Machine Code Generation for SIMNON on VAX-11

Essebo, Tommy

1981

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Essebo, T. (1981). *Machine Code Generation for SIMNON on VAX-11*. (Technical Reports TFRT-7217). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:
1

CODEN: LUTFD2/(TFRT-7217)/1-045/(1981)

MACHINE CODE GENERATION FOR SIMNON ON VAX-11

T ESSEBO

DEPARTMENT OF AUTOMATIC CONTROL
LUND INSTITUTE OF TECHNOLOGY
MAY 1981

LUND INSTITUTE OF TECHNOLOGY
DEPARTMENT OF AUTOMATIC CONTROL
Box 725
S 220 07 Lund 7 Sweden

Document name
Internal Report

Date of issue
May 1981

Document number
CODEN:LUTFD2/(TFRT-7217)/1-45/(1981)

Supervisor

Sponsoring organization

Author(s)
Tommy Essebo

Title and subtitle
Machine code generation for Simnon on VAX-11

Abstract
The compiler in the interactive simulation program SIMNON generates pseudocode of the equations to be simulated. The pseudocode is interpreted at simulation time.

This report describes a new stage in the compiler that generates machine code directly for the host computer from the pseudocode. The code generation for VAX-11 is described in detail.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
45

Recipient's notes

Security classification

Distribution: The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Telex: 33248 lubbis lund.

1. INTRODUCTION

SIMNON is a FORTRAN-written interactive simulation program for non-linear systems described by ordinary differential equations and difference equations (Elmqvist, 1975).

SIMNON contains a compiler for the simulation language that produces a pseudocode which is interpreted and executed by a FORTRAN subroutine (CALCUL) at simulation time. This makes the execution of the code slow compared to code from a compiler that produces machine code. It is possible to add a new stage to SIMNON that takes the pseudocode and generates machine code for a specific computer. The code is then executed directly by the machine rather than interpreted by a program. The cost of generating machine code from the pseudocode is small compared to the cost of producing the pseudocode from the source files. (The term cost in this case means resources such as memory and CPU-time.) The gain in execution speed has been 5 - 8 times for the existing implementations.

Chapter 2 describes the output from the compiler to be used as input for the code generator. This information is mostly taken from (Elmqvist, 1978).

Chapter 3 describes the code generation in general and chapter 4 describes the code generated for VAX-11.

2. OUTPUT FROM SIMNON

This chapter describes the format of the output from the SIMNON compiler which will be the input to the code generator.

The pseudocode is stored in an integer array in common /PSCODE/. It is organized as linked lists. A node contains one or more equations or a call of a section in an external FORTRAN system (via SYSTS). The pseudocode area contains five different lists for different kinds of computations in the simulation part such as: initial computations, derivative computations or computation of discrete states. For the code generator it is irrelevant when each list is used. The important thing is where each list starts and this is stored in commonblock /ENTRYS/. Each variable in /ENTRYS/ points to the head of a list in /PSCODE/. See fig. 1 for description of a node head.

FP	Forward pointer (points to next node)
BP	Backward pointer
LEN	Length of data in the node
IASYST	Index to subsystem at generation time
MODE	Compiler mode at generation time
NEQ	Number of equations in the node
	Pseudocode of equations LEN words

Fig. 1: Node organization

Each list in the pseudocode consists of a list head and zero or more nodes. A list head is an empty node (LEN=0).

The pseudocode consists of operators (integer 1 - 22) followed by zero or more integer operands. See fig. 2.

Output from SIMNON

OPERATION	MNEMONIC	CODE
Logical or	OR	1
Logical and	AND	2
Logical not	NOT	3
Test less than	TLT	4
Test greater than	TGT	5
Add	ADD	6
Subtract	SUB	7
Multiply	MUL	8
Divide	DIV	9
Negate	NEG	10
Raise	RAI	11
Jump if false	JMPF	12
Jump	JMP	13
Stack	FETCH	14 pointer
Unstack	DEPOS	15 pointer
Apply function	FUNC	16 function number
Call FORTRAN system	CALL	20 isyst ipart
Skip if not sampling	SCOND	21 system number
No operation	NOP	22

Fig. 2: Operation set

The SIMNON compiler produces RPN (Reverse Polish Notation) code operating on a stack.

A detailed description of the operations follows.

P(n) is top stack element.
n is the stack pointer
k is the index in the pseudocode (=PC, program counter)
Logical values are 0.0 (false) and 1.0 (true). A value is true if it is greater or equal 0.5
The pointer used in FETCH and DEPOS has the following meaning: if pnt) 10000 it points to a litteral stored in common /VALUES/V(pnt-10000) otherwise it points to a variable whose address is stored in common /VARTB2/IPNTS(pnt).

OR P(n-1):=(P(n-1)≥0.5) or (P(n)≥0.5); n:=n-1; k:=k+1

AND P(n-1):=(P(n-1)≥0.5) and (P(n)≥0.5); n:=n-1; k:=k+1

Output from SIMNON

NOT	P(n):= not (P(n)≥0.5); k:=k+1
TLT	P(n-1):=P(n-1)<P(n); n:=n-1; k:=k+1
TGT	P(n-1):=P(n-1)>P(n); n:=n-1; k:=k+1
ADD	P(n-1):=P(n-1)+P(n); n:=n-1; k:=k+1
SUB	P(n-1):=P(n-1)-P(n); n:=n-1; k:=k+1
MUL	P(n-1):=P(n-1)*P(n); n:=n-1; k:=k+1
DIV	P(n-1):=P(n-1)/P(n); n:=n-1; k:=k+1
NEG	P(n):=-P(n); k:=k+1
RAI	P(n-1):=P(n-1)**P(n); n:=n-1; k:=k+1
JMPF nr	k:=if P(n)<0.5 then k+nr+1 else k+2; n:=n-1 (nr > 0)
JMP nr	k:=k+nr+1 (nr > 0)
FETCH pnt	n:=n+1; P(n):= var_{pnt} ; k:=k+2
DEPOS pnt	var_{pnt} :=P(n); n:=n-1; k:=k+2
FUNC nr	one-argument function: P(n):= $func_{nr}$ (P(n)); k:=k+2 two-argument function: P(n-1):= $func_{nr}$ (P(n-1),P(n)); n:=n-1; k:=k+2
CALL isyst ipart	ISYST:=isyst; IPART:=ipart; call SYSTS; if ISTOP then EXIT from CALCUL; k:=k+3
SCOND nr	if LCOND(nr) then k:= next node else k:=k+2
NOP	k:=k+1

Output from SIMNON

The following functions are used in FUNC:

NR	NAME	DESCRIPTION
1	SQRT(X)	square root of X, X≥0
2	EXP(X)	exponential function of X
3	LN(X)	natural logarithm of X, X>0
4	LOG(X)	logarithm (base 10) of X, X>0
5	SIN(X)	sine of X (X in radians)
6	COS(X)	cosine of X (X in radians)
7	TAN(X)	tangent of X (X in radians)
8	ATAN(X)	arctangent of X, result in radians in interval [-π/2,π/2]
9	ABS(X)	absolute value of X
10	SIGN(X)	the sign of X: +1.0 if x>0; 0.0 if x=0; -1.0 if x<0
11	INT(X)	integer part of x
12	ATAN2(X,Y)	arctangent of x/y, result in radians in interval [-π,π]
13	MOD(X,Y)	X modulo Y (X-INT(X/Y)*Y)
14	MIN(X,Y)	minimum of X and Y
15	MAX(X,Y)	maximum of X and Y
16	ARCSIN(X)	arcsine of X in interval [-1,1]
17	ARCCOS(X)	arccosine of X in interval [-1,1]
18	SINH(X)	hyperbolic sine of X
19	COSH(X)	hyperbolic cosine of X
20	TANH(X)	hyperbolic tangent of X

A few examples of SIMNON pseudocode follows:

```
 Y1 = (-A)/(3 + B)
FETCH          A
NEG
FETCH         3.00
FETCH          B
ADD
DIV
DEPOS          Y1

 Y2 = A OR NOT B
FETCH          A
FETCH          B
NOT
OR
DEPOS          Y2

 Y3 = (A<B) + 2
FETCH          A
FETCH          B
TLT
FETCH         2.00
```

Output from SIMNON

```
ADD
DEPOS        Y3

 Y4 = IF A THEN 1 ELSE B↑2
FETCH        A
JMPF           5
FETCH      1.00
JMP            6
FETCH        B
FETCH      2.00
RAI
DEPOS        Y4

 Y5 = SIN(ATAN2(A,B + 1))
FETCH        A
FETCH        B
FETCH      1.00
ADD
FUNC         ATAN2
FUNC         SIN
DEPOS        Y5

 Y6 = IF A<B THEN (IF NOT B THEN 1 ELSE A + B) ELSE
      IF A + B THEN 2 ELSE B
FETCH        A
FETCH        B
TLT
JMPF          17
FETCH        B
NOT
JMPF           5
FETCH      1.00
JMP            6
FETCH        A
FETCH        B
ADD
JMP           14
FETCH        A
FETCH        B
ADD
JMPF           5
FETCH      2.00
JMP            3
FETCH        B
DEPOS        Y6

 Y7 = -(A*(-B)/C - A/B + B*C - 2)
FETCH        A
FETCH        B
NEG
FETCH        C
DIV
MUL
FETCH        A
FETCH        B
```

Output from SIMNON

```
      DIV
      SUB
      FETCH          B
      FETCH          C
      MUL
      ADD
      FETCH          2.00
      SUB
      NEG
      DEPOS          Y7
```

3. CODE GENERATOR - GENERAL PRINCIPLES

The code generator scans the pseudocode in one pass and generates machine instructions directly for each pseudocode instruction. Since the pseudocode contains forward jumps it is necessary to go back and insert the jump address in the generated code when the target of the jump is processed.

The stack used in the RPN pseudocode corresponds to an operand address stack used by the code generator but there is no explicit stack or stack instructions in the generated code. The code generator makes use of the available general purpose fast hardware registers for storing intermediate results of computations and the allocation is made from a stack of free registers. Since the number of available registers generally is smaller than a worst-case expression in SIMNON the code generator automatically allocates temporary memory cells when the register stack is exhausted. The code generator will in fact work even when only one register is available.

The operand stack and register allocation

Consider the equation Y=(A+2)*B. It generates the following pseudocode:

```
1 FETCH  A
3 FETCH  2
5 ADD
6 FETCH  B
8 MUL
9 DEPOS  Y
```

The code generator performs the following actions:

1: Push addr(A) on operand stack
3: Push addr(2) on operand stack
5: Pop the stack twice; if operands are registers, return the registers to the register stack; get a new register from the register stack; issue an instruction to add the two operands and store result in the register; push the register (or rather the address of the register) on the operand stack
6: Push addr(B)
8: Pop the stack twice; release registers; get a register; issue multiply instruction; push register
9: Pop the stack; release register; issue instruction to store operand in Y

The following instructions could be generated:

```
ADD   A,2   →R1
MULT  R1,B  →R1
STORE R1    →Y
```

If the code generator is to be used for a computer that only allows one of the operands to be a memory cell the code would be:

```
    LOAD  A     →R1
    ADD   R1,2        ; result is in register R1
    MULT  R1,B
    STORE R1    →Y
```

The logic would be: at step 5, 8 and 9 check that at least one of the popped operands is in a register, if not issue a load instruction

The operand address stack thus will contain a mixture of (addresses of) variables and registers. The registers represent results from already issued instructions. Note the difference between this stack and the stack used by the interpreter for the pseudocode that contains only values of variables or intermediate computations.

The register stack

Adresses of the free registers are stored in a stack that doesn't exist! The only variable actually associated with the stack is the stackpointer. The reason for this is that the registers are considered to be an ordered set and allocation/deallocation of the registers always made in that order. (Of course deallocation is made in the opposite order of allocation.) This makes it possible to compute the address of the register on top of the "stack" directly from the value of the stackpointer. This principle of allocating registers is important when generating code for conditional branches as shown later on.

Jumps

As mentioned earlier it is necessary to go back in the generated code and "patch" whenever a jump is encountered. The SIMNON compiler produces three branch or jump instructions, all of which are forward jumps. JMPF and JMP are results of IF-THEN-ELSE expressions and the displacement is given as a relative pseudocode address. SCOND is used for execution of equations only at specified conditions (i.e. time for sampling). The displacement is always to the end of the current node. The corresponding pseudocode address is easy to find using the information in the node head. When a jump instruction is encountered in the pseudocode the code generator issues a branch instruction with a void displacement and stores the following information in a special jump address table:

a) the target pseudocode address
b) the current abs. code address
c) the type of jump instruction

For each pseudo instruction processed a check is made if the current pseucode address is in column a) in the jump table. If it is found the current abs. code address is inserted in the abs. code at the address indicated by b) and the entry is removed from the table.

Conditional branches

The equation: Y = IF cond THEN expr1 ELSE expr2 generates the following pseudocode sequence:

```
     cond-code
     JMPF   L1
   -----------
     expr1-code   block A
   -----------
     JMP    L2
   -----------
 L1: expr2-code   block B
   -----------
 L2: DEPOS  Y
```

The interpreter evaluates the condition and then one of the blocks A or B. The code generator proceeds through both A and B and must generate correct code for both cases. Since both blocks in an IF-THEN-ELSE construction consists of expressions the result at the end of the block will be on top of the operand stack either in a register or in a variable (only if the expression is a simple variable). Furthermore the operand stack has increased with exactly one element from the beginning of the block to the end.

A necessary condition for correct code after the two branches is that at the end of both blocks the result is at the same address, i.e. the top element on the operand stack must be exactly the same. The code generator solves this the following way:

At the end of each block it checks if the top stack element is a variable, in which case a register is popped from the register stack and an instruction to load the variable into the register issued. The register operand is the pushed on the operand stack.

Furthermore it is necessary to restore both the operand stack and the register stack to the status of the beginning of the block at the end of block A to ensure that the status is the same at the end of both blocks. This is easily done by popping the operand stack and releasing the register.

The end of block A is always followed by a JMP instruction and the end of block B is followed by the target of a JMP instruction.

Code generator - general principles

Temporary variables

The temporary variables are allocated linearly from an array and are deallocated only at the end of an equation (DEPOS). There are two cases when temporary variables are needed. The first case is when a mathematical procedure is called and at least one argument is in a register. This is because of the parameter transfer mechanism that needs the address of an argument. Since registers are (usually) addressed differently from memory cells it would cause difficulties if an argument address was a register. Furthermore the mathematical procedure might use the register internally of course after saving the contents in a memory cell but the argument address would still point to the register.

The other case is when the register stack is exhausted. In this case the strategy is as follows: The operand stack is scanned from the top and when a register is found it is released and a load instruction to a temporary variable issued. The address of that variable will then replace the register address in the operand stack. The released register is of course always the last register in the register stack. This strategy will take care of any problem that could arise when generating code for conditional branches.

Interface SIMNON - absolute code

To execute code in one of the five lists SIMNON takes one of the values from /ENTRYS/ and stores it in /ENTRY/ and then calls CALCUL (without arguments). The code generator replaces the pseudocode entrypoints in /ENTRYS/ with absolute code start addresses and thus the value in /ENTRY/ is the absoulte start address in the generated code when CALCUL is called. CALCUL must be written in assembler but can be very simple: it makes a subroutine jump to the address in /ENTRY/ (there must be a possibility to return back to CALCUL after the code is executed).

Run-time tracing

The code generator generates one additional instruction for each new node that simply stores the node pointer for the pseudocode node in common /CSIERR/. This gives a possibility to give run-time error messages and print the source equation if the pseudocode is saved since there exists subroutines in SIMNON that will reconstruct and print out a source equation if a pointer to the pseudocode node is given. The kind of errors where this is used are arithmetic faults (floating-point overflow, divide with zero) or illegal arguments to a mathematical procedure, e.g. SQRT(-1). These errors will usually cause an interrupt or trap and most systems gives the user the possibility to write a routine that is automatically executed when such a condition occurs.

Code generator - general principles

Debugging

It is very valuable to be able to print out information concerning the code generation both for debugging purposes and checking of the generated code. The code generator will print information on four levels:

a) the source equations
b) the input pseudocode
c) the generated code in symbolic assember format
d) the genereated machine code in hexadecimal format

The printout is governed by flags in common /VXCLOG/ and the values are taken from global variables that can be changed by the LET command in SIMNON.
a) is turned on if LOGSRC.VCODE is non-zero
b) is turned on if LOGPSE.VCODE is non-zero
c) is turned on if LOGINS.VCODE is non-zero
d) is turned on if LOGHEX.VCODE is non-zero
The logical unit number for the output can be given in LUNLOG.VCODE. The information is printed incrementally, i.e. each time a new instruction is generated or next pseudoinstruction read.

4. CODE GENERATION FOR VAX-11

The architecture of the VAX includes variable-length instructions with a large number of addressing modes. Instructions can have up to three different operands which can either be registers or memory cells. There are 16 general purpose registers numbered R0 - R15 and R2 - R9 are used for the register stack in the code generator. The constants 0.5 and 1.0 are always in R10 and R11 since they are frequently used for logical testing and setting. All addresses refer to bytes. Detailed descriptions of the instructions and addressing modes are given in (VAX Architecture Handbook, 1979). The interface to the mathematical procedures is described in (VAX Run-Time Library, 1980)

The code generated for each pseudocode operation is described shortly. The VAX code is represented by symbolic assembler mnemonics somewhat simplified. The notation for the different addressing modes is not used. 0.5 and 1.0 are used as litterals instead of the registers R10 and R11. Rx means a register from the register stack (R2 - R9).

Logical operators: OR, AND, NOT, TLT, TGT

The logical operators will give as result the values 0.0 (false) or 1.0 (true). A common case is when the value is used for testing in an IF-construction. The code generator tests this by looking at the next pseudo instruction to see if it is JMPF. The generated code will then be different.

A_OR_B
```
     CMPF   A,0.5
     BGEQ   L1
     CMPF   B,0.5
     BGEQ   L1
     CLRF   Rx
     BRB    L2
L1:  MOVF   1.0,Rx
L2:
```

IF A OR B THEN ...
```
     CMPF   A,0.5
     BGEQ   L1
     CMPF   B,0.5
     BGEQ   L1
     BRW    J1
L1:
```

A_AND_B
```
     CMPF   A,0.5
     BLSS   L1
     CMPF   B,0.5
```

```
        BLSS    L1
        MOVF    1.0,Rx
        BRB     L2
L1:     CLRF    Rx
L2:
```

IF A AND B THEN ...

```
        CMPF    A,0.5
        BLSS    L1
        CMPF    B,0.5
        BLSS    L1
        BRB     L2
L1:     BRW     J1
L2:
```

NOT A

```
        CMPF    A,0.5
        BGEQ    L1
        MOVF    1.0,Rx
        BRB     L2
L1:     CLRF    Rx
L2:
```

IF NOT A THEN ...

```
        CMPF    A,0.5
        BLSS    L1
        BRW     J1
L1:
```

A<B

```
        CMPF    A,B
        BLSS    L1
        CLRF    Rx
        BRB     L2
L1:     MOVF    1.0,Rx
L2:
```

IF A<B THEN ...

```
        CMPF    A,B
        BLSS    L1
        BRW     J1
L1:
```

The code for TGT (>) is the same as for TLT except that BLSS is changed to BGTR.

Arithmetic binary operations: ADD, MUL, SUB, DIV

Since ADD and MUL are symmetric (commutative) they are grouped together and the only difference in the code is the instruction code: ADDFx or MULFx. If at least one of the operands are in a register the 2-address form of the instruction is used, otherwise the 3-address form.

Code generation for VAX-11

A + B
```
     ADDF3   B,A,Rx
```

A + Rx
```
     ADDF2   A,Rx
```

Rx + B
```
     ADDF2   B,Rx
```

SUB and DIV are also grouped together but here it's only if the first operand is a register that the 2-address form can be used.

A - B
```
     SUBF3   B,A,Rx
```

Rx - A
```
     SUBF2   A,Rx
```

A - Rx
```
     SUBF3   Rx,A,Rx
```

NEG - Unitary minus

-A
```
     MNEGF   A,Rx
```

RAI - Raise a number

The mathematical procedure from the Runtime Library for raising a real base to a real power is used. Note that the values of the arguments instead of the addresses are pushed on the stack in this case.

A ↑ B
```
     PUSHL   B
     PUSHL   A
     CALLS   2,MTH$POWRR
     MOVF    RO,Rx
```

Jump instructions: JMPF, JMP

Since the displacement might exceed 256 bytes a BRW instruction is used instead of the shorter BRB. JMPF is the conditional branch of an IF-THEN-ELSE expression. If the condition is the result of a logical operation it is already taken care of (see above) otherwise the following code is generated:

IF A THEN ...
```
     CMPF    A,0.5
     BGEQ    L1
```

```
     BRW     J1
L1:
```

Since JMP marks the end of the first branch block in an IF-THEN-ELSE an instruction to load the operand on top of the stack in a register is issued if it's in a variable.

FETCH_and_DEPOS

FETCH will not generate any code, DEPOS will generate an instruction to store the operand in a memory cell.

A_=_B
```
     MOVF    A,B
```

FUNC_-_library_function_call

Some of the functions are computed directly in inline code (ABS, SIGN, MIN, MAX) and the rest are calls to Runtime Library routines. In this case the argument addresses are pushed on the stack. If an argument is in a register it is first loaded in a temporary variable.

SIN(A)   one-argument function call
```
     PUSHAL  A
     CALLS   1,MTH$SIN
     MOVF    R0,Rx
```

ATAN2(Rx,A)   two-argument function call
```
     PUSHAL  A
     MOVF    Rx,TEMP$1
     PUSHAL  TEMP$1
     CALLS   2,MTH$ATAN2
     MOVF    R0,Rx
```

Inline_coded_functions:

MAX(A,B)   (BLSS is changed to BGTR in MIN)
```
     CMPF    B,A
     BLSS    L1
     MOVF    B,Rx
     BRB     L2
L1:  MOVF    A,Rx
L2:
```

SIGN(A)
```
     TSTF    A
     BGTR    L1
     BLSS    L2
     CLRF    Rx
     BRB     L3
L1:  MOVF    1.0,Rx
     BRB     L3
```

```
L2:  MNEGF  1.0,Rx
L3:
```

ABS(A)

```
     BICL3  00008000,A,Rx
```

CALL - calling external systems

Note that the values to be stored in ISYST and IPART are integers known at compile time.

```
     MOVZWL isyst,ISYST
     MOVZWL ipart,IPART
     CALLS  0,SYSTS
     BBC    0,ISTOP,L1
     RSB
L1:
```

SCOND - conditional skip of a node

The pseudocode interpreter makes a call to a FORTRAN-written logical function called LCOND to evaluate the condition for SCOND. The code generator generates inline code instead.

```
     BBC    0,LSAMP,L1
     BBS    0,LSAMPS(i),L1
     BRW    0
L1:
```

Code examples:

The examples listed earlier will generate the following code. (The instruction to store the node number for each new node is omitted.)

```
; Y1 = (-A)/(3 + B)
     MNEGF  A,R2
     ADDF3  B,3.0,R3
     DIVF2  R3,R2
     MOVF   R2,Y1

; Y2 = A OR NOT B
     CMPF   B,0.5
     BGEQ   L1
     MOVF   1.0,R2
     BRB    L1
     CLRF   R2
L1:  CMPF   R2,0.5
     BGEQ   L2
     CMPF   A,0.5
     BGEQ   L2
     CLRF   R2
```

Code generation for VAX-11

```
      BRB     L3
L2:   MOVF    1.0,R2
L3:   MOVF    R2,Y2

; Y3 = (A < B) + 2
      CMPF    A,B
      BLSS    L1
      CLRF    R2
      BRB     L2
L1:   MOVF    1.0,R2
L2:   ADDF2   2.0,R2
      MOVF    R2,Y3

; Y4 = IF A THEN 1 ELSE B↑2
      CMPF    A,0.5
      BGEQ    L1
      BRW     J1
L1:   MOVF    1.0,R2
      BRW     J2
J1:   PUSHL   2.0
      PUSHL   B
      CALLS   2,MTH$POWRR
      MOVF    R0,R2
J2:   MOVF    R2,Y4

; Y5 = SIN(ATAN2(A , B + 1))
      ADDF3   1.0,B,R2
      MOVF    R2,TEMP$1
      PUSHAL  TEMP$1
      PUSHAL  A
      CALLS   2,MTH$ATAN2
      MOVF    R0,R2
      MOVF    R2,TEMP$2
      PUSHAL  TEMP$2
      CALLS   1,MTH$SIN
      MOVF    R0,R2
      MOVF    R2,Y5

; Y6 = IF A < B THEN (IF NOT B THEN 1 ELSE A + B) ELSE
      IF A + B THEN 2 ELSE B
      CMPF    A,B
      BLSS    L1
      BRW     J3
L1:   CMPF    B,0.5
      BLSS    L2
      BRW     J1
L2:   MOVF    1.0,R2
      BRW     J2
J1:   ADDF3   B,A,R2
J2:   BRW     J5
J3:   ADDF3   B,A,R2
      CMPF    R2,0.5
      BGEQ    L3
      BRW     J4
L3:   MOVF    2.0,R2
```

Code generation for VAX-11

```
     BRW    J5
J4:  MOVF   B,R2
J5:  MOVF   R2,Y6

; Y7 = -(A*(-B)/C - A/B + B*C - 2)
     MNEGF  B,R2
     MULF2  A,R2
     DIVF2  C,R2
     DIVF3  B,A,R3
     SUBF2  R3,R2
     MULF3  C,B,R3
     ADDF2  R3,R2
     SUBF2  2.0,R2
     MNEGF  R2,R2
     MOVF   R2,Y7
```

5. REFERENCES

Elmqvist, H (1975): SIMNON - An interactive simulation program for nonlinear systems, User's manual, TFRT 7502

Elmqvist, H (1978): SIMNON - An interactive simulation program, Implementation, TFRT 7148

VAX-11 Architecture Handbook (1979-80) Digital Equipment Corporation

VAX-11 Run-Time Library Reference Manual (1980) Digital Equipment Corporation

6. Program listings

VCODE.FOR	- Main subroutine for code generation
CALCUL.MAR	- Interface routine between Simnon and machine code
GETREG.FOR	- Register assignment subroutine
NEXTOP.FOR	- Decodes pseudo code operation
UNSTK.FOR	- Handles operand stack
VBYTE.FOR	- Writes intruction bytes in code area
VINSTX.FOR	- Machine code instruction generator

```
      SUBROUTINE VCODE
C
C     GENERATES MACHINE CODE FOR VAX-11 FOR SIMNON ,USING RPN
C     PSEUDO CODE IN /PSCODE/ AS INPUT.
C
C     AUTHOR T. ESSEBO 1980-11-11
C     DEPARTMENT OF AUTOMATIC CONTROL
C     LUND INSTITUTE OF TECHNOLOGY
C     LUND , SWEDEN
C
C     SUBROUTINES REQUIRED
C          IADDR
C          NEXTOP
C          UNSTK
C          GETREG
C          VINSTX
C
C     DATA BASE:
C
C     /PSCODE/, /ENTRYS/, /DESTIN/, /USER/ , /VARTB2/ , /VALUES/ ,
C     /SIMN/, /MESSS/, /DEVICE/, /LIMITS/ AND /COND/ : SEE SIMNON
C
C     /UPNTS/  CONTAINS VARIOUS POINTERS
C              L-     CURRENT PSEUDO CODE INDEX
C              IHEAD- POINTER TO CURRENT LIST IN /PSCODE/
C              NODE- POINTER TO CURRENT NODE IN THE LIST
C              LEN- REMAINING LENGTH OF CURRENT NODE
C              IPCNT- REL. ADDR. FOR NEXT INSTRUCTION
C              IRGTOP- POINTER TO NEXT FREE REGISTER IN THE (NON-
C                    EXISTENT) REGISTER STACK. THE ADDRESS OF THE
C                    REGISTER IS COMPUTED FROM IRGTOP AS: -3 -IRGTOP
C              ITOP- STACK POINTER FOR OPERAND ADDRESS STACK
C
C     /UDATA/  CONTAINS TABLES AND STACKS
C              ISTACK(*)- OPERAND ADDRESS STACK
C              ICODE(*)- STORAGE FOR ABSOLUTE CODE
C
C     /UTEMPC/ CONTAINS TEMPORARY CELLS USED 1): IF THE REGISTER
C              STACK IS EMPTY AND 2): FOR STORING REGISTER
C              OPERANDS IN LIBRARY FUNCTION CALLS
C              ITMPNT - POINTER TO LAST USED TEMP. VARIABLE
C              MAXTMP - MAX VALUE ALLOWED OF ITMPNT
C              ITMPAD - ABS. ADDRESS TO ITEMP(1)
C              ITEMP(*) - TEMPORARY VARIABLES
C
C     /CSIERR/ INODE WILL CONTAIN THE NODE NUMBER OF THE EQUATON
C              CURRENTLY BEING EXECUTED IN A SIMU COMMAND. THIS GIVES
C              A POSSIBILITY TO PRINT ERROR MESSAGES AT RUN TIME
C              REFERENCING SOURCE LINE EQUATIONS.
C
C     /VXCLOG/ USED FOR DEBUGGING AND CHECKING OF CODE GENERATOR
C              LUNLOG - LOGICAL UNIT NUMBER FOR DEBUG OUTPUT
C              LOGSRC - IF NON-ZERO: PRINT SOURCE NODE EQUATION(S)
C              LOGPSE - IF NON-ZERO: PRINT PSEUDO-CODE INSTR.
C              LOGINS - IF NON-ZERO: PRINT ASSEMBLY INSTR.
C              LOGHEX - IF NON-ZERO: PRINT HEXADEC. MACHINE CODE
C
C     INTERNAL DATA FOR UCODE:
C
C     IFTAB(2,20) - FUNC TABLE
C              1,* - IF )0 :GOTO INDEX TO INTRINSIC CODE GENERATION
```

```
C                       SEQUENCE ELSE
C                 -(1,*) IS INDEX IN IFNADR(*) TO ABSOLUTE ENTRY
C                        OF THE LIBRARY FUNCTION AND (2,*) IS NR. OF ARGS
C
C      IBASE- ABSOLUTE BASE ADDRESS FOR THE CODE
C      JMPPNT- POINTER TO LAST USED ENTRY IN JMP TABLE
C      JMPTAB(3,*)- FORWARD JUMP TABLE
C            1- JUMP-TO PSEUDO CODE ADDR.
C            2- JUMP-FROM REL. CODE ADDR (IPCNT ADDR)
C            3- TYPE OF JUMP:
C              1: JMPF
C              2: JMP
C              3: SCOND
C
C      NOTES ON REGISTER USAGE:
C
C      THERE ARE 16  AVAILABLE REGISTERS NUMBERED FROM R0 TO R15
C      WITH THE CORRESPONDING ADDRESSES 0 TO 15. TO BE ABLE TO TELL
C      IF AN OPERAND ADDRESS IS A REGISTER ,THEIR NEGATIVE VALUES-1 ARE
C      USED IN THE PROGRAM.
C      R2 - R9 ARE USED IN THE REGISTER STACK, R10 AND R11 ARE USED
C      FOR CONSTANTS 0.5 AND 1.0
C
       LOGICAL ISTOP,NXJPF,NOSYST,NOCODE,LSAMP,
     1LSAMPS,EOL,LDUM1,LDUM2,LDUM3
C
       INTEGER MOVF,CLRF,MNEGF,CMPF,ADDF2,SUBF2,MULF2,DIVF2,BGEQ,
     *         ADDF3,SUBF3,MULF3,DIVF3,BRB,BRW,CALLS,BLSS,BBS,
     *         BGTR,MOVZWL,BICL2,BICL3,TSTF,RSB,PUSHL,PUSHAL,BBC
       INTEGER R0,R1,R2,RZP5,R1PZ,SIGNEG
C
       DIMENSION IFTAB(2,20),IFNADR(16),JMPTAB(3,10),HSRC(2),
     *           HLUN(2),HPSE(2),HINS(2),HHEX(2),HEXT(2),PSOP(23),
     *           NPSOP(23)
C
       COMMON/DEVICE/LKB,LTP,LLP,LDIS,LTO,LPLOT,LXXX,LDK1,LDK2,LDK3,LDK4
       COMMON/SIMN/NOSYST,LDUM1(4),NOCODE,LDUM2(4)
       COMMON/LIMITS/MPSC,IDUM3(4)
       COMMON/MESSS/MESS
       COMMON/PSCODE/IPSEUD(100)
       COMMON/VARTB2/IPNTS(100)
       COMMON/VALUES/VALUE(100)
       COMMON/ENTRYS/IENT(5)
       COMMON/COND/LSAMP,LSAMPS(5)
       COMMON/DESTIN/ISYST,IPART
       COMMON/USER/ISTOP,LDUM3(6),IDUM2
       COMMON/UPNTS/L,IHEAD,NODE,LEN,IPCNT,IRGTOP,ITOP
       COMMON/UDATA/ISTACK(25),ICODE(7000)
       COMMON/UTEMPC/ITMPNT,MAXTMP,ITMPAD,ITEMP(20)
       COMMON/CSIERR/INODE,INODEX
       COMMON/VXCLOG/LUNLOG,LOGSRC,LOGPSE,LOGINS,LOGHEX
C
       EXTERNAL OTS$POWRR,MTH$SQRT,MTH$EXP,MTH$ALOG,MTH$ALOG10,
     *          MTH$SIN,MTH$COS,MTH$TAN,MTH$ATAN,MTH$AINT,
     *          MTH$ASIN,MTH$ACOS,MTH$SINH,MTH$COSH,MTH$TANH,
     *          MTH$ATAN2,MTH$AMOD,SYSTS
C
       DATA
     *      MOVF/'00000050'X/
     *     ,MNEGF/'00000052'X/
     *     ,CLRF/'000000D4'X/
```

```
     *     ,CMPF/'00000051'X/
     *     ,ADDF2/'00000040'X/
     *     ,SUBF2/'00000042'X/
     *     ,MULF2/'00000044'X/
     *     ,DIVF2/'00000046'X/
     *     ,ADDF3/'00000041'X/
     *     ,SUBF3/'00000043'X/
     *     ,MULF3/'00000045'X/
     *     ,DIVF3/'00000047'X/
     *     ,BGEQ/'00000018'X/
     *     ,BLSS/'00000019'X/
     *     ,BGTR/'00000014'X/
     *     ,BRB/'00000011'X/
     *     ,BRW/'00000031'X/
     *     ,CALLS/'000000FB'X/
     *     ,MOVZWL/'0000003C'X/
     *     ,BICL2/'000000CA'X/
     *     ,BICL3/'000000CB'X/
     *     ,TSTF/'00000053'X/
     *     ,RSB/'00000005'X/
     *     ,PUSHL/'000000DD'X/
     *     ,PUSHAL/'000000DF'X/
     *     ,BBC/'000000E1'X/
     *     ,BBS/'000000E0'X/
C
      DATA IFTAB/
     *     -1,1,
     *     -2,1,
     *     -3,1,
     *     -4,1,
     *     -5,1,
     *     -6,1,
     *     -7,1,
     *     -8,1,
     *      1,0,
     *      2,0,
     *     -9,1,
     *     -10,2,
     *     -11,2,
     *      3,0,
     *      4,0 ,
     *     -12,1,
     *     -13,1,
     *     -14,1,
     *     -15,1,
     *     -16,1 /
C
      DATA
     *     RZP5/-11/
     *    ,R1PZ/-12/
     *    ,R0/-1/
     *    ,R1/-2/
     *    ,R2/-3/
     *    ,SIGNEG/'00008000'X/
      DATA PSOP/4HOR  ,4HAND ,4HNOT ,4HTLT ,4HTGT ,4HADD ,4HSUB ,
     *          4HMUL ,4HDIV ,4HNEG ,4HRAI ,4HJMPF,4HJMP ,4HFETC,
     *          4HDEPO,4HFUNC,4HJMS ,4HRET ,4HSTOP,4HCALL,4HSCON,
     *          4HNOP ,4HNODE/
      DATA NPSOP/11*0,6*1,0,0,2,1,0,0/
C
      DATA MAXJMP/10/
```

```
      DATA MXTEMP/76/
      DATA MAXCOD/27000/
C
      DATA HLUN/4HLUNL,4HOG   /
     *     ,HSRC/4HLOGS,4HRC   /
     *     ,HPSE/4HLOGP,4HSE   /
     *     ,HINS/4HLOGI,4HNS   /
     *     ,HHEX/4HLOGH,4HEX   /
     *     ,HEXT/4HVCOD,4HE    /
C
      IF(NOSYST .OR. MESS.NE.0 .OR. NOCODE) RETURN
C
C        CHECK DEBUG OPTIONS
      CALL FINT(HLUN,HEXT,I1,IND)
      IF(IND .LE. 0) THEN
          LUNLOG=I1
      ELSE
          LUNLOG=LTO
      ENDIF
C
      CALL FINT(HSRC,HEXT,I1,IND)
      IF(IND .LE. 0) THEN
          LOGSRC=I1
      ELSE
          LOGSRC=0
      ENDIF
C
      CALL FINT(HPSE,HEXT,I1,IND)
      IF(IND .LE. 0) THEN
          LOGPSE=I1
      ELSE
          LOGPSE=0
      ENDIF
C
      CALL FINT(HINS,HEXT,I1,IND)
      IF(IND .LE. 0) THEN
          LOGINS=I1
      ELSE
          LOGINS=0
      ENDIF
C
      CALL FINT(HHEX,HEXT,I1,IND)
      IF(IND .LE. 0) THEN
          LOGHEX=I1
      ELSE
          LOGHEX=0
      ENDIF
C
C        COMPUTE VARIOUS ABS. ADDRESSES
      IBASE=IADDR(ICODE(1))-1
      IF(LOGHEX .NE. 0) WRITE(LUNLOG,4010)IBASE
 4010 FORMAT(' ***** BASE ADDRESS: ',Z8.8,' *****')
      INODAD=IADDR(INODE)
      ITMPAD=IADDR(ITEMP)
      IISYST=IADDR(ISYST)
      IIPART=IADDR(IPART)
      IISTOP=IADDR(ISTOP)
      ISIGNG=IADDR(SIGNEG)
      ILSAMP=IADDR(LSAMP)
      ILSMPS=IADDR(LSAMPS(1))-4
      ISYSTS=IADDR(SYSTS)
```

```
      IPOWRR=IADDR(OTS$POWRR)
      IFNADR(1)=IADDR(MTH$SQRT)
      IFNADR(2)=IADDR(MTH$EXP)
      IFNADR(3)=IADDR(MTH$ALOG)
      IFNADR(4)=IADDR(MTH$ALOG10)
      IFNADR(5)=IADDR(MTH$SIN)
      IFNADR(6)=IADDR(MTH$COS)
      IFNADR(7)=IADDR(MTH$TAN)
      IFNADR(8)=IADDR(MTH$ATAN)
      IFNADR(9)=IADDR(MTH$AINT)
      IFNADR(10)=IADDR(MTH$ATAN2)
      IFNADR(11)=IADDR(MTH$AMOD)
      IFNADR(12)=IADDR(MTH$ASIN)
      IFNADR(13)=IADDR(MTH$ACOS)
      IFNADR(14)=IADDR(MTH$SINH)
      IFNADR(15)=IADDR(MTH$COSH)
      IFNADR(16)=IADDR(MTH$TANH)
C
C        INITIALIZE POINTERS & FLAGS
C
      IPCNT=1
      ITOP=0
      IRGTOP=0
      JMPPNT=0
      ITMPNT=0
      MAXTMP=MXTEMP
      NXJPF=.FALSE.
C
C     SCAN THE FIVE PSEUDO-CODE LISTS
      DO 3000 ILIST=1,5
      IHEAD=IENT(ILIST)
      NODE=IHEAD
      LEN=0
C
C     REPLACE PSEUDO CODE ENTRY WITH ABS. CODE ENTRY ADDRESS
      IENT(ILIST)=IPCNT+IBASE
      IF(LOGPSE .NE. 0) WRITE(LUNLOG,4030)ILIST
      IF(LOGHEX .NE. 0) WRITE(LUNLOG,4040)IENT(ILIST)
 4030 FORMAT(1X/' * START OF LIST:',I2,' *')
 4040 FORMAT(' ***** STARTADDRESS: ',Z8.8,' *****')
      EOL=.FALSE.
C
C     GET NEXT PSEUDO INSTRUCTION WORD
C
 1000 IOP=NEXTOP(EOL)
      IF(EOL) GO TO 2500
C
 1010 IF(IOP.LT.1 .OR. IOP.GT.23) GO TO 903
C
      IF(LOGPSE .NE. 0) THEN
          IF(NPSOP(IOP) .EQ. 0) THEN
              WRITE(LUNLOG,4100)L,PSOP(IOP)
          ELSE
              J1=L+1
              J2=NPSOP(IOP)+L
              WRITE(LUNLOG,4100)L,PSOP(IOP),(IPSEUD(I),I=J1,J2)
          ENDIF
      ENDIF
 4100 FORMAT(1X,I4,10X,4(1H*),1X,A4,1X,I5,I4)
C
      IF(IPCNT.GT.MAXCOD) GO TO 920
```

```
C
C      CHECK IF THIS IS A FORWARD JUMP ADDR
C
 220   CONTINUE
       IF(JMPPNT.EQ.0) GO TO 250
       DO 230 I=1,JMPPNT
       IF(JMPTAB(1,I).EQ.L) GO TO 240
 230   CONTINUE
       GO TO 250
C
C      MAKE SURE THAT TOP STACK ELEMENT IS A REGISTER IF IT IS AN
C      ITYP=2 JUMP (JMP) BEFORE INSERTING FORWARD JUMP ADDRESS IN CODE
C
 240   IF(JMPTAB(3,I).EQ.2 .AND. ISTACK(ITOP).GE.0) THEN
           CALL UNSTK(IAD1)
           CALL GETREG(IREG)
           CALL VINST2(MOVF,IAD1,IREG)
           CALL STACK(IREG)
       ENDIF
C
C      INSERT CURRENT ADDR IN THE JUMP INSTR. TO THIS ADDRESS
C
       CALL VJUMP(JMPTAB(2,I))
C
C      REMOVE ADDRESS FROM JMP TABLE
C
       IF(I .LT. JMPPNT) THEN
           NR=JMPPNT-I
           DO 247 J=1,NR
           IJ=I+J
           DO 247 K=1,3
 247       JMPTAB(K,IJ-1)=JMPTAB(K,IJ)
       ENDIF
       JMPPNT=JMPPNT-1
C
C      CHECK IF THERE IS MORE THAN 1 JUMP TO THIS ADDRESS
C
       GO TO 220
C
C      JUMP TO CODE GENERATION SEQUENCE
C
 250   GO TO(2010,2020,2030,2040,2040,2060,2070,2060,2070,2100,2110,2120,
     1 2130,2140,2150,2160,2170,2180,2190,2200,2210,1000,2230),IOP
C
C      OR
C
 2010  CALL UNSTK2(IAD1,IAD2)
       IOP=NEXTOP(EOL)
       NXJPF=IOP.EQ.12
       IF(.NOT.NXJPF) CALL GETREG(IREG)
       CALL VINST2(CMPF,IAD1,RZP5)
       CALL VINST1(BGEQ,0)
       IFROM1=IPCNT
       CALL VINST2(CMPF,IAD2,RZP5)
       CALL VINST1(BGEQ,0)
       IFROM2=IPCNT
       IF(.NOT.NXJPF) THEN
           CALL VINST1(CLRF,IREG)
           CALL VINST1(BRB,3)
           CALL VBRANC(IFROM1,IPCNT)
           CALL VBRANC(IFROM2,IPCNT)
```

```
          CALL VINST2(MOVF,R1PZ,IREG)
          CALL STACK(IREG)
      ELSE
          CALL VBRANC(IFROM1,IPCNT+3)
          CALL VBRANC(IFROM2,IPCNT+3)
      ENDIF
      GO TO 1010
C
C     AND
C
 2020 CALL UNSTK2(IAD1,IAD2)
      IOP=NEXTOP(EOL)
      NXJPF=IOP .EQ. 12
      IF(.NOT.NXJPF) CALL GETREG(IREG)
      CALL VINST2(CMPF,IAD1,RZP5)
      CALL VINST1(BLSS,0)
      IFROM1=IPCNT
      CALL VINST2(CMPF,IAD2,RZP5)
      CALL VINST1(BLSS,0)
      IFROM2=IPCNT
      IF(NXJPF) THEN
          CALL VINST1(BRB,3)
          CALL VBRANC(IFROM1,IPCNT)
          CALL VBRANC(IFROM2,IPCNT)
      ELSE
          CALL VINST2(MOVF,R1PZ,IREG)
          CALL VINST1(BRB,2)
          CALL VBRANC(IFROM1,IPCNT)
          CALL VBRANC(IFROM2,IPCNT)
          CALL VINST1(CLRF,IREG)
          CALL STACK(IREG)
      ENDIF
      GO TO 1010
C
C     NOT
C
 2030 CALL UNSTK(IAD1)
      IOP=NEXTOP(EOL)
      NXJPF=IOP.EQ.12
      IF(.NOT.NXJPF) CALL GETREG(IREG)
      CALL VINST2(CMPF,IAD1,RZP5)
C
C       SPECIAL CODE IF NEXT OP. IS JMPF
      IF(NXJPF) THEN
          CALL VINST1(BLSS,3)
      ELSE
          CALL VINST1(BGEQ,5)
          CALL VINST2(MOVF,R1PZ,IREG)
          CALL VINST1(BRB,2)
          CALL VINST1(CLRF,IREG)
          CALL STACK(IREG)
      ENDIF
      GO TO 1010
C
C     ( , )
C
 2040 CALL UNSTK2(IAD1,IAD2)
      IF(IOP .EQ. 4) THEN
          IOPER=BLSS
      ELSE
          IOPER=BGTR
```

```
      ENDIF
      IOP=NEXTOP(EOL)
      NXJPF=IOP .EQ. 12
      IF(.NOT.NXJPF) CALL GETREG(IREG)
      CALL VINST2(CMPF,IAD2,IAD1)
C
C       SPECIAL CODE IF NEXT OP. IS JMPF
      IF(NXJPF) THEN
          CALL VINST1(IOPER,3)
      ELSE
          CALL VINST1(IOPER,4)
          CALL VINST1(CLRF,IREG)
          CALL VINST1(BRB,3)
          CALL VINST2(MOVF,R1PZ,IREG)
          CALL STACK(IREG)
      ENDIF
      GO TO 1010
C
C     + , *
C
 2060 CALL UNSTK2(IAD1,IAD2)
      IF(IAD2 .LT. 0) THEN
          IF(IOP .EQ. 6) THEN
              IOPER=ADDF2
          ELSE
              IOPER=MULF2
          ENDIF
          CALL VINST2(IOPER,IAD1,IAD2)
          CALL STACK(IAD2)
      ELSE IF(IAD1 .LT. 0) THEN
          IF(IOP .EQ. 6) THEN
              IOPER=ADDF2
          ELSE
              IOPER=MULF2
          ENDIF
          CALL VINST2(IOPER,IAD2,IAD1)
          CALL STACK(IAD1)
      ELSE
          CALL GETREG(IREG)
          IF(IOP .EQ. 6) THEN
              IOPER=ADDF3
          ELSE
              IOPER=MULF3
          ENDIF
          CALL VINST3(IOPER,IAD1,IAD2,IREG)
          CALL STACK(IREG)
      ENDIF
      GO TO 1000
C
C     - , /
C
 2070 CALL UNSTK2(IAD1,IAD2)
      IF(IAD2 .LT. 0) THEN
          IF(IOP .EQ. 7) THEN
              IOPER=SUBF2
          ELSE
              IOPER=DIVF2
          ENDIF
          CALL VINST2(IOPER,IAD1,IAD2)
          CALL STACK(IAD2)
      ELSE
```

```
          IF(IOP .EQ. 7) THEN
              IOPER=SUBF3
          ELSE
              IOPER=DIVF3
          ENDIF
          IF(IAD1 .LT. 0) THEN
              CALL VINST3(IOPER,IAD1,IAD2,IAD1)
              CALL STACK(IAD1)
          ELSE
              CALL GETREG(IREG)
              CALL VINST3(IOPER,IAD1,IAD2,IREG)
              CALL STACK(IREG)
          ENDIF
      ENDIF
      GO TO 1000
C
C     UNITARY -
C
 2100 CALL UNSTK(IAD1)
      CALL GETREG(IREG)
      CALL VINST2(MNEGF,IAD1,IREG)
      CALL STACK(IREG)
      GO TO 1000
C
C     & (A&B = A**B)
C
 2110 CALL UNSTK2(IAD1,IAD2)
      CALL VINST1(PUSHL,IAD1)
      CALL VINST1(PUSHL,IAD2)
      CALL VINST2(CALLS,2,IPOWRR)
      CALL GETREG(IREG)
      CALL VINST2(MOVF,R0,IREG)
      CALL STACK(IREG)
      GO TO 1000
C
C     JMPF
C
 2120 IF(NXJPF) GO TO 2130
      CALL UNSTK(IAD1)
      CALL VINST2(CMPF,IAD1,RZP5)
      CALL VINST1(BGEQ,3)
C
C     JMP
C
 2130 JMPADR=NEXTOP(EOL)
C
C     STORE FORWARD JUMP ADDR IN JMP TABLE
C
      JMPPNT=JMPPNT+1
      IF(JMPPNT.GT.MAXJMP) GO TO 906
      JMPTAB(1,JMPPNT)=JMPADR+L
      JMPTAB(2,JMPPNT)=IPCNT
      IF(IOP.EQ.12) JMPTAB(3,JMPPNT)=1
      IF(IOP.EQ.13) JMPTAB(3,JMPPNT)=2
C
C     MAKE SURE THAT TOP STACK ENTRY IS A REGISTER AND
C     THEN RELEASE IT IF OPERATION IS JMP
C
      IF(IOP .EQ. 13) THEN
          CALL UNSTK(IAD1)
          IF(IAD1 .GE. 0) THEN
```

```
               CALL GETREG(IREG)
               CALL VINST2(MOVF,IAD1,IREG)
               JMPTAB(2,JMPPNT)=IPCNT
               IRGTOP=IRGTOP-1
           ENDIF
       ENDIF
C
       CALL VINST1(BRW,0)
       NXJPF=.FALSE.
       GO TO 1000
C
C      FETCH (STACK OPERAND ADDRESS)
C
 2140  IAD1=NEXTOP(EOL)
       IF(IAD1 .LT. 10000) THEN
C
C          VARIABLE OPERAND
           IAD1=IPNTS(IAD1)
       ELSE
C
C          LITTERAL OPERAND
           IAD1=IADDR(VALUE(IAD1-10000))
       ENDIF
       CALL STACK(IAD1)
       GO TO 1000
C
C      DEPOS (=)
C
 2150  CALL UNSTK(IAD1)
       IF(ITOP.NE.0) GO TO 907
       IF(IRGTOP.NE.0) GO TO 908
       IF(IAD1.LT.0 .AND. IAD1.NE.R2) GO TO 909
       IOPAND=NEXTOP(EOL)
       IF(IOPAND.GT.10000) GO TO 910
       IOPAND=IPNTS(IOPAND)
       ITMPNT=0
C
       CALL VINST2(MOVF,IAD1,IOPAND)
       GO TO 1000
C
C      FUNC
C
 2160  IFUNC=NEXTOP(EOL)
C
 2162  IFENT=IFTAB(1,IFUNC)
       IF(IFENT.LE.0) GO TO 2165
C
C        INTRINSIC CODED FUNCTIONS
       GO TO(601,602,603,603),IFENT
C
C      ABS
C
 601   CALL UNSTK(IAD1)
       IF(IAD1 .LT. 0) THEN
           IREG=IAD1
           CALL VINST2(BICL2,ISIGNG,IREG)
       ELSE
           CALL GETREG(IREG)
           CALL VINST3(BICL3,ISIGNG,IAD1,IREG)
       ENDIF
       CALL STACK(IREG)
```

```
      GO TO 1000
C
C     SIGN
C
 602  CALL UNSTK(IAD1)
      CALL GETREG(IREG)
      CALL VINST1(TSTF,IAD1)
      CALL VINST1(BGTR,6)
      CALL VINST1(BLSS,9)
      CALL VINST1(CLRF,IREG)
      CALL VINST1(BRB,8)
      CALL VINST2(MOVF,R1PZ,IREG)
      CALL VINST1(BRB,3)
      CALL VINST2(MNEGF,R1PZ,IREG)
      CALL STACK(IREG)
      GO TO 1000
C
C     MIN , MAX
C
 603  CALL UNSTK2(IAD1,IAD2)
      IF(IFENT .EQ. 4) THEN
          IOPER=BLSS
      ELSE
          IOPER=BGTR
      ENDIF
      CALL GETREG(IREG)
      CALL VINST2(CMPF,IAD1,IAD2)
      CALL VINST1(IOPER,0)
      IFROM1=IPCNT
      CALL VINST2(MOVF,IAD1,IREG)
      CALL VINST1(BRB,0)
      IFROM2=IPCNT
      CALL VBRANC(IFROM1,IPCNT)
      CALL VINST2(MOVF,IAD2,IREG)
      CALL VBRANC(IFROM2,IPCNT)
      CALL STACK(IREG)
      GO TO 1000
C
C     LIBRARY FUNCTION
C
 2165 NARGS=IFTAB(2,IFUNC)
      DO 2166 I=1,NARGS
      CALL UNSTK(IAD)
      IF(IAD .LT. 0) THEN
C
C         STORE REGISTER OPERAND IN TEMP. VARIABLE
          IF(ITMPNT .GT. MAXTMP) GO TO 911
          IAD1=ITMPAD+ITMPNT
          ITMPNT=ITMPNT+4
          CALL VINST2(MOVF,IAD,IAD1)
          IAD=IAD1
      ENDIF
 2166 CALL VINST1(PUSHAL,IAD)
C
      CALL VINST2(CALLS,NARGS,IFNADR(-IFENT))
      CALL GETREG(IREG)
      CALL VINST2(MOVF,R0,IREG)
      CALL STACK(IREG)
      GO TO 1000
C
C     JMS
```

```
C
 2170 CONTINUE
      GO TO 903
C
C     RET
C
 2180 CONTINUE
      GO TO 903
C
C     STOP
C
 2190 CONTINUE
      GO TO 903
C
C     CALL
C
 2200 JSYST=NEXTOP(EOL)
      JPART=NEXTOP(EOL)
      CALL VINST2(MOVZWL,JSYST,IISYST)
      CALL VINST2(MOVZWL,JPART,IIPART)
      CALL VINST2(CALLS,0,ISYSTS)
C
C     NO NEED TO TEST FOR ISTOP IF END OF LIST
      IOP=NEXTOP(EOL)
      IF(EOL) GO TO 2500
      CALL VINST3(BBC,0,IISTOP,1)
      CALL VINST0(RSB)
      GO TO 1010
C
C     SCOND
C
 2210 ICOND=NEXTOP(EOL)
      IF(ICOND.EQ.0) GO TO 1000
C
C     CHECK NEXT NODE BEFORE GENERATING SCOND INSTR.
      NOD=NODE
 2212 NOD=IPSEUD(NOD)
      IF(NOD.EQ.IHEAD) GO TO 2214
C
C     INHIBIT SCOND IN NEXT NODE IF IT IS SAME SUB-SYSTEM
      IF(IPSEUD(NOD+6).NE.IOP) GO TO 2218
      IF(IPSEUD(NOD+7).NE.ICOND) GO TO 2218
C
C     REMOVE THIS SCOND
      IPSEUD(NOD+7)=0
      GO TO 2212
C
C     END OF LIST FOUND: SPECIAL JUMP ADDR
 2214 NOD=-6
C
 2218 CALL VINST3(BBC,0,ILSAMP,8)
      CALL VINST3(BBS,0,ILSMPS+4*ICOND,3)
      JMPPNT=JMPPNT+1
      IF(JMPPNT.GT.20) GO TO 906
      JMPTAB(1,JMPPNT)=NOD+6
      JMPTAB(2,JMPPNT)=IPCNT
      JMPTAB(3,JMPPNT)=3
      CALL VINST1(BRW,0)
      GO TO 1000
C
C       START OF NEW NODE
```

```
C
 2230 CALL VINST2(MOVZWL,NODE,INODAD)
      IF(LOGSRC .NE. 0) THEN
          ISYOLD=0
          WRITE(LUNLOG,4230)
          CALL PREQND(LUNLOG,NODE,ISYOLD)
      ENDIF
 4230 FORMAT(1X)
      GO TO 1000
C
C     END OF LIST
 2500 IF(JMPPNT.EQ.0) GO TO 2520
      IF(JMPPNT.NE.1) GO TO 901
      IF(JMPTAB(1,1).NE.0) GO TO 901
C
C     SCOND JUMP TO LAST INSTR.
      CALL VJUMP(JMPTAB(2,1))
      JMPPNT=0
C
 2520 CALL VINSTO(RSB)
C
 3000 CONTINUE
      RETURN
C
C     ERRORS
C
C     JMP TABLE NOT EMPTY
 901  IERR=1
      GO TO 999
C
C     BAD PSEUDO INSTRUCTION
 903  IERR=3
      GO TO 999
C
C     JMP TABLE OVERFLOW
 906  IERR=6
      GO TO 999
C
C     STACK NOT EMPTY
 907  IERR=7
      GO TO 999
C
C     REGISTER STILL ASSIGNED
 908  IERR=8
      GO TO 999
C
C     MORE THAN ONE REGISTER USED
 909  IERR=9
      GO TO 999
C
C     OPERAND IS A LITTERAL
 910  IERR=10
      GO TO 999
C
C     TEMP TABLE OVERFLOW
 911  IERR=11
      GO TO 999
C
C     INTERNAL ERROR IN VCODE
 912  IERR=12
      GO TO 999
```

```
C
C        NO MORE ROOM FOR ABS CODE
 920  WRITE(LTO,1920)
 1920 FORMAT(' NO MORE ROOM FOR ABS CODE')
      NOSYST=.TRUE.
      RETURN
C
 999  WRITE(LTO,1999)IERR
 1999 FORMAT(' **VCODE** : IERR=',I3)
      STOP
      END
```

```
        .TITLE CALCUL
;
; INTERFACE BETWEEN FORTRAN AND CODE GENERATED
; BY VCODE
;
;   AUTHOR: TOMMY ESSEBO 1980-11-11
;
        .PSECT ENTRY,PIC,OVR,REL,GBL,SHR,NOEXE,RD,WRT,LONG
ENTRY:  .LONG
        .PSECT $CODE,PIC,CON,REL,LCL,SHR,EXE,RD,NOWRT,LONG
CALCUL::
        .WORD ^M<IV,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
        PUSHAL  SIMHDL              ;
        CALLS   #1,LIB$ESTABLISH            ;
        MOVL    ENTRY,R0            ;
        MOVF    #0.5,R10            ;
        MOVF    #1.0,R11            ;
        JSB     (R0)                ;
        RET                         ;
        .END
```

```
      SUBROUTINE GETREG(IREG)
C
C     ASSIGNS A NEW REGISTER FROM REGISTER STACK
C
C     AUTHOR TOMMY ESSEBO 1980-11-11
C
C     SUBROUTINES REQUIRED
C         VINSTX
C
      INTEGER MOVF
C
      COMMON/UPNTS/IDUM(5),IRGTOP,ITOP
      COMMON/UTEMPC/ITMPNT,MAXTMP,ITMPAD,ITEMP(20)
      COMMON/UDATA/ISTACK(25),IDUM1(5000)
      COMMON/DEVICE/LKB,LTP,LLP,LDIS,LTO,IDUM2(6)
C
      DATA MAXREG/7/
      DATA MOVF/'00000050'X/
C
      IF(IRGTOP.GE.MAXREG) GO TO 20
C
C     ASSIGN REGISTER
C
      IREG=-3-IRGTOP
      IRGTOP=IRGTOP+1
      RETURN
C
C     NO REGISTER IN REGISTER STACK
C     LOOK FOR REGISTER IN OPERAND STACK
 20   IF(ITOP .EQ. 0) GO TO 91
      DO 30 I=1,ITOP
      J=ITOP+1-I
      IF(ISTACK(J) .LT. 0) GO TO 40
 30   CONTINUE
      GO TO 91
C
C       MOVE OPERAND FROM REGISTER TO TEMP. VARIABLE
 40   IF(ITMPNT .GT. MAXTMP) GO TO 92
      IAD1=ITMPAD+ITMPNT
      ITMPNT=ITMPNT+4
      IREG=ISTACK(J)
      CALL VINST2(MOVF,IREG,IAD1)
      ISTACK(J)=IAD1
C
C       CHECK THAT THIS IS LAST REGISTER
      IF(IREG .NE. -2-MAXREG) GO TO 93
      RETURN
C
 91   WRITE(LTO,191)
 191  FORMAT(' **VCODE** NO REGISTERS IN OPERAND STACK')
      STOP
C
 92   WRITE(LTO,192)
 192  FORMAT(' **VCODE** TEMP TABLE OVERFLOW')
      STOP
C
 93   WRITE(LTO,193)
 193  FORMAT(' **VCODE** REGISTER ALLOCATION ERROR')
      STOP
      END
```

```
      INTEGER FUNCTION NEXTOP(EOL)
C
C     RETURNS NEXT PSEUDO CODE OPERATION FROM /PSCODE/
C
C     EOL - RETURNED .TRUE. IF END OF LIST
C
C     AUTHOR TOMMY ESSEBO 1980-11-11
C
C     SUBROUTINES REQUIRED
C         NONE
C
      LOGICAL EOL
      COMMON/PSCODE/IPSEUD(100)
      COMMON/UPNTS/L,IHEAD,NODE,LEN,IDUM(3)
C
      IF(LEN.GT.0) GO TO 20
C
C       NEW NODE
 10   NODE=IPSEUD(NODE)
      IF(NODE.EQ.IHEAD) GO TO 50
      L=NODE+5
      LEN=IPSEUD(NODE+2)
      IF(LEN.LE.0) GO TO 10
C
C       INDICATE NEW NODE FOR CODE GENERATOR
      NEXTOP=23
      RETURN
C
 20   L=L+1
      LEN=LEN-1
      NEXTOP=IPSEUD(L)
      RETURN
C
C       END OF LIST
 50   EOL=.TRUE.
      RETURN
      END
```

```
      SUBROUTINE UNSTK(IOPAND)
C
C     UNSTACKS TOP ITEM FROM OPERAND ADDRESS STACK AND FREES IT
C     IF IT IS A REGISTER
C
C     AUTHOR TOMMY ESSEBO 1980-11-11
C
C     SUBROUTINES REQUIRED
C         NONE
C
      DIMENSION IA(2)
C
      COMMON/DEVICE/LKB,LTP,LLP,LDIS,LTO,LPLOT,LXXX,LDK1,LDK2,LDK3,LDK4
      COMMON/UPNTS/IDUM(5),IRGTOP,ITOP
      COMMON/UDATA/ISTACK(25),IDUM3(5000)
C
      DATA MAXSTK/25/
C
      N=1
      GO TO 100
C
C
      ENTRY UNSTK2(IOPND1,IOPND2)
C
C     SAME AS UNSTK BUT FOR 2 ITEMS
C     (IOPND1 IS TOP ELEMENT)
C
      N=2
C
 100  DO 200 I=1,N
      IF(ITOP.LE.0) GO TO 91
      IA(I)=ISTACK(ITOP)
      ITOP=ITOP-1
      IF(IA(I).LT.0) THEN
          IR=-3-IA(I)
          IF(IR .NE. IRGTOP-1) GO TO 93
          IRGTOP=IR
      ENDIF
 200  CONTINUE
      GO TO (210,220),N
C
 210  IOPAND=IA(1)
      RETURN
C
 220  IOPND1=IA(1)
      IOPND2=IA(2)
      RETURN
C
C
      ENTRY STACK(IOPND)
C
C       STACKS OPERAND ADDRESS IOPND AND MAKES SURE THAT IF
C       IT IS A REGISTER THE REGISTER STACK IS UPDATED
C
      IF(ITOP.GE.MAXSTK) GO TO 92
      ITOP=ITOP+1
      ISTACK(ITOP)=IOPND
C
      IF(IOPND .LT. 0) THEN
          IR=-3-IOPND
          IF(IR.NE.IRGTOP .AND. IR.NE.IRGTOP-1) GO TO 93
```

```
            IRGTOP=IR+1
        ENDIF
        RETURN
C
C
 91     WRITE(LTO,191)
 191    FORMAT(' **VCODE** UNSTK:OPERAND STACK UNDERFLOW')
        STOP
C
 92     WRITE(LTO,192)
 192    FORMAT(' **VCODE** STACK: OPERAND STACK OVERFLOW')
        STOP
C
 93     WRITE(LTO,193)
 193    FORMAT(' **VCODE** STACK: REGISTER ASSIGNMENT ERROR')
        STOP
C
        END
```

```
      SUBROUTINE VBYTE(IBYTE)
C
C        WRITES BYTE IBYTE IN NEXT FREE BYTE IN ABS CODE
C
C        AUTHOR: TOMMY ESSEBO 1980-11-11
C
      BYTE IBYTE,ICODE,BYTES(4)
C
      COMMON/UDATA/IDUM1(25),ICODE(20000)
      COMMON/UPNTS/IDUM2(4),IPCNT,IDUM3(2)
      COMMON/VXCLOG/LUNLOG,LOGSRC,LOGPSE,LOGINS,LOGHEX
C
      EQUIVALENCE (IBYTES,BYTES(1))
C
      IF(LOGHEX .NE. 0) WRITE(LUNLOG,1000)IPCNT,IBYTE
 1000 FORMAT(13X,Z4.4,2X,Z2.2)
      ICODE(IPCNT)=IBYTE
      IPCNT=IPCNT+1
      RETURN
C
      ENTRY VWORD(IWORD)
C
C        SAME FOR WORD (2 BYTES) IWORD
C
      IF(LOGHEX .NE. 0) WRITE(LUNLOG,1100)IPCNT,IWORD
 1100 FORMAT(13X,Z4.4,2X,Z4.4)
      IBYTES=IWORD
      DO 20 I=1,2
      ICODE(IPCNT)=BYTES(I)
 20   IPCNT=IPCNT+1
      RETURN
C
      ENTRY VLONGW(ILONGW)
C
C        SAME FOR LONGWORD (4 BYTES) ILONGW
C
      IF(LOGHEX .NE. 0) WRITE(LUNLOG,1200)IPCNT,ILONGW
 1200 FORMAT(13X,Z4.4,2X,Z8.8)
      IBYTES=ILONGW
      DO 40 I=1,4
      ICODE(IPCNT)=BYTES(I)
 40   IPCNT=IPCNT+1
      RETURN
C
      END
```

```
C         FILE: VINSTX.FOR
C
C         GENERATES MACHINE CODE FOR VAX-11
C
C         AUTHOR: TOMMY ESSEBO 1980-11-11
C
C         SUBROUTINES REQUIRED
C             VBYTE VWORD VLONGW
C             PSPACE PHOLL PINT
C
      SUBROUTINE VINSTO(OP1)
C
      BYTE BYTADR(4),ICODE
C
      INTEGER OPTAB(5,27),BUFF(14),OP,OP1,OP2,OP3,OP4
C
      DIMENSION INTOPA(3)
C
      COMMON/UDATA/IDUM1(25),ICODE(20000)
      COMMON/UPNTS/L,IDUM2(3),IPCNT,IDUM3(2)
      COMMON/UTEMPC/IDUM4(2),ITMPAD
      COMMON/VXCLOG/LUNLOG,LOGSRC,LOGPSE,LOGINS,LOGHEX
C
      EQUIVALENCE (INTADR,BYTADR(1))
C
      DATA OPTAB /
C
C     OPCODE    NR OF OPERANDS    CASE   MNEMONIC
C
     *     '00000050'X , 2 , 1 , 4HMOVF,4H     ,
     *     '000000DD'X , 1 , 1 , 4HPUSH,4HL    ,
     *     '000000D4'X , 1 , 1 , 4HCLRF,4H     ,
     *     '00000052'X , 2 , 1 , 4HMNEG,4HF    ,
     *     '0000003C'X , 2 , 7 , 4HMOVZ,4HWL   ,
     *     '00000051'X , 2 , 1 , 4HCMPF,4H     ,
     *     '00000053'X , 1 , 1 , 4HTSTF,4H     ,
     *     '00000040'X , 2 , 1 , 4HADDF,4H2    ,
     *     '00000041'X , 3 , 1 , 4HADDF,4H3    ,
     *     '00000042'X , 2 , 1 , 4HSUBF,4H2    ,
     *     '00000043'X , 3 , 1 , 4HSUBF,4H3    ,
     *     '00000044'X , 2 , 1 , 4HMULF,4H2    ,
     *     '00000045'X , 3 , 1 , 4HMULF,4H3    ,
     *     '00000046'X , 2 , 1 , 4HDIVF,4H2    ,
     *     '00000047'X , 3 , 1 , 4HDIVF,4H3    ,
     *     '000000CA'X , 2 , 1 , 4HBICL,4H2    ,
     *     '000000CB'X , 3 , 1 , 4HBICL,4H3    ,
     *     '000000DF'X , 1 , 6 , 4HPUSH,4HAL   ,
     *     '00000014'X , 1 , 2 , 4HBGTR,4H     ,
     *     '00000018'X , 1 , 2 , 4HBGEQ,4H     ,
     *     '00000019'X , 1 , 2 , 4HBLSS,4H     ,
     *     '00000011'X , 1 , 2 , 4HBRB ,4H     ,
     *     '00000031'X , 1 , 3 , 4HBRW ,4H     ,
     *     '000000E1'X , 3 , 8 , 4HBBC ,4H     ,
     *     '000000E0'X , 3 , 8 , 4HBBS ,4H     ,
     *     '00000005'X , 0 , 4 , 4HRSB ,4H     ,
     *     '000000FB'X , 2 , 5 , 4HCALL,4HS    /
C
      DATA NROP/27/
C
      NADR=0
      OP=OP1
```

```
      GO TO 100
C
      ENTRY VINST1(OP2,IAD1)
      NADR=1
      INTOPA(1)=IAD1
      OP=OP2
      GO TO 100
C
      ENTRY VINST2(OP3,IAD2,IAD3)
      NADR=2
      INTOPA(1)=IAD2
      INTOPA(2)=IAD3
      OP=OP3
      GO TO 100
C
      ENTRY VINST3(OP4,IAD4,IAD5,IAD6)
      NADR=3
      INTOPA(1)=IAD4
      INTOPA(2)=IAD5
      INTOPA(3)=IAD6
      OP=OP4
      GO TO 100
C
      ENTRY VJUMP(JADR)
C
C       INSERTS CURRENT ADDR AT ADDRESS JADR IN ABS. CODE
C       (BRW INSTRUCTION JUMPS ONLY)
C
      INTADR=IPCNT-JADR-3
      IF(LOGINS .NE. 0) WRITE(LUNLOG,1000)IPCNT,JADR
 1000 FORMAT(6X,I5,8X,'JUMP FROM:',I5)
      IF(LOGHEX .NE. 0) THEN
          JADR1=JADR+1
          WRITE(LUNLOG,1100)JADR1,INTADR
 1100 FORMAT(13X,Z4.4,2X,Z4.4)
      ENDIF
      DO 40 I=1,2
 40   ICODE(JADR+I)=BYTADR(I)
      RETURN
C
      ENTRY VBRANC(IFROM,ITO)
C
C       INSERTS BYTE ADDDRESS IN CODE
C
C       ITO-IFROM IS INSERTED AT BYTE IFROM-1 IN ICODE
C
      INTADR=ITO-IFROM
      ICODE(IFROM-1)=BYTADR(1)
C
      IF(LOGINS .NE. 0) THEN
          IBR=IFROM-2
          WRITE(LUNLOG,1200)ITO,IBR
      ENDIF
 1200 FORMAT(6X,I5,8X,'BRANCH FROM:',I5)
      IF(LOGHEX .NE. 0) THEN
          IBR=IFROM-1
          WRITE(LUNLOG,1300)IBR,INTADR
      ENDIF
 1300 FORMAT(13X,Z4.4,2X,Z2.2)
      RETURN
C
```

```
C       ----------------------------------------------------------
C
C      FIND OPERATION
 100   DO 120 IOP=1,NROP
       IF(OP .EQ. OPTAB(1,IOP)) GO TO 140
 120   CONTINUE
       GO TO 901
C
C        CHECK NR OF OPERANDS
 140   IF(NADR .NE. OPTAB(2,IOP)) GO TO 902
       IF(LOGINS .NE. 0) THEN
           IP=1
           CALL PSPACE(IP,BUFF,56)
           IP=7
           CALL PINT(IP,BUFF,IPCNT)
           IP=19
           CALL PHOLL(IP,BUFF,OPTAB(4,IOP),8)
           IF(NADR .EQ. 0) GO TO 190
           DO 180 IOPAND=1,NADR
           IP=18+9*IOPAND
           IAD=INTOPA(IOPAND)
           IF(IAD .LT. 0) THEN
C
C               REGISTER OPERAND
               CALL PHOLL(IP,BUFF,4HR    ,4)
               CALL PINT(IP,BUFF,-1-IAD)
           ELSE
               DO 150 I=1,20
               IF(ITMPAD+(I-1)*4 .EQ. IAD) GO TO 160
 150           CONTINUE
C
               CALL PINT(IP,BUFF,IAD)
               GO TO 170
C
C               TEMP. VAR OPERAND
 160           CALL PHOLL(IP,BUFF,4HTMP$,4)
               CALL PINT(IP,BUFF,I)
C
 170           CONTINUE
           ENDIF
 180       CONTINUE
 190       WRITE(LUNLOG,1400)BUFF
 1400  FORMAT(1X,14A4)
       ENDIF
C
C        WRITE OPCODE
       CALL VBYTE(OPTAB(1,IOP))
C
C        EVALUATE OPERANDS
       GO TO(200,250,300,350,400,450,500,550),OPTAB(3,IOP)
C
C        CASE 1: ABSOLUTE OR REGISTER MODE OPERANDS
 200   DO 220 I=1,NADR
       IF(INTOPA(I) .LT. 0) THEN
C
C        REGISTER ADDRESS
         CALL VBYTE(79-INTOPA(I))
       ELSE
C
C        ABS. ADDRESS
         CALL VBYTE(159)
```

```
            CALL VLONGW(INTOPA(I))
        ENDIF
 220    CONTINUE
        RETURN
C
C         CASE 2: BYTE BRANCH
 250    CALL VBYTE(INTOPA(1))
        RETURN
C
C         CASE 3: WORD BRANCH
 300    IF(INTOPA(1) .NE. 0) GO TO 903
        CALL VWORD(INTOPA(1))
        RETURN
C
C         CASE 4: NO OPERAND
 350    RETURN
C
C         CASE 5: CALLS
 400    CALL VBYTE(INTOPA(1))
        CALL VBYTE(159)
        CALL VLONGW(INTOPA(2))
        RETURN
C
C         CASE 6: PUSHAL
 450    CALL VBYTE(159)
        CALL VLONGW(INTOPA(1))
        RETURN
C
C         CASE 7: MOVZWL
 500    CALL VBYTE(143)
        CALL VWORD(INTOPA(1))
        CALL VBYTE(159)
        CALL VLONGW(INTOPA(2))
        RETURN
C
C         CASE 8: BBC,BBS
 550    CALL VBYTE(0)
        CALL VBYTE(159)
        CALL VLONGW(INTOPA(2))
        CALL VBYTE(INTOPA(3))
        RETURN
C
C         ERRORS
C
C         ILLEGAL OPCODE
 901    IERR=1
        GO TO 999
C
C         ILLEGAL NR OF OPERANDS
 902    IERR=2
        GO TO 999
C
C         ILLEGAL BRANCH ADDRESS
 903    IERR=3
        GO TO 999
C
 999    WRITE(6,1999),IERR
 1999   FORMAT(' VINSTX ERROR:',I2)
        RETURN
C
        END
```