Microprocesoare
Anexe pentru lucrările de laborator

Dan NICULA [1]
Alexandru PIUKOVICI Radu GĂVRUŞ

August, 1999

[1]Universitatea TRANSILVANIA Braşov, Catedra de Electronică şi Calculatoare, Email: nicula@vega.unitbv.ro

Instrucţiunile limbajului de asamblare Z80

Pentru operanzii instructiunilor s-au folosit următoarele simboluri:

N - un octet cu valoarea cuprinsă între 00H - 0FFH. S-a folosit N = 20.

NN - doi octeţi cu valoarea cuprinsă între 0000H - 0FFFFH. S-a folosit NN = AA.

IND - deplasament ce se adună la conţinutul unui registru index pentru obţinerea adresei locaţiei cu care lucrează instrucţiunea respectivă. S-a folosit IND = 05.

DIS - deplasament ce se adună la adresa de asamblare a instrucţiunii respective pentru a obţine adresa de salt. S-a ales DIS = 03.

ADC

- Adună doi operanzi împreună cu indicatorul de transport

(CF). Rezultatul este salvat în locaţia operandului din

stînga. Adresarea poate fi imediată, implicită, indirectă sau indexată.

8E	ADC	A, (HL)
DD 8E 05	ADC	A, (IX+IND)
FD 8E 05	ADC	A, (IY+IND)
8F	ADC	A, A
88	ADC	A, B
89	ADC	A, C
8A	ADC	A, D
8B	ADC	A, E
8C	ADC	A, H
8D	ADC	A, L
CE 20	ADC	A, N
ED 4A	ADC	HL, BC
ED 5A	ADC	HL, DE
ED 6A	ADC	HL, HL
ED 7A	ADC	HL, SP

ADD

- operandul din dreapta este adunat la operandul din stănga.

Operanzii pot fi pe 8 sau 16 biti iar adresarea poate fi implicită sau

imediată.

86	ADD	A, (HL)
DD 86 05	ADD	A, (IX+IND)
FD 8E 05	ADD	A, (IY+IND)
87	ADD	A, A
80	ADD	A, B
81	ADD	A, C
82	ADD	A, D
83	ADD	A, E
84	ADD	A, H
85	ADD	A, L
C6 20	ADD	A, N
09	ADD	HL, BC
19	ADD	HL, DE
29	ADD	HL, HL
39	ADD	HL, SP
DD 09	ADD	IX, BC
DD 19	ADD	IX, DE
DD 29	ADD	IX, IX
DD 39	ADD	IX, SP
FD 09	ADD	IY, BC
FD 19	ADD	IY, DE
FD 29	ADD	IY, IX
FD 39	ADD	IY, SP

AND

- se efectuează o funcţie ŞI logic între conţinutul

registrului A şi operand. Operandul este pe 8 biţi. Adresarea poate fi

imediată sau implicită.

A6	AND	(HL)
DD A6 05	AND	(IX+IND)
FD AE 05	AND	(IY+IND)
A7	AND	A
A0	AND	B
A1	AND	C
A2	AND	D
A3	AND	E
A4	AND	H
A5	AND	L
E6 20	AND	N

BIT

- valoarea complementată a bitului operandului din

stînga, care se găseşte şi în operandul din dreapta, este

copiată în flagul Z. Adresarea poate fi implicită,

indirectă sau indexată.

CB 46	BIT	0, (HL)
DD CB 05 46	BIT	0, (IX+IND)
FD CB 05 46	BIT	0, (IY+IND)
CB 47	BIT	0, A
CB 40	BIT	0, B
CB 41	BIT	0, C
CB 42	BIT	0, D
CB 43	BIT	0, E
CB 44	BIT	0, H
CB 45	BIT	0, L
CB 4E	BIT	1, (HL)
DD CB 05 4E	BIT	1, (IX+IND)
FD CB 05 4E	BIT	1, (IY+IND)
CB 4F	BIT	1, A
CB 48	BIT	1, B
CB 49	BIT	1, C
CB 4A	BIT	1, D
CB 4B	BIT	1, E
CB 4C	BIT	1, H
CB 4D	BIT	1, L
CB 56	BIT	2, (HL)
DD CB 05 56	BIT	2, (IX+IND)
FD CB 05 56	BIT	2, (IY+IND)
CB 57	BIT	2, A
CB 50	BIT	2, B
CB 51	BIT	2, C
CB 52	BIT	2, D
CB 53	BIT	2, E
CB 54	BIT	2, H
CB 55	BIT	2, L
CB 5E	BIT	3, (HL)
DD CB 05 5E	BIT	3, (IX+IND)
FD CB 05 5E	BIT	3, (IY+IND)
CB 5F	BIT	3, A
CB 58	BIT	3, B
CB 59	BIT	3, C
CB 5A	BIT	3, D
CB 5B	BIT	3, E
CB 5C	BIT	3, H
CB 5D	BIT	3, L
CB 66	BIT	4, (HL)
DD CB 05 66	BIT	4, (IX+IND)
FD CB 05 66	BIT	4, (IY+IND)
CB 67	BIT	4, A
CB 60	BIT	4, B
CB 61	BIT	4, C
CB 62	BIT	4, D
CB 63	BIT	4, E
CB 64	BIT	4, H
CB 65	BIT	4, L
CB 6E	BIT	5, (HL)
DD CB 05 6E	BIT	5, (IX+IND)
FD CB 05 6E	BIT	5, (IY+IND)
CB 6F	BIT	5, A
CB 68	BIT	5, B
CB 69	BIT	5, C
CB 6A	BIT	5, D
CB 6B	BIT	5, E
CB 6C	BIT	5, H
CB 6D	BIT	5, L
CB 76	BIT	6, (HL)
DD CB 05 76	BIT	6, (IX+IND)
FD CB 05 76	BIT	6, (IY+IND)
CB 77	BIT	6, A
CB 70	BIT	6, B
CB 71	BIT	6, C
CB 72	BIT	6, D
CB 73	BIT	6, E
CB 74	BIT	6, H
CB 75	BIT	6, L
CB 7E	BIT	7, (HL)
DD CB 05 7E	BIT	7, (IX+IND)
FD CB 05 7E	BIT	7, (IY+IND)
CB 7F	BIT	7, A
CB 78	BIT	7, B
CB 79	BIT	7, C
CB 7A	BIT	7, D
CB 7B	BIT	7, E
CB 7C	BIT	7, H
CB 7D	BIT	7, L

CALL

- dacă apare un singur operand, acesta este adresa

cu care se încarcă PC, după salvarea conţinutului său în

vărful stivei. Dacă apar doi operanzi, cel din stînga este o

condiţie. Dacă condiţia este adevărată se execută saltul la

subrutină iar PC se încarcă cu adresa reprezentată de operandul din

dreapta. Dacă condiţia este falsă se execută

instrucţiunea

următoare din program. Adresarea poate fi directă, implicită,

sau indirectă.

DC BB AA	CALL	C, NN
FC BB AA	CALL	M, NN
D4 BB AA	CALL	NC, NN
CD BB AA	CALL	NN
C4 BB AA	CALL	NZ, NN
F4 BB AA	CALL	P, NN
EC BB AA	CALL	PE, NN
E4 BB AA	CALL	PO, NN
CC BB AA	CALL	Z, NN

CCF

- indicatorul de transport este complementat. Adresarea este implicită.

3F	CCF

CP

- se efectuează o scădere, conţinutul operandului se

scade din conţinutul registrului A. Ca rezultat se poziţionează

indicatorii de condiţie. Adresarea poate fi implicită, indirectă

sau indexată.

BE	CP	(HL)
DD BE 05	CP	(IX+IND)
FD BE 05	CP	(IY+IND)
BF	CP	A
B8	CP	B
B9	CP	C
BA	CP	D
BB	CP	E
BC	CP	H
BD	CP	L
FE 20	CP	N

CPD

- conţinutul acumulatorului A este comparat cu cel al

celulei de memorie adresată prin registrul dublu HL. Indicatorul

de adresă HL este incrementat cu 1 iar numărul de octeţi BC este

decrementat cu 1. Conţinutul lui A nu se schimbă iar rezultatul

comparaţiei se regăseşte în registrul indicator F. Adresarea poate fi

implicită şi indirectă.

ED A9	CPD

CPDR

- conţinutul acumulatorului A este comparat cu cel al

celulei de memorie specificată prin registrul dublu HL. Indicatorul

de adresă HL este decrementat cu 1, numărul de octeţi BC este

decrementat cu 1. Dacă rezultatul comparării arată egalitate

instrucţiunea se termină, dacă nu ea va fi reluată păna cînd

BC=0. Conţinutul lui A rămăne nemodificat. Adresarea poate fi

implicită sau indirectă.

ED B9	CPDR

CPI

- conţinutul acumulatorului A este comparat cu cel al

celulei de memorie adresată prin registrul dublu HL. Indicatorul

de adresă HL este decrementat cu 1, numărul de octeţi BC este

decrementat cu 1. Conţinutul lui A nu se schimbă iar rezultatul

comparaţiei se regăseşte in registrul de flag F. Adresarea poate fi

implicită sau indirectă.

ED A1	CPI

CPIR

- conţinutul acumulatorului A este comparat cu cel al

celulei de memorie specificată prin registrul dublu HL. Indicatorul

de adresă HL este incrementat cu 1, numărul de octeţi BC este

decrementat cu 1. Dacă rezultatul comparării arată egalitate

instrucţiunea se termină, dacă nu ea va fi reluată păna când

BC=0. Conţinutul lui A rămăne nemodificat. Adresarea poate fi

implicită sau indirectă.

ED B1	CPIR

CPL

- conţinutul acumulatorului A este complementat faţa de 1.

Rezultatul se obţine în registrul A. Adresarea este implicită.

2F	CPL

DAA

- dacă se foloseşte una din instrucţiunile ADD, ADC,

INC, SUB, SBC, DEC sau NEG, această instrucţiune ajustează sub

forma BCD conţinutul registrului A. Această instrucţiune impune

existenţa flagurilor N şi H. Adresarea este implicită.

27	DAA

DEC

- conţinutul operandului este decrementat cu 1. Operandul

poate fi pe 8 sau 16 biţi. Adresarea este implicită.

35	DEC	(HL)
DD 35 05	DEC	(IX+IND)
FD 35 05	DEC	(IY+IND)
3D	DEC	A
05	DEC	B
0B	DEC	BC
0D	DEC	C
15	DEC	D
1B	DEC	DE
1D	DEC	E
25	DEC	H
2B	DEC	HL
DD 2B	DEC	IX
FD 2B	DEC	IY
2D	DEC	L
3B	DEC	SP

DI

- sistemul de întreruperi se inhibă. Se acceptă numai

cererile de întrerupere nemascabile. Starea de inhibare poate fi

suspendată prin execuţia unei instrucţiuni EI. Adresarea este implicită.

3F	DI

DJNZ

- conţinutul registrului B este decrementat cu 1. Dacă

astfel s-a ajuns la valoarea B=0 se execută instrucţiunea

următoare din program, altfel se efectuează un salt relativ.

Adresarea poate fi imediată sau implicită.

10 03	DJNZ $+DIS

EI

- sistemul de întreruperi se validează. El va fi apt

sa accepte o întrerupere după execuţia primei instrucţiuni care

urmează după EI. Adresarea este implicită.

FB	EI

EX

- conţinutul celor doi operanzi este interschimbat. Adresarea poate

fi implicită sau indirectă.

E3	EX	(SP), HL
DD E3	EX	(SP), IX
FD E3	EX	(SP), IY
08	EX	AF, AF’
EB	EX	DE, HL

EXX

- regiştrii secundari devin regiştrii primari (de lucru) şi invers. Adresarea este implicită.

D9	EXX

HALT

- după execuţia acestei instrucţiuni microprocesorul

se opreşte. Ieşirea din această stare se face numai prin recepţia

unei întreruperi sau a unui semnal de RESET.

76	HALT

IM

- stabileşte modul de întrerupere specificat: 0, 1 sau 2.

Adresarea este implicită.

ED 46	IM	0
ED 56	IM	1
ED 5E	IM	2

IN

- conţinutul operandului din dreapta, care este conţinutul

unui port de intrare, este transferat în operandul din stînga.

Adresarea poate fi implicită, directă sau indirectă.

ED 78	IN	A, (C)
DB 20	IN	A, (N)
ED 40	IN	B, (C)
ED 48	IN	C, (C)
ED 50	IN	D, (C)
ED 58	IN	E, (C)
ED 60	IN	H, (C)
ED 68	IN	L, (C)

INC

- conţinutul operandului este incrementat cu 1. Adresarea este implicită.

ED 78	INC	(HL)
DB 20	INC	(IX+IND)
ED 40	INC	(IY+IND)
3C	INC	A
04	INC	B
03	INC	BC
0C	INC	C
14	INC	D
13	INC	DE
1C	INC	E
24	INC	H
23	INC	HL
DD 23	INC	IX
FD 23	INC	IY
2C	INC	L
33	INC	SP

IND

- conţinutul portului de intrare adresat prin

conţinutul registrului C, este transferat în memorie la adresa

specificată de registrul dublu HL. Indicatorul de adresă HL este

decrementat cu 1. Numărătorul de octeţi B este decrementat cu 1.

Adresarea este indirectă.

ED AA	IND

INDR

- conţinutul portului de intrare adre sat prin

conţinutul registrului C, este transferat în celula de memorie

adresată prin registrul dublu HL. Indicatorul de adresă HL este

decrementat cu 1. Numărătorul de octeţi B este decrementat cu 1.

Operaţia se repetă pîna cînd B=0. Adresarea este indirectă.

ED BA	INDR

INI

- conţinutul portului de intrare adresat prin

conţinutul registrului C, este transferat în memorie la adresa

specificată de registrul dublu HL. Indicatorul de adresă HL este

incrementat cu 1. Numărătorul de octeţi B este decrementat cu 1.

Adresarea este indirectă.

ED A2	INI

INIR

- conţinutul portului de intrare adresat prin

conţinutul registrului C, este transferat în celula de memorie

adresată prin registrul dublu HL. Indicatorul de adresă HL este

incrementat cu 1. Numărătorul de octeţi B este decrementat cu 1.

Operaţia se repetă pîna cînd B=0. Adresarea este indirectă.

ED B2	INIR

JP

- în contorul program PC se înscrie adresa specificată

prin operand, de la care se va executa în continuare programul. Dacă

apar doi operanzi, cel din stînga este o condiţie. Se testează

cîte un bit al registrului F. Dacă condiţia e adevărată se

efectuează saltul la adresa specificată prin operandul din dreapta,

dacă nu programul continuă cu instrucţiunea următoare.

Adresarea poate fi directă sau indirectă.

9E	JP	(HL)
DD 9E	JP	(IX)
FD 9E	JP	(IY)
DA BB AA	JP	C, NN
FA BB AA	JP	M, NN
D2 BB AA	JP	NC, NN
C3 BB AA	JP	NN
C2 BB AA	JP	NZ, NN
F2 BB AA	JP	P, NN
EA BB AA	JP	PE, NN
E2 BB AA	JP	PO, NN
CA BB AA	JP	Z, NN

JR

- deplasamentul este adunat la valoarea curentă a

programului contor PC. Programul continuă de la noua adresă. Dacă

apar doi operanzi, cel din stînga este o condiţie. Se testează

cîte un bit al registrului F. Dacă condiţia e adevărată se

efectuează saltul la adresa obţinută în urma adunării

deplasamentului la valoarea curentă a programului contor PC,

dacă nu programul continuă cu instrucţiunea următoare.

Adresarea este imediată.

38 03	JR	C, $+DIS
18 03	JR	$+DIS
30 03	JR	NC, $+DIS
20 03	JR	NZ, $+DIS
28 03	JR	Z, $+DIS

LD

- Copiază 8 sau 16 biţi din locaţia specificată de către operandul din

dreapta în locaţia specificată de operandul din stînga. Operandul poate

fi imediat sau se poate afla în registru sau memorie.

02	LD	(BC), A
12	LD	(DE), A
77	LD	(HL), A
70	LD	(HL), B
71	LD	(HL), C
72	LD	(HL), D
73	LD	(HL), E
74	LD	(HL), H
75	LD	(HL), L

36 20	LD	(HL), N
DD 77 05	LD	(IX+IND), A
DD 70 05	LD	(IX+IND), B
DD 71 05	LD	(IX+IND), C
DD 72 05	LD	(IX+IND), D
DD 73 05	LD	(IX+IND), E
DD 74 05	LD	(IX+IND), H
DD 75 05	LD	(IX+IND), L
DD 36 05 20	LD	(IX+IND), N
FD 77 05	LD	(IY+IND), A
FD 70 05	LD	(IY+IND), B
FD 71 05	LD	(IY+IND), C
FD 72 05	LD	(IY+IND), D
FD 73 05	LD	(IY+IND), E
FD 74 05	LD	(IY+IND), H
FD 75 05	LD	(IY+IND), L
FD 36 05 20	LD	(IY+IND), N
32 BB AA	LD	(NN), A
ED 43 BB AA	LD	(NN), BC
EI 53 BB AA	LD	(NN), DE
22 BB AA	LD	(NN), HL
ED 63 BB AA	LD	(NN), HL
DD 22 BB AA	LD	(NN), IX
FD 22 BB AA	LD	(NN), IY
ED 73 BB AA	LD	(NN), SP
0A	LD	A, (BC)
1A	LD	A, (DE)
7E	LD	A, (HL)
DD 7E 05	LD	A, (IX+IND)
FD 7E 05	LD	A, (IY+IND)
3A BB AA	LD	A, (NN)
7F	LD	A, A
78	LD	A, B
79	LD	A, C
7A	LD	A, D
7B	LD	A, E
7C	LD	A, H
7D	LD	A, L
ED 57	LD	A, I
3E 20	LD	A, N
ED 5F	LD	A, R
46	LD	B, (HL)
DD 46 05	LD	B, (IX+IND)
FD 46 05	LD	B, (IY+IND)
47	LD	B, A
40	LD	B, B
41	LD	B, C
42	LD	B, D
43	LD	B, E
44	LD	B, H
45	LD	B, L
06 20	LD	B, N
ED 4B BB AA	LD	BC,(NN)
01 BB AA	LD	BC, NN
4E	LD	C, (HL)
DD 4E 05	LD	C, (IX+IND)
FD 4E 05	LD	C, (IY+IND)
4F	LD	C, A
48	LD	C, B
49	LD	C, C
4A	LD	C, D
4B	LD	C, E
4C	LD	C, H
4D	LD	C, L
0E 20	LD	C, N
56	LD	D, (HL)
DD 56 05	LD	D, (IX+IND)
FD 56 05	LD	D, (IY+IND)
57	LD	D, A
50	LD	D, B
51	LD	D, C
52	LD	D, D
53	LD	D, E
54	LD	D, H
55	LD	D, L
16 20	LD	D, N
ED 5B BB AA	LD	DE,(NN)
11 BB AA	LD	DE, NN
5E	LD	E, (HL)
DD 5E 05	LD	E, (IX+IND)
FD 5E 05	LD	E, (IY+IND)
5F	LD	E, A
58	LD	E, B
59	LD	E, C
5A	LD	E, D
5B	LD	E, E
5C	LD	E, H
5D	LD	E, L
1E 20	LD	E, N
66	LD	H, (HL)
DD 66 05	LD	H, (IX+IND)
FD 66 05	LD	H, (IY+IND)
67	LD	H, A
60	LD	H, B
61	LD	H, C
62	LD	H, D
63	LD	H, E
64	LD	H, H

65	LD	H, L
26 20	LD	H, N
2A BB AA	LD	HL,(NN)
ED 6B BB AA	LD	HL,(NN)
21 BB AA	LD	HL, NN
ED 47	LD	I, A
DD 2A BB AA	LD	IX,(NN)
DD 21 BB AA	LD	IX, NN
FD 2A BB AA	LD	IY,(NN)
FD 21 BB AA	LD	IY, NN
6E	LD	L, (HL)
DD 6E 05	LD	L, (IX+IND)
FD 6E 05	LD	L, (IY+IND)
6F	LD	L, A
68	LD	L, B
69	LD	L, C
6A	LD	L, D
6B	LD	L, E
6C	LD	L, H
6D	LD	L, L
2E 20	LD	L, N
ED 4F	LD	R, A
ED 7B BB AA	LD	SP, (NN)
9F	LD	SP, HL
DD F9	LD	SP, IX
FD F9	LD	SP, IY
31 BB AA	LD	SP, NN

LDD

- conţinutul celulei de memorie adresată prin registrul

dublu HL este transferat în celula de memorie adresată prin registrul

dublu DE. Conţinutul regiştrilor DE, HL şi BC este decrementat cu 1.

Adresarea este inditectă.

ED A8 LDD

LDDR

- instrucţiunea transferă un bloc de date de lungime

egală cu BC dintr-o zonă de memorie în alta. Blocul sursă începe

la adresa specificată de registrul HL iar cel destinaţie la cea

specificată de DE. Transferul are loc în sens descrescător.

Adresarea este indirectă.

ED B8 LDDR

LDI

- conţinutul celulei de memorie adresată prin registrul

dublu HL este transferat în celula de memorie adresată prin registrul

dublu DE. Conţinutul regiştrilor DE şi HL este incrementat cu 1, iar

cel al registrului dublu BC decrementat cu 1.

Adresarea este indirectă.

ED A0 LDI

LDIR

- instrucţiunea transferă un bloc de date de lungime

egală cu BC dintr-o zonă de memorie în alta. blocul sursă începe

la adresa specificată de registrul HL iar cel destinaţie la cea

specificată de DE. Transferul are loc în sens crescător.

Adresarea este indirectă.

ED B0 LDIR

NEG

- conţinutul acumulatorului A este complementat faţă

de 2. Rezultatul se obţine în acumulatorul A. Adresarea este implicită.

ED 44	NEG

NOP

- incrementează contorul program PC.

00	NOP

OR

- Realizează funcţia logică OR bit cu bit între acumulator şi operandul

din dreapta. Operandul poate fi imediat sau conţinut într-un registru sau

locaţie de memorie.

B6	OR	(HL)
DD B6 05	OR	(IX+IND)
FD B6 05	OR	(IY+IND)
B7	OR	A
B0	OR	B
B1	OR	C
B2	OR	D
B3	OR	E
B4	OR	H
B5	OR	L
F6 20	OR	N

OTDR

- conţinutul celulei de memorie adresată prin registrul

dublu HL este transferat la portul de ieşire adresat prin conţinutul

registrului C. Indicatorul de adresă HL este decrementat cu 1.

Numărătorul de octeţi B este decrementat cu 1. Operaţia se reia

până când B=0. Adresarea este indirectă.

ED BB	OTDR

OTIR

- conţinutul celulei de memorie adresată prin registrul

dublu HL este transferat la portul de ieşire adresat prin conţinutul

registrului C. Indicatorul de adresă HL este incrementat cu 1.

Numărătorul de octeţi B este decrementat cu 1. Operaţia se reia

până când B=0. Adresarea este indirectă.

ED B3	OTIR

OUT

- conţinutul operandului din dreapta este transferat

la portul de ieşire cu adresa specificată de operandul din stînga.

Adresarea poate fi directă, implicită sau indirectă.

ED 79	OUT	(C), A
ED 41	OUT	(C), B
ED 49	OUT	(C), C
ED 51	OUT	(C), D
ED 59	OUT	(C), E

```
ED 61       OUT   (C), H
ED 69       OUT   (C), L
D3 20       OUT   (N), A
```

OUTD

- conţinutul celulei de memorie adresată prin registrul

dublu HL este transferat la portul de ieşire adresat prin

conţinutul registrului C. Indicatorul de adresă HL şi numărătorul

de octeţi B sunt decrementaţi cu 1. Adresarea este indirectă.

```
ED AB       OUTD
```

OUTI

- conţinutul celulei de memorie adresată prin registrul

dublu HL este transferat la portul de ieşire adresat prin

conţinutul registrului C. Indicatorul de adresă HL este

incrementat cu 1. Numărătorul de octeţi B este decrementat cu 1.

Adresarea este indirectă.

```
ED A3       OUTI
```

POP

- conţinutul operandului, care este un registru dublu,

este încărcat din memorie de la adresa specificată prin indicatorul

de stivă SP. Conţinutul celulei cu adresă inferioară este transferat

în octetul inferior al registrului. Indicatorul de stivă este

incrementat cu 2. Adresarea este implicită şi indirectă.

```
F1          POP   AF
C1          POP   BC
D1          POP   DE
E1          POP   HL
DD E1       POP   IX
FD E1       POP   IY
```

PUSH

- conţinutul operandului, care este un registru dublu,

este salvat în memorie la adresa specificată de indicatorul de stivă

SP. Salvarea se face la adrese descrescătoare, prima salvare

implicînd octetul superior al registrului. Adresarea este

implicită şi indirectă.

```
F5          PUSH  AF
C5          PUSH  BC
D5          PUSH  DE
E5          PUSH  HL
DD E5       PUSH  IX
FD E5       PUSH  IY
```

RES

- în bitul cu numărul specificat de operandul din stînga

din operandul din dreapta se inscrie valoarea zero. Adresarea poate fi

implicită, indirectă sau indexată.

CB 86	RES	0, (HL)
DD CB 05 86	RES	0, (IX+IND)
FD CB 05 86	RES	0, (IY+IND)
CB 87	RES	0, A
CB 80	RES	0, B
CB 81	RES	0, C
CB 82	RES	0, D
CB 83	RES	0, E
CB 84	RES	0, H
CB 85	RES	0, L
CB 8E	RES	1, (HL)
DD CB 05 8E	RES	1, (IX+IND)
FD CB 05 8E	RES	1, (IY+IND)
CB 8F	RES	1, A
CB 88	RES	1, B
CB 89	RES	1, C
CB 8A	RES	1, D
CB 8B	RES	1, E
CB 8C	RES	1, H
CB 8D	RES	1, L
CB 96	RES	2, (HL)
DD CB 05 8E	RES	2, (IX+IND)
FD CB 05 8E	RES	2, (IY+IND)
CB 97	RES	2, A
CB 90	RES	2, B
CB 91	RES	2, C
CB 92	RES	2, D
CB 93	RES	2, E
CB 94	RES	2, H
CB 95	RES	2, L
CB 9E	RES	3, (HL)
DD CB 05 9E	RES	3, (IX+IND)
FD CB 05 9E	RES	3, (IY+IND)
CB 9F	RES	3, A
CB 98	RES	3, B
CB 99	RES	3, C
CB 9A	RES	3, D
CB 9B	RES	3, E
CB 9C	RES	3, H
CB 9D	RES	3, L
CB A6	RES	4, (HL)
DD CB 05 A6	RES	4, (IX+IND)
FD CB 05 A6	RES	4, (IY+IND)
CB A7	RES	4, A
CB A0	RES	4, B
CB A1	RES	4, C
CB A2	RES	4, D
CB A3	RES	4, E
CB A4	RES	4, H
CB A5	RES	4, L
CB AE	RES	5, (HL)
DD CB 05 AE	RES	5, (IX+IND)
FD CB 05 AE	RES	5, (IY+IND)
CB AF	RES	5, A
CB A8	RES	5, B
CB A9	RES	5, C
CB AA	RES	5, D
CB AB	RES	5, E
CB AC	RES	5, H
CB AD	RES	5, L
CB B6	RES	6, (HL)
DD CB 05 B6	RES	6, (IX+IND)
FD CB 05 B6	RES	6, (IY+IND)
CB B7	RES	6, A
CB B0	RES	6, B
CB B1	RES	6, C
CB B2	RES	6, D
CB B3	RES	6, E
CB B4	RES	6, H
CB B5	RES	6, L
CB BE	RES	7, (HL)
DD CB 05 BE	RES	7, (IX+IND)
FD CB 05 BE	RES	7, (IY+IND)
CB BF	RES	7, A
CB B8	RES	7, B
CB B9	RES	7, C
CB BA	RES	7, D
CB BB	RES	7, E
CB BC	RES	7, H
CB BD	RES	7, L

RET

- adresa de revenire din subrutină în programul apelant

este presupusă a fi în vîrful stivei. Ea se încarcă în contorul program PC, după care se efectuează saltul. Dacă apare un

operand, acesta este o condiţie, care, dacă este adevărată, se

execută revenirea în programul apelant. Adresarea este indirectă.

C9	RET	
D8	RET	C
F8	RET	M
D0	RET	NC
C0	RET	NY
F0	RET	P
E8	RET	PE
E0	RET	P0
C8	RET	Z

RETI

- din vîrful stivei (SP) se încarcă adresa de revenire

în contorul program PC. Este necesar a se executa instrucţiunea EI

înainte pentru a se revalida întreruperile mascabile. Adresarea este

indirectă.

ED 4D	RETI

RETN

- din vîrful stivei (SP) se încarcă adresa de revenire

în contorul program PC. Conţinutul indicatorului de stivă SP este

incrementat cu 2. Adresarea este indirectă.

ED 45	RETN

RL

- conţinutul operandului este deplasat la stînga cu o

poziţie. Bitul 7 se transferă în carry iar carry în bitul 0.

Adresarea poate fi implicită, indirectă sau indexată.

CB 16	RL	(HL)
DD CB 05 16	RL	(IX+IND)
FD CB 05 16	RL	(IY+IND)
CB 17	RL	A
CB 10	RL	B
CB 11	RL	C
CB 12	RL	D
CB 13	RL	E
CB 14	RL	H
CB 15	RL	L

RLA

- conţinutul acumulatorului A este rotit cu o poziţie

la stînga. Bitul 7 se mută în Cy iar Cy se mută în bit 0.

Adresarea este implicită.

17	RLA

RLC

- conţinutul operandului este deplasat cu o poziţie

la stînga. Bitul 7 se transferă în carry şi în bitul 0.

Adresarea poate fi implicită, indirectă sau indexată.

CB 06	RLC	(HL)
DD CB 05 06	RLC	(IX+IND)
FD CB 05 06	RLC	(IY+IND)
CB 07	RLC	A
CB 00	RLC	B
CB 01	RLC	C
CB 02	RLC	D
CB 03	RLC	E
CB 04	RLC	H
CB 05	RLC	L

RLCA

- conţinutul acumulatorului A este deplasat cu o

poziţie la stînga. Bitul 7 iniţial se mută atăt în Cy cît

şi pe poziţia bit 0. Adresarea este implicită.

07	RLCA

RLD

- conţinutul celulei de memorie adresată prin registrul

dublu HL este rotit la stînga folosind digitul inferior al

registrului A. Adresarea poate fi implicită sau indirectă.

ED 6F	RLD

RR

- conţinutul operandului este deplasat la dreapta cu o

poziţie. Bitul 0 se transferă în carry iar carry în bitul 7.

Adresarea poate fi implicită, indirectă sau indexată.

CB 1E	RR	(HL)
DD CB 05 1E	RR	(IX+IND)
FD CB 05 1E	RR	(IY+IND)
CB 1F	RR	A
CB 18	RR	B
CB 19	RR	C
CB 1A	RR	D
CB 1B	RR	E
CB 1C	RR	H
CB 1D	RR	L

RRA

- conţinutul acumulatorului A este rotit cu o poziţie

la dreapta. Bitul 0 se mută în Cy iar Cy se mută în bit 7.

Adresarea este implicită.

1F	RRA

RRC

- conţinutul operandului este deplasat cu o poziţie

la dreapta. Bitul 0 se transferă în carry şi în bitul 7.

Adresarea poate fi implicită, indirectă sau indexată.

CB 0E	RRC	(HL)
DD CB 05 0E	RRC	(IX+IND)
FD CB 05 0E	RRC	(IY+IND)
CB 0F	RRC	A
CB 08	RRC	B
CB 09	RRC	C
CB 0A	RRC	D
CB 0B	RRC	E
CB 0C	RRC	H
CB 0D	RRC	L

RRCA

- conţinutul acumulatorului A este deplasat cu o

poziţie la dreapta. Bitul 0 iniţial se mută atît în Cy cît

şi pe poziţia bit 7. Adresarea este implicită.

0F	RRCA

RRD

- conţinutul celulei de memorie adresată prin registrul

dublu HL este rotit la dreapta folosind digitul inferior al

registrului A. Adresarea poate fi implicită sau indirectă.

ED 67	RRD	

RST

- se salvează adresa de revenire (PC) în vîrful stivei.

PC se încarcă cu valoarea operandului şi se execută saltul.

Conţinutul indicatorului de stivă SP este decrementat cu 2.

Adresarea poate fi implicită sau indirectă.

C7	RST	00H
CF	RST	08H
D7	RST	10H
DF	RST	18H
E7	RST	20H
EF	RST	28H
F7	RST	30H
FF	RST	38H

SBC

- se efectuează o scădere dublă: conţinutul operandului

din dreapta prin registrul dublu şi conţinutul indicatorului de

transport (carry) se scad din conţinutul registrului A.

Adresarea poate fi implicită, indirectă sau indexată.

9E	SBC	A, (HL)
DD 9E 05	SBC	A, (IX+IND)
FD 9E 05	SBC	A, (IY+IND)
9F	SBC	A, A
98	SBC	A, B
99	SBC	A, C
9A	SBC	A, D
9B	SBC	A, E
9C	SBC	A, H
9D	SBC	A, L
DE 20	SBC	A, N
ED 42	SBC	HL, BC
ED 52	SBC	HL, DE
ED 62	SBC	HL, HL
ED 72	SBC	HL, SP

SCF

- înscrie indicatorul de transport. Adresarea este implicită.

37	SCF	

SET

- în bitul cu numărul specificat de operandul din stînga

al celulei de memorie adresată prin operandul din dreapta se înscrie

valoarea 1. Adresarea poate fi implicită, indirectă sau indexată.

CB C6	SET	0, (HL)
DD CB 05 C6	SET	0, (IX+IND)
FD CB 05 C6	SET	0, (IY+IND)
CB C7	SET	0, A
CB C0	SET	0, B
CB C1	SET	0, C
CB C2	SET	0, D
CB C3	SET	0, E
CB C4	SET	0, H

CB C5	SET	0, L
CB CE	SET	1, (HL)
DD CB 05 CE	SET	1, (IX+IND)
FD CB 05 CE	SET	1, (IY+IND)
CB CF	SET	1, A
CB C8	SET	1, B
CB C9	SET	1, C
CB CA	SET	1, D
CB CB	SET	1, E
CB CC	SET	1, H
CB CD	SET	1, L
CB D6	SET	2, (HL)
DD CB 05 D6	SET	2, (IX+IND)
FD CB 05 D6	SET	2, (IY+IND)
CB D7	SET	2, A
CB D0	SET	2, B
CB D1	SET	2, C
CB D2	SET	2, D
CB D3	SET	2, E
CB D4	SET	2, H
CB D5	SET	2, L
CB DE	SET	3, (HL)
DD CB 05 DE	SET	3, (IX+IND)
FD CB 05 DE	SET	3, (IY+IND)
CB DF	SET	3, A
CB D8	SET	3, B
CB D9	SET	3, C
CB DA	SET	3, D
CB DB	SET	3, E
CB DC	SET	3, H
CB DD	SET	3, L
CB E6	SET	4, (HL)
DD CB 05 E6	SET	4, (IX+IND)
FD CB 05 E6	SET	4, (IY+IND)
CB E7	SET	4, A
CB E0	SET	4, B
CB E1	SET	4, C
CB E2	SET	4, D
CB E3	SET	4, E
CB E4	SET	4, H
CB E5	SET	4, L
CB EE	SET	5, (HL)
DD CB 05 EE	SET	5, (IX+IND)
FD CB 05 EE	SET	5, (IY+IND)
CB EF	SET	5, A
CB E8	SET	5, B
CB E9	SET	5, C
CB EA	SET	5, D
CB EB	SET	5, E
CB EC	SET	5, H
CB ED	SET	5, L
CB F6	SET	6, (HL)
DD CB 05 F6	SET	6, (IX+IND)
FD CB 05 F6	SET	6, (IY+IND)
CB F7	SET	6, A
CB F0	SET	6, B
CB F1	SET	6, C
CB F2	SET	6, D
CB F3	SET	6, E
CB F4	SET	6, H
CB F5	SET	6, L
CB FE	SET	7, (HL)
DD CB 05 FE	SET	7, (IX+IND)
FD CB 05 FE	SET	7, (IY+IND)
CB FF	SET	7, A
CB F8	SET	7, B
CB F9	SET	7, C
CB FA	SET	7, D
CB FB	SET	7, E
CB FC	SET	7, H
CB FD	SET	7, L

SLA

- conţinutul operandului este deplasat cu o poziţie

la stînga. Bitul 7 se transferă în carry iar în bitul 0 se inserează

valoarea 0. Adresarea poate fi implicită, indirectă sau indexată.

CB 26	SLA	(HL)
DD CB 05 26	SLA	(IX+IND)
FD CB 05 26	SLA	(IY+IND)
CB 27	SLA	A
CB 20	SLA	B
CB 21	SLA	C
CB 22	SLA	D
CB 23	SLA	E
CB 24	SLA	H
CB 25	SLA	L

SRA

- conţinutul operandului este deplasat cu o poziţie

la dreapta. Bitul 0 se transferă în carry iar bitul 7 rămăne

neschimbat. Adresarea poate fi implicită, indirectă sau indexată.

CB 2E	SRA	(HL)
DD CB 05 2E	SRA	(IX+IND)
FD CB 05 2E	SRA	(IY+IND)
CB 2F	SRA	A
CB 28	SRA	B
CB 29	SRA	C
CB 2A	SRA	D
CB 2B	SRA	E
CB 2C	SRA	H
CB 2D	SRA	L

SRL

- conţinutul operandului este deplasat cu o poziţie

la dreapta. Bitul 0 se transferă în carry iar în bitul 7 se

inserează valoarea 0. Adresarea poate fi implicită, indirectă sau

indexată.

CB 3E	SRL	(HL)
DD CB 05 3E	SRL	(IX+IND)
FD CB 05 3E	SRL	(IY+IND)
CB 3F	SRL	A
CB 38	SRL	B
CB 39	SRL	C
CB 3A	SRL	D
CB 3B	SRL	E
CB 3C	SRL	H
CB 3D	SRL	L

SUB

- conţinutul operandului se scade din conţinutul

registrului A. Adresarea poate fi implicită, indirectă sau indexată.

96	SUB	(HL)
DD 96 05	SUB	(IX+IND)
FD 96 05	SUB	(IY+IND)
97	SUB	A
90	SUB	B
91	SUB	C
92	SUB	D
93	SUB	E
94	SUB	H
95	SUB	L
D6 20	SUB	N

XOR

- se execută o functie SAU EXCLUSIV între conţinutul

registrului A şi operand. Operaţia se execută pe cîte 2 biţi,

fiecare avînd aceeaşi poziţie semnificativă. Adresarea poate fi

implicită, indirectă sau indexată.

AE	XOR	(HL)
DD AE 05	XOR	(IX+IND)
FD AE 05	XOR	(IY+IND)
AF	XOR	A
A8	XOR	B
A9	XOR	C
AA	XOR	D
AB	XOR	E
AC	XOR	H
AD	XOR	L
EE 20	XOR	N

Lista instrucţiunilor crescătoare a codurilor

00	NOP	
01 BB AA	LD	BC, NN
02	LD	(BC), A
03	INC	BC
04	INC	B
05	DEC	B
06 20	LD	B, N
07	RLCA	
08	EX	AF, AF'
09	ADD	HL, BC
0A	LD	A, (BC)
0B	DEC	BC
0C	INC	C
0D	DEC	C
0E 20	LD	C, N
0F	RRCA	
10 03	DJNZ	
11 BB AA	LD	DE, NN
12	LD	(DE), A
13	INC	DE
14	INC	D
15	DEC	D
16 20	LD	D, N
17	RLA	
18 03	JR	
19	ADD	HL, DE
1A	LD	A, (DE)
1B	DEC	DE
1C	INC	E
1D	DEC	E
1E 20	LD	E, N
1F	RRA	
20 03	JR	NZ, $+DIS
21 BB AA	LD	HL, NN
22 BB AA	LD	(NN), HL
23	INC	HL
24	INC	H
25	DEC	H
26 20	LD	H, N
27	DAA	
28 03	JR	Z, $+DIS
29	ADD	HL, HL
2A BB AA	LD	HL,(NN)
2B	DEC	HL
2C	INC	L
2D	DEC	L
2E 20	LD	L, N
2F	CPL	
30 03	JR	NC, $+DIS
31 BB AA	LD	SP, NN
32 BB AA	LD	(NN), A
33	INC	SP
35	DEC	(HL)
36 20	LD	(HL), N
37	SCF	
38 03	JR	C, $+DIS
39	ADD	HL, SP
3A BB AA	LD	A, (NN)
3B	DEC	SP
3C	INC	A
3D	DEC	A
3E 20	LD	A, N
3F	CCF	
3F	DI	
40	LD	B, B
41	LD	B, C
42	LD	B, D
43	LD	B, E
44	LD	B, H
45	LD	B, L
46	LD	B, (HL)
47	LD	B, A
48	LD	C, B
49	LD	C, C
4A	LD	C, D
4B	LD	C, E
4C	LD	C, H
4D	LD	C, L
4E	LD	C, (HL)
4F	LD	C, A
50	LD	D, B
51	LD	D, C
52	LD	D, D
53	LD	D, E
54	LD	D, H
55	LD	D, L
56	LD	D, (HL)
57	LD	D, A
58	LD	E, B
59	LD	E, C
5A	LD	E, D
5B	LD	E, E
5C	LD	E, H
5D	LD	E, L

5E	LD	E, (HL)	8F	ADC	A, A
5F	LD	E, A	90	SUB	B
60	LD	H, B	91	SUB	C
61	LD	H, C	92	SUB	D
62	LD	H, D	93	SUB	E
63	LD	H, E	94	SUB	H
64	LD	H, H	95	SUB	L
65	LD	H, L	96	SUB	(HL)
66	LD	H, (HL)	97	SUB	A
67	LD	H, A	98	SBC	A, B
68	LD	L, B	99	SBC	A, C
69	LD	L, C	9A	SBC	A, D
6A	LD	L, D	9B	SBC	A, E
6B	LD	L, E	9C	SBC	A, H
6C	LD	L, H	9D	SBC	A, L
6D	LD	L, L	9E	JP	(HL)
6E	LD	L, (HL)	9E	SBC	A, (HL)
6F	LD	L, A	9F	LD	SP, HL
70	LD	(HL), B	9F	SBC	A, A
71	LD	(HL), C	A0	AND	B
72	LD	(HL), D	A1	AND	C
73	LD	(HL), E	A2	AND	D
74	LD	(HL), H	A3	AND	E
75	LD	(HL), L	A4	AND	H
76	HALT		A5	AND	L
77	LD	(HL), A	A6	AND	(HL)
78	LD	A, B	A7	AND	A
79	LD	A, C	A8	XOR	B
7A	LD	A, D	A9	XOR	C
7B	LD	A, E	AA	XOR	D
7C	LD	A, H	AB	XOR	E
7D	LD	A, L	AC	XOR	H
7E	LD	A, (HL)	AD	XOR	L
7F	LD	A, A	AE	XOR	(HL)
80	ADD	A, B	AF	XOR	A
81	ADD	A, C	B0	OR	B
82	ADD	A, D	B1	OR	C
83	ADD	A, E	B2	OR	D
84	ADD	A, H	B3	OR	E
85	ADD	A, L	B4	OR	H
86	ADD	A, (HL)	B5	OR	L
87	ADD	A, A	B6	OR	(HL)
88	ADC	A, B	B7	OR	A
89	ADC	A, C	B8	CP	B
8A	ADC	A, D	B9	CP	C
8B	ADC	A, E	BA	CP	D
8C	ADC	A, H	BB	CP	E
8D	ADC	A, L	BC	CP	H
8E	ADC	A, (HL)	BD	CP	L

BE	CP	(HL)
BF	CP	A
C0	RET	NY
C1	POP	BC
C2 BB AA	JP	NZ, NN
C3 BB AA	JP	NN
C4 BB AA	CALL	NZ, NN
C5	PUSH	BC
C6 20	ADD	A, N
C7	RST	00H
C8	RET	Z
C9	RET	
CA BB AA	JP	Z, NN
CB 00	RLC	B
CB 01	RLC	C
CB 02	RLC	D
CB 03	RLC	E
CB 04	RLC	H
CB 05	RLC	L
CB 06	RLC	(HL)
CB 07	RLC	A
CB 08	RRC	B
CB 09	RRC	C
CB 0A	RRC	D
CB 0B	RRC	E
CB 0C	RRC	H
CB 0D	RRC	L
CB 0E	RRC	(HL)
CB 0F	RRC	A
CB 10	RL	B
CB 11	RL	C
CB 12	RL	D
CB 13	RL	E
CB 14	RL	H
CB 15	RL	L
CB 16	RL	(HL)
CB 17	RL	A
CB 18	RR	B
CB 19	RR	C
CB 1A	RR	D
CB 1B	RR	E
CB 1C	RR	H
CB 1D	RR	L
CB 1E	RR	(HL)
CB 1F	RR	A
CB 20	SLA	B
CB 21	SLA	C
CB 22	SLA	D
CB 23	SLA	E
CB 24	SLA	H
CB 25	SLA	L
CB 26	SLA	(HL)
CB 27	SLA	A
CB 28	SRA	B
CB 29	SRA	C
CB 2A	SRA	D
CB 2B	SRA	E
CB 2C	SRA	H
CB 2D	SRA	L
CB 2E	SRA	(HL)
CB 2F	SRA	A
CB 38	SRL	B
CB 39	SRL	C
CB 3A	SRL	D
CB 3B	SRL	E
CB 3C	SRL	H
CB 3D	SRL	L
CB 3E	SRL	(HL)
CB 3F	SRL	A
CB 40	BIT	0, B
CB 41	BIT	0, C
CB 42	BIT	0, D
CB 43	BIT	0, E
CB 44	BIT	0, H
CB 45	BIT	0, L
CB 46	BIT	0, (HL)
CB 47	BIT	0, A
CB 48	BIT	1, B
CB 49	BIT	1, C
CB 4A	BIT	1, D
CB 4B	BIT	1, E
CB 4C	BIT	1, H
CB 4D	BIT	1, L
CB 4E	BIT	1, (HL)
CB 4F	BIT	1, A
CB 50	BIT	2, B
CB 51	BIT	2, C
CB 52	BIT	2, D
CB 53	BIT	2, E
CB 54	BIT	2, H
CB 55	BIT	2, L
CB 56	BIT	2, (HL)
CB 57	BIT	2, A
CB 58	BIT	3, B
CB 59	BIT	3, C
CB 5A	BIT	3, D
CB 5B	BIT	3, E
CB 5C	BIT	3, H

CB 5D	BIT	3, L	CB 8E	RES	1, (HL)
CB 5E	BIT	3, (HL)	CB 8F	RES	1, A
CB 5F	BIT	3, A	CB 90	RES	2, B
CB 60	BIT	4, B	CB 91	RES	2, C
CB 61	BIT	4, C	CB 92	RES	2, D
CB 62	BIT	4, D	CB 93	RES	2, E
CB 63	BIT	4, E	CB 94	RES	2, H
CB 64	BIT	4, H	CB 95	RES	2, L
CB 65	BIT	4, L	CB 96	RES	2, (HL)
CB 66	BIT	4, (HL)	CB 97	RES	2, A
CB 67	BIT	4, A	CB 98	RES	3, B
CB 68	BIT	5, B	CB 99	RES	3, C
CB 69	BIT	5, C	CB 9A	RES	3, D
CB 6A	BIT	5, D	CB 9B	RES	3, E
CB 6B	BIT	5, E	CB 9C	RES	3, H
CB 6C	BIT	5, H	CB 9D	RES	3, L
CB 6D	BIT	5, L	CB 9E	RES	3, (HL)
CB 6E	BIT	5, (HL)	CB 9F	RES	3, A
CB 6F	BIT	5, A	CB A0	RES	4, B
CB 70	BIT	6, B	CB A1	RES	4, C
CB 71	BIT	6, C	CB A2	RES	4, D
CB 72	BIT	6, D	CB A3	RES	4, E
CB 73	BIT	6, E	CB A4	RES	4, H
CB 74	BIT	6, H	CB A5	RES	4, L
CB 75	BIT	6, L	CB A6	RES	4, (HL)
CB 76	BIT	6, (HL)	CB A7	RES	4, A
CB 77	BIT	6, A	CB A8	RES	5, B
CB 78	BIT	7, B	CB A9	RES	5, C
CB 79	BIT	7, C	CB AA	RES	5, D
CB 7A	BIT	7, D	CB AB	RES	5, E
CB 7B	BIT	7, E	CB AC	RES	5, H
CB 7C	BIT	7, H	CB AD	RES	5, L
CB 7D	BIT	7, L	CB AE	RES	5, (HL)
CB 7E	BIT	7, (HL)	CB AF	RES	5, A
CB 7F	BIT	7, A	CB B0	RES	6, B
CB 80	RES	0, B	CB B1	RES	6, C
CB 81	RES	0, C	CB B2	RES	6, D
CB 82	RES	0, D	CB B3	RES	6, E
CB 83	RES	0, E	CB B4	RES	6, H
CB 84	RES	0, H	CB B5	RES	6, L
CB 85	RES	0, L	CB B6	RES	6, (HL)
CB 86	RES	0, (HL)	CB B7	RES	6, A
CB 87	RES	0, A	CB B8	RES	7, B
CB 88	RES	1, B	CB B9	RES	7, C
CB 89	RES	1, C	CB BA	RES	7, D
CB 8A	RES	1, D	CB BB	RES	7, E
CB 8B	RES	1, E	CB BC	RES	7, H
CB 8C	RES	1, H	CB BD	RES	7, L
CB 8D	RES	1, L	CB BE	RES	7, (HL)

CB BF	RES	7, A
CB C0	SET	0, B
CB C1	SET	0, C
CB C2	SET	0, D
CB C3	SET	0, E
CB C4	SET	0, H
CB C5	SET	0, L
CB C6	SET	0, (HL)
CB C7	SET	0, A
CB C8	SET	1, B
CB C9	SET	1, C
CB CA	SET	1, D
CB CB	SET	1, E
CB CC	SET	1, H
CB CD	SET	1, L
CB CE	SET	1, (HL)
CB CF	SET	1, A
CB D0	SET	2, B
CB D1	SET	2, C
CB D2	SET	2, D
CB D3	SET	2, E
CB D4	SET	2, H
CB D5	SET	2, L
CB D6	SET	2, (HL)
CB D7	SET	2, A
CB D8	SET	3, B
CB D9	SET	3, C
CB DA	SET	3, D
CB DB	SET	3, E
CB DC	SET	3, H
CB DD	SET	3, L
CB DE	SET	3, (HL)
CB DF	SET	3, A
CB E0	SET	4, B
CB E1	SET	4, C
CB E2	SET	4, D
CB E3	SET	4, E
CB E4	SET	4, H
CB E5	SET	4, L
CB E6	SET	4, (HL)
CB E7	SET	4, A
CB E8	SET	5, B
CB E9	SET	5, C
CB EA	SET	5, D
CB EB	SET	5, E
CB EC	SET	5, H
CB ED	SET	5, L
CB EE	SET	5, (HL)
CB EF	SET	5, A

CB F0	SET	6, B
CB F1	SET	6, C
CB F2	SET	6, D
CB F3	SET	6, E
CB F4	SET	6, H
CB F5	SET	6, L
CB F6	SET	6, (HL)
CB F7	SET	6, A
CB F8	SET	7, B
CB F9	SET	7, C
CB FA	SET	7, D
CB FB	SET	7, E
CB FC	SET	7, H
CB FD	SET	7, L
CB FE	SET	7, (HL)
CB FF	SET	7, A
CC BB AA	CALL	Z, NN
CD BB AA	CALL	NN
CE 20	ADC	A, N
CF	RST	08H
D0	RET	NC
D1	POP	DE
D2 BB AA	JP	NC, NN
D3 20	OUT	(N), A
D4 BB AA	CALL	NC, NN
D5	PUSH	DE
D6 20	SUB	N
D7	RST	10H
D8	RET	C
D9	EXX	
DA BB AA	JP	C, NN
DB 20	IN	A, (N)
DB 20	INC	(IX+IND)
DC BB AA	CALL	C, NN
DD 09	ADD	IX, BC
DD 19	ADD	IX, DE
DD 21 BB AA	LD	IX, NN
DD 22 BB AA	LD	(NN), IX
DD 23	INC	IX
DD 29	ADD	IX, IX
DD 2A BB AA	LD	IX,(NN)
DD 2B	DEC	IX
DD 35 05	DEC	(IX+IND)
DD 36 05 20	LD	(IX+IND), N
DD 39	ADD	IX, SP
DD 46 05	LD	B, (IX+IND)
DD 4E 05	LD	C, (IX+IND)
DD 56 05	LD	D, (IX+IND)
DD 5E 05	LD	E, (IX+IND)

DD 66 05	LD	H, (IX+IND)
DD 6E 05	LD	L, (IX+IND)
DD 70 05	LD	(IX+IND), B
DD 71 05	LD	(IX+IND), C
DD 72 05	LD	(IX+IND), D
DD 73 05	LD	(IX+IND), E
DD 74 05	LD	(IX+IND), H
DD 75 05	LD	(IX+IND), L
DD 77 05	LD	(IX+IND), A
DD 7E 05	LD	A, (IX+IND)
DD 86 05	ADD	A, (IX+IND)
DD 8E 05	ADC	A, (IX+IND)
DD 96 05	SUB	(IX+IND)
DD 9E	JP	(IX)
DD 9E 05	SBC	A, (IX+IND)
DD A6 05	AND	(IX+IND)
DD AE 05	XOR	(IX+IND)
DD B6 05	OR	(IX+IND)
DD BE 05	CP	(IX+IND)
DD CB 05 06	RLC	(IX+IND)
DD CB 05 0E	RRC	(IX+IND)
DD CB 05 16	RL	(IX+IND)
DD CB 05 1E	RR	(IX+IND)
DD CB 05 26	SLA	(IX+IND)
DD CB 05 2E	SRA	(IX+IND)
DD CB 05 3E	SRL	(IX+IND)
DD CB 05 46	BIT	0, (IX+IND)
DD CB 05 4E	BIT	1, (IX+IND)
DD CB 05 56	BIT	2, (IX+IND)
DD CB 05 5E	BIT	3, (IX+IND)
DD CB 05 66	BIT	4, (IX+IND)
DD CB 05 6E	BIT	5, (IX+IND)
DD CB 05 76	BIT	6, (IX+IND)
DD CB 05 7E	BIT	7, (IX+IND)
DD CB 05 86	RES	0, (IX+IND)
DD CB 05 8E	RES	1, (IX+IND)
DD CB 05 8E	RES	2, (IX+IND)
DD CB 05 9E	RES	3, (IX+IND)
DD CB 05 A6	RES	4, (IX+IND)
DD CB 05 AE	RES	5, (IX+IND)
DD CB 05 B6	RES	6, (IX+IND)
DD CB 05 BE	RES	7, (IX+IND)
DD CB 05 C6	SET	0, (IX+IND)
DD CB 05 CE	SET	1, (IX+IND)
DD CB 05 D6	SET	2, (IX+IND)
DD CB 05 DE	SET	3, (IX+IND)
DD CB 05 E6	SET	4, (IX+IND)
DD CB 05 EE	SET	5, (IX+IND)
DD CB 05 F6	SET	6, (IX+IND)
DD CB 05 FE	SET	7, (IX+IND)
DD E1	POP	IX
DD E3	EX	(SP), IX
DD E5	PUSH	IX
DD F9	LD	SP, IX
DE 20	SBC	A, N
DF	RST	18H
E0	RET	P0
E1	POP	HL
E2 BB AA	JP	PO, NN
E3	EX	(SP), HL
E4 BB AA	CALL	PO, NN
E5	PUSH	HL
E6 20	AND	N
E7	RST	20H
E8	RET	PE
EA BB AA	JP	PE, NN
EB	EX	DE, HL
EC BB AA	CALL	PE, NN
ED 40	IN	B, (C)
ED 40	INC	(IY+IND)
ED 41	OUT	(C), B
ED 42	SBC	HL, BC
ED 43 BB AA	LD	(NN), BC
ED 44	NEG	
ED 45	RETN	
ED 46	IM	0
ED 47	LD	I, A
ED 48	IN	C, (C)
ED 49	OUT	(C), C
ED 4A	ADC	HL, BC
ED 4B BB AA	LD	BC,(NN)
ED 4D	RETI	
ED 4F	LD	R, A
ED 50	IN	D, (C)
ED 51	OUT	(C), D
ED 52	SBC	HL, DE
ED 53	BB AA	LD (NN), DE
ED 56	IM	1
ED 57	LD	A, I
ED 58	IN	E, (C)
ED 59	OUT	(C), E
ED 5A	ADC	HL, DE
ED 5B BB AA	LD	DE,(NN)
ED 5E	IM	2
ED 5F	LD	A, R
ED 60	IN	H, (C)
ED 61	OUT	(C), H
ED 62	SBC	HL, HL

ED 63 BB AA	LD	(NN), HL
ED 67	RRD	
ED 68	IN	L, (C)
ED 69	OUT	(C), L
ED 6A	ADC	HL, HL
ED 6B BB AA	LD	HL,(NN)
ED 6F	RLD	
ED 72	SBC	HL, SP
ED 73 BB AA	LD	(NN), SP
ED 78	IN	A, (C)
ED 78	INC	(HL)
ED 79	OUT	(C), A
ED 7A	ADC	HL, SP
ED 7B BB AA	LD	SP, (NN)
ED A0	LDI	
ED A1	CPI	
ED A2	INI	
ED A3	OUTI	
ED A8	LDD	
ED A9	CPD	
ED AA	IND	
ED AB	OUTD	
ED B0	LDIR	
ED B1	CPIR	
ED B2	INIR	
ED B3	OTIR	
ED B8	LDDR	
ED B9	CPDR	
ED BA	INDR	
ED BB	OTDR	
EE 20	XOR	N
EF	RST	28H
EI 53 BB AA	LD	(NN), DE
F0	RET	P
F1	POP	AF
F2 BB AA	JP	P, NN
F4 BB AA	CALL	P, NN
F5	PUSH	AF
F6 20	OR	N
F7	RST	30H
F8	RET	M
FA BB AA	JP	M, NN
FB	EI	
FC BB AA	CALL	M, NN
FD 09	ADD	IY, BC
FD 19	ADD	IY, DE
FD 21 BB AA	LD	IY, NN
FD 22 BB AA	LD	(NN), IY
FD 23	INC	IY
FD 29	ADD	IY, IX
FD 2A BB AA	LD	IY,(NN)
FD 2B	DEC	IY
FD 34 05	INC	(IY+IND)
FD 35 05	DEC	(IY+IND)
FD 36 05 20	LD	(IY+IND), N
FD 39	ADD	IY, SP
FD 46 05	LD	B, (IY+IND)
FD 4E 05	LD	C, (IY+IND)
FD 56 05	LD	D, (IY+IND)
FD 5E 05	LD	E, (IY+IND)
FD 66 05	LD	H, (IY+IND)
FD 6E 05	LD	L, (IY+IND)
FD 70 05	LD	(IY+IND), B
FD 71 05	LD	(IY+IND), C
FD 72 05	LD	(IY+IND), D
FD 73 05	LD	(IY+IND), E
FD 74 05	LD	(IY+IND), H
FD 75 05	LD	(IY+IND), L
FD 77 05	LD	(IY+IND), A
FD 7E 05	LD	A, (IY+IND)
FD 86 05	ADD	A, (IY+IND)
FD 8E 05	ADC	A, (IY+IND)
FD 96 05	SUB	(IY+IND)
FD 9E 05	SBC	A, (IY+IND)
FD A6 05	AND	(IY+IND)
FD AE 05	XOR	(IY+IND)
FD B6 05	OR	(IY+IND)
FD BE 05	CP	(IY+IND)
FD CB 05 06	RLC	(IY+IND)
FD CB 05 0E	RRC	(IY+IND)
FD CB 05 16	RL	(IY+IND)
FD CB 05 1E	RR	(IY+IND)
FD CB 05 26	SLA	(IY+IND)
FD CB 05 2E	SRA	(IY+IND)
FD CB 05 3E	SRL	(IY+IND)
FD CB 05 46	BIT	0, (IY+IND)
FD CB 05 4E	BIT	1, (IY+IND)
FD CB 05 56	BIT	2, (IY+IND)
FD CB 05 5E	BIT	3, (IY+IND)
FD CB 05 66	BIT	4, (IY+IND)
FD CB 05 6E	BIT	5, (IY+IND)
FD CB 05 76	BIT	6, (IY+IND)
FD CB 05 7E	BIT	7, (IY+IND)
FD CB 05 86	RES	0, (IY+IND)
FD CB 05 8E	RES	1, (IY+IND)
FD CB 05 96	RES	2, (IY+IND)
FD CB 05 9E	RES	3, (IY+IND)
FD CB 05 A6	RES	4, (IY+IND)

FD CB 05 AE	RES	5, (IY+IND)
FD CB 05 B6	RES	6, (IY+IND)
FD CB 05 BE	RES	7, (IY+IND)
FD CB 05 C6	SET	0, (IY+IND)
FD CB 05 CE	SET	1, (IY+IND)
FD CB 05 D6	SET	2, (IY+IND)
FD CB 05 DE	SET	3, (IY+IND)
FD CB 05 E6	SET	4, (IY+IND)
FD CB 05 EE	SET	5, (IY+IND)
FD CB 05 F6	SET	6, (IY+IND)
FD CB 05 FE	SET	7, (IY+IND)
FD E1	POP	IY
FD E3	EX	(SP), IY
FD E5	PUSH	IY
FD E9	JP	(IY)
FD F9	LD	SP, IY
FE 20	CP	N
FF	RST	38H

Instrucţiunile limbajului de asamblare 8086

Convenţiile de notaţii sînt prezentate în tabelul **??**.

Cîmp	*Semnificaţie*
reg	cîmp de 3 biţi care, împreună cu bitul w (0 - octet, 1 - cuvînt), desemnează un registru
mod	cîmp de 2 biţi care desemnează modul de adresare a memoriei
r/m	cîmp de 3 biţi care desemnează registrele implicate în adresarea memoriei; dacă $mod = 11$, acest cîmp semnifică un registru, similar cu reg

Tabelul 1: Convenţii de notaţii utilizate în anexă

AAA - ASCII Adjust for Addition (ajustare ASCII pentru adunare)

Operaţie:

```
if ((AL) & 0FH)>9 or (AF)= 1 then
   (AL)=(AL)+6
   (AH)=(AH)+1
   (AF)=1
(CF)=(AF)
(AL)=(AL) & 0FH
```

Descriere:

Modifică registrul AL într-o valoare de număr zecimal neîmpachetat. Cei mai semnificativi patru biţi ai registrului AL sînt schimbaţi în 0.

Codificare: 00110111

Indicatori: afectaţi - AF, CF, nedefiniţi - OF, PF, SF, ZF

AAD - ASCII Adjust for Division (ajustare ASCII pentru împărţire)

Operaţie:

```
(AL)=(AH)*0AH+(AL)
(AH)=0
```

Descriere:

Realizează o ajustare pentru împărţitorul din AL înaintea unei instrucţiuni de împărţire a doi operanzi zecimali neîmpachetaţi, astfel că rezultatul va fi un cît zecimal neîmpachetat. Ajustarea constă în înmulţirea cu 10 a octetului mai semnificativ al lui AH şi adunarea acestuia la AL.

Codificare: 11010101 00001010

Indicatori: afectaţi - PF, SF, ZF, nedefiniţi - AF, CF, OF

AAM - ASCII Adjust for Multiply (ajustare ASCII pentru înmulţire)

Operaţie:

```
(AH)=(AL)/0AH
(AL)=(AL)%0AH
```

Descriere:

Realizează o corecţie a rezultatului din AX după înmulţirea a doi operanzi zecimali neîmpachetaţi, producînd un produs zecimal neîmpachetat. Conţinutul lui AH este înlocuit cu rezultatul împărţirii lui AL cu 10. Apoi, conţinutul lui AL este înlocuit cu restul acestei împărţiri.

Codificare: 11010100 00001010

Indicatori: afectaţi - PF, SF, ZF, nedefiniţi - AF, CF, OF

AAS - ASCII Adjust for Subtraction (ajustare ASCII pentru scădere)

Operaţie:

```
if ((AL)&0FH)>9 or (AF)=1 then
   (AL)=(AL)-6
   (AH)=(AH)-1
   (AF)=1
(CF)=(AF)
(AL)=(AL)&0FH
```

Descriere:

Realizează o corecţie a rezultatului din AL în urma scăderii a doi operanzi zecimali, producînd o diferenţă zecimală neîmpachetată. Dacă jumătatea mai puţin semnificativă a lui AL este mai mică decît 9, sau dacă AF=1, atunci se scade 6 din AL şi 1 din AH. Vechea valoare a lui AL este înlocuită de un octet în care jumătatea superioară este 0 iar jumătatea inferioară este un număr determinat de scăderea anterioară.

Codificare: 00111111

Indicatori: afectaţi - AF, CF, nedefiniţi - OF, PF, SF, ZF

ADC - ADd with Carry (adună cu transport)

Operaţie:

```
if (CF)=1 then
   (DEST)=(LSRC)+(RSRC)+1
else
   (DEST)=(LSRC)+(RSRC)
```

Descriere:

ADC *destinaţie*, *sursă*

Adună doi operanzi, de dimensiune octet sau cuvînt, la care adaugă şi valoarea lui CF. Rezultatul este memorat în operandul destinaţie.

Codificare:

A) Operand în memorie sau registru cu operand în registru:

000100dw mod reg r/m

```
if d=1 then
   LSRC=REG, RSRC=EA, DEST=REG
else
   LSRC=EA, RSRC=REG, DEST=EA
```

B) Operand imediat la acumulator:

0001010w [data]

```
if w=0 then
  LSRC=AL, RSRC=data, DEST=AL
else
  LSRC=AX, RSRC=data, DEST=AX
```

C) Operand imediat la memorie sau în registru:

100000sw mod 010 r/m [data]

```
LRSC=EA, RSRC=data, DEST=EA
```

Exemple:

ADC DI, BX
ADC DX, MEM_WORD
ADC BETA[SI], DI
ADC AL, 3
ADC BETA[SI], 4
ADC CX, 432

Indicatori: afectaţi - AF, CF, OF, PF, SF, ZF

ADD - ADDition (adunare)

Operaţie:

```
(DEST)=(LSRC)+(RSRC)
```

Descriere:

ADD *destinaţie*, *sursă*

Adună doi operanzi, de dimensiune octet sau cuvînt. Rezultatul este memorat în operandul destinaţie. Dacă data imediată este octet, atunci acesta este extins la cuvînt (16 biţi).

Codificare:

A) Operand în registru sau memorie cu operand în registru:

000000dw mod reg r/m

```
if d=1 then
  LSRC=REG, RSRC=EA, DEST=REG
else
  LSRC=EA, RSRC=REG, DEST=EA
```

B) Operand imediat la acumulator:

0000010w [data]

```
if w=0 then
  LSRC=AL, RSRC=data, DEST=AL
else
  LSRC=AX, RSRC=data, DEST=AX
```

C) Operand imediat la memorie sau registru:

100000sw mod 000 r/m [data]

```
LRSC=EA, RSRC=data, DEST=EA
```

Exemple:

ADD DI, SI
ADD AX, BETA[SI]
ADD MEM_BYTE, BH
ADD AL, 3
ADD MEM_WORD, 48
ADD GAMMA[DI], IMM_84
ADD DX, 1776

Indicatori: afectaţi - AF, CF, OF, PF, SF, ZF

AND - logical AND ('şi' logic)

Operaţie:

```
(DEST)=(LSRC)&(RSRC)
(CF)=0
(OF)=0
```

Descriere:

AND *destinaţie*, *sursă*

Realizează funcţia logică 'şi' bit cu bit între cei doi operanzi. Indicatorii CF şi OF sînt resetaţi.

Codificare:

A) Operand în memorie sau registru cu operand registru:

001000dw mod reg r/m

```
if d=1 then
  LSRC=REG, RSRC=EA, DEST=REG
else
  LSRC=EA, RSRC=REG, DEST=EA
```

B) Operand imediat la acumulator:

0010010w [data]

```
if w=0 then
  LSRC=AL, RSRC=data, DEST=AL
else
  LSRC=AX, RSRC=data, DEST=AX
```

C) Operand imediat la memorie sau registru:

1000000w mod 100 r/m [data]

```
LSRC=EA, RSRC=data, DEST=EA
```

Exemple:

AND AX, BX
AND DH, ALPHA
AND ALPHA, AX
AND AL, 7AH
AND BL, 10011110B
AND MEM_WORD, 7A46

Indicatori: afectaţi - CF, OF, PF, SF, ZF, nedefiniţi - AF

CALL - CALL procedure (apel de procedură)

Operaţie:

```
if Inter-Segment then
  (SP)=(SP)-2
  ((SP)+1:(SP))=(CS)
  (CS)=SEG
(SP)=(SP)-2
((SP)+1:(SP))=(IP)
(IP)=DEST
```

Descriere:

CALL *nume_procedură*

Predă controlul unei proceduri după ce, în prealabil, s-a salvat pe stivă informaţia necesară reluării fluxului de instrucţiuni iniţial, la execuţia instrucţiunii RET. Asamblorul generează coduri diferite pentru instrucţiunea CALL, în funcţie de tipul procedurii, definit de către programator: NEAR sau FAR. Adresa procedurii se poate obţine direct din codul instrucţiunii, sau indirect, dintr-o locaţie de memorie sau registru la care se face referinţă în instrucţiune. Registrul IP este actualizat automat, înainte de a fi salvat pe stivă.

Pentru un *apel de procedură direct intra-segment*, SP este decrementat cu doi şi IP este salvat în stivă. Deplasamentul relativ al procedurii (pînă la $\pm 32k$) este adunat la IP.

Un *apel de procedură indirect intra-segment* se poate face printr-o locaţie de memorie sau registru. SP este decrementat cu doi şi IP este salvat în stivă. Offset-ul de 16 biţi al procedurii, în acelaşi segment de cod, se obţine dintr-o locaţie de memorie sau un registru şi înlocuieşte registrul IP.

Pentru un *apel de procedură direct inter-segment*, SP este decrementat cu doi şi CS este salvat în stivă. CS este înlocuit cu adresa de segment a procedurii, conţinută în instrucţiune. Apoi, SP este decrementat cu doi şi IP este salvat în stivă. IP este înlocuit cu offset-ul procedurii, conţinut în instrucţiune.

Un *apel de procedură indirect inter-segment* se poate face numai printr-o locaţie de memorie. SP este decrementat cu doi şi CS este salvat în stivă. CS este înlocuit cu conţinutul celui de-al doilea cuvînt al unui pointer dublucuvînt aflat în memorie, la care se face referire în instrucţiune. Apoi, SP este decrementat cu doi şi IP este salvat în stivă. IP este înlocuit cu conţinutul primului cuvînt al unui pointer dublucuvînt aflat în memorie, la care se face

referire în instrucţiune.

Codificare:

A) Intra-segment direct:

11101000 disp_low disp_high

`DEST=(EA)`

B) Intra-segment indirect:

11111111 mod 010 r/m

`DEST=(EA)`

C) Inter-segment direct:

10011010 offset-low offset-high seg-low seg-high

`DEST=offset, SEG=seg`

D) Inter-segment indirect:

11111111 mod 011 r/m

`(DEST)=(EA), SEG=(EA+2)`

Exemple:

```
CALL NEAR_LABEL
CALL WORD PTR [BX]
CALL BX
CALL WORD PTR VAR[BP][SI]
CALL FAR_PROC
CALL DWORD PTR [BX]
CALL DWORD PTR VAR[SI]
CALL MEM_DOUBLE
```

CBW - Convert Byte to Word (converteşte octet la cuvînt)

Operaţie:

```
if AL<80h then
   AH=0
else
   AH=FFh
```

Descriere:

Realizează o extensie de semn a lui AL în AH.

Codificare: 10011000

CLC - CLear Carry flag (şterge indicatorul de transport CF)

Operaţie:

`(CF)=0`

Descriere:

Indicatorul CF este resetat. Nu sînt afectaţi ceilalţi indicatori.

Codificare: 11111000

Indicatori: afectaţi - CF

CLD - CLear Direction flag (şterge indicatorul de direcţie DF)

Operaţie:

`(DF)=0`

Descriere:

Indicatorul DF este resetat, determinînd ca la operaţiile pe şiruri să se autoincrementeze pointerii de operaţie.

Codificare: 11111100

Indicatori: afectaţi - DF

CLI - CLear Interrupt flag (şterge indicatorul de validare a întreruperilor IF)

Operaţie:

`(IF)=0`

Descriere:

Indicatorul IF este resetat, dezactivînd întreruperile externe care apar pe pinul INTR.

Codificare: 11111010

Indicatori: afectaţi - IF

CMC - CoMplement Carry flag (complementează indicatorul CF)

Operaţie:

```
if (CF)=0 then
   (CF)=1
else
   (CF)=0
```

Descriere:

Complementează logic CF.

Codificare: 11110101

Indicatori: afectaţi - CF

CMP - CoMPare (compară)

Operaţie:

```
(LSRC)-(RSRC)
```

Descriere:

CMP *destinaţie*, *sursă*

Realizează scăderea celor doi operanzi determinînd afectarea indicatorilor, fără a memora rezultatul.

Codificare:

A) Operand din memorie sau registru cu operand registru:

001110dw mod reg r/m

```
if d=1 then
   LSRC=REG, RSRC=EA
else
   LSRC=EA, RSRC=REG
```

B) Operand imediat cu acumulator:

0011110w [data]

```
if w=0 then
  LSRC=AL, RSRC=data
else
  LSRC=AX, RSRC=data
```

C) Operand imediat cu operand din memorie sau registru:

100000sw mod 111 r/m [data]

```
LSRC=EA, RSRC=data
```

Exemple:

CMP AX, DX
CMP ALPHA[DI], DX
CMP AX, GAMMA[BP][SI]
CMP AL, 6
CMP BH, 7
CMP [BX][DI], 6ACEH

Indicatori: afectaţi - AF, CF, OF, PF, SF, ZF

CMPS - CoMPare String (compară şir)

Operaţie:

```
(LSRC)-(RSRC)
if (DF)=0 then
   (SI)=(SI)+DELTA
   (DI)=(DI)+DELTA
else
   (SI)=(SI)-DELTA
   (DI)=(DI)-DELTA
```

Descriere:

CMPS *şir_destinaţie*, *şir_sursă*

Scade octetul adresat de DI din cel adresat de SI şi afectează indicatorii, fără a memora rezultatul. Este un mod de a compara două şiruri. Folosind prefixe de repetiţie, se poate determina după al cîtelea element al şirurilor elementele sînt neegale, stabilind o ordine.

Numele operanzilor din instrucţiunea CMPS sînt utilizate numai de asamblor pentru a verifica tipul şi accesul, utilizînd registrele curente de segment. CMPS foloseşte numai SI şi DI pentru a pointa la locaţiile ce sînt comparate.

Codificare:

0010011w

```
if w=0 then
   LSRC=(SI), RSRC=(DI),
   DELTA=1
else
   LSRC=(SI)+1:(SI),
   (RSRC)=(DI)+1:(DI),
   DELTA=2
```

Exemple:

MOV SI, OFFSET STRING1
MOV DI, OFFSET STRING2
CMPS STRING1, STRING2

Indicatori: afectaţi - AF, CF, OF, PF, SF, ZF

CWD - Convert Word to Doubleword (converteşte cuvînt la dublucuvînt)

Operaţie:

```
if (AX)<8000H then
   (DX)=0
else
   (DX)=ffffH
```

Descriere:

Realizează o extensie de semn a lui AX în DX.

Codificare: 10011001

DAA - Decimal Adjust for Addition (ajustare zecimală pentru adunare)

Operaţie:

```
if (AL)&0FH>9 or (AF)=1 then
   (AL)=(AL)+6
   (AF)=1
if (AL)>9FH or (CF)=1 then
   (AL)=(AL)+60H
   (CF)=1
```

Descriere:

Realizează o corecţie a rezultatului adunării a doi operanzi zecimali împachetaţi, producînd o sumă zecimală împachetată.

Codificare: 10100111

Indicatori: afectaţi - AF, CF, PF, SF, ZF, nedefiniţi - OF

DAS - Decimal Adjust for Subtraction (ajustare zecimală pentru scădere)

Operaţie:

```
if (AL)&0FH>9 or (AF)=1 then
   (AL)=(AL)-6
   (AF)=1
if (AL)>9FH or (CF)=1 then
   (AL)=(AL)-60H
   (CF)=1
```

Descriere:

Realizează o corecţie a rezultatului scăderii a doi operanzi zecimali împachetaţi, producînd o diferenţa zecimală împachetată.

Codificare: 00101111

Indicatori: afectaţi - AF, CF, PF, SF, ZF, nedefiniţi - OF

DEC - DECrement (decrementare)

Operaţie:

```
DEST=DEST-1
```

Descriere:

Scade 1 din operandul destinaţie.

Codificare:

A) Operand registru:

01001 reg

```
DEST=REG
```

B) Operand registru sau memorie:

1111111w mod 001 r/m

```
DEST=EA
```

Exemple:

DEC DI
DEC AX
DEC BL
DEC MEM-WORD
DEC MEM[BX][SI]

Indicatori: afectaţi - AF, OF, PF, SF, ZF

DIV - DIVide (împărţire)

Operaţie:

```
(temp)=(NUMR)
if (temp)/(DIVR)>MAX then
   (QUO),(REM) undefined
   (SP)=(SP)-2
   ((SP)+1:(SP))=FLAGS
   (IF)=0
   (TF)=0
   (SP)=(SP)-2
   ((SP)+1:(SP))=(CS)
   (CS)=(2)
```

```
   (SP)=(SP)-2
   (IP)=(0)
else
   (QUO)=(temp)/(DIVR)
   (REM)=(temp)%(DIVR)
```

Descriere:

DIV *sursă*

Realizează o împărţire fără semn a acumulatorului la operandul sursă. Dacă operandul sursă este octet, atunci deîmpărţitul este în registrele AH şi AL. Cîtul este returnat în AL iar restul în AH. Dacă operandul sursă este cuvînt, atunci deîmpărţitul este în registrele AX şi DX. Cîtul este returnat în AX iar restul în DX. Cînd cîtul nu se poate reprezenta în locaţia de destinaţie (împărţire prin 0), atunci cîtul şi restul sînt nedefinite şi se generează o întrerupere de nivel 0.

Codificare:

1111011w mod 110 r/m

```
if w=0 then
   NUMR=AX, DIVR=EA,
   QUO=AL, REM=AH, MAX=ffH
else
   NUMR=DX:AX, DIVR=EA, QUO=AX,
   REM=DX, MAX=ffffH
```

Exemple:

DIV CL
DIV BX
DIV ALPHA
DIV TABLE[SI]

Indicatori: afectaţi - nici unul, nedefiniţi - AF, CF, OF, PF, SF, ZF

ESC - ESCape (instrucţiune pentru coprocesor)

Operaţie:

```
if mod<>11 then
   data bus=(EA)
```

Descriere:

Implementează un mecanism prin care alte procesoare (coprocesoare) pot primi instrucţiuni de la 8086. Procesorul 8086 nu execută altă operatie decît de a accesa un operand din memorie şi de a-l plasa pe magistrală.

Codificare:

11011 x mod x r/m

Exemple:

ESC EXTERNAL_OPCODE, ADDRESS

HLT - HaLT (oprire)

Operaţie:

nici una

Descriere:

Determină procesorul 8086 să intre în starea *halt*. Starea *halt* este părăsită în cazul apariţiei unei întreruperi externe sau reset.

Codificare: 11110100

IDIV - Integer DIVide (împărţire întreagă)

Operaţie:

```
(temp)=(NUMR)
if ((temp)/(DIVR)>MAX and
   (temp)/(DIVR)>0) or
   ((temp)/(DIVR)<0 and
   (temp)/(DIVR)<0-MAX-1) then
   (QUO),(REM) undefined
   (SP)=(SP)-2
   ((SP)+1:(SP))=FLAGS
   (IF)=0
   (TF)=0
   (SP)=(SP)-2
   ((SP)+1:(SP))=(CS)
   (CS)=(2)
   (SP)=(SP)-2
   ((SP+1:(SP))=(IP)
   (IP)=(0)
else
   (QUO)=(temp)/(DIVR)
   (REM)=(temp)%(DIVR)
```

Descriere:

IDIV *sursă*

Realizează o împărţire cu semn a acumulatorului la operandul sursă. Dacă operandul sursă este octet, atunci deîmpărţitul este în registrele AH şi AL. Cîtul este returnat în AL iar restul în AH. Cîtul este în domeniul $+127(7FH)$ şi $-127(81H)$. Dacă operandul sursă este cuvînt, atunci deîmpărţitul este în registrele AX şi DX. Cîtul este returnat în AX iar restul în DX. Cîtul este în domeniul $+32.767(7FFFH)$ şi $-32.767(8001H)$. Cînd cîtul nu se poate reprezenta în locaţia de destinaţie (împărţire prin 0), atunci cîtul şi restul sînt nedefinite şi se generează o întrerupere de nivel 0.

Codificare:

1111011w mod 111 r/m

```
if w=0 then
   NUMR=AX, DIVR=EA, QUO=AL,
   REM=AH, MAX=7fH
else
   NUMR=DX:AX, DIVR=EA, QUO=AX,
   REM=DX, MAX=7fffH
```

Exemple:

IDIV BL
IDIV CX
IDIV DIVISOR_BYTE[SI]
IDIV [BX], DIVISOR_WORD

Indicatori: afectaţi - nici unul, nedefiniţi - AF, CF, OF, PF, SF, ZF

IMUL - Integer MULtiply (înmulţire întreagă)

Operaţie:

```
(DEST)=(LSRC)*(RSRC)
if (EXT)=sign-extension of (LOW) then
   (CF)=0
else
   (CF)=1;
(OF)=(CF)
```

Descriere:

IMUL *sursă*

Realizează înmulţirea cu semn a acumulatorului cu operandul sursă. Rezultatul este returnat în (AH:AL) pentru operand octet, şi (DX:AX) pentru operand cuvînt. CF şi OF sînt setaţi dacă jumătatea superioară a rezultatului nu este o extensie de semn a jumătăţii inferioare.

Codificare:

1111011w mod 101 r/m

```
if w=0 then
   LSRC=AL, RSRC=EA,
   DEST=AX, EXT=AH, LOW=AL
else
   LSRC=AX, RSRC=EA, DEST=DX:AX,
   EXT=DX, LOW=AX
```

Exemple:

IMUL CL
IMUL BX
IMUL RATE_BYTE
IMUL RATE_WORD[BP][DI]

Indicatori: afectaţi - CF, OF, nedefiniţi - AF, PF, SF, ZF

IN - INput byte or word (intrare de la port)

Operaţie:

```
(DEST)=(SRC)
```

Descriere:

IN *acumulator, port*

Transferă un octet sau cuvînt de la un port de intrare în registrul AL sau AX. Portul este specificat ca o dată imediată (între 0 şi 255) sau indirect, prin registrul DX.

Codificare:

A) Port fix:

1110010w port

```
if w=0 then
   SRC=port, DEST=AL
else
   SRC=port+1:port, DEST=AX
```

B) Port variabil:

1110110w

```
if w=0 then
   SRC=(DX), DEST=AL
else
   SRC=(DX)+1:(DX), DEST=AX
```

Exemple:

IN AX, WORD_PORT
IN AL, BYTE_PORT
IN AX, DX
IN AL, DX

INC - INCrement (incrementare)

Operaţie:

```
(DEST)=(DEST)+1
```

Descriere:

INC *destinaţie*

Adună 1 la operandul destinaţie.

Codificare:

A) Operand registru:

01000 reg

```
DEST=REG
```

B) Operand registru sau memorie:

1111111w mod 000 r/m

```
DEST=EA
```

Exemple:

INC DI
INC AX
INC BL
INC MEM[BX][SI]

Indicatori: afectaţi - AF, OF, PF, SF, ZF

INT - INTerrupt (întrerupere)

Operaţie:

```
(SP)=(SP)-2
((SP)+1:(SP))=FLAGS
(IF)=0
(TF)=0
(SP)=(SP)-2
((SP)+1:(SP))=(CS)
(CS)=(TYPE*4+2)
(SP)=(SP)-2
((SP)+1:(SP))=(IP)
(IP)=(TYPE*4)
```

Descriere:

INT *tip_întrerupere*

Salvează starea programului şi transferă controlul cu un apel indirect unui vector de întrerupere. Operandul trebuie să fie o dată imediată, nu o referinţă la un registru sau memorie.

Codificare: 1100110v [tip]

```
if v=0 then
   TYPE=3
else
   TYPE=tip
```

Exemple:

INT 3
INT 67

Indicatori: afectaţi - IF, TF

INTO - INTerrupt on Overflow (întrerupere în caz de depăşire)

Operaţie:

```
if (OF)=1 then
   (SP)=(SP)-2
   ((SP)+1:(SP))=FLAGS
   (IF)=0
   (TF)=0
   (SP)=(SP)-2
   ((SP)+1:(SP))=(CS)
   (CS)=(12h)
   (SP)=(SP)-2
   ((SP)+1:(SP))=(IP)
   (IP)=(10h)
```

Descriere:

Salvează starea programului şi transferă controlul cu un apel indirect vectorului de întrerupere de la locaţia 4, dacă OF este setat.

Codificare: 11001110

IRET - Interrupt RETurn (întoarcere din întrerupere)

Operaţie:

```
(IP)=((SP)+1:(SP))
(SP)=(SP)+2
(CS)=((SP)+1:(SP))
(SP)=(SP)+2
FLAGS=((SP)+1:(SP))
(SP)=(SP)+2
```

Descriere:

Transferă controlul la adresa de întoarcere salvată de o operaţie anterioară de întrerupere. Se refac şi indicatorii.

Codificare: 11001111

Indicatori: afectaţi - toţi

JA/JNBE - Jump on Above / Jump on Not Below or Equal (salt dacă este mai mare / salt dacă nu este mai mic sau egal)

Operaţie:

```
if (CF)&(ZF)=0 then
   (IP)=(IP)+disp
```

Descriere:

Transferă controlul operandului ţintă dacă (CF) and (ZF) = 0.

Codificare: 01110110 disp

Exemple:

JA TARGET_LABEL
JNBE TARGET_LABEL

JAE/JNB - Jump on Above or Equal / Jump on Not Below (salt dacă este mai mare sau egal / salt dacă nu este mai mic)

Operaţie:

```
if (CF)=0 then
   (IP)=(IP)+disp
```

Descriere:

Transferă controlul operandului ţintă dacă (CF) = 0.

Codificare: 01110111 disp

Exemple:

JNB TARGET_LABEL
JAE TARGET_LABEL

JNAE/JB - Jump on Not Above or Equal / Jump on Below (salt dacă nu este mai mare sau egal / salt dacă este mai mic)

Operaţie:

```
if (CF)=1 then
   (IP)=(IP)+disp
```

Descriere:

Transferă controlul operandului ţintă dacă (CF) = 1.

Codificare: 01110010 disp

Exemple:

JB TARGET_LABEL
JNAE TARGET_LABEL

JNA/JBE - Jump on Not Above / Jump on Below or Equal (salt dacă nu este mai mare / salt dacă este mai mic sau egal)

Operaţie:

```
if (CF) or (ZF)=1 then
   (IP)=(IP)+disp
```

Descriere:

Transferă controlul operandului ţintă dacă (CF) or (ZF) = 1.

Codificare: 01110110 disp

Exemple:

JBE TARGET_LABEL
JNA TARGET_LABEL

JC - Jump on Carry (salt dacă s-a generat depăşire)

Operaţie:

```
if (CF)=1 then
   (IP)=(IP)+disp
```

Descriere:

Transferă controlul operandului ţintă dacă (CF) = 1.

Codificare: 01110010 disp

Exemple:

JC TARGET_LABEL

JCXZ - Jump if CX Zero (salt dacă CX este zero)

Operaţie:

```
if (CX)=0 then
   (IP)=(IP)+disp
```

Descriere:

JCXZ *etichetă*

Transferă controlul operandului ţintă dacă registrul CX este zero. Această instrucţiune este utilă la începutul unei bucle de program pentru a sări bucla dacă CX este zero.

Codificare: 11100011 disp

Exemple:

JCXZ TARGET_LABEL

JE/JZ - Jump on Equal / Jump on Zero (salt dacă este egal / salt dacă este zero)

Operaţie:

```
if (ZF)=1 then
   (IP)=(IP)+disp
```

Descriere:

Transferă controlul operandului ţintă dacă (ZF) = 1.

Codificare: 01110100 disp

Exemple:

JZ LABEL_ZERO

JG/JNLE - Jump on Greater / Jump on Not Less or Equal (salt dacă este mai mare / salt dacă nu este mai mic sau egal)

Operaţie:

```
if ((SF)=(OF)) or (ZF)=0 then
   (IP)=(IP)+disp
```

Descriere:

Transferă controlul operandului ţintă dacă ((SF) xor (OF)) or (ZF) = 0.

Codificare: 01111111 disp

Exemple:

JG TARGET_LABEL
JNLE TARGET_LABEL

JNL/JGE - Jump on Not Less / Jump on Greater or Equal (salt dacă nu este mai mic / salt dacă este mai mare sau egal)

Operaţie:

```
if (SF)=(OF) then
   (IP)=(IP)+disp
```

Descriere:

Transferă controlul operandului ţintă dacă ((SF) xor (OF)) = 0.

Codificare: 01111101 disp

Exemple:

JGE TARGET_LABEL
JNL TARGET_LABEL

JL/JNGE - Jump on Less / Jump on Not Greater or Equal (salt dacă este mai mic / salt dacă nu este mai mare sau egal)

Operaţie:

```
if (SF)<>(OF) then
   (IP)=(IP)+disp
```

Descriere:

Transferă controlul operandului ţintă dacă ((SF) xor (OF)) = 1.

Codificare: 01111100 disp

Exemple:

JNGE TARGET_LABEL
JL TARGET_LABEL

JLE/JNG - Jump on Less or Equal / Jump on Not Greater (salt dacă este mai mic sau egal / salt dacă nu este mai mare)

Operaţie:

```
if ((SF)<>(OF)) or ((ZF)=1) then
   (IP)=(IP)+disp
```

Descriere:

Transferă controlul operandului ţintă dacă ((SF) xor (OF)) or (ZF) = 1.

Codificare: 01111110 disp

Exemple:

JNG TARGET_LABEL
JLE TARGET_LABEL

JMP - JuMP unconditionally (salt necondiţionat)

Operaţie:

```
if inter-segment then
   (CS)=SEG
(IP)=DEST
```

Descriere:

JMP *ţintă*

Transferă necondiţionat controlul la adresa ţintă. Spre deosebire de instrucţiunea CALL, JMP nu salvează nimic în stivă.

Pentru un *salt direct intra-segment*, deplasamentul relativ este adunat la IP. Dacă asamblorul constată că deplasamentul relativ este mai mic de 127 de octeţi, generează automat o instrucţiune de doi octeţi denumită *salt scurt*. Altfel, se generează o instrucţiune obişnuită, cu un deplasament relativ de dimensiuni maxime $\pm 32k$.

Un *salt indirect intra-segment* se poate face printr-o locaţie de memorie sau registru. Adresa de salt de 16 biţi, în acelaşi segment de cod, se obţine dintr-o locaţie de memorie sau un registru şi înlocuieşte registrul IP.

La un *salt direct inter-segment*, CS şi IP sînt înlocuite cu valori conţinute în instrucţiune.

Un *salt indirect inter-segment* se poate face numai printr-o locaţie de memorie. Cel de-al doilea cuvînt al unui pointer dublucuvînt aflat în memorie, la care se face referire în instrucţiune înlocuieşte registrul CS, iar primul registul IP.

Codificare:

A) Intra-segment direct:

11101001 disp-low disp-high

`DEST=(IP)+disp`

B) Intra-segment direct scurt:

11101011 disp

`DEST=(IP)+disp`

C) Inter-segment direct:

11101010 offset-low offset-high seg-low seg-high

`DEST=offset, SEG=seg`

D) Inter-segment indirect:

11111111 mod 101 r/m

`(DEST)=(EA), SEG=(EA+2)`

Exemple:

JMP NEAR_LABEL
JMP SHORT NEAR_LABEL
JMP FAR_PROC
JMP FAR PTR NEAR_LABEL
JMP DWORD PTR [BX]
JMP DWORD PTR VAR[SI]
JMP MEM_DOUBLE

JNC - Jump on Not Carry (salt dacă nu s-a generat depăşire)

Operaţie:

```
if (CF)=0 then
   (IP)=(IP)+disp
```

Descriere:

Transferă controlul operandului ţintă dacă (CF) = 0.

Codificare: 01110011 disp

Exemple:

JNC NO_CARRY

JNE/JNZ - Jump on Not Equal / Jump on Not Zero (salt dacă nu este egal / salt dacă nu este zero)

Operaţie:

```
if (ZF)=0 then
   (IP)=(IP)+disp
```

Descriere:

Transferă controlul operandului ţintă dacă (ZF) = 0.

Codificare: 01110101 disp

Exemple:

JNZ TARGET_LABEL

JNO - Jump on Not Overflow (salt dacă nu există overflow)

Operaţie:

```
if (OF)=0 then
   (IP)=(IP)+disp
```

Descriere:

Transferă controlul operandului ţintă dacă (OF) = 0.

Codificare: 01110001 disp

Exemple:

JNO TARGET_LABEL

JNS - Jump on Not Sign (salt dacă indicatorul de semn este zero)

Operaţie:

```
if (SF)=0 then
   (IP)=(IP)+disp
```

Descriere:

Transferă controlul operandului ţintă dacă (SF) = 0.

Codificare: 01111001 disp

Exemple:

JNS TARGET_LABEL

JNP/JPO - Jump on Not Parity / Jump on Parity Odd (salt dacă indicatorul de paritate este zero / salt dacă paritatea este impară)

Operaţie:

```
if (PF)=0 then
   (IP)=(IP)+disp
```

Descriere:

Transferă controlul operandului ţintă dacă (PF) = 0.

Codificare: 01111011 disp

Exemple:

JPO TARGET_LABEL

JO - Jump on Overflow (salt dacă există overflow)

Operaţie:

```
if (OF)=1 then
   (IP)=(IP)+disp
```

Descriere:

Transferă controlul operandului ţintă dacă (OF) = 1.

Codificare: 01110000 disp

Exemple:

JO TARGET_LABEL

JP/JPE - Jump on Parity / Jump on Parity Even (salt dacă indicatorul de paritate este unu / salt dacă paritatea este pară)

Operaţie:

```
if (PF)=1 then
   (IP)=(IP)+disp
```

Descriere:

Transferă controlul operandului ţintă dacă (PF) = 1.

Codificare: 01111010 disp

Exemple:

JP TARGET_LABEL

JS - Jump on Sign (salt dacă indicatorul de semn este unu)

Operaţie:

```
if (SF)=1 then
   (IP)=(IP)+disp
```

Descriere:

Transferă controlul operandului ţintă dacă (SF) = 1.

Codificare: 01111000 disp

Exemple:

JS TARGET_LABEL

LAHF - Load AH from Flags (încarcă AH cu registrul de indicatori)

Operaţie:

```
(AH)=(SF):(ZF):X:(AF):X:(PF):X:(CF)
```

Descriere:

Copiază registrul indicatorilor de stare (SF, ZF, AF, PF şi CF) în biţii corespunzători ai registrului AH. Biţii 1, 3 şi 5 ai lui AH rămîn nedefiniţi.

Codificare: 10011111

LDS - Load pointer using DS (încarcă pointer în segmentul de date)

Operaţie:

```
(REG)=(EA)
(DS)=(EA+2)
```

Descriere:

LDS *destinaţie*, *sursă*

Transferă o variabilă pointer de 32 de biţi la un operand sursă în registrul destinaţie şi registrul DS. Offset-ul poate fi transferat în orice registru de 16 biţi (general, pointer, index) cu excepţia registrelor de segment.

Codificare: 11000101 mod reg r/m

Exemple:

LDS BX, ADRR_TABLE[SI]
LDS SI, NEWSEG[BX]

LEA - Load Effective Address (încarcă adresa efectivă)

Operaţie:

```
(REG)=(EA)
```

Descriere:

LEA *destinaţie*, *sursă*

Transferă offset-ul operandului sursă în registrul destinaţie. Operandul sursă trebuie să fie un operand în memorie. Operandul destinaţie poate fi orice registru de 16 biţi general, pointer sau index.

Codificare: 10001101 mod reg r/m

Exemple:

LEA BX, VARIABLE_7
LEA DX, BETA[BX][SI]
LEA AX, [BP][DI]

LES - Load pointer using ES (încarcă pointer în segmentul de date auxiliar)

Operaţie:

```
(REG)=(EA)
(ES)=(EA+2)
```

Descriere:

LES *destinaţie*, *sursă*

Transferă o variabilă pointer de 32 de biţi la un operand sursă în registrul destinaţie şi registrul ES. Offset-ul poate fi transferat în orice registru de 16 biţi (general, pointer, index) cu

excepţia registrelor de segment.

Codificare: 11000100 mod reg r/m

Exemple:

LES BX, ADRR_TABLE[SI]
LES SI, NEWSEG[BX]

LODS - LOaD String (încarcă şir)

Operaţie:

```
(DEST)=(SRC)
if (DF)=0 then
   (SI)=(SI)+DELTA
else
   (SI)=(SI)-DELTA
```

Descriere:

LODS *şir_sursă*

Transferă un operand de tip octet sau cuvînt, pointat de DS:SI, în registrul AL sau AX şi actualizează registrul index SI cu DELTA. Modul de actualizare depinde de tipul datelor din şir şi de valoarea indicatorului de direcţie DF. În mod obişnuit, această operaţie nu se repetă, însă poate fi folosită împreună cu prefixe de repetiţie. Operandul instrucţiunii LODS este folosit numai de asamblor pentru a verifica tipul şi accesibilitatea datelor din şir.

Codificare: 1010110w

```
if w=0 then
   SRC=(SI), DEST=AL, DELTA=1
else
   SRC=(SI)+1:(SI), DEST=AX, DELTA=2
```

Exemple:

LODS BYTE_STRING
REP LODS WORD_STRING

LOOP - LOOP (buclă)

Operaţie:

```
(CX)=(CX)-1
if (CX)<>0 then
   (IP)=(IP)+disp
```

Descriere:

LOOP *etichetă*

Decrementează CX cu 1 şi transferă controlul la adresa etichetei operand dacă CX nu este zero. Eticheta ţintă trebuie să fie între -128 şi +127.

Codificare: 11100010

Exemple:

LOOP AGAIN

LOOPE/LOOPZ - LOOP while Equal / LOOP while Zero (buclă cît timp este egal / buclă cît timp este zero)

Operaţie:

```
(CX)=(CX)-1
if (CX)<>0 and (ZF)=1 then
   (IP)=(IP)+disp
```

Descriere:

LOOPE/LOOPZ *etichetă*

Decrementează CX cu 1 şi transferă controlul la adresa etichetei dacă CX nu este zero şi dacă indicatorul ZF este 1. Eticheta ţintă trebuie să fie între -128 şi +127.

Codificare: 11100001 disp

Exemple:

LOOPE AGAIN

LOOPNE/LOOPNZ - LOOP while Not Equal / LOOP while Not Zero (buclă cît timp nu este egal / buclă cît timp nu este zero)

Operaţie:

```
(CX)=(CX)-1
if (CX)<>0 and (ZF)=0 then
   (IP)=(IP)+disp
```

Descriere:

LOOPNZ/LOOPNE *etichetă*

Decrementează CX cu 1 şi transferă controlul la adresa etichetei dacă CX nu este zero şi dacă

ZF este 0. Eticheta ţintă trebuie să fie între -128 şi +127.

Codificare: 11100000 disp

Exemple:

LOOPNE AGAIN

MOV - MOVe (mutare)

Operaţie:

```
(DEST)=(SRC)
```

Descriere:

MOV *destinaţie*, *sursă*

Copiază un octet sau un cuvînt din operandul sursă în operandul destinaţie.

Codificare:

A) În memorie de la acumulator:

1010001w addr-low addr-high

```
if w=0 then
   SRC=AL, DEST=addr
else
   SRC=AX, DEST=addr + 1:addr
```

B) În acumulator din memorie:

1010000w addr-low addr-high

```
if w=0 then
   SRC=addr, DEST=AL
else
  SRC=addr+1:addr, DEST=AX
```

C) În registru de segment din operand de tip memorie/registru:

10001110 mod 0 reg r/m

```
if reg<>01 then
   SRC=EA,DEST=REG
else
   undefined operation
```

D) În registru/memorie din registru segment:

10001100 mod 0 reg r/m

```
SRC=REG, DEST=EA, (DEST)=(SRC)
```

E) Între registru şi registru / memorie:

100010dw mod reg r/m addr_low addr_high

Ultimii doi octeţi lipsesc în cazul transferului registru la registru (MOV CX, DX) şi în cazul modurilor de adresare în care sînt folosite numai registre în mod indirect (MOV [BX][SI], DX).

```
if d=1 then
   SRC=EA, DEST=REG
else
   SRC=REG, DEST=EA
```

F) În registru din data imediată:

1011w reg data data-high

```
SRC=data, DEST=REG
```

G) În memorie / registru din data imediată:

1100011w mod 000 r/m data data-high

```
SRC=data, DEST=EA
```

Exemple:

MOV ALPHA_MEM, AX
MOV GAMMA_BYTE, AL
MOV CS:DATUM_BYTE, AL
MOV ES:ARRAY[BX][SI], AX
MOV AX, BETA_MEM
MOV AL, GAMMA_BYTE
MOV AX, ES:ARRAY[BX][SI]
MOV AL, SS:OTHER_BYTE
MOV ES, DX
MOV DS, AX
MOV SS, BX
MOV ES, SS:NEW_WORD[DI]
MOV DX, DS
MOV BX, ES
MOV ARRAY[BX][SI], SS
MOV BETA_MEM_WORD, DS
MOV GAMMA,CS
MOV AX, BX
MOV CL, DH
MOV CX, DI
MOV AX, MEM_VALUE
MOV DX, ARRAY[SI]
MOV DI, MEM[BX][DI]
MOV ARRAY[DI], DX
MOV MEM_VALUE, AX

MOV [BX][SI], DI
MOV AX, 77
MOV BX, VALUE_14_IMM
MOV SI, EQU_VAL_9
MOV DI, 618
MOV ARRAY[BX][SI], DATA_4
MOV MEM_BYTE, IMM_BYTE_3
MOV BYTE PTR [DI], 66
MOV MEM_WORD, 1999
MOV BX, 84
MOV DS:MEM_WORD[BP], 3989

MOVS - MOVe String (mutare şir)

Operaţie:

```
(DEST)=(SRC)
if (DF)=0 then
   (SI)=(SI)+DELTA
   (DI)=(DI)+DELTA
else
   (SI)=(SI)-DELTA
   (DI)=(DI)-DELTA
```

Descriere:

MOVS *şir_destinaţie*, *şir_sursă*

Transferă un octet sau un cuvînt din şirul sursă pointat de DS:SI la adresa pointată de ES:DI şi ajustează registrele SI şi DI cu DELTA. Împreună cu prefixe de repetiţie, această instrucţiune transferă blocuri în memorie.

Codificare: 1010010w

```
if w=0 then
   SRC=(SI), DEST=(DI), DELTA=1
else
   SRC=(SI)+1:(SI), DEST=(DI)+1:(DI),
   DELTA=2
```

Exemple:

MOV SI, OFFSET SOURCE
MOV DI, OFFSET DEST
MOV CX, LENGTH SOURCE
REP MOVS DEST, SOURCE

MUL - MULtiply (înmulţire)

Operaţie:

```
(DEST)=(LSRC)*(RSRC)
if (EXT)=0 then
   (CF)=0
else
   (CF)=1
(OF)=(CF)
```

Descriere:

MUL *şir_sursă*

Realizează înmulţirea întreagă a operandului sursă cu acumulatorul. Dacă sursa este octet, atunci înmulţitorul este registrul AL, iar produsul este returnat în AH şi AL. Dacă sursa este cuvînt, atunci înmulţitorul este registrul AX, iar produsul este returnat în DX şi AX.

Codificare: 111101w mod 100 r/m

```
if w=0 then
   LSRC=AL, RSRC=EA, DEST=AX, EXT=AH
else
   LSRC=AX, RSRC=EA, DEST=DX:AX, EXT=DX
```

Exemple:

MUL BL
MUL CX
MUL MONTH[SI]

Indicatori: afectaţi - CF, OF, nedefiniţi - AF, PF, SF, ZF

NEG - NEGate (neagare aritmetică)

Operaţie:

```
(EA)=SRC-(EA)
(EA)=(EA)+1
```

Descriere:

NEG *destinaţie*

Scade operandul destinaţie din zero şi îi adaugă unu. Aceste operaţii formează complementul faţă de doi a operandului specificat.

Codificare: 01111011w mod 011 r/m

```
if w=0 then
   SRC=0FFH
else
   SRC=0FFFFH
```

Exemple:

NEG AL
NEG MULTIPLIER

Indicatori: afectaţi - AF, CF, OF, PF, SF, ZF

NOP - NO Operation (nici o operaţie)

Operaţie:

nici una

Descriere:

Nu se execută nici o operaţie. Nu sînt afectaţi indicatorii.

Codificare: 10010000

NOT - Logical NOT (negare logică)

Operaţie:

```
(EA)=SRC-(EA)
```

Descriere:

NOT *destinaţie*

Complementează logic bit cu bit operandul destinaţie.

Codificare: 1111011w mod 010 r/m

```
if w=0 then
   SRC=0FFh
else
   SRC=0FFFFh
```

Exemple:

NOT AX
NOT CARACTER

OR - Logical OR ('sau' logic)

Operaţie:

```
(DEST)=(LSRC) or (RSRC)
(CF)=0
(OF)=0
```

Descriere:

OR *destinaţie*, *sursă*

Realizează funcţia logică 'sau' bit cu bit între cei doi operanzi. Indicatorii CF şi OF sînt resetaţi.

Codificare:

A) Operand din memorie sau registru cu operand registru:

000010dw mod reg r/m

```
if d=1 then
   LSRC=REG, RSRC=EA, DEST=REG
else
   LSRC=EA, RSRC=REG, DEST=EA
```

B) Operand imediat cu acumulator:

0000110w [data]

```
if w=0 then
   LSRC=AL, RSRC=data, DEST=AL
else
   LSRC=AX, RSRC=data, DEST=AX
```

C) Operand imediat cu operand din memorie sau registru:

1000000w mod 001 r/m [data]

```
LSRC=EA, RSRC=data, DEST=EA
```

Exemple:

OR AL, BL
OR SI, DX
OR AX, MEM_WORD
OR SI, ALPHA[BX][SI]
OR BETA[BX][SI]
OR MEM_BYTE, DH
OR AL, 0F6H
OR AX, 23F6H
OR AH, 0F6H
OR CL, 37
OR MEM_BYTE, 3DH
OR ALPHA[DI], VAL_EQUD_33H

Indicatori: afectaţi - CF, OF, PF, SF, ZF, nedefiniţi - AF

OUT - OUTput (ieşire la port)

Operaţie:

```
(DEST)=(SRC)
```

Descriere:

OUT *port*, *acumulator*

Transferă un octet sau un cuvînt din registrul AL sau AX la un port de ieşire. Portul este specificat ca o dată imediată (între 0 şi 255) sau indirect, prin registrul DX.

Codificare:

A) Port fix:

1110011w port

```
if w=0 then
   SRC=AL, DEST=port
else
  SRC=AX, DEST=port+1:port
```

B) Port variabil:

1110111w

```
if w=0 then
   SRC=AL, DEST=(DX)
else
   SRC=AX, DEST=(DX)+1:(DX)
```

Exemple:

OUT BYTE_PORT_VAL, AL
OUT WORD_PORT_VAL, AX
OUT 44, AX
OUT DX, AL
OUT DX, AX

POP - POP (extrage din stivă)

Operaţie:

```
(DEST)=((SP)+1:(SP))
(SP)=(SP)+2
```

Descriere:

POP *destinaţie*

Extrage din stivă un cuvînt şi îl transferă în operandul destinaţie.

Codificare:

A) Operand registru:

01011 reg

```
DEST=REG
```

B) Operand registru segment:

000 reg 111

```
if reg<>01 then
   DEST=REG
else
   undefined operation
```

C) Operand memorie/registru:

10001111 mod 00 r/m

```
DEST=EA
```

Exemple:

POP CX
POP SS
POP ALPHA

POPF - POP Flags (reface indicatorii din stivă)

Operaţie:

```
FLAGS=((SP)+1:(SP))
(SP)=(SP)+2
```

Descriere:

Extrage din stivă un cuvînt şi îl transferă în registrul de indicatori.

Codificare: 10011101

PUSH - PUSH (salvează cuvînt în stivă)

Operaţie:

```
(SP)=(SP)-2
((SP)+1:(SP))=(SRC)
```

Descriere:

PUSH *sursă*

Salvează în stivă operandul sursă de tip cuvînt.

Codificare:

A) Operand registru:

01010 reg

```
SRC=REG
```

B) Operand registru segment:

000 reg 110

```
SRC=REG
```

C) Operand memorie/registru:

11111111 mod 110 r/m

```
SRC=EA
```

Exemple:

PUSH AX
PUSH SS
PUSH BETA

PUSHF - PUSH Flags (salvează indicatorii în stivă)

Operaţie:

```
(SP)=(SP)-2
((SP)+1:(SP))=FLAGS
```

Descriere:

Salvează în stivă registrul de indicatori.

Codificare: 10011100

RCL - Rotate through Carry Left (roteşte stînga cu CF)

Operaţie:

```
(temp)=COUNT
do while (temp)<>0
  (tmpcf)=(CF)
  (CF)=MSB(EA)
  (EA)=(EA)*2+(tmpcf)
  (temp)=(temp)-1
    if COUNT=1 then
      if (MSB(EA))<>(CF) then
         (OF)=1
      else
         (OF)=0
    else
      (OF) undefined
```

Descriere:

RCL *destinaţie*, *număr*

Roteşte operandul la stînga, împreună cu CF, cu numărul de biţi specificat. Numărul este un operand imediat sau se află în registrul CL.

Codificare: 110100vw mod 010 r/m

```
if v=0 then
   COUNT=1
else
   COUNT=(CL)
```

Exemple:

RCL AH, 1
RCL BX, 1
RCL MEM_BYTE, 1
RCL ALPHA[DI], VAL_ONE
RCL DH, CL
RCL AX, CL
RCL MEM_WORD, CL
RCL BETA[BX][DI], CL

Indicatori: afectaţi - CF, OF

RCR - Rotate through Carry Right (roteşte dreapta cu CF)

Operaţie:

```
(temp)=COUNT
do while (temp)<>0
  (tmpcf)=(CF)
  (CF)=LSB(EA)
  (EA)=(EA)/2
  MSB(EA)=(tmpcf)
  (temp)=(temp)-1
    if COUNT=1 then
      if (MSB(EA)) <> (next_MSB(EA)) then
         (OF)=1
      else
         (OF)=0
    else
      (OF) undefined
```

Descriere:

RCR *destinaţie*, *număr*

Roteşte operandul la dreapta, împreună cu CF, cu numărul de biţi specificat. Numărul este un operand imediat sau se află în registrul CL.

Codificare: 110100vw mod 011 r/m

```
if v=0 then
```

```
   COUNT=1
else
   COUNT=(CL)
```

Exemple:

RCR AH, 1
RCR BX, 1
RCR MEM_BYTE, 1
RCR ALPHA[DI], VAL_ONE
RCR DH, CL
RCR AX, CL
RCR MEM_WORD, CL
RCR BETA[BX][DI], CL

Indicatori: afectaţi - CF, OF

REP, REPE/REPZ, REPNE/REPNZ - REPeat, REPeat while Equal / REPeat while Zero, REPeat while Not Equal / REPeat while Not Zero (repetă, repetă cît timp este egal/zero, repetă cît timp nu este egal / zero)

Operaţie:

```
do while (CX)<>0
  service pending interrupt (if any)
  execute primitive string operation
        in succeding byte
  (CX)=(CX)-1
  if primitive operation is CMPS or
        SCAS  and (ZF)<>z then
     exit from while loop
```

Descriere:

Prefixe de repetiţie utilizate împreună cu operaţiile asupra şirurilor: MOV, STOS, CMPS, SCAS. Operaţia asupra unui element se repetă pînă la terminarea şirului (CX = 0) sau pînă la neîndeplinirea condiţiei de continuare a buclei (ZF). Operaţiile repetitive asupra şirurilor sînt întreruptibile.

Codificare: 1111001z

Exemple:

REP MOVS DEST, SOURCE
REPE CMPS DEST, SOURCE
REPNZ SCAS DEST

RET - RETurn (întoarcere din procedură)

Operaţie:

```
(IP)=((SP)+1:(SP))
(SP)=(SP)+2
if inter-segment then
   (CS)=((SP)+1:(SP))
   (SP)=(SP)+2
   if Add immediate to SP then
      (SP)=(SP)+data
```

Descriere:

RET *valoare_opţională*

Registrul IP este înlocuit de cuvîntul din vîrful stivei. SP este incrementat cu doi. La întoarcerea din alt segment, registrul CS este înlocuit cu cuvîntul din vîrful stivei şi SP este din nou incrementat cu doi. Dacă s-a specificat o valoare imediată, ca operand al instrucţiunii RET, această valoare este adunată la SP, determinînd eliminarea din stivă a datelor plasate acolo de către procedură şi refacerea stivei în starea din momentul lansării procedurii.

Codificare:

A) Intra-segment:

11000011

B) Intra-segment cu dată imediată:

11000010 data-low data-high

C) Inter-segment:

11001011

D) Inter-segment cu dată imediată:

11001010 data-low data-high

Exemple:

RET
RET 4

ROL - ROtate Left (roteşte stînga)

Operaţie:

```
(temp)=COUNT
do while (temp)<>0
  (CF)=MSB(EA)
```

```
  (EA)=(EA)*2+(CF)
  (temp)=(temp)-1
    if COUNT=1 then
      if (MSB(EA)) <> (CF) then
         (OF)=1
      else
         (OF)=0
    else
      (OF) undefined
```

Descriere:

ROL *destinaţie*, *număr*

Roteşte operandul la stînga cu numărul de biţi specificat.

Codificare: 110100vw mod 000 r/m

```
if v=0 then
   COUNT=1
else
   COUNT=(CL)
```

Exemple:

ROL AH, 1
ROL BX, 1
ROL MEM_BYTE, 1
ROL ALPHA[DI], VAL_ONE
ROL DH, CL
ROL AX, CL
ROL MEM_WORD, CL
ROL BETA[BX][DI], CL

Indicatori: afectaţi - CF, OF

ROR - ROtate Right (roteşte dreapta)

Operaţie:

```
(temp)=COUNT
do while (temp)<>0
  (CF)=LSB(EA)
  (EA)=(EA)/2
  MSB(EA)=(CF)
  (temp)=(temp)-1
    if COUNT=1 then
      if (MSB(EA)) <> (CF) then
         (OF)=1
      else
         (OF)=0
    else
      (OF) undefined
```

Descriere:

ROR *destinaţie*, *număr*

Roteşte operandul la dreapta cu numărul de biţi specificat.

Codificare: 110100vw mod 001 r/m

```
if v=0 then
   COUNT=1
else
  COUNT=(CL)
```

Exemple:

ROR AH, 1
ROR BX, 1
ROR MEM_BYTE, 1
ROR ALPHA[DI], VAL_ONE
ROR DH, CL
ROR AX, CL
ROR MEM_WORD, CL
ROR BETA[BX][DI], CL

Indicatori: afectaţi - CF, F

SAHF - Store AH into Flags (copiază AH în registrul de indicatori)

Operaţie:

```
(SF):(ZF):X:(AF):X:(PF):X:(CF)=(AH)
```

Descriere:

Copiază biţii din registrul AH în registrul de indicatori, în ordinea indicată.

Codificare: 10011110

Indicatori: afectaţi - AF, CF, PF, SF, ZF

SAL/SHL - Shift Arithmetic Left / SHift logic Left (deplasare aritmetică / logică la stînga)

Operaţie:

```
(temp)=COUNT
do while (temp)<>0
```

```
  (CF)=MSB(EA)
  (EA)=(EA)*2
  (temp)=(temp)-1
    if COUNT=1 then
      if (MSB(EA))<>(CF) then
         (OF)=1
      else
         (OF)=0
    else
      (OF) undefined
```

Descriere:

SHL *destinaţie*, *număr*

Deplasează operandul destinaţie la stînga cu numărul de biţi specificat. Deplasarea este însoţită de introducerea lui 0 în biţii mai puţin semnificativi. Dacă bitul de semn îşi păstrează valoarea, atunci OF este resetat.

Codificare: 110100vw mod 100 r/m

```
if v=0 then
   COUNT=1
else
   COUNT=(CL)
```

Exemple:

SHL AH, 1
SHL BX, 1
SHL MEM_BYTE, 1
SHL ALPHA[DI], VAL_ONE
SHL DH, CL
SHL AX, CL
SHL MEM_WORD, CL
SHL BETA[BX][DI], CL

Indicatori: afectaţi - CF, OF, PF, SF, ZF, nedefiniţi - AF

SAR - Shift Arithmetic Right (deplasare aritmetică la dreapta)

Operaţie:

```
(temp)=COUNT
do while (temp)<>0
   (CF)=LSB(EA)
   (EA)=(EA)/2
   (temp)=(temp)-1
     if COUNT=1 then
       if (MSB(EA)) <> (next_MSB(EA))
          then
          (OF)=1
       else
          (OF)=0
     else
       (OF) undefined
```

Descriere:

SAR *destinaţie*, *număr*

Deplasează operandul destinaţie la dreapta cu un număr de biţi specificat. Bitul cel mai semnificativ (de semn) îşi păstrează valoarea.

Codificare: 110100vw mod 111 r/m

```
if v=0 then
   COUNT=1
else
   COUNT=(CL)
```

Exemple:

SAR AH, 1
SAR BX, 1
SAR MEM_BYTE, 1
SAR ALPHA[DI], VAL_ONE
SAR DH, CL
SAR AX, CL
SAR MEM_WORD, CL
SAR BETA[BX][DI], CL

Indicatori: afectaţi - CF, OF, PF, SF, ZF, nedefiniţi - AF

SBB - SuBtract with Borrow (scădere cu împrumut)

Operaţie:

```
if (CF)=1 then
   (DEST)=(LSRC)-(RSRC)-1
else
   (DEST)=(LSRC)-(RSRC)
```

Descriere:

SBB *destinaţie*, *sursă*

Scade operandul sursă din operandul destinaţie, scăzînd încă 1 dacă CF este 1. Rezultatul înlocuieste operandul destinaţie. Dacă se scade un octet imediat dintr-un cuvînt din registru sau memorie atunci acest octet este extins la 16 biţi.

Codificare:

A) Operand din memorie sau registru cu operand registru:

000110dw mod reg r/m

```
if d=1 then
   LSRC=REG, RSRC=EA, DEST=REG
else
   LSRC=EA,RSRC=REG, DEST=EA
```

B) Operand imediat cu acumulator:

0001110w [data]

```
if w=0 then
   LSRC=AL, RSRC=data, DEST=AL
else
   LSRC=AX, RSRC=data, DEST=AX
```

C) Operand imediat cu operand din memorie sau registru:

100000sw mod 011 r/m [data]

```
LSRC=EA, RSRC=data, DEST=EA
```

Exemple:

SBB AX, BX
SBB CH, DL
SBB DX, MEM_WORD
SBB DI, ALPHA[SI]
SBB MEM_WORD, AX
SBB GAMMA[BX][DI], SI
SBB AL, 4
SBB AL, VAL_SIXTY
SBB AX, 660
SBB 660
SBB BX, 2001
SBB CL, VAL_SIXTY
SBB MEM_BYTE, 12
SBB GAMMA[DI][BX], 1999

Indicatori: afectaţi - AF, CF, OF, PF, SF, ZF

SCAS - SCAn String (scanează şir)

Operaţie:

```
(LSRC)-(RSRC)
if (DF)=0 then
   (DI)=(DI)+DELTA
else
   (DI)=(DI)-DELTA
```

Descriere:

SCAS *şir_destinaţie*

Scade octetul sau cuvîntul adresat de ES:DI din AL sau AX şi afectează indicatorii dar nu întoarce rezultatul. Utilizînd prefixele de repetiţie REPNZ / REPNE, se poate scana un şir în vederea verificării aparitiei unei valori în şir. DI este actualizat în funcţie de dimensiunea elementelor şirului şi valoarea indicatorului DF.

Operandul numit în SCAS este utilizat numai de asamblor pentru a verifica tipul şi accesul folosind conţinutul registrului curent de segment. Operandul actual utilizează implicit DI pentru a adresa locaţia de scanat, fără a utiliza operandul numit în linia sursă.

Codificare: 1010111w

```
if w=0 then
   LSRC=AL, RSRC=(DI), DELTA=1
else
   LSRC=AX,RSRC=(DI)+1:(DI), DELTA=2
```

Exemple:

SCAS DEST_BYTE_STRING
SCAS WORD_STRING

Indicatori: afectaţi - AF, CF, OF, PF, SF, ZF

SHR - SHift logic Right (deplasare logică la dreapta)

Operaţie:

```
(temp)=COUNT
do while (temp)<>0
  (CF)=LSB(EA)
  (EA)=(EA)/2
  (temp)=(temp)-1
    if COUNT=1 then
      if (MSB(EA))<>(next_MSB(EA))
      then
        (OF)=1
      else
        (OF)=0
    else
      (OF) undefined
```

Descriere:

SHR *şir_destinaţie*, *şir_sursă*

Deplasează operandul destinaţie la dreapta cu un număr de biţi specificat. Bitul cel mai semnificativ este zero.

Codificare: 110100vw mod 101 r/m

```
if v=0 then
   COUNT=1
else
   COUNT=(CL)
```

Exemple:

SHR AH, 1
SHR BX, 1
SHR MEM_BYTE, 1
SHR ALPHA[DI], VAL_ONE
SHR DH, CL
SHR AX, CL
SHR MEM_WORD, CL
SHR BETA[BX][DI], CL

Indicatori: afectaţi - CF, OF, PF, SF, ZF, nedefiniţi - AF

STC - SeT Carry flag (setează indicatorul CF)

Operaţie:

```
(CF)=1
```

Descriere:

Setează indicatorul de transport CF.

Codificare: 11111001

Indicatori: afectaţi - CF

STD - SeT Direction flag (setează indicatorul de direcţie DF)

Operaţie:

```
(DF)=1
```

Descriere:

Setează indicatorul de direcţie DF.

Codificare: 11111101

Indicatori: afectaţi - DF

STI - SeT Interrupt-enable flag (setează indicatorul de validare a întreruperilor IF)

Operaţie:

```
(IF)=1
```

Descriere:

Setează indicatorul de validare a întreruperilor IF.

Codificare: 11111011

Indicatori: afectaţi - IF

STOS - STOre String (scrie şir în memorie)

Operaţie:

```
(DEST)=(SRC)
if (DF)=0 then
   (DI)=(DI)+DELTA
else
   (DI)=(DI)-DELTA
```

Descriere:

STOS *şir_destinaţie*

Copiază un octet sau cuvînt din AL sau AX într-o locaţie de memorie adresată de ES:DI şi actualizează registrul DI. Utilizînd prefixe de repetiţie, se poate iniţializa un bloc de memorie cu o valoare. Operandul numit în instrucţiunea STOS este folosit numai de asamblor pentru verificarea tipului şi accesului utilizînd

conţinutul registrului de segment curent. Operandul actual al instrucţiunii utilizează DI pentru a pointa la locaţia de memorie.

Codificare: 1010101w

```
if w=0 then
   SRC=AL, DEST=(DI), DELTA=1
else
   SRC=AX, DEST=(DI)+1:(DI), DELTA=2
```

Exemple:

STOS BYTE_DEST_STRING
STOS WORD_DEST

SUB - SUBtract (scădere)

Operaţie:

```
(DEST)=(LSRC)-(RSRC)
```

Descriere:

SUB *destinaţie*, *sursă*

Scade operandul sursă din operandul destinaţie. Rezultatul înlocuieşte operandul destinaţie. Dacă se scade un octet imediat dintr-un cuvînt din registru sau memorie atunci acest octet este extins la 16 biţi.

Codificare:

A) Operand din memorie sau registru cu operand registru:

001010dw mod reg r/m

```
if d=1 then
   LSRC=REG, RSRC=EA, DEST=REG
else
   LSRC=EA, RSRC=REG, DEST=EA
```

B) Operand imediat cu acumulator:

0010110w [data]

```
if w=0 then
   LSRC=AL, RSRC=data, DEST=AL
else
   LSRC=AX, RSRC=data, DEST=AX
```

C) Operand imediat cu operand din memorie sau registru:

100000sw mod 101 r/m [data]

```
LSRC=EA, RSRC=data, DEST=EA
```

Exemple:

SUB AX, BX
SUB CH, DL
SUB DX, MEM_WORD
SUB DI, ALPHA[SI]
SUB MEM_WORD, AX
SUB GAMMA[BX][DI], SI
SUB AL, 4
SUB AL, VAL_SIXTY
SUB AX, 660
SUB 660
SUB BX, 2001
SUB CL, VAL_SIXTY
SUB MEM_BYTE, 12
SUB GAMMA[DI][BX], 1984

Indicatori: afectaţi - AF, CF, OF, PF, SF, ZF

TEST - TEST (testare)

Operaţie:

```
(LSRC)&(RSRC)
(CF)=0
(OF)=0
```

Descriere:

TEST *destinaţie*, *sursă*

Realizează operaţia 'şi logic' între cei doi operanzi fără a returna rezultatul.

Codificare:

A) Operand din memorie sau registru cu operand registru:

1000010w mod reg r/m

```
LSRC=REG, RSRC=EA
```

B) Operand imediat cu acumulator:

1010100w [data]

```
if w=0 then
   LSRC=AL, RSRC=data
else
```

```
LSRC=AX, RSRC=data
```

C) Operand imediat cu operand din memorie sau registru:

1111011w mod 000 r/m [data]

```
LSRC=EA, RSRC=data
```

Exemple:

TEST AX, DX
TEST DX
TEST SI, BP
TEST BH, CL
TEST MEM_WORD, SI
TEST BETA[BX][SI], CX
TEST CH, GAMMA[BP][SI]
TEST AL, 6
TEST AX, 999
TEST 999
TEST AL, IMM_VAL_18
TEST BH, 7
TEST SI, 798
TEST [BP][DI], 6ACEH
TEST GAMMA[BX], IMM_BYTE

Indicatori: afectaţi - CF, OF, PF, SF, ZF, nedefiniţi - AF

WAIT - WAIT (aşteaptare)

Operaţie:

nici una

Descriere:

Determină intrarea procesorului în starea de aşteptare pînă la activarea pinul TEST. Starea WAIT poate fi părăsită şi datorită apariţiei unei întreruperi externe.

Codificare: 10011011

XCHG - eXCHanGe (schimbare)

Operaţie:

```
(temp)=DEST
(DEST)=(SRC)
(SRC)=(temp)
```

Descriere:

XCHG *destinaţie*, *sursă*

Schimbă conţinuturile operanzilor sursă şi destinaţie. Registrele de segment nu pot fi operanzi ai instrucţiunii XCHG. Utilizată împreună cu prefixul LOCK, instrucţiunea XCHG poate testa şi seta un semafor pentru controlul accesului la o resursă comună.

Codificare:

A) Operand registru cu acumulator:

10010 reg

```
SRC=REG, DEST=AX
```

B) Operand registru sau memorie cu operand registru:

1000011w mod reg r/m

```
SRC=EA, DEST=REG
```

Exemple:

XCHG AX, BX
XCHG SI, AX
XCHG BETA_WORD, CX
XCHG DH, ALPHA_BYTE
XCHG BL, AL

XLAT - transLATe (translatare)

Operaţie:

```
(AL)=((BX)+(AL))
```

Descriere:

XCHG *tabel_translatare*

Înlocuieşte octetul din registrul AL cu un octet dintr-o tabelă, de cel mult 256 de octeţi, adresată prin registrul BX.

Codificare: 11010111

Exemple:

XLAT TABLE_ENTRY

XOR - eXclusive OR ('sau exclusiv' logic)

Operaţie:

```
(DEST)=(LSRC) XOR (RSRC)
(CF)=0
```

```
(OF)=0
```

Descriere:

XOR *destinaţie*, *sursă*

Realizează funcţia logică 'sau exclusiv' bit cu bit între cei doi operanzi.

Codificare:

A) Operand din memorie sau registru cu operand registru:

001100dw mod reg r/m

```
if d=1 then
   LSRC=REG, RSRC=EA, DEST=REG
else
   LSRC=EA, RSRC=REG, DEST=EA
```

B) Operand imediat cu acumulator:

0011010w [data]

```
if w=0 then
   LSRC=AL, RSRC=data, DEST=AL
else
   LSRC=AX, RSRC=data, DEST=AX
```

C) Operand imediat cu operand din memorie sau registru:

1000000w mod 110 r/m [data]

```
LSRC=EA, RSRC=data, DEST=EA
```

Exemple:

XOR AL, BL
XOR SI, DX
XOR AX, MEM_WORD
XOR AI, ALPHA[BX][SI]
XOR BETA[BX][SI]
XOR MEM_BYTE, DH
XOR AL, 0F6H
XOR AX, 23F6H
XOR 23F6H
XOR AH, 0F6H
XOR CL, 37
XOR MEM_BYTE, 3DH
XOR ALPHA[DI], VAL_EQUD-33H

Indicatori: afectaţi - CF, OF, PF, SF, ZF, nedefiniţi - AF