3. Макропрограммирование

До сих пор созданием текстов на языке ассемблера (программированием) занимались мы сами, а ассемблер транслировал их в программы на машинном языке. Однако большинство ассемблеров могут кроме этого по определенным правилам сами генерировать команды на языке ассемблера из команд условной генерации и макрокоманд, написанных программистом.

Такие ассемблеры называют макроассемблерами. К ним относится и макроассемблер M80. Процесс трансляции макроассемблером может состоять из двух этапов:

1. анализ программы и генерация текста на языке ассемблера;
2. генерация программы в машинных кодах.

Таким образом, программирование на макроассемблере занимает промежуточное положение между программированием на языке ассемблера и программированием на языке высокого уровня типа Си, Паскаль, Ада.

Рассмотрим некоторые возможности макроассемблирования.

3.1. Генерация текста на языке ассемблера

Макроассемблер предоставляет различные возможности по автоматической генерации текста на языке ассемблера по заданным шаблонам.

3.1.1 Генерация текста несколько раз

Если некоторую группу команд нужно повторить несколько раз (подряд), можно использовать команду повторения REPT. Она имеет следующий вид:

```
REPT   выражение
команды-ассемблера
ENDM
```

Выражение задает количество повторений генерации команд на ассемблере до команды ENDM.

Например, напишем следующий исходный текст:

```
.Z80
LD    A,B
REPT  4
RLCA
ADD   a,3
ENDM
LD    B,A
AND   0fh
END
```

После трансляции макроассемблером M80 получим следующий листинг:

```
     MSX.M-80  1.00     01-Apr-85          PAGE         1
                                  .Z80
  0000'   78                      LD    A,B
                                  REPT  4
                                  RLCA
                                  ADD   a,3
                                  ENDM
  0001'   07              +       RLCA
  0002'   C6 03           +       ADD   a,3
  0004'   07              +       RLCA
```

```
  0005'   C6 03           +        ADD   a,3
  0007'   07              +        RLCA
  0008'   C6 03           +        ADD   a,3
  000A'   07              +        RLCA
  000B'   C6 03           +        ADD   a,3
  000D'   47                       LD    B,A
  000E'   E6 0F                    AND   0fh
                                   END
```

Обратите внимание, что макроассемблер отметил команды, которые он сам сгенерировал, знаком «+».

Кроме команд ассемблера в теле REPT могут стоять и некоторые директивы ассемблера, например, DB.

```
   MSX.M-80   1.00     01-Apr-85        PAGE          1
                                         .Z80
  0000'   3E 07'                         LD    a,data
  0002'   06 08'                         LD    b,data+1
  0004'   0E 09'                         LD    c,data+2
  0006'   C9                             RET
  0007'                          data    EQU   $
                                         REPT  3
                                         DB    1, 2
                                         DB    7
                                         ENDM
  0007'   01 02           +              DB    1, 2
  0009'   07              +              DB    7
  000A'   01 02           +              DB    1, 2
  000C'   07              +              DB    7
  000D'   01 02           +              DB    1, 2
  000F'   07              +              DB    7
                                         END
```

3.1.2. Генерация текста с параметрами

Иногда есть необходимость сгенерировать схожие в чем-то тексты, отличающиеся только некоторыми деталями. Для этого можно использовать одну из двух команд генерации:

```
IRP   параметр,<список>           IRPC   параметр,строка
команды-ассемблера         команды-ассемблера
ENDM                                ENDM
```

Параметр — это любое допустимое имя языка ассемблера. Ассемблер M80 допускает имена, содержащие знак «$». Их удобно использовать для обозначения параметров.

Команда IRP генерирует команды, каждый раз заменяя параметр в командах очередным значением из списка, а команда IRPC подставляет вместо параметра очередной символ строки.

Например, исходный текст:

```
             .Z80
             XOR   a
             LD    a,(data)
             RET
data         EQU   $
             IRP   $P,<1,2,4,7>
             DB    $P
             ENDM
             END
<code>
После трансляции  M80  получим:
<code>
```

```
 MSX.M-80 1.00 01-Apr-85 PAGE 1
                                        .Z80
 0000'   AF                             XOR   a
 0001'   3A 0005'                       LD    a,(data)
 0004'   C9                             RET
 0005'                     data         EQU   $
                                        IRP   $P,<1,2,4,7>
                                        DB    $P
                                        ENDM
 0005'   01         +                   DB    1
 0006'   02         +                   DB    2
 0007'   04         +                   DB    4
 0008'   07         +                   DB    7
                                        END
<code>

Пример использования команды  IRPC:
<code>
       MSX.M-80 1.00 01-Apr-85 PAGE 1
                                       .Z80
  0000'   AF                           XOR   a
                                       IRPC  $A,BCDE
                                       OR    $A
                                       CP    $A
                                       ENDM
  0001'   B0     +                     OR    B
  0002'   B8     +                     CP    B
  0003'   B1     +                     OR    C
  0004'   B9     +                     CP    C
  0005'   B2     +                     OR    D
  0006'   BA     +                     CP    D
  0007'   B3     +                     OR    E
  0008'   BB     +                     CP    E
  0009'   C9                           RET
                                       END
```

3.1.3. Условная генерация

Условная генерация — генерация в зависимости от некоторых условий различающихся или различных последовательностей команд ассемблера. Для условной генерации в системе DUAD и в M80 используются конструкции вида:

```
IF   условие                  IF   условие
команды-ассемблера-1    команды-ассемблера-1
ENDIF                         ELSE
              команды-ассемблера-2
                              ENDIF
```

Команды-ассемблера-1 генерируются, если условие истинно, команды-ассемблера-2 генерируются, если условие ложно.

Команды условной генерации применяются обычно, когда одна и та же исходная программа должна быть настраиваемой на различные условия эксплуатации. Изменив несколько строк в начале программы и перетранслировав её, можно получить объектный код, рассчитанный например, на другой тип машины или другую её конфигурацию. Например, пусть исходный текст имеет следующий вид:

```
        ORG    9000h
MSX     EQU    0
MSX2    EQU    1
PRINTER EQU    1
        EX     DE,HL
```

```
IF      MSX
        CALL   4Dh
ELSE
        CALL   177h
ENDIF
        EX     DE,HL
IF      PRINTER
        CALL   0A8h
        JR     nz,round
        LD     A,65
        CALL   0A5h
round   EQU    $
ENDIF
        RET
        END
```

После трансляции ассемблером DUAD получим следующий листинг:

```
                  Z80-Assembler   Page:    1
                  ORG     9000h
0000 =    MSX     EQU     0
0001 =    MSX2    EQU     1
0001 =    PRINTER EQU     1
9000 EB           EX      DE,HL
          IF      MSX
                  CALL    4Dh
          ELSE
9001 CD7701       CALL    177h
          ENDIF
9004 EB           EX      DE,HL
          IF      PRINTER
9005 CDA800       CALL    0A8h
9008 2005         JR      NZ,round
900A 3E41         LD      a,65
900C CDA500       CALL    0A5h
900F =    round   EQU     $
          ENDIF
900F C9           RET
                  END
```

Обратите внимание, что код, соответствующий MSX, не генерировался. Ниже приведен пример трансляции ассемблером M80 для других условий. Сгенерировано всего 6 байт.

```
      MSX.M-80  1.00    01-Apr-85      PAGE    1
                                        .Z80
  0001                          MSX     EQU     1
  0000                          MSX2    EQU     0
  0000                          PRINTER EQU     0
  0000'   EB                            EX      DE,HL
                                IF      MSX
  0001'   CD 004D                       CALL    4Dh
                                ELSE
                                        CALL    177h
                                ENDIF
  0004'   EB                            EX      DE,HL
                                IF      PRINTER
                                        CALL    0A8h
                                        JR      NZ,round
                                        LD      A,65
                                        CALL    0A5h
                                round   EQU     $
                                ENDIF
  0005'   C9                            RET
                                        END
```

```
No Fatal error(s)
```

Для команд условной генерации обычно не допускается вложенность одного оператора IF в другой. Если же вложенность макроассемблером допускается, ELSE отвечает ближайшему IF, не имеющему ELSE.

3.2. Трансляция сегментов программ

При трансляции ассемблер использует текущее значение счётчика адреса памяти. Этим значением является адрес следующего байта, для которого транслятор генерирует код.

Однако адрес может быть как абсолютным, так и заданным относительно данных, кодов или общей памяти. Относительный адрес задает смещение к абсолютному стартовому адресу.

Тип адресации задается директивами ассемблеру — ASEG, CSEG, DSEG, COMMON.

Определение абсолютного сегмента

Директива ASEG задает абсолютный режим адресации. При этом генерируются абсолютные коды, жестко привязанные к одному участку памяти.

После директивы ASEG директива ORG должна использоваться с аргументом 103h или больше, причем она задаёт абсолютный адрес трансляции.

Определение сегмента относительно кодов

Директива CSEG задает режим трансляции относительно кодов. Относительные адреса в этом случае помечаются в листинге апострофом (') после адреса.

Если после CSEG не использована директива ORG, то значению счетчика адресов присваивается то значение, которое было последним в режиме CSEG (по умолчанию - 0).

Директива ORG в режиме CSEG задает не абсолютный адрес, а смещение к последнему значению адреса в режиме CSEG.

Если требуется в режиме CSEG установить абсолютный адрес, то для сборщика используется ключ /P.

Режим CSEG является стандартным режимом работы ассемблера.

Определение сегмента относительно данных

Для задания этого режима адресации используется директива DSEG. Признаком этого режима трансляции являются двойные кавычки после адреса (").

Как и в режиме CSEG, устанавливается то значение счетчика адреса, которое было последним в режиме DSEG, а директива ORG задает относительное смещение адреса.

Для установки абсолютного адреса в сборщике используется ключ /D.

Определение блока общей области

Директива `COMMON /[ имя-блока]/` определяет некоторую общую область данных для всех блоков COMMON, известных редактору связей, и является неисполняемой директивой резервирования памяти.

Признак этого режима трансляции — восклицательный знак (!) после адреса. Как и раньше, директива ORG задает относительный адрес.

Через общие блоки с одним и тем же именем разные подпрограммы могут обмениваться данными и результатами.

Смещение

Иногда требуется временно хранить программу в одном месте для последующего переписывания и выполнения в другом. Для этого используется директива

```
.PHASE  выражение.
```

Выражение должно иметь абсолютное значение.

Директива `.DEPHASE` используется для обозначения конца трансляции такого смещенного блока кодов.

Ниже приводится пример программы, использующей некоторые директивы управления адресами. Эта программа работает посредством обработки прерываний от таймера (60 раз в секунду). Напомним, что по этому прерыванию центральный процессор выполняет подпрограмму обработки прерывания, находящуюся по адресу 0038h.

Как и любая другая подпрограмма обработки прерывания, она начинается с сохранения регистров (путем засылки их в стек), затем вызывается ловушка этого прерывания (0FD9Ah), в которой вначале находится команда возврата (RET).

При инициализации наша программа перемещает свой код в область, начиная с адреса 4000h (которая интерпретатором языка BASIC не используется) и через ловушку прерывания устанавливает точку входа.

Суть самой программы заключается в том, что она два раза в секунду печатает системное время в правом верхнем углу экрана (SCREEN 0, WIDTH 80). Мы уже сказали, что используемое прерывание происходит 60 раз в секунду (во всей доступной авторам литературе указывается число 50), т.е. каждый тридцатый вызов этого прерывания указывает на то, что прошло 1/2 секунды.

Наша программа имеет счетчик, который увеличивается при каждом вызове подпрограммы обработки прерывания (поскольку сначала выполняется наша подпрограмма, а затем уже подпрограмма обработки прерывания), и если этот счетчик получает значение 29, то он обнуляется и выводится новое время.

Системное время считывается с микросхемы таймера при помощи стандартных функций BDOS.

Приводимая ниже программа написана в мнемонике Intel 8080.

```
   MSX.M-80   1.00   01-Apr-85    PAGE 1
;---------------------------------------------
;
;       (c) 1989 by Igor BOCHAROV.
;
;---------------------------------------------
; ИСПОЛЬЗУЕМЫЕ ФУНКЦИИ И ТОЧКИ ВХОДА
;
0006     CallF   EQU     0006h    ; МЕЖСЛОТОВЫЙ ВЫЗОВ
FD9A     H.KeyI  EQU     0FD9Ah   ; ОБРАБОТКА ПРЕРЫВАНИЙ
F37D     BDOS    EQU     0F37Dh   ; ВХОДНАЯ ТОЧКА  BDOS
002C     GetTime EQU     2Ch      ; ФУНКЦИЯ ЧТЕНИЯ
                ; ВРЕМЕНИ В  BDOS
9000     Load    EQU     9000h    ; ЗАГРУЗОЧНЫЙ АДРЕС
4010     Work    EQU     4010h    ; РАБОЧИЙ АДРЕС
;---------------------------------------------
        .8080    ; ИСПОЛЬЗУЕТСЯ МНЕМОНИКА  Intel
0000'             ASEG         ; АБСОЛЮТНЫЙ СЕГМЕНТ ПРОГРАММЫ
0000 FE           DEFB    0FEh             ; OBJ  -  ФАЙЛ
0001 9000         DEFW    Load             ; АДРЕС ЗАГРУЗКИ
0003 9095         DEFW    Load+PrgEnd      ; АДРЕС КОНЦА
0005 9000         DEFW    TimeInit         ; АДРЕС ЗАПУСКА
               ;
               ;---------------------------------------------
       ; Инициализация
                  .PHASE  Load       ; АДРЕС ТРАНСЛЯЦИИ ЗАГРУЗЧИКА
9000      TimeInit:
```

```
9000 F3          DI
9001 DB A8       IN      0A8h      ; ТЕКУЩЕЕ ПОЛОЖЕНИЕ СЛОТОВ
9003 F5          PUSH    psw
9004 3E AA       MVI     a,0AAh    ; УСТАНАВЛИВАЕМ, КАК НАМ НАДО
9006 32 FFFF     STA     0FFFFh
9009 3E FC       MVI     a,0FCh
900B D3 A8       OUT     0A8h
900D 21 9035     LXI     h,ForInit  ; АДРЕС БЛОКА ДЛЯ ЛОВУШКИ
9010 11 FD9A     LXI     d,H.KeyI   ; ЛОВУШКА ПРЕРЫВАНИЯ
9013 01 0005     LXI     b,5        ; ДЛИНА БЛОКА ДЛЯ ЛОВУШКИ
9016 CD 902A     CALL    Ldir       ; Intel 8080 НЕ ИМЕЕТ LDIR
9019 21 903A     LXI     h,PrgBeg   ; ПЕРЕПИСЫВАЕМ ПОДПРОГРАММУ
901C 11 4010     LXI     d,Work
901F 01 005B     LXI     b,PrgEnd-(PrgBeg-Load)
9022 CD 902A     CALL    Ldir
9025 F1          POP     psw        ; ВОССТАНОВЛИВАЕМ СОСТОЯНИЕ
9026 D3 A8       OUT     0A8h
9028 FB          EI
9029 C9          RET
902A 7E    Ldir: MOV     a,m        ; ПЕРЕПИСЫВАНИЕ БЛОКА
902B 12          STAX    d          ; в ловушку
902C 23          INX     h
902D 13          INX     d
902E 0B          DCX     b
902F 78          MOV     A,B
9030 B1          ORA     c
9031 C2 902A     JNZ     ldir
9034 C9          RET
9035      ForInit:
9035 F7          RST     CallF
9036 8B          DEFB    8Bh        ; СЛОТ RAM
9037 4010        DEFW    TimeH      ; АДРЕС НАШЕЙ ПОДПРОГРАММЫ
9039 C9          RET                ; ВОЗВРАТ ИЗ ЛОВУШКИ
903A    PrgBeg   EQU   $
                 .DEPHASE
;---------------------------------------------
; ПЕЧАТЬ АСТРОНОМИЧЕСКОГО ВРЕМЕНИ
; ВИСИТ НА ПРЕРЫВАНИИ im2 (rst 38)
;---------------------------------------------
; СОБСТВЕННО САМА ПРОГРАММА
;
                 .PHASE  Work    ; ОСНОВНАЯ ПРОГРАММА
4010 21 405F TimeH:LXI   h,MyJiffy        ; TIME
4013 34          INR     m
4014 7E          MOV     a,m     ; 30 ТИКОВ = 1/2 СЕК
4015 D6 1D       SUI     29
4017 C0          RNZ             ; ВОЗВРАТ, НЕ ПРОШЛО 1/2 СЕК.
4018 36 07       MVI     m,a     ; ОБНУЛЯЕМ СЧЕТЧИК СЕКУНД
401A 21 4062     LXI     h,Timer+2        ; ":" ИЛИ " "
401D 3E 1A       MVI     a,':' XOR ' ' ; ИНВЕРТИРУЕМ ':' НА ''
401F AE          XRA     M          ; ' '=>':', ':'=>' '
4020 77          MOV     M,A        ; СОХРАНЯЕМ НОВОЕ СОСТОЯНИЕ
4021 0E 2C       MVI     c,GetTime  ; ФУНКЦИЯ BDOS:
4023 CD F37D     CALL    BDOS       ; СЧИТАТЬ ВРЕМЯ!
4026 11 4060     LXI     d,Timer    ; АДРЕС СТРОКИ-ШАБЛОНА
4029 7C          MOV     A,H        ; ЧАСЫ
402A CD 404D     CALL    DaaDig     ; ПРЕОБРАЗУЕМ В ДЕСЯТ.ВИД
402D 13          INX     d          ; ПРОПУСТИТЬ ДВОЕТОЧИЕ
402E 7D          MOV     A,L        ; МИНУТЫ
402F CD 404D     CALL    DaaDig     ; ТОЖЕ ПРЕОБРАЗУЕМ
; === С ТЕКУЩИМ ВРЕМЕНЕМ
4032 F3          DI
4033 3E 4B       MVI     a,75     ; КУДА ВЫВОДИТЬ (МЛАДШИЙ БАЙТ)
4035 D3 99       OUT     99h
```

```
4037 3E 00       MVI     A,00     ; СТАРШИЙ БАЙТ
4039 F6 40       ORI     40h      ; ФЛАГ: ЗАПИСЬ ВО  VRAM
403B D3 99       OUT     99h
403D E3          XTHL             ; ЗАДЕРЖКА
403E E3          XTHL
403F 21 4060     LXI     h,Timer  ; АДРЕС СТРОКИ-ШАБЛОНА
4042 06 05       MVI     B,5      ; СКОЛЬКО
4044 7E          MOV     A,M      ; ВЫВЕСТИ!
4045 D3 98       OUT     98h
4047 23          INX     H
4048 05          DCR     B
4049 C2 4044     JNZ     $-5
404C C9          RET              ; ВОЗВРАТ
;---------------------------------------------
; ПОЛУЧЕНИЕ ДЕСЯТИЧНОГО ЧИСЛА В КОДЕ  ASCII
; ВХОД:    [a]  -  ЧИСЛО, [de]  -  КУДА ЕГО ЗАПИСАТЬ
; ВЫХОД:   (de) = ДЕСЯТИЧНОЕ ЧИСЛО,
;           [de] = [de] + 3
;----------------------
404D 06 2F DaaDig:MVI    b,'0'-1 ; ПОЛУЧЕНИЕ ДЕСЯТИЧ. ЧИСЛА
404F 04          INR     b            ; ПО АДРЕСУ В  [de]  В ASCII,
4050 D6 0A       SUI     10           ; ЧИСЛО В  [a]
4052 D2 404F     JNC     DaaDig+2
4055 C6 3A       ADI     '9'+1
4057 4F          MOV     C,A          ; ЧИСЛО В  [b]  И [c]
4058 78          MOV     A,B          ; СТАРШИЙ РАЗРЯД ЧИСЛА
4059 12          STAX    D
405A 13          INX     D
405B 79          MOV     A,C          ; МЛАДШИЙ РАЗРЯД
405C 12          STAX    D
405D 13          INX     D
405E C9          RET
;---------------------------------------------
; РАБОЧАЯ ОБЛАСТЬ
;
405F 00   MyJiffy:DEFB   0          ; СЧЕТЧИК СЕКУНД
4060 3F3F3A3F Timer:  DEFB  "??:??" ; ШАБЛОН ДЛЯ ВЫВОДА
4064 3F                             ;      ВРЕМЕНИ
;---------------------------------------------
                 .DEPHASE
0095     PrgEnd  EQU   $-1  ; ОТНОСИТЕЛЬНЫЙ АДРЕС КОНЦА
                 end
```

3.3. Макрокоманды

Еще одна возможность макрогенерации — использование макрокоманд. В этом случае группе команд дается имя, и каждое использование этого имени в тексте будет означать подстановку соответствующей группы команд в текст. Описание макрокоманды называют макроопределением. Оно выглядит следующим образом: ``` имя MACRO параметры

```
   команды-ассемблера
      ENDM
```

``` Список параметров может отсутствовать. Тогда использование имени макрокоманды в тексте будем обозначать просто подстановку вместо него соответствующей группы команд.

Например, имеется исходный текст: ```

```
                .Z80
```

SHIFT MACRO
```

```
              LD     A,B
              RLCA
              RLCA
              RLCA
              LD     B,A
              ENDM
              Ld     b,76
              SHIFT
              LD     b,34h
              SHIFT
              RET
              END
```

``` После его трансляции M80 получим: ``` ┌──────────────────────────

```
    MSX.M-80   1.00   01-Apr-85   PAGE 1
                            .Z80
                 SHIFT      MACRO
                            LD     A,B
                            RLCA
                            RLCA
                            RLCA
                            LD     B,A
                            ENDM
```

0000' 06 4C LD b,76

```
                            SHIFT
```

0002' 78 + LD A,B 0003' 07 + RLCA 0004' 07 + RLCA 0005' 07 + RLCA 0006' 47 + LD B,A 0007' 06 34 LD b,34h

```
                            SHIFT
```

0009' 78 + LD A,B 000A' 07 + RLCA 000B' 07 + RLCA 000C' 07 + RLCA 000D' 47 + LD B,A 000E' C9 RET

```
                            END
```

└──────────────────────────────── ```

Если в заголовке макрокоманды были указаны параметры, то вместо них в тексте будут подставлены те значения, которые были использованы при вызове макрокоманды. Например, ``` ┌──────────────────────────────

```
 'bcd-hex convrt' MSX.M-80 1.00 01-Apr-85 PAGE  1
                       .Z80
                       TITLE 'bcd-hex convrt'
              ; === conversion of BCD-nmb to binary
```

A000 BCDARG EQU 0A000h A001 HEXRES EQU 0A001h

```
              MPLY10   MACRO   $reg,$wreg
                       LD      $wreg,$reg   ; копируем
                       SLA     $reg         ; умнож. на 8
                       SLA     $reg
                       SLA     $reg
                       ADD     $reg,$wreg   ; прибавить 2
                       ADD     $reg,$wreg   ; раза
                       ENDM
       ; === берем аргумент
```

0000' 21 A000 LD HL,BCDARG ; arg. address 0003' 46 LD B,(HL) ; arg. in B 0004' AF XOR A ; clear A 0005' ED 6F rld

```
       ; === умножаем десятки на 10
```
```

```
                        MPLY10  A,C     ; * 10
```

0007' 4F + LD C,A ; копируем 0008' CB 27 + SLA A ; умнож. на 8 000A' CB 27 + SLA A 000C' CB 27 + SLA A 000E' 81 + ADD A,C ; прибавить 2 000F' 81 + ADD A,C ; раза 0010' 4F LD C,A

```
        ; === прибавляем единицы
```

0011' 78 LD A,B ; restore argument 0012' E6 0F AND 0Fh ; mask 0014' 81 ADD A,C ; add & get result

```
        ; === возврат
```

0015' 32 A001 LD (HEXRES),A ; 0018' C9 RET

```
                        END
```

└──────────────────────────────── ```

Локальные метки макрокоманды

Возможны случаи, когда в теле макрокоманды нужно использовать метки. Использовать обычную метку нельзя, потому что при втором вызове макрокоманды появится ошибка «повторно определенная метка». Макроассемблер позволяет обойти это ограничение при помощи использования локальных меток макрокоманды. Вместо таких меток макроассемблер подставляет свои метки особого вида в диапазоне ``` ..0000 Ў ..FFFF. ```

Рассмотрим пример с использованием локальных макрометок. ``` ┌────────────────────────────────

```
     MSX.M-80   1.00   01-Apr-85   PAGE 1
                               .Z80
```

00A5 PUTchr EQU 0A5h

```
                        PUT     MACRO   $SY
                        LOCAL   $D
                                LD      ($D),A
                                LD      A,$SY
                                CALL    PUTchr
                                JP      $D+1
                        $D:     DS      1,0
                                ENDM
                                PUT     66
```

0000' 32 000B' + LD (..0000),A 0003' 3E 42 + LD A,66 0005' CD 00A5 + CALL PUTchr 0008' C3 000C' + JP ..0000+1 000B' + ..0000: DS 1,0

```
                                PUT     42
```

000C' 32 0017' + LD (..0001),A 000F' 3E 2A + LD A,42 0011' CD 00A5 + CALL PUTchr 0014' C3 0018' + JP ..0001+1 0017' + ..0001: DS 1,0 0018' C9 RET

```
                                END
```

└──────────────────────────────── ```

Дополнительные возможности макрокоманд

Во время компиляции можно использовать так называемые переменные времени компиляции. Для присваивания значения такой переменной используется директива SET: ``` имя SET выражение. ```

Для управления печатью листинга макроассемблера можно использовать директивы:

- LALL — выводит полный текст макрорасширения; - SALL — только объектный код расширения без текста; - XALL — выводит те строки, которые генерируют текст.

Операции: - & — связывание метки и параметра, например, ERROR&X; - ;; — макрокомментарий; - ! — означает, что за ним — литерал. Например, «!;» означает символ точка с запятой. - % — преобразование выражения в число. Например, %X+Y.

Заключение

На этом мы заканчиваем изучение команд и директив языков ассемблера и макроассемблера для микропроцессора Z80 в среде MSX-2. В ограниченном объеме книги не удалось подробно осветить некоторые тонкие вопросы программирования, но авторы надеются, что некоторое представление об архитектуре MSX-2 и управлении устройствами этой системы внимательный читатель все же получил.

Желаем Вам успехов в программировании и надеемся, что эта книга предоставила Вам ответы на многие вопросы, касающиеся системы MSX-2. Авторы будут благодарны за все замечания и предложения по содержанию книги.
