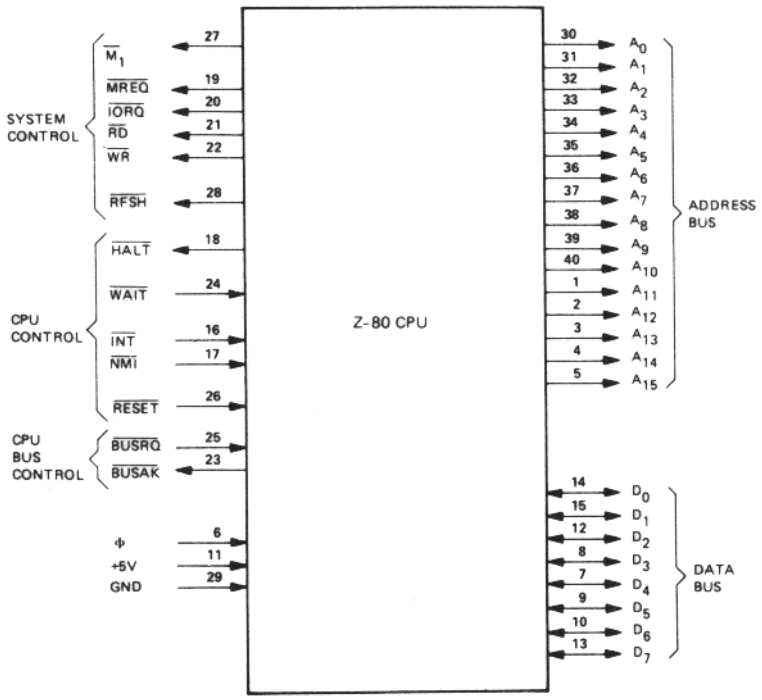
Istituto Tecnico Industriale " E. Fermi " - Ascoli Piceno

Specializzazione Elettronica e Telecomunicazioni

Classe IV Sez. A

Appunti delle Lezioni di Teoria dei Sistemi

Sistemi a Microprocessore
Linguaggio Assembly dello Z80
Programmazione Assembly dello Z80



Anno Scolastico 1999/2000

Prof. Lanfranco Curzi

Sistemi a Microprocessore

1-1 Introduzione ai µP

Nel progetto degli automi, si é visto che fondamentalmente esiste la corrispondenza:

1 Progetto ⟶ 1 Circuito (Hardware)

Il cercare di risolvere un problema (ove ciò é possibile) con questa tecnica, porta alla soluzione in **logica cablata** dello stesso, la soluzione si dice anche che é **solo di tipo Hardware**.

Il procedere in questo modo, aveva portato lo sviluppo di nuove applicazioni nei campi dell'elettronica e dell'automazione, ad un vicolo cieco sul finire degli anni 60, dato che ogni applicazione richiedeva una complessa e costosa fase progettuale e realizzativa.

Con l'avvento e la commercializzazione a basso costo del **microprocessore** (µP), quindi con lo sviluppo dei dispositivi in **logica programmabile**, lo sviluppo di un progetto viene fatto con la corrispondenza:

1 Progetto ⟶ 1 Circuito (Hardware) + 1 Programma specifico (Software)

Fermo restando la non varianza della parte più difficoltosa da realizzare (l'HW del dispositivo ovvero il µP + circuiteria di supporto), ciascun progetto (di una stessa tipologia) può essere realizzato ponendo in esecuzione una particolare procedura di funzionamento dell'HW: lo specifico **programma** per quel progetto.

Eseguendo un altro programma, il sistema funziona in un altro modo completamente differente.

L'avvento del µP ha rivoluzionato con l'informatica e sta rivoluzionando con la telematica il modo di vivere e lavorare dell'uomo.

1-2 Organizzazione di un sistema a µP

I dispositivi programmabili con µP, chiamati anche in genere microComputer hanno:

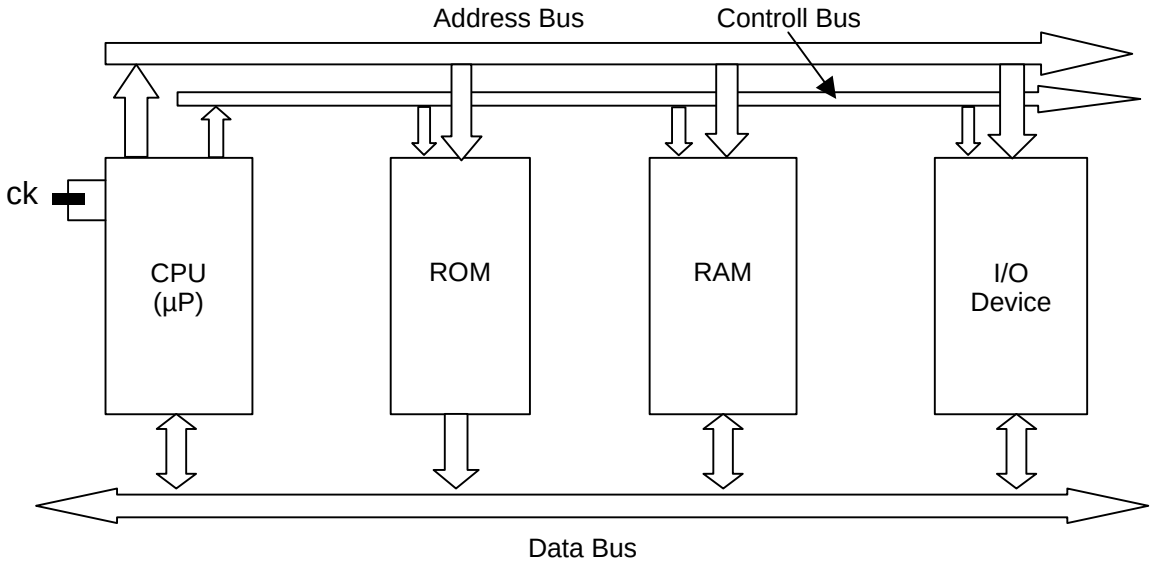
- forme;
- dimensioni;
- costi;
- usi

differenti.

Dal punto di vista logico-funzionale la stragrande maggioranza, possono essere ricondotti una **architettura a bus**:

Nel quale, su una unica board (piastra) vengono predisposti 3 collegamenti in parallelo tra linee di una stessa tipologia, chiamati:

1) **Bus Dati** Data Bus
2) **Bus Indirizzi** Address Bus
3) **Bus Controllo** Control Bus



Nello schema riportato notiamo:

a) La CPU (Central Processi Unit) cioé il cuore di tutto il sistema costituito dal µP;

b) Le memorie operative a semiconduttore:

ROM permanenti a sola lettura, nelle quali sono memorizzate le routine (programmi) fondamentali del **sistema operativo** del dispositivo

RAM volatili di lettura/scrittura, nelle quali sono memorizzati i programmi e i dati di che permettono di svolgere al sistema, il compito ad esso assegnato

c) I dispositivi di Input/Output (I/O Device), sono tutto quanto può essere utile e unzionale al sistema come :

monitor, stampanti, tastiera, mouse, hard disk, sistemi per acquisizione dati, attuatori, ecc.

d) Sul Bus Dati sono trasferiti **i dati e le istruzioni** trattati dal sistema

e) Sul Bus Indirizzi sono predisposte le stringhe di bit necessarie a selezionare (**indirizzare**) sia le locazioni di memoria ROM e/o RAM prescelte, sia i dispositivi I/O selezionati

f) Sul Control Bus sono predisposti i segnali (bit) di controllo necessari alla **sincronizzazione e comando** di tutte le operazioni svolte dal sistema.

Tutto il dispositivo funzione nel seguente modo:

- Esiste un programma da eseguire formato da una sequenza di istruzioni, esso può già esistere sia nelle memorie ROM, oppure viene caricatodalle memorie di massa esterne (in genere floppy ed hard disk) sulle memorie RAM

- Il µP

Carica una istruzione (detto anche **codice operativo**) dalle memorie a semiconduttore (chiamate anche memorie operative) e la trasferisce in un suo registro interno chiamato **registro istruzioni**

Questa operazione viene chiamata fase di **fetch**

- Il µP

  Decodifica l'istruzione (interpreta il codice operativo) ed la esegue, cioé:

  Carica al suo interno e/o trasferisce agli altri dispositivi del sistema dei dati;

  Invia opportune sequenze di segnali (bit) per il controllo di azioni esterne

  Si predispone ad eseguire la successiva istruzione del programma

  Questa operazione viene detta fase di **excute**

Le operazioni per l'esecuzione del programma devono essere eseguite con il sincronismo di un segnale di clock, esterno al µP.

Nei sistemi a µP di ultima generazione si tende ad avere 2 segnali di clock per il sistema:

- Uno con una frequenza maggiore che, da il sincronismo alle operazioni interne al µP
- Uno con una frequenza minore che, sincronizza le operazione con i bus esterni (per bus interni si intendono le linee di collegamento tra i blocchi funzionali all'interno del µP)

1-3 Caratteristiche principali dei µP

Le caratteristiche fondamentali che determinano le potenzialità di un µP sono:

a) Dimensione della **word** (parola) trattata, cioé il numero di bit minimo per formare una istruzione, ovvero in genere il numero di bit gestibili dal bus dati, commercialmente per i sistemi più usati abbiamo µP a: 8, 16, 32, 64 bit

b) Frequenza di clock (esterna ed interna)

c) Velocità di esecuzione delle istruzioni, espressa in**Mips** (milioni di istruzioni per secondo)

d) Capacità di indirizzamento della memoria, la quale può essere classificata del tipo:

- **Reale**, rappresenta il numero di locazioni di memoria (espresso in byte) che l'address bus può direttamente indirizzare e che formano l'estensione delle ROM+RAM

- **Virtuale**, rappresenta un numero di locazioni di memoria nettamente superiore a quello rappresentato dalle ROM+RAM, comprendente quindi anche le memorie di massa.

e) Presenza del coprocessore matematico

Il coprocessore matematico svolge funzioni e operazioni logiche/matematiche che nei primi µP erano svolti nella ALU dello stesso.

Successivamente é stato realizzato allo scopo, un integrato distinto dal µP e funzionante in sincronismo con lo stesso.

La tendenza attuale é quella di abbinare in un unico chip coprocessore matematico e µP

f) Presenza di memoria cache di primo livello

Con la memoria cache di primo livello si intende un certo banco di memoria presente nel chip del µP, avente lo scopo di ridurre il numero di accessi alle memorie operative esterne, onde velocizzare l'esecuzione di un programma.

Le potenzialità di un µP **non determinano da sole** le caratteristiche funzionali di tutto il sistema, potremo ad esempio avere un µP di grande qualità che, posto nel contesto di elementi (ROM, RAM, Bus, interfaccie I/O, etc) di mediocri caratteristiche, determini alfine un sistema di scadenti proprietà.

1-4 Memorie a Semiconduttore - Introduzione

Le memorie a semiconduttore hanno varie tipologie costruttive capaci di immagazzinare l'informazione binaria, da un punto di vista funzionale esse sono in grado di gestire per ogni singolo indirizzo, una **word** (parola) formata da un certo numero di bit.

Le più diffuse memorie presentano una organizzazione interna di:

- N word da 1 bit
- N word da 4 bit
- N word da 8 bit

dove N rappresenta il numero di locazioni di memoria complessive della stessa, allo stato attuale per singolo chip si arriva a memorie con capacità di 64 Mbit.

Per **tempo di accesso** di una memoria si intende il tempo intercorrente tra una richiesta di lettura e/o scrittura ed il tempo nel quale il dato sarà fruibile dal richiedente.

1-5 Classificazione delle memorie a semiconduttore

Le memorie possono essere classificate in:

a) Memorie di sola lettura ROM, PROM, EPROM, EEPROM

b) Memorie a lettura e scrittura RAM (DRAM, SRAM), NOVRAM

Un'altra classificazione può essere fatta a seconda della tecnologia di realizzazione delle stesse:

1) Tecnologia Bipolare

2) Tecnologia Mos

1-6 Memorie a sola lettura

Sono quelle **non volatili**, nelle quali l'informazione è acquisita in modo permanente nelle stesse, per cui sono memorie a sola lettura. Esse si distinguono in:

a -1) ROM (Read Only Memory)

Definite da costruttore, non possono in alcun modo essere manipolate (per quanto riguarda la modifica delle informazioni in esse contenute) dall'utente.

a -2) PROM (Programmable ROM)

Possono essere programmate una sola volta dall'utente tramite un dispositivo detto programmatore di PROM. Queste memorie escono dalla fabbrica con tutte le locazioni allo stato logico alto, con opportuni impulsi di tensione di programmazione, si bruciano dei collegamenti interni per quelle locazioni che si vogliono portare a livello logico basso.

Non hanno grandi capacità di memorizzazione per la presenza dei fusibili di programmazione.

a -3) EPROM (Erasable PROM)

Rappresentano una evoluzione delle PROM, nel senso che possono essere programmate più volte, ogni volta cancellata completamente l'informazione in esse contenute.
La programmazione avviene con un dispositivo di programmazione che con opportuni impulsi di tensione di programmazione, permettono di modificare lo stato di conduzione degli elementi attivi selezionati.
La cancellazione dell'informazione è ottenuta mettendo a contatto il chip con una luce ultravioletta, prodotta da una lampada di Wood a 2537 Å per circa 20 minuti, in questo modo si ha un ritorno allo stato iniziale prima della programmazione, delle cariche di conduzione dell'elemento attivo della memoria.
Opportunamente schermata la finestrella di protezione del chip, l'informazione definita dall'utente rimane stabile per anni.
Questi dispositivi possono essere riprogrammati qualche decina di volte e raggiungono in genere capacità di memoria dell'ordine dei 128 Kbyte.

a -4) EEPROM (Electrical EPROM)

Rappresentano una evoluzione delle EPROM, rispetto alle quali hanno i seguenti fondamentali vantaggi:

- Possibilità di riprogrammazione fino a circa 100.000 volte;

- Riprogrammazione e cancellazione dei dati in modo elettrico anche di un singolo bit, invece nella EPROM la riprogrammazione parziale comporta la cancellazione di tutta la memoria.

Esse hanno costi maggiori rispetto alle EPROM.

1-7 Memorie a lettura e scrittura - RAM (Random Access Memory)

Sono memorie volatili, in quanto l'informazione in esse contenute scompare se non sono più alimentate.
La loro classificazione (SRAM e DRAM) è strettamente collegata alla tecnologia con la quale esse sono realizzate:

a-1) Memorie a tecnologia polare – SRAM

In esse l'elemento attivo capace di immagazzinare un singolo bit è rappresentato dal transistor BJT in funzionamento modo interdizione e/o saturazione, esse vengono in genere chiamate **memorie statiche**;

a-2) Memorie a tecnologia unipolare – DRAM

In esse l'elemento attivo è rappresentato dal transistor ad effetto di campo (transistor di tipo MOS), l'informazione di un singolo bit è collegata allo stato di carica di un condensatore collegato al MOS, il condensatore carico corrisponde allo stato 1 del bit di memoria, il condensatore scarico corrisponde allo stato 0 del bit di memoria, esse in genere vengono chiamate **memorie dinamiche**.

a-3) Confronto tra le caratteristiche delle DRAM e SRAM

- Rispetto alle DRAM le SRAM hanno:

  Tempi di accesso minori, quindi maggiore velocità;

  Per contro hanno una densità di integrazione per singolo chip minore in quanto hanno un certo assorbimento di potenza dovuto al funzionamento dei BJT

- Rispetto alle SRAM le DRAM hanno:

  Maggiore densità di integrazione per singolo chip, in quanto il funzionamento in teoria è collegato alla carica di condensatori e non a dissipazione di potenza;

  Per contro poiché i condensatori tendono nel tempo a scaricarsi perdendo cosi l'informazione, essi devono essere costantemente e periodicamente ripristinati nel livello di carica, a questo assolvono opportuni circuiti di rinfresco (**refresh**). La presenza di detti circuiti e l'esecuzione del rinfresco rendono queste memorie meno veloci rispetto alle SRAM.

Normalmente le DRAM rappresentano il grosso delle memorie operative di un sistema a µP, le SRAM trovano impiego nei sistemi di memoria tipo **cache**, cioè in memorie di transito molto veloci interposte tra il µP e le DRAM.

b-4) Memorie NOVRAM (Non Volatile RAM)

Sono memorie di ultima generazione, composte da 2 aree identiche, una RAM ed una EEPROM. Durante il funzionamento normale i dati sono trattati dalla RAM, ma allo spegnimento del sistema essi sono trasferiti nell'area EEPROM.

1-8 Indirizzamento di una memoria

Dal punto di vista funzionale la memoria ha un numero di linee degli indirizzi in relazione alla sua capacità,come ad esempio é riportato nella seguente tabella

N° Linee	Byte	Kbyte	Mbyte
8	256		
9	512		
10	1024	1	
11	2048	2	
12	4096	4	
13	8192	8	
14	16634	16	
15	32768	32	
16	65536	64	
17		128	
18		256	
19		512	
20			1

Una organizzazione esadecimale degli indirizzi particolarmente utile per un sistema a µP Z80 é di seguito riportata:

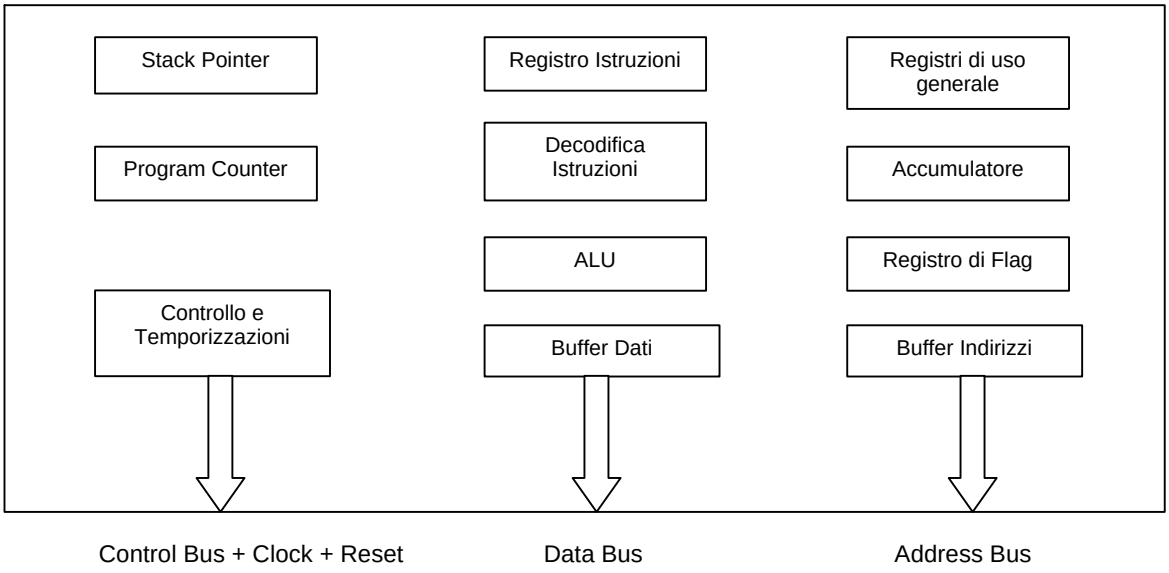
0000		
03FF	1 Kbyte	
0400		
07FF	1Kbyte	4 Kbyte
0800		
0BFF	1Kbyte	
0C00		
0FFF	1Kbyte	

0000		
0FFF	4 Kbyte	
1000		
1FFF	4Kbyte	16 Kbyte
2000		
2FFF	4Kbyte	
3000		
3FFF	4Kbyte	

0000	16 Kbyte	64 Kbyte
3FFF		
4000	16Kbyte	
7FFF		
8000	16Kbyte	
BFFF		
C000	16Kbyte	
FFFF		

1-9 Struttura interna di un µP

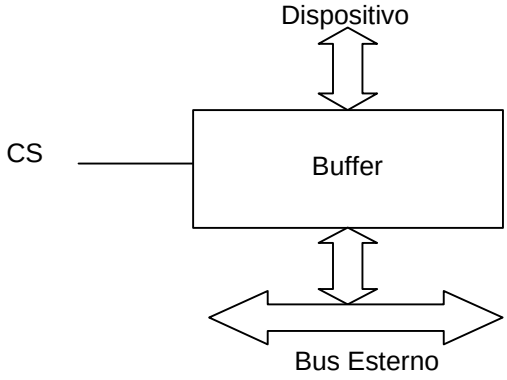
Ogni µP ha una propria architettura interna, dal punto di vista funzionale esso é carattetizzato dai seguenti blocchi operativi:



dove i vari blocchi funzionali sono collegati tra loro tramite opportuni bus interni.

a) Buffer Dati e Buffer Indirizzi

Tutti i dispositivi che dialogano con i bus esterni, non sono collegati direttamente ad essi, per evitare problemi di conflittualità ed interferenze varie il collegamento ai bus é fatto tramite opportuni dispositivi chiamati **buffer**.



Un buffer può essere monodirezionale o bidirezionale, nella figura é riportato un buffer bidirezionale.

Se in dispositivo non é abilitato (CS = 1), tra esso ed il bus esterno vi é una condizione di **alta impedenza** detta anche condizione **tri-state** (Hi-Z), che non permette lo scambio delle informazioni tra i dispositivi.

b) **ALU**

La ALU rappresenta l'unità aritmetico logica del µP, integrata nello stesso e sostituita dal coprocessore matematico nei dispositivi più evoluti.

Essa svolge tutte le operazioni logico e/o matematiche del sistema

c) **Registri di uso generale**

Rappresentano le "locazioni di memoria" interne al µP ove memorizzare le variabili del programma in esecuzione.

I registri interni sono in numero estremamente ridotto, quindi continuamente bisogna dialogare durante l'esecuzione di un programma con le memorie operative.

d) **Accumulatore**

E' un registro che si differenzia da quelli di uso generale fondamentalmente per 2 proprietà:

1) Su di esso vanno a finire i risultati delle operazioni della ALU

2) Esso ha un set di istruzioni che, ne fa il registro privilegiato per lo scambio di dati tra µP e dispositivi esterni (memorie e I/O device)

e) **Registro Istruzioni**

Come di é già detto, il registro istruzioni contiene il **codice operativo** della istruzione in esecuzione

f) **Registro di Flag**

Il registro di flag contiene valori di bit che sono influenzati dalle operazioni svolte dalla ALU, esso viene usato per le operazioni di controllo effettuate nel programma in esecuzione.

g) **Program Counter - PC**

Questo registro viene chiamato anche "contatore di programma", in esso risiede l'indirizzo di memoria dell'istruzione di programma in esecuzione

Quando il µP carica il codice operativo di una istruzione, il registro PC viene aggiornato al valore dell'indirizzo del codice operativo dell'istruzione successiva da eseguire.

All'inizio dell'esecuzione di un programma il registro PC deve essere inizializzato al primo indirizzo del programma.

h) **Stack Pointer - SP**

Questo registro assolve a 2 funzioni:

1) Su di esso é salvato il contenuto del registro PC quando si esegue una **subroutine** (sottoprogramma) del programma principale

2) Su di esso vengono memorizzati temporaneamente i valori dei registri del µP, quando questi registri in fase di esecuzione del programma devono contenere altre variabili

i) **Decodifica Istruzione - Controllo e Temporizzazioni**

Rappresentano i blocchi funzionali (non gestibili direttamente dal programmatore) che interpretano il codice operativo di ogni istruzione ed, inviano su control bus i segnali di comando per l'esecuzione esterna dell'istruzione stessa.

1-10 Il µP Z80

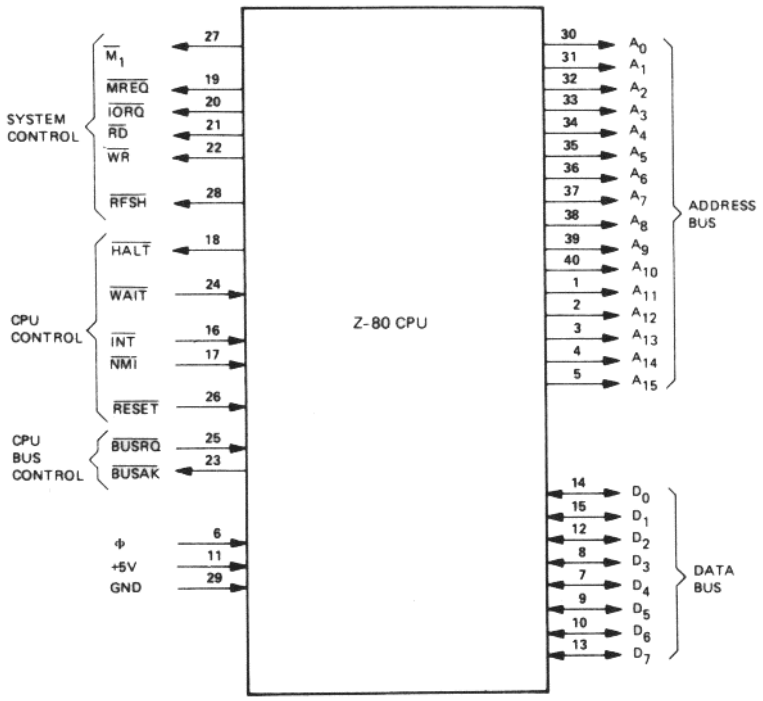
Lo Z80 é un µP presentato nel 1976 dalla Zilog, sul finire degli anni 70 ed inizio anni 80 ha avuto molto successo anche come base per sistemi di elaborazione dati, allo stato attuale viene ancora utilizzato per sistemi di controllo industriale di non elevata complessità.

Malgrado sia un prodotto superato tecnologicamente, dal punto di vista didattico risulta ancora attuale per comprendere il funzionamento di sistemi a µP.

Lo Z80 é un µP ad 8 bit, con address bus a 16 bit, quindi con capacità di indirizzamento diretta pari a 64 Kbyte.

Esso é realizzato nella forma dual-pin a 20+20 piedini.

Dal punto di vista funzionale con la nomenclatura definita dal costruttore, i pin del µP possono essere raggruppati nel modo seguente:



a) **Address bus**

Sono le linee contraddistinte dai segnali da A0 a A15

b) **Data Bus**

Sono le linee contraddistinte dai segnali da D0 a D15

c) **System Control**

Con queste linee il µP controlla i dispositivi esterni, come si può osservare, l'attivazione dei segnali di controllo avviene con il livello logico 0.

Le operazioni di System Control sono eseguite attivando una o più delle seguenti linee:

c-1) **M1 Primo Ciclo Macchina**

Quando la CPU é nella fase di fetch, essa segnala all'esterno che é in corso la lettura di un codice operativo.

c-2) **MREQ Memory Request**

Con l'attivazione di questo segnale, il µP segnala che sull'address bus é presente l'indirizzo valido di una locazione di memoria operativa

c-3) **IOREQ IO Request**

Con l'attivazione di questo segnale il µP, segnala la fase di execute di lettura e/o scrittura su un I/O device

c-4) **RD** **Read**

Attivo insieme a MREQ e/o IOREQ, segnala il trasferimento dati dalla memoria e/o da un I/O device verso il µP

c-5) **WD** **Write**

Attivo insieme a MREQ e/o IOREQ segnala il trasferimento dati dal µP alla memoria e/o od a un I/O device

c-6) **RFSH** **Refresh**

E' attivo durante la fase di fetch e segnala che é in corso il refresh delle memorie DRAM eventualmente presenti nel sistema

d) **CPU Control**

Con queste linee i dispositivi esterni influenzano il funzionamento del µP, l'attivazione dei segnali di CPU control avviene con il livello logico 0.

d-1) **HALT**

Se é attivo questo segnale il µP é in uno stato di NOP (not operation) nel quale esso si limita ad attivare le sole operazioni di refresh.

L'uscita dallo stato di HALT avviene tramite una Interrupt oppure azionando il Reset.

d-2) **INT** **Interrupt Request**

Attivando un dispositivo di I/O questa linea, se é attivo anche il segnale di BUSRQ, il µP esegue se autorizzato dalla CPU un **interrupt mascherabile** nella quale:

- blocca l'esecuzione del programma principale
- attiva l'esecuzione di una sobroutine specifica per quella richiesta
- ritorna successivamente all'esecuzione del programma principale

d-3) **NMI** **Non Maskable Interrupt**

Attivando un dispositivo di I/O questa linea, se é attivo anche il segnale di BUSRQ, il µP esegue un **interrupt non mascherabile** nella quale:

- blocca l'esecuzione del programma principale
- attiva l'esecuzione di una sobroutine partendo dall'indirizzo 0066

- ritorna successivamente all'esecuzione del programma principale

d-4) **WAIT**

E' un segnale che viene attivato ogni volta che un dispositivo esterno **non é pronto** a ricevere e/o trasmettere dati al µP, testando la linea del WAIT la CPU allunga i tempi di lettura e/o scrittura dati con i dispositivi esterni.

d-5) **RESET**

E' il segnale che **inizializza** il funzionamento del sistema

e) **CPU Bus Control**

L'attivazione dei segnali di CPU Bus Control controllo avviene con il livello logico 0.

e-1) **BUSRQ Bus Request**

Questo segnale é attivato da un dispositivo di I/O, il quale chiede che venga posto sotto il suo controllo i bus del sistema.

Il µP alla richiesta, termina l'istruzione in corso e successivamente si pone in uno stato di HALT portando in alta impedenza le sue linee di comunicazioni.

Un esempio tipoco di richiesta dei Bus,é quando si esegue una DMA (**Direct Memory Access**) nella quale i dispositivi di I/O dialogano con le memorie operative del sistema.

e-2) **BUSAK Bus Acknowledge**

E' il segnale di risposta inviato dal µP per comunicare l'accettazione di un BUSRQ, senza l'attivazione di questo segnale la periferica non può iniziare il controllo dei Bus.

f) **Clock**

Il segnale di clock (TTL compatibile) permette la sincronizzazione delle operazioni della CPU, lo Z80 lavora con frequenze di clock tra 2 e 8 MHz

1-11 I Registri dello Z80

I registri interni dello Z80 sono registri a 8 e 16 bit, i primi sono contraddistinti da 1 lettera, i secondi da 2.

Registri a 8 bit:

a) Registri di uso generale A, B, C, D, E, F, H, L

di questi in particolare

A = Accumulatore

F = Registro di Flag

F

7	6	5	4	3	2	1	0
S	Z	X	H	X	P/V	N	C

S = flag di segno
Z = flag di zero
H = flag di half carry
P/V = flag di parity/overflow
N = flag di addizione/sottrazione
C = flag di carry

I registri di uso generale possono essere usati a 16 bit tramite gli accoppiamenti:

AF, BC, DE, HL

Di particolare importanza é l'uso del registro HL il quale, ha un proprio set d'istruzioni particolari quali per le operazioni aritmetiche e per l'indirizzamento ed il trasferimento dati dalla memoria.

b) Registri copia dei registri generali A', B', C', D', E', F', H', L'

c) Registro d'interrupt I

d) Registro di refresh R

Registri a 16 bit:

1) Stack Pointer SP

2) Program Counter PC

3) Registri Indice IX, IY

Linguaggio Assembly dello Z80

2-1 Aspetti generali sulla programmazione di un µP

La programmazione di un µP, ha lo scopo di fare eseguire materialmente alla macchina un determinato problema già logicamente organizzato con l'algoritmo dello stesso.

Il sistema a µP per propria costituzione esegue operazioni in logica binaria, questo modo di procedere per l'uomo risulta praticamente improponibile, dato che esso da millenni ha costruito un propio modo di ragionare completamente differente.

Il **linguaggio di programmazione** rappresenta **l'interfaccia** ovvero la connessione tra il modo di procedere della macchina ed il modo di procedere dell'uomo.

Esistono disparati linguaggi di programmazione ognuno dei quali, ha specifiche caratteristiche e pecularietà.

Secondo la più semplice schematizzazione possibile, essi li possiamo dividere in 3 grandi categorie:

a) Linguaggi di programmazione a basso livello (linguaggi macchina);

b) Linguaggi di programmazione ad alto livello;

c) Pacchetti applicativi avanzati

Le categorie a) e b) sono utilizzate dagli specialisti della programmazione i quali se ne servono per fornire agli utilizzatori generici i SW della categoria c)

Nella programmazione a basso livello, il programmatore **deve conoscere in dettaglio** la struttura hardware della macchina sulla quale girerà il SW che esso si propone di realizzare, poche sono le utility del **sistema operativo** a cui esso potrà far ricorso.

Per sistema operativo si intendono tutti quei programmi di utilità che il costruttore della macchina mette a disposizione in varia forma all' utente.

I programmi di sistema operativo risiedono all'interno del calcolatore (nelle ROM) e nelle memorie di massa del sistema (Hard-Disk) dalle quali volta per volta il sistema é programmato a prelevarne delle parti a seconda delle specifiche richieste.

Senza alcun programma di sistema operativo materialmente il Computer non può funzionare, infatti già i messaggi che compaiono sul monitor di un calcolatore rappresentano l'esecuzione di un particolare **programma di start** dello stesso.

Nella programmazione ad alto livello, la conoscenza della struttura hardware del sistema é molto meno pronunciata, anche se sono necessarie conoscenze specifiche dello stesso HW per applicazioni importanti quali:

- Reti di Computer
- Gestione dei File

Nell'utilizzazione di pacchetti applicativi avanzati, si é in mano completamente ai programmi di sistema operativo, l'utilizzatore (il quale può anche programmare in modo significativo applicazioni importanti quale ad esempio Excel, Access, Autocad, etc.) può anche completamente ignorare la struttura HW del calcolatore in uso.

2-2 Programmazione Assembly dello Z80

Di seguito si tratterà la programmazione in linguaggio macchina di sistemi con il µP Z80.

Il programma che il µP deve eseguire come si é già detto, deve essere presente **nell'area programma** delle memorie ROM o RAM.

Una istruzione é formata idealmente da 2 termini:

a) **Codice operativo**

b) **Operando**

Il codice operativo formato da 1 fino a 3 byte, interpretato dal µP determina la natura dell'operando il quale, può dar luogo:

c) trasferimenti di 1 o 2 byte
d) operazioni aritmetico-logiche
e) indirizzamenti vari

L'operando se é formato da dei byte, segue nell'area programma i byte del codice operativo, ad esempio l'istruzione:

LD B, 57

significa: carica sul registro B il byte 57 (espresso in esadecimale -hex); e pertanto nell'area di programma d'indirizzo 1000 e successivi, la stessa istruzione sarà memorizzata con i seguenti byte:

Indirizzo Memoria	Bytes Istruzione	Commento
1000	06	Byte di Codice Operativo
1001	57	Byte operando

Ad esempio l'istruzione:

LD B, A

significa: carica in B il contenuto del registro A; essa non ha byte di operando

Indirizzo Memoria	Bytes Istruzione	Commento
1000	47	Byte di Codice Operativo

Materialmente le istruzioni da eseguire sono costituite da stringhe di numeri binari, anche se queste stringhe usualmente si compattano con l'uso del codice esadecimale, rimangono sempre alla mente umana di difficile comprensione e memorizzazione.

Pertanto la programmazione viene fatta con un opportuno linguaggio **mnemonico** chiamato **linguaggio assembly**, nel quale le istruzioni vengono organizzate in un modo da essere maggiormente comprensibile alla mente umana.

2-3 Set di istruzioni dello Z80

Le istruzioni dello Z80 possono essere divise fondamentalmente nei seguenti blocchi logici - funzionali:

1) Istruzioni di caricamento a 8 bit
2) Istruzioni di caricamento a 16 bit
3) Istruzioni di controllo della CPU
4) Istruzioni aritmetiche (a 8 bit e 16 bit)
5) Istruzioni di salto
6) Istruzioni logiche
7) Istruzioni di bit, reset, set
8) Istruzioni di rotazione e di shift
9) Istruzioni di call e retun
10) Istruzioni di trasferimento e ricerca di blocchi
11) Istruzioni di scambio
12) Istruzioni di input e output

2-4 Istruzioni di caricamento a 8 bit

Premesso che ogni istruzione di caricamento (Load) ha la sintassi:

LD destinazione, sorgente

- Destinazione = registro, memoria, I/O device
- Sorgente = registro, memoria, i/O device, dato numerico

le istruzioni di **caricamento** o **indirizzamento** ad 8 bit , danno luogo al trasferimento di 1 byte e, si dividono in:

a) **Istruzioni di indirizzamento tra registri**

LD r, r'	esempio	LD A, C	carica il byte di C in A

Queste istruzioni sono formate dal solo byte di codice operativo

b) **Istruzioni di indirizzamento diretto**

LD A, (nn)	esempio	LD A, (4012)	carica in A il byte contenuto in memoria d'indirizzo 4012
LD (nn), A	esempio	LD (0100), A	carica in 0100 il byte contenuto nell'accumulatore A

da notare che questa istruzione può essere usata con il solo accumulatore A

Queste istruzioni sono formate da: 1 byte di codice operativo
2 bytes di operandi (l'indirizzo di memoria)

c) **Istruzioni di indirizzamento indiretto tramite il registro HL**

LD r, (HL)	esempio	LD B, (HL)	carica in B il byte di memoria il cui indirizzo risiede in HL
LD (HL), r	esempio	LD (HL), D	carica nella memoria il cui indirizzo risiede in HL il byte contenuto in D

Queste istruzioni sono formate dal solo byte di codice operativo

d) **Istruzioni di indirizzamento indiretto tramite i registri indice IY ed IX**

Hanno modalità di trasferimento come le istruzioni di cui al punto c), al posto del registro HL in questo caso si usano un dei due registri indice IX e IY, secondo la sintassi:

LD (IX+d), r **LD r, (IX+d)**
LD (IY+d), r **LD r, (IY+d)**

d = byte di spiazzamento (displacement), la memoria indirizzata é data dal risultato della somma in complemento a 2 tra il valore di IX o IY con il valore di d

d in valore decimale varia tra -128 e +127

Queste istruzioni sono formate da: 2 bytes di codice operativo
1 byte di operando (lo spiazzamento d)

e) **Istruzioni di indirizzamento immediato**

LD r, n	esempio	LD E, B2	carica il byte B2 in E

Queste istruzioni sono formate da: 1 byte di codice operativo
1 byte di operando (il dato)

f) **Istruzioni di indirizzamento dei registri I ed R**

I registri specializzati: **I** (registro delle interruzioni) e **R** (registro di refresh), hanno con il solo accumulatore A, la stessa tipologia d'istruzione dell'indirizzamento tra registri

LD I, A **LD A, I**
LD R, A **LD A, R**

Queste istruzioni sono formate dal solo byte di codice operativo

2-5 Istruzioni di caricamento a 16 bit

Le istruzioni di caricamento a 16 bit con modalità analoghe, trasferiscono 2 bytes su un registro a 16 bit o nell'area di memoria.

Esse si dividono in:

a) **Indirizzamento immediato esteso**

Con 2 bytes si carica uno dei seguenti registri:

BC, DE, HL, SP, IX, IY

LD rr', nn esempio LD HL, 402C

Queste istruzioni sono formate da: 1 o 2 bytes di codice operativo
2 bytes di operando (i bytes da caricare)

b) **Indirizzamento diretto esteso**

LD rr', (nn) esempio LD BC, (4088) carica in C il byte indirizzato da 4088 e carica in B il byte indirizzato da 4089

esse coinvolgono sempre 2 locazioni adiacenti di memoria

Queste istruzioni sono formate da: 1 byte di codice operativo
2 bytes di operando

c) **Indirizzamento tra SP ed i registri HL, IX, IY**

LD SP, rr' esempio LD SP, HL carica in P il byte contenuto in L e carica in S il byte contenuto in H

esse coinvolgono sempre lo stack pointer SP, mentre i registri sorgente possono essere: HL, IX, IY

Queste istruzioni sono formate da: 1o 2 bytes di codice operativo

Tra le istruzioni di caricamento a 16 bit ci sono le istruzioni di PUSH e POP che verranno analizzate successivamente.

2-6 Istruzioni di controllo della CPU

Sono istruzioni che predispongono il certo funzionamento della CPU

a) **NOP** (No OPeration)

Non effettua nessuna operazione nella CPU, anche se:

per eseguirla necessitano 4 periodi di clock (1 periodo di clock viene usualmente chiamato **ciclo T**);

da luogo all'incremento del Program Counter

b) **HALT**

Essa arresta l'esecuzione di un programma, per cui la CPU si blocca e per ripartire attende un **RESET** oppure una **interrupt** (interruzione)

c) **Istruzioni di set e reset del flag di carry**

SCF ⟶ pone a 1 (stato di set) il flag di carry ⟶ cy = 1

CCF ⟶ pone a 0 (stato di reset) il flag di carry⟶ cy = 0

Tra le istruzioni di controllo della CPU ci sono le istruzioni di definizione delle modalità dell'interrupt: IM0, IM1, IM2 che verranno viste successivamente.

2-7 Istruzioni di Restart

Le istruzioni di RESTART sono delle **chiamate** a istruzioni residenti nella **pagina zero** di un sistema a Z80.

Per pagina zero si intendono gli indirizzi in memoria compresi tra 0000 e 00FF

Mnemonico	Operazione svolta
RST 00	L'esecuzione del programma salta a 0000
RST 08	L'esecuzione del programma salta a 0008
RST 10	L'esecuzione del programma salta a 0010
RST 18	L'esecuzione del programma salta a 0018
RST 20	L'esecuzione del programma salta a 0020
RST 28	L'esecuzione del programma salta a 0028
RST 30	L'esecuzione del programma salta a 0030
RST 38	L'esecuzione del programma salta a 0038

Sono istruzioni con 1 solo byte di codice operativo, normalmente gli indirizzi delle Restart sono Il lancio di sottoprogrammi del sistema operativo.

Ad esempio la RST 38 in genere serve per terminare l'esecuzione di un programma e restituire il controllo delle operazioni al sistema operativo del µP.

2-8 Istruzioni aritmetiche

Le istruzioni aritmetiche che coinvolgono operandi ad 8 bit oppure a 16 bit, sono molto importanti nella manipolazione dei dati da parte della CPU.

Nella pratica lo Z80 é in grado di **svolgere direttamente** solo operazioni di **addizione** e **sottrazione**.

Le istruzioni aritmetiche si suddividono in

1) Istruzioni di addizione ADD
2) Istruzioni di addizione con carry ADC
3) Istruzioni di sottrazione SUB

4) Istruzioni di sottrazione con prestito SBC
5) Istruzioni di incremento INC
6) Istruzioni di decremento DEC

2-9 Istruzioni di addizione a 8 bit

a) Istruzioni di addizione a 8 bit senza riporto

Mnemonico	Operazione svolta
ADD A, n	A = A + n
ADD A, B	A = A + B
ADD A, C	A = A + C
ADD A, D	A = A + D
ADD A, E	A = A + E
ADD A, F	A = A + F
ADD A, H	A = A + H
ADD A, L	A = A + L
ADD A, (HL)	A = A + (HL)
ADD A, (IX+d)	A = A + (IX+d)
ADD A, (IY+d)	A = A + (IY+d)

Dopo l'esecuzione di una di queste istruzioni vengono modificati tutti i bit utili del registro di Flag F, che assume la seguente configurazione:

flag	Valore del flag	Tipo di risultato
7	S = 0	Il risultato é positivo
6	Z = 0	Il risultato non é zero
5	x	x
4	H = 0	Non c'é mezzo riporto
3	x	x
2	P/V = 0	Non c'é overflow
1	N = 0	L'operazione é un'addizione
0	C = 0	Non c'é riporto

b) Istruzioni di addizione a 8 bit con il riporto

Queste istruzioni differiscono dalle precedenti per il solo fatto di aggiungere ai due termini della somma il valore in quel momento presente nel flag di carry

Mnemonico	Operazione svolta
ADC A, n	A = A + n + cy
ADC A, B	A = A + B + cy
ADC A, C	A = A + C + cy
ADC A, D	A = A + D + cy
ADC A, E	A = A + E + cy
ADC A, H	A = A + H + cy
ADC A, L	A = A + L + cy
ADC A, (HL)	A = A + (HL) + cy
ADC A, (IX+d)	A = A + (IX+d) + cy
ADC A, (IY+d)	A = A + (IY+d) + cy
ADC A, A	A = A + A + cy

2-10 Istruzioni di addizione a 16 bit

Le istruzioni di addizione a 16 bit, hanno come registro destinatario del risultato uno dei seguenti: HL, IX, IY

l'altro registro necessario per svolgere l'operazione é uno dei registri: BC, DE, HL, IX, IY, SP

Mnemonico	Operazione svolta
ADD HL, BC	HL = HL + BC
ADD HL, DE	HL = HL + DE
ADD HL, HL	HL = HL + HL
ADD HL, SP	HL = HL + SP
ADD IX, BC	IX = IX + BC
ADD IX, DE	IX = IX + DE
ADD IX, HL	IX = IX + HL
ADD IX, SP	IX = IX + SP
ADD IY, BC	IY = IY + BC
ADD IY, DE	IY = IY + DE
ADD IY, HL	IY = IY + HL
ADD IY, SP	IY = IY + SP
ADC HL, BC	HL = HL + BC + cy
ADC HL, DE	HL = HL + DE + cy
ADC HL, HL	HL = HL + HL + cy
ADC HL, SP	HL = HL + SP + cy

Le addizioni a 16 bit non agiscono su tutti i flags di F

flag	Valore del flag	Tipo di risultato
7	S	Non modifica il flag S
6	Z	Non modifica il flag Z
5	x	x
4	H	H = 1 se c'é riporto dal 11° bit, altrimenti é H = 0
3	x	x
2	P/V = 0	Qualunque sia il risultato
1	N = 0	Qualunque sia il risultato
0	C = 0	C = 1 se c'é riporto dal 15° bit, altrimenti é C = 0

2-11 Istruzioni di sottrazione binaria a 8 bit

Mnemonico	Operazione svolta
SUB n	A = A - n
SUB A	A = A - A
SUB B	A = A - B
SUB C	A = A - C
SUB D	A = A - D
SUB E	A = A - E
SUB H	A = A - H
SUB L	A = A - L
SUB (HL)	A = A + (HL)
SUB (IX+d)	A = A - (IX+d)
SUB (IY+d)	A = A - (IY+d)

Dopo l'esecuzione di una di queste istruzioni vengono modificati tutti i bit utili del registro di Flag F, che assume la seguente configurazione:

flag	Valore del flag	Tipo di risultato
7	S = 1	Il risultato é negativo
6	Z = 0	Il risultato non é zero
5	x	x
4	H = 0	Non c'é mezzo riporto
3	x	x
2	P/V = 0	Non c'é overflow
1	N = 1	L'operazione é una sottrazione
0	C = 1	C'é prestito dal bit MSB

2-12 Istruzioni di sottrazione binaria a 8 e 16 bit con prestito

Vengono usate quando si devono compiere sottrazioni a 16 e 32 bit

Mnemonico	Operazione svolta
SBC n	A = A - n - borrow
SBC A	A = A - A - borrow
SBC B	A = A - B - borrow
SBC C	A = A - C - borrow
SBC D	A = A - D - borrow
SBC E	A = A - E - borrow
SBC H	A = A - H - borrow
SBC L	A = A - L - borrow
SBC (HL)	A = A + (HL) - borrow
SBC (IX+d)	A = A - (IX+d) - borrow
SBC (IY+d)	A = A - (IY+d) - borrov
SBC HL, BC	HL = HL - BC - borrow
SBC HL, DE	HL = HL - DE - borrow
SBC HL, HL	HL = HL - HL - borrow
SBC HL, SP	HL = HL - SP - borrow

2-13 Istruzioni di incremento a 8 e 16 bit

Permettono di incrementare di 1 il registro e/o la locazione di memoria al quale l'istruzione é applicata.

Gli incrementi a 8 bit modificano i bit del registro F, ad eccezione del bit di carry cy

Gli incrementi a 16 bit non modificano i bit di F

Mnemonico	Operazione svolta	Registro e/o locazione memoria
INC r	r = r +1	r = A, B, C, D, E, H, L
INC (HL)	(HL) = (HL) + 1	(HL)
INC rr'	rr' = rr' +1	rr' = BC, DE, HL, SP, IX, IY
INC (IX+d)	(IX+d) = (IX+d) + 1	IX
INC (IY+d)	(IY+d) = (IY+d) + 1	IY

2-14 Istruzioni di decremento a 8 e 16 bit

Permettono di decrementare di 1 il registro e/o la locazione di memoria al quale l'istruzione é applicata.

I decrementi a 8 bit modificano i bit del registro F, ad eccezione del bit di carry cy

I decrementi a 16 bit non modificano i bit di F

Mnemonico	Operazione svolta	Registro e/o locazione memoria
DEC r	r = r -1	r = A, B, C, D, E, H, L
DEC (HL)	(HL) = (HL) - 1	(HL)
DEC rr'	rr' = rr' -1	rr' = BC, DE, HL, SP, IX, IY
DEC (IX+d)	(IX+d) = (IX+d) - 1	IX
DEC (IY+d)	(IY+d) = (IY+d) - 1	IY

2-15 Istruzioni di confronto CP

Le istruzioni di confronto eseguono delle **sottrazioni virtuali** tra il registro A ed il registro, il dato o la locazione di memoria al quale l'istruzione é applicata.

Queste istruzioni quindi agiscono **sul solo contenuto** del registro di flag F

Mnemonico	Operazione svolta	Registro, locazione memoria, dato
CP n	A = A - n	n = byte di dato
CP r	A = A - r	r = A, B, C, D, E, H, L
CP (HL)	A = A -(HL)	(HL)
CP (IX+d)	A = A - (IX+d)	(IX+d)
CP (IY+d)	A = A - (IY+d)	(IY+d)

Le condizioni di F sono:

flag	Valore del flag	Tipo di risultato
7	S = 0	Il risultato é positivo
6	Z = 1	Il risultato non é zero
5	x	x
4	H = 0	Non c'é il prestitoo
3	x	x
2	P/V = 0	Non c'é overflow
1	N = 1	L'operazione é una sottrazione
0	C = 1	Non c'é prestito

2-16 Istruzioni di salto incondizionato

Le istruzioni di salto permettono di cambiare l'ordine di esecuzione delle istruzioni di un programma.

In definitiva esse cambiano il contenuto del Program Counter dell'istruzione successiva al salto.

Le istruzioni di salto si dividono in:

a) Istruzioni di salto incondizionato

b) Istruzioni di salto condizionato

Le istruzioni di salto incondizionato sono l'equivalente del GOTO ad esempio del Pascal e, si classificano come:

Mnemonico	Operazione svolta
JP nn	PC = nn
JP (HL)	PC = (HL)
JP (IX)	PC = (IX)
JP (IY)	PC = (IY)
JR d	PC = PC + d

JP = salto assoluto, in questo caso l'esecuzione del programma può essere spostata ad una locazione qualsiasi dell'area di memoria

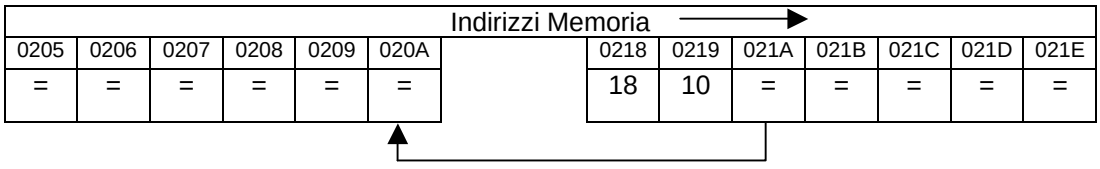
JR = salto relativo, in questo caso dal valore del PC l'esecuzione del programma può essere:

1) spostata in avanti di +127 locazioni
2) all'indietro di -128 locazioni

dato che il displecement (d) si somma al PC in complemento a 2

Ad esempio supponiamo che all'indirizzo 0218 di memoria, il programma prevede l'istruzione JR 10, il µP dopo la fase di fetch normalmente porta il PC al valore 021A, a questo punto somma il valore 10, cioé esegue

PC = (021A + 10) in complemento a 2



021A + 10

2-17 Istruzioni di salto condizionato

In questo caso lo spostamento dell'esecuzione del programma deriva dal riscontro di una condizione logica, come per quelle già visteesse si suddividono in:

a) JP salto assoluto condizionato

b) JR salto relativo condizionato

Mnemonico	Operazione svolta
JP Z, nn	PC = nn se Z = 1 cioé il risultato dell'operazione **é zero**
JP NZ, nn	PC = nn se Z = 0 cioé il risultato dell'operazione **non é zero**
JP C, nn	PC = nn se C = 1 cioé il risultato dell'operazione **ha generato un riporto**
JP NC, nn	PC = nn se C = 0 cioé il risultato dell'operazione **non ha generato un riporto**
JP M, nn	PC = nn se S = 1 cioé il risultato dell'operazione **é negativo**
JP P, nn	PC = nn se S = 0 cioé il risultato dell'operazione **é positivo**
JP PE, nn	PC = nn se P/V = 1 cioé il risultato dell'operazione **da un overflow**
JP PO, nn	PC = nn se P/V = 0 cioé il risultato dell'operazione **non da un overflow**
JR Z, d	PC = PC + d se Z = 1 cioé il risultato dell'operazione **é zero**
JR NZ, d	PC = PC + d se Z = 0 cioé il risultato dell'operazione **non é zero**
JR C, d	PC = PC + d se C = 1 cioé il risultato dell'operazione **ha generato un riporto**
JR NC, d	PC = PC + d se C = 0 cioé il risultato dell'operazione **non ha generato un riporto**

2-18 Istruzione DJNZ d

Questa istruzione opera sul registro B e rappresenta una delle **macroistruzioni** svolte dallo Z80

Per macroistruzione intendiamo lo svolgimento nella esecuzione di un algoritmo , con un unico comando di 2 o più operazioni elementari

Dal punto di vista funzionale la DJNZ d effettua:

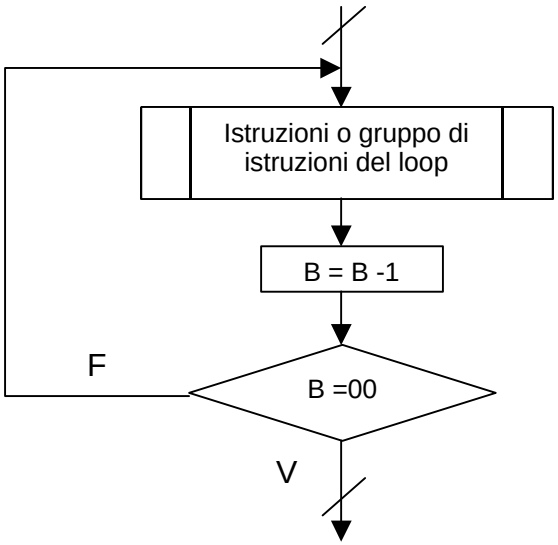
a) il decremento B = B -1

b) il controllo del valore B =00

c) d é il displacement che nella somma in complemento a 2 riporta il valore del PC a:

PC = PC + d

se B 00



Queta istruzione permette ad un programma con un loop di essere **rilocabile**, cioé lo stesso può essere fatto girare senza nessun problema a partire da indirizzi di programma diversi rispetto a quelli originari.

2-19 Istruzioni logiche

Sono le istruzioni: AND, OR, XOR, CPL, NEG

Per quanto riquarda CPL e NEG:

a) **CPL** ⟶ Complemento a 1 dell'accumulatore A

b) **NEG** ⟶ Complemento a 2 dell'accumulatore A

2-19 -1 Prodotto logico AND

L'istruzione AND esegue il prodotto logico tra il valore dell'accumulatore e quello di un altro operando che può essere anche l'accumulatore stesso.

Il prodotto logico é eseguito bit per bit corrispondenti, ad esempio per A AND B abbiamo:

Prima

A							
1	0	0	1	1	0	1	0

B							
0	1	1	1	0	0	1	1

Dopo

A							
0	0	0	1	0	0	1	0

B							
0	1	1	1	0	0	1	1

Mnemonico	Operazione svolta	Registro, locazione memoria, dato
AND n	A = A AND n	n = byte di dato
AND r	A = A AND r	r = A, B, C, D, E, H, L
AND (HL)	A = A AND (HL)	(HL)
AND (IX+d)	A = A AND (IX+d)	(IX+d)
AND (IY+d)	A = A AND (IY+d)	(IY+d)

Le condizioni assunte da F sono:

flag	Valore del flag	Tipo di risultato
7	S	Valore dipendente da risultato
6	Z	Valore dipendente da risultato
5	x	x
4	H = 1	Qualunque sia il risultato
3	x	x
2	P/V	Valore dipendente da risultato
1	N = 0	Qualunque sia il risultato
0	C = 0	Qualunque sia il risultato

2-19 -2 La somma logica OR

L'istruzione OR esegue la somma logica tra il valore dell'accumulatore e quello di un altro operando che può essere anche l'accumulatore stesso.

La somma logica é eseguita bit per bit corrispondenti, ad esempio per A OR B abbiamo:

Prima

A							
1	0	0	1	1	0	1	0

B							
0	1	1	1	0	0	1	1

Dopo

A							
1	1	1	1	1	0	1	1

B							
0	1	1	1	0	0	1	1

Mnemonico	Operazione svolta	Registro, locazione memoria, dato
OR n	A = A OR n	n = byte di dato
OR r	A = A OR r	r = A, B, C, D, E, H, L
OR (HL)	A = A OR (HL)	(HL)
OR (IX+d)	A = A OR (IX+d)	(IX+d)
OR (IY+d)	A = A OR (IY+d)	(IY+d)

Le condizioni assunte da F sono:

flag	Valore del flag	Tipo di risultato
7	S	Valore dipendente da risultato
6	Z	Valore dipendente da risultato
5	x	x
4	H = 0	Qualunque sia il risultato
3	x	x
2	P/V	Valore dipendente da risultato
1	N = 0	Qualunque sia il risultato
0	C = 0	Qualunque sia il risultato

2-19 -3 OR esclusivo XOR

L'istruzione XOR esegue l'OR esclusivo tra il valore dell'accumulatore e quello di un altro operando che può essere anche l'accumulatore stesso.

La XOR é eseguita bit per bit corrispondenti, ad esempio per A OR B abbiamo:

Prima

A							
1	0	0	1	1	0	1	0

B							
0	1	1	1	0	0	1	1

Dopo

A							
1	1	1	0	1	0	0	1

B							
0	1	1	1	0	0	0	1

Mnemonico	Operazione svolta	Registro, locazione memoria, dato
XOR n	A = A XOR n	n = byte di dato
XOR r	A = A XOR r	r = A, B, C, D, E, H, L
XOR (HL)	A = A XOR (HL)	(HL)
XOR (IX+d)	A = A XOR (IX+d)	(IX+d)
XOR (IY+d)	A = A XOR (IY+d)	(IY+d)

Le condizioni assunte da F sono:

flag	Valore del flag	Tipo di risultato
7	S	Valore dipendente da risultato
6	Z	Valore dipendente da risultato
5	x	x
4	H = 0	Qualunque sia il risultato
3	x	x
2	P/V	Valore dipendente da risultato
1	N = 0	Qualunque sia il risultato
0	C = 0	Qualunque sia il risultato

Ad esempio uno degli usi dell'istruzione é quella di annullare eseguendo XOR A, il contenuto dell'accumulatore

2-20 Istruzioni di SET, RES, BIT

Queste istruzioni permettono di **manipolare un singolo bit** di un registro o di una locazione di memoria, esse possono essere usate in alternativa alle istruzioni AND ed OR le quali, manipolano invece tutti i bit del byte trattato.

In queste istruzioni é necessario specificare la posizione del bit trattato con un numero di identificazione tra 0 e 7 in ordine crescente di peso.

2-20 -1 Istruzioni di SET

Sono istruzioni che agiscono su registri e locazioni di memoria a 8 bit e, consistono nel **settare a 1** il bit prescelto.

Ad esempio l'istruzione SET 5, B esegue:

Prima

B							
7	6	5	4	3	2	1	0
1	0	**0**	1	1	0	1	0

Dopo

B							
7	6	5	4	3	2	1	0
1	0	**1**	1	1	0	1	0

Mnemonico	Registro, locazione memoria
SET b, r	r = A, B, C, D, E, H, L
SET b, (HL)	(HL)
SET b, (IX+d)	(IX+d)
SET b, (IY+d)	(IY+d)

2-20 -2 Istruzioni di RES

Sono istruzioni che agiscono su registri e locazioni di memoria a 8 bit e, consistono nel **resettare a 0** il bit prescelto.

Ad esempio l'istruzione RES 0, B esegue:

Prima

B							
7	6	5	4	3	2	1	0
1	0	0	1	1	0	1	**1**

Dopo

B							
7	6	5	4	3	2	1	0
1	0	1	1	1	0	1	**0**

Mnemonico	Registro, locazione memoria
RES b, r	r = A, B, C, D, E, H, L
RES b, (HL)	(HL)
RES b, (IX+d)	(IX+d)
RES b, (IY+d)	(IY+d)

2-20 -3 Istruzioni di BIT

Sono istruzioni che agiscono su registri e locazioni di memoria a 8 bit e, permettono di **testare** il bit prescelto

L'operazione pone nel flag Z il **complemento del bit di test**

Ad esempio l'istruzione BIT 4, E esegue:

E							
7	6	5	4	3	2	1	0
1	0	0	**1**	1	0	1	1

Prima

Z
x

Dopo

Z
0

Mnemonico	Registro, locazione memoria
BIT b, r	r = A, B, C, D, E, H, L
BIT b, (HL)	(HL)
BIT b, (IX+d)	(IX+d)
BIT b, (IY+d)	(IY+d)

2-21 Istruzioni di rotazione

Le istruzioni di rotazione permettono spostare verso sinistra e/o destra i bit di un registro a 8 bit o di una locazione di memoria

Esse si dividono in:

a)	Istruzioni di rotazione a sinistra	RL (Rotate Left)
b)	Istruzioni di rotazione a destra	RR (Rotate Right)
c)	Istruzioni di rotazione a sinistra con carry	RLC (Rotate Left with Carry)
d)	Istruzioni di rotazione a destra con carry	RRC (Rotate Right with Carry)
e)	Istruzione RLD (Rotate Left Digit)	
f)	Istruzione RRD (Rotate Right Digit)	

2-21 -1 Istruzioni di rotazione a sinistra RL

La sorgente e la destinazione dell'istruzione coincidono

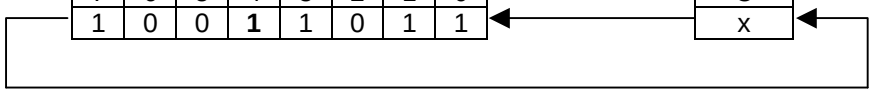
Mnemonico	Registro, locazione memoria
RL r	r = A, B, C, D, E, H, L
RLA	A
RL (HL)	(HL)
RL (IX+d)	(IX+d)
RL (IY+d)	(IY+d)

Ad esempio l'istruzione RL B esegue:

Prima

B							
7	6	5	4	3	2	1	0
1	0	0	**1**	1	0	1	1

C
x



Dopo

B							
7	6	5	4	3	2	1	0
0	0	1	**1**	0	1	1	x

C
1

Per il registro F

flag	Valore del flag	Tipo di risultato
7	S = 1	Risultato rotazione negativo
6	Z = 1	Risultato rotazione 0
5	x	x
4	H = 0	Qualunque sia il risultato
3	x	x
2	P/V = 1	Risultato rotazione pari
1	N = 0	Qualunque sia il risultato
0	C = 0	Contiene il valore del bit 7

L'istruzione RLA agisce come la RL A con la differenza:

- E' costituita da un solo byte
- Non modifica i flags: S, P/V, Z

Da notare che una azione RL esegue una **moltiplicazione per 2** del numero contenuto nel registro e/o memori, supposto che inizialmente il carry abbia bit 0

Prima

D							
7	6	5	4	3	2	1	0
0	0	0	1	1	0	1	**1**
(27)dec							

RL D

Dopo

D							
7	6	5	4	3	2	1	0
0	0	1	1	0	1	1	**0**
(54)dec							

2-21 -2 <u>Istruzioni di rotazione a destra RR</u>

La sorgente e la destinazione dell'istruzione coincidono

Mnemonico	Registro, locazione memoria
RR r	r = A, B, C, D, E, H, L
RRA	A
RR (HL)	(HL)
RR (IX+d)	(IX+d)
RR (IY+d)	(IY+d)

Ad esempio l'istruzione RR B esegue:

Prima

B							
7	6	5	4	3	2	1	0
1	0	0	**1**	1	0	1	1

C
x

Dopo

B							
7	6	5	4	3	2	1	0
x	1	0	**0**	1	1	0	1

C
1

Per il registro F

flag	Valore del flag	Tipo di risultato
7	S = 1	Risultato rotazione negativo
6	Z = 1	Risultato rotazione 0
5	x	x
4	H = 0	Qualunque sia il risultato
3	x	x
2	P/V = 1	Risultato rotazione pari
1	N = 0	Qualunque sia il risultato
0	C = 0	Contiene il valore del bit 0

L'istruzione RRA agisce come la RR A con la differenza:

- E' costituita da un solo byte
- Non modifica i flags: S, P/V, Z

Da notare che una azione RR esegue una **divisione per 2** del numero contenuto nel registro e/o memoria, supposto che inizialmente il carry abbia bit 0

Prima

D							
7	6	5	4	3	2	1	0
0	0	0	1	1	0	1	**1**
(27)dec							

RR D

Dopo

D							
7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	**1**
(13)dec							

Con resto = 1 nel carry

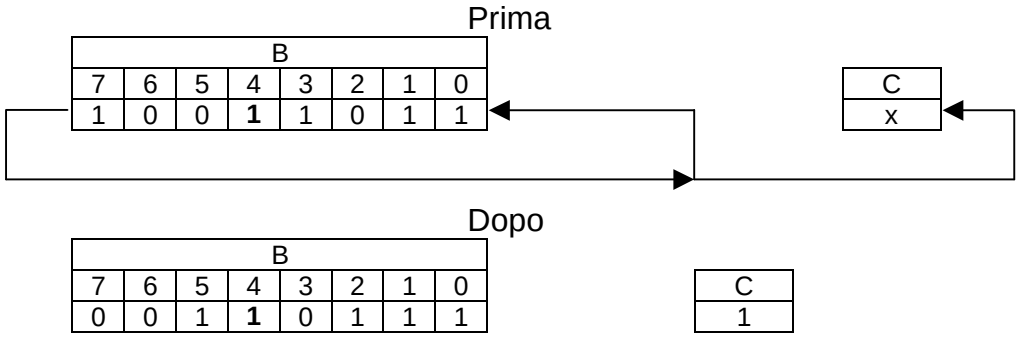
2-21 -3 Istruzioni di rotazione a sinistra RLC con carry

La sorgente e la destinazione dell'istruzione coincidono, in questo caso la differenza rispetto alle istruzioni RL é:

- viene perso il valore iniziale residente sul flag di carry
- Il bit 7 viene portato nel carry e sul bit 0

Mnemonico	Registro, locazione memoria
RLC r	r = A, B, C, D, E, H, L
RLCA	A
RLC (HL)	(HL)
RLC (IX+d)	(IX+d)
RLC (IY+d)	(IY+d)

Ad esempio l'istruzione RLC B esegue:



Per il registro F

flag	Valore del flag	Tipo di risultato
7	S = 1	Risultato rotazione negativo
6	Z = 1	Risultato rotazione 0
5	x	x
4	H = 0	Qualunque sia il risultato
3	x	x
2	P/V = 1	Risultato rotazione pari
1	N = 0	Qualunque sia il risultato
0	C = 0	Contiene il valore del bit 7

L'istruzione RLCA agisce come la RLC A con la differenza:

- E' costituita da un solo byte
- Non modifica i flags: S, P/V, Z

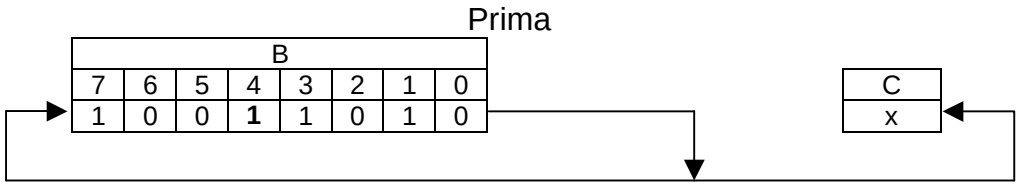
2-21 -4 Istruzioni di rotazione a destra RRC con carry

La sorgente e la destinazione dell'istruzione coincidono, in questo caso la differenza rispetto alle istruzioni RR é:

- viene perso il valore iniziale residente sul flag di carry
- Il bit 0 viene portato nel carry e sul bit 7

Mnemonico	Registro, locazione memoria
RRC r	r = A, B, C, D, E, H, L
RRCA	A
RRC (HL)	(HL)
RRC (IX+d)	(IX+d)
RRC (IY+d)	(IY+d)

Ad esempio l'istruzione RRC B esegue:



Dopo

B							
7	6	5	4	3	2	1	0
0	1	0	**0**	1	1	0	1

C
0

Per il registro F

flag	Valore del flag	Tipo di risultato
7	S = 1	Risultato rotazione negativo
6	Z = 1	Risultato rotazione 0
5	x	x
4	H = 0	Qualunque sia il risultato
3	x	x
2	P/V = 1	Risultato rotazione pari
1	N = 0	Qualunque sia il risultato
0	C = 0	Contiene il valore del bit 0

L'istruzione RRCA agisce come la RRC A con la differenza:

- E' costituita da un solo byte
- Non modifica i flags: S, P/V, Z

Un esempio di applicazione della RRC é quella di scambiare due numeri in BCD contenuti in un registro:

B							
7	6	5	4	3	2	1	0
0	0	1	**0**	1	0	0	0
Na				Nb			

Applicando 4 volte l'istruzione RRC B avremo:

B							
7	6	5	4	3	2	1	0
1	0	0	**0**	0	0	1	0
Nb				Na			

2-21 -5 Istruzione RLD - Rotate Left Digit

L'istruzione RLD al pari della sua equivalente RRD, opera tra l'accumulatore e un byte in memoria indirizzato da HL, essa é utilizzata per la manipolazione di numeri in formato BCD.

Prima di RLD

Dopo la RLD

2-21 -6 Istruzione RRD - Rotate Right Digit

L'istruzione RRD opera tra l'accumulatore e un byte in memoria indirizzato da HL, essa é utilizzata per la manipolazione di numeri in formato BCD.

Prima di RRD

Dopo la RLD

2-22 Istruzioni di shift

Le istruzioni di shift permettono lo scorrimento a destra od a sinistra di un bit, in un registro a 8 bit oppure in una locazione di memoria.

Queste istruzioni si dividono in:

a) shift logico a destra SRL (Shift Right Logical)
b) shift aritmetico a sinistra SRL (Shift Left Arithmetic)
c) shift aritmetico a destra SRA (Shift Right Arithmetic)

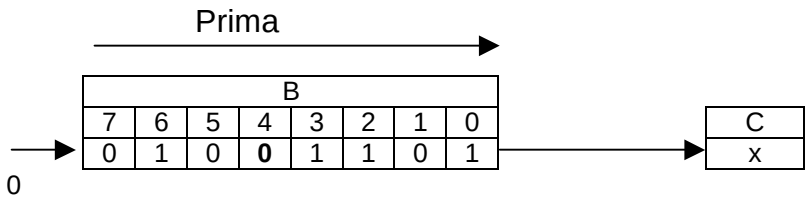
2-22 -1 Istruzioni di shift logico a destra SRL

L'istruzione SRL:

- Fa **scorrere a destra** tutti i bit della locazione di memoria o del registro
- Introduce uno 0 nel bit 7
- Salva il bit 0 nel carry che perde il suo valore iniziale
- Esegue la divisione per 2 del byte trattato

Mnemonico	Registro, locazione memoria
SRL r	r = A, B, C, D, E, H, L
SRL (HL)	(HL)
SRL (IX+d)	(IX+d)
SRL (IY+d)	(IY+d)

Ad esempio eseguendo l'istruzione SRL B:



Dopo

B							
7	6	5	4	3	2	1	0
0	0	1	**0**	0	1	1	0

C
1

Per il registro F

flag	Valore del flag	Tipo di risultato
7	S = 1	Risultato shift negativo
6	Z = 1	Risultato shift 0
5	x	x
4	H = 0	Qualunque sia il risultato
3	x	x
2	P/V = 1	Risultato shift pari
1	N = 0	Qualunque sia il risultato
0	C = 0	Contiene il valore del bit 0

2-22 -2 Istruzioni di shift aritmetico a sinistra SLA

L'istruzione SLA:

- Fa **scorrere a sinistra** tutti i bit della locazione di memoria o del registro
- Introduce uno 0 nel bit 0
- Salva il bit 7 nel carry che perde il suo valore iniziale
- Esegue la moltiplicazione per 2 del byte trattato

Mnemonico	Registro, locazione memoria
SRL r	r = A, B, C, D, E, H, L
SLA (HL)	(HL)
SLA (IX+d)	(IX+d)
SLA (IY+d)	(IY+d)

Ad esempio eseguendo l'istruzione SLA B:

Prima

B							
7	6	5	4	3	2	1	0
0	1	0	**0**	1	1	0	1

0

C
x

Dopo

B							
7	6	5	4	3	2	1	0
1	0	0	**1**	1	0	1	0

C
0

Per il registro F

flag	Valore del flag	Tipo di risultato
7	S = 1	Risultato shift negativo
6	Z = 1	Risultato shift 0
5	x	x
4	H = 0	Qualunque sia il risultato
3	x	x
2	P/V = 1	Risultato shift pari
1	N = 0	Qualunque sia il risultato
0	C = 0	Contiene il valore del bit 7

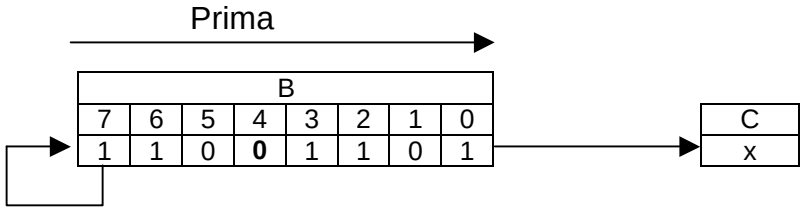
2-22 -3 Istruzioni di shift aritmetico a destra SRA

L'istruzione SRA:

- Fa **scorrere a destra** tutti i bit della locazione di memoria o del registro
- **Salva** il valore del bit 7
- Salva il bit 0 nel carry che perde il suo valore iniziale
- Esegue la divisione per 2 di un byte in rappresentazione con il segno (complemento a 2)

Mnemonico	Registro, locazione memoria
SRA r	r = A, B, C, D, E, H, L
SRA (HL)	(HL)
SRA (IX+d)	(IX+d)
SRA (IY+d)	(IY+d)

Ad esempio eseguendo l'istruzione SRA B:



Per il registro F

flag	Valore del flag	Tipo di risultato
7	S = 1	Risultato shift negativo
6	Z = 1	Risultato shift 0
5	x	x
4	H = 0	Qualunque sia il risultato
3	x	x
2	P/V = 1	Risultato shift pari
1	N = 0	Qualunque sia il risultato
0	C = 0	Contiene il valore del bit 0

2-23 Istruzione di CALL

Essa insieme all'istruzione RET, permette la gestione di sottoprogrammi (subroutines) all'interno del programma principale.

L'istruzione CALL

- carica in **modo diretto**
- carica in seguito al **test di un flag** del registro F

il registro PC all'indirizzo di partenza del sottoprogramma da eseguire

Il valore corrente del PC viene salvato automaticamente nell'area di Stack, definita dal valore del registro SP

Mnemonico	Registro, locazione memoria
CALL nn	PC = nn
CALL C, nn	PC = nn se C = 1
CALL NC, nn	PC = nn se C = 0
CALL Z, nn	PC = nn se Z = 1
CALL NZ, nn	PC = nn se Z = 0
CALL M, nn	PC = nn se S = 1
CALL P, nn	PC = nn se S = 0
CALL PE, nn	PC = nn se P/V = 1
CALL PO, nn	PC = nn se P/V = 0

Le modalità operative di una CALL sono le seguenti:

a) Supponiamo che nel programma principale all'indirizzo 0200 si incontri l'istruzione CALL FC00 (in codice macchina CD 00 FC

b) Supponiamo che l'area di Stack sia definita dal valore SP = FEA0
(SP definisce il **limite superiore** dell'area di Stack)

main subroutine

Area Programma →											
0200	0201	0202	0203	0204	0205	FC00	FC01	FC02	FC40	FC41
CD	00	FC	=	=	=	=	=	=	=	RET

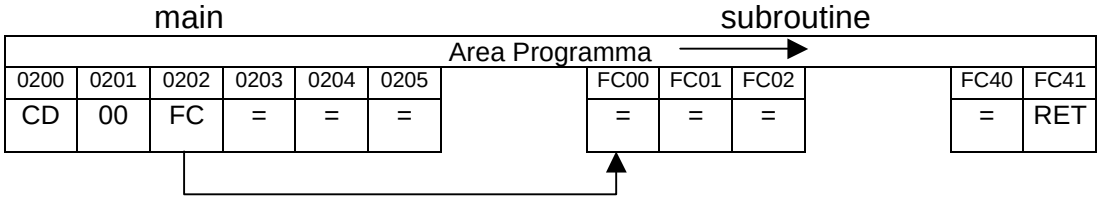
Area di Stack					
FE9B	FE9C	FE9D	FE9E	FE9F	**FEA0**

c) L'esecuzione di CALL FC00 al termine della fase di fetch della stessa:

- Salva nell'area di stack l'indirizzo 0203 dell'istruzione successiva alla CALL esattamente ponendo in

(SP - 1) byte MSB 02
(SP - 2) byte LSB 03

- Pone SP = FE9E
- Carica in PC l'indirizzo FC00 della prima istruzione del sottoprogramma



0200	0201	0202	0203	0204	0205	FC00	FC01	FC02	FC40	FC41
CD	00	FC	=	=	=	=	=	=	=	RET

Area di Stack					
FE9B	FE9C	FE9D	**FE9E**	FE9F	FEA0
			03	02	

Successivamente, quando nel sottoprogramma si incontrerà una istruzione RET, il registro PC sarà caricato al valore 0203 ed il µP ripartirà ad eseguire il programma principale.

Il valore dell'indirizzo del PC, viene nell'area di stack **caricato** (con la CALL) e **scaricato** (con la RET) automaticamente secondo la tecnica **LIFO** (Last Input First Output)

2-24 Istruzione di RET

L'istruzione RET

- carica in **modo diretto**
- carica in seguito al **test di un flag** del registro F

il registro PC all'indirizzo di interruzione del programma in esecuzione ed interrotto a seguito di una CALL

Mnemonico	Registro, locazione memoria
RET	Ritorna al main programm
RET C	Ritorna al main se C = 1
RET NC	Ritorna al main se C = 0
RET Z	Ritorna al main se Z = 1
RET NZ	Ritorna al main se Z = 0
RET M	Ritorna al main se S = 1
RET P	Ritorna al main se S = 0
RET PE	Ritorna al main se P/V = 1
RET PO	Ritorna al main se P/V = 0

Area di Stack all'esecuzione della RET (SP = FE9E)

Area di Stack					
FE9B	FE9C	FE9D	**FE9E**	FE9F	FEA0
			03	02	

Area di Stack dopo la RET (SP = FEA0)

Area di Stack					
FE9B	FE9C	FE9D	FE9E	FE9F	**FEA0**

2-25 Istruzioni PUSH e POP

Queste istruzioni permettono il salvataggio e il successivo recupero dall'area di Stack, dei valori dei registri dello Z80, allo scopo di utilizzarli nuovamente nell'ambito del programma in esecuzione.

a) **L'istruzione PUSH**:

salva nell'area di Stack il valore dei seguenti registri

Mnemonico	Registro, locazione memoria
PUSH AF	(SP) = AF
PUSH BC	(SP) = BC
PUSH DE	(SP) = DE
PUSH HL	(SP) = HL
PUSH IX	(SP) = IX
PUSH IY	(SP) = IY

Ad esempio si vuole salvare nella 'area di Stack (definita con SP =FEA0) i registri BC e HL, bisognerà eseguire in successione le istruzioni PUSH BC e PUSH HL

Prima di PUSH BC

Area di Stack					
FE9B	FE9C	FE9D	FE9E	FE9F	**FEA0**

Dopo PUSH BC (SP = FE9E)

Area di Stack					
FE9B	FE9C	FE9D	**FE9E**	FE9F	FEA0
			C	B	

Dopo PUSH HL (SP = FE9C)

Area di Stack					
FE9B	**FE9C**	FE9D	FE9E	FE9F	FEA0
	C	B	L	H	

b) **L'istruzione POP**:

recupera dall'area di Stack il valore dei seguenti registri

Mnemonico	Registro, locazione memoria
POP AF	AF = (SP)
POP BC	BC = (SP)
POP DE	DE = (SP)
POP HL	HL = (SP)
POP IX	IX = (SP)
POP IY	IY = (SP)

Per recuperare dall'area di Stack i valori dei registri BC e HL precedentemente salvati, bisognerà nell'ordine eseguire POP HL e POP BC

Dopo POP HL (SP = FE3E)

Area di Stack					
FE9B	FE9C	FE9D	**FE9E**	FE9F	FEA0
			C	B	

Dopo POP BC (SP = FEA0)

Area di Stack					
FE9B	FE9C	FE9D	FE9E	FE9F	**FEA0**

2-26 Istruzioni di scambio EX e EXX

Esse ad ogni esecuzione scambiano tra loro i valori di registri a 16 bit, lo scambio può anche essere effettuato tra registri usuali e registri copia

Mnemonico	Registro, locazione memoria
EX DE, HL	DE < -- > HL
EX AF, A'F'	AF < -- > A'F'
EX (SP), HL	(SP) < -- > HL
EX (SP), IX	(SP) < -- > IX
EX (SP), IY	(SP) < -- > IY
EXX	BC < -- > B'C' DE < -- > D'E'' HL < -- > H'L'

2-27 Istruzioni di trasferimento di blocchi dati

Sono delle macroistruzioni che permettono di trasferire tra diverse aree di memoria dei blocchi multibyte di dati.

Esse coinvolgono allo stesso modo i seguenti registri:

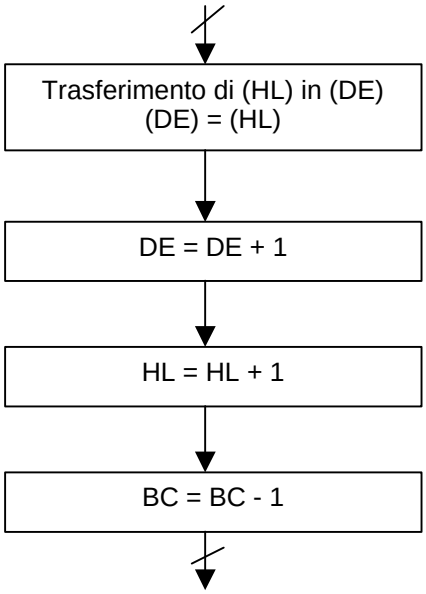
a) HL indirizza l'area memoria sorgente

b) DE indirizza l'area di memoria di destinazione

c) BC esegue il conteggio dei bytes da trasferire

2-27 -1 Istruzione di trasferimento LDI

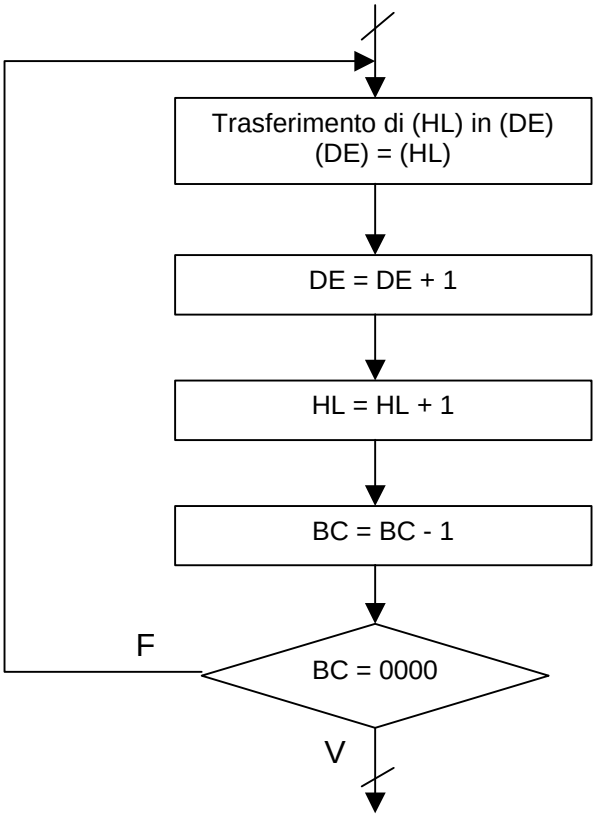
L'esecuzione di LDI (**carica ed incrementa**) svolge le seguenti operazioni fondamentali:



essa permette di trasferire un solo byte e di predisporre il trasferimento del successivo

2-27 -2 Istruzione di trasferimento LDIR

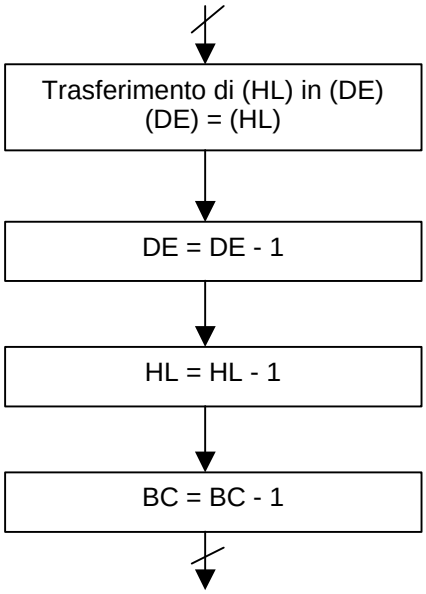
L'esecuzione di LDIR (**carica incrementa e ripeti**) svolge le seguenti operazioni fondamentali:



essa permette di trasferire l'intero blocco di dati

2-27 -3 Istruzione di trasferimento LDD

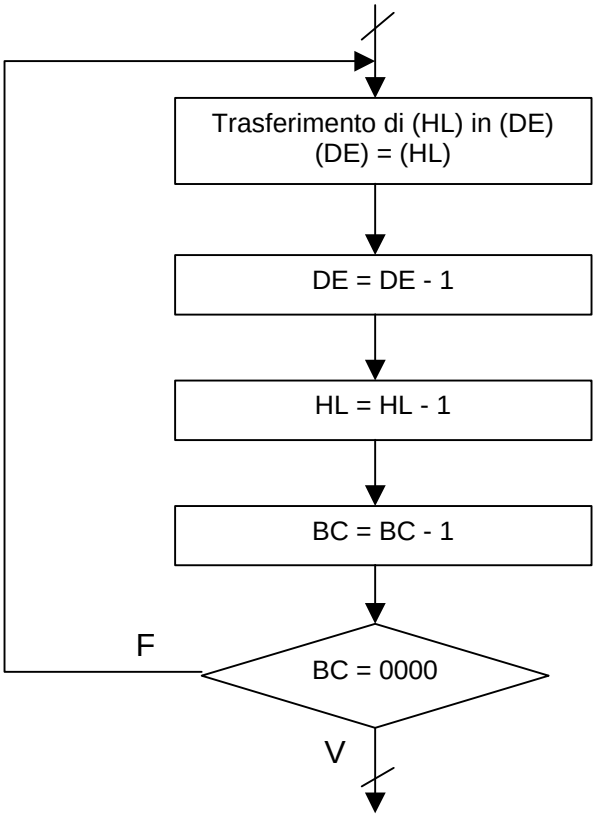
L'esecuzione di LDI (**carica e decrementa**) svolge le seguenti operazioni fondamentali:



essa permette di trasferire un solo byte e di predisporre il trasferimento del successivo

2-27 -4 Istruzione di trasferimento LDDR

L'esecuzione di LDIR (**carica decrementa e ripeti**) svolge le seguenti operazioni fondamentali:



essa permette di trasferire l'intero blocco di dati

2-28 Istruzioni di confronto su blocchi dati

Sono delle macroistruzioni che permettono di confrontare il dato contenuto sull'accumulatore con dati residenti in memoria

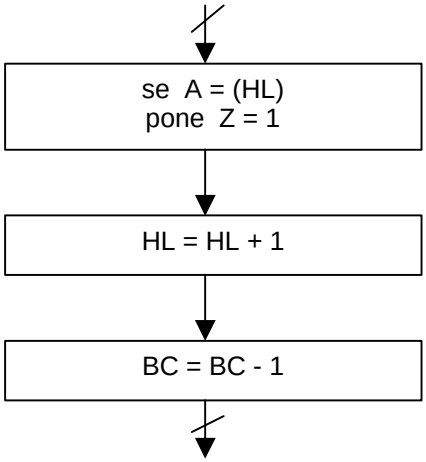
Esse coinvolgono allo stesso modo i seguenti registri:

a) HL indirizza l'area memoria

b) BC esegue il conteggio dei bytes di confronto

2-28 -1 Istruzione di confronto CPI

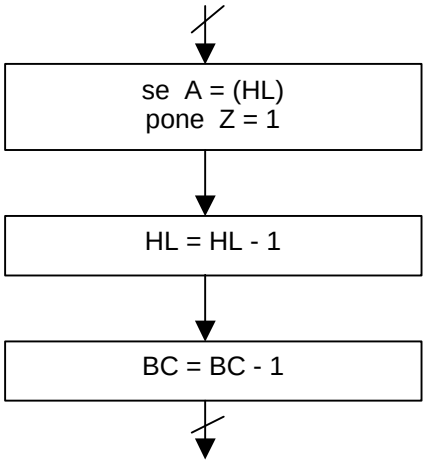
L'esecuzione di CPI (**confronta ed incrementa**) svolge le seguenti operazioni fondamentali:



Confronta il byte in memoria con A e predispone il confronto successivo

2-28 -2 Istruzione di confronto CPD

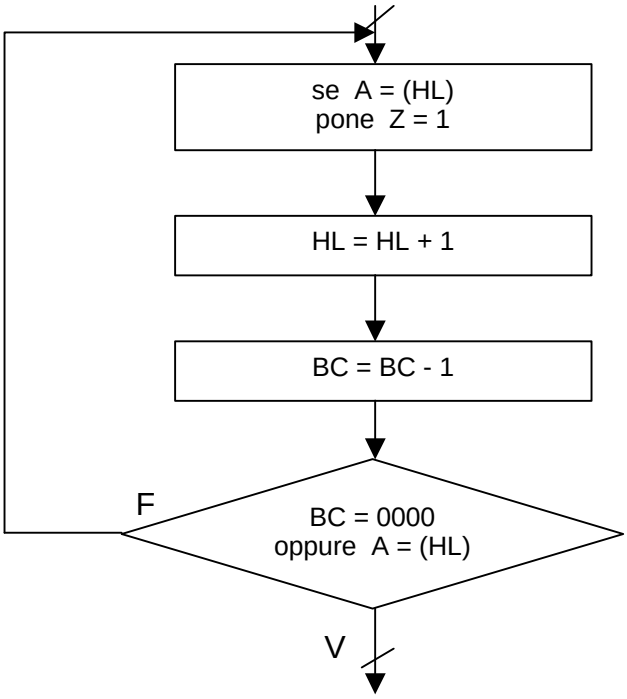
L'esecuzione di CPD (**confronta e decrementa**) svolge le seguenti operazioni fondamentali:



Confronta il byte in memoria con A e predispone il confronto successivo

2-28 -3 Istruzione di confronto CPIR

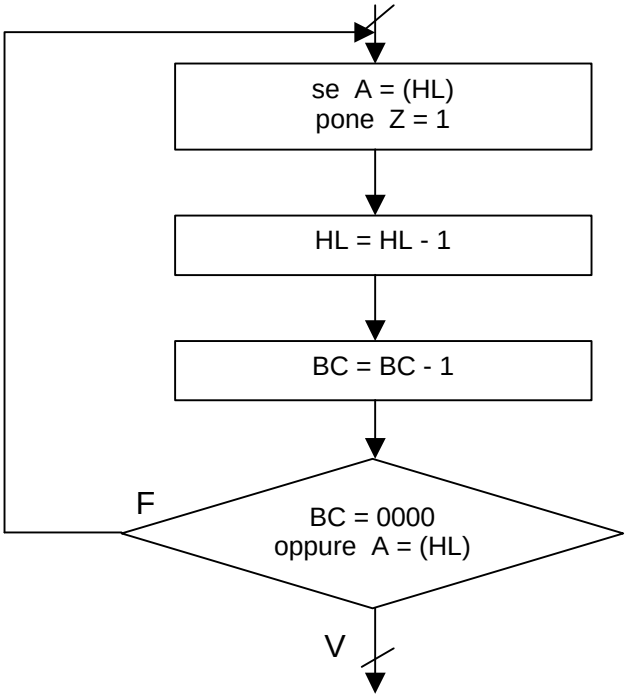
L'esecuzione di CPIR (**confronta incrementa e ripeti**) svolge le seguenti operazioni fondamentali:



Confronta il byte in memoria con A e continua ad eseguire il confronto fino a che A = (HL) oppure arriva al termine dei dati da esaminare.

2-28 -4 Istruzione di confronto CPDR

L'esecuzione di CPDR (**confronta decrementa e ripeti**) svolge le seguenti operazioni fondamentali:



Confronta il byte in memoria con A e continua ad eseguire il confronto fino a che A = (HL) oppure, arriva al termine dei dati da esaminare.

2-29 Istruzioni di OUT

Sono istruzioni dedicate al trasferimento di dati dalla CPU o dalle memorie verso le periferiche, esse si suddividono in:

a) Istruzioni di uscita di singolo byte

b) Istruzioni di uscita di blocchi di bytes

2-29 -1 Istruzioni di uscita di singolo byte

Mnemonico	Registri ed operazione svolte
OUT (n), A	(n) = A
OUT (C), r	(C) = r r = A, B, C, D, E, H, L

n = numero in esedecimale compreso tra 00 e FF contiene(come C) l'indirizzo della posta di uscita

2-29 -2 Istruzioni di uscita per blocchi di bytes

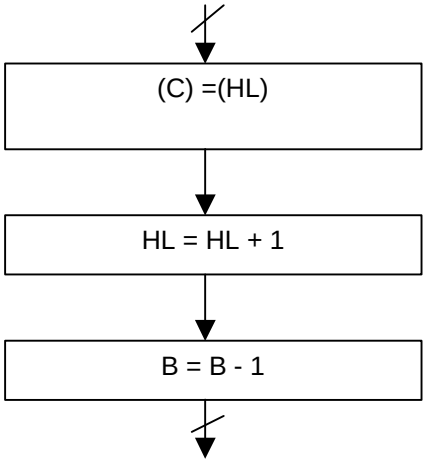
Permettono di trasferire in uscita blocchi di dati residenti in memoria, esse coinvolgono obbligatoriamente i registri:

a) HL per l'indirizzamento dei dati in memoria

b) B per il conteggio dei dati da trasferire

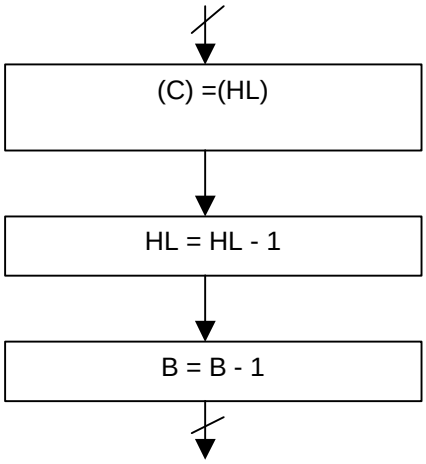
c) C per l'indirizzamento della periferica di OUT

1) **OUTI** **Out ed incrementa**



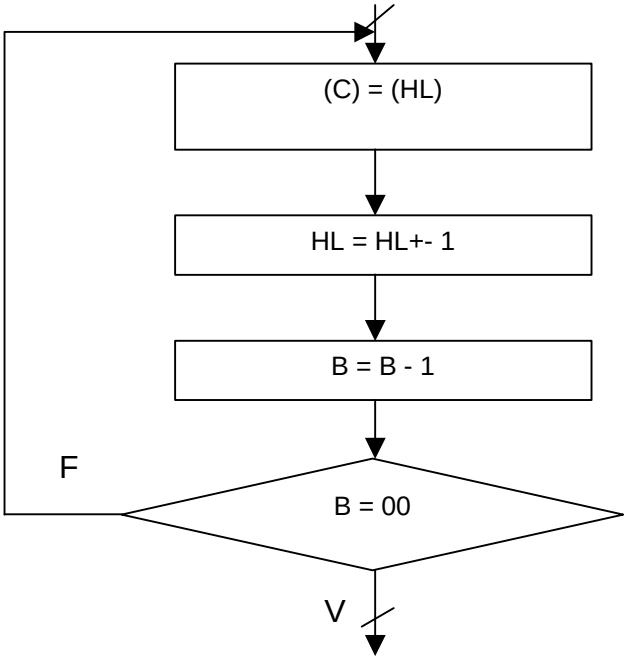
Trasferisce un byte sulla periferica e si predispone all'invio del successivo

2) **OUTD** **Out e decrementa**



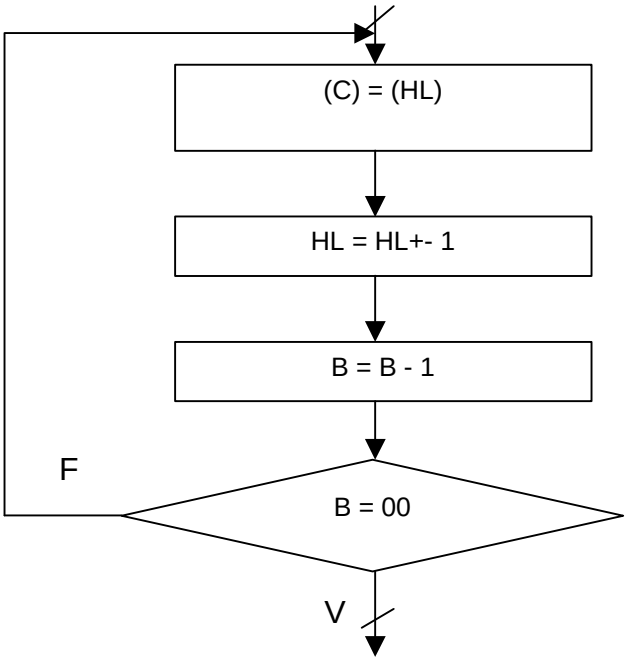
Trasferisce un byte sulla periferica e si predispone all'invio del successivo

3) **OTIR** **Out incrementa e ripeti**



Esegue il trasferimento dell'intero blocco di dati

4) **OTDR** **Out decrementa e ripeti**



Esegue il trasferimento dell'intero blocco di dati

2-30 Istruzioni di IN

Sono istruzioni dedicate al trasferimento di dati dalle periferiche verso la CPU o le memorie, esse si suddividono in:

a) Istruzioni di input di singolo byte

b) Istruzioni di input per blocchi di bytes

2-30 -1 Istruzioni di di input di singolo byte

Mnemonico	Registri ed operazione svolte
IN A, (n)	A = (n)
IN r, (C)	r = (C) r = A, B, C, D, E, H, L

n = numero in esedecimale compreso tra 00 e FF contiene(come C) l'indirizzo della posta di input

2-30 -2 Istruzioni di ingresso per blocchi di bytes

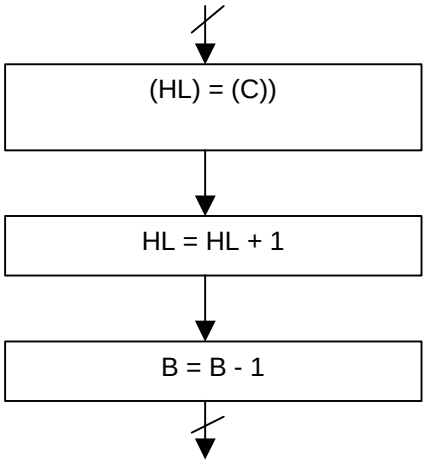
Permettono di trasferire in dalle periferiche alle memorie blocchi di dati, esse coinvolgono obbligatoriamente i registri:

a) HL per l'indirizzamento dei dati in memoria

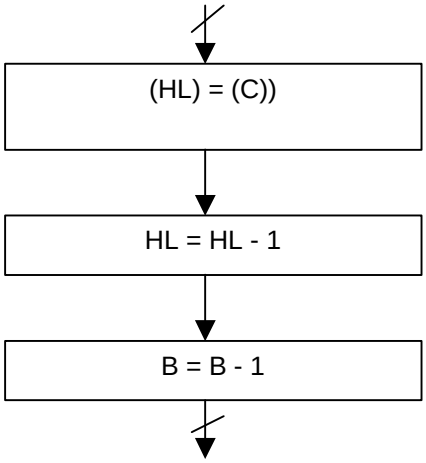
b) B per il conteggio dei dati da trasferire

c) C per l'indirizzamento della periferica di OUT

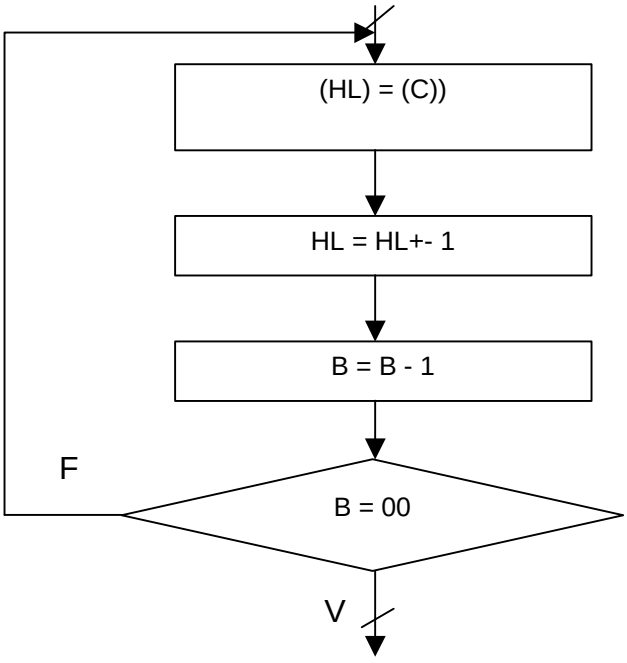
1) **INI** **Input ed incrementa**



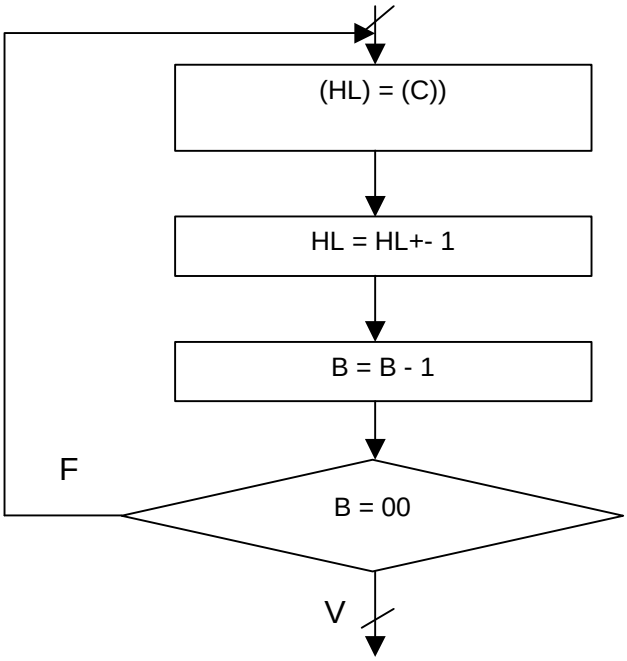
2) **IND** **Input e decrementa**



3) **INIR** **Input incrementa e ripeti**



4) **INDR** **Input decrementa e ripeti**



La Programmazione Assembly dello Z80

3-1 Strumenti di supporto per la programmazione in Linguaggio Assembly di un µP

Un buon programma professionale (a partire dalla programmazione in assembly) per lo sviluppo del software di un µP, deve avere le seguenti funzioni:

a) **Editor**

Costituito da un programma di elaborazione testi con il quale viene scritto l'assembly

b) **Assembler**

Trasforma il **codice sorgente** dell'assembly nel **codice macchina oggetto** del µP con tutti i dati e gli indirizzi associati

c) **Linker**

Il codice macchina oggetto generato dall'assembler non é eseguibile, per esserlo esso deve essere sottoposto al linker, il quale lo trasforma in **file eseguibile** del tipo EXE, HEX, etc.

Il linker permette di collegare tra loro diversi file generati separatamente, in modo da formare logicamente un'unica applicazione.

d) **Loader**

Permette di caricare ed eseguire l'applicazione creata, inoltre permette di **rilocare** in aree diverse di memoria del sistema a µP, programmi i quali in partenza erano stati scritti per una specifica area di memoria, questo é necessario quando caricando per l'esecuzione un modulo SW, non sono conosciuti a priori i moduli SW successivamente eseguiti.

e) **Debugger**

Permette di verificare la funzionalità (logica) del SW realizzato, questo lo si può fare in vari modi quali ad esempio:

- Esecuzione di una istruzione alla volta del programma (modo trace)
- Esecuzione di un numero limitato e ben definito di istruzioni
- Conversione inversa del programma dal codice oggetto al codice sorgente

3-2 Modalità di Edit di un programma Assembly

Rimanendo nell'ambito del solo Edit di un programma in Assembly, la stesura di un software specifico la faremo seguendo in singolo o contemporaneamente 2 vie:

1) Definizione del flow-chart del problema, usando come simbologia la stessa di quella dei registri e delle istruzioni dello Z80

2) Scrittura di ogni riga (**statement**) di programma suddivisa in 5 campi

- campo memoria nel quale viene indicato l'indirizzo di partenza dell'istruzione in esecuzione
- campo label (etichetta)
- campo codice mnemonico
- campo operando
- campo commento

Ad esempio l'istruzione: LD A, B che é eseguita a partire dal label "esempio", da luogo alla riga di'istruzione:

Memoria	Label	Codice	Operando	Commento
0100	esempio:	LD	A, B	Trasferisce B in A

3-3 Esempi di trasferimento di dati

a) Esempio N° 1

Trasferire il byte 4F nella locazione di memoria 0200

Memoria	Label	Codice	Operando	Commento
0100		LD	A, 4F	Carica in A il byte 4F
0102		LD	(0200), A	Trasferisci il byte nella memoria
0105		RST	38	Restituisci il controllo al S.O. del Computer

b) Esempio N° 2

Trasferire il byte dalla locazione di memoria 0300 in B

Poiché il trasferimento non può essere realizzato direttamente, esso lo si dovrà fare o tramite l'ausilio di A o di HL

Memoria	Label	Codice	Operando	Commento
0100		LD	A, (0300)	Carica in A il byte dalla memoria
0103		LD	B, A	Trasferisci il byte in B
0104		RST	38	Restituisci il controllo al S.O. del Computer

Memoria	Label	Codice	Operando	Commento
0100		LD	HL, 0300	Inizializza HL all'indirizzo di memoria
0103		LD	B, (HL)	Trasferisci il byte in B
0104		RST	38	Restituisci il controllo al S.O. del Computer

c) Esempio N° 3

Trasferire il byte dalla locazione di memoria 0200 nella locazione 0300

Utilizzando il registro A

Memoria	Label	Codice	Operando	Commento
0100		LD	A, (0200)	
0103		LD	(0300), A	
0106		RST	38	

Utilizzando il registro HL ed un registro di appoggio diverso da A

Memoria	Label	Codice	Operando	Commento
0100		LD	HL, 0200	
0103		LD	C, (HL)	
0104		LD	HL, 0300	
0106		LD	(HL), C	
0107		RST	38	

3-4 Esempio di addizione di due numeri a 2 bytes

Esempio N° 4

Sommare i numeri 045B e 789A e salvare il risultato a partire dall'indirizzo 3000

Memoria	Label	Codice	Operando	Commento
0100		LD	HL, 045B	Carica 1° termine somma
0103		LD	BC, 789A	Carica 2° termine somma
0106		ADD	HL, BC	
0107		LD	(3000), HL	Trasferisce in memoria il risultato
010A		RST	38	

3-5 Somma senza carry di una lista di numeri ad un byte

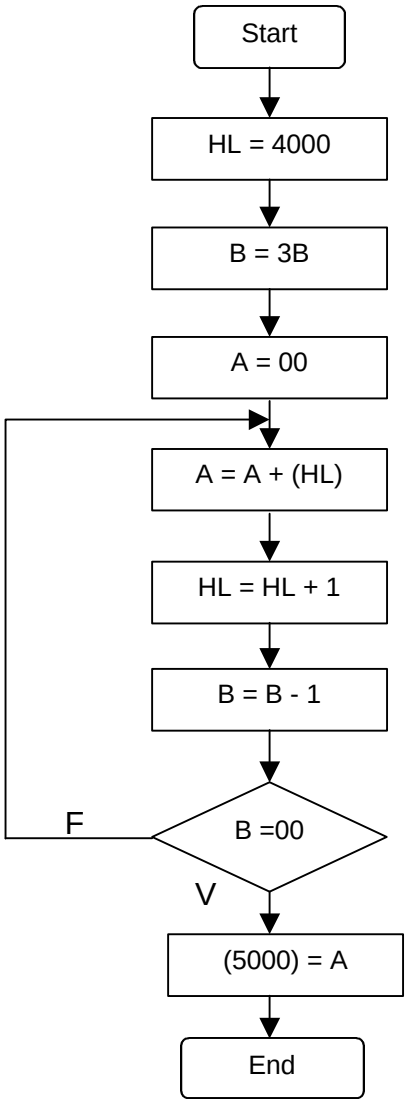
Esempio N° 5

Dati 3B valori residenti in memoria a partire dall'indirizzo 4000, eseguire la somma degli stessi senza tenere in conto dell'eventuale riporto, salvare in memoria il risultato in 5000.

Per eseguire l'operazione abbiamo bisogno dei seguenti registri (variabili):

- Accumulatore A per l'aggiornamento della somma
- HL per il puntamento degli indirizzi di memoria
- Un registro a 8 bit (ad esempio B) per contare le operazioni di somma, questo registro é caricato al valore max 3B e viene decrementato in quanto, il controllo del conteggio é eseguito con il flag di zero (zy)
  E' da notare che l'istruzione DEC B deve essere eseguita immediatamente prima del controllo per avere zy con il reale valore.

Flow-Chart del problema:



Memoria	Label	Codice	Operando	Commento
0100		LD	HL, 4000	Inizializzazione di HL
0103		LD	B, 3B	Inizializzazione di B
0105		LD	A, 00	Inizializzazione di A
0107	Loop:	ADD	A, (HL)	Somma di A con il byte indirizzato da HL
0108		INC	HL	
0109		DEC	B	
010A		JP	NZ, Loop	Controllo del conteggio
010D		LD	(5000), A	Salvataggio in memoria del risultato
0110		RST	38	

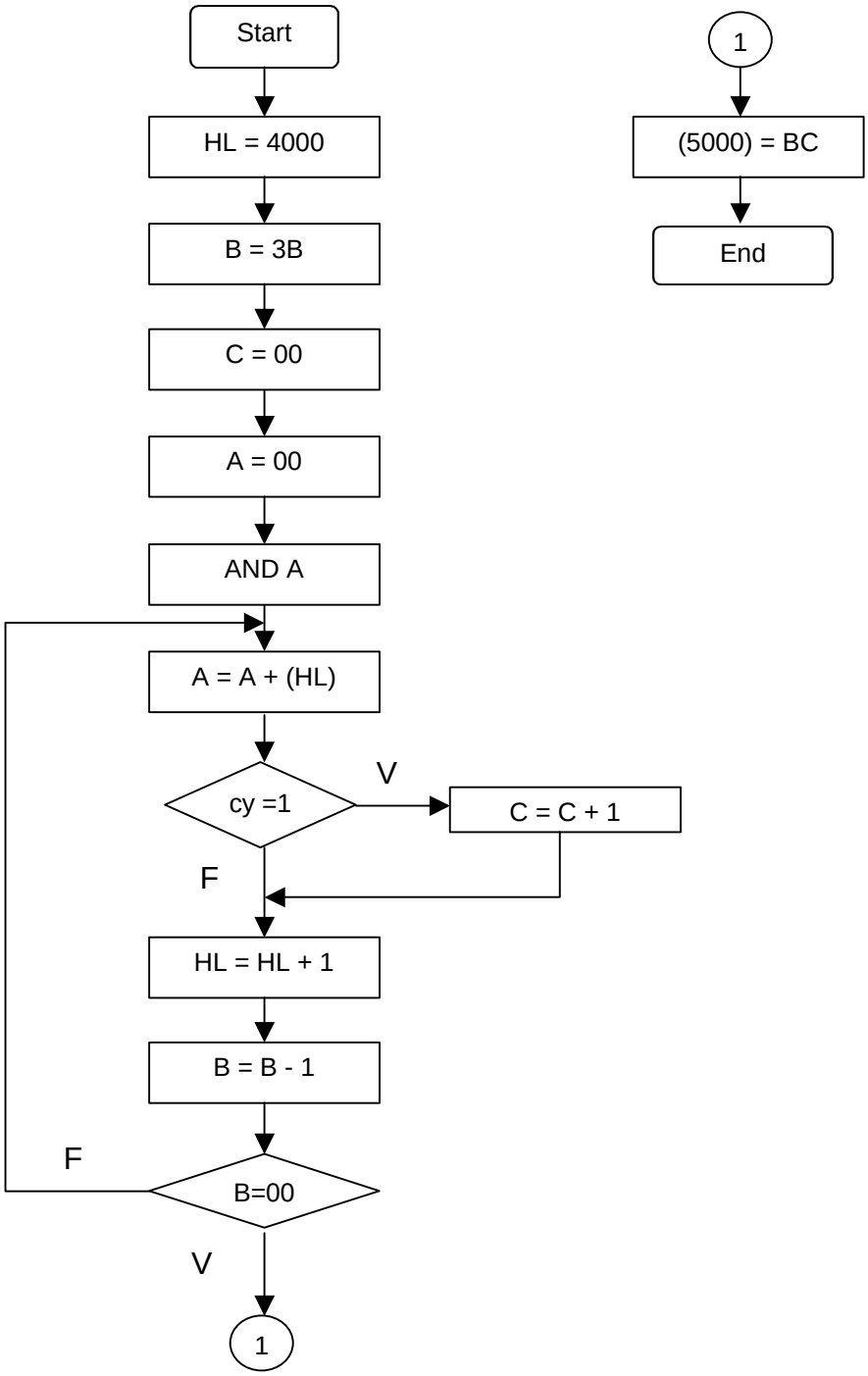
3-6 Somma con carry di una lista di numeri ad un byte

Esempio N° 6

Dati 3B valori residenti in memoria a partire dall'indirizzo 4000, eseguire la somma con riporto degli stessi, salvare il risultato in memoria a partire da 5000.

Fermo restando le ipotesi dell'esempio precedente, in questo caso abbiamo bisogno in aggiunta di un ulteriore registro (ad esempio C) per conteggiare in esso i riporti della somma, il risultato finale sarà presente nel registro BC.

Flow-Chart del problema:



Memoria	Label	Codice	Operando	Commento
0100		LD	HL, 4000	Inizializzazione di HL
0103		LD	B, 3B	Inizializzazione di B
0105		LD	C, 00	Inizializzazione di C
0107		LD	A, 00	Inizializzazione di A
0109		AND	A	Azzeramento del carry iniziale
010A	Loop2:	ADC	A, (HL)	Somma di A con il byte indirizzato da HL
010B		JP	C, Loop1	Salta se cy = 1
010E		INC	C	Aggiorna il riporto
010F	Loop1:	INC	HL	
0110		DEC	B	
0111		JP	NZ, Loop2	Controllo del conteggio
0114		LD	(5000), BC	Salvataggio risultato in memoria
0118		RST	38	

3-7 Moltiplicazione di 2 numeri a 8 bit

Lo Z80 come si é già affermato, non ha implementata l'operazione di moltiplicazione che, bisogna quindi eseguirla indirettamente.

Si possono avere due casi:

a) Il risultato del prodotto **non eccede** la capacità dell'accumulatore: Esempio N° 7

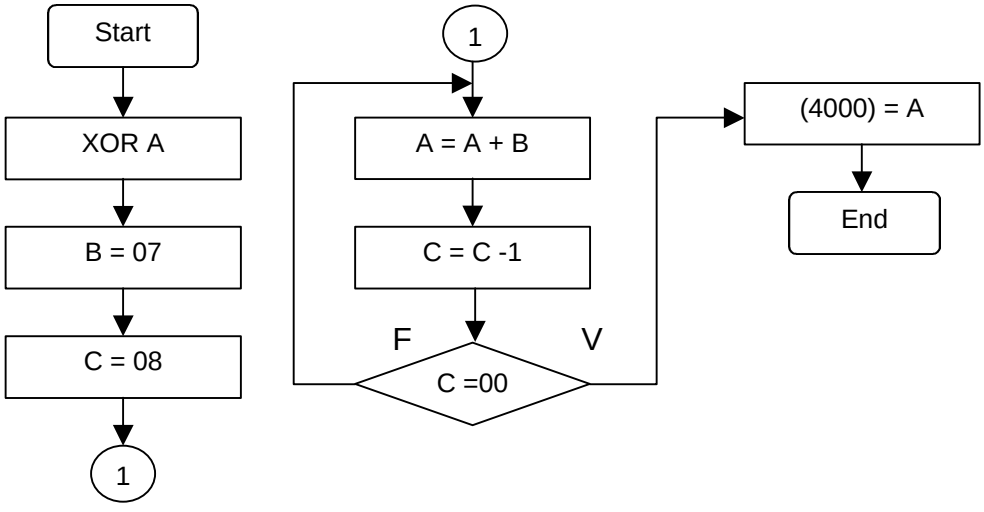
b) Il risultato del prodotto **eccede** la capacità dell'accumulatore: Esempio N° 8

Esempio N° 7

La moltiplicazione viene eseguita con il metodo della somma successiva

Si vuole ad esempio il risultato della moltiplicazione 7 x8, basterà sviluppare la somma:

7 x 8 = 7+7+7+7+7+7+7+7

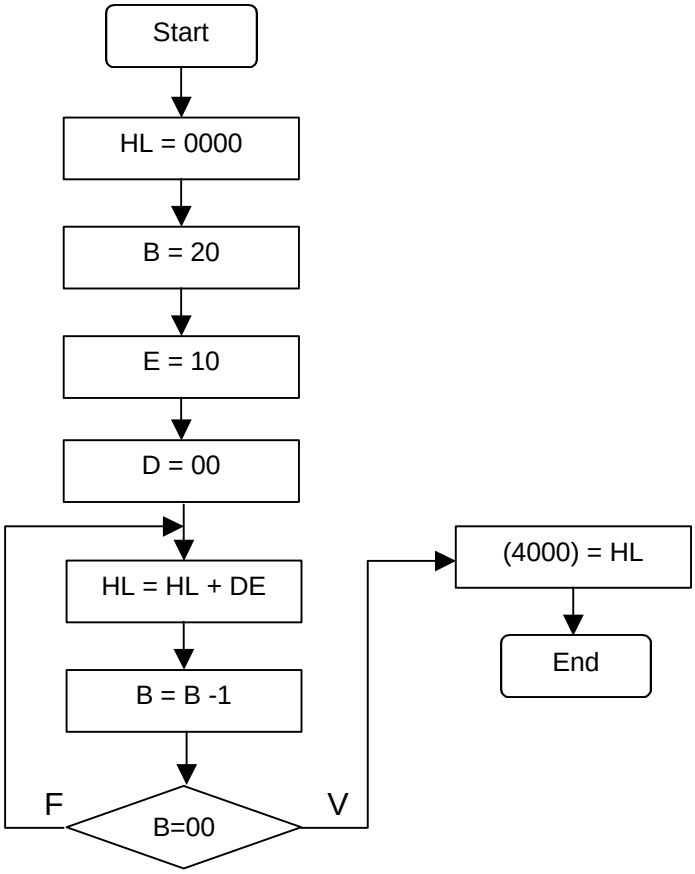


Da notare che l'azzeramento dell'accumulatore lo si esegue più facilmente facendo un OR esclusivo dello stesso.

Memoria	Label	Codice	Operando	Commento
0100		XR	A	Azzeramento (inizializzazione) di A
0101		LD	B, 07	Inizializzazione di B
0103		LD	C, 08	Inizializzazione di A
0106	Loop:	ADD	A, B	Somma di A con il valore fisso di B
0107		DEC	C	
0108		JP	NZ, Loop	Controllo del conteggio
010B		LD	(4000), A	Salvataggio in memoria del risultato
010E		RST	38	

Esempio N° 8

Si procede in maniera analoga al caso precedente, utilizzando però dei registri a 16 bit.



Nell'esempio si é effettuata la moltiplicazione in esadecimale di 10 x 20

Nell'assembly si é utilizzato con il registro B l'istruzione DJNZ

Memoria	Label	Codice	Operando	Commento
0100		LD	HL, 0000	Inizializzazione del risultato della moltipl.
0103		LD	B, 20	Inizializzazione di B
0105		LD	D, 00	Inizializzazione di D
0107		LD	E, 10	Inizializzazione di E
0109	Loop:	ADD	HL, DE	Somma di HL con il valore fisso di DE
010A		DJNZ	Loop	Controllo del conteggio di B
010C		LD	(4000), HL	Salvataggio in memoria del risultato
0110		RST	38	

3-8 Sottrazione di 2 numeri a 16 bit

La sottrazione di 2 numeri a 16 bit deve essere eseguita tenendo presente dell'eventuale prestito (borrow) che il byte più significativo (MSB) deve effettuare al byte meno significativo (LSB) del minuendo.

Esempio N° 9

Eseguire la sottrazione tra i numeri (hex) 0200 - 0001

Posto il minuendo (0200) in BC ed il sottraendo (0001) in DE, abbiamo in questo caso il riporto tra i byte del minuendo:

B								C									D								E							
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

1 ⟶ prestito

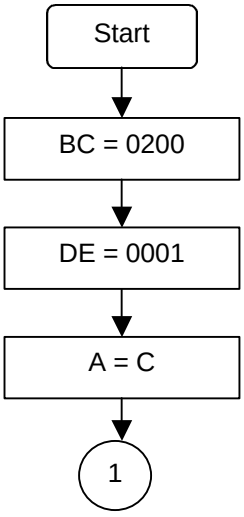
Posto il risultato in BC:

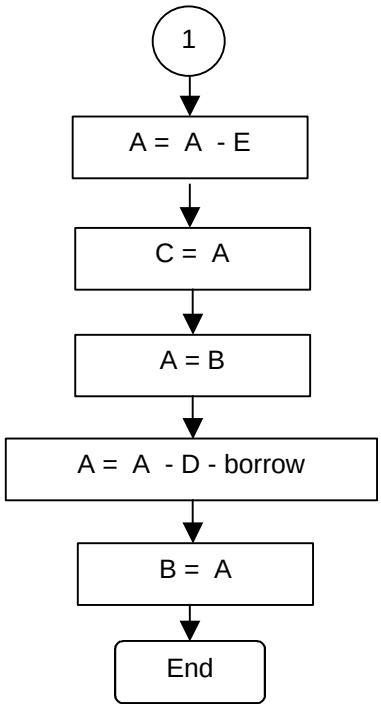
B								C							
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

L'algoritmo risolutivo si sviluppa nei passi:

1) sottrazione semplice tra C ed E

2) sottrazione con riporto tra tra B e D



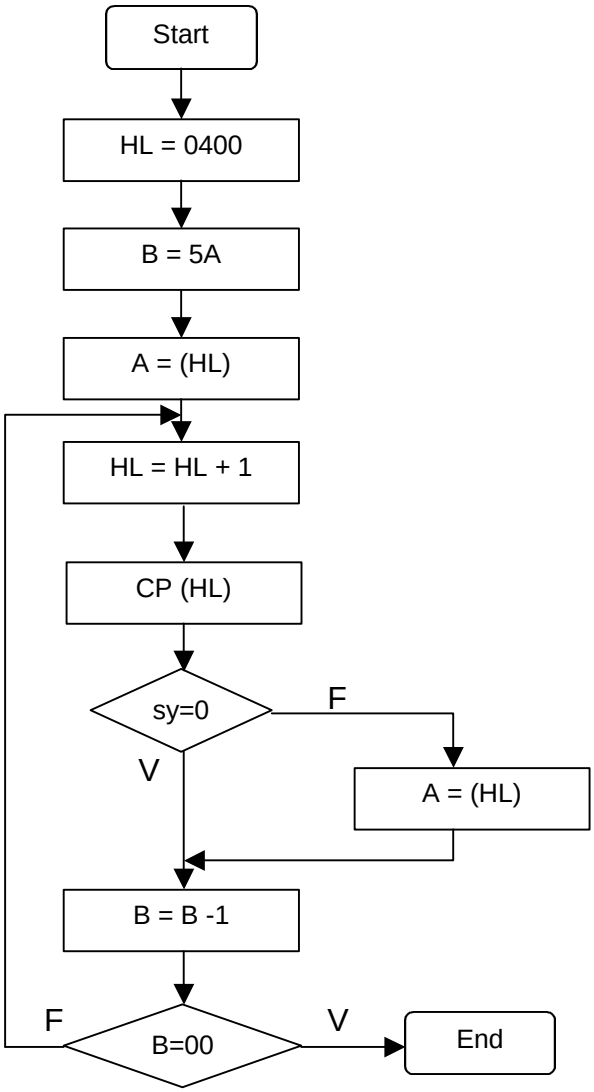


Memoria	Label	Codice	Operando	Commento
0100		LD	BC, 0200	minuendo
0103		LD	DE, 0001	sottraendo
0106		LD	A, C	
0107		SUB	E	A = A - E
0108		LD	C, A	
0109		LD	A, B	
010A		SBC	A, D	A = A - D - borrow
010B		LD	B, A	
0110		RST	38	

3-9 Massimo valore di una lista di numeri

Esempio N° 10

Trovare il max dei valori di una lista di 5B dati, indirizzati a partire dalla locazione 0400



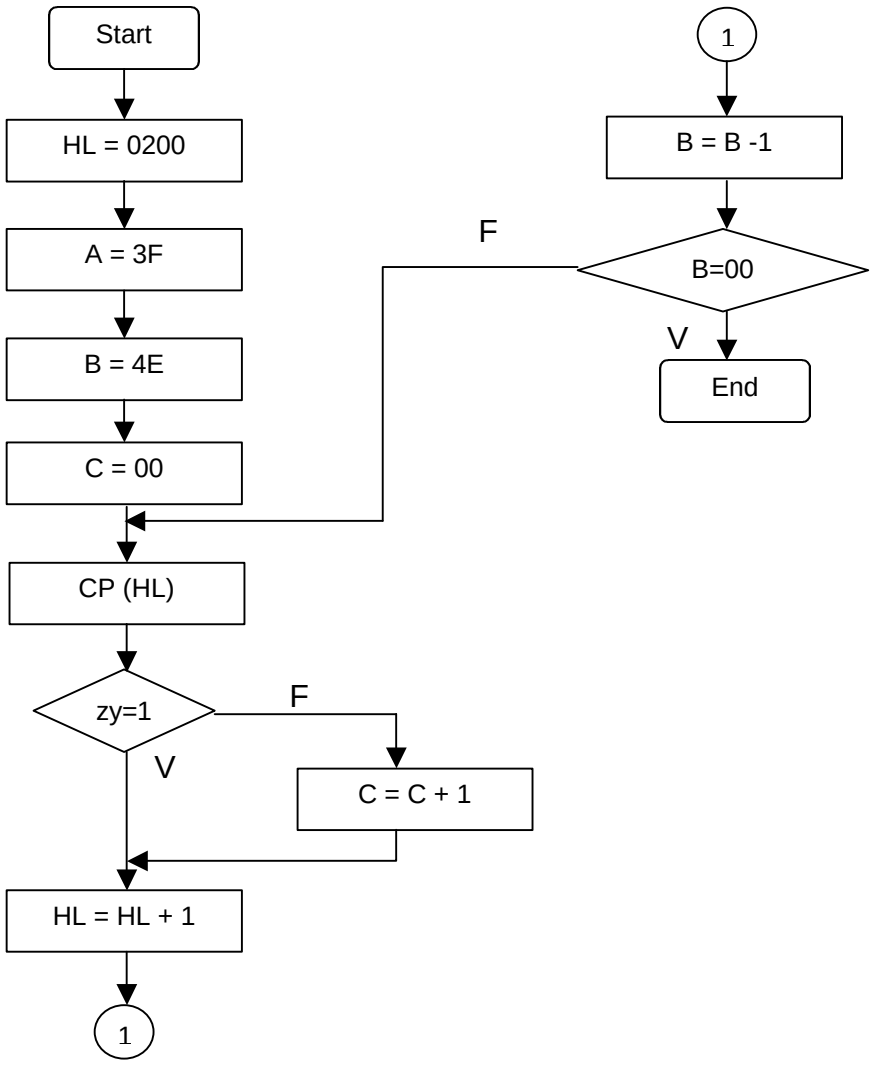
Memoria	Label	Codice	Operando	Commento
0100		LD	HL, 0400	Inizializzazione di HL
0103		LD	B, 5A	Inizializzazione di B
0105		LD	A, (HL)	Carica il primo valore della lista in A
0106	Loop2:	INC	HL	
0107		CP	(HL)	Sottrazione virtuale A =(HL)
0108		JP	P, Loop1	Se la sottrazione non é negativa salta
010B		LD	A, (HL)	Altrimenti definisci il nuovo max
010C	Loop1:	DEC	B	
010D		JP	NZ, Loop2	
0110		RST	38	Fine programma

Al termine in A ci sarà il max valore

3-10 Conta di un valore in una lista di numeri

Esempio N° 11

Trovare quante volte esiste il valore 3F nella lista indirizzata tra 0200 e 024D



Memoria	Label	Codice	Operando	Commento
0100		LD	HL, 0200	Inizializzazione di HL
0103		LD	A, 3F	Carica in A del dato da ricercare
0105		LD	B, 4E	Numeri totali della lista
0107		LD	C, 00	C conta quante volte esiste 3F
0109	Loop2:	CP	(HL)	Sottrazione virtuale A =(HL)
010A		JP	Z, Loop1	Se il risultato é 0 salta a Loop1
010D		INC	C	Altrimenti incrementa C
010E	Loop1:	INC	HL	
010F		DEC	B	
0110		JP	NZ, Loop2	
0113		RST	38	Fine programma

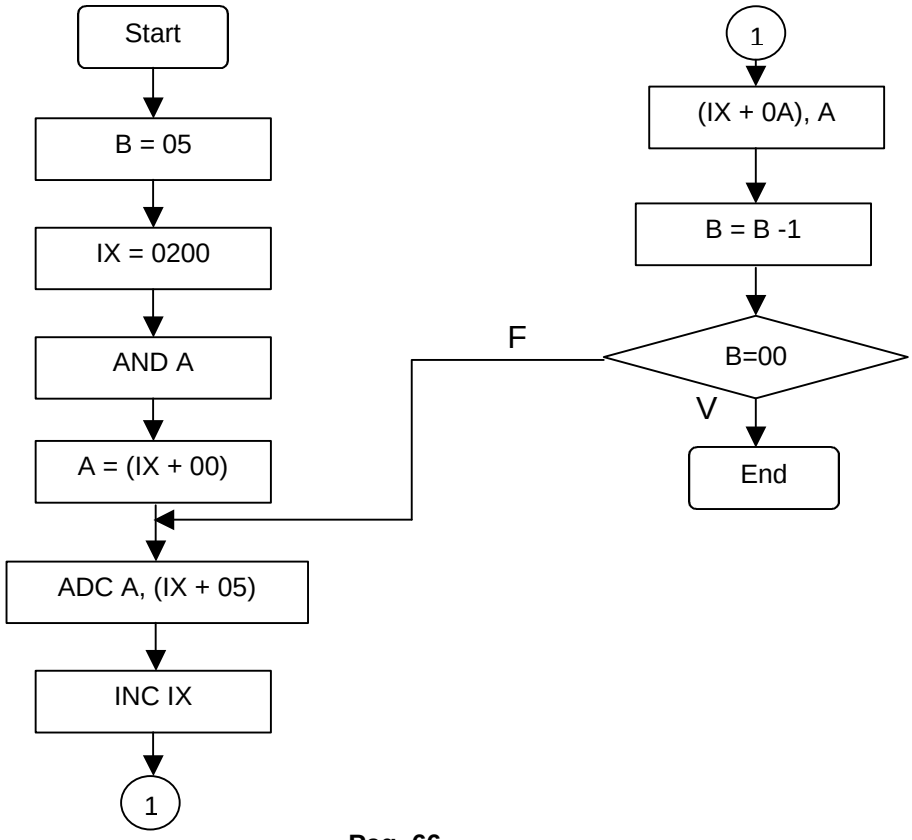
3-11 Somma di 2 numeri a 5 byte

Un uso dei registri indice IX e IY, é quello di gestire le operazioni aritmetiche di numeri molto grandi quindi rappresentati ciascuno da molti bytes.

Esempio N° 12

Eseguire la somma di 2 numeri a 5 bytes residenti in memoria e, salvare il risultato stesso secondo il seguente schema d'indirizzamento:

Indirizzi Memoria →														
0200	0201	0202	0203	0204	0205	0206	0207	0208	0209	020A	020B	020C	020D	020E
=	=	=	=	=	+	+	+	+	+	§	§	§	§	§
1° addendo					2° addendo					Risultato				

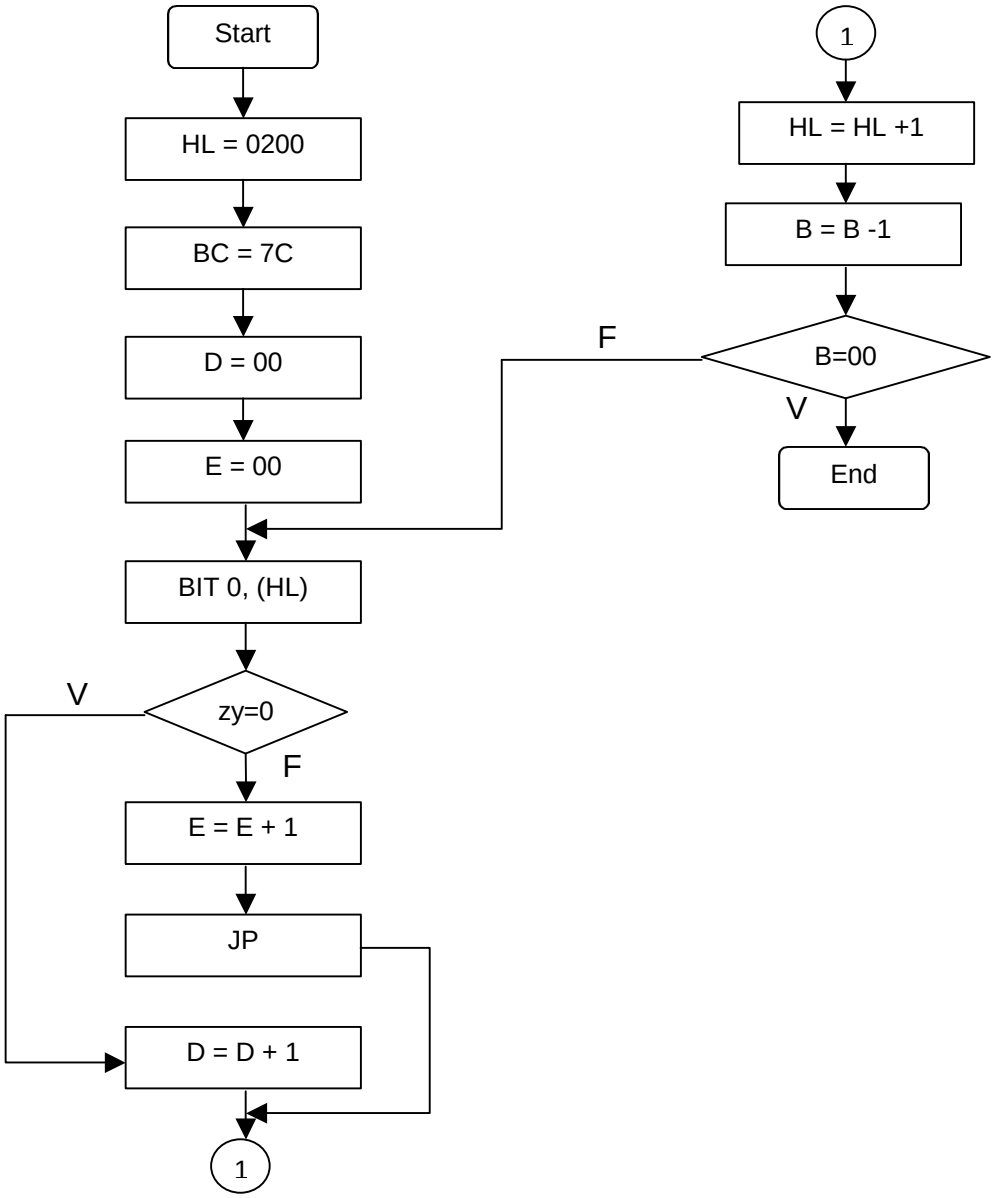


Memoria	Label	Codice	Operando	Commento
0100		LD	B, 05	
0102		LD	IX, 0200	
0106		AND	A	Azzera il carry iniziale
0107	Loop:	LD	A, (IX + 00)	Carica il 1° addendo della somma
010A		ADC	A, (IX + 05)	Somma di bytes corrispondenti
010D		LD	(IX + 0A)	Salvataggio del risultato
0110		INC	IX	Altrimenti incrementa C
0112		DJNZ	Loop	
0114		RST	38	Fine programma

3-12 Conta dei pari e dei dispari esistenti in una lista

Esempio N° 13

Determinare il numero dei pari e dei dispari esistenti nella lista tra 0200 e 027B



Memoria	Label	Codice	Operando	Commento
0100		LD	HL,0200	
0103		LD	B, 7C	Numeri della lista
0105		LD	D, 00	D contiene i dispari
0107		LD	E, 00	E contiene i dpari
0109	Loop3	BIT	0, (HL)	Test del bit 0 dell'elemnto della lista
010A		JP	NZ, Loop1	Se zy = 0 vai alla conta dei dispari
010D		INC	E	Incrementa i pari
010E		JP	Loop2	
0111	Loop1	INC	D	Incrementa i dispari
0112	Loop2	INC	HL	
0113		DEC	B	
0114		JP	NZ, Loop3	
0117		RST	38	Fine programma

Nell'esempio riportato si può notare che il set di istruzioni dello Z80 non gestisce in maniera diretta l'oerazione tipo: **IF** cond **Then** istruz 1 **Else** Istruz 2

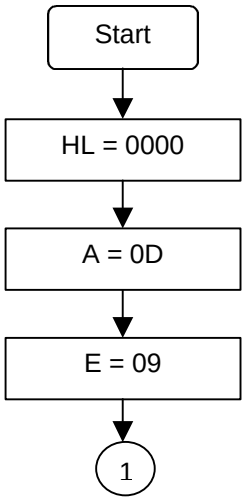
3-13 Algoritmo generale della moltiplicazione tra 2 numeri a 8 bit

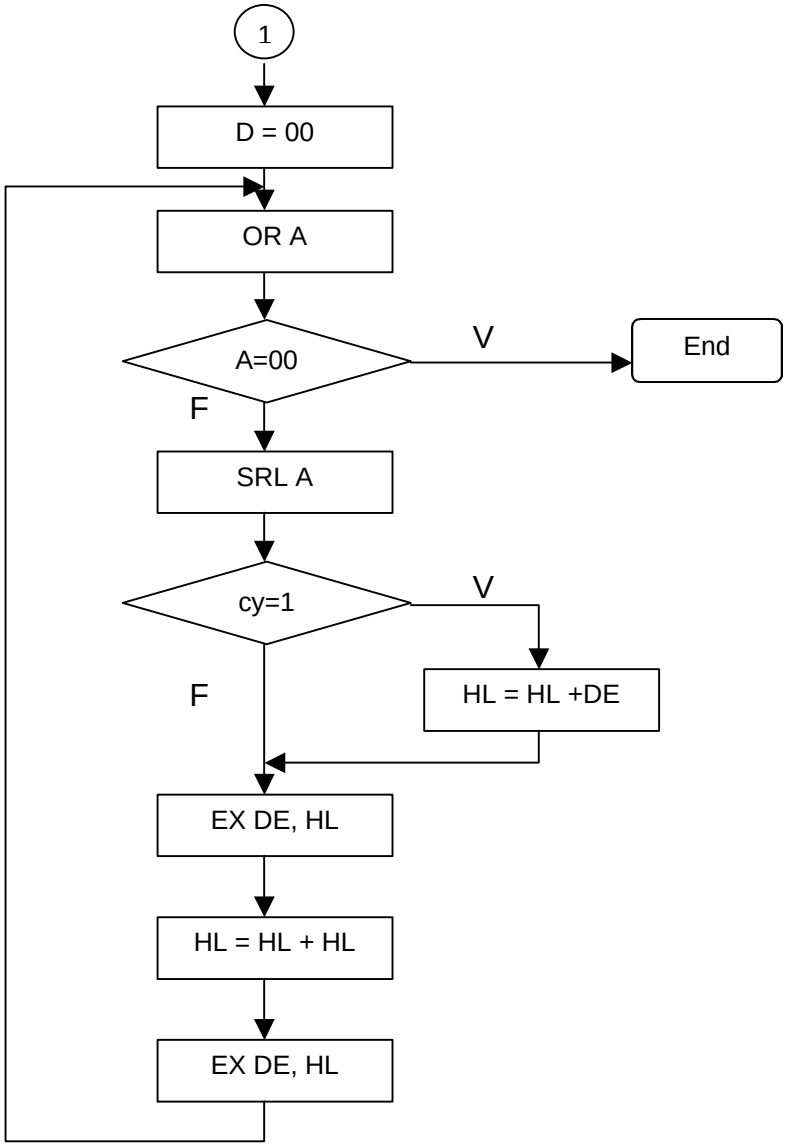
Il metodo si basa direttamente sull'algoritmo della moltiplicazione binaria tra 2 numeri

Esempio N° 14

Eseguire la moltiplicazione tra i byte 09 e 0D

```
      1 0 0 1      x     moltiplicando
      1 1 0 1      =     moltiplicatore
      ---------
      1 0 0 1
    0 0 0 0
  1 0 0 1
1 0 0 1
--------------------
0 1 1 1 0 1 0 1            risultato
```





Le istruzioni EX + ADD HL, HL + EX permettono di eseguire lo shift a sinistra del registro a 16 bit DE

Memoria	Label	Codice	Operando	Commento
0100		LD	HL, 0000	HL contiene il risultato della moltiplicazione
0103		LD	A, 0D	A = moltiplicatore
0105		LD	E, 09	
0107		LD	D, 00	D+E = moltiplicando
0109	Loop2	OR	A	Controllo se il moltiplicatore é nullo
010A		JP	Z, Fine	Se A = 0 termina
010D		SRL	A	Shift a destra di A
010E		JP	NC, Loop	Se cy = 1
0111		ADD	HL, DE	Aggiungere ad HL il moltiplicando
0112	Loop1	EX	DE, HL	Inizio dello shift a sinistra del moltiplicando
0113		ADD	HL, HL	
0114		EX	DE, HL	Fine dello shift
0115		JP	Loop2	
0118	Fine	RST	38	Fine programma

3-14 Divisione tra 2 byte con il metodo delle sottrazioni successive

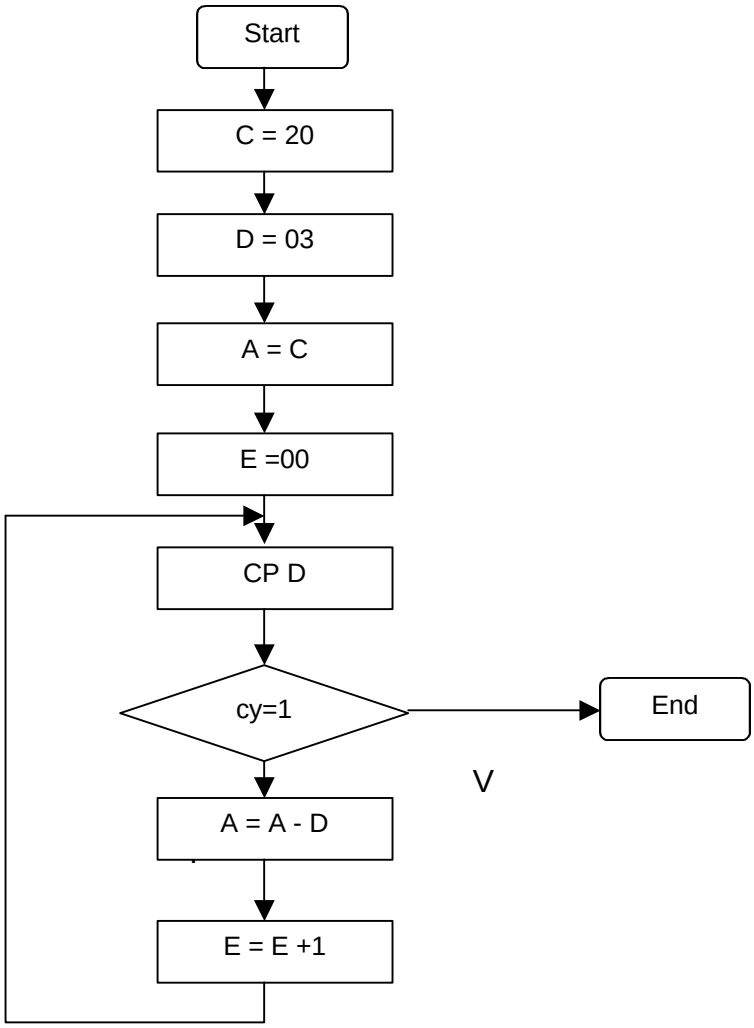
Il metodo si basa nel sottrarre tante volte al dividendo il divisore, fino a quando quest'ultimo non é maggiore del dividendo stesso.

Ad esempio: Dividendo = 21 Divisore = 5

18 : 5 = 18 - 5 -5 -5 - 5 Quoziente = 4 Resto = 3

Esempio N° 15

Divisione tra i byte 20 e 03



Memoria	Label	Codice	Operando	Commento
0100		LD	C, 20	C = dividendo
0102		LD	D, 03	D = divisore
0104		LD	A, C	
0105		LD	E, 00	E = quoziente
0107	Loop	CP	D	Operazione virtuale A - D
0108		JP	C, Fine	Se cy = 1 allora A - D < 0
010B		SUB	D	Al dividendo si sottrae il divisore
010C		INC	E	Incrementa il quaziente
010D		JP	Loop	
0110	Fine	RST	38	Fine programma

3-15 Algoritmo generale della divisione tra 2 numeri a 8 bit

Si basa sul seguente algoritmo:

Sia $(150)_{10} = (10010110)_2$ Dividendo
$(20)_{10} = (00010100)_2$ Divisore

si dispongano: dividendo, divisore, quoziente come di seguito:

HL = Dividendo															
0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0

C = Divisore							
0	0	0	1	0	1	0	0

E =Quoziente							

a) Passo 1 si effettua lo shift a sinistra del dividendo ed il confronto
H e C, H < C si pone 0 nel I° bit più significativo del quoziente

HL = Dividendo															
0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0

C = Divisore							
0	0	0	1	0	1	0	0

E =Quoziente							
0							

b) Passo 2 si effettua lo shift a sinistra del dividendo ed il confronto
H e C, H < C si pone 0 nel II° bit più significativo del quoziente

HL = Dividendo															
0	0	0	0	0	0	1	0	0	1	0	1	1	0	0	0

C = Divisore							
0	0	0	1	0	1	0	0

E =Quoziente							
0	0						

c) Passo 3 si effettua lo shift a sinistra del dividendo ed il confronto
H e C, H < C si pone 0 nel III° bit più significativo del quoziente

HL = Dividendo															
0	0	0	0	0	1	0	0	1	0	1	1	0	0	0	0

C = Divisore							
0	0	0	1	0	1	0	0

E =Quoziente							
0	0	0					

d) Passo 4 si effettua lo shift a sinistra del dividendo ed il confronto
H e C, H < C si pone 0 nel IV° bit più significativo del quoziente

HL = Dividendo															
0	0	0	0	1	0	0	1	0	1	1	0	0	0	0	0

C = Divisore							
0	0	0	1	0	1	0	0

E =Quoziente							
0	0	0	0				

e) Passo 5 si effettua lo shift a sinistra del dividendo ed il confronto
H e C, H < C si pone 0 nel V° bit più significativo del quoziente

HL = Dividendo															
0	0	0	1	0	0	1	0	1	1	0	0	0	0	0	0

C = Divisore							
0	0	0	1	0	1	0	0

E =Quoziente							
0	0	0	0	0			

f) Passo 6 si effettua lo shift a sinistra del dividendo ed il confronto
H e C, H > C si pone 1 nel VI° bit più significativo del quoziente

HL = Dividendo															
0	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0

C = Divisore							
0	0	0	1	0	1	0	0

E =Quoziente							
0	0	0	0	0	1		

Si fa la differenza tra H e C, il risultato é il nuovo valore di H, si incrementa di 1 il registro L

HL = Dividendo															
0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	1

g) Passo 7 si effettua lo shift a sinistra del dividendo ed il confronto
H e C, H > C si pone 1 nel VII° bit più significativo del quoziente

HL = Dividendo															
0	0	1	0	0	0	1	1	0	0	0	0	0	0	1	0

C = Divisore							
0	0	0	1	0	1	0	0

E =Quoziente							
0	0	0	0	0	1	1	

Si fa la differenza tra H e C, il risultato é il nuovo valore di H, si incrementa di 1 il registro L

HL = Dividendo															
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1

h) Passo 8 si effettua lo shift a sinistra del dividendo ed il confronto H e C, H > C si pone 1 nel VIII° bit del quoziente

HL = Dividendo															
0	0	0	1	1	1	1	0	0	0	0	0	0	1	1	0

C = Divisore							
0	0	0	1	0	1	0	0

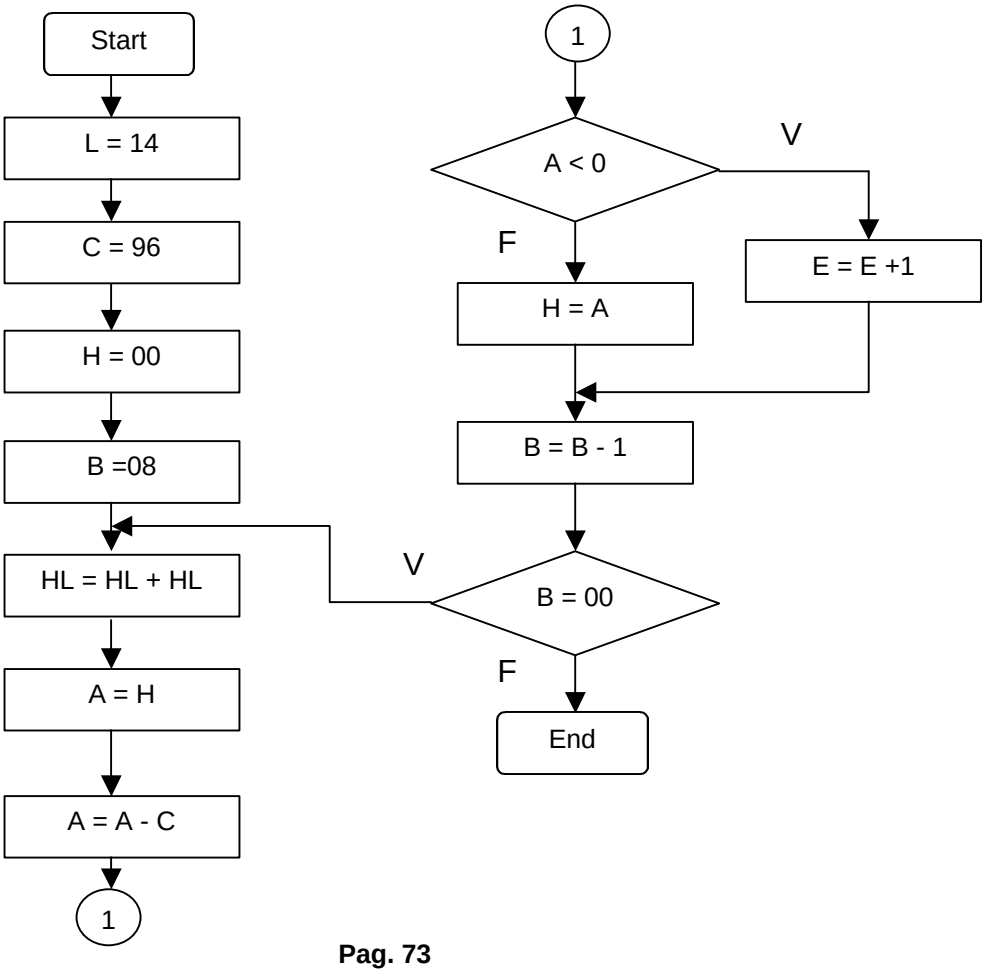
E =Quoziente							
0	0	0	0	0	1	1	1

Si fa la differenza tra H e C, il risultato é il nuovo valore di H, si incrementa di 1 il registro L

HL = Dividendo															
0	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1

Si é terminata la procedura, il valore di H é il resto della divisione, il valore di L coincide con il quoziente E

Esempio N° 16



Memoria	Label	Codice	Operando	Commento
0100		LD	L, 14	L = dividendo
0102		LD	C, 96	C = divisore
0104		LD	H, 00	
0106		LD	B, 08	Conteggio delle operazioni
0108	Loop2	ADD	HL, HL	Shift a sinistra di HL
0109		LD	A, H	
010A		SUB	C	Al dividendo si sottrae il divisore
010B		JP	M, Loop1	Se A = H - L < 0 salta
010E		LD	H, A	Aggiorna il dividendo
010F	Loop1	INC	E	E al termine sarà il quoziente
0110		Dec	B	
0111		JP	NZ, Loop2	
0110	Fine	RST	38	Fine programma

3-16 Generazione di cicli di ritardo

Un aspetto importante della programmazione per il controllo di un processo é quello di realizzare far funzionare il µP come Timer.

Operativamente ciò si realizza facendo eseguire al µP, delle istruzioni cicliche di decremento dei valori dei suoi registri.

In questo caso é importante conoscere del sistema a µP:

a) la frequenza di funzionamento del clock

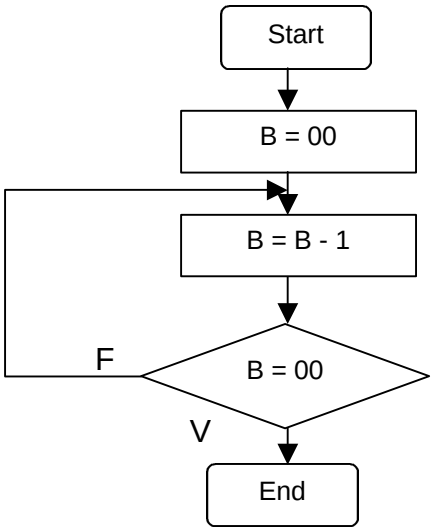
b) il numero di **cicli T** necessari all'esecuzione di ogni istruzione eseguita

Negli esempi riportati si farà riferimento ad una **f_{ck} = 2,5 MHz** che, posta ad una durata del ciclo T pari a:

$$T = 1/f_{ck} = 0{,}40\ \mu sec$$

3-17 Cicli di ritardo ad un unico registro

Permettono di contare tempi compresi tra **0 e Tmax = 1,441 msec**



7 T	durata LD B, 00
4 x 256 T	durata 256 volte DEC B
10 x 256T	durata 256 volte JP NZ, Loop
11 T	durata RST 38

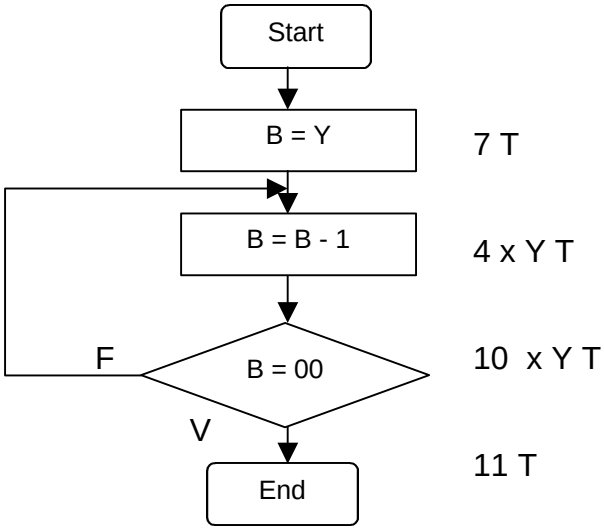
Tmax é calcolato avendo riportato la durata di ciascuna istruzione e quante volte l'istruzione specifica é stata eseguita

Tmax = (14x256 + 18) x T = 1,441 msec

Esempio N° 17

Determinare un ciclo di ritardo di 1,00 msec

In questo caso il registro B deve essere inizializzato ad un valore da calcolare Y



Il valore incognito Y si ricava tramite la proporzione:

$1{,}441 : 256 = 1{,}000 : Y \longrightarrow Y = 177{,}644 \longrightarrow Y = (177)_{10} = (B1)_{16}$

Con il valore assunto da B, in definitiva si realizzerà un ciclo pari a:

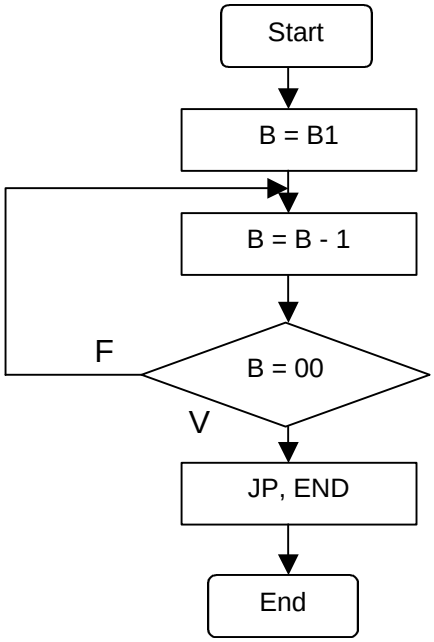
177/(177,644) x 1,00 = 0,996 msec

con un errore pari a : 1,000 - 0,996 = 0,004 = 4 µsec

Per compensare l'errore di conteggio, in genere si usa una successione di istruzioni **NOP** (No Operation) ciascuna delle quali, ha una durata di:

$T_{NOP} = 4 \times T = 1{,}6\ \mu sec$

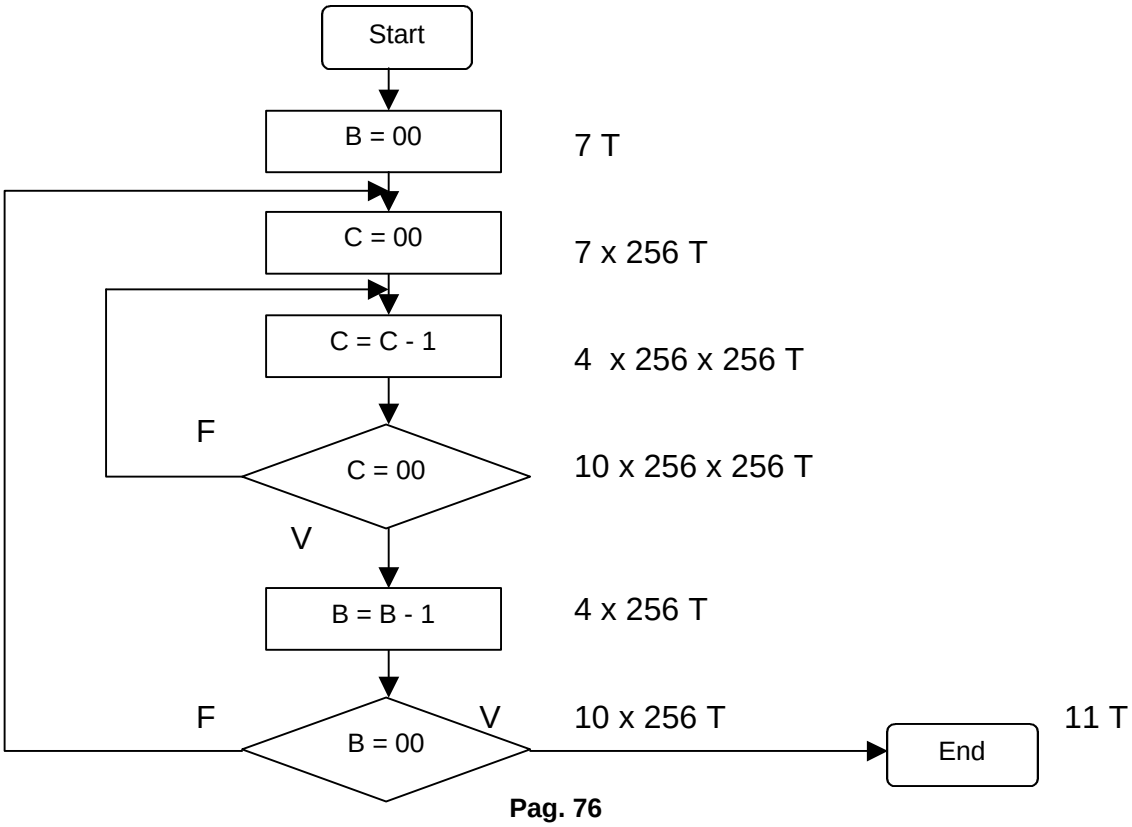
In questo specifico caso, si può utilizzare per avere una migliore approssimazione rispetto all'uso di 2 NOP, l'istruzione JP, Loop che ha proprio la durata di 10 T = 4 µsec



Memoria	Label	Codice	Operando	Commento
0100		LD	B, B1	
0102	Loop1	DEC	B	
0103		JP	NZ, Loop1	
0106		JP	Loop2	
0109	Loop2	RST	38	

3-17 Cicli di ritardo a 2 registri

Permettono di contare tempi compresi tra **0 e Tmax = 369,159 msec**

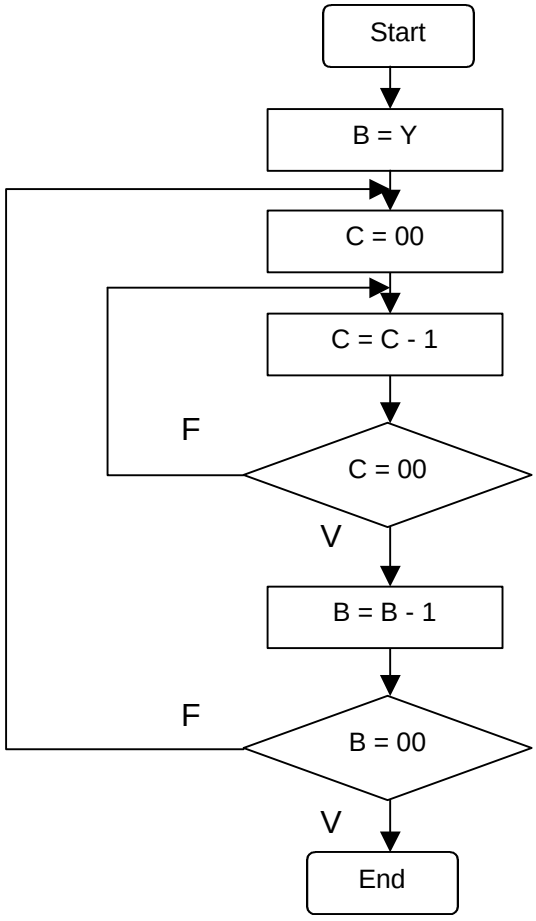


Tmax = (18 + 21 x 256 + 14 x 256 x 256) x T = 369,159 msec

Esempio N° 18

Determinare un ciclo di ritardo di 100 msec

In questo caso il registro B deve essere inizializzato ad un valore da calcolare Y



Il valore incognito Y si ricava tramite la proporzione:

$$369,159 : 256 = 100 : Y \longrightarrow Y = 69,347 \longrightarrow Y = (69)_{10} = (45)_{16}$$

Con il valore assunto da B, in definitiva si realizzerà un ciclo pari a:

69/(69,347) x 100 = 99,499 msec

con un errore pari a : 100 - 99,499 = 0,501 msec

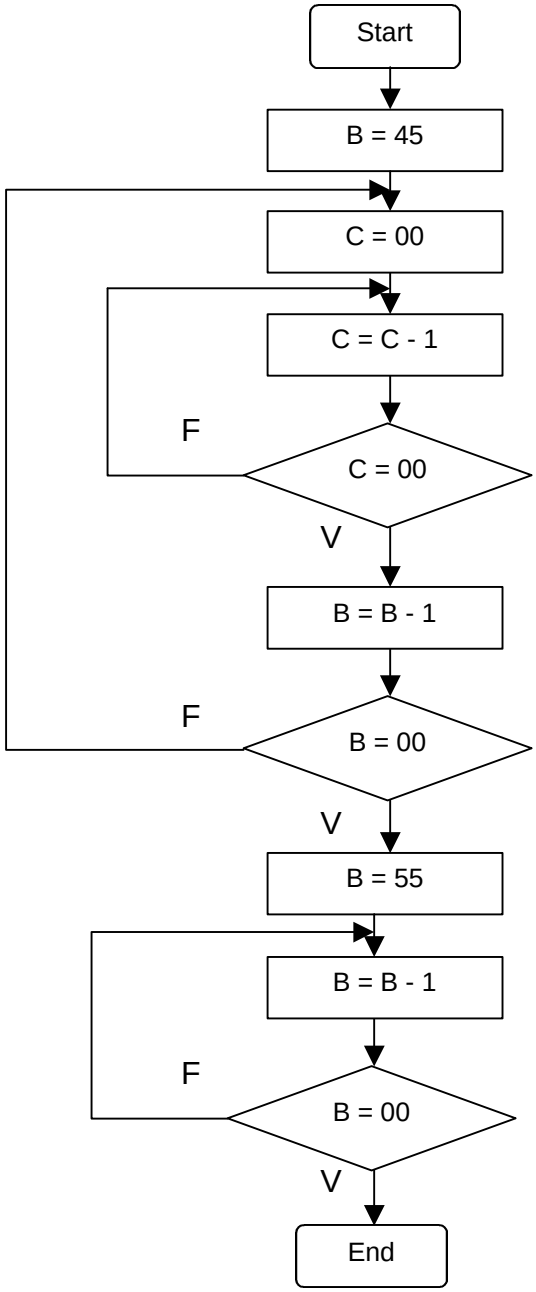
L'errore commesso lo si compensa introducendo alla fine dei cicli concatenati, un loop di ritardo ad unico ciclo inizializzato al valore K

$$1,441 : 256 = 0,501 : K \longrightarrow K = 89,005 \longrightarrow K = (89)_{10} = (55)_{16}$$

che sviluppa un tempo:

89/(89,005) x 0,501 = 0,50097 msec

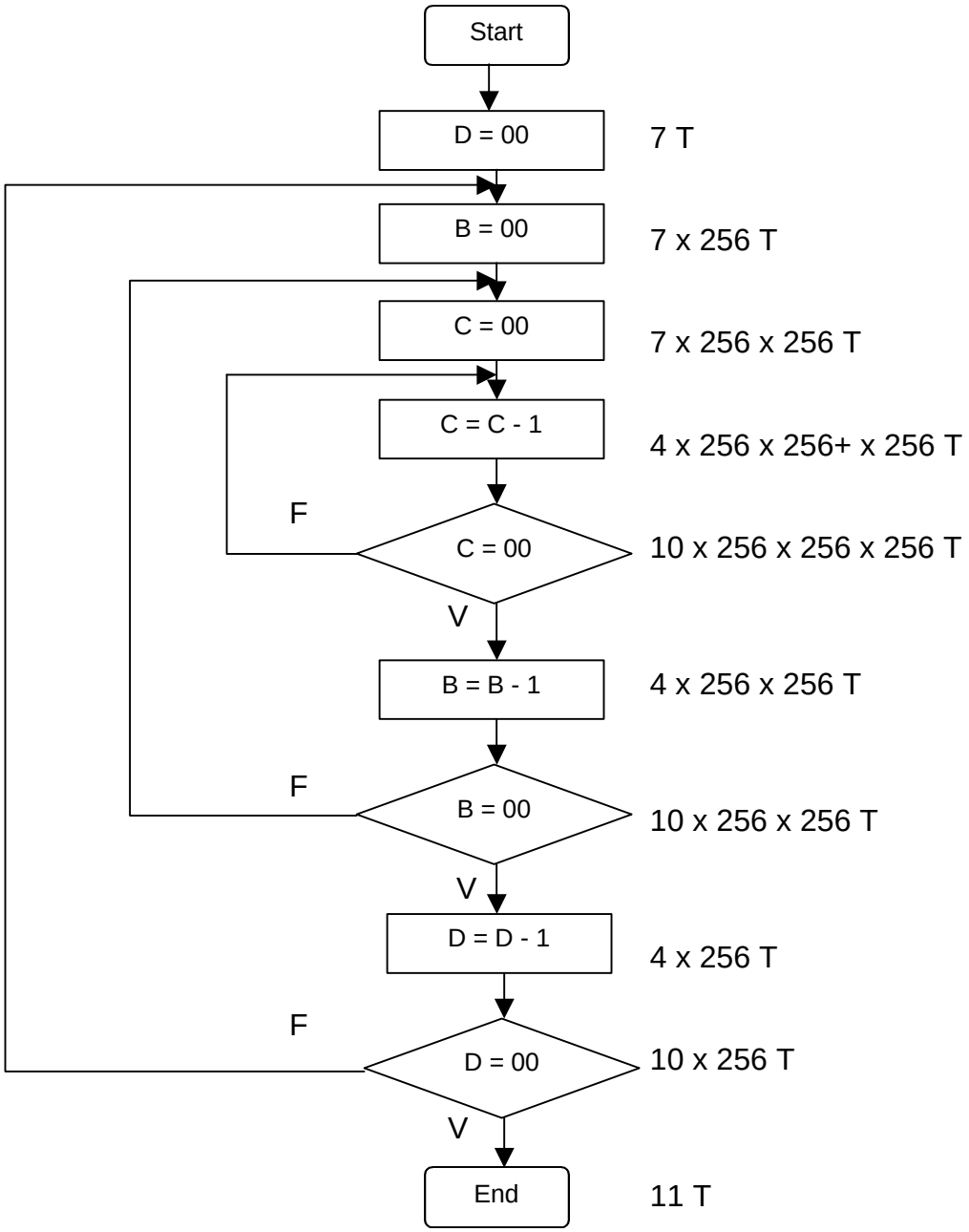
Pertanto l'errore residuo pari a: 0,50100 - 0,50097 = 0,3 µ sec non può essere ulteriormente compensato (la minima istruzione sviluppa 1,6 µsec)



Memoria	Label	Codice	Operando	Commento
0100		**LD**	**B, 45**	**Inizio ciclo di ritardo principale**
0102	**Loop2**	**LD**	**C, 00**	
0104	**Loop1**	**DEC**	**C**	
0105		**JP**	**NZ, Loop1**	
0108		**DEC**	**B**	
0109		**JP**	**NZ, Loop2**	**Fine ciclo di ritardo principale**
011C		LD	B, 55	**Inizio ciclo di ritardo in compensazione**
0102	Loop3	DEC	B	
0103		JP	NZ, Loop3	**Fine ciclo di ritardo in compensazione**
0109		RST	38	

3-19 Cicli di ritardo a 3 registri

Permettono di contare tempi compresi tra **0 e Tmax = 94,505 sec**



$$T_{max} = (18 + 21 \times 256 + 21 \times 256 \times 256 + 14 \times 256 \times 256 \times 256) \times T = 94{,}505 \text{ sec}$$

Esempio N° 19

Determinare un ciclo di ritardo di 10 sec

In questo caso il registro D deve essere inizializzato ad un valore da calcolare Y

Il valore incognito Y si ricava tramite la proporzione:

$$94{,}505 : 256 = 10 : Y \longrightarrow Y = 27{,}088 \longrightarrow Y = (27)_{10} = (1B)_{16}$$

Con il valore assunto da D, in definitiva si realizzerà un ciclo pari a:

$$27/(27{,}088) \times 10 = 9{,}967 \text{ sec}$$

con un errore pari a : 10 - 9,967= 33 msec

L'errore commesso lo si compensa introducendo alla fine dei cicli concatenati, un loop di ritardo ad 2 cicli inizializzato al valore K

$$369{,}59 : 256 = 33 : K \longrightarrow K = 22{,}858 \longrightarrow K = (22)_{10} = (16)_{16}$$

Il tempo compensato da quest'ultimo loop di ritardo é pari a:

$$22/(22{,}858) \times 33 = 31{,}761 \text{ msec}$$

con un errore pari a : 33,000 - 31,761= 1,239 msec

Il tempo residuo viene compensato da un loop di ritardo ad un solo registro, caricato al valore N dato da:

$$1{,}441 : 256 = 1{,}239 : N \longrightarrow N = 220{,}114 \longrightarrow N = (220)_{10} = (DC)_{16}$$

Il tempo compensato da quest'ultimo loop di ritardo é pari a:

$$220/(220{,}114) \times 1{,}239 = 1{,}238 \text{ msec}$$

con un errore pari a : 1,239 - 1,238 = 1 µsec