АВТОМАТИЗАЦИЯ ПРОЕКТИРОВАНИЯ В МИКРОЭЛЕКТРОНИКЕ.
ТЕОРИЯ, МЕТОДЫ, АЛГОРИТМЫ
(Вычислительные системы)
1978 год Выпуск 77

УДК 681.142.2

СИСТЕМА SMAPS ДЛЯ СТРУКТУРНОГО ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ АССЕМБЛЕРА ОС ЕС ЭВМ

Артур А.Крепский

Целью языка SMAPS (A System of Macros and Procedures for Structured Programming) является облегчение структурного программирования на языке ассемблера ОС ЕС ЭВМ. Достигается это обогащением языка ассемблера такими конструкциями языков высокого уровня, чтобы можно было создавать удобночитаемые и хорошо структурированные программы, не теряя достоинства машинозависимости ассемблера. Необходимость в таком инструменте появилась в процессе разработки транслятора с языка "Паскаль" для машин ЕС в Институте оснований информатики Польской Академии наук.

Система SMAPS оказалась полезной и в случаях

- программирование на языке ассемблера машин ЕС;
- отладки программ, написанных на языке ассемблера;
- обучения программированию дисциплинированным способом на уровне ассемблера.

Язык SMAPS реализован на машинах ЕС средствами ассемблера и макрогенератора. Система SMAPS состоит из макроопределений и процедур, вызовы которых устраивают любую конструкцию языка. Благодаря этому операторы языка SMAPS можно свободно перемешивать с инструкциями ассемблера, частично или полностью подменять их своими (добавляя нужные макроопределения).

Вызов системы SMAPS заключается в конкатенации библиотеки макроопределений SMAPS . MACLIB и системной (общей) библиотеки SYS1. MACLIB.

Настоящая работа основана на докладе [4], но полная документация содержится в работах [1-3]. Актуальную информацию о системе

SMAPS можно получить в объектной программе после вызова макроопределения SMAPSINF .

I. Обзор языка SMAPS

I. С т р у к т у р а п р о г р а м м ы на языке SMAPS определяется служебными словами SECTION, SEND (начало и конец подпрограммы) и BEGIN, BEND (начало и конец блока).

⟨программа⟩ ::=

⟨идентификатор⟩ SECTION [BASEREG = ⟨регистр⟩]

[GLOBAL
{⟨описание переменной⟩}$_1^\infty$]
[{⟨описание процедуры или функции⟩}$_1^\infty$]
⟨блок⟩
SEND

После глобальных (доступных и всей программе) переменных следуют описания процедур (функций) и тело главной программы. Выполнение программы начинается с тела главной программы. Оператор SECTION определяет параметры всей программы: идентификатор программы, базовый регистр BASEREG (по умолчанию R 12).

Описание процедуры (функции) определяет некоторый фрагмент программы и связывает с ним идентификатор, так что потом можно активизировать процедуру (функцию) оператором вызова процедуры.

Описание процедуры (функции) состоит из заголовка и тела процедуры (функции). Заголовок пишется в следующем виде:

⟨идентификатор⟩{PROC | FUNC} [⟨список формальных параметров⟩]
[,SAVE = {+|-} [,RESREC = ⟨регистр⟩]
[⟨спецификация формальных параметров⟩]

Включенный параметр SAVE (SAVE=+) означает сохранение всех регистров (т.е. запоминается их значение при входе в блок процедуры и восстанавливается при выходе). RESREG назначает регистр результата для функции (по умолчанию R 1).

Список формальных параметров - это перечень идентификаторов всех формальных параметров, которые непосредственно специфицируются, т.е. описывается, каким образом заменять формальные параметры фактическими.

⟨спецификация формальных параметров ⟩::=

[VAL
{⟨спецификация параметра-значения⟩}$_1^\infty$]
[VAR
{⟨спецификация параметра-переменной⟩}$_1^\infty$]
[REF ⟨параметр-адрес⟩ {, ⟨параметр-адрес⟩}$_0^\infty$]

Имеется четыре типа вызова формальных параметров процедуры (функции).

I. Параметр-значение в регистре означает, что перед вызовом процедуры (функции) в указанный параметром регистр занесено уже значение, например, спецификация параметра N в виде

```
VAL
    INTEGER (N, R3 )
```

значит, что перед вызовом процедуры (функции) регистр N (имя N становится символом R3) содержит значение, интерпретированное в теле процедуры (функции) как целое число.

2. Параметр-значение означает, что перед входом в тело процедуры (функции) значение фактического параметра (константы) заносится в память, зарезервированную для формального параметра, например:

```
VAL
    BYTE T
```

3. Параметр-переменная следует понимать так же, как параметр-значение, но фактический параметр является переменной (а не константой).

4. Параметр-адрес выделяет регистр и заносит в него адрес фактического параметра, например, REF (X,R 4) .

Вызов процедуры (функции) осуществляет следующие действия:

- подготовку замены формальных параметров фактическими,
- переход в тело процедуры (функции),
- выполнение тела процедуры (функции) и возврат в главную программу.

Вызов процедуры (функции) пишется в следующем виде:

EXEC ⟨идентификатор процедуры или функции⟩

[,⟨список фактических параметров⟩] [, LINK = ⟨регистр⟩]

Фактическими параметрами могут быть константы или переменные. Параметр LINK указывает регистр, содержащий адрес возврата из процедуры (функции); по умолчанию значением параметра LINK = 14 (если употребляется другое значение, надо в операторе BEND, кончающем тело процедуры (функции), еще раз повторить спецификацию параметра LINK).

В блоке (процедуры, функции или главной программы) допускается описание (локальных) объектов: синонимов, регистров и переменных.

Описание синонимов связывает выражение (допускаемое операндом псевдокоманды EQU) с идентификатором и пишется в следующем виде:

SYNONYM (⟨ идентификатор ⟩, ⟨ выражение ⟩)
{,(⟨ идентификатор⟩ , ⟨ выражение ⟩)}$_0^\infty$

Описание регистров следует непосредственно после слова BEGIN и служит для сохранения регистров, т.е. значение указанных регистров запоминается при входе в блок и восстанавливается при выходе из него.

ПРИМЕР. REGISTER I,3,(6,9).

Описание переменных обсуждается в п.4 (см.стр. 61).

2. О п е р а т о р ы с т р у к т у р и р о в а н и я.

⟨ оператор структурирования ⟩::=⟨ оператор ЕСЛИ ⟩|
⟨ оператор селекции⟩ |⟨ оператор ТЕСТ⟩ |⟨ оператор выбора ⟩|
⟨ оператор цикла ⟩

Результат выполнения большинства операторов структурирования зависит от заданного предиката. Подробное описание общего вида предикатов, допускаемых языком SMAPS, дадим в конце настоящего подраздела. Тут заметим только, что предикат является отрицанием, конъюнкцией или дизъюнкцией элементарных предикатов. Элементарный предикат является байтовым (может быть маскированным) или бинарным отношением, кодированным трехсимвольным идентификатором. Его первые две буквы обозначают вид отношения (EQ, NE, LT, LE, GE, GT), а третья - вид операторов (R: регистр-регистр, H, F: регистр-память, C: память-память, P: десятичная память-память, I: память-байтовый образец).

Оператор ЕСЛИ пишется в следующем виде:

IF ⟨ предикат⟩
THEN
⟨ оператор⟩
[ELSE
⟨ оператор⟩]
IFEND

Оператор селекции является обобщением оператора ЕСЛИ, допускающим больше чем один предикат:

CHOICE ⟨предикат$_1$ ⟩,...,⟨ предикат$_n$⟩DO
INSTR 1
⟨ оператор$_1$ ⟩
. . .

```
INSTR n
   <оператор_n>
[DEFAULT          ]
[  <оператор>     ]
CHOICEND
```

Специфицированные предикаты вычисляются по очереди слева направо, и выполняется оператор, соответствующий первому в очереди предикату, значение которого является истиной. Если нет такого предиката, выполняется разрешающий оператор (если он задан). Во всех остальных случаях оператор селекции игнорируется.

Для случая, когда предикаты являются сравнениями одних и тех же операндов, n раз генерируется одна и та же команда сравнения операндов. В этом случае вместо оператора селекции CHOICE предлагается употреблять более эффективный оператор TEST.

ПРИМЕР. Оператор селекции CHOICE:

а)
```
CHOICE (R3,LTF,=F'Ø'),(R3,EQF,=F'Ø'),(R3,GTF,=F'Ø') DO
INSTR 1
   L RØ,=F'-1'
INSTR 2
   SR RØ, RØ
INSTR 3
   LA RØ, 1
CHOICEND
```

б) эквивалентный оператор TEST (более эффективный):

```
TEST (R3,=F'Ø'),(LTF,1),(EQF,2),(GTF,3) DO
INSTR 1
   L RØ,=F'-1'
INSTR 2
   SR RØ,RØ
INSTR 3
   LA RØ,1
TESTEND
```

Оператор выбора CASE указывает, что должен быть выполнен один оператор из списка операторов в зависимости от значения указанного регистра (предполагается, что значение этого регистра есть целое положительное число).

ПРИМЕР.

```
CASE
INSTR 1
   AR R1,R2
INSTR 2
   SR R1,R2
CASEEND
```

Оператор цикла предписывает многократное выполнение некоторого оператора I (тела цикла):

```
LOOP[FOR =< регистр > [,FROM= <начало>] [,BY =< шаг>] ,
    {TO | DOWNTO} =< предел > DO]
WHILE < предикат > DO
  I
BREAK
UNTIL < предикат >
REPEAT
```

Выполнение оператора цикла можно прекратить безусловно (оператором BREAK) или условно - с помощью одного из следующих операторов, особенно рекомендуемых пользователю:

а) цикла с условным началом:

```
LOOP
WHILE < предикат> DO
  I
REPEAT
```

б) цикла с условным окончанием:

```
LOOP
  I
UNTIL < предикат>
REPEAT
```

в) цикла с параметром:

```
LOOP FOR =<регистр>[,FROM = < начало >] [, BY =< шаг>] ,
     {TO | DOWNTO} =< предел > DO
  I
REPEAT
```

Параметр цикла - регистр, указанный параметром FOR, - можно инициировать значением параметра FROM. Если значение шага (параметр BY) положительное (отрицательное), то верхний (нижний) предел значений параметра цикла задается параметром TO (DOWNTO).

ЗАМЕЧАНИЕ. Рассмотрим подробнее общий вид предикатов языка SMAPS

⟨ предикат ⟩::=

(⟨ элементарный предикат ⟩)|(NOT,⟨ элементарный предикат ⟩)|([NOT,]OR,⟨элементарный предикат ⟩{,⟨ элементарный предикат ⟩}$_1^\infty$)|([NOT,] AND,⟨ элементарный предикат ⟩ {,⟨ элементарный преди - кат ⟩}$_1^\infty$)

⟨элементарный предикат ⟩::=

(⟨ регистр ⟩,⟨ отношение регистр-регистр ⟩,⟨ регистр ⟩)|(⟨ регистр ⟩,⟨ отношение регистр-памяти ⟩,⟨ адрес ⟩)|(⟨ адрес памяти с заданной длиной ⟩,⟨ отношение память-память ⟩,⟨ неиндексируемый адрес памяти ⟩)|(⟨ адрес памяти с заданной длиной ⟩,⟨ де - сятичное отношение память-память ⟩,⟨ адрес памяти с заданной длиной ⟩)|(⟨ неиндексируемый адрес памяти ⟩,⟨ отношение память-бай - товый образец ⟩,⟨ байтовый образец ⟩)|([NOT,] ⟨ адрес ⟩, [⟨ маска ⟩])

⟨ отношение регистр-регистр ⟩::= ⟨вид отношения ⟩ R

⟨ отношение регистр-память ⟩::=⟨ вид отношения ⟩{H|F}

⟨ отношение память-память ⟩::=⟨ вид отношения ⟩ C

⟨ десятичное отношение память-память ⟩::=⟨ вид отношения ⟩ P

⟨ отношение память-байтовый образец ⟩::=⟨ вид отношения ⟩ I

⟨ регистр ⟩::=⟨ первый операнд инструкции LA ⟩| R0 R1|R2|R3|R4|R5|R6|R7|R8|R9|R10|R11|R12|R13|R14|R15

⟨ адрес ⟩::=⟨ второй операнд инструкции LA⟩

⟨ адрес памяти с заданной длиной ⟩::=⟨ первый операнд инст - рукции CLC ⟩

⟨ неиндексируемый адрес памяти ⟩::=⟨ второй операнд инструк - ции CLC ⟩

⟨ байтовый образец ⟩::=⟨ второй операнд инструкции CLI⟩

⟨ маска ⟩::=⟨ префикс маски ⟩⟨ байтовый образец ⟩

⟨ префикс маски ⟩::= ¬|*

⟨ вид отношения ⟩::= EQ|NE|LT|LE|GE|GT

⟨ тип отношения ⟩::= R|H|F|C|P|I

⟨ отношение ⟩::=⟨ вид отношения ⟩⟨ тип отношения ⟩

Скажем, что значение предиката истинно тогда и только тогда, когда выполнено одно из следующих условий:

I) Указанное в предикате отношение выполнено, причем значе - ние операндов определяется

- значением регистра (если операнд находится в виде ⟨ ре - гистр⟩),

- значением памяти (если операнд является адресом).

Таблица

Вид отношения	CC
EQ	0
NE	I или 2
LT	I
LE	0 или I
GE	0 или 2
GT	2

Результат отношения описан так называемым кодом условия CC('condition code'), полученным в результате выполнения инструкций CR,CH, C,CLC, CP, CLI, генерированных соответственно для отношений R,H,F,C,P,I, согласно приведенной таблице.

2) Байт памяти, указанный в предикате (⟨ адрес ⟩), содержит 'IIII IIII' или не содержит 'IIII IIII' в случае вида (NOT,⟨ адрес⟩).

3) Обобщением вышеуказанного вида предиката является следующий: (⟨ адрес ⟩,⟨ маска ⟩).

Оно заключается в том, что для указанного байта памяти проверяется значение тех битов, которые вырезаны маской (т.е. биты, соответствующие единицам в маске, рассматриваемой как двоичное число; например, маска X'FF' вырезает все биты, маска X'01' вырезает только последний бит).

Если маска не префиксована, то значение предиката (3) есть истина, когда биты, вырезанные маской, состоят только из единиц.

Если маска префиксована через '¬', то значение предиката (3) есть истина, когда биты, вырезанные маской, состоят только из нулей.

Если маска префиксована через '*', то значение предиката (3) есть истина, когда биты, вырезанные маской, состоят по крайней мере из одной единицы и одного нуля.

3. Процедуры ввода-вывода написаны на языке ассемблера; они совсем простые и исполняют самые элементарные функции при работе с системным вводом-выводом (карточный ввод, АЦПУ).

К объектной программе добавляются только тела тех процедур, которые действительно вызваны в программе. Открытие и закрытие файлов производится автоматически.

Следующий перечень содержит процедуры ввода-вывода, доступные в настоящей версии системы SMAPS:

а) Процедуры ввода READCHR [⟨адрес ⟩],READINT [⟨ адрес ⟩] предназначены для чтения с входной строки соответственно символа и целого числа. Результат - код символа или целое число заносятся в память (если указан адрес) или в регистр R1 (код символа - в крайний правый байт).

б) Процедуры вывода: WRITECHR [⟨адрес⟩], WRITETXT ⟨текст⟩ , WRITEINT ⟨адрес⟩ , [⟨ длина ⟩], WRITEREG ⟨ регистр ⟩ [{,⟨ регистр ⟩}$_1^{14}$],WRITEHEX ⟨ адрес⟩ ,⟨ длина ⟩,WRITEEOL, WRITEEOP предназначены для распечатки символа, текста,целого числа, значения регистров, значения байтов памяти в шестнадцатиричном виде,перехода к новой строке и соответственно к новой странице.

ЗАМЕЧАНИЯ.

а) ⟨ адрес ⟩ означает любое выражение, являющееся адресом на языке ассемблера;

б) текст (операнд процедуры WRITETXT) - это последовательность символов (но не более чем 256), заключенная в кавычки;

в) процедура WRITEREC печатает значения указанных регистров (за исключением базового регистра BASEREG) в следующем виде: ␣␣␣ REG ⟨I⟩␣:␣␣⟨ 8 ⟩(H)␣␣␣␣⟨ IO ⟩(D) , причем

⟨ I ⟩ - шестнадцатиричный номер регистра;

⟨ 8⟩ - восьмизначное шестнадцатиричное число, являющееся значением регистра;

⟨ IO⟩ - десятизначное десятичное число, являющееся значением регистра;

г) операнд ⟨ длина⟩ процедуры WRITEHEX,являющийся целым числом (дополненным до кратного четырем), считается шестнадцатиричным числом распечатываемых байтов (после 4 байтов,т.е. после восьми шестнадцатиричных цифр печатаются два пробела).

4. Описание переменных и констант*) Допускаются переменные глобальные (описываемые после спецификации GLOBAL в начале программы) и локальные для процедуры (функции). Различаются переменные: элементы памяти (BYTE, CHAR, HALFWORD , FVLLWORD, INTEGER, REAL, DUBLWORD) и сегменты памяти (BPOOL,HPOOL, FPOOL, DPOOL). Любые описания переменных отводят память соответствующего типа (байт, полуслово, слово, двойное слово) и длины и связывают с ней заданный идентификатор.

Описание константы отводит память и присваивает соответствующее значение.

---

*) Описанные в настоящем подразделе черты языка SMAP8 планируется удалить как по причинам сложности реализации и применения, так и вследствие их малой используемости при программировании на языке ассемблера.

ПРИМЕРЫ.

```
INTEGER I,J. K  (три слова)
CHAR   A        (байт)
HPOOL (A,2Ø)    (2I полуслово)
CONSTANT (ONE, F'1'), (TRUE, X 'FF')
```

ЗАМЕЧАНИЯ.

В основном используется следующее правило определения идентификаторов: все идентификаторы программы должны различаться между собой и состоять не больше чем из восьми литер.

Исключением из этого правила является возможность обеспечения тем же самым идентификатором разных переменных, но только для переменных, имеющих ту же длину и находящихся в параллельных блоках.

Примеры программы на языке SHAPS приведены в работе [4].

Литература

1. KREPSKI A.A. SMAPS -system makrodefinicji i procedur do programowania strukturalnego. - Prace COPAN. N 236. Warszawa,1976.

2. KREPSKI A.A. SMAPS - system makrodefinicji i procedur do programowania strukturalnego.-Prace COPAN. N 262. Warszawa, 1976.

3. KREPSKI A.A. SMAPS - system makrodefinicji i procedur do programowania strukturalnego; Opis dia uzytkownika.- Prace COPAN, N 263. Warszawa, 1976.

4. KREPSKI A.A. Structured Programming in Assembly Language, Beiträge zur Konferenz Methodik der Programmirung und Programmierifikation. Teil 1, Heft 22/77, Weiterbildungszeutrum für Mathematische Kybernetik und Rechentechnik, Technische Universität Dresden,1976, S.104–125.

Поступила в ред.-изд.отд.
I8 мая I978 года