

Design Description

Software: Pascal support in T4 code generator
Software Version: B004 Pascal beta release
Design Version: 1
Document Id:
File: ALEF::D1:[COMPILERS.T424.DOCS]PASICODE.LAY

[Rev 001	5-DEC-86, ADC]	completed first draft
[Rev 000	2-DEC-86, ADC]	created

Copyright © 3L Ltd., 1987 - 1992

1 Objectives and Scope

The software design proposed in this document will allow the transputer (T4) back end to be extended to handle the extra I-codes for Pascal.

2 Related Specifications and Documents

Some of the extra Pascal set handling I-codes are not at present specified in SYSDOC_ICODE_Working.Notes (V1.3). They are specified below.

3 Product

This design will be implemented in the beta release version of the T4 Pascal compiler for Inmos.

4 Design Description

4.1 Extra I-Codes

This section specifies those I-codes needed for Pascal which are not specified in V1.3 of SYSDOC_ICODE_Working.Notes and which have not already been implemented in the T4 back end (mostly those to do with SET data types). A functional notation is used to express the effect of I-code on the compile-time stack, which is written as a list in square brackets of items separated by semicolons. The result of each function is a (transformed) stack.

Note: the current T4 implementation of the explicit Pascal run-time checking I-codes (TEST-FOR-NIL, CHECK-DYNAMIC-BOUNDS etc) is as no-ops.

The arithmetic operator I-codes +, - and * are extended to mean set union, difference and intersection respectively when they are applied to set type operands. Assignment and parameter passing must also handle sets.

ADD-MEMBER (128+'1')

Add a member to a set.

```
ADD-MEMBER [e;S;rest] =  
  if integral(e) and type(S)=sets then [S U {e}; rest] else error
```

ADD-RANGE (128+'2')

Add a range of members to a set.

```
ADD-RANGE (hi;lo;S;rest) =  
  if integral(hi) and integral(lo) and type(S)=sets then  
    [S U {lo...hi}; rest]  
  else error
```

COMPILE-IN-1 (128+'I')

Test set membership (Boolean result).

```
COMPILE-IN-1 [S;e;rest] =  
  if type(S)=sets and integral(e) then  
    [e in S; rest]  
  else error
```

COMPILE-IN-0 (128+'i')

Test set membership (sets condition code).

```
COMPILE-IN-0 [S;e;rest] =  
  if type(S)=sets and integral(e) then  
    gen("CC := e in S");  
    [rest]  
  else error
```

EMPTY-SET (128+'Z')

Push descriptor of empty set.

```
EMPTY-SET [stack] = [ {} ; stack ]
```

4.2 Run-Time Representation

At run time, all sets are represented by a 256 bit (8 word) bit vector. This allows SET OF CHAR to be represented.

This means that expressions in set-constructors must be integral expressions in the range 0 to 255.

4.3 Compile-Time Descriptors

Set descriptors have TYPE=SETS.

Set descriptors can have form VinS or VinRec which indicate the address in memory of the first word of the bit vector representing the set.

Additionally a set descriptor can have a new form: UNBUILT SET, which represents a Pascal set-constructor expression like [e] or [e1..e2,e3] while it is being built (by ADD-MEMBER and ADD-RANGE I-codes).

A set descriptor with FORM=UNBUILT SET has a MEMBERS field (which can be overlapped with the EXTRA field) which points to a list of member records.

An UNBUILT SET with a NIL members list represents the empty set.

Each member record contains two fields A and B which point to descriptors. If B is NIL, A is a descriptor for a single member expression. Otherwise, the member record represents a range e1..e2; B is the descriptor for e1 and A is the descriptor for e2.

A single compile-time routine, BUILD SET, transforms descriptors from UNBUILT SET to VinS by dumping the constant parts of the bit vector in memory and calling library routines to set the bits corresponding to member expressions which are not constants.

BUILD SET leaves descriptors which already have FORM VinS or VinRec untouched.

To prevent multiple copies of the run time representation of the empty set (eight zero words) being dumped, BUILD SET maintains a static descriptor for the location in memory of the empty set (if one is ever generated), and returns copies of this descriptor for all further uses of the empty set.

Compile-time routines which handle set operands by passing their addresses to run time library routines must apply BUILD SET to their operands first, to make sure they have an address in memory.

4.4 Set Construction

BUILD SET generates sequences of calls on the run time library routines below to construct sets containing non-constant member expressions. The routines set the bit in the set corresponding to the value of the member expression (0..255). ADD RANGE sets all bits in the range [LOW..HIGH].

Note that the SET FM record is specified as an array of bytes rather than words: this is to allow the same implementation of the library routines to work on transputers with different word lengths (e.g. T4/T2).

record format Set Fm (byte array B(0:31))

external routine spec Add alias "PASCAL__ADD_MEMBER" -
(record(Set Fm) name S, integer Member)

external routine spec Add Range alias "PASCAL__ADD_RANGE" -
(record(Set Fm) name S, integer Low, High)

4.5 Assignment

Simple assignment is handled by the same routine (MOVE BYTES) which is used for record assignment. It generates a transputer block move instruction. As elsewhere, BUILD SET must be applied before the right-hand side of the assignment can be assumed to have a valid memory address.

*attention
SET ASSIGN*

4.6 Parameter Passing

VAR parameters are passed by reference. Set value parameters are also passed by reference, but the called procedure makes a local copy of the value. DEFINE VAR transforms set value objects which appear in parameter lists into single-word objects (pointers) and marks their descriptors with the BY REF flag which causes FINISH PARAMS to make a local copy of the value referenced by the pointer.

4.7 Set Operators

OPERATE is extended to call a new specialised routine, SET OPERATE, if either operand is of type SETS.

SET OPERATE calls BUILD SET to construct memory images of its operands if required, evaluates the operand addresses (AMAP), and passes the operand addresses to an appropriate run time library routine along with the address of a temporary (set) variable to receive the result of the operation. If the immediately following I-code is an assignment, the target variable can be used instead of a temporary. Similarly, if either

operand is already marked as a temporary variable (TEMP FLAG), that temporary can be used to store the result.

The run time library routines required are:

```
external routine spec Union alias "PASCAL__SET_UNION" -  
  ( record(Set Fm) name Result, Lhs, Rhs )
```

```
external routine spec Difference -  
  alias "PASCAL__SET_DIFFERENCE" -  
  ( record(Set Fm) name Result, Lhs, Rhs )
```

```
external routine spec Intersection -  
  alias "PASCAL__SET_INTERSECTION" -  
  ( record(Set Fm) name Result, Lhs, Rhs )
```

← mark still work if 2 or more of the args are the same variable.

4.8 Set Comparisons

TEST CONDITION is extended to call a specialised routine, COMPARE SETS, if either operand has type SETS.

Compare sets applies BUILD SET to its operands, and checks that the comparison operator is one of =, <>, >=, <=. A >= B is transformed into B <= A.

A <= B is tested by calling the library routine

```
external integer fn Leq alias "PASCAL__SET_INCLUSION" -  
  ( record(Set Fm) name A, B )
```

The result is Boolean: 1 if A is included in B, 0 otherwise.

A = B and A <> B are handled by calling the following library routine.

```
external integer fn Eq alias "PASCAL__SET_EQUALITY" -  
  ( record(Set Fm) name A, B )
```

Again, the result is Boolean: 1 if A equals B, 0 otherwise.

For an inequality test <>, COMPARE SETS must generate a code fragment to invert the result returned by the library routine.

Membership tests (e in S) are handled by a separate compile time routine, COMPILE IN. This generates a call on the following library routine.

```
external integer fn In alias "PASCAL__SET_IN" -  
  ( integer e, record(Set Fm) name S )
```

The result is 1 if e is a member of S (i.e. the eth bit is set in the representation of S). Otherwise the result is 0.

The COMPILE-IN-0 I-code, which must set the I-code "condition code", is implemented by comparing the result returned by PASCAL__SET_IN with zero, then assuming that the next (PENDING) I-code is either BF or BT, and changing it into BEQ or BNE respectively.

note 10000 in S is always false with the 0..255 limit

5 Assumptions and Limitations

No run-time checking is provided.