto

INC -- *incrementa*

Mnemonica : INC

Funzione : decrementa

Sintassi : INC dst

Descrizione : incrementa di 1 il contenuto dell'accumulatore, di un registro o di una variabile.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
INC A	7F FF	2	4	yes	No
INC X	15	1	4	yes	No
INC Y	55	1	4	yes	No
INC V	95	1	4	yes	No
INC W	D5	1	4	yes	No
INC (X)	67	1	4	yes	No
INC (Y)	6F	1	4	yes	No
INC rr	7F rr	2	4	yes	No

rr : indirizzo di uno spazio di memoria (1 byte) all'interno della memoria dati;

Il flag **Z** è settato se il risultato è 0, resettato se diverso da 0
Il flag **C** non è influenzato

Esempio:

Prima X FAh C 0 Z 0

INC X

Dopo X FBh C 0 Z 0

Modi di indirizzamento: corto diretto, diretto, indiretto

JP – *salto (incondizionato)*

Mnemonica : JP

Funzione : salto (incondizionato)

Sintassi : JP abc

Descrizione : l'istruzione JP sostituisce il valore associato al PC con il valore dei 12 bit del abc, facendo cosi un semplice salto incondizionato in un'altra locazione nella memoria programma. Il valore precedente del PC viene perso.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
JP abc	c1001 ab	2	4	no	no

abc : etichetta dell'indirizzo di memoria (indirizzo a 12 bit)

I flag **Z** e **C** non sono influenzati

Esempio:

Prima Program Counter (PC) 345h

JP 5CDh

Dopo Program Counter (PC) 5CDh

Il numero 345h viene perso ed al PC viene associato il numero 5CDh, l'elaborazione continua da questa locazione.

Modi di indirizzamento: Extended

JRC– *salta se c'è riporto*

Mnemonica : JRC

Funzione : salta se c'è riporto

Sintassi : JRC e

Descrizione : Se il flag di carry (C) è uguale ad "1" salta alla locazione con indirizzo uguale al contenuto del PC più o meno il valore di "e". Il range di "e" va da +16 a –15.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
JRC e	e110	1	2	no	no

e : numero che rappresenta la distanza in byte dell'etichetta di salto rispetto all'istruzione, "e" può essere anche un'etichetta , che viene trasformata in numero (+16, -15) dall'assemblatore.

I flag **Z** e **C** non sono influenzati

Esempio: **C** = 1

Prima Program Counter (PC) 316h

JRC +3

Dopo Program Counter (PC) 319h

Modi di indirizzamento: Program Counter Relative

JRNC– *salta se non c'è riporto*

Mnemonica : JRNC

Funzione : salta se non c'è riporto

Sintassi : JRNC e

Descrizione : Se il flag di carry (C) è uguale a "0" salta alla locazione con indirizzo uguale al contenuto del PC più o meno il valore di "e". Il range di "e" va da +16 a –15.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
JRNC e	e010	1	2	no	no

e : numero che rappresenta la distanza in byte dell'etichetta di salto rispetto all'istruzione, "e" può essere anche un'etichetta , che viene trasformata in numero (+16, -15) dall'assemblatore.

I flag **Z** e **C** non sono influenzati

Esempio: C = 0

Prima Program Counter (PC) 846h

JRNC -5

Dopo Program Counter (PC) 841h

Modi di indirizzamento: Program Counter Relative

JRNZ– *salta se l'operazione non da 0*

Mnemonica : JRNZ

Funzione : salta se l'operazione non da 0.

Sintassi : JRNZ e

Descrizione : Se il flag di zero (Z) è uguale a "0" salta alla locazione con indirizzo uguale al contenuto del PC più o meno il valore di "e". Il range di "e" va da +16 a –15.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
JRNZ e	e000	1	2	no	no

e : numero che rappresenta la distanza in byte dell'etichetta di salto rispetto all'istruzione, "e" può essere anche un'etichetta , che viene trasformata in numero (+16, -15) dall'assemblatore.

I flag **Z** e **C** non sono influenzati

Esempio: Z = 0

Prima Program Counter (PC) 846h

JRNZ +5

Dopo Program Counter (PC) 84Bh

Modi di indirizzamento: Program Counter Relative

JRR– *salta se bit è 0*

Mnemonica : JRR

Funzione : salta se bit è 0.

Sintassi : JRR b,rr,ee

Descrizione : Se un determinato bit (b) di una variabile (rr) è resettato, il PC salta alla locazione con indirizzo uguale al contenuto del valore del PC più o meno il valore di "ee". Il range di "ee" va da +129 a –126.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
JRR b,rr,ee	b00011 rr ee	3	5	no	yes

b : numero del bit da testare (da 0 a 7)
rr : byte di indirizzo della variabile
ee : numero che rappresenta la distanza in byte dell'etichetta di salto rispetto all'istruzione JRR, "ee" può essere anche un'etichetta , che viene trasformata in numero (+129, -126) dall'assemblatore.

Il flag **Z** non è influenzato.
Il bit testato viene copiato nel Carry (flag C)

Esempio:

Prima Program Counter (PC) 110h Loc. 70h 001**0**0100

JRR 4,70h,-20

Dopo Program Counter (PC) 90h

Se il bit 4 della locazione di memoria 70h è resettato ("0"), salta a PC=PC-20=110-20=90h

Modi di indirizzamento: Bit Test & Branch

JRS– *salta se bit è 1*

Mnemonica : JRS

Funzione : salta se bit è 1.

Sintassi : JRS b,rr,ee

Descrizione : Se un determinato bit (b) di una variabile (rr) è settato ("1"), il PC salta alla locazione con indirizzo uguale al contenuto del valore del PC più o meno il valore di "ee". Il range di "ee" va da +129 a –126.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
JRR b,rr,ee	b10011 rr ee	3	5	no	yes

b : numero del bit da testare (da 0 a 7)
rr : byte di indirizzo della variabile
ee : numero che rappresenta la distanza in byte dell'etichetta di salto rispetto all'istruzione JRS, "ee" può essere anche un'etichetta , che viene trasformata in numero (+129, -126) dall'assemblatore.

Il flag **Z** non è influenzato.
Il bit testato viene copiato nel Carry (flag C)

Esempio:

Prima Program Counter (PC) 110h Loc. port 00100**1**00

JRS 2,port,lab

Dopo Program Counter (PC) lab

Prima Program Counter (PC) 110h

JRS 2,port,lab

Dopo Program Counter (PC) lab

Testa lo stato del **bit 2** del byte di memoria definito con il nome **port** . Se questo bit è a "1" il PC va all'indirizzo contraddistinto dall'etichetta: **lab**.

Modi di indirizzamento: Bit Test & Branch

JRZ– *salta se l'operazione da 0*

Mnemonica : JRZ

Funzione : salta se l'operazione da 0.

Sintassi : JRZ e

Descrizione : Se il flag di zero (Z) è uguale a "1" salta alla locazione con indirizzo uguale al contenuto del PC più o meno il valore di "e". Il range di "e" va da +16 a –15.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
JRZ e	e100	1	2	no	no

e : numero che rappresenta la distanza in byte dell'etichetta di salto rispetto all'istruzione, "e" può essere anche un'etichetta , che viene trasformata in numero (+16, -15) dall'assemblatore.

I flag **Z** e **C** non sono influenzati

Esempio: Z = 1

Prima Program Counter (PC) 423h

JRNZ +7

Dopo Program Counter (PC) 42Ah

Modi di indirizzamento: Program Counter Relative

LD – *carica registro*

Mnemonica : LD

Funzione : carica registro

Sintassi : LD dst, src

Descrizione : carica il contenuto del byte sorgente (src) nel byte destinazione (dst). Per questa istruzione va sempre usato l'accumulatore.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
LD A,X	35	1	4	yes	no
LD A,Y	75	1	4	yes	no
LD A,V	B5	1	4	yes	no
LD A,W	F5	1	4	yes	no
LD X,A	3D	1	4	yes	no
LD Y,A	7D	1	4	yes	no
LD V,A	BD	1	4	yes	no
LD W,A	FD	1	4	yes	no
LD A,(X)	07	1	4	yes	no
LD A,(Y)	87	1	4	yes	no
LD (X),A	0F	1	4	yes	no
LD (Y),A	8F	1	4	yes	no
LD A,rr	1F rr	2	4	yes	no
LD rr,A	9F rr	2	4	yes	no

A = registro accumulatore
X,Y,V,W = registri del micro
rr = indirizzo della variabile (1 byte)

Il flag **Z** e' settato se il risultato e' 0, resettato se diverso da 0
Il flag **C** non e' influenzato

Esempio: Se la locazione del registro 34h contiene il valore 45h allora l'istruzione;

LD A,34h

Può causare la carica dell'accumulatore con il valore 45h. Così il registro 34h conserverà il valore 45h.

LDI – *carica registro immediato*

Mnemonica : LDI

Funzione : carica registro immediato

Sintassi : LDI dst, src
il byte sorgente /src) è sempre un dato immediato (src=nn) mentre il byte destinazione (dst) può essere l'accumulatore, uno dei registri X,Y,V,W o una locazione disponibile nello spazio dati dei registri.

Descrizione : Serve per caricare un numero in una variabile, in un registro o nell'accumulatore.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
LDI A,nn	17 nn	2	4	Yes	no
LDI X,nn	0D 80 nn	3	4	no	no
LDI Y,nn	0D 81 nn	3	4	no	no
LDI V,nn	0D 82 nn	3	4	no	no
LDI W,nn	0D 83 nn	3	4	no	no
LDI rr,nn	0D rr nn	3	4	no	no

A = registro accumulatore
X,Y,V,W = registri del micro
rr = indirizzo della variabile (1 byte)
nn = numero di 1 byte

Il flag **Z** e' settato se il risultato e' 0, resettato se diverso da 0
Il flag **C** non e' influenzato

Esempio: L'istruzione LDI 34h,45h

Prima Loc. 34h | 01h | C | 0 | Z | 0 |

LDI 34h, 45h

Dopo Loc 34h | 45h | C | 0 | Z | 0 |

Carica il byte 45h nella locazione 34h

Modi di indirizzamento: Sorgente: Immediato

Destinazione: Diretto

NOP – *nessuna operazione*

Mnemonica : NOP

Funzione : nessuna operazione

Sintassi : NOP

Descrizione : Esegue 2 cicli macchina a vuoto. Viene usata per creare dei piccoli ritardi.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
NOP	04	1	2	no	no

I flag **Z** e **C** non sono influenzati.

Modi di indirizzamento: Program Counter Relative

RES – *Resetta bit*

Mnemonica : RES

Funzione : Resetta bit

Sintassi : RES n.bit, variabile/accumulatore

Descrizione : Resetta uno degli 8 bit di una variabile o dell'accumulatore.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
RES b,A	b01011 FF	2	4	no	no
RES b,rr	B01011 rr	2	4	no	no

rr : indirizzo della variabile (1 byte)
A : registro accumulatore
b : numero del bit da resettare(da 0 a 7)

I flag **Z** e **C** non sono influenzati

Esempio:

Prima Loc. 23h 11111111 C 0 Z 0

RES 4, 23h

Dopo Loc 23h 111**0**1111 C 0 Z 0

Modi di indirizzamento: Bit Direct

RET – *ritorna da una subroutine*

Mnemonica : RET

Funzione : Ritorna da una subroutine

Sintassi : RET

Descrizione : Esce dalla subroutine e ritorna al punto della chiamata CALL

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
RET	CD	1	2	no	no

I flag **Z** e **C** non sono influenzati

Esempio:

Prima Program Counter 456h Stack Register 3DFh

RET

Dopo Program Counter 3DFh Stack Register 3DFh

Modi di indirizzamento: Inherent

RETI – *Ritorna da un interrupt*

Mnemonica : RETI

Funzione : Ritorna da un interrupt

Sintassi : RETI

Descrizione : Esce dalla subroutine di interrupt e ritorna al punto precedente all'evento di interrupt.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
RETI	4D	1	2	yes	yes

I flag **Z** e **C** vengono riportati alla condizione in cui si trovavano prima dell'interrupt

Esempio:

Prima Program Counter 4A6h Stack Register 3DAh

RETI

Dopo Program Counter 3DAh Stack Register 3DAh

Modi di indirizzamento: Inherent

RLC –*ruota a sinistra con riporto*

Mnemonica : RLC

Funzione : ruota a sinistra con riporto

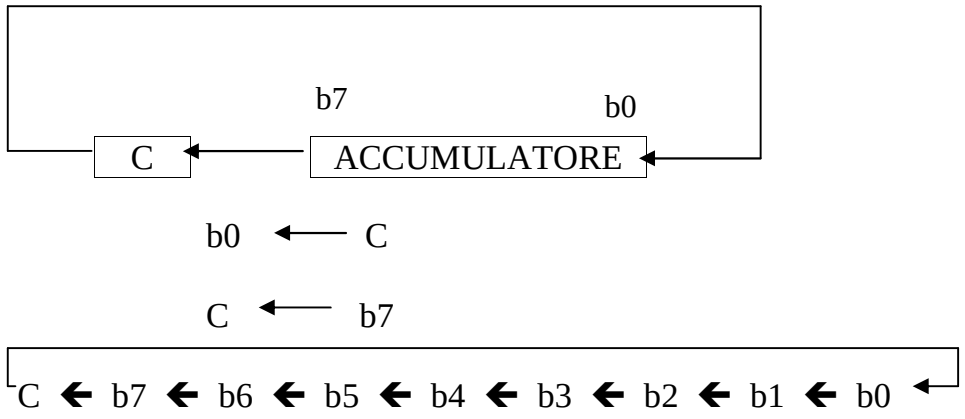
Sintassi : RLC A

Descrizione : Ruota a sinistra i bit dell'accumulatore.Trasferisce il bit 7 (MSB) nel Carry (flag C), mentre il contenuto precedente del Carry passa nel bit 0 dell'accumulatore

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
RLC A	AD	1	4	Yes	Yes

Il flag **Z** è settato se il risultato è 0, resettato se diverso da 0
Il flag **C** riporta il valore del bit 7



Esempio:

Prima A 10001001 C 0 Z 0

RLC A

Dopo A 00010010 C 1 Z 0

Modi di indirizzamento: inherent

SET – *Setta bit*

Mnemonica : SET

Funzione : Setta bit

Sintassi : SET n.bit, variabile/accumulatore

Descrizione : Setta uno degli 8 bit di una variabile o dell'accumulatore.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
SET b,A	B11011 FF	2	4	no	no
SET b,rr	B11011 rr	2	4	no	no

rr : indirizzo della variabile (1 byte)
A : registro accumulatore
b : numero del bit da resettare(da 0 a 7)

I flag **Z** e **C** non sono influenzati

Esempio:

Prima Loc. 2Fh | 10000110 | C | 0 | Z | 0 |

SET 6, 2Fh

Dopo Loc 23h | 1**1**000110 | C | 0 | Z | 0 |

Modi di indirizzamento: Bit Direct

SLA –*ruota a sinistra senza riporto*

Mnemonica : SLA

Funzione : ruota a sinistra senza riporto

Sintassi : SLA A

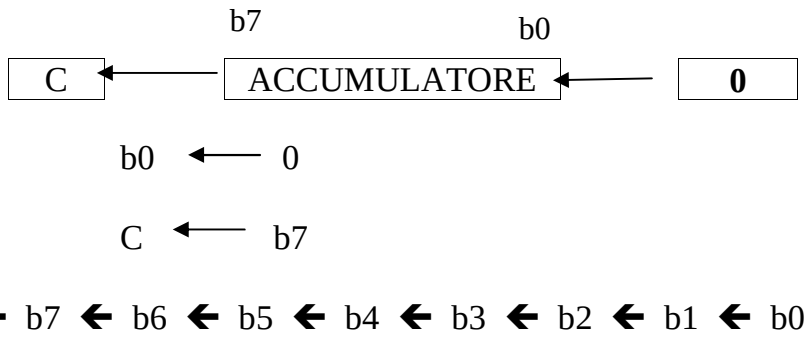
Descrizione : Ruota a sinistra i bit dell'accumulatore.Trasferisce il bit 7 (MSB) nel Carry (flag C) cancellando il valore precedente.

Questa operazione equivale a una moltiplicazione per 2.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
SLA A	5F FF	2	4	Yes	Yes

Il flag **Z** è settato se il risultato è 0, resettato se diverso da 0
Il flag **C** riporta il valore del bit 7



Esempio:

Prima A 00001101 $(13)_{10}$ C 0 Z 0

SLA A

Dopo A 00011010 $(26)_{10}$ C 0 Z 0

Modi di indirizzamento: inherent

STOP – *blocca il funzionamento del microcontrollore*

Mnemonica : STOP

Funzione : blocca il funzionamento del microcontrollore

Sintassi : STOP

Descrizione : Blocca l'oscillatore di clock mettendo in standby tutto il microcontrollore.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
STOP	6D	1	2	no	no

I flag **Z** e **C** non sono influenzati.

Modi di indirizzamento: Inherent

SUB -- *sottrazione*

Mnemonica : SUB

Funzione : sottrazione

Sintassi : SUB A,src

Descrizione : il contenuto di una variabile (byte sorgente src) viene sottratto al contento dell'accumulatore ed il risultato è memorizzato nell'accumulatore.

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
SUB A,A	DF FF	2	4	yes	yes
SUB A,X	DF 80	2	4	yes	yes
SUB A,Y	DF 81	2	4	yes	yes
SUB A,V	DF 82	2	4	yes	yes
SUB A,W	DF 83	2	4	yes	yes
SUB A,(X)	C7	1	4	yes	yes
SUB A,(Y)	CF	1	4	yes	yes
SUB A,rr	DF rr	2	4	yes	yes

rr : indirizzo di uno spazio di memoria (1 byte) all'interno della memoria dati;

Il flag **Z** è settato se il risultato è 0, resettato se diverso da 0
Il flag **C** è settato se il contenuto dell'accumulatore è minore della variabile (byte sorgente src), resettato se maggiore o uguale.

Esempio:

Prima A 78h Y 23h Loc. Ram 23h 53h C 0 Z 0

SUB A,(Y)

Dopo A 25h Y 23h Loc. Ram 23h 53h C 0 Z 0

Modi di indirizzamento: Source : diretto, indiretto
Destinazione : accumulatore

SUBI – *sottrazione immediata*

Mnemonica : SUBI

Funzione : sottrazione immediata

Sintassi : SUBI A,nn

Descrizione : sottrae un numero (nn) con il contenuto del registro accumulatore (A), il risultato viene scritto nel registro accumulatore

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
SUBI A,nn	D7 nn	2	4	yes	yes

nn : dato rappresentato da un byte ovvero un numero da 00h a FFh

Il flag **Z** è settato se il risultato è 0, resettato se diverso da 0
Il flag **C** è settato se il contenuto dell'accumulatore è minore della variabile (byte sorgente src), resettato se maggiore o uguale.

Esempio:

Prima A 56h C 0 Z 0

SUBI A,25

Dopo A 31h C 0 Z 0

Modi di indirizzamento: Source : immediato
Destinazione : accumulatore

WAIT – *stato di attesa*

Mnemonica : WAIT

Funzione : stato di attesa

Sintassi : WAIT

Descrizione : mette in standby il microcontrollore, senza fermare l'oscillatore di clock con un basso assorbimento di corrente. Per ripristinarlo occorre un interrupt esterno o interno (es. Timer, ADC, .. ecc).

In tabella i formati possibili

Instruction Format	**Opcode (Hex)**	**Bytes**	**Cycles**	**Flags**	
				Z	**C**
WAIT	ED	1	2	no	no

I flag **Z** e **C** non sono influenzati.

Modi di indirizzamento: Inherent

SET DI ISTRUZIONE PER ST6260/65

Load e store instructions

IST	ADDRESSING	MODE	BYTE	CYCLES	FLAG Z	FLAG C
LD	a,x	short direct	1	4	si	no
LD	a,y	short direct	1	4	si	no
LD	a,v	short direct	1	4	si	no
LD	a,w	short direct	1	4	si	no
LD	y,a	short direct	1	4	si	no
LD	w,a	short direct	1	4	si	no
LD	a,var	direct	2	4	si	no
LD	var,a	direct	2	4	si	no
LD	a,(x)	indirect	1	4	si	no
LD	a,(y)	indirect	1	4	si	no
LD	(x),a	indirect	1	4	si	no
LD	(y),a	indirect	1	4	si	no
LDI	a,#nr	immediate	2	4	si	no
LDI	var,#nr	immediate	3	4	si	no

Manipolazione di bit

IST	ADDRESSING	MODE	BYTE	CYCLES	FLAG Z	FLAG C
SET	b,var	Bit direct	2	4	no	no
RES	b,var	Bit direct	2	4	no	no

Controllo Istruzioni

IST	ADDRESSING	MODE	BYTE	CYCLES	FLAG Z	FLAG C
NOP		Inherent	1	2	no	no
RET		Inherent	1	2	no	no
RETI		Inherent	1	2	no	no
STOP		Inherent	1	2	no	no
WAIT		Inherent	1	2	no	no
CALL	Label=12 bit addr.	Extended	2	4	no	no
JP	Label=12 bit addr.	Extended	2	4	no	no

a,x,y,v,w = registri interni
var = nome variabili usate (data space register)
#nr valore numerico immediato (decimale, ottale, esadecimale, binario)

Arithmetic e logic instructions

IST	ADDRESSING	MODE	BYTE	CYCLES	FLAG Z	FLAG C
ADD	a,(x)	indirect	1	4	si	si
ADD	a,(y)	indirect	1	4	si	si
ADD	a,var	direct	2	4	si	si
ADDI	a,#nr	immediate	2	4	si	si
AND	a,(x)	indirect	1	4	si	no
AND	a,var	direct	2	4	si	no
ANDI	a,#nr	immediate	2	4	si	no
CLR	a	short direct	2	4	si	si
CLR	var	direct	3	4	no	no
COM	a	indirect	1	4	si	si
CP	a,(x)	indirect	1	4	si	si
CP	a,(y)	indirect	1	4	si	si
CP	a,var	direct	2	4	si	si
CPI	a,#nr	immediate	2	4	si	si
DEC	x	short direct	1	4	si	no
DEC	y	short direct	1	4	si	no
DEC	v	short direct	1	4	si	no
DEC	w	short direct	1	4	si	no
DEC	a	direct	2	4	si	no
DEC	var	direct	2	4	si	no
DEC	(x)	indirect	1	4	si	no
DEC	(y)	indirect	1	4	si	no
INC	x	short direct	1	4	si	no
INC	y	short direct	1	4	si	no
INC	v	short direct	1	4	si	no
INC	w	short direct	1	4	si	no
INC	a	direct	2	4	si	no
INC	var	direct	2	4	si	no
INC	(x)	indirect	1	4	si	no
INC	(y)	indirect	1	4	si	no
RLC	a	indirect	1	4	si	no
SLA	a	inerent	2	4	si	no
SUB	a,(x)	indirect	1	4	si	si
SUB	a,(y)	indirect	1	4	si	si
SUB	a,var	direct	2	4	si	si
SUBI	a,#nr	immediate	2	4	si	si

Istruzioni di salto condizionato

(b=3 bit address e= 5 bit diplacement 15 +16 ee 8 bit displacement –126 + 129)

IST	ADDRESSING	MODE	BYTE	CYCLES	FLAG Z	FLAG C
JRC	e	if c=1	1	2	no	no
JRNC	e	if c=0	1	2	no	no
JRZ	e	if z=1	1	2	no	no
JRNZ	e	if z=0	1	2	no	no
JRR	b,var,ee	if bit=0	3	5	no	si
JRS	b,var,ee	if bit=1	3	5	no	si

SET DI ISTRUZIONE PER ST6260/65

Load e store instructions

IST	ADDRESSING	MODE	BYTE	CYCLES	FLAG Z	FLAG C
LD	a,x	short direct	1	4	si	no
LD	a,y	short direct	1	4	si	no
LD	a,v	short direct	1	4	si	no
LD	a,w	short direct	1	4	si	no
LD	y,a	short direct	1	4	si	no
LD	w,a	short direct	1	4	si	no
LD	a,var	direct	2	4	si	no
LD	var,a	direct	2	4	si	no
LD	a,(x)	indirect	1	4	si	no
LD	a,(y)	indirect	1	4	si	no
LD	(x),a	indirect	1	4	si	no
LD	(y),a	indirect	1	4	si	no
LDI	a,#nr	immediate	2	4	si	no
LDI	var,#nr	immediate	3	4	si	no

Manipolazione di bit

IST	ADDRESSING	MODE	BYTE	CYCLES	FLAG Z	FLAG C
SET	b,var	Bit direct	2	4	no	no
RES	b,var	Bit direct	2	4	no	no

Controllo Istruzioni

IST	ADDRESSING	MODE	BYTE	CYCLES	FLAG Z	FLAG C
NOP		Inherent	1	2	no	no
RET		Inherent	1	2	no	no
RETI		Inherent	1	2	no	no
STOP		Inherent	1	2	no	no
WAIT		Inherent	1	2	no	no
CALL	Label=12 bit addr.	Extended	2	4	no	no
JP	Label=12 bit addr.	Extended	2	4	no	no

a,x,y,v,w = registri interni
var = nome variabili usate (data space register)
#nr valore numerico immediato (decimale, ottale, esadecimale, binario)

Arithmetic e logic instructions

IST	ADDRESSING	MODE	BYTE	CYCLES	FLAG Z	FLAG C
ADD	a,(x)	indirect	1	4	si	si
ADD	a,(y)	indirect	1	4	si	si
ADD	a,var	direct	2	4	si	si
ADDI	a,#nr	immediate	2	4	si	si
AND	a,(x)	indirect	1	4	si	no
AND	a,var	direct	2	4	si	no
ANDI	a,#nr	immediate	2	4	si	no
CLR	a	short direct	2	4	si	si
CLR	var	direct	3	4	no	no
COM	a	indirect	1	4	si	si
CP	a,(x)	indirect	1	4	si	si
CP	a,(y)	indirect	1	4	si	si
CP	a,var	direct	2	4	si	si
CPI	a,#nr	immediate	2	4	si	si
DEC	x	short direct	1	4	si	no
DEC	y	short direct	1	4	si	no
DEC	v	short direct	1	4	si	no
DEC	w	short direct	1	4	si	no
DEC	a	direct	2	4	si	no
DEC	var	direct	2	4	si	no
DEC	(x)	indirect	1	4	si	no
DEC	(y)	indirect	1	4	si	no
INC	x	short direct	1	4	si	no
INC	y	short direct	1	4	si	no
INC	v	short direct	1	4	si	no
INC	w	short direct	1	4	si	no
INC	a	direct	2	4	si	no
INC	var	direct	2	4	si	no
INC	(x)	indirect	1	4	si	no
INC	(y)	indirect	1	4	si	no
RLC	a	indirect	1	4	si	no
SLA	a	inerent	2	4	si	no
SUB	a,(x)	indirect	1	4	si	si
SUB	a,(y)	indirect	1	4	si	si
SUB	a,var	direct	2	4	si	si
SUBI	a,#nr	immediate	2	4	si	si

Istruzioni di salto condizionato

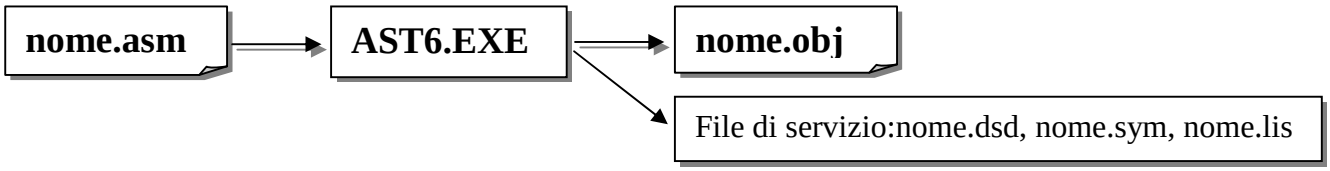
(b=3 bit address e= 5 bit diplacement 15 +16 ee 8 bit displacement –126 + 129)

IST	ADDRESSING	MODE	BYTE	CYCLES	FLAG Z	FLAG C
JRC	e	if c=1	1	2	no	no
JRNC	e	if c=0	1	2	no	no
JRZ	e	if z=1	1	2	no	no
JRNZ	e	if z=0	1	2	no	no
JRR	b,var,ee	if bit=0	3	5	no	si
JRS	b,var,ee	if bit=1	3	5	no	si

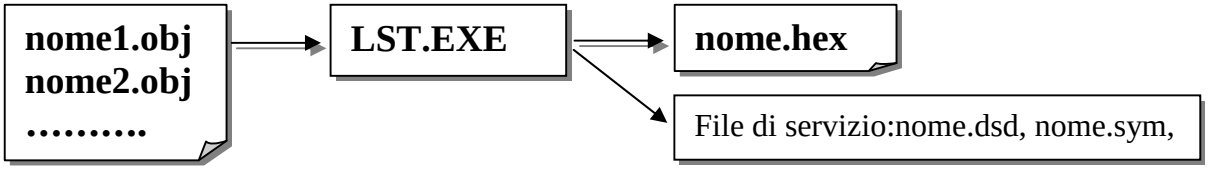
1.6 Programmazione

Scrivere un programma vuol dire inserire delle istruzioni nella sequenza in cui si vuole che esse siano eseguite dal mocrocontrollore, si deve tradurre in linguaggio assembler l'algoritmo rappresentato dal Flow_Chart.
La programmazione può essere suddivisa in fasi:

1. **Scrittura file sorgente**: tramite un editor di testo (Es. Blocco note di Windows), rispettando il set di istruzioni del micro, si traduce il flow-chart in programma, si archivia il programma con estensione **.asm** (esempio: temperatura.asm).
2. **Compilazione/Assemblaggio**:il file sorgente viene trasformato in linguaggio macchina comprensibile al microcontrollore (diventa file oggetto, file con estensione **.obj**); per questa operazione si usa il compilatore fornito dalla ditta produttrice (STM), diviso in Assemblatore (**AST6.EXE**) e Linker (**LST6.EXE**).



La compilazione, oltre al file oggetto (nome.obj) genera alcuni file di servizio (nome.dsd, nome.sym, nome.lis).
Se il programma è composto da più file, si ripete l'operazione per tutti i file.



Il Linker (LST6.EXE) concatena i moduli .obj in un unico file .hex eseguibile pronto per essere trasferito tramite **Starter Kit** all'interno della memoria programma del microcontrollore, inoltre il Linker genera alcuni file di servizio (nome.sym, nome.dsd).
Se il programma è composto di un unico file .asm non occore il Linker, la compilazione viene eseguita con un unico comando.

AST6.EXE –L –S nome.asm

3. **Memorizzazione**:tramite lo **Starter Kit** collegato al PC si memorizza il programma in formato opcode (**nome.hex**) nella memoria EPROM/OTP del microcontrollore, ove possibile si effettua l'emulazione, cioè si simula attraverso lo Starter kit il funzionamento del software.

la sequenza delle istruzioni deve essere scritta in uno o più **file sorgente** con l'estensione **.asm**, che indica un programma in linguaggio assembler. Per scrivere questo file si usa un editor di testo (es. Blocco note di Windows).
Dopo aver scritto il programma

Impostare un programma

In questa pagina verra' descritto dettagliatamente il programma START che puo' anche essere prelevato nella pagina delle routine di esempio. Questo programma e' un esempio tipico di come va impostato un programma per micro ST6. L'esempio si riferisce ai micro ST621x e 2x, ma e' valido anche per altri modelli.

Inizio del programma

```
;************************************************************************
;*                                                      *
;*       FILE SORGENTE DI BASE PER I MICRO ST6210/16/20/25        *
;*                                                      *
;************************************************************************

; by Giuseppe Di Paolo 1996/98
; gdipaolo@mail2.clio.it
```

Tutte queste righe sono dei COMMENTI, perche' iniziano con il segno di punto e virgola ; e quindi verranno ignorate dall'assemblatore. E' molto utile inserire questi commenti in tutte le parti del programma, per descrivere il funzionamento delle singole parti e anche delle singole istruzioni. In questo caso, all'inizio del programma viene inserito il titolo, l'autore e una breve descrizione. E' utilissimo anche descrivere qui le connessioni delle porte di I/O del micro, le varie modifiche e aggiunte apportare durante lo sviluppo del programma e tutte le informazioni che ritenete utili per rendere piu' facile una comprensione e una lettura del programma anche a distanza di tempo.

Direttive assembler

```
        .title  "TITOLO DEL PROGRAMMA"
```

```
        .vers   "ST62E20"       ; TIPO DI MICROPROCESSORE USATO
```

Le direttive .title e .vers permettono di specificare il titolo del programma e la vesione di micro da usare. In questa parte di programma si possono inserire anche altre direttive come ad esempio .w_on per usare la DRW. Notate che la scritta "TIPO DI MICROPROCESSORE USATO" e' un commento perche' e' preceduto dal punto e virgola.

Registri del micro

```
;*********************************************************************
;*                    VARIABILI DEL MICRO                            *
;*********************************************************************

        .input  "st62reg.inc"
```

Qui vengono definiti tutti i registri interni del micro che sono indispensabili in qualsiasi programma. Poiche' i registri sono sempre uguali per qualsiasi programma, sono stati definiti all'interno di un altro file chiamato st62reg.inc che viene incluso nel nostro programma tramite la direttiva .input. Per i micro ST626x c'e' un altro file chiamato invece st626reg.inc. Questi files possono essere prelevati nella pagina degli esempi, oppure cliccando QUI

Variabili del programma

```
;*********************************************************************
;*                    VARIABILI DEL PROGRAMMA                        *
;*********************************************************************

prova   .def    084h    ;DEFINISCE LA VARIABILE 'PROVA'

tempo   .def    085h    ;DEFINISCE LA VARIABILE 'TEMPO'
```

Le variabili sono delle celle di memoria RAM contenenti i dati che vengono elaborati e manipolati all'interno del programma. In questo esempio la variabile "prova" corrisponde alla cella di memoria RAM con indirizzo 084h. Non si possono usare due variabili con lo stesso nome.

Impostazioni iniziali

```
;*********************************************************************
;*                    SETTAGGIO INIZIALE                             *
;*********************************************************************

; PER I MICRO ST6210/15 CON 2Kbyte DI ROM/EPROM USATE L'ISTRUZIONE :
        .org    0880h

; PER I MICRO ST6220/25/60/65 CON 4Kbyte DI ROM/EPROM USATE :
        .org        080h
```

```
inizio          ;  INIZIALIZZAZIONE DELLE PERIFERICHE
        ldi     wdog,0ffh       ; RICARICA IL WATCH DOG

; ************* IMPOSTAZIONE DELLE PORTE I/O *******************
        ldi     pdir_a,11111111b
        ldi     popt_a,11111111b
        ldi     port_a,00000000b        ; porta A come uscita push-pull

        ldi     pdir_b,00000000b
        ldi     popt_b,00000000b
        ldi     port_b,00000000b        ; porta B come input pull-up

        ldi     pdir_c,11111111b
        ldi     popt_c,00000000b
        ldi     port_c,00000000b        ; porta C come uscita OC

        ldi     adcr,0          ;DISABLE A/D INTERRUPT
        ldi     tscr,0          ;DISABLE TIMER INTERRUPT
        ldi     ior,0           ;DISABLE ALL INTERRUPT
        reti                            ;RIPRISTINA I FLAG PRINCIPALI
        jp      main                    ;SALTA AL PROGRAMMA PRINCIPALE
```

La direttiva .org specifica l'indirizzo di partenza del programma all'interno della ROM/EPROM del micro a va modificata a seconda della dimensione della memoria del micro usato. Quando il micro ST6 viene resettato o viene alimentato, il programma salta automaticamente all'etichetta "inizio" per cui le successive istruzioni servono ad inizializzare il microprocessore. Vengono definite le configurazioni delle porte di I/O in funzione dell'uso che poi dovrete farne all'interno del programma. Andate alla pagina "Porte di I/O" per maggiori informazioni. Le successive istruzioni servono a disabilitare tutti gli interrupt. L'istruzione "jp main" serve poi per saltare al programma principale.

Subroutine di interrupt

```
;*********************************************************************
;*                  SUBROUTINE DI INTERRUPT                  *
;*********************************************************************

ad_int                  ;INTERRUPT DEL CONVERTITORE A/D
        reti

tim_int             ;INTERRUPT DEL TIMER
        reti
```

```
BC_int              ;INTERRUPT DELLE PORTE A e B
            reti

A_int               ;INTERRUPT DELLA PORTA A
            reti

nmi_int             ;INTERRUPT NON MASCHERABILE
            reti
```

Quando si verifica un interrupt durante l'esecuzione del programma, il micro salta ad una di queste subroutine in funzione del tipo di interrupt. Notate che tutte queste subroutine devono terminare con l'istruzione "reti".

Subroutine

```
;*********************************************************************
;*                    SUBROUTINE                         *
;*********************************************************************

; ESEMPIO
clock                                                   ; subroutine CLOCK
            "   "   "       ; ISTRUZIONI DELLA SUBROUTINE
            "   "   "
            "   "   "

            ret                             ; esce dalla subroutine
```

Le subroutine sono dei pezzi di programma che vengono richiamati dal programma principale o da altre subroutine tramite l'istruzione "call". Tutte le subroutine devono sempre iniziare con una etichetta (esempio "clock") e terminare con l'istruzione "ret". E' bene suddidere tutto il programma in subroutine, ognuna con una specifica funzione, in modo da rendere piu' facile la lettura e la correzione del programma e permettendo di riutilizzare alcune parti del programma anche in altre future applicazioni. Inoltre e' consigliabile inserire all'interno delle subroutine delle parti di programma che devono essere richiamate frequentemente durante il funzionamento in modo da non riscrivere piu' volte le stesse istruzioni.

Programma principale

```
;*********************************************************************
;*                PROGRAMMA PRINCIPALE                   *
;*********************************************************************

main                            ; INIZIO DEL PROGRAMMA PRINCIPALE

            ldi     wdog,0feh   ;RICARICA IL WATCH DOG
            call    clock       ;RICHIAMA LA SUBROUTINE 'clock' (ESEMPIO)
;       ...         ;ALTRE ISTRUZIONI
```

```
;       ...             ; "     "     "
```

Questo e' il programma principale che deve iniziare sempre con l'etichetta "main".

Vettori di interrupt

```
;*********************************************************************
;*                  VETTORI DI INTERRUPTS                            *
;*********************************************************************

; Questa parte deve essere riportata alla fine del MAIN e prima della
; istruzione finale .END .
; Queste istruzioni non dovranno essere modificate.

        .org    0ff0h
        jp      ad_int  ;INTERRUPT DEL CONV. A/D     vector #4
        jp      tim_int ;INTERRUPT DEL TIMER         vector #3
        jp      BC_int  ;INTERRUPT PORTE A e B       vector #2
        jp      A_int   ;INTERRUPT PORTA A           vector #1
        .org    0ffch
        jp      nmi_int ;INTERRUPT NON MASCHERABILE  vector #0
        jp      inizio  ;INTERRUZIONE PER IL SETTAGGIO INIZIALE
```

Queste istruzioni vanno inserire alla fine del programma MAIN per definire tutti i vettori di interrupt. In pratica queste righe specificano a quali etichette il micro deve saltare in corrispondenza di un interrupt. Notate la presenza dell'etichetta "inizio" che corrisponde al vettore di RESET (quando il micro viene resettato o alimentato salta a "inizio").

Fine del programma

```
        .end                ; Termine del programma
```