Сапожников А.П.
Б2-11-85-659

ОБЪЕДИНЕННЫЙ ИНСТИТУТ ЯДЕРНЫХ ИССЛЕДОВАНИЙ

Б2-11-85-659

ДЕПОНИРОВАННАЯ ПУБЛИКАЦИЯ

Дубна 19 85г

ОБЪЕДИНЕННЫЙ ИНСТИТУТ ЯДЕРНЫХ ИССЛЕДОВАНИЙ
Лаборатория вычислительной техники и автоматизации

А.П.Сапожников

КРОСС-СИСТЕМА М И К Р О Б

Дубна, 1985.

Рукопись поступила
в издательский отдел
.. 06 ---- 09 -- 1985

В процессе создания ЭВМ, работающих по принципу микропрограммного управления, достаточно трудоемкой работой является подготовка микропрограмм. Кроме того, для сокращения сроков разработки в целом, изготовление микропрограмм необходимо вести параллельно с изготовлением аппаратуры, используя кросс-средства на инструментальных ЭВМ. К числу таких кросс-средств принадлежит и описываемая здесь кросс-система МИКРОБ, работающая на ЭВМ БЭСМ-6.

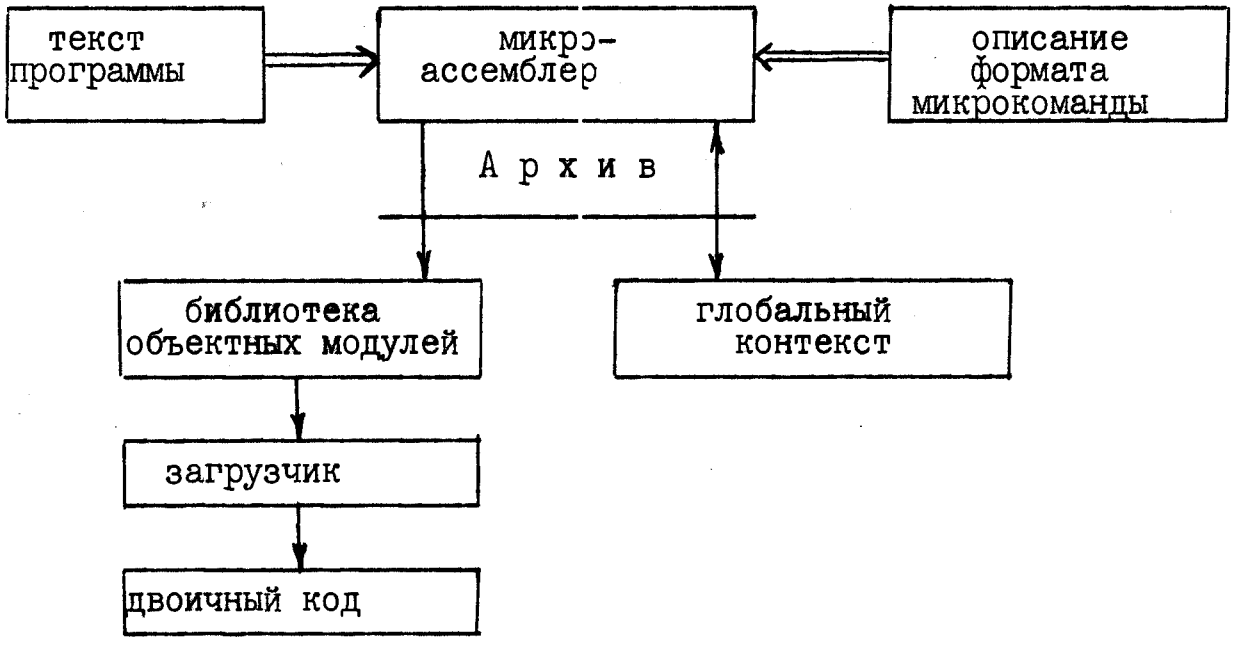
ОБЩИЕ ПОЛОЖЕНИЯ

Микропрограмма - последовательность слов микропрограммной памяти (микрокоманд), адресуемых целочисленным адресом = 0,I,2, ... Все микрокоманды имеют одну и ту же разрядность N , определенную заранее.

Микрокоманда состоит из фиксированного числа частей (полей) заранее определенных размеров. Каждое поле выполняет самостоятельную функцию, смысл которой определяется аппаратурой. Исключение составляют два поля: поле кода микрооперации и поле адреса следующей микрокоманды, функции которых одинаковы при любом микропроцессоре.

Такой подход характерен для т.н. горизонтального программирования. При этом микроассемблер, по существу, представляет из себя генератор битовых строк постоянной длины. Универсальность же его заключается в способности настраиваться на заданный форма микрокоманды.

Архитектура кросс-системы МИКРОБ



Кросс-система МИКРОБ имеет много общего со "взрослыми" системами программирования на языках высокого уровня:

- допускается раздельная трансляция подпрограмм;
- результат трансляции записывается в библиотеку модулей загрузки;

- для получения готовой двоичной программы используется связывающий загрузчик.

I. Вызов транслятора

Пакет задачи, запускаемой на БЭСМ-6 с целью трансляции микропрограмм, выглядит следующим образом:

```
*NAME .....
*PASS:.....
*TIME:.....
*LIBRARY:23,25
< ЗАКАЗ АРХИВА ПОД НОМЕРОМ 30 >
< ЗАКАЗ, ЕСЛИ НАДО, ФАЙЛА С ТЕКСТОМ >
*CALL MICROB [:<УК.МЛ >]
```

или:

```
*CALL MICINI [:< УК.МЛ >]
```

В квадратные скобки [ ] здесь и далее мы будем заключать необязательные параметры или синтаксические конструкции. Здесь необязательный параметр <УК.МЛ> задает адрес текста микропрограммы на внешнем носителе. При отсутствии <УК.МЛ> текст читается из пакета задачи. Признаком конца текста считается карта *END

MICROB - вход для трансляции. При этом считается, что описание формата микрокоманды (т.н. глобальный контекст) уже записано в архив ассемблера.

MICINI - вход при первоначально пустом архиве. При этом текст, подлежащий трансляции, предваряется описанием формата микрокоманды.

2. Трансляция глобального контекста

Глобальный контекст микропрограммы состоит из

- описания формата микрокоманды;
- определения глобальных имен.

Глобальный контекст оформляется в виде псевдопрограммы, имеющей имя DEFINE :

```
DEFINE:PROG <число >                    - заголовок
    ..........
      END:                              - конец
```

Здесь < число > задает разрядность микрокоманды N. биты в микрокоманде нумеруются справа налево: N,N-1,.,.,2,1.

Внутри псевдопрограммы DEFINE находятся инструкции , определяющие поля микрокоманды и глобальные имена, которые считаются предописанными во всех микропрограммах.

Несколько замечаний терминологического характера:

<число> - целое десятичное, если нет специальных оговорок;

<имя> - идентификатор в смысле алгола, длиной не более 6 символов;

<выражение> - список термов, отделенных друг от друга знаками "+" или "-";

<терм> - число, имя или специальный символ "*" .

По аналогии с ассемблерами уровня 2, на язык так и просятся слова "переменная" и "значение". Однако в микропрограммах не бывает рабочих ячеек памяти, а содержимое полей микрокоманд не меняется в ходе работы. Поэтому, говоря о микроассемблере, мы должны иметь в виду, что любое имя в программе - это всего лишь мнемоника для обозначения числа или адреса.

Все-таки в дальнейшем, ради краткости, мы будем называть значением выражения сумму его термов, взятых со своими знаками. Термы, входящие в выражение, могут быть трех типов:

- число, количество термов этого типа в выражении не ограничено;

- адрес относительно начала "своей" программы. В правильном выражении должно выполняться соотношение

$$0 <= NP-NN <= 1 ,$$

где NP - число относительных адресов со знаком "+" ,

NN - число относительных адресов со знаком "-";

- адрес внешней программы. В выражении допускается только один терм этого типа, со знаком "+", и при условии, что NP=NN.

2.I. Формат микрокоманды

Кроме разрядности N формат микрокоманды характеризуется еще набором полей, ее составляющих. Для описания одного поля служит инструкция FIELD :

< имя поля >:FIELD N1,N2 [,N3]

N1,N2 – числа, задающие левую и правую границу поля (в любом порядке);

N3 – выражение, значение которого задает начальное содержимое поля. При неуказании N3 считается нулем;

Допускается вложение одних полей в другие, перекрытие полей не допускается (аналогично фортранным DO -циклам). Порядок описания полей - произвольный.

Для поля кода операции секвенсора в качестве параметра N3 необходимо указывать маркирующую конструкцию /C/

Для поля адреса микропрограммы в качестве параметра N3 необходимо указывать маркирующую конструкцию /A/

2.2. Определение глобальных имен

Глобальные имена определяются с помощью инструкций EQU и BLOCK :

< ИМЯ > :EQU < ВЫРАЖЕНИЕ > ;

Здесь новое < имя > определяется как число, равное значению < выражения >. Трансляция глобального контекста производится за один проход, поэтому все имена, входящие в < выражение >, должны быть определены.

< ИМЯ > :BLOCK < СПИСОК ИМЕН > ;

Этот способ "группового" определения имен, аналогичный фортрановскому COMMON -блоку, < имя > должно быть уже определено.

```
NAME:BLOCK NAM1,NAM2(2),NAM3;                    -эквивалентно

NAM1:EQU NAME;  NAM2:EQU NAM1+1;  NAM3:EQU NAM2+2;
```

В левой части инструкции BLOCK разрешено указывать шестнадцатиричные числа с префиксом "*":

```
*1A:BLOCK A,B(5),C;                              - эквивалентно
A:EQU $1A;    B:EQU $1B;    C:EQU $20;
```

Шестнадцатиричные числа в правой части инструкции (входящие в качестве термов в выражения) снабжаются префиксом " $ " /доллар/).

2.3. Немного о синтаксисе

На одной карте находится одна инструкция. Формат - не фиксированный. Инструкция может начинаться с любой позиции карты. Пробелы, вообще говоря, являются разделителями. Пробелы

Пробелы между разделителем и началом следующей лексемы игнорируются. Разделители: двоеточие, точка с запятой, запятая, плюс, минус. Общий формат инструкции:

[< ИМЯ > :]< КОД ИНСТРУКЦИИ > [< ПАРАМЕТРЫ >] ;

Все символы после ";" до конца текущей карты рассматриваются как комментарий. В частности, на карте может находиться только комментарий, если ";" расположена в I-й позиции. Если после инструкции комментария нет, символ ";" можно опустить.

Недолгий опыт работы с микроассемблером показал, что очень часто для размещения инструкций одной карты не хаватает. Поэтому принято такое решение: символ " > ", находящийся в первой позиции, означает, что эта карта является продолжением предыдущей. Ассемблер способен "заглядывать вперед" по файлу чтения. Обнаружив " > " в первой позиции следующей карты, он просто "подклеивает ее в хвост" текущей. Число карт продолжения - не более 20.

2.4. П Р И М Е Р:

```
*LIBRA:23,25
*FILE:SCRATCH,30,W
*CALL MICINI
 DEFINE:PROG 72;          - разрядность = 72
;      определим поля микрокоманды:
 SEQV:FIELD 72,64,/C/;  - код микрооперации
    A:FIELD 50,63,/A/;  - адреса переходов
 АЛУ :FIELD 49,40;       - КОП АЛУ
   CI:FIELD 39,
>39,1;                   - пример карты-продолжения
;      определим коды секвенсора:
 JUMP:EQU 0
 TEST:EQU JUMP+1
;      определим коды А Л У :
 *O:BLOCK AND,OR,SUB,ADD,MUL:
;      и еще несколько глобальных имен:
  ONE:EQU 1
  TWO:EQU 2-ONE+1;
  TEN:EQU 9+TWO-ONE
      END;
*END
```

3. Трансляция микропрограмм

Вся микропрограмма складывается из отдельных блоков - подпрограмм. Допускается раздельная трансляция подпрограмм. Результатом трансляции является модуль загрузки, записанный в библиотеку объектных модулей. При этом, если в библиотеке уже была программа с таким же именем, то старая версия уничтожается (аналогично библиотекам в мониторной системе "Дубна"). При трансляции используется уже имеющийся в архиве глобальный контекст. Перечислим теперь синтаксические конструкции, из которых складывается подпрограмма:

1. заголовок подпрограммы:
   < имя подпрограммы >:**PROG**;
2. Конец подпрограммы
   END;
3. описание точки входа (**ENTRY**)
   < имя входа >:**ENTRY**;
4. Описание внешних программ
   EXTERN < СПИСОК ИМЕН >;
5. Определение локальных имен
   < ИМЯ > :**EQU** < ВЫРАЖЕНИЕ >;
   < ИМЯ ИЛИ *< ШЕСТН. ЧИСЛО >>:**BLOCK** < СПИСОК ИМЕН >;

Эти конструкции "работают" так же, как в 2.2. Но, в отличие от глобального контекста, здесь значением выражения может быть не только число, но и адрес. Допускается использование еще не определенных имен. Переопределение глобальных имен не допускается.

6. Изменение начального содержимого поля
   < ИМЯ ПОЛЯ > :**VALUE** < ВЫРАЖЕНИЕ >;

Начальным состоянием заданного поля считается значение < выражения > в правой части инструкции. Инструкция "работает" только в пределах транслируемой подпрограммы. При переходе к следующей подпрограмме ассемблер вновь извлекает информацию о начальных значениях полей из глобального контекста.

7. Микрокоманда
   [< МЕТКА > :] < КОП > < СПИСОК ПОЛЕЙ И ИХ ЗНАЧЕНИЙ >;

КОП - код микрооперации (его еще иногда называют кодом операции секвенсора). Это символическое имя, определенное в глобальном или локальном контексте.

Например: *6:BLOCK POP,PUSH;

Ассемблер подставляет значение имени <КОП> в то поле микрокоманды, которое помечено маркировкой /C /) (см. 2.I).

Заполнение остальных полей микрокоманды производится по общему алгоритму, на основании <списка полей> в правой части микрокоманды. Элементы списка задаются в произвольном порядке и отделяются друг от друга запятыми, каждый элемент списка представляется либо парой:

<имя поля> = <выражение>

либо просто именем поля (для однобитовых полей). Если в списке какое-то поле не указано, то в команду заносится начальное значение этого поля, указанное инструкцией **VALUE** или взятое из глобального контекста.

Здесь возможны неоднозначности при наличии вложенных полей. Для определенности принято следующее решение: незаполненные поля просматриваются в порядке убывания их ширины (т.е. расстояния между левой и правой границами). Далее, если поле получило значение, то заполненными считаются и все вложенные в него поля.

Маркировка адресного поля (см.2.I) позволяет отслеживать случаи ошибочного заполнения неадресных полей адресной информацией.

Примеры микрокоманд:

Рассмотрим небольшой фрагмент микропрограммы. При этом мы воспользуемся глобальным контекстом, описанным в (2.4).

```
CHECK1:TEST A=CHECK2,CI, АЛУ=SUB
       JUMP A=*-1
CHECK2:JUMP A=INTER, АЛУ=ADD
```

Поле **SEQV** (оно отмечено маркировкой /C/) заполняется в первой команде кодом I (**TEST=1**), а во второй и третьей - кодом 0 (**JUMP=0**).

В поле **A** попадут соответственно адреса меток **CHECK2,CHECK1** и **INTER** . Терм "*" в адресных выражениях, как и в большинстве обычных ассемблеров, означает текущее состояние счетчика относительных адресов внутри подпрограммы. Окончательная настройка программы по адресам выполняется загрузчиком.

В поле АЛУ попадает в первом случае код 2, в третьем - код 3. Для второй команды "сработает" правило умолчания, и в поле АЛУ попадает код 0.

Поле CI во всех трех командах равно I. В первой команде оно явно указано в списке полей (наличие имени однобитового поля в списке влечет заполнение его кодом I). В остальных командах оно заполнится "I" по умолчанию.

Если бы нам требовалось при умолчании полагать CI =0 , то следовало бы изменить начальное значение этого поля с помощью инструкций VALUE : CI:VALUE 0;

3.I. Управление печатью листинга

Карта *NO LIST , встретившаяся в любом месте текста программы, отключает печать листинга. Карта *FULL LIST , встретившаяся в любом месте текста программы, включает печать листинга. Ошибочные карты распечатываются безусловно.

Листинг программы печатается строками шириной I28 позиций. Строка соответствует одной инструкции микроассемблера. Формат строки:

AAAA HHH....H LLLLLL:KKKK....K

A - шестнадцатиричный относительный адрес или пусто;

H - строка объектного кода в шестнадцатиричном виде. Если в инструкции обнаружена ошибка, то в этом поле появляется текст диагностики.

L - метка в команде или пусто;

K - код операции, параметры и комментарий.

Если инструкция не умещается в I28 позиций, производится перенос в следующую строку листинга, причем поля A - L печатаются пустыми. Перенос производится на границе элементов списка полей микрокоманды.

При ошибках в программе ошибочный фрагмент отмечается символом "*" в следующей строке листинга. В большинстве случаев местонахождение ошибки отмечается с точностью до символа. В крайнем случае - с точностью до лексемы.

Перечислим возможные диагностические сообщения:

"ошибка в заголовке программы " - нет ":" после имени или нет слова PROG .

"неверно задана разрядность" - в конструкции
DEFINE:PROG < число > ; число задано неверно.

"нет глобального контекста" - в архиве отсутствует описание глобального контекста.

"не описаны поля микрокоманды" - в глобальном контексте отсутствует описание полей.

"неверное задание границ поля" - в инструкции FIELD одна из границ поля превышает разрядность команды.

"нет поля с маркировкой /С/" - в глобальном контексте отсутствует поле кода операции секвенсора.

"> 1 поля с маркировкой /С/" - поле кода операции секвенсора описано неоднократно.

"не описан идентификатор" - в инструкции используется несуществующее имя.

"дважды описано имя" - повторное вхождение имени в левой части инструкции.

"синтаксическая ошибка" - неверный формат числа, выражения и т.п.

"переполнение поля" - содержимое поля не вмещается в его границы.

"наложение с полем" - повторное заполнение одного и того же поля или попытка заполнения сразу обоих вложенных полей.

"недопустимый вид выражения" - два внешних адреса или сумма относительных адресов.

"неопознанная инструкция" - как правило, при неуказании кода операции секвенсора.

"ошибка в идентификаторе" - метка не является идентификатором.

" нет такого поля" - имя поля отсутствует в глобальном контексте.

"нет такого КОП" - символическое имя кода операции секвенсора отсутствует в глобальном контексте.

"не адресное поле" - адресная информация попала в неадресное поле.

После текста подпрограммы в листинге печатается таблица использования меток и внешних имен. Для каждого имени выдается список относительных адресов программы, где используется это имя. В начале списка находится относительный адрес метки в программе или символ "E" (для внешних программ).

4. Архив Кросс-системы

Архив всегда заказывается в пакете задачи под логическим номером 30 и в режиме записи. С точки зрения системы архив представляет из себя виртуальную память прямого доступа. Каждое слово этой памяти адресуется своим целочисленным адресом =0,I,2.

Распределение виртуальной памяти:

0 - код наличия глобального контекста;
I - разрядность микрослова N ;
2 - ближайшее к N сверху число, кратное 4;
3 - указатель свободного места в архиве;
20 - таблица имен;
20000 - библиотека модулей загрузки;

Элемент таблицы имен занимает 5 слов. Работа с таблицей производится методом хеширования, причем в качестве ключа поиска используются идентификатор и тип объекта:

Тип=I - локальное имя. По окончании трансляции подпрограммы все локальные имена изымаются из таблицы;
Тип=2 - глобальное имя;
Тип=3 - описание поля. Глобальные имена и описания полей появляются в таблице при трансляции глобального контекста и живут там во все время существования архива;
Тип=4 - описание подпрограммы;
Тип=5 - описание ENTRY -входа.

Таким образом, таблица имен является одновременно и каталогом библиотеки модулей загрузки. При исключении подпрограммы из библиотеки автоматически исключаются и все ее ENTRY -входы. В описании подпрограммы хранится виртуальный адрес начала ее модуля загрузки в библиотеке и размеры модуля загрузки. Модуль состоит из двух частей:

- группа команд;
- список адресных ссылок.

Поскольку размеры адресного поля микрокоманды не фиксированы, то организация модуля загрузки с использованием условных адресов вызывает затруднения. Вместо этого мы использовали в некотором смысле противоположный метод. Список адресных ссылок содержит информацию для загрузчика: в каких командах и каким образом следует модифицировать адресное поле. Список упорядочен по возрастанию относительных адресов команд, что позволяет провести настройку адресов за один просмотр модуля загрузки.

Кроме того, в этом же списке сохраняются и имена локальных меток программы, переписываемые туда из таблицы имен по окончании трансляции. Это сделано впрок, для удобства работы с интерпретатором, если таковой появится в составе нашей кросс-системы.

Свободная память в архиве используется системой как рабочая область. В частности, ассемблер использует ее для хранения текста транслируемой программы (схема трансляции - двухпроходная).

5. Получение двоичной программы

Для получения двоичной программы используется связывающий загрузчик. Пакет задачи в этом случае выглядит следующим образом:

```
*NAME .....
*PASS:.....
*TIME:.....
*LIBRARY:23,25
 <ЗАКАЗ АРХИВА ПОД НОМЕРОМ 30 >
*CALL MICLOA:<ИМЯ ГОЛОВНОЙ ПРОГРАММЫ >
*END FILE
```

Разумеется, вызвать загрузчик можно и сразу же по окончании трансляции текста программы.

Загрузка всегда начинается с нулевого адреса. Двоичный образ загруженной программы помещается в свободную область виртуальной памяти. Оттуда он может быть выведен, например, на перфоленту. В настоящее время загрузчик, закончив свою работу, просто распечатывает получаемый код в шестнадцатиричном виде.

Таблица загрузки печатается в таком же виде, что и таблица загрузки в мониторной системе "Дубна". И точно также, в списке загрузки могут печататься диагностические сообщения:

нет программы < имя программы >
длинный адрес в < адрес команды >
свободно < адрес >