1. ВВЕДЕНИЕ

Программный продукт "UniCROSS - Кросс-ассемблер" - мощный кросс-компилятор, позволяющий транслировать программы на языке ассемблера для микропроцессора СМ 601 (аналог процессора МС 6800) работая в среде операционной системы MS DOS или PC DOS на персональных компьютерах типа IBM PC XT/AT и совместимых с ними. Компьютер должен иметь как минимум 256 КВ оперативной памяти и одно запоминающее устройство на гибких магнитных дисках.

Продукт разработан специально для пользователей микрокомпьютера "Пълдин" в среде операционной системы UniDOS и учитывает их особенности. Полную информацию о компьютере и операционной системе можно найти в документах "UniDOS - Руководство пользователя" и "UniBIOS - Руководство пользователя".

Настоящий документ является руководством пользователя программного продукта и написан исходя из предположения, что читатель уже владеет программированием на ассемблере микропроцессора СМ601, а также усвоил основные понятия и приемы работы с операционными системами MS DOS (PC DOS) и UniDOS. Руководство не является учебником языка ассемблера.

При создании следующих версии транслятора изменения и дополнения к настоящему руководству будут записаны в файле READ_ME.CR, который будет поставляться в составе дистрибутивного материала.

Кросс-ассемблер UniCROSS позволяет генерировать выходные файлы трех типов:

— перемещаемые выполнимые файлы (.PGM);
— неперемещаемые выполнимые файлы (.CMD);
— стандартные объектные модули (.OBJ).

Объектные модули могут быть обработаны связывающим редактором UniLINK в перемещаемые программы.

Важно отметить, что объектные модули могут быть транслированы и отлажены независимо друг от друга. Таким образом общая отладка полной программы выполняется легко и быстро.

В описании кросс-транслятора UniCROSS приняты следующие обозначения:

<.....> - обязательная составляющая - вводится конкретное значение описанного. Например, <имя_файла> означает, что на указанном месте следует ввести имя файла;

[.....] - необязательная составляющая;

{.....} - составляющая, которая может быть обязательной или необязательной в зависимости от конкретнего случая - например, параметр в зависимости от команды.

Термин "символ" применяется для обозначения "графического символа" и "символа языка ассемблера". Интерпретация термина определяется подтекстом.

2. ВЫЗОВ КРОСС-АССЕМБЛЕРА

Кросс-ассемблер может быть вызван двумя командами. Первая из них имеет вид:

```
A:\> UNICROSS <source> <listing> <object>
```

где: <source> - имя вводного (исходного) файла. Если расширение имени не указано, то по умолчанию принимается расширение .ASM;
<listing> - имя файла листинга трансляции. Если расширение имени не указано, то по умолчанию принимается расширение .LST;
<object> - имя выходного файла. Если расширение не указано, то по умолчанию принимается .PGM.

Параметры <listing> и <object> можно заменить символом ";". В таком случае файлы листинга и результата будут иметь имена, одинаковые с именами вводного файла и соответствующие расширения по умолчанию.

Примеры:
```
Unicross try_1.txt try_1.pro try_1

unicross try_1;
```

Кросс-ассемблер может быть вызван без указания параметров командой:

```
unicross
```

После загрузки программы необходимые для работы параметры будут уточнены последовательными вопросами:

```
Source filename [.ASM]:
Listing filename [.LST]:
Object filename [.PGM]:
```

Если после ввода имени исходного файла будет введен символ ";", диалог прекратится и остальные параметры будут определены по умолчанию.

3. ФОРМАТ ИСХОДНОГО ФАЙЛА

Длина строки исходного файла должна быть не более 255 символов. В одной строке должна быть записана только одна инструкция.

Каждая строка состоит из 4 полей:

```
[метка] <операция> {операнд} [комментарий]
```

Соседние поля разделяются одним или более пробелами или табуляциями. В файле листинга поля разделяются одной табуляцией. В целях улучшения читаемости программы рекомендуем начинать каждое поле с постоянной позиции строки.

Пустые строки и строки, в первой колонке которых записан символ ";", считаются строками комментария и не обрабатываются транслятором.

При синтаксическом анализе строчные и прописные буквы неразличимы, кроме случая строки символов в качестве параметра.

ПОЛЕ МЕТКИ

Если первый символ строки не пробел, табуляция или ";", то первое поле каждой строки - поле метки. Поле метки необязательное. Допускается ввод строки, содержащей только поле метки.

Метка рассматривается как определенный в программе символ, значение которого равняется текущему значению счетчика адресов.

Счетчиком адреса во время трансляции определяется адрес или сдвиг инструкций программы в памяти. Исходное значение счетчика адресов равно начальному адресу программы. После каждой обработанной строки значение счетчика увеличивается на длину инструкции. Таким образом, метка является адресом следующей за ней инструкции.

Например, если текущее значение счетчика равно $100, при вводе инструкций

```
loop        aba
abcd
```

метке loop будет присвоено значение $100, а метке abcd - значение $101.

Метки могут быть использованы в программах для переходов к соответствующим командам, без необходимости подсчета адресов.

Например:

```
bne   loop
```

Таким образом программа становится независимой от ее расположения в памяти.

Значительное удобство для программиста представляет возможность разделения вводного файла на процедуры. Все метки в одной процедуре имеют локальное значение. Существует специальная инструкция для объявления определенной метки глобальной.

В рамках одной процедуры доступны ее собственные метки, метки охватывающих ее процедур ("родительских" процедур) и глобальные метки.

Допустимо переопределение метки родительской процедуры, но оно имеет локальное значение (только в рамках процедуры, где осуществлено переопределение).

ПОЛЕ ОПЕРАЦИИ

В строке программы поле операции следует за полем метки. Если поле метки отсутствует, перед полем операции должен быть введен хотя бы один пробел или табуляция.

Поле операции может содержать мнемоническое обозначение машинной команды или инструкцию транслятора.

Если в поле операции записана команда, то генерируется соответствующий ей машинный код. Если записана директива - она выполняется.

Все директивы описаны ниже, а мнемонические обозначения всех машинных команд процессора СМ 601 даны в приложении Б.

В поле операции может быть записана команда INT xx вызова определенной специальной функции операционной системы UniDOS INT xx (xx - шестнадцатеричное число номера функции). Все функции INT xx описаны в документе "UniBIOS - Руководство пользователя".

ПОЛЕ ОПЕРАНДА

Содержание поля операнда определяется операцией. Операнд указывает объект действия команды.

Если команда двухоперандная, первый операнд записывается как часть мнемоники команды. Например, для команды ADD можно записать ADDA или

ADDB в зависимости от того, над каким регистром выполняется операция.

В качестве операнда могут быть записаны: метка; константа; вычисляемое выражение, список выражений, строка символов. В некоторых командах операнды отсутствуют.

В выражениях допустимы следующие арифметические и логические операции в нисходящем порядке приоритета выполнения (от найболее приоритетных к наименее приоритетным). Операции одного приоритета выполняются в порядке следования. В кавычках указаны графические отображения операций.

```
Приоритет 1     — скобки "(" и ")";

Приоритет 2     — логическое отрицание - "!";
                — смена знака - "-";

Приоритет 3     — умножение - "*";
                — деление - "/";
                — получение остатка деления - "%";
                — логическое И - "&";
                — включающее логическое ИЛИ - "|";
                — исключающее логическое ИЛИ - "^";

Приоритет 4     — сложение - "+";
                — вычитание - "-".
```

В арифметических операциях могут принимать участие в качестве операндов метки, при соблюдении следующих ограничений, в случаях выходного файла типа PGM или OBJ:

— допустимо только сложение метки с константой и вычитание метки из метки. В последнем случае в результате операции получается константа;

— если директивой EXTERN метка объявлена внешней, то она не может участвовать в арифметических выражениях.

В качестве операнда в выражении допустимо применение символа "*", которым обозначается текущее значение счетчика адреса. Например:

```
JMP   * + 3
```

В ассемблере допустимы константы следующих типов:

— десятичное число

```
Пример: 123 234 456
```

— шестнадцатеричное число - идентифицируется знаком "$"

```
Пример: $AB $12 $F2
```

— двоичное число - идентифицируется знаком "%"

```
Пример: %11 %01 %0001
```

— графический символ - заключается в апострофы или кавычки и константе присваивается значение, равное ASCII-коду символа.

```
Пример: 'ш' '' 'A'
```

Константы ассемблера представляются шестнадцатеричным числом без знака.

Счетная система чисел может быть изменена директивой RADIX.

ПОЛЕ КОММЕНТАРИЯ

Первым символом поля комментария является символ ";". В поле комментария можно записать любые символы. Они не влияют на ход трасляции и на генерируемый код.

Напомним, что хороший стиль программирования требует применения комментариев, поясняющих действие программы. Комментарии незаменимы в процессе отладки программ.

4. АДРЕСАЦИЯ МИКРОПРОЦЕССОРА СМ 601

Микропроцессор СМ 601 допускает 7 видов адресации, представленных в следующей таблице:

Наименование		Синтаксис	Длина	Пример	
Внутренная	Inherent		1	CLC	
Аккумуляторная	Accumulator		1	CLRA	
Непосредственная	Immediate	#выражение	2(3)	LDAB	#$FF
Прямая	Direct	выражение	2	LDAB	0
Расширенная	Extended	выражение	3	STAA	$FFFF
Индексная	Indexed	выражение,X X,выражение	2	LDX	0,X
Относительная	Relative	выражение	2	BRA	.

Операции с ВНУТРЕННЕЙ И АККУМУЛЯТОРНОЙ адресацией не предусматривают операнды. Примеры:

```
asla
clc
comb
decb
```

Операции с НЕПОСРЕДСТВЕННОЙ адресацией имеют синтаксис:

```
<код_операции> #<выражение>

<код_операции> #/<выражение>
```

Если в поле операнда введено #<выражение>, то учитывается младший байт значения выражения, если инструкция не относится к 16-битовым регистрам.

Если в поле операнда введено #/<выражение>, то учитывается старший байт значения выражения.

Примеры:

```
ldaa  #$FF
adcb  #1
ldx   #$FFFF
adda  #/end_code - start_code
```

Операции с ПРЯМОЙ адресацией имеют синтаксис:

```
<код_операции> <выражение>
```

Значение выражения должно быть в интервале [0, 255]. Прямая адресация охватывает только нулевую страницу памяти - адреса в интервале [$00, $FF]. Примеры:

```
ldab  0
stab  $20
stx   $FE
```

РАСШИРЕННАЯ адресация отличается от прямой только тем, что значением выражения в поле операнда может быть любой доступный адрес памяти компьютера. Примеры:

```
ldaa  $2000
stx   $1FFE
```

Операции с ИНДЕКСНОЙ адресацией имеют синтаксис:

```
<код_операции> X,<выражение>
<код_операции> <выражение>,X
<код_операции> X
```

В третьем случае считается, что выражение имеет значение 0.

Примеры:

```
ldaa  X
adda  X,1
ldx   X,field1
ldab  $33,X
```

Операции с ОТНОСИТЕЛЬНОЙ адресацией имеют синтаксис:

```
<код_операции> <выражение>
```

К этой группе операции относятся операции условного и безусловного перехода, исключая JSR и JMP. В качестве операнда принимается значение выражения за вычетом текущего значения счетчика адреса. Значение операнда должно быть в интервале [-129, +125].

Примеры:

```
bra   exit
bne   not_equ
```

5. ДИРЕКТИВЫ КРОСС-АССЕМБЛЕРА

5.1. EQU

СИНТАКСИС: <метка> EQU <выражение>
<метка> = <выражение>

ПРЕДНАЗНАЧЕНИЕ: присвоение в качестве значения метки значения выражения.

<метка> не должна быть уже определенной. В <выражение> не должна содержаться внешняя метка (объвленная директивой EXTERN). Примеры:

```
code_start =    $100
lo_byte         EQU   hi_byte + 1
```

5.2. DB (define byte)

СИНТАКСИС: DB <выражение> [,<выражение>, ...]

ПРЕДНАЗНАЧЕНИЕ: запись в объектном коде одного или более байтов, каждый из которых является значением очередного выражения. Если выражение имеет значение больше 255, то учитывается младший байт значения выражения.

<выражение> может быть арифметическим выражением или строкой символов, заключенных в апострофы или кавычки.

Пример:

```
num_base   DB    16
hex_chars  DB    '0123456789ABCDEF'
string     DB    'This is a string.', 13, 10, 0
hi_bytes   DB    first / 256, second / 256
lo_bytes   DB    first, second
```

5.3. DW (define word)

СИНТАКСИС: DW <список арифметических выражений>

ПРЕДНАЗНАЧЕНИЕ: запись в объектном коде последовательности слов (старший байт, младший байт), каждое из которых является значением очередного выражения.

Примеры:

```
words DW    $C000, $D000, $E000, $F000
      DW    5*4096 + 3*16 +1
      DW    first_table, second_table, 0
```

5.4. DS (define space)

СИНТАКСИС: DS <выражение_1> [,<выражение_2>]

ПРЕДНАЗНАЧЕНИЕ: заполнение области памяти длиной в <выражение_1> байтов символами, являющимися значениями <выражение_2>.

В выражениях не должны участвовать метки. Если значение выражения_2 больше 255, то учитывается младший байт значения. Если выражение_2 не указано, область заполняется нулями.

Примеры:

```
buffer      DS    $200
            DS    $10, 32
```

5.5. SECTION

СИНТАКСИС: SECTION [выражение]

ПРЕДНАЗНАЧЕНИЕ: формирование фиктивной секции, начиная с адреса, являющегося значением выражения. Не допускается вложение фиктивных секций. Конец фиктивной секции формируется директивой ENDS. Внутри фиктивной секции не генерируется код.

5.6. ENDS

СИНТАКСИС: ENDS

ПРЕДНАЗНАЧЕНИЕ: формирование конца фиктивной секции, начало которой задано директивой SECTION. При выполнении директивы счетчику адреса присваивается значение, которым обладал до выполнения директивы SECTION. Недопустимо использование директивы вне фиктивной секции.

Пример фиктивной секции:

```
            SECTION 0
            DS   $80
byte_1      DS   1
byte_2      DS   1
byte_3      DS   1
word_1      DS   2
word_2      DS   2
word_3      DS   2
ENDS
```

5.7. PROC (procedure)

СИНТАКСИС: <метка> PROC

ПРЕДНАЗНАЧЕНИЕ: определение начала процедуры. Конец определяется директивой ENDP.

Одна процедура может содержать в себе неограниченное число других, вложенных процедур.

Все метки в одной процедуре, не обявленные глобальными директивой GLOBAL, считаются локальными. Поиск некоторой метки осуществляется сначала в текущей процедуре, а потом среди меток "родительских" процедур.

5.8. ENDP (end of procedure)

СИНТАКСИС: ENDP

ПРЕДНАЗНАЧЕНИЕ: определение конца процедуры. Начало определяется директивой PROC.

5.9. GLOBAL

СИНТАКСИС: GLOBAL <список меток>

ПРЕДНАЗНАЧЕНИЕ: объявление одной или более меток в качестве глобальных.

Директива GLOBAL должна следовать непосредственно за директивой PROC.

Пример процедуры:

```
print PROC
          GLOBAL print_1, print_2
          .
          .
print_1   ldx   #message_1
          .
print_2   ldx   #message_2
          .
          .
          ENDP
```

5.10. ORG

СИНТАКСИС: ORG <выражение>

ПРЕДНАЗНАЧЕНИЕ: присвоение значения счетчику адреса.

Выражение не должно включать метку и результат его вычисления должен быть типа константы. Не разрешается применение директивы внутри фиктивной секции.

5.11. EXTERN

СИНТАКСИС: EXTERN <список меток>

ПРЕДНАЗНАЧЕНИЕ: объявление внешних для данного модуля меток.

5.11. PUBLIC

СИНТАКСИС: PUBLIC <список меток>

ПРЕДНАЗНАЧЕНИЕ: объявление меток данного модуля, которые должны быть доступными для других модулей программы.

5.12. ERROR

СИНТАКСИС: ERROR <выражение>

ПРЕДНАЗНАЧЕНИЕ: возбуждение ситуации "ОШИБКА" в случае, если значение выражения отлично от нуля.

Пример:

```
ERROR * % $100   ; ошибка, если текущее
                 ; значение счетчика адреса не
                 ; равно границе страницы
                 ; памяти
```

5.13. RADIX

СИНТАКСИС: RADIX <выражение>

ПРЕДНАЗНАЧЕНИЕ: изменение текущей счетной системы на систему с основанием, определенным значением выражения. Допустимы значения выражения в интервале [2, 16].

Пример:

```
        ldaa    #$3F
        RADIX 16
        ldaa    #3F
```

5.14. INCLUDE

СИНТАКСИС: INCLUDE <имя_файла>

ПРЕДНАЗНАЧЕНИЕ: включение в транслируемый файл другого файла с именем <имя_файла>.

Параметр <имя_файла> должен задавать полное имя включаемого файла с указанием устройства и/или директории, если они отличны от текущих. Включаемый файл может содержать, в свою очередь, директиву INCLUDE. Допускаются до 8 уровней вложения директивы.

Пример:

```
        INCLUDE    b:\dir_1\memory.inc
```

5.15. LIST

СИНТАКСИС: LIST <ключ>

ПРЕДНАЗНАЧЕНИЕ: включение/выключение генерации файла листинга кросс-ассемблирования. Допускаются два значения ключа: ON - включить (по умолчанию); OFF - выключить.

Пример:

```
LIST ON
.
.
LIST OFF
```

5.16. TRUNC

СИНТАКСИС: TRUNC <ключ>

ПРЕДНАЗНАЧЕНИЕ: включение/выключение прерывания после первой строки генерации листинга в зоне директив DB, DW и DS. Допускаются два значения ключа: ON - включить (по умолчанию); OFF - выключить.

Удобство, предоставляемое этой командой программисту, заключается в повышении читаемости листинга путем его сокращения за счет малоинформативных частей.

5.17. END

СИНТАКСИС: END [выражение]

ПРЕДНАЗНАЧЕНИЕ: формирование конца программы со стартовым адресом, равняющимся значению выражения в случаях ассемблирования перемещаемых модулей (.PGM и .OBJ). Текст после директивы END не обрабатывается. Директива необязательна.

Если выражение пропущено, подразумевается стартовый адрес, соответствующий началу программы.

Пример:

```
END  start ; start - метка в рамках программы
```

5.18. CHKSUM (check sum)

СИНТАКСИС: CHKSUM

ПРЕДНАЗНАЧЕНИЕ: формирование в неперемещаемой выполнимой программе (.CMD) контрольного байта, переводящего контрольную сумму файла в ноль. Контрольная сумма считается однобайтовым сложением без переноса.

Рекомендуем применение директивы в случаях ассемблирования программ, предназначенных для записи в постоянной (ROM) памяти микрокомпьютера.

6. ДИАГНОСТИЧЕСКИЕ СООБЩЕНИЯ

Диагностические сообщения выдаются в следующем формате:

```
<имя_файла>(<номер строки>): ERROR: <текст сообщения>
```

Тексты сообщений на английском языке приведены ниже в алфавитном порядке с переводом и комментариями, если характер ошибки не очевиден.

6.1. Branch out of range. — Переход вне модуля

6.2. Can't open include file. — Нельзя открыть файл для вложения.

6.3. Can't open listing file. — Нельзя открыть файл листинга.

6.4. Can't open source file. — Нельзя открыть исходный файл. - вероятнее всего не существует файл с указанным именем.

6.5. Digit out of radix. — Цифра вне текущей счетной системы.

6.6. ENDS without SECTION. — применение директивы ENDS без директивы SECTION.

6.7. Error writing object file. — Ошибка во время записи объектного файла - вероятнее всего нет свободного пространства на диске.

6.8. Expected close bracket. — Отсутствует закрывающая скобка.

6.9. Expected close quote. — Отсутствует закрывающая кавычка или апостроф.

6.10. Expected constant expression. — Не найдено обязательное выражение типа константы.

6.11. Expected expression. — Не найдено обязательное для ввода выражение.

6.12. EXTERN not allowed in not relative mode. — Директива EXTERN запрещена в случаях неперемещаемого выходного файла.

6.13. General syntax error. — Общая синтаксическая ошибка - вероятнее всего допущена ошибка при написании имени инструкции или директивы.

6.14. Illegal address mode. — Неправильный тип адресации.

6.15. Illegal character in text. — В тексте найден не разрешенный символ.

6.16. Illegal expression. — Неправильное арифметическое выражение.

6.17. Illegal name. — Недопустимое имя - недопустимое сочетание символов в поле метки.

6.18. Illegal size of operand. — Недопустимое значение операнда - вероятнее всего значение операнда типа байт превышает 255.

6.19. Improper use of CHECKSUM directive. — Неправильное применение директивы CHKSUM. - директива CHKSUM применена второй раз или применена в случае генерации перемещаемого выходного файла.

6.20. Include file name expected. — Не введено имя файла для вложения.

6.21. Include files too nested. — Уровень вложения исходных файлов больше допустимого (до 8 уровней).

6.22. Integer overflow. — Целочисленное переполнение (число больше 65535).

6.23. Missing ENDS. — Отсутствует директива ENDS.

6.24. ORG inside SECTION.—применениедирективыORGвнутрификтивной секции.

6.25. ORG not allowed in relative mode. — Не разрешено применение директивы ORG в режиме генерации перемещаемого выходного файла.

6.26. Output buffer overflow. — Переполнение буфера объектного файла - длина объектного (выходного) файла не должна быть больше 32КВ.

6.27. Proc nesting error. — Неправильное вложение процедур.

6.28. PUBLIC not allowed in not relative mode. — Директива PUBLIC запрещена в случаях неперемещаемого выходного файла.

6.29. Redefinition of symbol <метка>. — Переопределение метки <метка>.

6.30. There is no memory available. — Нет больше доступной (свободной) памяти.

6.31. Undefined identifier <имя>. — Неопределенный идентификатор <имя>.

6.32. Unexpected directive END. — Неправильное применение директивы END. - вероятнее всего директива END применена во вложенном файле.

6.33. User error. — Потребительская ошибка - сообщение генерируется во время выполнения директивы ERROR, если значение операнда отлично от нуля.

Приложение А - ФОРМАТ ФАЙЛА ЛИСТИНГА

Каждая строка файла листинга имеет следующий формат:

```
<строка> <адрес> <код>     <строка вводного файла>
```

Пример:

```
1  0000                              org   $100
2  0100  CE 01 05                    ldx   #message
3  0103  3F 23                       int   $23
4  0105
5  0105  48 65 6C 6C  message        db     'Hello world !', 0
6  0113                              end
```