Существует режим пошагового исполнения программы. Команда STEP позволяет исполнить одно предложение программы и вернуться в состояние приостановки.

При переходе в режим приостановки на терминале печатается номер строки, перед (в) которой произошел останов. Далее система находится в калькуляторном режиме, причем доступны все программные объекты. Часто для контроля за ходом работы программы требуется производить просмотр одних и тех же объектов или выполнение однотипных действий. Существует команда

DISPLAY *строка*

которая позволяет запомнить строку, являющуюся аргументом команды, и исполнять ее как самостоятельную команду при каждом переходе в режим приостановки. Например, после команды

DISPLAY TYPE 'A, B, C =', A, B, C

при каждой приостановке будут распечатываться значения переменных A, B и C. Команда

DISPLAY CALL 1000

позволяет автоматически выполнять на каждой приостановке сколь угодно сложную подпрограмму.

В системе BUSIC имеются средства трассировки программы, управления режимами компиляции и исполнения, а также другие возможности технического характера. Все команды и операторы, относящиеся к режиму отладки, допускают сокращение до одной буквы, что удобно для оперативной диалоговой работы.

Заключение. Представленная работа демонстрирует пример внедрения языковой программной системы в сферу деятельности инженеров-разработчиков аппаратуры. Здесь не описаны существующие в системе специализированные средства, имеющие отношение к конкретным особенностям задачи [3]. Мы попытались дать неформальное представление о наиболее общих решениях, принятых при разработке.

Авторы считают необходимым отметить вклад в работу С. Л. Ивашина, реализовавшего ряд внутренних механизмов системы и все компоненты драйверного уровня. Авторы признательны также большому коллективу инженеров, которые фактически проводили тестирование системы, принимая ее непосредственно в рабочую эксплуатацию.

ЛИТЕРАТУРА

1. **Бредихин С. В., Песляк П. М.** Простой интерактивный язык для КАМАК.— Новосибирск, 1975. (Препринт/АН СССР, Сиб. отд-ние, ИАиЭ; № 28).
2. **Бредихин С. В., Песляк П. М.** CATY-M: система для программирования аппаратуры КАМАК. (Модифицированный вариант).— Новосибирск, 1981. (Препринт/АН СССР, Сиб. отд-ние, ИАиЭ; № 166).
3. **Ковалев А. М., Талныкин Э. А.** Машинный синтез визуальной обстановки.— Автометрия, 1984, № 4.

Поступила в редакцию 20 апреля 1984 г.

УДК 681.3.06

И. М. КАГАНСКИЙ

(Новосибирск)

СТРУКТУРИРОВАНИЕ ПРОГРАММ СРЕДСТВАМИ МАКРОАССЕМБЛЕРА В ОС ЕС

Введение. Язык программирования Ассемблер предоставляет программисту полную свободу в отношении стиля. Нередко ассемблер-программы написаны так, что к ним неприменимо понятие стиля. Была

поставлена задача разработать языковые инструментальные средства, имеющие сравнимые с Ассемблером показатели по эффективности, но обеспечивающие более высокую производительность труда программистов. Нужно было определить некий набор соглашений и придать им статус правил программирования. В соответствии с прагматическими целями: ясность стиля написания программ, самодокументируемость (насколько это возможно при использовании языка низкого уровня), упрощение процедур локализации и устранения ошибок в программах — осуществлена попытка реализовать концепции структурного программирования [1—6] средствами Ассемблера.

Описываемый пакет, состоящий из нескольких десятков макроопределений, основан на использовании Ассемблера/E. Тем самым достигнута переносимость соответствующего программного обеспечения из среды ДОС АСВТ в ОС ЕС.

Ассемблер/E со встроенным макропроцессором, несмотря на некоторую ограниченность средств (по сравнению с Ассемблером/F для ОС ЕС), как и всякий макроассемблер, позволяет определить и реализовать новые понятия, отсутствующие в базовом языке, в форме текстовых макросов. Фактически эти понятия не являются синтаксическими объектами языка, так как макровычисление происходит до компиляции и к лексическому анализатору на вход поступает макрорасширение; однако эти понятия, записанные как макрокоманды, расширяют базовый язык и служат разным целям. Это могут быть макросы, порождающие вызов подпрограмм, реализующие арифметические операции или описывающие наборы данных и их атрибуты. Наличие библиотеки макроопределений позволяет программисту не определять макросы всякий раз заново, а пользоваться готовыми, хранящимися в макробиблиотеке.

Для получения программного инструмента, не уступающего по выразительности проблемному языку, необходимо описать в виде макроопределений операции для работы с данными (выражения присваивания), а также операции доступа к наборам данных в памяти и на внешних носителях (массивы, строки, деревья, списки и т. д.). Эти аспекты применения макросов существенно зависят от приложений и здесь не рассматриваются.

В соответствии с концепциями структурного программирования и методикой, описанной в [4], определим программный сегмент как объект структурированной программы. Рассмотрим два типа сегментов — процедура и модуль. С помощью процедур осуществляется декомпозиция задачи, а средства модуляризации уменьшают ее общую сложность, облегчают процесс трансляции [2, 7].

Макросы управляющих структур (циклы, условные операторы, операторы выбора) есть мощное средство расширения базового языка для управления вычислительным процессом. Программы, написанные с использованием этих макрокоманд, приобретают выразительность и компактность и могут быть составлены с применением рекомендаций структурного программирования.

Сегментация программ и макрокоманды программных структур. Процесс разделения ассемблер-программ на части называется секционированием. Для осуществления секционирования в Ассемблере имеются псевдоинструкции START, CSECT, COM и DSECT; для указания о том, что некоторое символическое имя может быть доступно в другой программной секции, служит утверждение ENTRY, а предложение EXTRN указывает имена, описанные в других программных секциях и используемые в текущей.

Аппарат независимой трансляции программных секций с последующим редактированием связей и объединением независимо оттранслированных модулей в рабочую программу служит основой модуляризации программ. Наличие команд переходов и переходов с возвратом, наряду со средствами секционирования в Ассемблере, предоставляет программисту необходимый инструмент для организации межпрограммного взаи-

модействия. Какой механизм этого взаимодействия будет выбран, зависит от многих факторов, в частности от вкусов и квалификации автора программы.

При конструировании макросов, обеспечивающих процедурный механизм, мы исходили из следующей посылки: должны существовать, по меньшей мере, два типа процедур — процедуры местного определения, доступные внутри программной секции, и процедуры глобальные, доступ к которым разрешен из других модулей. В первом и втором случаях по-разному решается вопрос сохранения контекста в точке вызова для обеспечения правильного возврата в точку вызова.

Если разрабатываемая программа невелика, то не возникает организационных вопросов распараллеливания труда разработчиков и выработки жестких соглашений о связях между модулями. В этом случае можно ограничиться использованием процедур местного определения. Для правильного выхода из процедуры необходимо сохранение/восстановление регистра возврата: в этом случае сам программист заботится о сохранности контекста в точке вызова и организует механизм передачи параметров.

При использовании средств модуляризации уже недостаточно ограничиться только мерами по сохранению регистра возврата: следует учитывать тот факт, что программные модули могут создаваться разными людьми и в разное время. Поэтому необходимо разработать и специфицировать интерфейс и тщательно контролировать связи между отдельными сегментами.

Определим процедуру как выполняемую единицу программы, текстуально ограниченную структурными скобками BEGIN-END и начинающуюся с заголовка — пролога к процедуре.

Рассмотрим четыре типа процедур, которые организуют сегментацию структурированных программ на Ассемблере.

1. MAIN — определить главную программу. Макрокоманда MAIN обеспечивает интерфейс с операционной средой. Форма макрокоманды:

```
MAIN NAME,
     BASE = регистр,
     STACK = число,
     SAVE = метка
```

NAME — имя программной секции в предложении CSECT, которое будет сгенерировано по команде; при опускании параметра генерируется неименованная секция. BASE — базовый регистр; при опускании параметра принимается по умолчанию R15. STACK — запрос на выделение памяти в словах под стек для сохранения регистров; указатель стека — R7; при опускании параметра память под стек не отводится. SAVE — имя области сохранения; если параметр опущен, то цепочка областей сохранения не строится.

2. PROC — определить локальную процедуру. Форма макрокоманды:

```
PROC NAME
```

NAME — имя процедуры: идентификатор, допустимый лексикой Ассемблера. Процедура не имеет собственного базового регистра. В теле процедуры обеспечивается сохранение регистра R14 для правильного возврата в точку вызова. Выход из процедуры PROC осуществляется по макрокоманде RTN NAME, которая восстанавливает регистр связи, а передача управления — по макрокоманде PCALL NAME.

```
MAIN EXAMPLE
BEGIN
  .
  .
  .
  PCALL P1
```

```
        .
        .
        .
    ENDB
        .
        .
        .
    PROC P1
    BEGIN
        .
        .
        .
        RTN P1
        .
        .
        .
    ENDB
```

3. SUB — определить глобальную процедуру. Форма макрокоманды:

SUB NAME

NAME — имя процедуры. Процедура базируется регистром R15. Выход из процедуры выполняется по макрокоманде RTS NAME, а передача управления процедуре — по макрокоманде JMS NAME. Эти макрокоманды обеспечивают сохранение/восстановление регистров на стеке и правильный возврат в точку вызова.

4. SEG NAME — определить сегмент-процедуру. Форма макрокоманды, параметры, способ сохранения регистров, вызов и возраст из процедуры аналогичны используемым для макрокоманды SUB. Отличие состоит в том, что по макрокоманде SEG строится программная секция.

В качестве иллюстрации техники сегментирования структурированной ассемблер-программы приведем пример описания и вызова процедур.

```
MAIN EXAMPLE, STACK = 5Ø
  BEGIN
      .
      .
      .
      JMS #P3
      .
      .
      .
      PCALL P1   вызов процедуры местного определения
      .
      .
      .
      JMS #P2    вызов процедуры
      PROC P1  ┐
        BEGIN  │
          .    │
          .    }процедура местного определения
          .    │
        RTN P1 │
        ENDB   ┘
      .
      .
      .
  ENDB
END
```

```
SEG P2                    )
   BEGIN                  |
      .                   |
      .                   |
      .                   |
      JMS P3              |  сегмент-процедура
      .                   |
      .                   |
      .                   |
      RTS P2              |
   ENDB                   )

SUB P3                    )
   BEGIN                  |
      .                   |
      .                   |  глобал-процедура
      .                   |
      RTS P3              |
   ENDB                   |
END                       )
```

Управление логикой программы и макрокоманды управляющих структур. Методика структурного программирования (в терминах [5]) может быть описана как последовательная декомпозиция задачи, приводящая к построению результирующей программы как многошагового процесса, каждый шаг которого реализуется с помощью одного из базисных операторов. При этом на каждом шаге используются только структуры типа конкатенация, альтернатива или итерация.

Под конкатенацией понимается представление решения некоторой задачи в виде последовательного решения двух подзадач; под альтернативой — проверка некоторого условия и выполнение в зависимости от его истинности одной или другой задачи; под итерацией — повторение выполнения задачи до наступления некоторого события, называемого условием прекращения цикла.

Нетрудно видеть, что для конкатенации двух последовательных управляющих структур специальные языковые средства не нужны, тогда как для выбора альтернативного пути в программе и организации циклов следует определить управляющие конструкции типа IF-THEN-ELSE и WHILE-DO.

Структура типа альтернатива имеет следующий вид:

IF C THEN S_1 [ELSE S_2] FI

или в соответствии с требованием Ассемблера, где строка есть команда,—

```
IF                                  IF
    if-mecm                             if-mecm
C   THEN                        C     THEN
    then-оператор S1            или         then-оператор S1
ELSE                                FI
    else-оператор S2
FI
```

Составное условие, образуемое комбинацией простых условий с помощью логических операций AND и OR, есть утверждение, истинность которого проверяется. Фактически тестируется код условия CC (Condition Code — двухбайтовое поле в слове состояния программы, в котором хранится признак результата последней выполненной команды) и генерируется команда условного перехода.

Простые условия указываются следующим образом.

Для логических операций:

EQ — выполняется *then-оператор*, если CC = 0,
NE — выполняется *then-оператор*, если CC = 1,
LT — выполняется *then-оператор*, если CC = 2,
GT — выполняется *then-оператор*, если CC = 3,
LTEQ — выполняется *then-оператор*, если CC ≠ 1,
GTEQ — выполняется *then-оператор*, если CC ≠ 2.

Для арифметических операций:

NIL — выполняется *then-оператор*, если CC = 0,
MIN — выполняется *then-оператор*, если CC = 1,
PL — выполняется *then-оператор*, если CC = 2,
OVR — выполняется *then-оператор*, если CC = 3,
CRY — выполняется *then-оператор*, если CC = 3,
NENIL — выполняется *then-оператор*, если CC ≠ 0,
NEMIN — выполняется *then-оператор*, если CC ≠ 1,
NEPL — выполняется *then-оператор*, если CC ≠ 2,
NEOVR — выполняется *then-оператор*, если CC ≠ 3,
NECRY — выполняется *then-оператор*, если CC ≠ 3.

Для команд проверки по маске:

ZERO — выполняется *then-оператор*, если CC = 0,
MIX — выполняется *then-оператор*, если CC = 1,
ONE — выполняется *then-оператор*, если CC = 3,
NEZERO — выполняется *then-оператор*, если CC ≠ 0,
NEMIX — выполняется *then-оператор*, если CC ≠ 1,
NEONE — выполняется *then-оператор*, если CC ≠ 3.

В составных условиях во всех приведенных выше макрокомандах вместо параметра THEN указываются параметры OR или AND:

```
IF
   CLC FIELD1, FIELD2
EQ AND
   AR R1, R2
PL  THEN
     then-оператор
FI
```

Структура типа итерация имеет следующий вид:

```
WHILE C DO S ENDWHILE
```

или на Ассемблере:

```
WHILE
        while-тест
C DO
        do-часть
ELIHW
```

Здесь и ниже C — составное условие, которое конструируется так же, как и в структуре IF-THEN-ELSE.

Кроме структуры WHILE-DO, для организации циклов введены дополнительные структуры.

Цикл с постусловием

```
DO S UNTIL C ENDDO
```

на Ассемблере записывается так:

```
DO
   do-часть
UNTIL
   until-тест
OD
```

do-часть повторяется до тех пор, пока не становится ложным утверждение C.

Итеративный цикл:

```
FOR I FROM A [STEPK] TO B DO S ENDFOR
```

или на Ассемблере:

```
FOR R_i STEP ⟨число⟩ TO R_j DO
     do-часть
FOR R_i, ⟨число⟩, TO, R_j
```

Третий параметр может изображаться как TO или DOWNTO, что означает отрицательный шаг цикла. Два регистра R_i и R_j задают границы переменной цикла. Когда R_i достигает значения, равного R_j, цикл прекращается.

Цикл по счетчику:

```
TO N DO S ENDTO
TO R DO
    do-часть
OT R
```

Здесь требуемое число повторений цикла должно быть предварительно помещено в регистр R. При каждом выполнении тела цикла значение регистра R уменьшается на единицу, и при R, равном нулю, цикл завершается.

Структурные переходы:

```
LEAVE L              CYCLE L
```

LEAVE предписывает принудительно прекратить выполнение некоторой структуры, а CYCLE — начать исполнение заново, например:

```
LEAVE IF, BLOCK
CYCLE BLOCK
```

Удобным средством для организации ветвлений в программах и конструирования переключателей являются операторы выбора. Нами определены макросы для последовательного выбора:

```
SELECT R_i  OF
        A_1 : A_2 : S_1
        :
        :
        A_n       S_n
        ELSE
                  S_0
ENDSELECT
```

и параллельного:

```
CASE R_i OF S_1, ..., S_k [ELSE S_0] ENDCASE
```

в форме, которую представим на примере

```
SELECT R OF
 SET
  SEL 1
       sel_1-часть
  SEL 'B'
       sel_2-часть
       :
  SEL 'C', 10
       sel_k-часть
 TES
       else-часть
TCELES
```

С содержимым регистра R последовательно сравниваются число 1, значение внутреннего представления символа B, значение внутреннего пред-

ставления символа C и число 10. При первом же совпадении поиск заканчивается и выполняется выбранный оператор:

```
CASE R OF
 SET
  CAS   метка 1
        cas1-часть
  CAS   метка 2
        cas2-часть
   ⋮
  TES   метка 1, метка 2, ..., метка n
        else-часть
ESAC
```

Содержимое регистра R рассматривается как порядковый номер метки, определяющий выбор *cas-оператора*, на который передается управление.

Форма представления программ. При использовании описанной выше методики структурированная программа составляется из сегментов, представляющих собой программные единицы, организуемые в виде процедур одного из четырех типов. Внутри процедур логика управляется структурами типа IF-THEN-ELSE, WHILE-DO, DO-UNTIL или SELECT-CASE. Вся программа и каждый ее сегмент должны читаться последовательно сверху вниз.

Следует отметить, что расположение текста программы играет в структурном программировании немаловажную роль (см., например, [8], где развит формализм для описания правил наглядного размещения). В частности, следует делить текст программы на абзацы; выделять структурные операторы абзацными отступами для того, чтобы различать управляющие структуры по подчиненности.

Реализация. Описанный пакет реализован в ДОС АСВТ и в ОС ЕС [11]. Эти работы проводились с 1976 г. и носили поначалу экспериментальный характер. Впоследствии на структурном Ассемблере были написаны пакет программ для обеспечения коммуникаций в задачах АНИ [12], программы сбора экспериментальных данных [13], инструментальный комплекс программ и компилятор с языка системного программирования на М-4030 [14, 15]. Переписанный на макроязык Ассемблер/F для ОС ЕС пакет и в настоящее время является удобным инструментом для написания структурированных ассемблер-программ. Надо отметить, что эти аспекты развивали и продолжают развивать разные авторы [6, 16, 17]. Широко распространен пакет макросредств для структурного программирования, разработанный фирмой IBM. Анализ результатов указанных работ и опыт многолетней эксплуатации пакета подтвердили эффективность реализованного подхода к структурированию ассемблер-программ.

ЛИТЕРАТУРА

1. **Дал У., Дейкстра Э., Хоор К.** Структурное программирование.— М.: Мир, 1975.
2. **Миллс Х.** Программирование больших систем по принципу сверху вниз.— В кн.: Средства отладки больших систем. М.: Статистика, 1977.
3. **Вирт Н.** Систематическое программирование. Введение.— М.: Мир, 1977.
4. **Лингер Р., Миллс Х., Уитт Б.** Теория и практика структурного программирования.— М.: Мир, 1982.
5. **Ершов А. П.** Введение в теоретическое программирование. Беседы о методе.— М.: Наука, 1977.
6. **Вельбицкий И. В., Ходаковский В. Н., Шолмов Л. И.** Технологический комплекс производства программ на машинах ЕС ЭВМ и БЭСМ-6.— М.: Статистика, 1980.
7. **Брукс Ф. П.** Как проектируются и создаются программные комплексы.— М.: Наука, 1979.
8. **Mills H. D.** Syntax-directed documentation for PL 360.— Comm. ACM, 1970, vol. 13, p. 216—222.
9. **Кэмпбел-Келли М.** Введение в макросы.— М.: Сов. радио, 1978.
10. **Браун П.** Обзор макропроцессоров.— М.: Статистика, 1975.
11. **Каганский И. М., Талыкин Э. А.** Реализация управляющих конструкций средствами макроассемблера ЕС ЭВМ.— В кн.: Автоматизация научных исследований

на основе применения ЭВМ: Материалы Всесоюз. конф. Новосибирск: ИАиЭ СО АН СССР, 1977.
12. **Бредихин С. В. и др.** Программирование средств связи и управление вводом-выводом в унифицированной магистральной системе обмена (УМСО).— В кн.: Автоматизация эксперимента. Новосибирск: ИАиЭ СО АН СССР, 1976.
13. **Бредихин С. В., Каганский И. М., Песляк П. М.** Аппаратные и программные средства сбора данных в эксперименте ДОППЛЕР.— В кн.: Автоматизация экспериментальных исследований: Тез. Всесоюз. конф. Куйбышев: КУАИ, 1978.
14. **Каганский И. М.** RIE — система диалогового взаимодействия.— Новосибирск, 1983. (Препринт/АН СССР, Сиб. отд-ние, ИАиЭ; № 201).
15. **Каганский И. М., Талныкин Э. А., Фризен Д. Г.** Язык системного программирования, ориентированный на архитектуру ЕС ЭВМ.— В кн.: Автоматизация научных исследований на основе применения ЭВМ: Материалы Всесоюз. конф. Новосибирск: ИАиЭ СО АН СССР, 1977.
16. **Фурман М. Е.** Макрогенерация структурирования программ в ОС ЕС.— Программирование, 1982, № 4.
17. **Алгол-68.** Методы реализации/Под ред. Г. С. Цейтлина.— Л.: ЛГУ, 1976.

Поступила в редакцию 26 декабря 1983 г.

УДК 681.3.06

Э. А. ТАЛНЫКИН
(Новосибирск)

PED — ГРАФИЧЕСКИЙ РЕДАКТОР В СИСТЕМЕ ПРОЕКТИРОВАНИЯ ПЕЧАТНЫХ ПЛАТ

Введение. В работе [1] приведены основные принципы построения программного комплекса для проектирования печатных плат, развиваемого в Институте автоматики и электрометрии СО АН СССР на базе ЭВМ семейства «Электроника» или СМ-4. В настоящей статье рассмотрена основная компонента комплекса — интерактивная система графического редактирования топологии печатных плат (PED). Система работает либо на ЭВМ, использующей графические возможности через линию связи, либо в микро-ЭВМ на автономной конфигурации рабочего места [2]. Далее будут применяться термины и понятия, введенные в [1, 2]. (Поскольку настоящая статья не является полной инструкцией по работе с системой, некоторые элементы, относящиеся к техническим деталям, здесь не описаны).

Принципы работы редактора. Конструкции печатных плат хранятся в архиве в виде файлов. Для работы с печатными платами в редакторе имеется виртуальное поле, соответствующее размеру 1600×1600 мм на плате при разрешении 0,025 мм. Информация может размещаться на 15 слоях виртуального поля, что позволяет одновременно редактировать до 15 слоев печатной платы. Сразу после вызова системы виртуальное поле пусто и можно начинать создание новой конструкции. При необходимости редактировать или просматривать с целью контроля уже существующую в архиве печатную плату ее нужно вызвать на редакцию («прочитать» в виртуальное поле). Исходная копия печатной платы в процессе редактирования не участвует. По завершении процесса редактирования полученная конструкция может быть записана в файл (обычно в новую версию исходного файла).

Поскольку конструкция печатной платы состоит из печатных элементов и проводников, редактирование естественным образом сводится к удалению старых и образованию новых элементов с использованием средств автоматизации, предоставляемых графическим редактором.

В процессе редактирования часть виртуального поля печатной платы (прямоугольное окно) составляет поле изображения. Фрагмент поля изображения может быть выведен на экран дисплея. Поле изображения