DAI PERSONAL COMPUTER

assembler

8080

8080 assembler

INSTRUCTION	FUNCTION									HEX
MOVE GROUP		reg	A	B	C	D	E	H	L	M
MOV A, reg	(A)←(reg)		7F	78	79	7A	7B	7C	7D	7E
MOV B, reg	(B)←(reg)		47	40	41	42	43	44	45	46
MOV C, reg	(C)←(reg)		4F	48	49	4A	4B	4C	4D	4E
MOV D, reg	(D)←(reg)		57	50	51	52	53	54	55	56
MOV E, reg	(E)←(reg)		5F	58	59	5A	5B	5C	5D	5E
MOV H, reg	(H)←(reg)		67	60	61	62	63	64	65	66
MOV L, reg	(L)←(reg)		6F	68	69	6A	6B	6C	6D	6E
MOV M, reg	(M)←(reg)		77	70	71	72	73	74	75	—
ACCUMULATOR GROUP										
ADD reg	(A)←(A)+(reg)	•	87	80	81	82	83	84	85	86
ADC reg	(A)←(A)+(reg)+(CY)	•	8F	88	89	8A	8B	8C	8D	8E
SUB reg	(A)←(A)-(reg)	•	97	90	91	92	93	94	95	96
SBB reg	(A)←(A)-(reg)-(CY)	•	9F	98	99	9A	9B	9C	9D	9E
ANA reg	(A)←(A)∧(reg)	•	A7	A0	A1	A2	A3	A4	A5	A6
XRA reg	(A)←(A)⊻(reg)	•	AF	A8	A9	AA	AB	AC	AD	AE
ORA reg	(A)←(A)∨(reg)	•	B7	B0	B1	B2	B3	B4	B5	B6
CMP reg	(A)-(reg)	•	BF	B8	B9	BA	BB	BC	BD	BE
INCREMENT/DECREMENT REGISTER										
INR reg	(reg)←(reg)+1	••	3C	04	0C	14	1C	24	2C	34
DCR reg	(reg)←(reg)-1	••	3D	05	0D	15	1D	25	2D	35
REGISTER PAIR GROUP		rp	B	D	H	SP	PSW			
INX rp	(rp)←(rp)+1		03	13	23	33	—			
DCX rp	(rp)←(rp)-1		0B	1B	2B	3B	—			
LDAX rp	(A)←((rp))		0A	1A	—	—	—			
STAX rp	((rp))←(A)		02	12	—	—	—			
DAD rp	(H,L)←(H,L)+(rp) ••		09	19	29	39	—			
PUSH rp	((SP)-1)←(rn),((SP)-2)←(rl), (SP)←(SP)-2		C5	D5	E5	—	F5			
POP rp	(rl)←((SP)),(rh)←((SP)+1), (SP)←(SP)+2		C1	D1	E1	—	F1	•		
DIRECT ADDRESS GROUP										
LDA addr	(A)←(addr)		3A	al	ah					
STA addr	(addr)←(A)		32	al	ah					
LHLD addr	(L)←(addr),(H)←(addr+1)		2A	al	ah					
SHLD addr	(addr)←(L),(addr+1)←(H)		22	al	ah					
IMMEDIATE GROUP										
MVI A, data	(A)←data		3E	dd						
MVI B, data	(B)←data		06	dd						
MVI C, data	(C)←data		0E	dd						
MVI D, data	(D)←data		16	dd						
MVI E, data	(E)←data		1E	dd						
MVI H, data	(H)←data		26	dd						
MVI L, data	(L)←data		2E	dd						
MVI M, data	(M)←data		36	dd						
ADI data	(A)←(A)+data	•	C6	dd						
ACI data	(A)←(A)+data+(CY)	•	CE	dd						
SUI data	(A)←(A)-data	•	D6	dd						
SBI data	(A)←(A)-data-(CY)	•	DE	dd						
ANI data	(A)←(A)∧data	•	E6	dd						
XRI data	(A)←(A)⊻data	•	EE	dd						
ORI data	(A)←(A)∨data	•	F6	dd						
CPI data	(A)-data	•	FE	dd						
LXI B, addr	(B)←ah,(C)←al		01	al	ah					
LXI D, addr	(D)←ah,(E)←al		11	al	ah					
LXI H, addr	(H)←ah,(L)←al		21	al	ah					
LXI SP,addr	(SP_h)←ah,(SP_l)←al		31	al	ah					

INSTRUCTION	FUNCTION		HEX
JUMP GROUP			
JMP addr	(PC)←addr		C3 al ah
JNZ addr	If Z=0, (PC)←addr		C2 al ah
JZ addr	If Z=1, (PC)←addr		CA al ah
JNC addr	If CY=0,(PC)←addr		D2 al ah
JC addr	If CY=1,(PC)←addr		DA al ah
JPO addr	If P=0, (PC)←addr		E2 al ah
JPE addr	If P=1, (PC)←addr		EA al ah
JP addr	If S=0, (PC)←addr		F2 al ah
JM addr	If S=1, (PC)←addr		FA al ah
PCHL	(PC_h)←(H), (PC_l)←(L)		E9
CALL GROUP			
CALL addr	((TOS)←(PC),(PC)←addr		CD al ah
CNZ addr	If Z=0, ((TOS)←(PC),(PC)←addr		C4 al ah
CZ addr	If Z=1, ((TOS)←(PC),(PC)←addr		CC al ah
CNC addr	If CY=0,((TOS)←(PC),(PC)←addr		D4 al ah
CC addr	If CY=1,((TOS)←(PC),(PC)←addr		DC al ah
CPO addr	If P=0, ((TOS)←(PC),(PC)←addr		E4 al ah
CPE addr	If P=1, ((TOS)←(PC),(PC)←addr		EC al ah
CP addr	If S=0, ((TOS)←(PC),(PC)←addr		F4 al ah
CM addr	If S=1, ((TOS)←(PC),(PC)←addr		FC al ah
	N.B. ((TOS)←(PC) designates the following:- ((SP)-1)←(PC_h),((SP)-2)←(PC_l),(SP)←(SP)-2		
RETURN GROUP			
RET	(PC)←((TOS)		C9
RNZ	If Z=0, (PC)←((TOS)		C0
RZ	If Z=1, (PC)←((TOS)		C8
RNC	If CY=0,(PC)←((TOS)		D0
RC	If CY=1,(PC)←((TOS)		D8
RPO	If P=0, (PC)←((TOS)		E0
RPE	If P=1, (PC)←((TOS)		E8
RP	If S=0, (PC)←((TOS)		F0
RM	If S=1, (PC)←((TOS)		F8
	N.B. (PC)←((TOS) designates the following:- (PC_l)←((SP)),(PC_h)←((SP)+1),(SP)←(SP)+2		
RESTART GROUP			
RST 0	((TOS)←(PC),(PC)←0_{16}		C7
RST 1	((TOS)←(PC),(PC)←8_{16}		CF
RST 2	((TOS)←(PC),(PC)←10_{16}		D7
RST 3	((TOS)←(PC),(PC)←18_{16}		DF
RST 4	((TOS)←(PC),(PC)←20_{16}		E7
RST 5	((TOS)←(PC),(PC)←28_{16}		EF
RST 6	((TOS)←(PC),(PC)←30_{16}		F7
RST 7	((TOS)←(PC),(PC)←38_{16}		FF
ROTATE/CONTROL/SPECIAL GROUP			
RLC	(A_{n+1})←(A_n),(A_0)←(A_7),(CY)←(A_7)	•••	07
RRC	(A_n)←(A_{n-1}),(A_7)←(A_0),(CY)←(A_0)	•••	0F
RAL	(A_{n+1})←(A_n),(A_0)←(CY),(CY)←(A_7)	•••	17
RAR	(A_n)←(A_{n+1}),(A_7)←(CY),(CY)←(A_0)	•••	1F
NOP	No operation		00
HLT	Processor stopped until interrupt or reset		76
DI	Interrupts disabled		F3
EI	Interrupts enabled after next instruction		FB
XTHL	(L)←((SP)),(H)←((SP)+1)		E3
SPHL	(SP_h)←(H),(SP_l)←(L)		F9
XCHG	(H)←(D),(L)←(E)		EB
DAA	Decimal adjust accumulator	•	27
CMA	(A)←$\overline{(A)}$		2F
STC	(CY)←1	•••	37
CMC	(CY)←$\overline{(CY)}$	•••	3F
OUT port	Not used in DCE Systems		D3 port
IN port			DB port

00	NOP	
01	LXI	B,addr
02	STAX	B
03	INX	B
04	INR	B
05	DCR	B
06	MVI	B,dd
07	RLC	
08	---	
09	DAD	B
0A	LDAX	B
0B	DCX	B
0C	INR	C
0D	DCR	C
0E	MVI	C,dd
0F	RRC	
10	---	
11	LXI	D,addr
12	STAX	D
13	INX	D
14	INR	D
15	DCR	D
16	MVI	D,dd
17	RAL	
18	---	
19	DAD	D
1A	LDAX	D
1B	DCX	D
1C	INR	E
1D	DCR	E
1E	MVI	E,dd
1F	RAR	
20	---	
21	LXI	H,addr
22	SHLD	addr
23	INX	H
24	INR	H
25	DCR	H
26	MVI	H,dd
27	DAA	
28	---	
29	DAD	H
2A	LHLD	addr
2B	DCX	H
2C	INR	L
2D	DCR	L
2E	MVI	L,dd
2F	CMA	
30	---	
31	LXI	SP,addr
32	STA	addr
33	INX	SP
34	INR	M
35	DCR	M
36	MVI	M,dd
37	STC	
38	---	
39	DAD	SP
3A	LDA	addr
3B	DCX	SP
3C	INR	A
3D	DCR	A

3E	MVI	A,dd
3F	CMC	
40	MOV	B,B
41	MOV	B,C
42	MOV	B,D
43	MOV	B,E
44	MOV	B,H
45	MOV	B,L
46	MOV	B,M
47	MOV	B,A
48	MOV	C,B
49	MOV	C,C
4A	MOV	C,D
4B	MOV	C,E
4C	MOV	C,H
4D	MOV	C,L
4E	MOV	C,M
4F	MOV	C,A
50	MOV	D,B
51	MOV	D,C
52	MOV	D,D
53	MOV	D,E
54	MOV	D,H
55	MOV	D,L
56	MOV	D,M
57	MOV	D,A
58	MOV	E,B
59	MOV	E,C
5A	MOV	E,D
5B	MOV	E,E
5C	MOV	E,H
5D	MOV	E,L
5E	MOV	E,M
5F	MOV	E,A
60	MOV	H,B
61	MOV	H,C
62	MOV	H,D
63	MOV	H,E
64	MOV	H,H
65	MOV	H,L
66	MOV	H,M
67	MOV	H,A
68	MOV	L,B
69	MOV	L,C
6A	MOV	L,D
6B	MOV	L,E
6C	MOV	L,H
6D	MOV	L,L
6E	MOV	L,M
6F	MOV	L,A
70	MOV	M,B
71	MOV	M,C
72	MOV	M,D
73	MOV	M,E
74	MOV	M,H
75	MOV	M,L
76	HLT	
77	MOV	M,A
78	MOV	A,B
79	MOV	A,C
7A	MOV	A,D
7B	MOV	A,E

7C	MOV	A,H
7D	MOV	A,L
7E	MOV	A,M
7F	MOV	A,A
80	ADD	B
81	ADD	C
82	ADD	D
83	ADD	E
84	ADD	H
85	ADD	L
86	ADD	M
87	ADD	A
88	ADC	B
89	ADC	C
8A	ADC	D
8B	ADC	E
8C	ADC	H
8D	ADC	L
8E	ADC	M
8F	ADC	A
90	SUB	B
91	SUB	C
92	SUB	D
93	SUB	E
94	SUB	H
95	SUB	L
96	SUB	M
97	SUB	A
98	SBB	B
99	SBB	C
9A	SBB	D
9B	SBB	E
9C	SBB	H
9D	SBB	L
9E	SBB	M
9F	SBB	A
A0	ANA	B
A1	ANA	C
A2	ANA	D
A3	ANA	E
A4	ANA	H
A5	ANA	L
A6	ANA	M
A7	ANA	A
A8	XRA	B
A9	XRA	C
AA	XRA	D
AB	XRA	E
AC	XRA	H
AD	XRA	L
AE	XRA	M
AF	XRA	A
B0	ORA	B
B1	ORA	C
B2	ORA	D
B3	ORA	E
B4	ORA	H
B5	ORA	L
B6	ORA	M
B7	ORA	A
B8	CMP	B
B9	CMP	C

BA	CMP	D
BB	CMP	E
BC	CMP	H
BD	CMP	L
BE	CMP	M
BF	CMP	A
C0	RNZ	
C1	POP	B
C2	JNZ	addr
C3	JMP	addr
C4	CNZ	addr
C5	PUSH	B
C6	ADI	dd
C7	RST	0
C8	RZ	
C9	RET	
CA	JZ	
CB	---	
CC	CZ	addr
CD	CALL	addr
CE	ACI	addr
CF	RST	1
D0	RNC	
D1	POP	D

D2	JNC	addr
D3	OUT	dd
D4	CNC	addr
D5	PUSH	D
D6	SUI	dd
D7	RST	2
D8	RC	
D9	---	
DA	JC	addr
DB	IN	dd
DC	CC	addr
DD	---	
DE	SBI	dd
DF	RST	3
E0	RPO	
E1	POP	H
E2	JPO	addr
E3	XTHL	
E4	CPO	addr
E5	PUSH	H
E6	ANI	dd
E7	RST	4
E8	RPE	
E9	PCHL	

EA	JPE	addr
EB	XCHG	
EC	CPE	addr
ED	---	
EE	XRI	dd
EF	RST	5
F0	RP	
F1	POP	PSW
F2	JP	addr
F3	DI	
F4	CP	addr
F5	PUSH	PSW
F6	ORI	dd
F7	RST	6
F8	RM	
F9	SPHL	
FA	JM	addr
FB	EI	
FC	CM	addr
FD	---	
FE	CPI	dd
FF	RST	7

ASCII - HEX - ASCII CONVERSION TABLE

LSD	MSD	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	↑	n	~
F	1111	SI	US	/	?	O	←	o	DEL

CHARGEMENT DU PROGRAMME:

```
UT
Z3
R     ( Positionnez la bande au début, le programme trouvé le
        curseur s'arrête de clignoter, patientez, le curseur
        clignote alors tapez: )
GII00
      ( Une question buferdim dec ?: répondre un chiffre entre
        0 et 16 suivant l'importance du programme )
Exemple tapez : 5
```

EXEMPLE DE PROGRAMME:

Tapez, en respectant les espaces. Sauf ce qui est entre parenthèses

```
A.
  ORG :3000      Implantation du programme(à partir de l'adresse
                                                  3000 (HEX))
  PUSH H         Sauvegarde des registres
  PUSH B              "        "       "
  PUSH D              "        "       "
  PUSH PSW            "        "       "
  MVI B,2        Initialisation à 2(du registre qui servira de
                                                   compteur)
  COMPT MVI A,79 O= CHR$(79)
  CALL :D695     Subroutine d'affichage
  MVI A,75       K= CHR$(75)
  CALL :D695
  DCR B          Décrémentation du compteur
  JNZ COMPT      Saut conditionnel  (si B=0)
  POP PSW        Restitution des registres(premier entré dernier
  POP D               "        "       "                  sorti)
  POP B               "        "       "
  POP H               "        "       "
  RET            Retour(au programme qui a appellé)
  END
  .              ( Ce point permet le retour à l'assembleur )
```

NOTE: UN PROGRAMME SOURCE COMMENCE TOUJOURS PAR ' ORG ' ET
FINI TOUJOURS PAR ' END '

RESUME DES COMMANDES

```
/.      Initialise le buffer détruit le programme
A.      Rentre du texte à partir de la ligne I(en décalant le reste
An.     Rentre du texte entre la ligne n et la ligne n+1
AF.     Rentre du texte à partir de la fin.
Ln.     Liste la ligne n
Ln M.   Liste de la ligne n à la ligne M
Ln F.   Liste de la ligne n à la fin

Kn.     Détruit la ligne n
Kn M.   Détruit de la ligne n à la ligne M
Kn F.   Détruit de la ligne n à la fin.
```

En. Edite la ligne n
En M. Edite de la ligne n à la ligne M
En F. Edite de la ligne n à la fin

Wn. Sauvegarde la ligne n sur cassette
Wn M. sauvegarde de la ligne n à la ligne M
Wn F. sauvegarde de la ligne n à la fin

R. Lecture d'une source sur cassette en l'ajoutant au source déja en mémoire
H. Met l'imprimante en marche
u. Retour à l'utilitaire
B. Retour au basic
#E. Cherche les erreurs
#L. Liste avec adresses et codes
#P. Assemble et met en mémoire une source
#O. Assemble et sauvegarde (codes)

POUR EXECUTER NOTRE PETIT PROGAMME TAPEZ #P. espce B. CALLM #3ϕϕϕ

D E S A S S E M B L E U R 8 0 8 0 A.

Programme écrit en BASIC. Faites LOAD et RUN.
STARTADRESSE ? Répondre ici l'adresse de début du désassemblage, mettre 2 points (:) si c'est en hexadecimal.
Exemple :CFFF

L E S C O M M A N D E S

La barre d'espacement : Désassemble une ligne en mémonique.

" S " : Permet de donner un label à une adresse. EXEMPLE tapez " S ". Répondre à "SYMBOL ?" par cinq lettres maximum de votre choix puis mettez l'adresse à laquelle vous voulez qu'elles soient attribuées. Sans oubier les 2 points si c'est en hexadécimal.

" T " : Considère le code en mémoire comme code ASCII. En donne son expression.

" C " : Permet de sauvegarder les labels. Permet de lire des labels sur cassette.

" L " : Table des symboles.

" ↑ " : Retour en arrière.

A S S E M B L E U R DE D A I N A M I C (dna)

EDITEUR RESIDENT - ASSEMBLEUR - CHARGEUR (loader) pour le langage

L'objet de ce programme est de créer, editer et de traduire le langage symbolique du 8080A en langage machine.

Sur la cassette, vous trouverez :

1) L'assembleur
2) Un chargeur en BASIC (loader) : utilisé pour charger le programme machine et le mettre sur les adresses normallement occupé par l'assembleur.
3) Un désassembleur.

DESCRIPTION DE LA MEMOIRE DU DAI POUR L'ASSEMBLEUR.

ADRESSE EN HEXADECIMAL	-	SIGNIFICATION
De ØØØØ à Ø2EB		cf. le manuel du DAI
Ø2EC 1ØFF		espace libre pour les pro--gramme en langage machine
11ØØ 2FFF		Assembleur DNA
3ØØØ 3ØØØ + DIM - 1		entrée/sortie et tampon edit
3ØØØ 3ØØØ + 2 X DIM -1		Tampon source (1)
3ØØØ + 2 X DIM B34F		Place pour les prog BASIC

DIM est la taille des tampons (mémoire). Quand le DNA a étè lancé avec " G11ØØ ", vous pouvez choisir la dimension des tampons. 1 unité équivaut à 1K octet.

La dimension maximale pour un DAI de 48K est de 16 unités.
IMPORTANT : Si votre programme machine n'entre pas dans l'espace réservé " Ø2EC-1ØFF ", vous devrez utiliser l'ASSEMBLEUR pour créer le programme machine, et le sauver sur cassette, puis le charger avec le LOADER pour le mettre en RAM.

(1) Le tampon source est l'espace mémoire qui est réservé pour votre programme.
C'est vous qui choisissez la taille des tampons lorque vous lancez l'assembleur.

DNA - UTILITY - BASIC

Pour passer du Basic à l'assembleur, tapez " CALLM 12ØØØ ".
Pour passer de l'utility à l'assembleur, tapez " G11ØØ ".
Pour passer de L'ASSEMBLEUR au BASIC, tapez " B. ".
Pour passer de L'ASSEMBLEUR à L'UTILITY, tapez, " U. ".

NOTE : Pour savoir si vous êtes en BASIC, en UTILITY, ou ASSEMBLEUR, un caractère - guide sera affiché :


Indiquera que vous vous trouvez dans le " Basic "
" " " " " " " " " L'utility "
" " " " " " " " " L'assembleur DNA'

CHARGEMENT ET MISE EN ROUTE DU DNA.

1) Allumer le DAI, appuyer sur la touche " ESPACE ".
2) UT
3) Z3 -initialisation.
4) R -lecture sur cassette du DNA.
5) G11ØØ -mise en oeuvre de l'assembleur.

SORTIE SUR IMPRIMANTE (hardcopy flag)

H. Mise en route ou arrêt de l'imprimante par l'interface RS 232.
N.B. : RS232 est adapté pour les imprimantes EPSON TX30 ou MX30.

PROCEDURE SUR CASSETTE.

Les programmes peuvent être lus ou écrits sur cassette avec ou sans nom.

N.B. : Les programmes en langages assembleur ont les mêmes caracté-ristiques que les programmes de L'UTILITY : alors ATTENTION, car charger un programme UTILITY lorsque l'assembleur est en mémoire peut LE DETRUIRE.

MODIFICATION DE L'ASSEMBLEUR DNA POUR UTILISER UN AUTRE TYPE DE PROG.

Changer dans L'UTILITY :

```
2EED : 21   F9   2E
2E5Ø : C3   F3   2E
2EF9 : 3E   23   C3   F2   EE   Ø6   23   C3   19   EF
```

Après cette modification, le caractère "#" est utilisé pour les programmes (sources) en langages assembleur.

I. EDITEUR ASSEMBLEUR 8080A

L'éditeur du DNA permet grâce à ses commandes d'écrire un programme source.
On peut :

1) Ecrire ou ajouter des lignes.
2) Supprimer des lignes.
3) Lister le programme source.
4) Ecrire un programme source sur cassette.
5) Lire un programme source d'une cassette.
6) Se servir, avec une commande spéciale, du mode EDIT, qui possède les mêmes caractéristiques que L'EDIT du BASIC.

DESCRIPTION GENERALE DES COMMANDES DE L'EDITEUR

N.B. : Il est à noter que le caractère "]" qui se trouve dans les exemples n'est pas écrit par l'utilisateur mais est affiché par l'ordinateur.

COMMANDE	SIGNIFICATION
/.	Cette commande a pour effet d'effacer le tampon source, c'est à dire que tous les programmes sources se trouvant en mémoire sont détruit. Le " . " est l'équivalent de " RETURN " en BASIC Il termine chaque commande. Si à la place du point, vous tapez un autre caractère, un point d'interrogation apparaitra sur l'écran, et l'assembleur attendra une nouvelle commande. Ainsi, si vous vous trompez de commande, il suffira d'appuyer sur un caractère inconnu de l'assembleur (exemple : X) et celui-ci attendra une autre commande et vous pourrez rectier votre erreur.

ECRIRE OU AJOUTER DES LIGNES.

A. An. AF.
A.

Lorsque vous tapez cette commande, vous passez en mode insertion bufferisé permettant l'inser--tion de lignes.
Chaque ligne est terminée par un " RETURN " pour arrêter l'insertion, tapez le caractère "."

Ne mettez pas de numéro de ligne, l'assembleur s'en charge, et elles apparaitront automatiquement lors du listing du programme source (commande L).

EXEMPLE :

] A. Debut EQU ØØØØ	on passe en mode insertion on écrit les lignes terminées par un RETURN
MOV A,B	RETURN
JMP LABEL	RETURN
.	on sort du mode insertion
]	l'assembleur attend une nouvelle commande.

COMMANDE — SIGNIFICATION

AJOUT DE LIGNES

An.
AF.

Pour ajouter des lignes il suffit d'indiquer après le A le numéro de ligne n à partir duquel on veut insérer des lignes.

" AF. " signifie que l'insertion commencera après la derniére ligne du programme.

EXEMPLE :

```
]L1  F.                    On list le programme que l'on
                           vient d'entrer précédement.
ØØØØ    DEBUT   EQU   ØØØØ
ØØØ1            MOV   A,B  affichage du listing
ØØØ2            JMP   LABEL
]
```

Après avoir listé le programme dont les N° de ligne servira de repère, on insére une ligne PUSH PSW et l'on veut qu'elle se trouve après MOV A,B c'est à dire après la ligne 1. on fera donc :

```
]A1                1'ordinateur passe en mode inserti
                   après la ligne 1
PUSH PSW ↵         nous écrivons l'instruction,
                   terminée  par RETURN
.                  on tape un point, pour repasser
]                  dans l'éditeur.
```

Puis nous listons :

```
]L1 F.
ØØØØ    DEBUT   EQU   ØØØØ
ØØØ1            MOV   A,B
ØØØ2            PUSH  PSW
ØØØ3            JMP   LABEL
]
```

La ligne est bien insérée.

COMMANDE	SIGNIFICATION
REMARQUES:	A. EST EQUIVALLENT A AO. Si vous uti--lisez cette commande après avoir déjà tapé un bout de programmme, l'insertion commencera avant la ligne 1.
Exemple :	]A. DEB PUSH B]. Après cela, la ligne 1 deviendra la ligne 2, la ligne 2 deviendra la ligne 3 etc... Et la ligne que l'on vient de renter deviendra la ligne 1.
	Pour effacer le dernier caractère entré, tapez sur le curseur " ← ".
	LE NUMERO DE LIGNE n DOIT ÊTRE UN ENTIER
	Le point redonne la main à l'assembleur et doit être écrit en début d'une nouvelle ligne.
LISTER LE PROGRAMME SOURCE	
Ln.	cette commande liste la ligne n.
Ln m.	Ici, la commande listera le programme de la ligne n à la ligne m.
Ln F.	liste le programme à partir de la ligne
LF.	liste la dernière ligne du programme.
SUPPRIMER DES LIGNES	
Kn.	Efface la ligne n
Kn m.	supprime de la ligne n à la ligne m
Kn F.	Efface le programme à partir de la ligne
KF.	Efface la dernière ligne.
REMARQUE	Les lignes supprimées font place à celle qui suivent.
EXEMPLE :	]K2. Ici, vous supprimez la ligne 2] Et la ligne 3 deviendra la ligne 2, la ligne 4 la 3, etc...

COMMANDE	SIGNIFICATION

MODE EDIT

En.	Edite la première ligne
En m.	edite de la ligne n à la ligne m
En F.	edite à partir de la ligne n.
EF.	edite la dernière ligne.

REMARQUE :

-Les lignes doivent suivre le même format que la commande A (voir dans le chap II, 'format d'une ligne source et rappels ' page 12).

-Un point indique que toutes les lignes se trouvant derrière lui seront supprimé

-Utilisez le symbole " & " pour les valeurs octales. Nous vous conseillons, si votre programme est long, de l'éditer en plusieurs parties.

SAUVER UN PROGRAMME SOURCE SUR CASSETTE

Wn.	Sauve sur cassette la ligne n
Wn m.	SAUVE DE la ligne n à la ligne m
Wn F.	Sauve à partir de la ligne n.
WF.	Sauve la dernière ligne.

ATTENTION, cette commande ne sert que pour sauver des programmes sources et elle ne doit pas être utilisée pour les programme objets.

Après avoir tapé le point, l'ordinateur attend que vous appuyez sur la barre d'espacement pour commencer à enregistre

LIRE UN PROGRAMME SOURCE D'UNE CASSETTE

R. Cette commande lit un programme source d'une cassette .

REMARQUES :

L'ordinateur commence à lire le programm dès que le point est tapé. Donc, avant de l'écrire, mettre le magnétophone en position de lecture.

COMMANDE - MESSAGE | SIGNIFICATION

-Cette commande ne lit que des programmes sources. Pour lire les programmes objets, vous devez vous servir du 2 èmes programme de la cassette assembleur : Le LOADER en BASIC.

M E S S A G E D ' E R R E U R S

" BUFFER FULL " : Ce message indique qu'il n'y a plus assez d'espace mémoire pour continuer votre programme. Solution : Tout d'abord supprimer la dernière ligne que vous venez d'enter. puis ajouter cette ligne spéciale :

"RETURN" puis "!" "RETURN". Esuite, sauver votre programme sur casse Si jamais vous ne tapez pas cette ligne spéciale, vous ne pourrez sauver le programme sur cassette avec la commande " Wl F. ". Cette dernière ligne indique à l'assembleur que le programme n'est pa terminé. Après, vous avez le choix entre deux solutions :

1) Réservez plus d'espace mémoire lors du lancement de l'assem
2) Ou alors, découper votre programme en plusieurs parties.

" IO BUFFER FUL ": L'assembleur ne peut sauver votre progra me d'un seul trait.

EXEMPLE : Votre programme contient 250 lignes et lorque vous voulez le sauvegarder, le message " IO BUFFER FULL " apparait. ne vous inquietez pas, il vous suffit simplement de sauver votre programme en plusieurs parties. comme ceci :

Wl 100 puis
Wl01 250.

ATTENTION : Notez précieusement la taill mémoire que vous avez utilisé pour votre programme car vous le changerez, vous devrez prendre au minimum la même DIMension pour pouvoir le charger.

II . L E L A N G A G E A S S E M B L E U R

Dans ce chapitre, nous verrons comment est constituée une ligne source. Puis, nous passerons aux pseudo-instructions et aux directives d'assemblages que possède le DNA.
C'est plus tard, dans le chapitre III, que nous regarderons les commandes d'assemblage.

STRUCTURE D'UN PROGRAMME SOURCE

Le programme source doit satisfaire à certaines contraintes pour que le DNA puisse en faire une analyse précise.
Une ligne est decoupée en quatre paries : de la gauche vers la droite. Elle est disposée comme suit :

zone étiquette
zone code opération
zone opérante
zone commentaire

1) ZONE ETIQUETTE

Une étiquette est le nom d'une adresse mémoire. Elle commence obligatoirement par une lettre (A-Z) et se termine par des chiffres ou des caractères. Sa longueur ne doit pas dépasser six caractères. Sur le DNA, vous avez le droit d'utiliser les caractères spéciaux du clavier pour une étiquette exepté le signe plus, le signe moins le blanc et la virgule.
EXEMPLES : JKL(), A'!89 peuvent être des noms d'étiquettes !
Une étiquette est facultative. (dans ce cas, commencez la ligne par un blanc).
Elle est terminée par un espace.

LES ETIQUETTES SONT UTILISEES DANS LES CAS SUIVANTS :

1) Pour répérer une instruction qui est la destination de :
   a) Une des instructions de branchement conditionnel : CC, CM, CNC, CNZ, CP, CPE, CPO, CZ, JC, JM, JNC, JNZ, JPE, JPO, JZ, et les pseudo-instructions du DNA de conditions(voir page 15).

   b) Une des instructions inconditionnelles de branchement : JMP, PCHL

   c) L'instruction de branchement à un sous-programme CALL. De ce fait, la zone opérande de l'instruction de bran--chement contiendra uniquement le symbole de l'étiquette Dans le cas de l'instruction CALL, ce symbole peut être considérè comme le nom d'un sous-programme.

II . LE LANGAGE ASSEMBLEUR

2) Pour repérer une instruction qui est la destination de saut utilisée dans le mode d'adressage étendu :

EXEMPLE : JMP saut inconditionnel

Donc, la zone opérande de l'instruction " SAUT " utilisée dans le mode d'adressage étendu, contiendra le symbole de l'étiquette.

3) Pour repérer une zone mémoire indiquée par une des directives d'assemblage :

ASC, DATA, RES, DBL

Les instructions faisant référence à cette zone mémoire contientdront dans leur zone opérande le symbole de l'étiquette.

4) Une étiquette doit être utilisée dans chaque instruction source contenant la directive d'assemblage EQU. l'étiquette sera identique au symbole défini par la directive EQU.
N.B. : Dans le DNA, vous pouvez utiliser comme nom d'étiquette, le nom des instructions, pseudo-instructions, directives d'assemblage, et des registres du 8080. Mais alors, ATTENTION A LA CONFUSION que cela pourrait entraîner!

2) ZONE CODE OPERATION

Les opérateurs mnémoniques reconnus par l'assembleur DNA comprennent les 79 instructions exécutables du 8080A (qui sont traduites par l'assembleur en code machine), les pseudo-instructions (voir page 14) et les directives d'assemblage du DNA (voir page 17).

EXEMPLE : MOV, etc.

3) ZONE OPERANDE

Les informations situées dans la zone opérande sont transmises à un système opérateur mnémonique particulier. pour les 79 instructions exécutables, le microprocesseur utilise différents mode d'adressages. Certains mode d'adressage ont besoin d'une opérande pour travailler. Il y a l'adressage par registre, l'adressage immédiat et l'adressage indirect.

EXEMPLE :	LDA	:FF	adressage immédiat
	SUB	H	adressage par registre
	MOV	A,M	adressage indirect

Ce qui est souligné en rouge s'écrit dans la zone opérande.

L'assembleur reconnaît les nombres, les symboles et les expressions situées dans le champ opérande.

II . LE LANGAGE ASSEMBLEUR

NOMBRES

L'assembleur accepte les nombres dans les formats suivants :

- nombre (décimal : base 10)
- :nombre (hexadécimal : base 16)
- @nombre (octal : base 8)

Un préfixe : ou @ indique à l'assembleur que le nombre suivant le préfixe est un nombre hexadécimale ou octal. Un nombre sans suffixe signifie qu'il est décimal. Un caractère peut être ecrit dans les bases que nous venons de voir ou tout simplemen être écris entre côtes.

EXEMPLE : A peut être écrit sous la forme hexadécimal : :C1 ou entre côtes : 'A'

SYMBOLES

Un symbole est constitué de 1 à 6 caractères (y compris les caractères spéciaux, sauf le signe plus, le signe moins, la virgule et le blanc). Le premier caractère du symbole doit être obligatoirement une lettre (A-Z). Tous les symboles doivent être définis dans le programme source, excepté bien entendu le nom des resgistres du 8080A. Il y a trois manières de définir un symbole :

1) Une instruction exécutable du programme source peut avoir une étiquette identique au symbole. La valeur du symbole est alors la l'adresse numérique correspondant à l'instruction.

2) Une des directives d'assemblage DATA, ASC, RES ou DBL peut être écrite avec une étiquette identique au symbole. la valeur du symbole est alors l'adresse numérique du premier octet du code machine génèré par l'assembleur pour la directive comprenant l'étiquette.

3) Le symbole peut être défini par la directive d'assemblage EQU. La directive EQU est précédée d'une étiquette identique au symbole.
La valeur du symbole est celle de l'opérande qui suit la directive EQU. cette opérande peut être un nombre, un autre symbole ou une expression.

II . LE LANGAGE ASSEMBLEUR

EXPRESSIONS

Une expression est une combinaison de symboles et/ou de nombre séparés par les opérateurs arithmétques + et -.

L'assembleur évalue les expressions algébriquement de la gauche vers la droite sans parenthèsage et sans priorité sur les opérateurs. un résultat fractionnaire, total ou partiel, sera arrondi à une valeur entière.

C'EST L'ASSEMBLEUR QUI EVALUE CES EXPRESSIONS ET NON LE MICROP

4) ZONE COMMENTAIRE

Les commentaires sont facultatifs. Ils peuvent contenir tous les caractètres du clavier excepté le retour du chariot(RETURN Les commentaires sont utilisés pour faciliter la compréhension et la mise au point d'un programme.

FORMAT D'UNE LIGNE SOURCE ET RAPPELS

ETIQUETTE-INSTRUCTION-OPERANDE-COMMENTAIRE
Le " - " indique un espace.

1) Une étiquette est facultative. Elle commence directement au début de la ligne : IL N'Y A PAS DE BLANC. Elle se cloture par un espace.

2) L'instruction est un mnémonique de trois ou quatre lettres du 8080A. ATTENTION : L'instruction n'est pas l'opérantde EXEMPLE : MOV A,B ici, l'instruction est MOV et l'opérande est A,B. une instruction peut être aussi une pseudo-instruction ou une directive d'assemblage.
Elle se termine par un blanc. Si la ligne n'a pas d'étiquette, COMMENCEZ LA LIGNE PAR UN BLANC.

3) L'opérande est soit le nom d'un registre (ex: H). un nombre, une expression ou un symbole.
ATTENTION : L'OPERANDE NE DOIT PAS CONTENIR D'ESPCE.

EXEMPLE :	MOV A, B	écriture incorrecte
	MOV A,B	écriture correcte
	MOV A,LABEL - 2 + TT	écriture incorrecte
	MOV A,LABEL-2+TT	écriture correcte

L'opérande se termine par un blanc.

4) Le commentaire est facultatif. Il ne doit pas dépasser 27 caractères.

5) FIN DE LIGNE

Lorsque votre ligne est terminée, tapez sur la touche RETURN comme pour le BASIC.

II . L E L A N G A G E A S S E M B L E U R

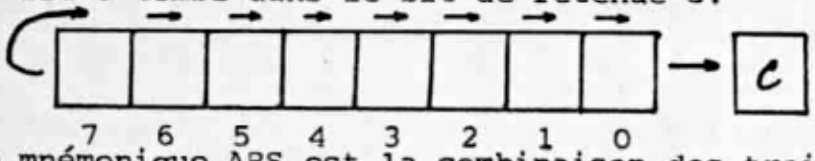
LES PSEUDO-INSTRUCTIONS DU DNA

Si vous commencez à apprendre la programmation du langage assembleur, vous pouvez sauter cette partie qui traite des speudo-instructions du DNA et qui sont surtout utiles pour les programmeurs avancés.

NOM DES INSTRUCTIONS SPECIALES DU DNA

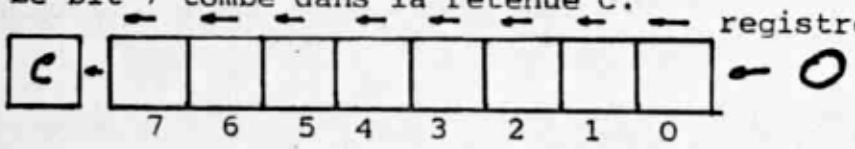
ARS Décalage arithmétique droit sur le registre A.
(Arithmétic right shift regiter A)

Décale le contenu du registre A d'un bite vers la droite en répètant le bit de signe (bit 7).
Le bit 0 tombe dans le bit de retenue C.



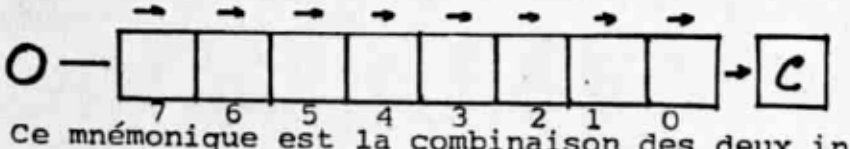
Le mnémonique ARS est la combinaison des trois insructuons : PLC, RAR, RAR.

LLA Décalage logique à gauche. (Logical left shift, register A).
Décale le contenu de A d'un bit vers la gauche.
Il entre un zero sur la droite.
Le bit 7 tombe dans la retenue C.



Ce mnémonique est le synonyme de l'instruction ADD A.

LRA Décale logique à droite. (logical right shift register A).
Décale d'un cran vers la droite les bits du registr A. Un zéro est mis dans le bit 7 et le bit 0 est transféré dans la retenue.



Ce mnémonique est la combinaison des deux instructions ORA A et RAR.

II . L E L A N G A G E A S S E M B L E U R

RSC (Reset Carry)
On met à zéro le bit de retenue C.
Ce mnémonique est le synonyme de l'instruction : ORA A.

STOP (ENABLE Iterrup et halt)
Combinaison des deux instructions d'interruptions : EI et HLT.

TCA Complément à deux registre A (twos complement, register A)
Ce qui revient à prendre l'opposé du registre A.
Ce mnémonique est la combinaison des instructions : CMA et INR A.

TRP TRAP
synonyme de RST.

WAIT Equivalent à STOP

ZAR (Zero register A)
Met à zéro le contenu du registre A.
Ce mnémonique est le symbole de XRA A.

PSEUDO-INSTRUCTIONS FACILITANT L'EMPLOI DES INSTRUCTIONS CONDITIONNELLES

Les instructions suivantes sont complétées par "J" (JUMP), "C" (CALL) ou "R" (RETURN). tous ces mnémoniques sont utilisés après l'instruction : CMP ou CPI.

EQ Si égale. (if equal).
EXEMPLE : CEQ signifie : appel d'un sous-programme si le registre A est égale à l'opérande comparée.

NE Si non égale (if not equal).
EXEMPLE : RNE : Retour d'un sous-programme si ce n'est pas égale.
(équivallent à l'instruction NZ)

LES DEUX INSTRUCTIONS QUI SUIVENT NE TIENNENT PAS COMPTE DU BIT DE SIGNE

LS Si A est plus petit.(if A regiter less)
EXEMPLE : JLS :Saut si A est plus petit(equiv à JC)

GE Si A est plus grand ou égale.(if A register CREATER or EQUAL to)

LES DEUX INSTRUCTIONS SUIVANTES TIENNENT COMPTE DU BIT DE SIGNE

< (if A register more negative)
Si le registre A est "PLUS" négatif que l'opérnde comparée précédemment(equivallent à l'instruction : RM)

>= (if A register MORE POSITIVE THAN or EQUAL)
Si le registre A est " PLUS " positif que l'opérande comparée.

EXEMPLE : -2 est plus positif que -3

EXEMPLE :

CEQ : Appel d'un sous-programme si A est égale à l'opérande comparée.

J>= : Saut à une adresse précisèe si A (signé) est " PLUS " positif que l'opérande signée.

RGE : Retour d'un sous-programme si A est plus grand ou égal(sur 8 bits) que l'opérande comparée.

II L E L A N G A G E A S S E M B L E U R

LES DIRECTIVES D'ASSEMBLAGES

N.B. : On peut ajouter ou soustraire des opérandes ou des constantes directement sous forme symbolique : EX: LDA OP + 1

Les directives d'assemblage sont des ordres donnés à l'assembleur par l'intermédiaire de codes écrits dans le programme source.

LISTE DES DIFFERENTES DIRECTIVES DU DNA

ORG : définit l'origine d'une adresse
END : indique la fin du programme source
EQU : assigne une valeur à un symbole
LST - UNL: autorise ou interdit l'édition du listing du programme source
DATA : crée une table de données
DBL : Définition d'une constante de deux octets
ASC : Définition d'une chaine de caractères
RES : Réservation d'octets
XXX : équivant à l'instruction NOP
TITL : spécifie le nom du programme source
REM : remarque (comme dans le BASIC)
* : ligne de commentaires étendus (52 caractères)

DESCRIPTION

N.B. : n et m sont des octets de 8 bits.
Toutes les directives s'écrivent dans la zone code (elles sont donc précédées d'un blanc) excepté la directive " * ".

ORG nm : Cette directive définit l'adresse où elle se trouve.
EXEMPLE : 0012 ORG AFFF

Lors de l'assemblage le numéro de ligne 0012 deviendra la ligne AFFF. bien entendu, toutes les lignes après 0012 partiront après l'adresse AFFF. C'est à dire que 0013 deviendra B000, 0014 deviendra B001, etc...

END : Cette directive signale à l'assembleur la fin du programme source.

EQU : Cette directive est utilisée pour donner une valeur à un symbole. Elle doit être précédée d'une étiquette. La zone opérante doit contenir la valeur numérique du symbole, ou tout autre symbole pouvant être évalué par l'assembleur.

Exemple :
```
BETTER EQU :OFFF   (ou :FFF)
MORE EQU BETTER
XXX EQU MORE-BETTER+2
...
```

ATTENTION : Une même étiquette ne peut être déclarée plusieurs fois.

EXEMPLE :
```
FOIS EQU FFFF      Cette deuxième déclaration provo-
FOIS EQU AF        quera un message d'erreur lors de
                   l'assemblage.
```

LST UNL : Ces directives autorisent ou interdisent selon le choix du programmeur l'édition du listing.
UNL : signifie: ne plus lister
LST : signifie: reprendre le listing.

EXEMPLE :
```
UNL   à partir d'ici le listing sera défendu
LABEL MVI A,B
ADD A
LST   Le listing reprendra à partir d'ici.
```

DATA n,m,... :
Cette directive dit à l'assembleur que les expressions n,m,... devront être rangées consécutivement à partir de l'adresse de la directive DATA.

Les expressions entre virgules (n,m,...) doivent être des octets de 8 bits. C'est à dire que si une valeur est plus grande qu'un octet (EX : DATA 300, 300 étant plus grand que 255, nombre limite d'un octet), un message d'erreur sera formulé lors de l'assemblage.

EXEMPLES :
```
DATA  :8A,'H',4
LABEL DATA 'B'
```

II . LE LANGAGE ASSEMBLEUR

DBL nn : Cette directive est utilisée pour ranger en mémoire la donnée nn sur deux octets, l'octet bas un premier.

EXEMPLE : DBL :AOBF

Lors de l'assemblage, l'assembleur rangera en mémoire BF puis AO.
Cette directive correspond à DW sur d'autre assembleu

ASC'XXXX'Traduit les caractères entre côtes et les ranges sur des octets consécutif en mémoire. la chaine de caractères entre côtes ne doivent pas dépasser 27 caractères.

EXEMPLE : MES1 ASC 'stop'

Les octets correspondants aux caractères 'STOP' seron rangés à partir de l'adresse mémoire MES1 lors de l'assemblage.

RES n,m: Réserve un nombre d'octets consécutifs d'une longueur de n. " ,m " est facultatif. " m " est un octet de 8 bits. Cela signifie que les n octets réservés à seront remplis par le nombre m. m peut être un symbol ou une expression mais on ne peut pas dépasser un octet.

EXEMPLE : RES 6-FOIS

Lors de l'assemblage, l'assembleur réservera à partir de l'adresse de RES, 6-FOIS octets consécutifs.

EXEMPLE : RES 6,'A'

Lors de l'assemblage, l'assembleur réservera 6 octets à partir de l'adresse où se trouve RES, mais CES 6 octets auront la valeur du caractère 'A'.

XXX : Ces trois astériques signifient qu'il faudra mêttre OO à l'adresse où ils se trouvent. XXX est équivalent à NOP.

II . LE LANGAGE ASSEMBLEUR

TITL : Cette directive indique à l'assembleur le nom devant être imprimé en tête de chaque page du listing. Il n'y a pas de génération de code objet mais TITL apparait dans le listing d'assemblage

EXEMPLE : TITL Programme d'addition

Dans le listing d'assemblage, on verra en tête :

PAGE 1 Programme d'addition
PAGE 2 , etc...

REM : REM est l'équivallent de REM en BASIC.
On s'en sert lorsque la ligne ne comporte pas d'instructions. on peut mettre 27 caractères après REM.
A notez que dans le listing d'assemblage, REM n'apparaitra pas sur l'écran et que le commentaire sera psitionné dans la zone commentaire.

EXEMPLE : REM commentaire

Lors du listing d'assemblage :

commentaire

SYMBOLE SPECIAL

* : l'astérisque seul indique que ce qui va suivre est un commentaire. Ce commentaire peut atteindre 52 caractères. Le commentaire commencera dès le debut de ligne dans le listing d'assemblage.

IMPORTANT: ON NE MET PAS D'ESPACE devant l'astérisque, il commence directement en début de ligne.

Special 8080 Codes

If you are just getting started with assembly-language programming you can skip this part, which is about some short-cut mnemonics useful to the more advanced programmer.

Combined operations, miscellaneous synonyms

ARS ARITHMETIC RIGHT SHIFT, REGISTER A. An arithmetic right shift involves copying the sign bit into the vacated most significant magnitude position, so a negative twos complement number will remain negative. The ARS mnemonic assembles as three bytes: RLC, RAR, RAR, which accomplish the required propagation of the sign bit.

LLA LOGICAL LEFT SHIFT, REGISTER A. This is a synonym for ADD A. Register A is shifted left one, and a zero is shifted into the least significant bit.

LRA LOGICAL RIGHT SHIFT, REGISTER A. This mnemonic generates two bytes of code: ORA A followed by RAR. The carry is cleared, and the A-register is rotated right one, so that a zero is moved into the most significant bit.

RSC RESET CARRY. A synonym for ORA A, which resets the carry flag.

STOP ENABLE INTERRUPT AND HALT. Assembles as EI, HLT.

TCA TWOS COMPLEMENT, REGISTER A. This instruction complements and increments register A, assembling as a two-byte instruction, CMA followed by INR A.

TRP TRAP. A synonym for restart (RST); requires an operand designating which restart address, as for RST.

WAIT Same as STOP.

ZAR ZERO A REGISTER. Synonym for XRA A.

Synonyms for conditional operations

The following one- or two-character modifiers are to be added to J (jump), C (call) or R (return) as appropriate, to make up a two- or three-character mnemonic. All of the following are intended to be used right after a CMP or CPI instruction.

EQ IF EQUAL. Equivalent to Z; CEQ means CZ: call if A-register was equal to the comparand word of the Compare instruction.

NE IF NOT EQUAL. RNE means return if not equal. Equivalent to NZ.

The next two mnemonics are meant to refer to *unsigned* eight-bit compares:

LS IF A-REGISTER LESS than comparand word of compare instruction preceding. Equivalent to C, as in JC.

GE IF (A-register) GREATER OR EQUAL TO 8-bit comparand word of compare instruction preceding. Equivalent to NC.

The next two mnemonics are meant to refer to *signed* (twos complement) compares:

< IF (A-register) MORE NEGATIVE THAN comparand word of compare instruction preceding. Equivalent to M, as in RM.

> = IF (A-register) MORE POSITIVE THAN OR EQUAL TO comparand word of compare instruction preceding. Equivalent to P, as in CP. Negative 2 is more positive than negative 3.

EXAMPLES: CEQ Call if A equal to comparand word.
J > = Jump if signed A more positive than or equal to signed comparand word.
RGE Return if A greater than or equal to positive 8-bit comparand word.

Error Messages during Assembly

Seven kinds of error are sensed and reported by NTA80 during assembly. An error message consists of one line which is the assembled listing, and a second in which the single letter designating the error is printed.

M

MULTIPLE DEFINITION. A symbol has been defined more than once. This happens most commonly from using the same label in two places in a long program — the programmer forgets it has already been used. Another way is to define a symbol once with an EQU statement, and then use the same symbol as a label. Of course you can also do the trick with two EQU statements having different operands but the same symbol on the left. Multiple definitions can also arise from using symbols which are longer than 6 characters, and share the first 6 characters (i.e., SYMBOLS1 and SYMBOLS2).

When a multiple definition occurs, the first definition is the one that sticks; the second is ignored.

R

REGISTER SPECIFICATION ERROR. An instruction that requires the symbol for an 8080 register — that is, a member of the set BCDEHLMA — contains something else. MVI Q,1 would result in an R message.

O

OPERATION INVALID. The assembler has found, in the instruction column, something that does not appear in its tables of valid instructions. As previously mentioned, if you forget to indent an op-code that is not preceded by a label, the operand will seem to be the op-code, and will probably generate an O message.

P

POST-DEFINED. On Pass 1, symbols in the operand fields of ORG, EQU, or RES pseudo-operation will cause a P error message if those symbols have not previously been defined.

U

UNDEFINED. U messages are generated during Pass 2. A symbol is defined in one of two ways: by being used as the label starting at the left margin of a normal program statement, or by being used as the left member of an EQU statement, the operand of which is a number or a previously-defined symbol. Symbols used as operands of statements other than ORG, EQU, or RES need not be defined the first time used, but must be defined before the END statement of the program.

L

LENGTH. A number that is required to be expressed in eight bits is too large; for example, in MVI A,300, the literal number must fit into the word following the code for MVI A, and the maximum permissible number in decimal is 255. There is no test for 16-bit numbers being too large; if two 16-bit numbers are added together, as can occur when a constant is added to a symbol in an operand field, the resulting number is expressed modulo 65536. It "wraps around."