

ВВЕДЕНИЕ@u'
ВВЕДЕНИЕ@u'

Будем вести изложение в следующей форме:

Текст программы на Модуле - 2	Генерируемый код с комментариями

3) опущены команды динамического контроля (например, контроля границ массивов).

ОПЕРАТОРЫ@u '
 ОПЕРАТОРЫ@u '

G:=1;	LI1 SGW2
G:=G+255;	LGW2 LIB FF ADD SGW2
- - - - -	- - - \ - / - - - -

G	255	'+'	G:=

B:={0..31};	LIW FFFFFFFF SGW3

END M.	\-/-
	{0..31} B:=

1.2. Доступ к глобальным переменным

MODULE M;

VAR G2,G3,G4, ... ,G255:INTEGER;
G256: INTEGER;

BEGIN

G2 := G2;	LGW02 SGW02
G255:=G255;	LGW FF SGW FF (1.2.1)
G256:=G256;	--- (1.2.2)

END M.	

Примечание 1.2.1. К первым 14 глобалам с номерами 2..15 доступ однобайтовый, к глобалам с номерами 16..255 - двухбайтовый.

Примечание 1.2.2. Это не компилируется текущей версией компилятора, но может порождаться так:

LGA FF LSW1 LGA FF SSW1

1.3. Доступ к внешним переменным

Внешними переменными называются глобальные переменные других модулей.

DEFINITION MODULE M;
VAR i: INTEGER;
END M.

номер модуля M в локальной DFT модуля N	
MODULE N;	Номер
FROM M IMPORT i;	переменной i в модуле M
BEGIN	

i:=i;	LEW 01 02 SEW 01 02

END N.	

1.4. Условный оператор (IF)

[illegible]

Примечание 1.4.2. Безусловный переход (если есть FALSE-альтернатива).

```

BEGIN                                -->  - - - - -
- - - - - --/                      |--> JSF 02 --|
| LOOP   EXIT END;                  |
- - - - - \                          JSB 04 --|
END  M.                             -->  - - - - -
                                     |-->

```

```
MODULE M;
VAR G2:INTEGER;
BEGIN
- - - - -> |-->|-----|
| LOOP      |    | код для CASE |
|   CASE G2 OF |    | размером более
```

	0..127: G2:=0				255 байтов			- -> (1.5.1)
	END;							
	END;		----		JLB offset			
-	- - - - - - - - - - ->				_____		_____	--
END M.								

Примечание 1.5.1. Двухбайтовое смещение для переходов длиннее 255 байт.

Генерация LOOP-цикла с помощью команд ENTS и XIT:

MODULE M_i ;

```
VAR G2:INTEGER;
```

BEGIN	G2:=0	
- - - - - / - - - -		
G2:=0;	LI0 SGW2	
LOOP	ENTS 000C	(1.5.2)
- - - - -		

```

- - - - ->      |          G2       1   +   G2:=      |
                  |          |         |           |
G2:=G2+1;        |--> LGW2    LI1     ADD    SGW2      |
                  |          |         |           |
IF G2>5 THEN EXIT END;      LGW2    LI5    GTR    JSFC    01    XIT    JSB    0C
- - - - ->      |          |         |           |
END;              |          |         |           |
END M.            G2      5      >

```

Примечание 1.5.2. ENTS кладет на вершину Р-стека текущее значение PC + значение следующих за командой двух байтов. XIT берет с верхушки Р-стека новое значение PC, совершая переход.

1.6. Оператор цикла (REPEAT)

MODULE M;

```
VAR G2:INTEGER; bool: BOOLEAN;
```

BEGIN

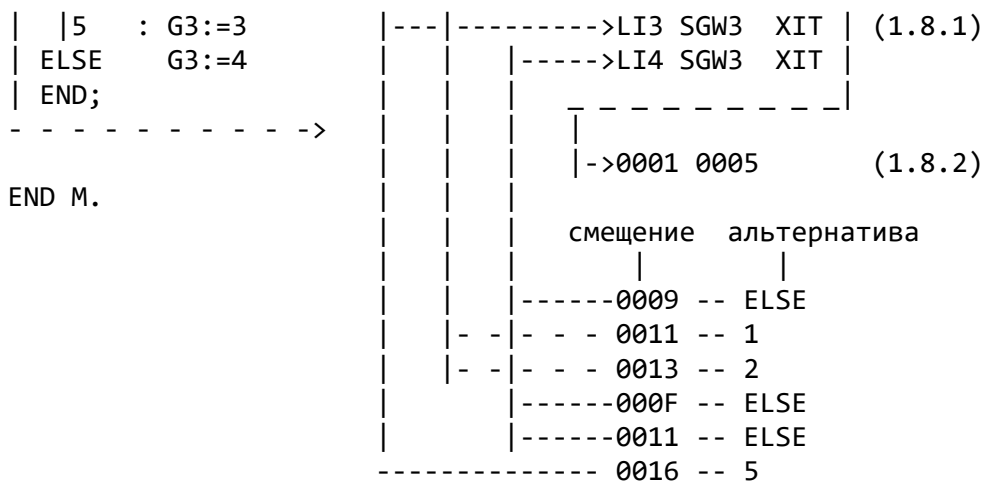
```
- - - - ->
| REPEAT G2:=1 UNTIL bool;      | --> LI1   SGW2    |
- - - - ->                      LGW3 JSBC 05 --|
END M.
```

1.7. Оператор цикла (FOR)

MODULE M;

```
VAR i,G3:INTEGER;
```

BEGIN



Примечание 1.8.1. ENTC выбирает двухбайтовое смещение из кода и переходит в таблицу выбора.

Примечание 1.8.2. Из кода выбираются минимальное (lo=1) и максимальное (hi=5) значения альтернатив. Затем вычисляется PC точки выхода PC'=PC+2*(hi-lo)+4 и записывается на P-стек. Затем с A-стека выбирается значение параметра i и подготавливается переход к нужной альтернативе: PC:=PC+2*(i-lo+1). В случае i>hi или i<lo PC остается неизменным. Теперь все готово к переходу: из кода выбирается двухбайтовое смещение delta, определяющее переход PC:=PC-delta. После выполнения группы операторов выбранной альтернативы выполняется команда XIT, которая осуществляет выход из CASE путем выполнения PC:=PC'. PC' берется командой XIT с P-стека.

.PAGE	
.HEAD 1 '@УКРОНОС В КАРТИНКАХ	ПРОЦЕДУРЫ@u'
.HEAD 0 '@УКРОНОС В КАРТИНКАХ	ПРОЦЕДУРЫ@u'

2. ПРОЦЕДУРЫ

2.1. Описание и вызов примитивной процедуры

```

MODULE M;

PROCEDURE P; (* процедура 1 *)

- - - - -
BEGIN | RETURN END P;          RTN |
- - - - -

BEGIN          (* процедура 0 *)

```

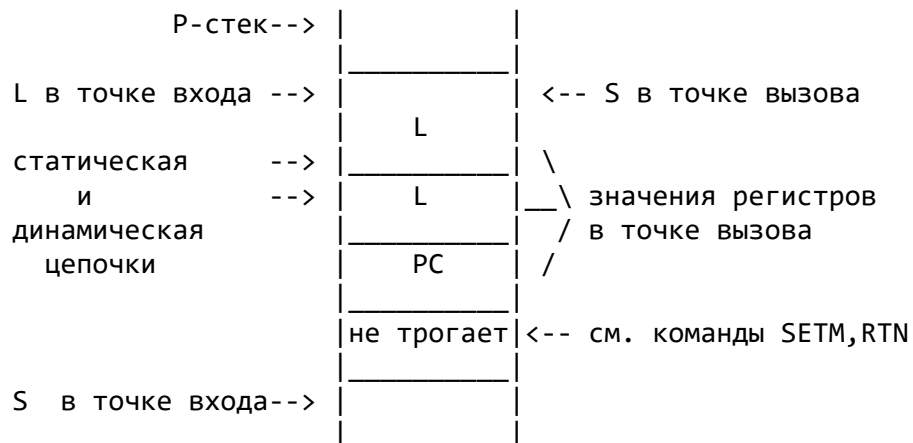
```

- - - - -
| P;                                CL1 | (2.1.1)
- - - - -
END M.

```

вызов локальной процедуры 1

Примечание 2.1.1. CL1 маркирует Р-стек следующим образом:



После маркировки в L засылается значение S, а в S значение S+4. По номеру процедуры (в данном случае - 1) из процедурной таблицы (см. Примечание 2.1.2) извлекается смещение до начала кода соответствующей процедуры и засылается в PC, после чего начинают исполняться команды тела процедуры. RTN берет из области связей процедуры значение PC и L в точке их вызова и засылает их в регистры PC и L. В регистр S засылается значение L в точке возврата (которое является старым значением S в точке вызова).

Примечание 2.1.2. Процедурная таблица - таблица, по номеру процедуры указывающая смещение начала кода процедуры относительно начала кода модуля. Нулевая процедура - инициализирующая часть модуля.

2.2. Работа с локалами процедуры

```

MODULE M;
PROCEDURE P; (* процедура 1 *)

```

число локальных переменных процедуры

```

- - - - -
VAR  | L4: INTEGER;          ENTR 01      | (2.2.1)
BEGIN | L4:=0; RETURN        LI0 SLW4 RTN |
- - - - -
END P;

```

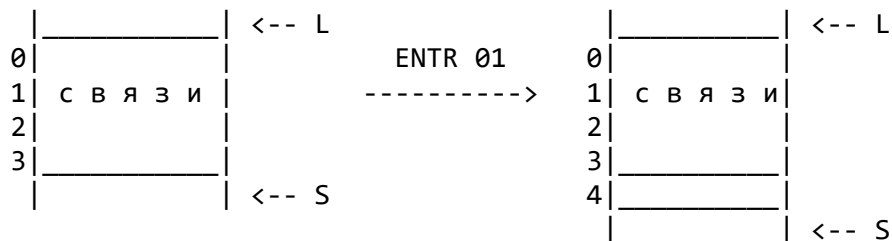
первые четыре слова (0..3)
заняты под область связей

```

BEGIN      (* процедура 0 *)
-----
| P;                      CL1 |
-----
END M.

```

Примечание 2.2.1. ENTR 01 передвигает регистр S на одно слово:



После этого становится возможным работать с локальной переменной командами LLW, SLW (аналогичными командам LGW, SGW), адресующими данные относительно L-регистра.

2.3. Вложенные процедуры

```
MODULE M;
```

```

PROCEDURE p1; (* процедура 1*)
  VAR p1L4:INTEGER;

```

```

  PROCEDURE p2; (* процедура 2*)
.PAGE
  VAR p2L4:INTEGER;
  BEGIN
    |p1L4:=12;      GB1 LI12 SSW4 |
    |p2L4:=11;      LI11 SLW4 RTN |
  END p1;
  BEGIN
    |p1L4:=2;       LI2  SLW4   |
    |p2;            LLA 00 CI 02 RTN | (2.3.1)
  END p1;

```

кладёт на A-стек
L-регистр охватывающей
процедуры p1

номер процедуры в процедурной таблице

Значение L-регистра

команда вызова процедуры промежуточного уровня


```

BEGIN
- - - - -
| p1;                                CL1 |
- - - - -
END M.

```

2.4. Вызов внешней процедуры

```

DEFINITION MODULE N;

PROCEDURE proc1;

END N.

```

```

MODULE M;

FROM N IMPORT proc1;

```

```

PROCEDURE p1;
BEGIN
- - - - -
| proc1;                                CX 01 01 RTN | (2.4.1)
- - - - -
END p1;

END M.

```

номер модуля
 |
 номер процедуры модуля
 |
 - - - - -

Примечание 2.4.1. В области связей модуля элемент с номером 1 соответствует модулю N и после загрузки модуля в память ссылается на слово, содержащее адрес начала области глобальных данных модуля N, в первом слове которой содержится F-регистр модуля N, позволяющий добраться до его сегмента кода.

CX помещает в нулевое слово области связей процедуры G-регистр модуля, вызвавшего эту процедуру, и помечает, выставя признак во втором локальном слове, что вызов был внешним. Команда RTN анализирует этот признак и в случае необходимости восстанавливает значение G-регистра.

2.5. Размещение мультимножеств

```

MODULE M;

```

VAR	i: INTEGER;	ENTR 02	
	A: ARRAY [0..15] OF INTEGER;	LIB 10 ALLOC SLW5	(2.5.1)

i:=0;	LI0	SLW4		
	A	i		
A[i]:=1;	LLW5	LLW4	LI1 SXW	(2.5.2)
				(2.5.3)

END M.

Примечание 2.5.3. При возврате из процедуры команда RTN переставляет S в L, и тем самым освобождает всю память, занятую локалами, в том числе и память, выделенную на P-стеке командой ALLOC.

```

сохранить А-стек на Р-стеке
| загрузить процедурное значение Р1
| из локального слова 4 процедуры Р
BEGIN |
      | записать на Р-стек значение
      | с А-стека

```

```

- - - - -| - -| - -| - -| - -| - -| - -| - -|
|p1(1);          SLW4  LLW4  STOT  LI1  CF  RTN  |
- - - - -| - -| - -| - -| - -| - -| - -| - -|
END P;
                                |
                                | параметр |
                                |         |
                                |         | вызов формальной
                                |         | процедуры

```

```

PROCEDURE p(w: INTEGER); (* процедура # 2 *)
BEGIN

```

```

- - - - -| - -| - -| - -| - -| - -| - -| - -|
|          SLW4  RTN  |
- - - - -| - -| - -| - -| - -| - -| - -| - -|
END p;

```

```

VAR v: proc1;          номер модуля =0, т.к. проц. собственная
                        | номер процедуры
BEGIN
                        | |

```

```

- - - - -| - -| - -| - -| - -| - -| - -| - -|
|v:=p;          LPC 00 02  SGW2  |
|v(5);          LGW2  STOT  LI5   CF  |
|P(v);          LGW2  CL1   |
|P(p);          LPC 00 02  CL1   |
- - - - -| - -| - -| - -| - -| - -| - -| - -|
END M.

```

Процедурное значение:

<-- о д н о с л о в о -->									
31..24				23 0					
-----				-----					
номер				24-разрядный адрес					
процедуры				G-регистра модуля					

Процедурные связи при вызове процедуры командой CF (аналогично CX):

```

L'--> |         |
      |         |
      |         |
S', L-->|-----|
      | G-регистр |
      |-----|
      |   L'   |
      |         |
S -->  |         |
      |         |
      |         |

```

2.7. Передача параметров

```

MODULE M;

```

```

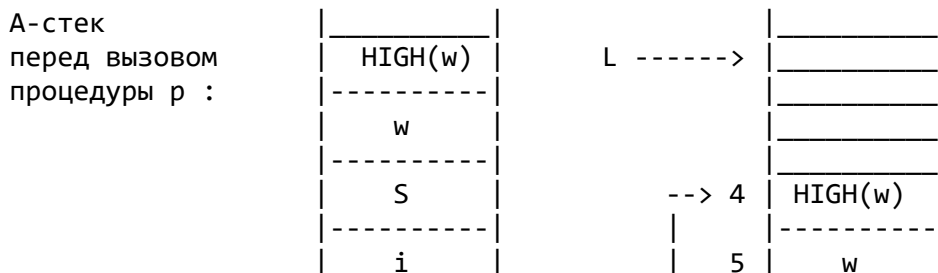
TYPE String=ARRAY [0..255] OF CHAR;

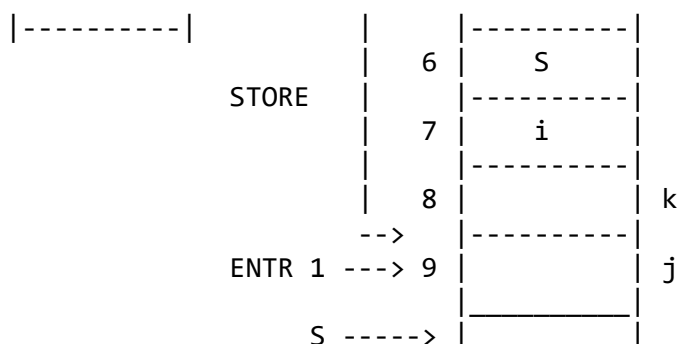
PROCEDURE P(i: INTEGER; S: String; VAR w: ARRAY OF CHAR);
  VAR k,j:INTEGER;
BEGIN
  (* сохранение параметров и размещение переменных *)
  -----
  |                                     STORE ENTR 01      | (2.7.1)
  -----
  (* копирование массива S, переданного по значению *)
  -----
  |k:=HIGH(S);                                LIB FF  SLW8   |
  |j:=HIGH(w);                                LLW4 SLW9 RTN   | (2.7.2)
  |                                     |
  |                                     |
  |                                     |
  -----
END P;

VAR
  -----
  |str8:ARRAY[0..7] OF CHAR;  LI2 ALLOC SGW2   |
  |str :String;              LIB 40 ALLOC SGW3   |
  -----
BEGIN
  загрузка на А-стек адреса константы 'abc'
  |
  -----
  |P(1,'abc',str8);          LI1 LSTA 0001 LGW2 LI7 CL1 |
  |str:='def';              LGW3 LSTA *ind* LI1 MOVE   | (2.7.3)
  |P(2,str,str);            LI2 LGW3 LGW3 LIB FF CL1   |
  -----
END M.

```

Примечание 2.7.1. STORE записывает А-стек на Р-стек, выделяя на Р-стеке память под локалы: 4-е слово - HIGH(w), 5-е слово - w(адрес), 6-е слово - S (адрес), 7-е слово - i, 8-е слово - число параметров - используется для локала k. Команда ENTR 01 завершает отведение памяти; 9-е слово отводится для j :





Примечание 2.7.2. Здесь приведена группа команд, которая копирует переданный по значению массив S. Приводим три возможных варианта кода:

I)	LIB 40	II)	LLW6	III)	LLW6
	ALLOC		LIB FF		LIB 3F
	COPT		CPCOP 06		PCOP 06
	LLW6				
	LIB 40				
	MOVE				
	SLW6				

II) LLW6
LIB FF
CPCOP 06

III) LLW6
LIB 3F
PCOP 06

Очевидна целесообразность введения в систему команд `CRCPOR` и `PCOR`, предназначенных для размещения и копирования мультипараметров.

Примечание 2.7.3. LSTA *ind* по относительному смещению *ind* в строковом пуле грузит на A-стек адрес соответствующей строковой константы.

2.8. Вызов функции (на непустом стеке)

MODULE M_i ;

```
PROCEDURE f(i,j: INTEGER): INTEGER;  
BEGIN
```

```

- - - - -
| RETURN    i+j                STORE LLW5 LLW4 ADD RTN |
- - - - -
END f;

```

```
PROCEDURE p(i,j: INTEGER);
  VAR k:INTEGER;
BEGIN
```

```
| k:=i+j;                                STORE LLW5 LLW4 ADD SLW6 RTN |
- - - - -
END p;
```

END p;

```

VAR v: PROCEDURE(INTEGER,INTEGER): INTEGER;
BEGIN
  - - - - -
  | p(1,f(2,3));   LI1      STORE LI2 LI3 CL1 LODFV CL2 |
  | v:=f;          LPC 00 01 SGW2                       |
  | p(1,v(2,3));   LI1 LGW2 STOFV LI2 LI3 CF LODFV CL2  |(2.8.1)
  | - - - - -
END M.

```

Примечание 2.8.1. Последовательность команд LI1 LI2 CL1 CL2 была бы неверна, поскольку CL1 в точке входа забирает со стека все значения (STORE), в том числе и 1, предназначенную не для нее (!).

```

.PAGE
.HEAD 1 '@УКРОНОС В КАРТИНКАХ                                ВЫРАЖЕНИЯ@u'
.HEAD 0 '@УКРОНОС В КАРТИНКАХ                                ВЫРАЖЕНИЯ@u'

```

3. ВЫРАЖЕНИЯ

3.1. Индексация словных массивов

```

MODULE M;

VAR x: ARRAY [0..3] OF INTEGER;
    i: INTEGER;

BEGIN
  - - - - -
  | x[i]:=1;       LGW2 LGW3 LI3 CHKZ LI1 SXW | -- с контролем
  | i:=x[1];       LGW2 LI1  LXW SGW3        |   границ
  | - - - - -
  | x[i]:=1;       LGW2 LGW3 LI1 SXW | -- без контроля границ
  | i:=x[1];       LGW2 LI1  LXW SGW3 |   (3.1.1)
  | - - - - -
END M.

```

Примечание 3.1.1. Компилятор может использовать константную индексацию, например:

```

- - - - -
|   i:=x[1]       LGW2 LSW1     SGW2   |
- - - - -

```

и таким образом, контроль границ становится не нужен.

3.2. Индексация байтовых массивов

```
MODULE M;
- - - - -
VAR | A:ARRAY [0..0Fh] OF CHAR;   LI4 ALLOC SGW2-|--(3.2.1)
- - - - -
    i:INTEGER;                      (3.2.2)
- - - - -
BEGIN
- - - - -
|i:=0;                             LI0 SGW3 |
- - - - -
```

Примечание 3.2.1. В глобальном слове 2 адрес размещенного массива.

Примечание 3.2.3. LXB и SXB работают аналогично LXW и SXW, но читают и пишут соответствующий байт, а не слово. θ <байтовый адрес> LXB и θ < байтовый адрес > SXB осуществляют абсолютную байтовую адресацию памяти.

```
MODULE M;  
  
VAR A:ARRAY [0..0Fh] OF CHAR;  
    i:INTEGER;
```

```
BEGIN i:=0;
```

```

| WHILE i#HIGH(A) DO -> LIW3 LI0F NEQ JSFC 0E |
|   A[i]:=A[i+1];      -> LGW2 LGW3 LI0F CHKZ |
|                       LGW2 LGW3 LI1 ADD LI0F |
|                       |   |   |   |   |   |
|                       A   i   1   '+' HIGH(A) |
| END (*WHILE *)       CHKZ LXB SXB JSB 013 |
|                       |                   |
|                       (3.3.1)              |
|-----|

```

Примечание 3.3.1. CHKZ проверяет, лежит ли второй элемент A-стека между 0 и верхним элементом (задающим границу); если это так, то счеркивает границу, иначе возбуждает TRAP(4Ah).

3.4. Проверка принадлежности диапазону

```

MODULE M;
VAR i: INTEGER;
VAR x:[10h..20h];
BEGIN
|-----|
| x:=i;                                LGW2 LIB 10 LIB 20 CHK SGW3 |
|-----|
END M.
|
|
| (3.4.1)

```

Примечание 3.4.1. CHK проверяет принадлежность диапазону. Если бы в тексте было x:=13h, проверку произвел бы компилятор и породил код LIB13 SGW2.

3.5. Работа с объектами типа BITSET.

```

MODULE M;
VAR b1, b2:BITSET;
BEGIN
|-----|
| b1:={}; b2:={1};                    LI0 SGW2 LI2 SGW3 |
| b1:=b1+b2;                          LGW2 LGW3 OR SGW3 |
| (*      *                            AND                |
|      /                                XOR                |
|      -                                BIC      *)        |
|
|      INCL(b1,2);                      LGA 2 LI2 INCL      |
|      EXCL(b1,2);                      LGA 2 LI2 EXCL      |
| (* 2 IN b1                            LI2 LGW2 IN      *)

```



```

- - - - -
END M.

```

3.6. Команды ANDJP и ORJP

```

MODULE M;
.
.
BEGIN
- - - - -
| IF FALSE AND TRUE THEN END;  LI0  ANDJP 1  LI1  JSFC  |
| IF TRUE OR FALSE THEN END;   LI1  ORJP 1  LI0  JSFC  |
- - - - -
END M.
(3.6.1)

```

Примечание 3.6.1. Компилятор вместо этого смешного кода, оптимизируя, не породит ничего.

```

.PAGE
.HEAD 1 '@УАРХИТЕКТУРА
.HEAD 0 '@УАРХИТЕКТУРА
                ИНДЕКС-ТАБЛИЦА СИСТЕМЫ КОМАНД
                ИНДЕКС-ТАБЛИЦА@u'
                ИНДЕКС-ТАБЛИЦА@u'

```

Мнемоника	Код	Страница описания
-----------	-----	-------------------

ABS	0A6h	55
ACTIV	0FAh	85
ADD	88h	46
ADDPC	0BCh	65
ALLOC	0C8h	69
AND	0A9h	56
ANDJP	0BFh	66
ARRCMP	95h	49
BBLT	0EEh	77
BBP	0EDh	77
BBU	0ECh	76
BIC	0ABh	57
BIT	0ADh	58
BM	97h	50
BMG	0F9h	81
CF	0CEh	71
CHK	0C6h	68
CHKBX	0F8h	81
CHKNIL	0C1h	67
CHKZ	0C7h	68
CI	0CDh	71
CL	0CFh	72
CL0..CL0F	0D0h..0DFh	72
CM	0F7h	80
COMP	0C3h	67
COPT	0B5h	61
CPCOP	0B6h	61
CX	0CCh	70

DEC	0E7h	75
DEC1	0E5h	74
DECS	0B0h	59
DIV	8Bh	47
DROP	0B1h	59
ENTC	0BAh	63
ENTR	0C9h	69
EQU	0A4h	55
EXCL	0E1h	73
FABS	9Dh	53
FADD	98h	51
FCMP	9Ch	52
FDIV	9Bh	52
FFCT	9Fh	53
FMUL	9Ah	51
FNEG	9Eh	53
FOR1	0B8h	62
FOR2	0B9h	63
FSUB	99h	51
GB	0C4h	68
GB1	0C5h	68
GEQ	0A3h	55
GETM	82h	44
GTR	0A2h	54
IDLE	87h	46
IN	0ACh	58
INC	0E6h	75
INC1	0E4h	74
INCL	0E0h	73
INL	0E2h	73
INVLD	0FFh	86
IO0..IO4	90h..94h	49
JBL	1Dh	38
JBLC	1Ch	38
JBS	1Fh	38
JBSC	1Eh	38
JFL	19h	37
JFLC	18h	37
JFS	1Bh	37
JFSC	1Ah	37
JMP	0BDh	65
LEA	17h	36
LEQ	0A1h	54
LEW	22h	39
LGA	15h	35
LGW	21h	39
LGW2..LGW0F	42h..4Fh	42
LI0..LI0F	00h..0Fh	34
LIB	10h	34
LID	11h	34
LIN	13h	35
LIW	12h	34
LLA	14h	35
LLW	20h	39

LLW4..LLW0F	24h..2Fh	40
LODFV	0B2h	60
LODT	0E9h	76
LPA	0F1h	79
LPC	0EBh	76
LPW	0F2h	79
LSA	16h	36
LSS	0A0h	54
LSTA	0C2h	67
LSW	23h	40
LSW0..LSW0F	60h..6Fh	43
LXA	0EAh	76
LXB	40h	42
LXW	41h	42
MOD	0AFh	59
MOVE	0C0h	66
MUL	8Ah	47
NEG	0A7h	56
NEQ	0A5h	55
NII	0FDh	86
NOP	0CBh	70
NOT	0AEh	59
OR	0A8h	56
ORJP	0BEh	65
PCOP	0B7h	62
PDX	0EFh	78
QUIT	81h	44
QUOT	0E3h	73
RCHK	0F5h	80
RCHZ	0F6h	80
ROL	8Eh	48
ROR	8Fh	48
RTN	0CAh	70
SETM	83h	44
SEW	32h	41
SGW	31h	41
SGW2..SGW0F	52h..5Fh	43
SHL	8Ch	47
SHR	8Dh	48
SLW	30h	40
SLW4..SLW0F	34h..3Fh	41
SPW	0F3h	79
SSW	33h	41
SSW0..SSW0F	70h..7Fh	44
SSWU	0F4h	79
STOFV	0B4h	60
STORE	0B3h	60
STOT	0E8h	75
SUB	89h	46
SWAP	0F0h	78
SXB	50h	42
SXW	51h	43
SYS	0FCh	86
TR	86h	45

TRA	85h	45
TRAP	84h	45
USR	0FBh	85
WM	96h	50
XIT	0BBh	65
XOR	0AAh	57