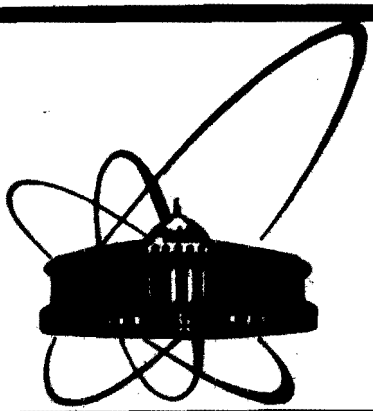

сообщения
объединенного
института
ядерных
исследований
дубна

С-302

Р11-87-892

Б.Ю.Семенов*, В.Е.Жильцов

НЕКОТОРЫЕ ВОПРОСЫ СОВМЕСТНОГО ИСПОЛЬЗОВАНИЯ ПРОГРАММ НА ЯЗЫКАХ ПАСКАЛЬ И АССЕМБЛЕР ДЛЯ РЕШЕНИЯ ЗАДАЧ РЕАЛЬНОГО ВРЕМЕНИ

*Научно-исследовательский институт прикладной физики Ташкентского государственного университета, Ташкент

1987

Стремительное расширение числа компьютеризированных систем реального времени требует повышенного внимания к их программному обеспечению, особенно той его части, которая тесно связана со спецификой конкретной системы. Программы такого рода часто целесообразно разрабатывать с использованием одновременно машинно-ориентированного языка /Ассемблера/ и языка высокого уровня. При корректном исполнении гибридная программа будет обладать качествами, недоступными каждому языку в отдельности: быстродействием, экономным использованием памяти и простотой в общении с оборудованием, как у Ассемблеров; простотой разработки программ, надежностью, гибкостью и удобочитаемостью, как у современных языков высокого уровня.

Паскаль-Рафос (OMSI PASCAL-1)[1] предусматривает ряд средств для выхода на машинно-ориентированный уровень, которые описываются в руководствах. К таким средствам относятся:

- введение вставок на языке Ассемблер {$C };
- получение информации об адресах переменных @;
- размещение переменных по абсолютным адресам памяти ORIGIN;
- использование восьмеричных констант B;
- использование логических операций с целыми числами.

Все эти средства, так же как и способы обращения во вставках на Ассемблере к глобальным и локальным переменным через индексацию регистров R5 и R6, ориентированы на использование во вставках Макро-кода в исходных Паскаль-программах. Вопросы же, касающиеся объединения отдельно откомпилированных программ, в языке Паскаль-Рафос не рассмотрены.

В работе идет речь о разработке программ для систем реального времени с использованием отдельно откомпилированных программ на языках Макроассемблер /Макро/ СМ ЭВМ и Паскаль-Рафос. При этом обсуждается распределение памяти, взаимный вызов Паскаль-программ и программ на Макро, передача параметров для этих случаев. Предлагается механизм образования общих масси-

Объединенный институт ядерных исследований

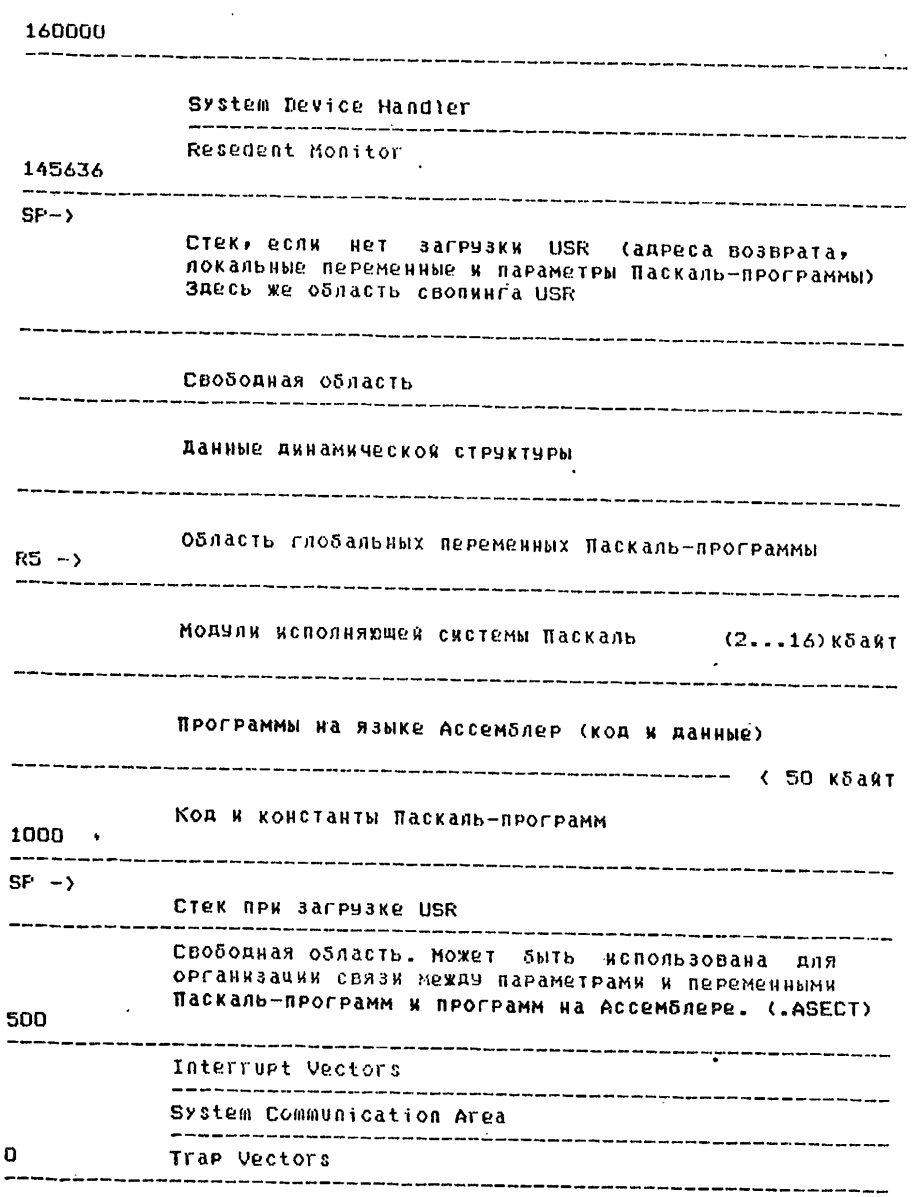

Рис. 1. Память при использовании программы, составленной из отдельно откомпилированных модулей на языках Макроассемблер и Паскаль-Рафос.

вов и переменных. Рассмотрен вопрос использования внешней процедуры Паскаль-программы для обслуживания прерывания.

РАСПРЕДЕЛЕНИЕ ПАМЯТИ

При загрузке программ, образованных линкированием программ Паскаль-Рафос и ассемблерных, кроме кодов констант и данных этих программ в ОЗУ размещаются /рис. 1/ статические компоненты операционной системы и модули исполняющей системы Паскаль (OTS). Для минимизации области, занятой этими модулями, необходимо вводить их при линкировании с помощью библиотечного файла PASLIB.OBJ (LIBRARY/CREATE PASLIB.OBJ PASHND.OBJ), иначе в загрузочную программу войдут все модули OTS с объемом не менее 8К слов. При использовании библиотечного модуля объем OTS в простых программах может быть снижен до 1К слов.

Основную информацию о распределении памяти программ содержит главная Паскаль-программа, поэтому среди линкируемых программ она должна быть первой. Для программы на ассемблере может быть заказана именованная или неименованная относительная секция - в любом случае распределение в памяти будет соответствовать изображенному на рис. 1.

ВЫЗОВ ПАСКАЛЬ-ПРОГРАММ ИЗ ПРОГРАММ НА АССЕМБЛЕРЕ

Внешние процедуры вызываются в Паскаль-программах оператором JSR с использованием программного счетчика в качестве регистра связи и имени процедуры в качестве глобальной метки точки входа в вызываемую программу. Аналогичным образом внешние процедуры Паскаль-программ могут быть вызваны и из программ на Макро: JSR PC, ИМЯ ПРОЦЕДУРЫ, или же в виде: CALL ИМЯ ПРОЦЕДУРЫ. При этом, если в процедуре отсутствуют локальные переменные или формальные параметры, кроме сохранения и восстановления используемых регистров, никаких других действий предпринимать не требуется. Если же таковые имеются, то их значения следует передавать через стек /см. MAIN.MAC/, используя для передачи целого или символьного типа одно слово, а для вещественного - два. Необходимая при этом коррекция стека на выходе, так же как и коррекция на входе и выходе при наличии локальных параметров или переменных, производится операторами процедуры. Для уточнения механизма передачи параметров и работы стека можно воспользоваться листингами программ, подобными приведенным ниже.

SHOW.PAS

```
PROCEDURE show(a,b:CHAR; c:INTEGER; VAR y:INTEGER); external;
VAR d,e: CHAR;
  BEGIN
  END;
```

MAIN.PAS

```
PROGRAM main;
VAR x:INTEGER;
PROCEDURE show(a,b:CHAR; c:INTEGER; VAR y:INTEGER); external;
BEGIN
  show('A','B',3,x)
END.
```

MAIN.MAC

```
        . . .

        .GLOBL  SHOW
LO:     JMP     SHOW
        . . .
        . . .
                                ;Передача параметров и переменной
;  show('A','B',3,x)            ;из главной программы во внешнюю
        MOV     #65,-(6)        ;процедуру. Аналогичным образом
        MOV     #66,-(6)        ;может быть организована передача
        MOV     #3,-(6)         ;параметров и переменных из прог-
        MOV     %5,-(6)         ;рамм на Ассемблере.
;END.
        JSR     %7,LO           ;Вызов процедуры SHOW.
        JMP     $END
        .END
```

SHOW.MAC

```
        . . .
;BEGIN
LO:
        .GLOBL  SHOW
SHOW:
        CLR     -(6)        ->  ;Подготовка стека для размещения
        . . .                   ;локальных переменных [d,e:CHAR]
        . . .
;END;
        MOV     2(6),10(6)  ->  ;Коррекция адреса адреса возврата
        ADD     #10,%6      ->  ;Коррекция указателя стека,
        RTS     %7              ;связанная с наличием формальных
        .END                    ;параметров [a,b,y,c] и локальных
                                ;переменных [d,e]
```

Использование нестандартного вызова процедуры требует нестандартного возвращения. Наличие при этом локальных параметров приводит к необходимости коррекции указателя стека при выходе из процедуры. Пример такой коррекции при использовании внешней процедуры для обслуживания прерывания по таймеру показан ниже.

MAIN.PAS

```
        . . .
        . . .
 ($C    MOV  #service,^0100
  1$:   TST  arg(R5)
        BNE  1$     )
        . . .
        . . .
```

Представленный фрагмент программы обеспечивает: введение адреса программы обслуживания в первое слово вектора прерывания, ожидание таймерного прерывания(операторы TST, BNE) и выход из ожидания по условию arg=0.

SERV.PAS

```
PROCEDURE service;external;
VAR a,b,c: INTEGER;
BEGIN
        . . .
        . . .
  ($C
        ADD  #6,SP
        RTT
  )
END;
```

Необходимую для нормального функционирования программы service коррекцию SP выполняет оператор ADD, так как выход из процедуры возможен только через оператор RTT, минуя END. (См. SERV.MAC)

SERV.MAC

```
; PROCEDURE service; external;
        . . .
        . . .
; VAR a,b,c: INTEGER;
; BEGIN
LO:
        .GLOBL  SERVICE
SERVICE:
        SUB     #6,%6       ->  ;Подготовка места в стеке для ло-
        . . .                   ;кальных переменных
        . . .

; ($C

        ADD     #6,SP       ->  ;Замена операторов группы END при
        RTT                     ;нестандартном выходе из процедуры
; END;
        ADD     #6,%6
        RTS     %7
        .END
```

Следует отметить, что при нестандартном вызове внешней процедуры нельзя использовать стандартный для Паскаль-программ аппарат формальных параметров. В этом случае передачу данных из главной программы во внешнюю процедуру необходимо проводить какими-либо другими способами.

ВЫЗОВ ПРОГРАММ НА АССЕМБЛЕРЕ ИЗ ПАСКАЛЬ-ПРОГРАММ

Вызов программ на Ассемблере из Паскаль-программ легко осуществить путем использования в Паскаль-программе ассемблерной вставки вида {$C CALL MACRO}, при этом возврат из вызываемой программы должен быть организован через оператор RETURN. При вызове из Паскаль-программ программ на Макро необходимо сохранять, а при возвращении восстанавливать содержимое используемых регистров. Передача параметров и переменных из Паскаль-программы в ассемблерную и обратно может быть организована с помощью вставок в Паскаль-программе вида:

{$C MOV VARNAME(R5), GLBLA}

и

{$C MOV GLBLA, VARNAME(R5)},

где VARNAME - имя переменной, объявленной в Паскаль-программе, а GLBLA - глобальная метка в программе на Ассемблере.

ОБРАЗОВАНИЕ ОБЩИХ ПЕРЕМЕННЫХ

При использовании отдельно откомпилированных Паскаль-программ и программ на Ассемблере возникает проблема организации общего доступа к отдельным простым переменным или массивам, решаемая в Фортран-программах путем использования COMMON - блоков в Фортране и именованных программных секций в Ассемблере. Похожий механизм может быть образован и в рассматриваемом случае путем размещения переменных или их адресов по абсолютным адресам с использованием ключевого слова ORIGIN, и адресного оператора "@" в Паскаль-программе и абсолютной секции памяти ASECT - в программе на Макро /рис. 1/. Ниже приведен пример образования доступных для программ MAIN.PAS и ASM.MAC массивов DATA и SAVE.

MAIN.PAS

```
          . . .
          . . .
          . . .
TYPE buf=ARRAY[1..64]OF integer;   Объявлено два нестандартных типа
     adrbuf = ^buf;                BUF - тип одномерного массива из 64
          . . .                    целых чисел и ADRBUF - ссылочный
          . . .                    (адресный) тип для переменных типа
          . . .                    BUF.
VAR  data:                buf;     Объявлено два массива DATA, SAVE,
     adrd origin 500B: adrbuf;     и ссылочные переменные ADRD, и ADRS,
     save:                buf;     размещаемые по абсолютным адресам
     adrs origin 502B: adrbuf;     памяти 500B и 502B.
          . . .
          . . .
          . . .
```



```
BEGIN
  adrd:=@data; adrs:=@save;        Информация об адресах массивов DATA
          . . .                    и SAVE заносится в ячейки 500B
          . . .                    и 502B.
          . . .
```

ASM.MAC

```
--------  . . .
          . . .
          . . .
         .ASECT
         .=500          При совместном линкировании MAIN.OBJ и ASM.OBJ
ADRD::   .WORD          ячейки ADRD и ADRS будут содержать адреса
ADRS::   .WORD          массивов DATA и SAVE.
          . . .
          . . .
          . . .
          . . .
```

Информация о размещении данных в Паскаль-программе может быть передана в программу на Ассемблере и без использования абсолютных адресов памяти, например путем передачи адресов с помощью фрагмента на Макро, как это выполнено в программах TRD1, TRD2 и PRDATA, причем в программе TRD2 при динамической структуре данных.

TRD1.PAS

```
TYPE buf=ARRAY[1..64] OF integer;
     adrbuf = ^buf;
VAR  data:              buf;
     adrd:              adrbuf;
     j:                 integer;
BEGIN
  adrd:=@data;
  ($C   MOV     adrd(R5),adrdA
        CALL    prdata              ->  PRDATA::MOV #64,R0
  )                                         MOV adrdA,R1
  FOR j := 1 TO 64 DO                       CLR R2
    write(data[j]:4);                1$:    MOV R2,(R1)+
  writeln                                   INC R2
END.                                        SOB R0,1$
                                            RETURN
                                            .END
```

PRDATA.MAC

```
----------
 . . .

.GLOBL adrdA
 . . .
```

TRD2.PAS

```
TYPE buf     = ARRAY[1..64] OF integer;
     adrbuf = ^buf;
VAR  adrd: adrbuf;
     j: integer;
```

```
BEGIN
  new(adrd);
  (*C   MOV     adrd(R5),adrdA
        CALL    prdata
  *)
  FOR j := 1 TO 64 DO
    write(adrd^[j]:4);
  writeln
END.
```

ЗАМЕЧАНИЕ ПО ИСПОЛЬЗОВАНИЮ ПРОЦЕДУР READ И READLN

В Паскаль-Рафос еще до первого использования процедуры read в файл INPUT записывается код окончания строки (EOLN), а файловая переменная приобретает значение кода пробела, при этом процедура read(b), где b: CHAR, т.е. процедура, имеющая в качестве файла по умолчанию файл INPUT, а в качестве буфера ячейку для одиночного символа, не обеспечивает нормального функционирования. Процедура с такими параметрами записывает в буфер код пробела (b:=40), а вслед за этим инициирует переход программы на выполнение следующего за процедурой оператора. Нормальное функционирование для этого случая может быть обеспечено использованием вторичного вызова процедуры, например,

```
read(b); read(b); ... ,
```

или предварительным вызовом процедуры readln, например

```
readln;  read(b); ... .
```

Возможность использования последней конструкции обусловлена тем, что процедура readln, работая по алгоритму/2/

```
WHILE NOT eoln(filename) DO get (filename);
get(filename)...,
```

не производит записи в буфер, а лишь перемещает указатель в файле за позицию, соответствующую коду окончания строки CR или LF. Следует отметить также, что процедура read всегда при появлении в файле кода CR или LF преобразует их в файловой переменной и буфере в код пробела /40/.

ЛИТЕРАТУРА

1. OMSI PASCAL-1 Language Specification. Oregon Microcomputer Software Inc., 1978.
2. Грогоно П. Программирование на языке Паскаль. М.: Мир,1982.

Рукопись поступила в издательский отдел
22 декабря 1987 года.

Семенов Б.Ю., Жильцов В.Е. P11-87-892

Некоторые вопросы совместного использования программ на языках Паскаль и Ассемблер для решения задач реального времени

Рассматриваются вопросы разработки программ для систем реального времени с использованием отдельно откомпилированных программ на языках Макроассемблер /Макро/ СМ ЭВМ и Паскаль-Рафос. Обсуждается распределение памяти, взаимный вызов Паскаль-программ и программ на Макро, передача параметров для этих случаев. Предлагается механизм образования общих массивов и переменных. Рассмотрен вопрос использования внешней процедуры Паскаль-программы для обслуживания прерывания.

Работа выполнена в Общеинститутском научно-методическом отделении ОИЯИ.

Сообщение Объединенного института ядерных исследований. Дубна 1987

Перевод О.С.Виноградовой

Semenov B.Yu., Zhiltsov V.E. P11-87-892

Some Questions Joint Usage of Assembler and Pascal Programs for Solving the Real Time Problems

Some questions of real time program development with the use of separately compiled Assembler and Pascal routines on SM computers are described. Memory layout, parameter transfer and cross-calling of Pascal and Macro11 routine are considered. A way to manage common arrays and variables is suggested. Some points of using external Pascal procedures for interrupt servicing is discussed.

The investigation has been performed at the Science-Methodical Department of JINR.

Communication of the Joint Institute for Nuclear Research. Dubna 1987