Appunti sullo Z80

MAPPA MEMORIA disponibile con il MPF 1PLUS

ROM	8K	0000-FFFF	(EPROM)
RAM	2 RAM 48K totali	F000-FFFF	(CHIP U4)

Z80 :
SET di 158 Istruzioni
$f_{MAX} = 2{,}5\ M_{HZ}$

MPF 1PLUS opera a 1,79 M_{HZ}

Area espansione memoria : EPROM 2000-3FFF

DISPLAY max 20 caratteri ciascuno componibile con 16 segmenti.

TASTIERA con 49 tasti Alfanumerici (A-Z, 0 --9) più quelli funzione

Alimentazione del sistema in C.C. con Alimentatore 220V/9V 600 mA assorbimento 450 mA.

COMANDI DEL MPF 1 PLUS:

M [INDIRIZZO] <INVIO> [visualizza sul DISPLAY il contenuto delle celle di memorie, a partire dall'indirizzo digitato (4 celle di memoria per volta, da qui ci si potrà spostare con i tasti ↑o↓ per visualizzare tutte le altre celle di memoria).

Ad es. : M F100 <INVIO> visualizzerà sul display il contenuto delle celle F100, F101, F102, F103

M [INDIRIZZO] **<:>** permette di modificare il contenuto delle celle di memoria

Ad es. M F100 **<:>** Digitando da tastiera dei valori (in formato esadecimale, ogni valore seguito da uno spazio) questi saranno acquisiti dal buffer della tastiera, il quale può contenere fino ad un massimo di venti valori. I valori digitati saranno trasferiti nelle celle di memorie corrispondenti quando si premerà il tasto di <INVIO>. Se si digitano due valori, questi occuperanno le celle di memoria F100 ed F101. Quando il buffer della tastiera non accetta più dei valori, si dovrà inviare in memoria quelli già digitati con il tasto <INVIO>. Per inserire altri valori si dovrà ripetere il comando, ovviamente a partire dalla cella di memoria seguente l'ultimo valore inserito.

G [INDIRIZZO] <INVIO> Manda in esecuzione il programma a partire dall'istruzione contenuta nella cella di memoria dell'indirizzo digitato.

S [INDIRIZZO] <INVIO> Permette di eseguire il programma una sola istruzione per volta, ad ogni pressione successiva del tasto S, verrà eseguita un'istruzione. Anche questo comando è utile per eseguire il debugging del programma.

ISTRUZIONE	EFFETTO	INDIRIZZAMENTO	DESCRIZIONE	CODICE OGGETTO
RST 38H		PAGINA ZERO	Restituisce il controllo al programma monitor. La si utilizza come ultima istruzione di un programma al posto di HALT perchè quest'ultima costringerebbe ad eseguire un RESET per poter dare altri comandi al MPF 1PLUS.	FF

Con il tasto **RESET** si ha un "WARM RESET": il controllo viene ripreso dal programma monitor (operazioni della CPU interrotte) che reinizializzerà la CPU.
Il monitor controlla se sta avvenendo un "COLD" o "WARM" RESET.
Il "COLD" si ha all'accensione od in seguito ad una interruzione dell'alimentazione, comporta che anche le variabili d'utente vengono inizializzate.
Il "WARM" RESET lascia inalterate le variabili d'utente.
Lo si può eseguire ogni qualvolta la CPU ha eseguito un'istruzione sconosciuta o si "perde" (impianta) durante l'esecuzione di un comando o istruzione, restituendo in questo modo il controllo al programma monitor.

RST 30H causa un break software al programma in esecuzione.
Il programmatore può piazzare questa istruzione alla locazione dove intende esaminare il risultato (ad es. Il contenuto dei registri o di locazioni di memoria) dopo aver eseguito un certo numero di istruzioni.
Breakpoint software multipli possono essere inseriti in modo da consentire il debugging del programma.
Dopo che la CPU ha eseguito un'istruzione RST 30H il controllo viene ripreso dal programma monitor; premendo il tasto <G> seguito da <INVIO> la CPU del MPF 1PLUS continuerà ad eseguire il programma dall'istruzione che segue il break software.

PRINCIPALI ISTRUZIONI DELLO Z80

Codice Mnemonico	DESCRIZIONE
LD	LOAD
ADD	ADDIZIONA
CPL	COMPLEMENTA
SUB	SOTTRAZIONE
RST	RESTART
HALT	ALT
NEG	NEGAZIONE (Cambia segno)
NOP	NO OPERATION
INC	INCREMENTA
DEC	DECREMENTA
JR	SALTO RELATIVO
JP	SALTO ASSOLUTO
DJNZ	DECREMENTA B e SALTA in maniera relativa
CP	COMPARA
AND	AND logico
OR	OR logico
XOR	OR ESCLUSIVO logico
BIT	TEST SU BIT
SET	SETTA BIT
RES	RESETTA BIT
PUSH	SPINGE NELL'AREA DI STACK
POP	PRELEVA DALL'AREA DI STACK
CALL	CHIAMATA DI UNA SUBROUTINE
RET	RITORNO DA UNA SUBROUTINE
EX AF,A'F'	SCAMBIA LA COPPIA AF CON LA COPPIA A'F'
EXX	SCAMBIA TUTTO IL BANCO DI REGISTRI PRINCIPALI CON IL BANCO DI REGISTRI ALTERNATIVI
IN	ACQUISIZIONE NELL'ACCUMULATORE DI UN DATO IN INGRESSO DA UNA DELLE DUE PORTE DELLA PIO
OUT	TRASMETTE UN DATO DALL'ACCUMULATORE AD UNA DELLE DUE PORTE DELLA PIO
SCAN1	SUBROUTINE MEMORIZZATA NELLA ROM CHE SCANDISCE TUTTA LA TASTIERA PER VERIFICARE SE è STATO PREMUTO UN TASTO , SI TROVA ALL'INDIRIZZO: 02 9BH (richiede 15,67 mS circa per essere eseguita)

TECNICHE DI INDIRIZZAMENTO

ISTRUZIONE	INDIRIZZAMENTO	DESCRIZIONE	CODICE OGGETTO
LD r,n	**IMMEDIATO**	Carica il registro r con il dato immediato n: r← n r può essere A,B,C,D,E,H,L	7D __
Esempio:			
LD A,08H			7D 08
LD rr',NN	**IMMEDIATO ESTESO**	Carica la coppia di registri con il dato immediato NN, : rr' ← NN rr' possono essere : BC, DE, HL	
LD r,r'	**IMPLICITO**	Il contenuto del registro sorgente r' è copiato nel registro destinazione r r ed r' possono essere A,B,C,D,E,H,L	
Esempio: **LD A,B**		Copia nell'accumulatore il contenuto del registro B	
LD A,(NN)	**DIRETTO**	Copia nell'accumulatore il contenuto della cella di memoria il cui indirizzo è precisato nell'istruzione stessa Nel codice oggetto l'indirizzo va scritto mettendo per primo il byte basso e dopo il byte alto	
Esempio:			
LD A,(F100)		Copia in A il contenuto della locazione di memoria F100	3A 00 F1
LD A,(HL)	**INDIRETTO**	Copia nell'accumulatore il contenuto della cella di memoria il cui indirizzo è il valore della coppia di registri HL	
LD (IX+d),n	**INDICIZZATO-IMMEDIATO**	Carica il valore immediato n nella cella di memoria il cui indirizzo è dato dal valore del registro indice IX più il byte di spiazzamento d. Nel codice oggetto bisognerà precisare il valore d che verrà considerato come valore con segno, quindi è possibile precisare valori di spiazzamento che vanno da +127 a -128. Se il valore di d è uguale a zero, la cella di memoria sarà proprio quella di indirizzo IX.	
Esempio: **LD (IX+3),07**		Se ad es. IX=FB00 il valore immediato 07H verrà copiato nella cella di memoria di indirizzo FB03	
LD (IX+d),r	**INDICIZZATO**	Copia il contenuto del registro r nella cella di memoria di indirizzo uguale a IX+d	
Esempi: **LD (IX+2),B**			DD 70 02
LD (IX+0A),C			DD 71 0A

ISTRUZIONI DELLO Z80 UTILIZZATE PIÙ FREQUENTEMENTE

ISTRUZIONE	EFFETTO	INDIRIZZAMENTO	DESCRIZIONE	CODICE OGGETTO
CPL	A ⟵ A	**IMPLICITO**	Viene fatto il complemento ad uno del contenuto dell'accumulatore ed il risultato viene scritto nell'accumulatore stesso	2F
Esempio: **CPL**			Se A=1101 0100 ($D4_H=212_{10}$) dopo l'esecuzione di CPL si avrà: A=0010 1011 ($2B_H=43_{10}$)	2F
ADD A,n	A ⟵ A+n	**IMMEDIATO**	Il contenuto di A viene sommato al valore immediato n ed il risultato va in A	C6 n
Esempio: **ADD A,CF**	A ⟵ A+CF		Se ad es. A=10 dopo sarà A=DF	C6 CF
ADD A,r	A ⟵ A+r	**IMPLICITO**	Il contenuto di A viene sommato con il contenuto del registro r ed il risultato va in A. r può essere A,B,C,D,E,H,L	
Esempio: **ADD A,A**	A ⟵ A+A		Il contenuto di A viene sommato a se stesso ed il risultato va in A	87
ADD A,B	A ⟵ A+B		Il contenuto di A viene sommato al contenuto di B ed il risultato va in A	80
ADD A,C	A ⟵ A+C		Il contenuto di A viene sommato al contenuto di C ed il risultato va in A	81
ADD A,(IX+d)	A⟵ A+(IX+d)	**INDICIZZATO**	All'accumulatore viene sommato il contenuto della cella di memoria di indirizzo IX+d	DD 86 d
Esempio: **ADD A,(IX+F0)**	A⟵A+(IX+F0)		Se IX=FC00 all'accumulatore verrà sommato il contenuto della cella di memoria FCF0	DD 86 F0
ADD A,(HL)	A⟵A+(HL)	**INDIRETTO**	All'accumulatore verrà sommato il contenuto della cella di memoria puntata da HL	86
Esempio: **ADD A,(HL)**	A⟵A+(HL)	**INDIRETTO**	Se HL=F100 ad A viene sommato il contenuto della cella di memoria F100	86
NEG	A⟵0 - A	**IMPLICITO**	Cambia segno all'accumulatore Il contenuto di A è sottratto da zero ed il risultato va in A	ED 44
NOP	NESSUNO		Non esegue nessuna operazione per un ciclo M(circa 2 mS a 2 MHZ), vengono soltanto rinfrescate le memorie dinamiche	
HALT	CPU sospesa		Ferma la CPU. La CPU sospende il suo funzionamento ed esegue i NOP così da continuare i cicli di rinfresco della memoria fino a quando è ricevuto un interrupt o un reset da tastiera.	76
RST p	(SP-1)⟵ PC_{ALTO}; (SP-2)⟵ PC_{BASSO} SP⟵SP-2; PC_{ALTO}⟵0 PC_{BASSO} ⟵p	**PAGINA ZERO**	Il contenuto del Program Counter è spinto nello STACK (come con l'istruzione PUSH) il valore p è caricato nel PC la cui parte bassa viene azzerata (per questo si parla di indirizzamento in pagina zero) ; la prossima istruzione che verrà eseguita sarà proprio quella contenuta all'indirizzo p (a partire da cui sono memorizzate alcune subroutines) p può essere: 00H =000 / 08H=001 / 10H=010 / 18H=011 / 20H=100 / 28H=101 / 30H=110 / 38H=111	
RST 38H		PAGINA ZERO	Restituisce il controllo al programma monitor. La si utilizza come ultima istruzione di un programma al posto di HALT perchè quest'ultima costringerebbe ad eseguire un RESET per poter dare altri comandi al MPF 1PLUS.	FF

IL REGISTRO DEI FLAG (F)

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

- **RIPORTO (C).**
  Il bit C viene influenzato dal risultato di operazioni aritmetiche (ADD e SUB) nel caso in cui si verifichi un riporto; da istruzioni di TEST (CP); logiche (AND, OR, XOR) in questo caso il bit C sarà sempre resettato, quindi queste istruzioni possono essere usate per azzerarlo. Inoltre tutte le istruzioni di SHIFT e ROTATE lo usano e lo influenzano a seconda del valore del bit che esce dal registro.

- **SOTTRAZIONE (N).**
  Il bit N non è normalmente usato dal programmatore in quanto non può essere testato.
  Esso viene posto a zero da operazioni di somma, logiche, rotazione, LD, BIT.
  Viene posto a uno da SUB, CP, NEG, CPL, I/O.

- **PARITÀ/OVERFLOW (P/V)**
  Questo flag compie due diverse funzioni: alcune istruzioni lo influenzano a seconda della parità del risultato, altre a seconda se si è verificato o meno un overflow.
  La parità è determinata dal numero degli "uno", se questo numero è dispari il bit sarà regolato a zero, se è pari il bit sarà regolato a uno.
  In riferimento alle modalità di controllo nelle operazioni di trasmissione dati si parla di parità dispari (PO) o parità pari (PE).
  La seconda funzione essenziale di questo bit del registro di flag è quella di segnalare il verificarsi di un overflow, cioè il fatto che, durante un'addizione o una sottrazione il segno del risultato è "accidentalmente" cambiato a causa dell'overflow del risultato nel bit di segno.

- **MEZZO RIPORTO (H).**
  Indica un eventuale riporto dal bit 3 al bit 4 durante un'operazione aritmetica, rappresenta cioè il riporto dal nibble (4 bit) di ordine inferiore in quello di ordine superiore.
  È principalmente usato nelle operazioni BCD.
  Sarà posto a uno quando c'è un riporto dal 3° al 4° bit ed azzerato quando non c'è nessun riporto.
  È condizionato da operazioni di ADD, SUB, INC, DEC, CP e logiche.
  Anche per questo bit (analogamente a quello di sottrazione) non sono disponibili istruzioni che permettono di testarlo.

- **ZERO (Z).**
  È usato per indicare se il valore di un byte che è stato calcolato, e che sta per essere trasferito, è zero.
  È anche usato per le istruzioni di CP per indicare uguaglianza (e per altre funzioni miste). Nel caso di un'operazione che produce risultato uguale a zero è posto a uno, altrimenti viene resettato.
  Nel caso delle istruzioni di CP, il bit Z è posto a uno ogni volta che il confronto riesce, altrimenti viene resettato.

- **SEGNO (S).**
  Riflette il valore del bit più significativo di un risultato oppure di un byte che sta per essere trasferito (bit N° 7).
  Nel codice mnemonico si usa la notazione P ad indicare il segno positivo ed M ad indicare il segno negativo.

Ad es:

```
        AND A
        JP P,F400       esegue il salto se il contenuto di A è positivo
```

```
        AND A
        JP M,F400       esegue il salto se il contenuto di A è negativo
```

Le istruzioni di **JUMP**

Le istruzioni di salto permettono di modificare l'ordine sequenziale con cui vengono eseguite le istruzioni del programma.
L'esecuzione di un'istruzione di salto può essere condizionata al verificarsi di determinate condizioni, se la condizione posta è verificata si esegue il salto all'indirizzo precisato nell'istruzione

di salto stessa, altrimenti si continua con l'esecuzione dell'istruzione successiva così come normalmente avviene.
Ciò rende possibile condizionare l'esecuzione di gruppi di istruzioni al verificarsi di determinate situazioni (che è possibile testare solo perchè esistono le corrispondenti istruzioni di salto che permettono di farlo) e quindi strutturare il programma come se fosse costituito da più blocchi di istruzioni invece che da un unico blocco che viene sempre eseguito dalla prima all'ultima istruzione.

Si hanno a disposizione due tipi di salto: **Relativo** ed **Assoluto**.

Entrambi possono essere **condizionati** od **incondizionati**.

Le condizioni che è possibile testare sono i valori di alcuni bit del registro di flag.
La differenza fra i due tipi di salto consiste sia nella diversa modalità di determinazione dell'indirizzo di destinazione del salto sia nelle condizioni che è possibile testare.

SALTO RELATIVO (JR)

Per quanto riguarda la modalità di determinazione dell'indirizzo di destinazione, il salto relativo (JR) la calcola sommando al valore del PC (Program Counter) un valore byte considerato con segno che viene precisato nell'istruzione stessa.
Questo valore viene chiamato di **offset** o **spiazzamento** ed essendo interpretato con segno permette di definire valori che vanno da -128 a +127. Di conseguenza è possibile definire come destinazione del salto istruzioni che iniziano da -126 (salto indietro) a + 129 (salto avanti) byte rispetto il valore del PC perchè quest'ultimo punta alla cella di memoria in cui inizia l'istruzione successiva a quella di salto essendo questa costituita da due byte.
Per quanto riguarda le condizioni che è possibile testare con il salto relativo queste sono limitate a due soli bit del registro di flag: il bit Z ed il bit C.
Le condizioni sono quindi quattro: **Z**, **NZ**, **C**, **NC**.

SALTO ASSOLUTO (JP)

Il salto assoluto (JP) determina l'indirizzo di destinazione sostituendo interamente il valore del PC con il valore precisato nell'istruzione di salto stessa.
Di conseguenza l'istruzione di salto assoluto risulta costituita da tre byte (invece che da due come il salto relativo) ma ha il vantaggio di non avere nessun limite per quanto riguarda la destinazione di salto perchè è possibile precisare come destinazione del salto una qualunque delle 64 K di celle di memorie indirizzabili con il bus indirizzi (a 16 bit) dello Z80.
Per quanto riguarda le condizioni che è possibile testare con il salto assoluto, queste sono tutte quelle permesse e cioè lo stato dei quattro bit Z, C, P/V, S del registro di flag.
Le condizioni sono quindi otto: Z, NZ, C, NC, PO, PE, P ed M.

La tabella seguente mostra le condizioni che è possibile testare con i due tipi di salto.

	{	{	**Z**=>Zero (Z=1)
	{	**JR** {	**NZ**=>Non zero (Z=0)
	{	{	**C**=>Riporto (C=1)
JP	{	{	**NC**=>Nesssun Riporto (C=0)
	{		**PO**=>Parità Dispari (P=0)
	{		**PE**=>Parità Pari (P=1)
	{		**P**=>Positivo (S=0)
	{		**M**=>Negativo (S=1)

Come si può vedere dalla tabella il salto relativo può testare solo quattro condizioni mentre il salto assoluto tutte le otto condizioni possibili.

È ovvio che nelle istruzioni di salto incondizionate il salto viene sempre eseguito.

ISTRUZIONI DELLO Z80 UTILIZZATE PIÙ FREQUENTEMENTE (continua)

ISTRUZIONE	EFFETTO	INDIRIZZAMENTO	DESCRIZIONE	CODICE OGGETTO
JR e	PC ← PC+e		Salto relativo non condizionato (viene sempre eseguito) Il valore di offset (e) è sommato al PC usando la notazione in complemento a due, così da permettere salti in avanti o indietro. La prossima istruzione che sarà eseguita in questo modo è quella che inizia dall'indirizzo PC+e.	18 __
JR cc,e	se cc è vero allora: PC ← PC+e		Come sopra solo che il salto viene eseguito solo se la condizione testata è vera. cc può essere: NZ, Z, NC, C.	
JP pq	PC ← PC+pq		Salto assoluto non condizionato Il contenuto della locazione di memoria che segue quella in cui è contenuto il codice operativo dell'istruzione stessa (q) viene caricato nella parte bassa del PC e quella che segue (p) viene caricata nella parte alta del PC. Quindi l'istruzione che verrà eseguita dopo questa sarà quella contenuta nella locazione di memoria di indirizzo pq.	C3 qp
JP (IX)	PC ← IX		Il contenuto del registro indice IX viene trasferito nel PC, per cui la prossima istruzione che verrà eseguita sarà quella il cui codice oggetto inizia dall'indirizzo di valore IX.	
JP (IY)	PC ← IY		Come sopra	
JP (HL)	PC ← HL		Come sopra	
JP cc,pq	se cc è vera: PC ← pq		Come per il salto assoluto incondizionato solo che l'istruzione ha effetto solo se è verificata la condizione posta, altrimenti non viene fatto il trasferimento di pq nel PC e si continua l'esecuzione del programma dall'istruzione successiva a quella di salto. cc può essere: NZ, Z, NC, C, PO, PE, P, M.	
DJNZ e	B ← B-1 se B><0 allora PC ← PC+e		Decrementa B e salta in modo relativo se B >< 0. Il contenuto del registro B è decrementato e se il risultato del decremento è diverso da zero il valore di offset (e) è sommato al PC così come avviene nel salto relativo peremttendo salti sia indietro che avanti. L'istruzione DJNZ equivale alle istruzioni: DEC B JR NZ,e	10 __
CP s	A-s		Esegue la sottrazione A-s e scarta il risultato, ottenendo l'effetto di influenzare il registro di flag, per cui dopo è possibile testare con istruzioni di salto condizionato la relazione esistente tra il contenuto dell'accumulatore e l'operando s.	

L'istruzione CP s confronta l'operando s con l'accumulatore.
L'operando specificato è sottratto dall'accumulatore ed il risultato è scartato, per cui il contenuto di A rimane invariato, l'esecuzione dell'istruzione influenza però il registro dei flag che può essere usato per la verifica della condizione di un'istruzione di salto.
Ad es.: facendo CP s, se il bit C (carry) del flag è posto a 1 dopo l'esecuzione di Cp, significa che il dato contenuto nell'accumulatore è minore del dato puntato (o contenuto) in s poichè è stato generato un riporto dall'operazione A-s quindi si ha : A<s.
Se C=0 allora A>=s.
Se il contenuto di A è uguale ad s, sarà posto ad 1 il bit Z poichè l'operazione A-s avrà dato risultato uguale a zero.
Quindi se si vuole verificare l'uguaglianza, dopo l'istruzione CP andrà programmata un'istruzione di salto condizionata al valore del bit Z.

L'operando s può essere: n, (HL), (IX+d), (IY+d)

In breve:
C=0 => (A>s) o (A=s) C=1 => (A<s) Z=1 => (A=s)

ISTRUZIONI DELLO Z80 UTILIZZATE PIÙ FREQUENTEMENTE **(continua)**

ISTRUZIONI LOGICHE

ISTRUZIONE	EFFETTO	INDIRIZZAMENTO	DESCRIZIONE	CODICE OGGETTO
AND s	A ⟵ A ^ s	IMPLICITO	Esegue l'operazione di AND logico fra l'accumulatore e l'operando s e pone il risultato in A. L'operando s può essere: n, r, (HL), (IX+d), (IY+d). Un'operazione di AND logico può essere utilizzata per fare una "mascheratura" di un valore in modo che alcuni suoi bit (quelli messi a zero nel valore di "maschera" che si imposta come valore costante) non possano mai assumere valore uno , qualunque sia il valore dell'altro operando che si elabora.[1]	
OR s	A ⟵ A∨ s	IMPLICITO	Esegue l'operazione di OR logico fra l'accumulatore e l'operando s e pone il risultato in A. L'operando s può essere: n, r, (HL), (IX+d), (IY+d). Un'operazione di OR logico può essere utilizzata per fare una "forzatura" di un valore in modo che alcuni suoi bit (quelli messi a uno nel valore di "maschera " che si imposta come costante) assumano sempre valore 1 , qualunque sia il valore dell'altro operando che si elabora.	
XOR s	A ⟵ A ∀ s	IMPLICITO	Esegue l'operazione di OR ESCLUSIVO tra l'accumulatore e l'operando s ed il risultato è depositato in A. L'operazione XOR A (tra l'accumulatore e se stesso) azzera l'accumulatore.	

[1] Le operazioni logiche AND A ed OR A (tra l'accumulatore e se stesso) possono servire ad influenzare i bit del registro di flag senza modificare il valore contenuto nell'accumulatore.

ISTRUZIONI DI MANIPOLAZIONE BIT

ISTRUZIONE	EFFETTO	INDIRIZZAMENTO	DESCRIZIONE	CODICE OGGETTO
BIT b,r	Z ← br (negato)		Prova il bit b del registro r. Viene testato il bit b del registro r e viene impostato il bit Z del registro di flag. Subito dopo quest'istruzione si possono eseguire dei salti condizionati al valore di Z verificando così il valore del bit testato. Se Z=1 allora il bit b vale zero, se Z=0 allora il bit b vale 1.	CB __
BIT b,(HL)	Z ← b(HL) (negato)	INDIRETTO	Come sopra, solo che il bit b è quello del dato contenuto nella cella di memoria indirizzata da HL.	CB __
BIT b,(IX+d)		INDICIZZATO	Come sopra, solo che il bit b è quello del dato contenuto nella cella di memoria indirizzata da IX+d.	DD CB __
BIT b,(IY+d)		INDICIZZATO	Come sopra, solo che il bit b è quello del dato contenuto nella cella di memoria indirizzata da IY+d.	FD CB __

SET b,s	S_b ← 1		Pone ad 1 il bit b dell'operando s. Serve a "forzare" al valore 1 (settare) il bit b dell'operando s. L'operando s può essere r, (HL), (IX+d), (IY+d).	
SET b,r	rb ← 1		Pone ad 1 il bit b del registro r.	CB __
SET b,(HL)	(HL)b ← 1	INDIRETTO	Pone ad 1 il bit b della cella di memoria indirizzata da HL.	CB __
SET b,(IX+d)	(IX+d)b ← 1	INDICIZZATO	Pone ad 1 il bit b della cella di memoria indirizzata da IX+d.	DD CB __
SET b,(IY+d)	(IY+d)b ← 1	INDICIZZATO	Pone ad 1 il bit b della cella di memoria indirizzata da IY+d.	FD CB __
RES b,r	rb ← 0		Pone a 0 il bit b dell'operando s. Serve a "forzare" al valore 0 (resettare) il bit b dell'operando s. L'operando s può essere r, (HL), (IX+d), (IY+d).	CB __
RES b,(HL)	(HL)b ← 0	INDIRETTO	Pone ad 0 il bit b della cella di memoria indirizzata da HL.	CB __
RES b,(IX+d)	(IX+d)b ← 0	INDICIZZATO	Pone ad 0 il bit b della cella di memoria indirizzata da IX+d.	DD CB __
RES b,(IY+d)	(IY+d)b ← 0	INDICIZZATO	Pone ad 0 il bit b della cella di memoria indirizzata da IY+d.	FD CB __

AREA DI STACK

L'area di STACK (detta anche "REGISTRO di STACK) è una zona della memoria RAM utilizzata per memorizzare temporaneamente dei valori per poterli utilizzare successivamente.
Le operazioni di inserimento e prelievo possono essere programmate in successione non rigorosamente prefissata (non è obbligatorio dopo aver inserito dei dati prelevarli prima di poter inserire altri dati) bisogna perciò stare molto attenti all'ordine con cui si sono inseriti i dati e quindi all'ordine con cui li si preleva per non correre il rischio di scambiare i dati fra di loro.
Non ci sono limiti prefissati al numero di dati che è possibile inserire contemporaneamente nello stack, dipenderà comunque dalla quantità di memoria di cui si dispone o che si vuole destinare all'utilizzo come area di stack.
L'organizzazione dell'area di stack è quella di un registro di tipo LIFO (LAST IN FIRST OUT) cioè, di volta in volta, l'ultimo dato che è stato inserito è il primo che si può prelevare e così via. In altri termini, l'ordine di prelievo dei dati è inverso all'ordine di inserimento, se ad es. si sono inseriti tre dati, il primo dato inserito potrà essere prelevato solo quando sono stati prelevati gli altri due.
Per realizzare questo modo di funzionamento occorre un puntatore delle celle di memoria dell'area di stack in modo che ad ogni operazione che viene fatta con essa si vada a scrivere (o prelevare i dati) nella cella di memoria corretta senza sovrascrivere dei valori già inseriti o prelevare dei dati non giusti.
Nello Z80 lo STACK è organizzato in modo che l'area di memoria ad esso destinata cresca verso l'alto quando si inserisce un dato, cioè il puntatore (SP) viene decrementato ad ogni byte inserito e incrementato ad ogni byte prelevato.
Il puntatore SP punta sempre all'ultimo dato inserito.
Per inserire un valore nello stack si ha a disposizione l'istruzione PUSH e per prelevarlo l'istruzione POP.

ISTRUZIONI DI PUSH e POP

ISTRUZIONE	EFFETTO	INDIRIZZAMENTO	DESCRIZIONE	CODICE OGGETTO
PUSH qq	(SP-1)⟵ qq_{ALTO}; (SP-2) ⟵qq_{BASSO}; SP ⟵ SP-2		Spinge la coppia di registri qq nello STACK. Il puntatore dello stack è decrementato ed il contenuto del byte alto della coppia di registri specificata è caricato in (SP-1), SP è decrementato di nuovo ed il contenuto del byte basso della coppia di registri è caricato in (SP-2). SP alla fine è decrementato di 2 e punta comunque all'ultimo dato inserito nello stack. qq possono essere: BC, DE, HL, AF.	
PUSH IX			Spinge il valore del registro indice IX nello stack.	DD E5
PUSH IY			Spinge il valore del registro indice IY nello stack.	FD E5
POP qq	qq_{BASSO} ⟵ (SP) qq_{ALTO} ⟵ (SP+1) SP ⟵ SP+2		Ripristina gli ultimi due byte inseriti nello stack nella coppia di registri specificata. Il contenuto di (SP) è caricato nel byte basso della coppia di registri specificata ed SP è incrementato; il contenuto di (SP+1) è caricato nel byte alto della coppia ed SP viene incrementato di nuovo in modo da puntare sempre all'ultimo dato inserito nello stack.	
POP IX			Ripristina gli ultimi due byte inseriti nell'area di stack nel registro indice IX.	DD E1
POP IY			Ripristina gli ultimi due byte inseriti nell'area di stack nel registro indice IY.	FD E1

Scambio di valori di coppie di registri con le istruzioni PUSH e POP:

Le istruzioni PUSH e POP possono servire a copiare il valore contenuto in una coppia di registri od a scambiare il contenuto di due coppie di registri (uno solo nel caso di registri a 16 bit). Questo perché una volta che il dato è stato inserito nello stack (qualunque sia stata la sua origine) lo si può ripristinare in una qualunque delle coppie e non necessariamente quella che conteneva il dato originario.
Basta quindi programmare nella giusta sequenza le operazioni di PUSH e POP per ottenere l'effetto voluto.

Esempi: per salvare il contenuto di BC e successivamente ripristinarlo:

```
PUSH BC
.....
POP BC
```

per copiare il contenuto della coppia BC nella coppia HL la sequenza è:

```
PUSH BC
.....
POP HL
```

per scambiare il contenuto della coppia DE con la coppia HL:

```
PUSH DE
PUSH HL
POP DE
POP HL
```

Le SUBROUTINES

Spesso si presenta la necessità di dover eseguire più volte determinate operazioni in momenti diversi, ciò comporta che se il programma è organizzato sottoforma di un unico listato di istruzioni risulta molto complesso riuscire a far sì che un gruppo di istruzioni venga eseguito più volte in momenti diversi (e di volta in volta continuare poi l'esecuzione del programma da punti diversi) senza essere costretti a riscrivere ogni volta le stesse istruzioni.
Ciò comporterebbe un notevole spreco di memoria.
Si può ovviare a questo inconveniente facendo in modo che un gruppo di istruzioni possa essere eseguito quando occorre e terminata la loro esecuzione il microprocessore ritorni ad eseguire le istruzioni seguenti del programma.
In questo caso il gruppo di istruzioni in questione è opportuno che risieda in una zona di memoria diversa da quella in cui è memorizzato il programma principale.
Per interrompere il flusso sequenziale di esecuzione delle istruzioni si dovrà eseguire un salto all'indirizzo della cella di memoria che contiene la prima istruzione di questo gruppo e, cosa molto importante, tenere traccia dell'indirizzo dell'istruzione successiva a quella di chiamata per poter tornare a riprendere la sequenza normale di esecuzione.
Un gruppo di istruzioni con queste caratteristiche viene chiamato subroutine.
Le subroutines sono in pratica dei sottoprogrammi che possono essere richiamati dal programma principale per essere eseguiti tutte le volte che occorre senza dover riscrivere le istruzioni ogni volta.
Le istruzioni che è possibile programmare in una subroutine sono le stesse del programma principale, per quanto riguarda ciò che è possibile fare non vi è quindi nessuna differenza tra un programma ed una subroutine.
Da una subroutine è quindi possibile chiamarne un'altra e così via annidandole a diversi livelli.
Una stessa subroutine può essere chiamata più volte indifferentemente dal programma principale o da altre subroutines.
Al termine dell'esecuzione di una subroutine si dovrà riprendere l'esecuzione del programma principale (o subroutine chiamante) esattamente dall'istruzione successiva a quella di chiamata.
A ciò servono le istruzioni di chiamata e ritorno da una subroutine.

ISTRUZIONI DI CHIAMATA E RITORNO DALLE SUBROUTINES (CALL e RET)

Le istruzioni di chiamata delle subroutines sono in pratica delle istruzioni di salto assoluto con la differenza che devono anche tenere traccia del punto di programma da cui è stata fatta la chiamata della subroutine in modo da poter tornare al programma chiamante (principale o altra subroutine che sia) una volta terminata l'esecuzione della stessa, ed esattamente all'istruzione successiva quella di chiamata per proseguire nell'elaborazione delle istruzioni.
Per far ciò le istruzioni di chiamata di una subroutine, prima di eseguire il salto assoluto alla prima istruzione della subroutine, memorizzano nello stack il contenuto del Program Counter.
Tale valore sarà ripristinato nel PC stesso dalla istruzione di ritorno dalla subroutine. Bisognerà fare attenzione nel caso in cui all'interno della subroutine si utilizzi lo stack, di prelevarne tutti i valori immessi prima di eseguire il ritorno al programma o subroutine chiamante.
Questo perchè l'istruzione di ritorno dalla subroutine si limita a ripristinare nel PC gli ultimi due byte che trova nello stack e ad aggiornare il puntatore dello stack, quindi se nello stack sono stati immessi dei valori durante l'esecuzione della subroutine e non sono stati prelevati, l'istruzione di ritorno ripristinerebbe nel PC un indirizzo che non è quello corretto con effetti imprevedibili.
Analogamente alle istruzioni di salto, le istruzioni CALL e RET possono essere condizionate oppure no.
C'è da tenere presente che **CALL e RET non hanno effetto sul registro di flag.**

ISTRUZIONE	EFFETTO	INDIRIZZAMENTO	DESCRIZIONE	CODICE OGGETTO
CALL pq	(SP-1) ← PC_{ALTO} (SP-2) ←PC_{BASSO} SP ← SP-2 PC ← pq		Chiama senza condizioni la subroutine che inizia dall'istruzione di indirizzo pq. Salva il contenuto del Program Counter nelle prime due posizioni dell'area di STACK, decrementa di due il puntatore dello stack e sostituisce il contenuto del PC con l'indirizzo d'inizio della subroutine che veine chiamata.	CD qp
Ad es.: **CALL Delay**			Se la subroutine Delay inizia dall'indirizzo F3 AC il codice oggetto sarà:	CD AC F3
CALL cc,pq			Come sopra ma solo se la condizione posta è verificata, altrimenti l'esecuzione del programma continua normalmente dall'istruzione successiva. cc è costituita da tutte le condizioni che è possibile testare, proprio come le istruzioni di salto assolute.	
RET	PC_{BASSO} ←(SP) PC_{ALTO} ←(SP+1) SP ← SP+2		Ritorno incondizionato dalla subroutine. Ripristina nel PC gli utlimi due byte presenti nello stack, in questo modo la prossima istruzione ad essere eseguita sarà quella successiva l'istruzione di chiamata dell asubroutine, ciò provoca un ritorno al programma chiamante.	C9
RET cc	Come sopra ma solo se cc è vera.		Ritorno condizionato dalla subroutine. Solo se la condizione posta è vera si ha il ripristino nel PC degli ultimi due valori presenti nello stack. Se la condizione non è vera l'esecuzione delle istruzioni continua dall'istruzione successiva della subroutine. cc può essere: NZ, Z, NC, C, PO, PE, P, M come nelle istruzioni di salto assoluto.	

Funzionamento della PIO Z80.

La PIO Z80 è dotata di due porte ad 8 bit: porta **A** e porta **B**.

Queste due porte possono funzionare nei modi:

USCITA, INGRESSO , BIDIREZIONALE (solo la porta A) e CONTROLLO.

USCITA: modo 0 Lo scambio dei dati avviene solo nella direzione CPU --> PORTA. Cioè in "uscita" guardando dalla CPU.
Non è possibile il trasferimento dati in senso inverso e quindi la CPU può soltanto "trasmettere" dati alla periferica ma non può acquisirne da quest'ultima.

INGRESSO: modo 1 Lo scambio dei dati può avvenire solo nella direzione PORTA-->CPU cioè in "ingresso" guardando dalla CPU.
Non è possibile il trasferimento dati in senso inverso e quindi la CPU può soltanto "acquisire" dati dalla periferica ma non trasmetterli.

BIDIREZIONALE: modo 2 Solo la porta A può funzionare con questa modalità ed in questo modo lo scambio dei dati può avvenire sia dalla CPU verso la PORTA sia in senso inverso. In pratica è una combinazione dei due modi precedenti.
Contemporaneamente a questo modo di funzionamento della porta A, la porta B può funzionare solo in modalità CONTROLLO.

CONTROLLO: modo 3 Questo modo consente il trasferimento continuo dei dati tra CPU e periferica (tramite la PIO), la direzione del trasferimento può essere definita relativamente a ciascun bit del bus dati, tramite una "parola" di controllo che definisce quali bit sono considerare come "ingressi"

(quelli messi ad 1) e quali sono da considerare come "uscite" (quelli messi a 0)*.

* Ad es. Volendo programmare come "ingressi" i primi tre bit della porta e come "uscite" gli altri, si invierà alla porta, dopo averla programmata nel modo "controllo", un byte, detto "maschera di controllo" che varrà 0000 0111 quindi il valore 07H.

Esempio:

```
LD A,FF
OUT (6B),A
LD A,07H
OUT (69),A
```

Programmazione della PIO

Impostazione dei modi di funzionamento:

per programmare una porta nel modo di funzionamento occorre mandare al "**registro controllo**" della porta selezionata una parola opportuna che viene così formata:

7	6	5	4	3	2	1	0
M1	**M0**	**X**	**X**	**1**	**1**	**1**	**1**

I due bit 4 e 5 (X) non sono significativi, si usa però impostarli sempre a zero tranne quando si seleziona il modo "controllo" in cui vengono messi a 1.
I due bit M1 ed M0 assumono i valori corrispondenti al modo di funzionamento prescelto secondo la seguente tabella:

M1	M0	MODO	BYTE
0	0	USCITA	0F
0	1	INGRESSO	4F
1	0	BIDIREZIONALE	8F
1	1	CONTROLLO	FF

Gli indirizzi dei registri "**CONTROLLO**" e "**DATI**" sono:

	CONTROLLO	DATI
PORTA **A**	**6A**	**68**
PORTA **B**	**6B**	**69**

Per lo Z80, soltanto la porta A è programmabile nel modo "BIDIREZIONALE" ed in questo caso la porta B può funzionare contemporaneamente solo nel modo "CONTROLLO".

ISTRUZIONI DI I/O

Per trasmettere dati tra la CPU e la porta PIO si hanno a disposizione le due istruzioni IN e OUT, che permettono rispettivamente di acquisire ed inviare dati ad una delle due porte della PIO Z80.
I dati dovranno comunque transitare nell'accumulatore sia quando vengono acquisiti sia quando vengono inviati.

ISTRUZIONE	EFFETTO	INDIRIZZAMENTO	DESCRIZIONE	CODICE OGGETTO
IN A,(N)	A ← (N)		Trasferisce nell'accumulatore il valore presente nel registro dati della porta , specificato con (N) (cioè il byte che segue il codice operativo dell'istruzione).	DB __
IN r,(C)	r ← (C)		Trasferisce nel registro r il valore presente nel registro dati della porta specificato tramite il valore del registro C. r può essere uno qualunque dei registri di uso generale: A, B, C, D, E, H, L.	ED __
OUT (N),A	(N) ← A	IMMEDIATO	Trasferisce il contenuto dell"accumulatore nel registro dati della porta specificato con (N) (cioè il byte che segue il codice operativo dell'istruzione).	D3 __
OUT (C),r	(C) ← r		Trasferisce il contenuto di r nel registro dati della porta specificato tramite il valore del registro C. r può essere uno qualunque dei registri di uso generale: A, B, C, D, E, H, L.	ED __

Quindi se si vuole trasferire ad una periferica collegata alla porta A della PIO Z80 un valore qualunque (ad es. 01H) bisogna innanzitutto mandare al registro "controllo" della porta il byte di valore OF che imposta la porta nel modo di funzionamento "USCITA" e successivamente si potrà inviare al registro "dati" della porta stessa il valore che si vuole trasmettere alla periferica:

LD A,0FH	} Seleziona modo "uscita"
OUT (6A),A	}
LD A,01H	) Invia dato
OUT (68),A	)

Se si vuole invece acquisire un dato da una periferica collegata alla porta B, si dovrà prima programmare la porta B in modalità "INGRESSO" e successivamente acquisire il dato dalla porta B nell'accumulatore.

Se si vuole invece acquisire un dato da una periferica collegata alla porta B, si dovrà prima programmare la porta B in modalità "INGRESSO" e successivamente caricare il dato dalla porta B all'accumulatore:

LD A,4FH	} Seleziona modo "ingresso "
OUT (6B),A	}
IN A,(69)	) Acquisisci dato

* Si può acquisire un valore numerico impostato tramite i DIP-SWITCHES sulla scheda del MPF-1PLUS. Per fare ciò bisogna utilizzare l'indirizzo 6C che corrisponde ai DIP-SWITCHES, dopo aver impostato il valore sugli stessi (ON = 0, OFF = 1) si acquisisce il valore direttamente nell'accumulatore con l'istruzione :

IN A,(6C)