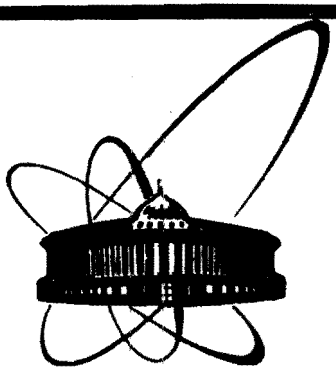

сообщения
объединенного
института
ядерных
исследований
дубна

Р11-86-76

С.М.Фроликов, М.Н.Шумаков

ПРИМЕНЕНИЕ ДИАЛОГОВОЙ СИСТЕМЫ СТРУКТУРИРОВАННОГО ПРОГРАММИРОВАНИЯ ДЛЯ АССЕМБЛИРОВАНИЯ И ОТЛАДКИ МИКРОПРОГРАММ

1986

ВВЕДЕНИЕ

Существует два основных метода создания микропрограмм для специализированных процессоров/1/:

1/ с помощью универсального микроассемблера /см., например,/3//,

2/ с помощью программы, написанной разработчиками процессора специально для данного случая/4-5/.

Достоинства и недостатки обоих методов очевидны: универсальный микроассемблер предоставляет хороший сервис, но сравнительно сложен и громоздок в использовании и может не учитывать какие-либо специфические особенности конкретного процессора. В самодельной программе можно, в принципе, учесть все особенности процессора, но на ее создание нужно тратить силы и время.

В рамках диалоговой системы структурированного программирования /ДССП//2/ оказалось возможным построить достаточно удобный микроассемблер простым и нетрудоемким способом. Кроме того, такой микроассемблер легко совмещается с отладочными программами и программами прожигания микросхем.

Ниже приводится пример построения микроассемблера. Основные процедуры ассемблирования приведены полностью. Процедуры, определяющие мнемоники языка, ввиду их однообразия приведены частично. Приводимые в качестве примеров фрагменты микропрограмм показывают типичные случаи программирования для серии К1804. Устройство конкретного процессора не обсуждается.

Учитывая, что ДССП в настоящее время только начинает распространяться, сделаем некоторые пояснения.

ДССП представляет собой автономную систему программирования для микро-ЭВМ, близкую к распространенному за рубежом языку четвертого поколения FORTH/6/и включающую в себя экранный редактор, интерпретатор, компилятор, отладчик и ретранслятор. Система полностью размещается в ОЗУ и занимает /в машинах типа "Электроника-60"/ около 12К байт. Язык манипуляции данными - обратная польская запись, т.е. выражение: A = B + C записывается в виде:B C + ! A /взять B в стек, взять C в стек, сложить, поместить в A/.

Начало процедуры выделяется двоеточием, конец - точкой с запятой. Вызов процедуры выглядит следующим образом:
A B C PPOC1 /аналог на фортране: CALL PPOC1 (A,B,C)/.

Процедуры из входного потока /т.е. получаемые системой из входного файла или с клавиатуры дисплея/ тут же компилируются, их имена заносятся в словарь и могут использоваться наравне с именами базовых процедур. Правильность работы процедуры сразу

после ее написания может быть проверена с помощью отладчика. Создание программы выглядит здесь не столько как реализация какого-то сложного алгоритма, сколько как расширение списка операций, которые понимает и умеет выполнять машина. Это и делает ДССП удобным инструментом создания прикладных языков.

ДССП имеет следующие важные для нас отличия от FORTH:

1/ В теле процедуры допускается обращение к еще не определенному имени, что позволяет писать программы "сверху вниз" и использовать предлагаемый ниже способ отладки микропрограмм.

2/ Текстовые файлы FORTH имеют формат экрана, что неудобно для оформления текста микропрограммы; ДССП такого ограничения не имеет.

1. ФОРМИРОВАНИЕ ОДНОЙ МИКРОКОМАНДЫ

Микроассемблер должен формировать код микрокоманды /т.е. слово большой длины, обычно до 128 бит/ из кодов отдельных полей. В нашем случае длина микрокоманды - 80 бит, количество полей - 30.

Объявим массив длиной 80 бит:

```
4 VCTR   MC                /микрокоманда/ .          /1/
```

Определим процедуру DEF, такую, что обращение

```
VALUE  F I J  DEF                                    /2/
```

вызовет занесение значения VALUE в поле микрокоманды номер F, занимающее биты от I-го до J-го включительно.

Введем процедуры, заносящие значения в конкретные поля:

```
: CRI    2   4   5   DEF ;  [CARRY INPUT]
: ASRC   5  22  24   DEF ;  [ALU SOURCE]
: AFUN   6  19  21   DEF ;  [ALU FUNCTION]            /3/
: ADST   7  25  27   DEF ;  [ALU DESTINATION]
. . . . . .
```

Поскольку эти процедуры полностью описывают структуру микрокоманды, будем называть их в дальнейшем процедурами определения полей.

Теперь обращение VALUE AFUN вызовет занесение значения VALUE в поле номер 6, занимающее биты с 19 по 21. Всего нами было определено более 20 процедур типа /3/ и 5 аналогичных по назначению процедур для более сложных случаев - "разорванных" полей и полей, смысл и границы которых меняются в зависимости от контекста. Здесь текст этих процедур не приводится ввиду их узкой специфичности. Отметим, что в предлагаемом подходе можно описать более сложную структуру микрокоманды, чем в случае универсального микроассемблера, поскольку описание делается не в виде набора управляющих параметров к некоторой фиксированной программе, а в виде процедур алгоритмического языка.

Приведем пример микрокоманды, написанной с помощью процедур определения полей /3/:

```
2 CP    Ø CR    1 A   Ø B       1 ASRC      Ø AFUN   3 ADST
Ø SHIF  Ø SIN  17 SF  Ø RS      Ø WS
1 SRC   7 DST                                          /4/
Ø EIO   1 EM    1 RW  1 UD      1 EC        1 RAPC
3 CNE   2 S     1 RE 37 COND    Ø DATA
```

Очевидно, что работать с формой записи /4/ неудобно, для ее улучшения введем процедуры более высокого уровня, имеющие более запоминающиеся имена /процедуры-мнемоники/. Воспользуемся, в частности, тем, что имя в ДССП может состоять и из таких символов, как "=" /равно/ или "," /запятая/. Для операций центрального процессорного элемента К1804ВС1 введем процедуры-мнемоники, обращающиеся к процедурам определения полей /3/:

[Источник операнда и функция АЛУ]

```
: F=A+Q        Ø AFUN     Ø ASRC     Ø CRI;
: F=A+B+1      Ø AFUN     1 ASRC     1 CRI;          /5а/
: F=A-B        1 AFUN     1 ASRC     1 CRI;
. . . . . .
```

[Приемник результата операции АЛУ]

```
: Q=F          Ø ADST;  [результат - в рег. Q]
[. . . . . .   1 ADST]  [NOP - устанавливается по умолчанию]
: B=F, Y=A     2 ADST;  [результат - в рег.B, наружу - рег.A]
: B=F          3 ADST;  [результат - в рег.B]
: B=F/2,Q=Q/2  4 ADST;  [в рег.B - результат, B и Q сдви-   /5б/
                         нуть вправо]
: B=F/2        5 ADST;  [в рег.B - результат, сдвинутый вправо]
: B=2F, Q=2Q   6 ADST;  [в рег.B - результат, B и Q сдвинуть
                         влево]
: B=2F         7 ADST;  [в рег.B - результат, сдвинутый  влево]
```

В примере /5а/ приведены 3 из 68 использовавшихся мнемоник для функций АЛУ, в примере /5б/ - все 8 мнемоник для приемника АЛУ. Приведем также пример двух типичных мнемоник, определяющих операции с программным счетчиком /PC/ и регистром адреса /RA/:

```
: RA=@PC+     MEMORY   READ  PC+   IBUS=RAM  RA=IBUS;
: RA=@RA-     MEMORY   READ  RA-   IBUS=RAM  RA=IBUS;    /5в/
```

Как видим, за счет введения процедур-мнемоник можно сделать достаточно удобочитаемую форму записи микрокоманды. Расшифровать смысл записи /5в/ достаточно легко: в RA засылается содержимое ячейки памяти, на которую указывает PC, PC инкрементировать. Процедура-мнемоника RA=@PC+ вызывает процедуры-мнемоники более низкого уровня:

```
MEMORY      -  разрешить операцию с памятью;
READ        - тип операции: чтение;
PC+         - адрес памяти взять из программного счетчика
              PC, счетчик инкрементировать;                    /6/
IBUS=RAM    - данные из памяти заслать на внутреннюю шину;
RA=IBUS     - данные с внутренней шины заслать в регистр
              адреса.
```

Процедуры /6/, в свою очередь, вызывают процедуры еще более низкого уровня. Пройдем "сверху вниз" одну из ветвей /5в/. В ДССП это можно сделать с помощью ретранслятора, который в нашем случае можно рассматривать как справочник.

Например, на команду \L RA=@PC+ система ответит первой из строчек /5в/. Далее, на команду \L READ система ответит:

```
: READ   000001     RW ;
```

а на команду \L RW система ответит

```
: RW     000017    000045    000045    DEF ;
```

В микрокоманде, как правило, активны не все поля, поэтому удобно ввести мнемонику, определяющую значения полей, устанавливаемые по умолчанию:

```
: CONTEXT     [список умолчаний]
-1 !!! MC  [занести -1 во все элементы массива MC]
Ø CP       Ø CRI        DBGOFF      F=Ø
Ø SHIF     Ø SIN        17 A        17 B                    /7/
S=MPC      NOIF         SF=NOP      Ø RS1    Ø WS1   1 SRC
1 EIO1     1 EM1        READ        1 EC1    7 DST
PC         CNE=NOP      1 FEL1      CREG=NOP
```

Теперь ту же самую микрокоманду /4/ удается записать в гораздо более компактной форме:

```
CONTEXT     RA=@PC+    1 A   Ø B    F=A+B    B=F .        /8/
```

Действия, выполняемые микрокомандой /8/, легко понять: установить значения полей по умолчанию, занести в регистр адреса содержимое ячейки памяти, на которую указывает программный счетчик, сложить регистр № A /A=1/ с регистром № B /B=0/, результат поместить в регистр B.

2. ФОРМИРОВАНИЕ ВЫХОДНОГО ФАЙЛА

Объем памяти микропрограмм в большинстве случаев невелик, что позволяет держать выходной файл микроассемблера в оперативной памяти ЭВМ "Электроника-60" в виде массива:

```
10240    VCTR   BMC   [буфер микрокоманд].                  /9/
```

Объявим переменную: VAR AM /адрес микрокоманды/ и введем процедуру END, заносящую микрокоманду MC в буфер BMC, на место,

указываемое адресом AM, и увеличивающую значение AM:

```
VAR  ABUFF
: END   AM 5 * ! ABUFF   Ø    5    DO    END1   !1+ AM ;
: END1  ABUFF  C  MC   ABUFF  !    BMC   1+   !1+  ABUFF ;   /10/
```

Начальный абсолютный адрес микропрограммы можно определять оператором присваивания: ADDRES ! AM.

Символьные метки описываются как переменные:

```
VAR  LABEL1      VAR LABEL2
```

их значения определяются в теле микропрограммы оператором присваивания AM ! LABEL1. Для того, чтобы четче выделять в тексте начало и конец микрокоманды введем переобозначения

```
: <<  CONTEXT ;
: >>  END ;
```

теперь микропрограмму можно записать в виде последовательности слов /операторов/ ДССП, в качестве примера приведем микропрограмму сложения двух целых чисел:

```
VAR  STORE     VAR  ADI$IM     /описание меток/
640  !  AM     /начальный адрес/                          /11/
AM ! STORE
<< @RA=RO    JMPCRG>>    [занести в ОЗУ по адресу, указанному
                          в RA, содержимое выходного регистра RO,
                          выполнить следующую команду]
1250  !  AM    [начальный адрес]
AM ! ADI$IM         [сложение параметр - память, аналог
                     ADD  #I, @#M]
<< RI=@PC+ >>       [рег. ввода = 1 операнд]
<< RA=@PC+ Q=RI >>  [RA=2 операнд, рег. Q = 1 операнд]
<< RI=@RA >>        [рег. ввода = 2 операнд]
<<F=D+Q   RO=Y  FETCH
JMP STORE AD >>     [сложить операнды, результат - в RO /рег.
                    вывода/, выбрать следующую команду, передать
                    управление на адрес STORE].
```

Запись /11/ имеет двоякий смысл: с одной стороны /11/ - программа на построенном нами языке микроассемблер, с другой - последовательность процедур ДССП. Поэтому поиск синтаксических ошибок выполняет ДССП.

Для трансляции микропрограммы необходимо дважды задать директиву ДССП "PF" /PERFORM - выполнить/. Двукратное выполнение процедур ДССП /11/ соответствует двум проходам обычного ассемблера: при первом определяются значения меток, при втором создается выходной файл, содержащий коды микрокоманд с правильными значениями поля адреса. Пример задания поля адреса имеется в последней микрокоманде /11/: при выполнении слова STORE в стек заносится значение метки, при выполнении слова AD это

значение заносится в поле адреса, поэтому имя метки должно стоять перед словом AD.

3. ВСПОМОГАТЕЛЬНЫЕ ПРОЦЕДУРЫ

При работе с ассемблером использовались следующие вспомогательные процедуры, которые вызывались обращениями:

SAVEBMC - выходной файл /9/ записать на диск,

LOADBMC - загрузить с диска выходной файл,

N A1 A2 FIRE - прожечь микросхему ППЗУ номер N, адреса с A1 по A2,

N A1 A2 COMPARE - сравнить содержимое микросхемы с содержимым выходного файла,

A1 A1 PRF - распечатка выходного файла с адреса A1 по A2.

Микроассемблер, вспомогательные программы и выходной файл объемом 2К 80-битных слов вместе с ДССП занимают около 27К слов и размещаются в оперативной памяти. Объем исходного текста микроассемблера вместе с комментариями в нашем случае составил 13К байт. Время трансляции микропрограммы объемом 1000 микрокоманд - около 20 мин.

4. СПОСОБ ОТЛАДКИ МИКРОПРОГРАММ

Микропрограмме, записанной в виде /11/ можно придать еще один - третий - смысл за счет изменения контекста, в котором эта микропрограмма понимается интерпретатором ДССП. Для этого нужно вместо процедур определения полей /3/ ввести процедуры, имеющие те же названия, но выполняющие действия не над полями микрокоманды, а над переменными, описывающими состояние модели процессора. В тексте микропрограммы нужно заменить операторы, задающие значения меток: AM ! LABEL1, на имена процедур: : LABEL1 и в конце каждой процедуры поставить символ ";" /это делается контекстным макросом в символьном редакторе и не занимает много времени/.

Таким образом, мы получаем возможность отладки микропрограммы непосредственно в обозначениях исходного языка и диалоговом режиме с использованием всех возможностей ДССП - запуска с любой точки микропрограммы, распечатки или задания текущих значений переменных, создания точек контрольного останова и т.д.

Объем памяти, остающейся после компиляции микроассемблера /без буфера микропрограмм/ и алгоритмической модели процессора достаточен для отладки микропрограммы длиной несколько сот операторов /больше 10К байт исходного текста/.

Заметим, что состояние модели после выполнения микрокоманды может зависеть от того, в каком порядке выполняются действия, задаваемые процедурами /3/, поэтому изменение переменных нужно производить по последнему слову микрокоманды в правильном порядке.

При отладке микропрограммы нужно учитывать, что команды типа JMP LABEL /переход на адрес LABEL/ на самом деле будут выполняться как CALL LABEL /вызов подпрограммы/, поэтому нужно или следить за тем, чтобы не переполнился стек адресов возврата, или при моделировании перехода вмешиваться в работу системных процедур ДССП, меняя значение указателя стека возврата.

ЗАКЛЮЧЕНИЕ

Предложенным способом нами были построены микроассемблеры для двух различных процессоров. Создание первой работающей версии микроассемблера и выработка системы обозначений языка требовали примерно одинаковых усилий и вместе занимали около одной недели рабочего времени. Отладка процедур ассемблирования трудностей не вызывала ввиду их линейности. Так, в микроассемблере, фрагменты которого приводятся выше, используется всего 6 операторов ветвления, причем 4 из них - для описания ситуаций, специфических для данного процессора. Таким образом, можно сделать вывод, что построение микроассемблера данным способом сравнимо по трудоемкости с созданием файлов описаний к универсальному микроассемблеру, но предоставляет разработчикам несравненно большую свободу действий, поскольку позволяет:

- описывать более сложные структуры микрокоманд, т.к. описанием микрокоманды служит набор процедур алгоритмического языка, а не набор управляющих параметров для фиксированной программы;
- в ходе работы развивать язык за счет введения новых процедур-мнемоник;
- в ходе работы вводить процедуры, проверяющие или предупреждающие те ошибочные ситуации, которые часто встречаются на практике;
- совместить микроассемблер с процедурами прожигания микросхем, отладочными кросс-средствами и средствами отладки аппаратуры.

Отметим, что предложенный метод может применяться и для построения кросс-ассемблеров.

Авторы выражают благодарность Н.П.Брусенцову и Ю.А.Семенову - за полезные консультации, С.А.Сидорову, И.А.Рудневу и В.Б.Захарову - за помощь в освоении ДССП, Т.С.Григалашвили и Л.Л.Неменову - за поддержку данной работы.

ЛИТЕРАТУРА

1. Kunz P.F. Proc. of the 1978 CERN School of Computing. Jadwisin, Poland, 1978.
2. Брусенцов Н.Н. Микрокомпьютеры. "Наука", М., 1985.

3. Семенов Ю.А. Препринт ИТЭФ, 154-1984, М., 1982.
4. Григалашвили Т.С., Фроликов С.М., Шумаков М.Н. ОИЯИ, 10-84-664, Дубна, 1984.
5. Семенов Ю.А., Чудаков В.Н. Препринт ИТЭФ, 153-1982, М., 1982.
6. Семенов Ю.А. Препринты ИТЭФ, 30-1984, 63-1984, 72-1984, М., 1984.

Рукопись поступила в издательский отдел
7 февраля 1986 года.

Фроликов С.М., Шумаков М.Н. Р11-86-76

Применение диалоговой системы структурированного программирования для ассемблирования и отладки микропрограмм

Описывается простой способ и приводится пример построения микроассемблера в программной среде диалоговой системы структурированного программирования (ДССП). Приводится текст основных процедур ассемблирования. Предлагается способ отладки микропрограмм.

Работа выполнена в Серпуховском научно-экспериментальном отделе ОИЯИ.

Сообщение Объединенного института ядерных исследований. Дубна 1986

Перевод О.С.Виноградовой

Frolikov S.M., Shumakov M.N. P11-86-76

Application of Dialogue System of Structured Programming for Assemblying and Microprogram Debugging

A simple way is described and an example of construction of microassembler in a program medium of Forth-like DSSP system is presented. The text of basic procedures of assemblying is given. A way of microprogram debugging is proposed.

The investigation has been performed at the Serpukhov Scientific-Experimental Department, JINR.

Communication of the Joint Institute for Nuclear Research. Dubna 1986