UNIVERSITA' DEGLI STUDI DI PISA

ISTITUTO DI SCIENZE DELL'INFORMAZIONE

C.Montangero - G.Pacini

IL LINGUAGGIO ASSEMBLATORE SIMBOLCANE
MANUALE DI RIFERIMENTO

Note per il Corso di

Teoria ed applicazioni delle macchine calcolatrici

Anno Accademico 1972/73

IL LINGUAGGIO ASSEMBLATORE SIMBOLCANE(°)

0. Introduzione

Il Simbolcane prevede tre tipi di dichiarazioni:

a) dichiarazione di istruzione;

b) dichiarazione di dati;

c) dichiarazione di comando (pseudo-istruzione).

Ogni dichiarazione può essere riguardata come costituita da tre campi:

a) campo etichetta;

b) campo codice;

c) campo indirizzo.

Per la perforazione delle dichiarazioni su schede devono essere rispettate le le seguenti regole:

1) ogni scheda può contenere una sola dichiarazione;

2) ogni dichiarazione deve essere contenuta in una sola scheda;

3) i contenuti dei campi etichetta, codice ed indirizzo devono essere perforati rispettivamente a partire da colonna 1, 8 e 13;

4) le colonne 7 e 12 non devono avere alcuna perforazione.

Il campo etichetta può essere vuoto oppure contenere un indirizzo simbolico; il campo codice deve contenere (a)il codice mnemonico di una istruzione del CANE, nel caso di dichiarazione di istruzione (b) uno dei simboli OTT, DEC od ALFA, nel caso di dichiarazione di dati (c)uno dei simboli ORIG, FINE o BCS nel caso di dichiarazioni di comando. Il contenuto del campo indirizzo (colonne 13-72) dipende fortemente dal tipo di dichiarazione.

Nel seguito utilizzeremo le seguenti definizioni:

(°) Questa breve nota intende descrivere il linguaggio assemblatore del CANE(Simbolcane) e le sue modalità di utilizzo. Si presuppone che il lettore sia familiare con i linguaggi di tipo assemblatore e conosca i principi di funzionamento dei programmi assemblatori. In particolare si veda: F. Preparata, Introduzione agli assemblatori.

⟨carattere⟩ := uno dei caratteri della tab. I

⟨simbolo⟩ := una sequenza di n caratteri ($1 \leq n \leq 6$) di cui il primo alfabetico, e che non contenga nessuno dei seguenti caratteri speciali

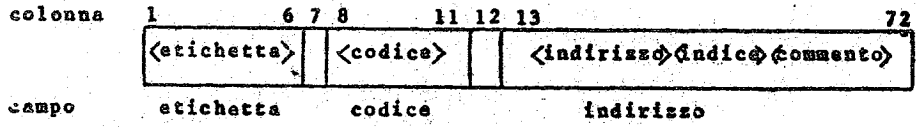
+ - * / = ( ), b

⟨numero 10⟩ := una sequenza di n cifre decimali ($1 \leq n \leq 6$)(***)

⟨numero 8⟩ := sequenza di n cifre ottali ($1 \leq n \leq 6$)

2. Dichiarazione di istruzione

Il formato è il seguente:



Valgono le seguenti definizioni:

⟨etichetta⟩ := ⟨simbolo⟩ | Λ(*)

⟨codice⟩ := ALT | TRA ..... | NOP(**)

⟨indirizzo⟩ := ⟨espressione⟩ | ⟨costante⟩

⟨espressione⟩ := Λ | * | ⟨operando⟩ | ⟨espressione⟩ + ⟨operando⟩ | ⟨espressione⟩ - ⟨operando⟩

⟨operando⟩ := ⟨numero 10⟩ | ⟨simbolo⟩

⟨costante⟩ := =D+⟨numero 10⟩ | =D-⟨numero 10⟩ | =O⟨numero 8⟩ | =A⟨carattere⟩⟨carattere⟩⟨carattere⟩

⟨indice⟩ := Λ | ,⟨cifra ottale⟩

⟨commento⟩ := b⟨sequenza di caratteri qualunque⟩

(*) Il simbolo Λ denota la sequenza vuota (cioè di lunghezza nulla) da non confondersi con il carattere b.

(**) I codici mnemonici del CANE sono riportati in Appendice II.

(***) Il carattere Ø rappresenta lo zero, distinto da O.

Esempi di dichiarazioni di istruzioni:

```
colonna     1       8     13
                    TRA   X
                    MEA   ,1
            ADDA    ADDA  -19,2
                    TRB   ADDA+2,7
                    SAZ   *+41
            BBB     SAP   *+ADDA-BBB
                    TRA   =0723
                    TRA   =0000723
```

Osservazioni

1) ⟨espressione⟩ è sostanzialmente una espressione aritmetica, in cui gli operandi possono essere numeri decimali, indirizzi simbolici e il carattere speciale * ; gli operatori possono essere + e -.

Il simbolo speciale * indica il valore attuale del contatore di locazione, cioè l'indirizzo della locazione in cui verrà memorizzata l'istruzione tradotta. Si noti che il simbolo * può comparire solo come primo operando di una espressione e non può essere preceduto da operatori.

Il valore delle espressioni è calcolato modulo 512.

Una espressione vuota ha valore zero.

2) La specifica =D definisce costanti decimali (il cui valore deve essere compreso tra -131071 e +131071);
la specifica =O definisce costanti ottali;
la specifica =A definisce costanti alfanumeriche.

3) Il commento è completamente trascurato durante la traduzione; in altri termini, il programma assemblatore prende in considerazione soltanto i caratteri del campo indirizzo compresi tra colonna 13 ed il primo carattere b: ad esempio, le seguenti dichiarazioni

```
TRA     X+1 ,2
TRA     X+1
```

vengono tradotte allo stesso modo. Nella prima, infatti, i caratteri ,2 vengono considerati un commento. Unica eccezione a questa regola

è la costante alfanumerica: ad esempio, il campo indirizzo della dichiarazione

```
TRA     =A +X    TRASFERIRE IN RA 6Ø2Ø67
```

definisce correttamente la costante alfanumerica 'b+X' (cfr. tab. I).

3. Dichiarazione di dati

Il Simbolcane prevede tre dichiarazioni di dati: OTT(ottale), DEC (decimale) ed ALFA (alfanumerica). Le prime due permettono di predisporre in memoria delle costanti numeriche, rappresentate, nella dichiarazione, in ottale ed in decimale rispettivamente. L'ultima permette di predisporre in memoria delle configurazioni alfanumeriche.

3.1. Dichiarazione DEC

Il formato è il seguente:

campo

etichetta	codice	indirizzo
⟨etichetta⟩	DEC	⟨sequenza di decimali⟩⟨commento⟩

Valgono le seguenti definizioni:

⟨sequenza di decimali⟩ := ⟨dato decimale⟩ | ⟨sequenza di decimali⟩,⟨dato decimale⟩

⟨dato decimale⟩ := +⟨numero 1Ø⟩ | -⟨numero 1Ø⟩ | ⟨numero 1Ø⟩ | Λ

Esempio:

```
colonna   1       8     13
          D       DEC   +10,-14,,32  DICHIARAZIONE DI 4 DATI
```

Osservazioni

La dichiarazione dell'esempio ha come effetto la memorizzazione della rappresentazione binaria dei quattro dati decimali (1Ø,-14, Ø e 32) in quattro celle consecutive. L'etichetta D assume come valore l'indirizzo della locazione in cui è memorizzata la prima delle costanti dichiarate. Il contatore di locazione (CL) viene incrementato di quattro unità; in generale l'incremento del CL è pari al numero di dati dichiarati. Λ equivale a zero.

3.2. Dichiarazione OTT

Il formato è il seguente:

campo

etichetta	codice	indirizzo
⟨etichetta⟩	OTT	⟨sequenza di ottali⟩⟨commento⟩

Valgono le seguenti definizioni:

⟨sequenza di ottali⟩ := ⟨dato ottale⟩ | ⟨sequenza di ottali⟩,⟨dato ottale⟩

⟨dato ottale⟩ := ⟨numero 8⟩ | Λ

Esempio:

```
colonne   1       8     13
          DATOT OTT     736,,347
```

Osservazioni

Valgono le osservazioni fatte per la dichiarazione DEC.

3.3. Dichiarazione ALFA

Il formato è il seguente:

campo

etichetta	codice	indirizzo
⟨etichetta⟩	ALFA	⟨contatore⟩,⟨sequenza di alfanumer.⟩

Valgono le seguenti definizioni:

⟨contatore⟩ := ⟨cifra 1Ø⟩⟨cifra 1Ø⟩

⟨sequenza di alfanumer.⟩ := ⟨dato alfanumer.⟩ | ⟨sequenza di alfanumer.⟩⟨dato alfanumer.⟩

⟨dato alfanumer.⟩ := ⟨carattere⟩⟨carattere⟩⟨carattere⟩

Esempio:

```
colonne   1       8     13
          DATALF ALFA Ø3,ABb X+72A COMMENTO
```

Osservazioni

Detto n il valore del contatore, vengono considerati i primi 3n caratteri dopo la virgola che segue il contatore. Tali caratteri vengono memorizzati, a gruppi di tre per cella, in n celle consecu-

tive. L'etichetta viene associata all'indirizzo della prima cella interessata dalla dichiarazione.

In riferimento all'esempio, supponendo che CL=3Ø3 quando l'assemblatore incontra la dichiarazione, essa viene tradotta come segue (cfr. tab. I):

```
3Ø3  212260
3Ø4  546720
3Ø5  Ø7Ø221
```

I caratteri successivi vengono ignorati, cioè vengono considerati come commento.

Il CL viene incrementato di una quantità pari al contatore.

4. Dichiarazione di comando

4.1. Dichiarazione ORIG

Il formato è il seguente:

campo	etichetta	codice	indirizzo
	<etichetta>	ORIG	<espressione><commento>

La dichiarazione ORIG definisce il valore del CL, che viene posto uguale al valore dell'espressione nel campo indirizzo. Lo stesso valore viene associato ad una eventuale etichetta.

Osservazioni

L'<espressione>, nelle dichiarazioni ORIG, può contenere simboli solo se essi sono già definiti, cioè se appaiono nel campo etichetta di una dichiarazione dello stesso programma che preceda la dichiarazione ORIG in questione. Ad esempio, la sequenza di dichiarazioni

```
colonna     1     8     13
            A     TRA   B,2
                  .......
                  .......
                  ORIG  A+1ØØ
```

è corretta, mentre

```
                  ORIG  A+25
                  ......
                  ......
            A     TRA   B,2
```


è una sequenza errata.

4.2. Dichiarazione BCS

Il formato è il seguente:

campo	etichetta	codice	indirizzo
	<etichetta>	BCS	<espressione><commento>

Questa dichiarazione incrementa CL del valore dell'espressione. L'etichetta, se presente, assume il valore del CL prima dell'incremento.

Osservazioni

La dichiarazione BCS permette di riservare una o più celle di memoria, senza predisporne il contenuto, che verrà definito solo durante l'esecuzione del programma tradotto. L'etichetta viene associata all'indirizzo della prima cella del blocco di celle riservate.

Valgono le osservazioni fatte a proposito della dichiarazione ORIG: l'<espressione>deve contenere sólo simboli definti.

4.3. Dichiarazione FINE

Il formato è il seguente:

campo	etichetta	codice	indirizzo
	<etichetta>	FINE	<espressione><commento>

Questa dichiarazione indica la fine fisica del programma: per definizione è quindi l'ultima dichiarazione considerata dall'assemblatore.

Il valore dell'espressione fornisce l'indirizzo della istruzione da cui deve iniziare l'esecuzione del programma tradotto.

5. Schede commento

L'assemblatore trascura tutte le schede in cui sia perforato, in colonna 1, il carattere /. In questo modo è possibile inserire, tra le dichiarazioni, dei commenti. Tali commenti vengono stampati, dal programma assemblatore, nella lista del programma sorgente.

6. Utilizzo dell'assemblatore SIMBOLCANE.

Questo ultimo paragrafo descrive le modalità di impiego dell'assemblatore del linguaggio SIMBOLCANE. Poichè dette modalità risentono fortemente e delle limitate dimensioni della memoria del CANE e più in generale della voluta semplicità degli organi periferici, si è ritenuto opportuno iniziare il paragrafo con una descrizione molto schematica dei procedimenti in uso nei sistemi reali.

6.1 Traduzione ed esecuzione di programmi in linguaggio simbolico.

I sistemi di calcolo sono equipaggiati con un insieme di programmi di servizio (software) assai nutrito. L'insieme di detti programmi non può risiedere, per ragioni di spazio, permanentemente in memoria centrale. Il software si trova memorizzato su memorie d'appoggio dette memorie periferiche di massa, solitamente dischi o tamburi magnetici, vedi fig.1.

In memoria centrale risiede in modo permanente solo una piccola parte del sistema software, detta nucleo. Lo scopo fondamentale del nucleo è portare in memoria centrale parti del sistema software (per esempio traduttori) in accordo con le richieste delle schede controllo contenute nel pacco programma sottoposto alla elaborazione del sistema di calcolo.

Vediamo, ad esempio, molto schematicamente, come viene elaborato un pacco programma composto da un programma redatto in simbolico, dai suoi dati e dalle opportune schede controllo. Il pacco inizierà con una scheda controllo richiedente l'assemblatore. Il nucleo, cui è affidato il compito di interpretare le schede controllo, carica in memoria il programma assemblatore, leggendolo da una delle memorie di massa vedi fig. 2a). Il nucleo "cede il controllo" all'assemblatore. L'assemblatore legge le schede contenenti la descrizione simbolica del programma, le elabora e produce un corrispondente programma oggetto (di solito in forma binaria rilocabile).

Il programma oggetto viene temporaneamente memorizzato ancora su una memoria di massa, fig. 2b).

Termina così la fase di traduzione e il programma assemblatore, esaurito il suo compito, restituisce il controllo al nucleo. A questo punto il nucleo, in cui si è supposto incorporato un programma caricatore, in obbedienza ad una scheda controllo che richiede l'esecuzione, carica in memoria centrale il programma prodotto durante la fase di assemblaggio e da inizio alla esecuzione, fig. 2c). Durante l'esecuzione verranno infine lette le schede dati, fig. 2d).

Notiamo che l'elaborazione si articola in due fasi successive e nettamente distinte. Una prima fase di assemblaggio in cui il programma attivo è l'assemblatore: le schede che contengono la descrizione simbolica del programma dell'utente forniscono, in questa fase, l'insieme di dati per l'assemblatore. Nella seconda fase, esecuzione, il programma attivo è il programma di utente, o, più precisamente, il programma binario che l'assemblatore ha prodotto a partire dalla descrizione simbolica che l'utente ha fornito.

6.2 Modalità di impiego dell'assemblatore del CANE.

Come noto il CANE non dispone di memorie periferiche di massa. D'altronde, vista la dimensione della memoria, è irragionevole pretendere che un programma assemblatore risieda permanentemente in memoria centrale.

La difficoltà è stata superata introducendo una organizzazione resa possibile dal fatto che il CANE è in realtà simulato da un programma scritto per un grosso calcolatore, attualmente lo IBM 370/155 del CNUCE. Anche l'assemblatore del CANE è stato realizzato scrivendo un programma per l'IBM 370/155 ed è quindi da riguardare come una "entità esterna" al vero e proprio sistema CANE.

Durante la fase di assemblaggio, l'assemblatore, in esecuzione sul 370/155, legge le schede contenenti il testo simbolico, produce un programma oggetto binario assoluto (non rilocabile) e lo carica direttamente nella memoria del CANE, nella posizione specificata dalla scheda ORIG, fig. 3a).

L'esecuzione del programma tradotto ha inizio immediatamente dopo[1]. L'indirizzo di inizio esecuzione è quello specificato nella scheda FINE.

All'inizio della fase esecutiva, si deve supporre che nel cassetto del lettore di schede del CANE siano rimaste tutte le schede che, nel

(1) Non c'è quindi necessità di una scheda controllo che richieda la esecuzione. L'esecuzione è comunque subordinata alla assenza di errori nel testo simbolico.

pacco programma, seguono quelle contenenti la descrizione del programma in simbolico (cioè, tutte quelle dalla scheda FINE esclusa in poi), fig. 3b).

Poichè, come ormai chiarito, l'assemblatore appare come un agente esterno al vero e proprio sistema CANE, non si è ritenuto opportuno introdurre una scheda controllo per chiedere l'intervento dell'assemblatore. L'intervento dell'assemblatore viene richiesto perforando ASS nelle colonne 58-60 della scheda asterisco (vedi manuale di riferimento del CANE).

La presenza della dicitura ASS sulla scheda asterisco provoca l'immediata attivazione dell'assemblatore, dando inizio al procedimento descritto in fig. 3) ; è quindi necessario che le schede simboliche seguano immediatamente la scheda asterisco.

Il software di servizio descritto nella nota "manuale di riferimento" è, dopo l'assemblaggio, a disposizione e può essere utilizzato normalmente. Si tenga presente, comunque, che l'assemblatore non esegue alcun controllo sulla scheda ORIG. E' quindi possibile caricare il programma oggetto nella zona normalmente dedicata al software descritto nel manuale di riferimento. Si noti che questo permette di sfruttare tutta la memoria del CANE.

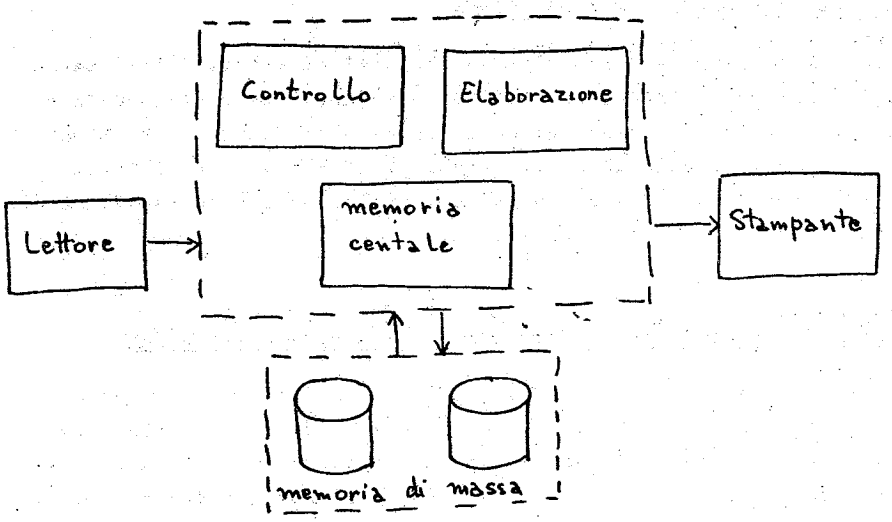
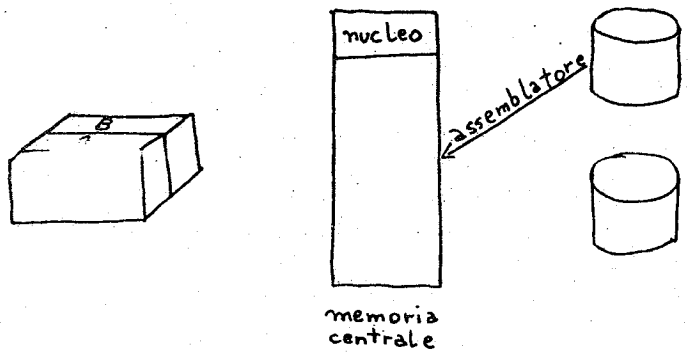
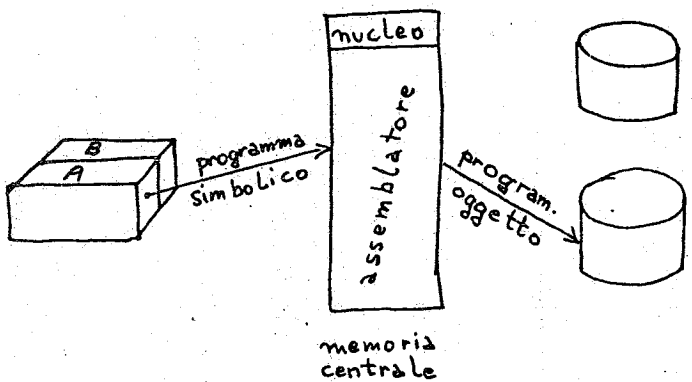


fig. 1

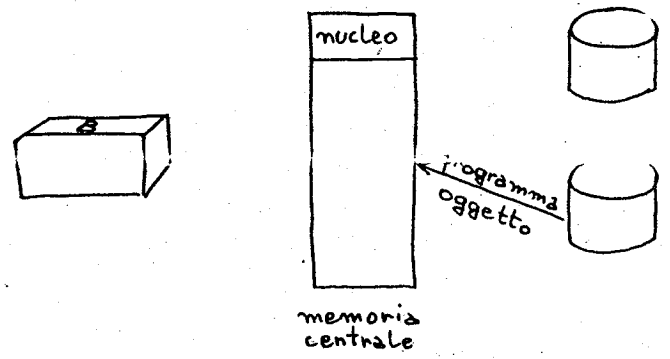


a)

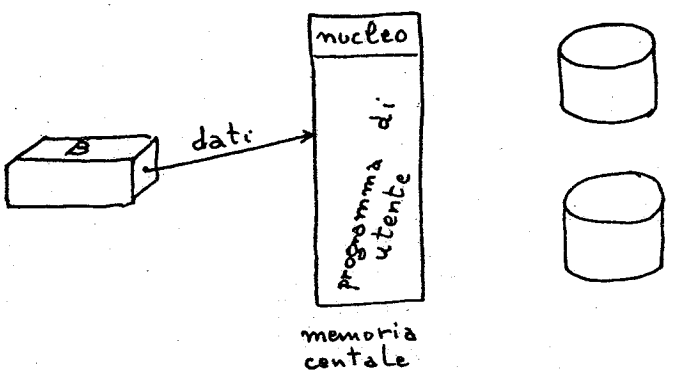


b)

fig. 2

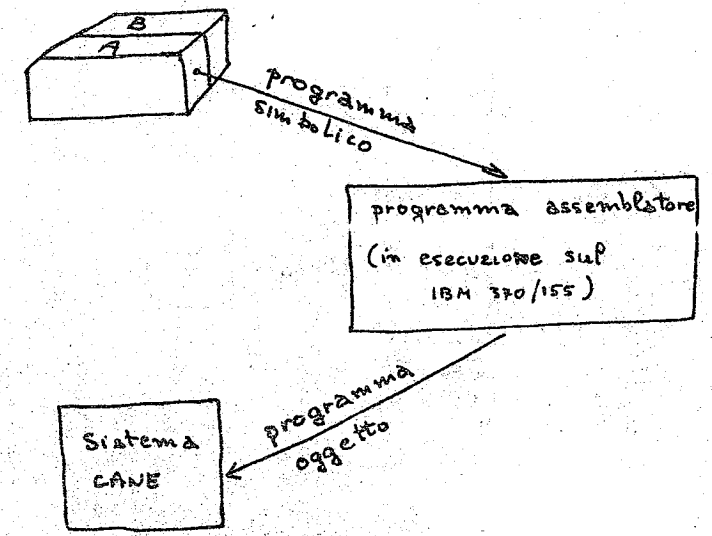


c)

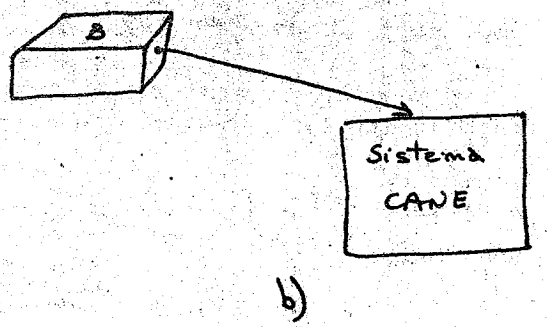


d)

fig. 2



a)



b)

fig. 3

APPENDICE I

Carattere	Codice	Carattere	Codice
A	010001	0	000000
B	010010	1	000001
C	010011	2	000010
D	010100	3	000011
E	010101	4	000100
F	010110	5	000101
G	010111	6	000110
H	011000	7	000111
I	011001	8	001000
J	100001	9	001001
K	100010		
L	100011	b(spazio)	110000
M	100100	.	011011
N	100101	(	111100
O	100110	+	010000
P	100111	$	101011
Q	101000	*	101100
R	101001	)	011100
S	110010	-	100000
T	110011	/	110001
U	110100	,	111011
V	110101	=	001011
W	110110		
X	110111		
Y	111000		
Z	111001		

Tabella I - Codice caratteri

APPENDICE II

Elenco delle istruzioni

00	ALT	
01	TRA	trasferimento: in rA
02	TRAN	trasferimento: in rA negativo
03	TDA	trasferimento: diretto in rA
04	TDAN	trasferimento: diretto in rA negativo
05	TRB	trasferimento: in rB
06	TRX	trasferimento: in rX[J]
07	TDX	trasferimento: diretto in rX[J]
10	MEA	memorizzazione: di rA
11	MEB	memorizzazione: di rB
12	MEZ	memorizzazione: di zero
13	MEX	memorizzazione: di rRX[J]
14	ADA	addizione ad rA
15	SOA	sottrazione da rA
16	ADDA	addizione diretta ad rA
17	SODA	sottrazione diretta da rA
20	ADB	addizione ad rB
21	SOB	sottrazione da rB
22	ADX	addizione ad rX[J]
23	SOX	sottrazione da rX[J]
24	ADDX	addizione diretta ad rX[J]
25	SODX	sottrazione diretta da rX[J]
26	MOL	moltiplicazione
27	DIV	divisione
30	COP	complemento
31	AND	prodotto logico
32	OR	somma logica
33	ORX	somma modulo 2
34	CFA	confronto: con rA
35	CFB	confronto: con rB
36	CFX	confronto: con rX[J]
37	CFXD	confronto: diretto con rX[J]
40	SLT	salto incondizionato
41	SUB	salto a sottoprogramma
42	SCP	salto: se supero
43	SMIN	salto: se minore
44	SMAG	salto: se maggiore
45	SUG	salto: se uguale
46	SAN	salto: se rA negativo
47	SAP	salto: se rA positivo
50	SAZ	salto: se rA zero
51	SBN	salto: se rB negativo
52	SBP	salto: se rB positivo
53	SBZ	salto: se rB zero
54	AVD	spostamento: aritmentico di rA verso destra
55	AVS	spostamento: aritmetico di rA verso sinistra
56	ALD	spostamento: logico di rA verso destra
57	ALS	spostamento: logico di rA verso sinistra
60	BVD	spostamento: aritmetico di rB verso destra
61	BVS	spostamento: aritmetico di rB verso sinistra
62	ABLD	spostamento: logico di rA e rB verso destra
63	ABLS	spostamento: logico di rA e rB verso sinistra
64	TCA	trasferimento carattere in rA
65	MCA	memorizzazione carattere da rA
66	CAR	conversione in caratteri
67	NUM	conversione in numero binario
70	ESG	esegui
72	ENB	entrata binaria
73	ENA	entrata alfanumerica
74	USC	uscita alfanumerica
75	CEN	controllo entrata
76	CUS	controllo uscita
77	NOP	nessuna operazione