# I-code Working-Notes 2v0

Peter S. Robertson

1 October 1984

Updated by

John D. McMullin

30th September 2023

Note:  This document is not intended to be a complete formal description of I-code

Lattice Logic Limited
9, Wemyss Place
Edinburgh EH3 6DH
Scotland

# Contents

# 1. Philosophy

I-code is an intermediate code used to provide an interface between the machine-independent and machine-dependent sections of a compiler.

Most intermediate codes in use today describe the execution of an abstract machine which performs the desired computation. For example, the intermediate code for a statement of the form:  X = Y+Z would describe the operations of the abstract machine which would compute the value of Y+Z and assign it to X. Using this sort of code, the machine-dependent section of the compiler then maps the abstract machine onto the real target machine.

I-code uses a fundamentally different model. It describes the execution of an abstract compiler which generates target code to perform the desired computation; it does not describe an abstract machine which will perform that computation. It is vital to understand that it does not describe the function of the program directly but describes it indirectly via the abstract compiler. It is this indirection which gives I-code the power to be machine-independent without sacrificing efficiency in the executable programs which it can be used to generate.

There are two important corollaries of this. Firstly, the structures and operations associated with the abstract compiler need have no counterparts in the object program. For example, the target machine need have no hardware or software stack and neither need it have a true condition code. Secondly the code assumes that the operations it describes will be performed by the abstract compiler in the order specified with no omissions. In particular, the control transfer instructions do not transfer control in the abstract compiler but indicate changes of control flow in the program which is being compiled. This also does not mean that any of the operations need have counterparts in the object program, nor that the order of the generated code corresponds to the order of the I-code.

For example, the Algol-60 statement: A := if B then C else  D;  could not be encoded in the seemingly obvious way:

```
Stack A
Stack B; Test-Boolean; BF 1
Stack C; Forward 2
Label 1
Stack D
Label 2
Assign-Value
```

as this would assign D to C, the last two objects stacked prior to the Assign-Value instruction and leave A on the stack.

Encode instead as:

```
Stack B; Test-Boolean; BF 1
Stack A; Stack C; Assign-Value; Forward 2
Label 1
Stack A; Stack D; Assign-Value
Label 2
```

This is similar to that generated from the code fragment: if B then A := C else  A := D;

N.B. Stack has the same usage as PUSH.

## 2. Definitions

The ICODE instructions and parameters are stored as encoded data in a binary file using the following definitions.

| | |
|---|---|
| byte-order | All multi-byte values are specified with the least-significant byte first. |
| Unsigned | A natural binary number. |
| Signed | A 2's complement binary number. |
| <b> | A one-byte unsigned number. |
| <integer> | A four-byte signed integer number. |
| <label> | An unsigned 16-bit value used to identify a simple label. |
| <n> | An unsigned 16-bit number. |
| <string> | A byte-counted string constant. |
| <real> | A real constant in a textual encoding. |
| <tag> | An unsigned 16-bit value used to identify a tag (descriptor). |
| condition-code | A conceptual flag which is set at run-time by the instructions listed in Appendix 2.<br><br>This flag need not exist in the target machine but is defined in order to simplify the definitions of certain instructions.<br><br>The values which this flag may take are: equal, less than, greater than, true, false<br>This setting only remains valid for the duration of the next instruction which must be one of the instructions listed in Appendix 3. |
| Integer | A general integer value, including subranges of integers. |
| Labels | I-code distinguishes two sorts of label.<br><br>1 – Simple Labels: have the property that they are only jumped to in one direction, that is, all references to an instance of a simple label are either all forward or all backward. This means that the same denotation may be used for many simple labels. For example, the following is valid:<br>**Goto** 1 >-----+<br>... \|<br>**Label** 1 <-----+<br>...<br>**Goto** 1 >-----+<br>... \|<br>**Label** 1 <-----+<br>...<br>**Label** 1 <-----+<br>... \|<br>**Repeat** 1 >-----+<br>All uses of a particular simple label must be in the same block as the definition of that label.<br>Simple labels are encoded as two-byte unsigned integers although code generators may assume that their values are within a fairly small range.<br>(1 .. 50 is common.)<br><br>2. General Labels - have none of the restrictions of simple labels. They are identified by tags and will be defined automatically if necessary when they are first used. General labels are referenced by the instructions: |

| | |
|---|---|
| | **Jump** <tag><br>**Locate** <tag><br>**Def** <tag> |
| Real | A floating-point value, either single or double precision. |
| SOS | The second-top item on the stack. |
| Stack | A first-in, last-out structure used to imply the operands required by various I-code instructions.<br>The first item which can be removed from the stack is called TOS and the item which can be removed after TOS is called SOS. |
| Tags | Tags are definitions of objects which are to be manipulated by the compiler. These definitions are created in a nested fashion; all tags defined in a block are deleted when the end of that block is reached.<br>On definition the machine-independent description of the object is converted into the appropriate machine-dependent description of the actual object to be used. Within this document the term 'tag' is used to describe both this descriptor and the unsigned sixteen-bit integer used as an index value to select it from the collection of all tags.<br>Apart from the resolution of forward references to procedures, tags are never altered.<br>When a copy of a tag value is pushed onto the stack the value becomes known as a descriptor. Descriptors may be modified. |
| tag list | A tag list is an ordered sequence of tag definitions used to describe either the parameters required by a procedure or the fields of a record. Components of a tag list are referred to either explicitly by their position in the list (see SELECT) or implicitly by sequence starting with the first to be specified (see Assign-Parameter). |
| TOS | The top item on the stack. |
| list flag | An internal flag which is set during the processing of explicit lists of tag definitions. Its only purpose is to prevent certain nested list structures. |

# 3. Conventions

1. It is assumed that the reader is familiar with the IMP language and its terminology.

2. Whenever a pointer-variable is used in the context of a value the value of the data item pointed to will be used.

3. In general, diagnostic checks are implied rather than explicitly specified.

4. Items on the stack are intended to be 'rules for generating' values or references rather than the values or references themselves. For simplicity the descriptions of instructions will often refer to items as if they contain values or references.

5. By convention tag index values will often be replaced in examples by the identifier which is assumed to have been associated with the tag in question.

   For example, given 'DEFINE 57,Fred......' then 'Stack 57' could be written 'Stack Fred'
   PUSH is equivalent to Stack

6. The term 'error' is used to indicate a condition discovered by the code-generator which should terminate the compilation with a suitable error message.

# 4. ICODE INSTRUCTIONS as ASCII

| *ASCII Value* | *ASCII Character* | *ICODE INSTRUCTION* | *ICODE PARAMETERS* |
|---|---|---|---|
| 0..9 | <NUL>..<HT> | unused | |
| 10 | <LF> | EOF | |
| 11..31 | <VT>..<US> | unused | |
| 32 | <Space> | unused | |
| 33 | ! | OR | |
| 34 | ” | COMPARED | |
| 35 | # | JNE | Tag |
| 36 | $ | DEF | Tag String ‘,’ Tag ‘,’ Tag ‘,’ Tag |
| 37 | % | XOR | |
| 38 | & | AND | |
| 39 | ’ | PUSHS | String |
| 40 | ( | JLE | Tag |
| 41 | ) | JGE | Tag |
| 42 | * | MUL | |
| 43 | + | ADD | |
| 44 | , | parameter separator | |
| 45 | - | SUB | |
| 46 | . | CONCAT | |
| 47 | / | QUOT | |
| 48..57 | 0..9 | unused | |
| 58 | : | LOCATE | Tag |
| 59 | ; | END | |
| 60 | < | JL | Tag |
| 61 | = | JE | Tag |
| 62 | > | JG | Tag |
| 63 | ? | COMPARE | |
| 64 | @ | PUSH | Tag |
| 65 | A | INIT | Tag |
| 66 | B | REPEAT | Tag |
| 67 | C | COMPAREA | |
| 68 | D | PUSHR | Real |
| 69 | E | CALL | |
| 70 | F | GOTO | Tag |
| 71 | G | ALIAS | String |
| 72 | H | BEGIN | |
| 73 | I | unused | |
| 74 | J | JUMP | Tag |
| 75 | K | FALSE | |
| 76 | L | LABEL | Tag |
| 77 | M | MAP | |
| 78 | N | PUSHI | Integer |
| 79 | O | LINE | Tag |
| 80 | P | PLANT | Byte |
| 81 | Q | DIVIDE | |
| 82 | R | RETURN | |
| 83 | S | ASSVAL | |
| 84 | T | TRUE | |

| | | | |
|---|---|---|---|
| 85 | U | NEGATE | |
| 86 | V | RESULT | |
| 87 | W | SJUMP | Tag |
| 88 | X | IEXP | |
| 89 | Y | unused | |
| 90 | Z | ASSREF | |
| 91 | [ | LSH | |
| 92 | \ | NOT | |
| 93 | ] | RSH | |
| 94 | ^ | SETFORMAT | Tag |
| 95 | _ | SLABEL | Tag |
| 96 | ` | unused | |
| 97 | a | ACCESS | |
| 98 | b | BOUNDS | |
| 99 | c | unused | |
| 100 | d | DIM | Tag ‘,’ Tag |
| 101 | e | EVENT | Tag |
| 102 | f | FOR | Tag |
| 103 | g | unused | |
| 104 | h | unused | |
| 105 | i | INDEX | |
| 106 | j | JAM | |
| 107 | k | JZ | Tag |
| 108 | l | LANG | Tag |
| 109 | m | MONITOR | |
| 110 | n | SELECT | Tag |
| 111 | o | ON | Tag ‘,’ Tag |
| 112 | p | ASSPAR | |
| 113 | q | SUBA | |
| 114 | r | RESOLVE | Tag |
| 115 | s | STOP | |
| 116 | t | JNZ | Tag |
| 117 | u | ADDA | |
| 118 | v | MOD | |
| 119 | w | MCODE | String ‘;’ |
| 120 | x | REXP | |
| 121 | y | DIAG | Tag |
| 122 | z | CONTROL | Tag |
| 123 | { | START | |
| 124 | ¦ | unused | |
| 125 | } | FINISH | |
| 126 | ~ | ALT | Byte |
| 127..255 | <DEL>.. | unused | |

All other byte values (in the range 0..255) are currently ILLEGAL/UNUSED.

Extra ICODE instructions can be added by using unused ASCII characters with parameters as needed.

# 5. ICODE GROUPS

## 5.1. Stack Operations

These ICODE instructions represent operations placed on an operation stack for code generation on the IMP variables.

| | | | |
|---|---|---|---|
| Instruction: | ADD | ‘+’ | Load Operation( ADDx ) |
| Effect: | TOS is replaced by the sum of values of TOS, SOS.<br>Integer values will be converted into floating-point if one operand is integer and the other is real. | | |
| Notes: | | | |
| Error: | 1. The stack is empty.<br>2. TOS,SOS are not integer or real. | | |
| Example: | X = Y+Z | PUSH X<br>PUSH Y<br>PUSH Z<br>ADD<br>ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | ADDA | ‘u’ | Load Double( ADDx ) |
| Effect: | TOS is replaced by the sum of values of TOS, SOS. | | |
| Notes: | Used to add two address values/two double real values | | |
| Error: | 1. The stack is empty.<br>2. TOS,SOS are not integer or real. | | |
| Example: | X = Y+Z | PUSH X<br>PUSH Y<br>PUSH Z<br>ADDA<br>ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | AND | ‘&’ | Load Operation( ANDx ) |
| Effect: | TOS,SOS are replaced by the logical and of the values of TOS, SOS. | | |
| Notes: | Used to AND two logical values | | |
| Error: | 1. The stack has less than 2 items.<br>2. TOS,SOS are not integer values. | | |
| Example: | X = Y & Z | PUSH X<br>PUSH Y<br>PUSH Z<br>AND<br>ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | CONCAT | ‘.’ | Load Operation( CONCx ) |
| Effect: | TOS and SOS are removed from the stack and the string concatenation of their values, SOS.TOS, is stacked. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two values.<br>2. TOS is not a string value.<br>3. SOS is not a string value. | | |
| Example: | S = T.U.V | PUSH S<br>PUSH T<br>PUSH U<br>CONCAT<br>PUSH V<br>CONCAT<br>ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | DIVIDE | ‘Q’ | Load Operation( RDIVx ) |
| Effect: | TOS and SOS are removed from the stack and the real quotient, SOS / TOS, is stacked. Integer values will be converted into floating-point before the division is attempted. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is neither integer nor real type.<br>3. SOS is neither integer nor real type. | | |
| Example: | A = B / C | PUSH A<br>PUSH B<br>PUSH C<br>DIV<br>ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | IEXP | ‘X’ | Load Operation( EXPx ) |
| Effect: | TOS and SOS are removed from the stack and the value of SOS raised to the integer power of the value of TOS, SOS^^TOS, is stacked. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is not an integer value.<br>3. SOS is not an integer value. | | |
| Example: | J = K^^3 | PUSH J<br>PUSH K<br>PUSHI 3<br>IEXP<br>ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | LSH | ‘[‘ | Load Operation( LSHx ) |
| Effect: | TOS and SOS are removed from the stack and the value of SOS logically shifted left by the value of TOS, SOS << TOS, is stacked. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is not an integer value.<br>3. SOS is not an integer value.<br>4. The value of TOS is negative or greater than or equal to the number of bits in an integer. | | |
| Example: | A = B << C | PUSH A<br>PUSH B<br>PUSH C<br>LSH<br>ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | MOD | ‘v’ | Load Operation( ABSx ) |
| Effect: | TOS and SOS are removed from the stack and are replaced by the value 'SOS MOD TOS' where MOD is as defined in section 6.7.2.2 of the Pascal standard BS 6192:1982. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is not an integer value.<br>3. SOS is not an integer value. | | |
| Example: | m := p MOD q; | PUSH m<br>PUSH p<br>PUSH q<br>MOD<br>ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | MUL | ‘*’ | Load Operation( MULx ) |
| Effect: | TOS and SOS are removed from the stack and the value of SOS multiplied by the value of TOS, SOS * TOS, is stacked. Integer values will be converted into floating-point if one operand is integer and the other is real. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is neither integer nor real type.<br>3. SOS is neither integer nor real type. | | |
| Example: | A = B * C | PUSH A<br>PUSH B<br>PUSH C<br>MUL<br>ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | NEGATE | ‘U’ | Load Operation(NEGx) |
| Effect: | The value in TOS is negated and left as TOS. | | |
| Notes: | | | |
| Error: | 1. The stack is empty.<br>2. TOS is not an integer or real value. | | |
| Example: | A = -B | PUSH A<br>PUSH B<br>NEGATE<br>ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | NOT | ‘\’ | Load Operation( NOTx ) |
| Effect: | The value in TOS is logically inverted and left on TOS | | |
| Notes: | | | |
| Error: | 1. The stack is empty.<br>2. TOS is not an integer or real value. | | |
| Example: | | | |

| | | | |
|---|---|---|---|
| Instruction: | OR | ‘!’ | Load Operation( ORx ) |
| Effect: | TOS and SOS are removed from the stack and the value of SOS logically ORed with the value of TOS, SOS ! TOS, is stacked. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is not an integer value.<br>3. SOS is not an integer value. | | |
| Example: | A = B ! C | PUSH A<br>PUSH B<br>PUSH C<br>OR<br>ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | QUOT | ‘/’ | Load Operation( DIVx ) |
| Effect: | TOS and SOS are removed from the stack and the value of SOS integer-divided by the value of TOS, SOS // TOS, is stacked. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is not an integer.<br>3. SOS is not an integer. | | |
| Example: | A = B // C | PUSH A<br>PUSH B<br>PUSH C<br>QUOT<br>ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | REXP | ‘x’ | Load Operation(REXPx) |
| Effect: | TOS and SOS are removed from the stack and the value of SOS raised to the integer power TOS, SOS^TOS, is stacked. The type of this value is real. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two items.<br>2. SOS is neither an integer nor a real value.<br>3. TOS is not an integer value. | | |
| Example: | R = 12^X | PUSH R<br>PUSHI 12; PUSH X; REXP;<br>ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | RSH | ‘]’ | Load Operation( RSHx ) |
| Effect: | TOS and SOS are removed from the stack and the value of SOS logically left shifted by the value of TOS, SOS >> TOS, is stacked. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is not an integer value.<br>3. SOS is not an integer value.<br>4. The value of TOS is negative or greater than or equal to the number of bits in an integer. | | |
| Example: | A = B >> C | PUSH A<br>PUSH B; PUSH C; RSH<br>ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | SUB | ‘-‘ | Load Operation( SUBx ) |
| Effect: | TOS and SOS are removed from the stack and the value of SOS minus the value of TOS, SOS - TOS, is stacked.<br>Integer values will be converted into floating-point if one operand is integer and the other is real. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is neither integer nor real type.<br>3. SOS is neither integer nor real type. | | |
| Example: | A = B - C | PUSH A<br>PUSH B; PUSH C; SUB<br>ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | SUBA | ‘q‘ | Load DoubleOp( SUBx) |
| Effect: | Similar to Sub except generated code is for addresses | | |
| Notes: | Equivalent to -- | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is neither integer nor real type.<br>3. SOS is neither integer nor real type. | | |
| Example: | | | |

| | | | |
|---|---|---|---|
| Instruction: | XOR | ‘%’ | Load Operation( XORx ) |
| Effect: | TOS and SOS are removed from the stack and the value of SOS exclusively ORed with the value of TOS, SOS !! TOS, is stacked. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is not an integer value.<br>3. SOS is not an integer value. | | |
| Example: | A = B !! C | PUSH A<br>PUSH B; PUSH C; XOR<br>ASSVAL | |

### 5.2. Jumps and Labels

These ICODE instructions are used to control the flow of executable code by the program.

There are two types of labels:

- User specified labels
- Compiler generated labels

| | | | |
|---|---|---|---|
| Instruction: | FOR <label> | 'f' | Compile For( Tag ) |
| Effect: | This instruction marks the start of a FOR statement.<br><label> is the label to be jumped to on repeat and <label+1> is the label to be jumped to on an exit. | | |
| Notes: | The corresponding repeat will be the next instruction of the form:<br>Backward <label>.<br><label+1> should only be defined explicitly if the loop contains an explicit exit instruction. | | |
| Error: | 1. The stack contains less than four items.<br>2. The top three items on the stack are not integer values.<br>3. The fourth item on the stack is not a reference to an integer variable.<br>4. No Backward <label> instruction occurs before the end of the current block. | | |
| Example: | A(j) = 0 for J = 1, 1, N | PUSH J<br>PUSHI 1<br>PUSHI 1<br>PUSH N<br>FOR 40<br>PUSH A<br>PUSH J<br>ACCESS<br>PUSHI 0<br>ASSVAL<br>REPEAT 40 | |

| | | | |
|---|---|---|---|
| Instruction: | GOTO <label> | 'F' | Jump Forward( Tag, Always ) |
| Effect: | Control is transferred forward unconditionally to <label> | | |
| Notes: | | | |
| Error: | 1. <label> does not get defined by a Label instruction before the end of the current block. | | |
| Example: | if X=0 then Y=1 else Y=2 | PUSH X<br>PUSHI 0<br>COMPARE<br>JNE 20<br>PUSH Y<br>PUSHI 1<br>ASSVAL<br>GOTO 21<br>LABEL 20<br>PUSH Y<br>PUSHI 2<br>ASSVAL<br>LABEL 21 | |

| Instruction: | JE <label> | ‘=’ | Jump Forward( Tag, EQ ) |
|---|---|---|---|
| Effect: | When execution of the object program reaches this point, control is to be transferred to the given simple label if the condition code is set 'equal', otherwise control is to pass onto the next instruction. | | |
| Notes: | | | |
| Error: | 1. The previous instruction did not set the condition-code.<br>2. <label> does not get defined by a LABEL instruction before the end of the current block. | | |
| Example: | IF x <> y THEN p = q; | PUSH x; PUSH y; COMPARE; JE 12<br>PUSH p; PUSH q; ASSVAL<br>LABEL 12 | |

| Instruction: | JG <label> | ‘>’ | Jump Forward( Tag, GT ) |
|---|---|---|---|
| Effect: | When execution of the object program reaches this point, control is to be transferred to the given simple label if the condition code is set 'greater than', otherwise control is to pass onto the next instruction. | | |
| Notes: | <label> must refer to a simple label which must be defined somewhere after the JG instruction, that is JG can only specify a forward jump (although the object program may use a backward jump). | | |
| Error: | 1. The previous instruction did not set the condition-code.<br>2. <label> does not get defined by a LABEL instruction before the end of the current block. | | |
| Example: | if X <= Y then P = Q | PUSH X<br>PUSH Y<br>COMPARE<br>JG 12<br>PUSH P<br>PUSH Q<br>ASSVAL<br>LABEL 12 | |

| Instruction: | JGE <label> | ‘)’ | Jump Forward( Tag, GE ) |
|---|---|---|---|
| Effect: | When execution of the object program reaches this point control is to be transferred to the given simple label if the condition code is set 'greater than' or 'equal' , otherwise control is to pass onto the next instruction. | | |
| Notes: | <label> must refer to a simple label which must be defined somewhere after the JGE instruction, that is JGE can only specify a forward jump (although the object program may use a backward jump). | | |
| Error: | 1. The previous instruction did not set the condition-code.<br>2. <label> does not get defined by a LABEL instruction before the end of the current block. | | |
| Example: | if X < Y then P = Q | PUSH X<br>PUSH Y<br>COMPARE<br>JGE 12<br>PUSH P<br>PUSH Q<br>ASSVAL<br>LABEL 12 | |

| Instruction: | JL <label> | ‘<’ | Jump Forward( Tag, LT ) |
|---|---|---|---|
| Effect: | When execution of the object program reaches this point control is to be transferred to the given simple label if the condition code is not set 'equal', otherwise control is to pass onto the next instruction. | | |
| Notes: | <label> must refer to a simple label which must be defined somewhere after the BLT instruction, that is BLT can only specify a forward jump (although the object program may use a backward jump). | | |
| Error: | 1. The previous instruction did not set the condition-code.<br>2. <label> does not get defined by a LABEL instruction before the end of the current block. | | |
| Example: | if X >= Y then P = Q | PUSH X<br>PUSH Y<br>COMPARE<br>JL 12<br>PUSH P<br>PUSH Q<br>ASSVAL<br>LABEL 12 | |

| | | | |
|---|---|---|---|
| Instruction: | JLE <label> | ‘(‘ | Jump Forward( Tag, LE ) |
| Effect: | When execution of the object program reaches this point, control is to be transferred to the given simple label if the condition code is set 'less than' or 'equal', otherwise control is to pass onto the next instruction. | | |
| Notes: | <label> must refer to a simple label which must be defined somewhere after the JLE instruction, that is JLE can only specify a forward jump (although the object program may use a backward jump). | | |
| Error: | 1. The previous instruction did not set the condition-code.<br>2. <label> does not get defined by a LABEL instruction before the end of the current block. | | |
| Example: | if X > Y then P = Q | PUSH X<br>PUSH Y<br>COMPARE<br>JLE 12<br>PUSH P<br>PUSH Q<br>ASSVAL<br>LABEL 12 | |

| | | | |
|---|---|---|---|
| Instruction: | JNE <label> | ‘#’ | Jump Forward( Tag, NE ) |
| Effect: | When execution of the object program reaches this point control is to be transferred to the given simple label if the condition code is not set 'equal', otherwise control is to pass onto the next instruction. | | |
| Notes: | <label> must refer to a simple label which must be defined somewhere after the JNE instruction, that is JNE can only specify a forward jump (although the object program may use a backward jump). | | |
| Error: | 1. The previous instruction did not set the condition-code.<br>2. <label> does not get defined by a LABEL instruction before the end of the current block. | | |
| Example: | if X = Y then P = Q | PUSH X<br>PUSH Y<br>COMPARE<br>JNE 12<br>PUSH P<br>PUSH Q<br>ASSVAL<br>LABEL 12 | |

| Instruction: | JNZ <label> | ‘t’ | JNZ |
|---|---|---|---|
| Effect: | When execution of the object program reaches this point, control is to be transferred to the given simple label if the condition code is set 'true', otherwise control is to pass onto the next instruction. | | |
| Notes: | Branch on TRUE (# 0)<br><label> must refer to a simple label which must be defined somewhere after the JNZ instruction, that is JNZ can only specify a forward jump (although the object program may use a backward jump). | | |
| Error: | 1. The previous instruction did not set the condition-code.<br>2. <label> does not get defined by a LABEL instruction before the end of the current block. | | |
| Example: | if not B then P := Q | PUSH B<br>Test-Boolean<br>JNZ 12<br>PUSH P<br>PUSH Q<br>ASSVAL<br>LABEL 12 | |

| Instruction: | JUMP <tag> | ‘J’ | User Jump( Tag ) |
|---|---|---|---|
| Effect: | Control is transferred unconditionally to the general label <tag>. | | |
| Notes: | Jump to user label<br>This control transfer may pass over block boundaries. | | |
| Error: | 1. <tag> is defined but is not a general label.<br>2. <tag> is undefined but does not become defined in a suitable block before the end of the program. A suitable block is one which is either the same block as the Jump instruction or is a block which properly contains that instruction. | | |
| Example: | X = 1<br>Pos: Y = Y+1<br>->Pos if Y < 0 | PUSH X; PUSHI 1; ASSVAL<br>LOCATE Pos<br>PUSH Y; PUSH Y; PUSHI 1; ADD; ASSVAL<br>PUSH Y; PUSHI 0; COMPARE<br>JGE 18<br>JUMP Pos<br>LABEL 18 | |

| | | | |
|---|---|---|---|
| Instruction: | JZ <label> | ‘k’ | Jump Forward( Tag, FF ) |
| Effect: | When execution of the object program reaches this point, control is to be transferred to the given simple label if the condition code is set 'false', otherwise control is to pass onto the next instruction. | | |
| Notes: | Branch on FALSE (= 0)<br><label> must refer to a simple label which must be defined somewhere after the BF instruction, that is BF can only specify a forward jump (although the object program may use a backward jump). | | |
| Error: | 1. The previous instruction did not set the condition-code.<br>2. <label> does not get defined by a LABEL instruction before the end of the current block. | | |
| Example: | if B then P := Q; | PUSH B; Test-Boolean; JZ 12<br>PUSH P; PUSH Q; ASSVAL<br>LABEL 12 | |

| | | | |
|---|---|---|---|
| Instruction: | LABEL <label> | ‘L’ | Define User Label( Tag ) |
| Effect: | The simple label <label> is defined to be here. If outstanding references to the label exist they are satisfied and the label ceases to be defined. This ensures that all references to this label are in the same direction. | | |
| Notes: | Define user label | | |
| Error: | | | |
| Example: | S = "**" if S = "" | PUSH S; PUSHS "**"; COMPARE<br>JNE 26<br>PUSH S; PUSHS ""; ASSVAL<br>LABEL 26 | |

| | | | |
|---|---|---|---|
| Instruction: | LOCATE <tag> | ‘:’ | Define Compiler Label( Tag ) |
| Effect: | The tag is defined as a general label if necessary and made to reference the current position in the program. | | |
| Notes: | Define compiler label | | |
| Error: | 1. <tag> is already defined. | | |
| Example: | X = 1<br>Pos: Y = Y+1<br>->Pos if Y < 0 | PUSH X; PUSHI 1; ASSVAL<br>LOCATE Pos<br>PUSH Y; PUSH Y; PUSHI 1; ADD; ASSVAL<br>PUSH Y; PUSHI 0; COMPARE<br>JGE 18<br>JUMP Pos<br>LABEL 18 | |

| | | | |
|---|---|---|---|
| Instruction: | REPEAT <label> | ‘B’ | Jump Backward( Tag ) |
| Effect: | Control is to be transferred unconditionally to <label> at run-time. | | |
| Notes: | Backward Jump | | |
| Error: | 1. <label> is currently undefined. | | |
| Example: | X=X+1 while A(X) = 0 | LABEL 16<br>PUSH A; PUSH X; ACCESS<br>PUSHI 0; COMPARE; BNE 17<br>PUSH X; PUSH X; PUSHI 1; ADD; ASSVAL<br>REPEAT 16<br>LABEL 17 | |

| | | | |
|---|---|---|---|
| Instruction: | SJUMP <tag> | ‘W’ | Switch Jump( Tag ) |
| Effect: | TOS is used to index into the switch vector and control is then transferred to the selected label. TOS is removed from the stack. | | |
| Notes: | Jump to switch | | |
| Error: | 1. The stack is empty.<br>2. <tag> is not a switch. | | |
| Example: | ->Sw(J) | Stack J; Switch-Jump Sw | |

| | | | |
|---|---|---|---|
| Instruction: | SLABEL <tag> | ‘_’ | Switch Label( Tag ) |
| Effect: | The label selected from the switch vector is defined to be here. TOS is removed from the stack. | | |
| Notes: | Define switch label | | |
| Error: | 1. The stack is empty.<br>2. <tag> is not a switch in the current block.<br>3. TOS is not an integer value within the bounds of <tag>. | | |
| Example: | Sw(12): | PUSHI 12; SLABEL Sw | |

### 5.3. Assignment

These ICODE instructions are used to access and store data within the program.

| | | | |
|---|---|---|---|
| Instruction: | ASSPAR | ‘p’ | Load Assign( -1 ) |
| Effect: | TOS is passed as the next parameter to SOS. TOS is removed from the stack leaving SOS as the new TOS. | | |
| Notes: | Parameters must be specified in the order of the definition of the parameter list. If the parameter list is empty the occurrence of this instruction implies that the procedure has a variable number of parameters and so the parameters are to be passed in a C-like manner; this also requires that the procedure be called using the Variable-Call instruction. | | |
| Error: | 1. The stack contains less than two items.<br>2. SOS is not a procedure descriptor.<br>3. TOS is unsuitable for this parameter. | | |
| Example: | J = Calc(1, K) | PUSH J<br>PUSH Calc<br>PUSHI 1; ASSPAR<br>PUSH K; ASSPAR<br>CALL | |

| | | | |
|---|---|---|---|
| Instruction: | ASSREF | ‘Z’ | Load Assign( 0 ) |
| Effect: | The pointer variable referenced by SOS is made to point at the variable referenced by TOS. Both TOS and SOS are removed from the stack. | | |
| Notes: | Assign address '==' | | |
| Error: | 1. The stack contains less than two items.<br>2. SOS is not a reference to a pointer variable.<br>3. TOS is not a reference to a variable.<br>4. The types of TOS and SOS are different. | | |
| Example: | P == Q | PUSH P; PUSH Q; ASSREF | |

| | | | |
|---|---|---|---|
| Instruction: | ASSVAL | ‘S’ | Load Assign( 1 ) |
| Effect: | The value of TOS is assigned to the data item referenced by SOS. Both TOS and SOS are removed from the stack. | | |
| Notes: | Integer values will be converted to real if necessary.<br>Normal value assignment | | |
| Error: | 1. The stack contains less than two items. | | |
| Example: | A = B+C | PUSH A<br>PUSH B; PUSH C; ADD<br>ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | FALSE | ‘K’ | Load Return( False ) |
| Effect: | The current block returns false. | | |
| Notes: | %false<br>This is usually accomplished by setting the true condition-code appropriately. | | |
| Error: | 1. The current block is not a predicate. | | |
| Example: | false if Flag = 0 | PUSH Flag; PUSHI 0; COMPARE<br>JNE 42<br>FALSE<br>LABEL 42 | |

| | | | |
|---|---|---|---|
| Instruction: | JAM | ‘j’ | Load Assign( 2 ) |
| Effect: | The value of TOS is assigned to the data item referenced by SOS. Both TOS and SOS are removed from the stack. | | |
| Notes: | JAM transfer<br>Truncate/Jam the source to fit into the destination<br>Excess byte data will be omitted. | | |
| Error: | 1. The stack contains less than two items. | | |
| Example: | ! A is a string(15)<br>! B is a string(31)<br>A = B | PUSH A<br>PUSH B<br>JAM | |

| | | | |
|---|---|---|---|
| Instruction: | MAP | ‘M’ | Load Return( Map ) |
| Effect: | The address of the object referenced by TOS is returned as the result of the map. | | |
| Notes: | MAP result | | |
| Error: | 1. The current block is not a map.<br>2. The stack is empty.<br>3. TOS is not a reference to a data object. | | |
| Example: | result == X | PUSH X; MAP | |

| | | | |
|---|---|---|---|
| Instruction: | RESULT | ‘V’ | Load Return ( Fn ) |
| Effect: | TOS is removed from the stack and returned as the result of the function defined by the current block. | | |
| Notes: | FN result<br>Integer values will be converted to real if necessary. | | |
| Error: | 1. The stack is empty.<br>2. The current block is not a function. | | |
| Example: | result = "Hello" | PUSHS "Hello"<br>RESULT | |

|  |  |  |  |
|---|---|---|---|
| Instruction: | RETURN | ‘R’ | Load Return( Routine ) |
| Effect: | A return sequence is generated to return control from the current block. |  |  |
| Notes: | RETURN |  |  |
| Error: |  |  |  |
| Example: | return if X # 0 | PUSH X; PUSHI 0; COMPARE<br>JE 19<br>RETURN<br>LABEL 19 |  |

|  |  |  |  |
|---|---|---|---|
| Instruction: | TRUE | ‘T’ | Load Return( True ) |
| Effect: | The current block returns true. |  |  |
| Notes: | %true<br>This is usually accomplished by setting the true condition-code appropriately. |  |  |
| Error: | 1. The current block is not a predicate. |  |  |
| Example: | true if Flag = 0 | PUSH Flag; PUSHI 0; COMPARE<br>JNE 42<br>TRUE<br>LABEL 42 |  |

### 5.4. Embedding Machine Code

These ICODE instructions allow a programmer to insert assembler code or binary values into the program’s instruction sequence. Multiple PLANT instructions can be used to insert legal machine code instructions that the embedded assembler function (MCODE) does not yet recognise.

| | | | |
|---|---|---|---|
| Instruction: | MCODE <string> | ‘w’ | Machine Code( Get Ascii( ';' ) ) |
| Effect: | The string is decoded as assembler text and added to the instruction stream | | |
| Notes: | escape to assembler code<br>Assembler code is encoded by the IMP Compiler Pass1 | | |
| Error: | | | |
| Example: | *MOV_ %eax,#59 | MCODE …. | |

| | | | |
|---|---|---|---|
| Instruction: | PLANT <byte> | ‘P’ | Load Plant |
| Effect: | Stores the byte in the instruction stream | | |
| Notes: | Machine code literal | | |
| Error: | | | |
| Example: | *45 | PLANT 45 | |

### 5.5. Event/Signal

The IMP language has a simple mechanism to signal events/exceptions and insert handler code which the following ICODE instructions direct the code-generator to cater for.

| | | | |
|---|---|---|---|
| Instruction: | EVENT <n> | ‘e’ | Signal Event( Tag ) |
| Effect: | The event <n>,SOS,TOS is signalled. | | |
| Notes: | %signal event | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is not an integer value.<br>3. SOS is not an integer value. | | |
| Example: | signal 1,2,3 | PUSHI 3; PUSHI 2; EVENT 1 | |

| | | | |
|---|---|---|---|
| Instruction: | ON <n> <label> | ‘o’ | EventTrap( TagC, Tag ) |
| Effect: | This instruction marks the start of an on event block.<br><n> is a sixteen-bit set of flags where each trapped event is represented by a 1-bit, with the least-significant bit corresponding to event 0 and the most-significant bit event 15. | | |
| Notes: | %on %event block<br><label> is the simple label which marks the end of the event block. | | |
| Error: | 1. <n> does not have any bits set.<br>2. <label> is not defined before the end of the current block | | |
| Example: | on 9 start; return; finish | ON 512 17<br>RETURN<br>LABEL 17 | |

| | | | |
|---|---|---|---|
| Instruction: | STOP | ‘s’ | Load Perm(stop, 0) |
| Effect: | Generates code to stop the target program (not the compiler). | | |
| Notes: | %stop<br>This is equivalent to %signal 0,0,0 | | |
| Error: | | | |
| Example: | | | |

### 5.6. Compare/Test

These ICODE instructions are used to indicate code required to compare values.

| | | | |
|---|---|---|---|
| Instruction: | COMPARE | ‘?’ | Load Compare Values |
| Effect: | SOS is compared to TOS and the condition-code is set appropriately. TOS and SOS are then removed from the stack. | | |
| Notes: | The types of both TOS and SOS must be one of the following:<br>Integer, Real, String, Record, Set<br>Integer values will be converted to real if necessary.<br>The comparison is signed where integer values are concerned. | | |
| Error: | 1. The stack contains less than two items.<br>2. The types of TOS and SOS are incompatible. | | |
| Example: | if S < "123" then X = 0 | PUSH S; PUSHS "123"; COMPARE; BGE 13<br>PUSH X; PUSHI 0; ASSVAL<br>LABEL 13 | |

| | | | |
|---|---|---|---|
| Instruction: | COMPAREA | ‘C’ | Load Compare Addresses |
| Effect: | The address of SOS is compared to the address of TOS and the condition-code is set appropriately. TOS and SOS are then removed from the stack. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is not a reference for a data object.<br>3. SOS is not a reference for a data object. | | |
| Example: | integername M; integer N<br>if N == M then Newline | PUSH N; PUSH M; COMPAREA; BNE 14<br>PUSH Newline; CALL<br>LABEL 14 | |

| | | | |
|---|---|---|---|
| Instruction: | COMPARED | ‘”’ | Load Compare Double |
| Effect: | SOS is compared to TOS and the condition-code is set appropriately. TOS and SOS are then removed from the stack. | | |
| Notes: | The types of both TOS and SOS must be one of the following:<br>Integer, Real, String, Record, Set<br>Integer values will be converted to real if necessary.<br>The comparison is signed where integer values are concerned. | | |
| Error: | 1. The stack contains less than two items.<br>2. The types of TOS and SOS are incompatible. | | |
| Example: | if S < "123" then X = 0 | PUSH S; PUSHS "123"; COMPARED; BGE 13<br>PUSH X; PUSHI 0; ASSVAL<br>LABEL 13 | |

### 5.7. Parameter/Variable Declaration

These ICODE instructions help specify the type of variables present inside the program source.

Although IMP allows spaces within a variable name, internally the spaces are removed from variable names by the first stage of the compiler when generating the ICODE sequence.

The ICODE instruction, DEF, indicates the data-type, size and structure of a variable. When referencing the variable, the tag value is used.

| | | | |
|---|---|---|---|
| Instruction: | ALIAS <string> | 'G' | Get Alias Value( ReadString ) |
| Effect: | <string> is noted as the current alias. | | |
| Notes: | See 'Begin' and 'Define' | | |
| Error: | None | | |
| Example: | external integer Thing alias "SS$THING" | ALIAS "SS$THING"<br>DEF THING.......... | |

| | | | |
|---|---|---|---|
| Instruction: | ALT <b> | '~' | AlternateFormat( ReadByte ) |
| Effect: | This instruction marks an alternative sequence of tag definitions. | | |
| Notes: | The instructions ALT-start and ALF-end are brackets and must be properly nested.<br><b> = 'A' =>ALT-start<br><b> = 'C' =>ALT-next<br><b> = 'B' =>ALT-end | | |
| Error: | 1. List flag is not set. | | |
| Example: | recordformat F(integer X, (integer Y or real Z))<br>DEF F.......<br>START<br>  DEF X.....<br>  ALT-start<br>    DEF Y.....<br>  ALT-next<br>    DEF Z.....<br>  ALT-end<br>FINISH | | |

| | | | |
|---|---|---|---|
| Instruction: | BOUNDS | 'b' | Load Constant Bounds |
| Effect: | The value of TOS is noted as 'upper-bound' and the value of SOS is noted as 'lower-bound'. TOS and SOS are removed from the stack. | | |
| Notes: | This is used to define a constant bounded Dope Vector<br>This instruction is used as a preliminary to defining switch vectors and own, const and external arrays. | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is not an integer value.<br>3. SOS is not an integer value.<br>4. The value of TOS is less than the value of SOS. | | |
| Example: | switch Sw(-3:3) | PUSHI 3; Negate; PUSHI 3; BOUNDS<br>DEF Sw....... | |

| | | | |
|---|---|---|---|
| Instruction: | DEF <tag> [id] <a> <b> <c> | '$' | Define Var( Tag, AsciiC, TagC, TagC, Tag ) |
| Effect: | A new tag value is created.<br><br><tag> defines the tag index which will be used to select the value.<br><br>[id] specifies the actual identifier associated with the described object. It is a sequence of zero or more characters terminated by a comma. This identifier will be used for external linkage if necessary unless overridden by an Alias. [id] will also be used for run-time diagnostic information.<br><br><a> A two-byte value: <a> = T<<4+F where:<br>T = 0 : void<br>T = 1 : integer {qualified by <b>}<br>T = 2 : real {qualified by <b>}<br>T = 3 : string {maximum length <b>}<br>T = 4 : record {format <b>}<br>T = 5 : boolean<br>T = 6 : set<br>T = 7 : 8-bit-enumerated {format <b>}<br>T = 8 : 16-bit-enumerated {format <b>}<br>T = 9 : pointer<br>T = 10 : char<br>T = 11 : unsigned {qualified by <b>}<br>T = 12-15 : undefined {error}<br><br>F = 0 : void<br>F = 1 : simple {byte}<br>F = 2 : indirect {bytename}<br>F = 3 : general label<br>F = 4 : recordformat<br>F = 5 : undefined {error}<br>F = 6 : switch<br>F = 7 : routine<br>F = 8 : function<br>F = 9 : map<br>F = 10 : predicate<br>F = 11 : array {array}<br>F = 12 : array indirect {arrayname}<br>F = 13 : indirect array {namearray}<br>F = 14 : indirect array indirect {namearrayname}<br>F = 15 : undefined {error}<br><br><b> If T is INTEGER <b> takes the following meanings b=:<br>1, full range (depends on machine i.e. 32-bit or 64-bit)<br>2, range 0 .. 255 (=0..2^8-1)<br>3, range -32768 .. 32767 (=-2^15..2^15-1)<br>4, range -2^31..2^31-1<br>5, range -2^63..2^63-1 | | |

| | |
|---|---|
| | If T is UNSIGNED <b> takes the following meanings b=:<br>1, full range (depends on machine i.e. 32-bit or 64-bit)<br>2, range 0..2^8-1<br>3, range 0..2^16-1<br>4, range 0..2^32-1<br>5, range 0..2^64-1<br><br>If T is REAL <b> takes the following meanings b=:<br>1, normal precision<br>4, double precision<br><br>If T is STRING <b> gives the maximum length of the string.<br>If T is RECORD <b> gives the tag of the corresponding recordformat.<br>If T is enumerated <b> gives the tag of the dummy format used to identify the enumerated value identifiers.<br><br><c> is a two-byte value: U<<5+I<<4+S<<3+X where:<br>U = 1 check the object for unassigned, U = 0 otherwise<br>I = 1 if the object is an indirect object, I = 0 otherwise<br>S = 1 if this is a spec, S = 0 otherwise<br>X = 0 :: automatic (stack) allocation<br>X = 1 :: own<br>X = 2 :: constant<br>X = 3 :: external<br>X = 4 :: system<br>X = 5 :: dynamic<br>X = 6 :: primitive<br>X = 7 :: permanent<br><br>An indirect object (I=1) differs from F=2 in that F=2 implies that the actual object created will be a pointer and will be dereferenced whenever used unless explicit action is taken (e.g. use of Assign-Reference).<br>If I=1 a pointer will be created (usually as an integer) and will be treated as an integer (or address) with no automatic dereferencing taking place. |
| Notes: | The tag values within a block should be dense and preferably consecutive.<br><br>All tag values within a block must have values greater than the maximum tag value yet defined within the enclosing block.<br><br>Tag definitions remain valid until the end of the enclosing block.<br><br>The tag values used within a recordformat definition must all be zero; the fields of a record are selected by their position in the format, numbered starting from one. |

x = illegal combination of T,F values

| | | F | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| T | 0 | | X | | | X | X | | | X | | | X | | X | |
| | 1 | X | | | X | X | X | X | X | | | X | | | | |
| | 2 | X | | | X | X | X | X | X | | | X | | | | |
| | 3 | X | | | X | X | X | X | X | | | X | | | | |
| | 4 | X | | | X | | X | X | X | | | X | | | | |
| | 5 | X | | | X | X | X | X | X | | | X | | | | |
| | 6 | X | | | X | X | X | X | X | | | X | | | | |
| | 7 | X | | | X | X | X | X | X | | | X | | | | |
| | 8 | X | | | X | X | X | X | X | | | X | | | | |
| | 9 | X | | | X | X | X | X | X | | | X | | | | |
| | 10 | X | | | X | X | X | X | X | | | X | | | | |

| | |
|---|---|
| Error: | |
| Example: | |

| Instruction: | DIM <n><d> | 'd' | Dimension( TagC, Tag ) |
|---|---|---|---|
| Effect: | <d> pairs of integer values on the stack are used to define the bounds of the last <n> arrays to have been defined. Code is generated, if necessary, to allocate the arrays and the definitions are adjusted to reference the appropriate storage. | | |
| Notes: | The pairs of values are stacked in order of the declaration, that is, first dimension first.<br>In each pair of values the lower bound is stacked before the upper bound.<br>The last <n> tags must have had consecutive index values.<br>NB in params: d =0 -> simple array, # 0 -> array-in-record | | |
| Error: | 1. The stack contains less than 2*<d> items.<br>2. The last <n> definitions were not all arrays. | | |
| Example: | integerarray A, B, C(1:2, Low:4) | DEF A......<br>DEF B......<br>DEF C......<br>PUSHI 1; PUSHI 2<br>PUSH Low; PUSHI 4<br>DIM 3 2 | |

| | | | |
|---|---|---|---|
| Instruction: | FINISH | ‘}’ | Finish Params |
| Effect: | This instruction marks the end of a list of tag definitions corresponding to either a parameter list or a recordformat definition. List flag is cleared and the tag list is processed in any ways necessary.<br>If the tag list is associated with a procedure spec or a recordformat this instruction also marks the 'end' of the associated 'block'. | | |
| Notes: | The list of definitions may be empty. | | |
| Error: | 1. List flag is clear.<br>2. There has been an unmatched Alt-Start. | | |
| Example: | routine Test(integer j,k) | DEF Test.........<br>START<br>DEF j......<br>DEF k......<br>FINISH | |
| | recordformat F(integer P or real R) | DEF F..........<br>START<br>DEF P......<br>ALT-next<br>DEF R......<br>FINISH | |

| | | | |
|---|---|---|---|
| Instruction: | INIT <n> | ‘A’ | Init( Tag ) |
| Effect: | <n> copies of the init-value are added to the list of values associated with the init-variable. The init-value is either the default value (unassigned) if the stack is empty, the value of TOS (possibly converted to real) if TOS is a constant, or the address of TOS if TOS is a variable. The init-variable is the last static object to have been defined using Define. | | |
| Notes: | Used to initialise an OWN variable | | |
| Error: | 1. TOS, if it exists, is not of the same type as the init-variable. | | |
| Example: | ownintegerarray A(1:5) = 1(3), 4, 99 | PUSHI 1; PUSHI 5; BOUNDS<br>DEF A.......<br>PUSHI 1; PUSHI 3<br>PUSHI 4; INIT 1<br>PUSHI 99; INIT 1 | |
| | owninteger P = -1 | PUSHI 1; NEGATE<br>DEF P......<br>INIT 1 | |

|  |  |  |  |
|---|---|---|---|
| Instruction: | SETFORMAT <tag> | ‘^’ | Set Record Format( Tag ) |
| Effect: | TOS is converted to be a record of format <tag>. This never involves any instructions being executed in the object program. |  |  |
| Notes: |  |  |  |
| Error: | 1. <tag> is not a recordformat.<br>2. The stack is empty.<br>3. TOS is not a reference to a variable. |  |  |
| Example: |  |  |  |

|  |  |  |  |
|---|---|---|---|
| Instruction: | START | ‘{‘ | StartParams |
| Effect: | This instruction marks the start of a list of tags defining the parameters to a procedure or the components of a record. List flag is set. |  |  |
| Notes: | This instruction must always follow the definition of a procedure or recordformat tag. There must be a matching 'Finish' before the end of the block. |  |  |
| Error: | 1. List flag is set.<br>2. The last instruction was not a 'Define' which introduced a procedure or recordformat. |  |  |
| Example: | routine Newline; ....; end | DEF Newline.........<br>START<br>FINISH<br>......<br>END |  |

### 5.8. Access Variables

These ICODE instructions are used to access the variables defined in the previous set of ICODE instructions.

| | | | |
|---|---|---|---|
| Instruction: | ACCESS | ‘a’ | Load Array Ref( 0 ) |
| Effect: | TOS is used as the final index into SOS. TOS is removed from the stack leaving SOS as the new TOS. | | |
| Notes: | This instruction is used to process the final dimension of an N dimensional array. See Index for the previous dimensions. | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is not an integer value.<br>3. SOS does not describe an array. | | |
| Example: | A(j) = 0 | PUSH A<br>PUSH j; ACCESS<br>PUSHI 0; ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | INDEX | ‘i’ | Load Array Ref( 1 ) |
| Effect: | TOS is used as the next index into SOS. TOS is removed from the stack leaving SOS as the new TOS. | | |
| Notes: | This instruction is used to process the first N-1 dimensions of an N dimensional array. See Access for the final dimension. | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is not an integer value.<br>3. SOS is not an array descriptor. | | |
| Example: | | | |

| | | | |
|---|---|---|---|
| Instruction: | PUSH <tag> | ‘@’ | Load Var( Tag ) |
| Effect: | The tag with index value <tag> is pushed onto the stack. | | |
| Notes: | Load variable descriptor (calls Stack Var) | | |
| Error: | 1. <tag> has not been defined. | | |
| Example: | A = B | PUSH A; PUSH B; ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | PUSHI <integer> | ‘N’ | Load Const( ReadInteger ) |
| Effect: | The integer value <integer> is pushed into the stack and becomes the new TOS. | | |
| Notes: | Load integer constant (calls Push Const)<br>The instruction 'Byte' is an abbreviation for this instruction when the value is in the range 0..255. | | |
| Error: | | | |
| Example: | X = 2500 | PUSH X; PUSHI 2500; ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | PUSHR <real> | ‘D’ | Input Real Value( ReadReal ) |
| Effect: | The real constant <real> is pushed onto the stack. | | |
| Notes: | Stack real constant | | |
| Error: | | | |
| Example: | Z = -1.23 | PUSH Z; PUSHR <1.23>; NEGATE<br>ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | PUSHS <string> | ‘”’ | Input String Value( ReadString ) |
| Effect: | The string constant <string> is pushed onto the stack. | | |
| Notes: | Stack string constant | | |
| Error: | | | |
| Example: | S = "Hello" | PUSH S<br>PUSHS "Hello"<br>ASSVAL | |

| | | | |
|---|---|---|---|
| Instruction: | RESOLVE <flag> | ‘r’ | Resolve( Tag ) |
| Effect: | | | |
| Notes: | Split the string S ->A.(B).C | | |
| Error: | | | |
| Example: | | | |

| | | | |
|---|---|---|---|
| Instruction: | SELECT <n> | ‘n’ | SelectField( Tag ) |
| Effect: | TOS is replaced by the <n>'th item in the format of TOS. Fields within records are numbered from 1; alternative markers have no effect on the numbering. | | |
| Notes: | Select member from record format | | |
| Error: | 1. The stack is empty.<br>2. TOS is not a record.<br>3. The format of TOS does not contain at least <n> items. | | |
| Example: | recordformat F(integer P or record (F)name Q)<br>record (F) R<br>R_Q_P = 0 | <br><br>PUSH R<br>SELECT 2<br>SELECT 1<br>PUSHI 0<br>ASSVAL | |

### 5.9. Compiler Directives

These ICODE instructions are used to control the operation of the various compiler stages.

| | | | |
|---|---|---|---|
| Instruction: | CONTROL <n> | ‘z’ | Set Control( ReadTag ) |
| Effect: | The value <n> is of the form p<<14+q. The value q is to be used by the p'th pass of the compiler in an implementation-specific manner. | | |
| Notes: | | | |
| Error: | | | |
| Example: | | | |

| | | | |
|---|---|---|---|
| Instruction: | DIAG <n> | ‘y’ | Set Diagnose( ReadTag ) |
| Effect: | The value <n> is of the form p<<14+q. The value q is to be used by the p'th pass of the compiler in an implementation-specific manner. | | |
| Notes: | | | |
| Error: | | | |
| Example: | | | |

| | | | |
|---|---|---|---|
| Instruction: | EOF | 10 | |
| Effect: | The compilation is to be abandoned with an error message. | | |
| Notes: | This instruction is required at the end of an I-code file even though in correct programs it will never be executed. It is there to provide a check on the operation of the code-generator and to permit the code-generator to use a one-character look-ahead. | | |
| Error: | | | |
| Example: | | | |

| | | | |
|---|---|---|---|
| Instruction: | LANG <flags> | ‘l’ | Load Language Flags( Tag ) |
| Effect: | The 16-bit parameter <flags>sets the language flags. | | |
| Notes: | Currently only support standard IMP | | |
| Error: | | | |
| Example: | | | |

| | | | |
|---|---|---|---|
| Instruction: | LINE <n> | ‘O’ | Update Line( Tag ) |
| Effect: | The current position is associated with the start of the code for source line <n>. | | |
| Notes: | | | |
| Error: | 1. The stack is not empty. | | |
| Example: | X = 1<br>Y = 3; Z = 4<br>LINE 1; PUSH X; PUSHI 1; ASSVAL<br>LINE 2; PUSH Y; PUSHI 3; ASSVAL<br>LINE 2; PUSH Z; PUSHI 4; ASSVAL | | |

| | | | |
|---|---|---|---|
| Instruction: | MONITOR | ‘m’ | Load Monitor |
| Effect: | Plant code to execute the specific monitor action | | |
| Notes: | | | |
| Error: | | | |
| Example: | | | |

### 5.10. Block Structure/Call

These ICODE instructions indicate the block structure of the program, and calling sequence between the different blocks of code.

| | | | |
|---|---|---|---|
| Instruction: | BEGIN | ‘H’ | Load Compile Begin |
| Effect: | An anonymous procedure is defined here and called once. A new block is entered. If an alias has been noted (see Alias) that string will be used for identifying the block in diagnostic information leaving no alias noted. | | |
| Notes: | The sequence "Begin ...... End" may always be replaced by a sequence of the form:<br>Define X......; Start; Finish; ...... End; Stack X; Call<br>where X is a suitable unique tag. | | |
| Error: | None | | |
| Example: | begin; Newline; end | BEGIN<br>PUSH Newline; CALL<br>END | |

| | | | |
|---|---|---|---|
| Instruction: | CALL | ‘E’ | Load Compile Call( top ) |
| Effect: | The procedure described by TOS is called. If TOS is a procedure which returns a result TOS is replaced by that result, otherwise TOS is removed from the stack. | | |
| Notes: | Predicates do not return a result but set the condition-code. | | |
| Error: | 1. The stack is empty.<br>2. TOS does not describe a procedure.<br>3. There has not been the same number of parameters assigned using Assign-Parameter as is specified by the parameter list. | | |
| Example: | Newlines(4) | PUSH Newlines<br>PUSHI 4; ASSPAR<br>CALL | |

| | | | |
|---|---|---|---|
| Instruction: | END | ‘;’ | Load End of Block |
| Effect: | This instruction marks the end of a block. All tags defined within the block are deleted (made undefined) and become available for re-use. If the block is a routine this instruction also implies a 'return' instruction. | | |
| Notes: | | | |
| Error: | 1. The stack is not empty. | | |
| Example: | | | |

## 6. Unused ICODE Instructions

The current compiler implementation uses a limited set of ICODE instructions which are described in the previous section.

The following ICODE instructions are currently not implemented in this version of the IMP compiler suite.

Some of these unused ICODE instructions could be used to represent instances of calls to IMP run-time library routines or be replaced by embedded implicit code to be inserted by the code generation phase of the compiler.

Other unused ICODE instructions could be used to define constructs in the non-IMP compilers (i.e. C/Pascal) implemented by P.S. Robertson.

| | | | |
|---|---|---|---|
| Instruction: | Absolute | | |
| Effect: | TOS is replaced by the absolute value of TOS. | | |
| Notes: | | | |
| Error: | 3. The stack is empty.<br>4. TOS is not integer or real. | | |
| Example: | X = \|Y+Z\| | Stack X<br>Stack Y; Stack Z; Add; Absolute<br>Assign-Value | |

| | | | |
|---|---|---|---|
| Instruction: | Address | | |
| Effect: | TOS is replaced by the address of the object it describes. | | |
| Notes: | Commonly the type of an address will be indistinguishable from integer. | | |
| Error: | 1. The stack is empty.<br>2. TOS does not have an address. | | |
| Example: | P = Addr(Q) | Stack P<br>Stack Q; Address<br>Assign-Value | |

| | | | |
|---|---|---|---|
| Instruction: | Adjust | | |
| Effect: | The address of SOS is adjusted forwards or backwards by TOS items of the same size as SOS. This may be thought of as an array accessing instruction where SOS defines the zero'th element. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is not an integer.<br>3. SOS does not reference a data object. | | |
| Example: | N == N[X] | Stack N<br>Stack N; Stack X; Adjust<br>Assign-Reference | |

| | | | |
|---|---|---|---|
| Instruction: | Byte <b> | | |
| Effect: | The unsigned byte value <b> is stacked. | | |
| Notes: | This is a compact form for the Integer instruction when small values are to be stacked. | | |
| Error: | None | | |
| Example: | X = 200 | Stack X; Byte 200; Assign-Value | |

| | | | |
|---|---|---|---|
| Instruction: | Compare-Repeated-Values | | |
| Effect: | SOS is compared to TOS and the condition-code is set appropriately. SOS is then removed from the stack leaving TOS | | |
| Notes: | The types of both TOS and SOS must be one of the following:<br>Integer, Real, String, Record, Set<br>Integer values will be converted to real if one operand is real. | | |
| Error: | 1. The stack contains less than two items.<br>2. The types of TOS and SOS are incompatible. | | |
| Example: | if 1 <= X <= 12 then X = 0 | Byte 1; Stack X; Compare-Repeated-Values; BGT 15<br>Byte 12; Compare-Values; BGT 15<br>Stack X; Byte 0; Assign-Values<br>Label 15 | |

| | | | |
|---|---|---|---|
| Instruction: | Compare-Unsigned-Values | | |
| Effect: | SOS is compared against TOS with the values being interpreted as unsigned values. The condition-code is set accordingly and TOS and SOS are removed from the stack. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is not an integer value.<br>3. SOS is not an integer value. | | |
| Example: | if U1 < U2 then U2 = 0 | Stack U1; Stack U2; Compare-Unsigned-Values; Bge 31<br>Stack U2; Byte 0; Assign-Value<br>Label 31 | |

| | | | |
|---|---|---|---|
| Instruction: | Complement | | |
| Effect: | TOS is replaced by the ones-complement of TOS. | | |
| Notes: | | | |
| Error: | 1. The stack is empty<br>2. TOS is not an integer value. | | |
| Example: | P = \Q | Stack P; Stack Q; Complement; Assign-Value | |

| | | | |
|---|---|---|---|
| Instruction: | Define-Range <tag> | | |
| Effect: | The given tag is defined to be the integer range defined by lower-bound and upper-bound. | | |
| Notes: | | | |
| Error: | 1. The two bounds have not been defined. | | |
| Example: | Var x:1..10; ...... x := i; | Byte 1; Byte 10; Bounds<br>Define-Range 99.......; Define x........<br>…<br>Stack x; Stack i; Test-Range 99; Assign-Value | |

| | | | |
|---|---|---|---|
| Instruction: | Duplicate | | |
| Effect: | A copy of TOS is pushed onto the stack. | | |
| Notes: | After this instruction TOS and SOS are identical. | | |
| Error: | 1. The stack is empty. | | |
| Example: | int A[10],x; A[x]++; | Stack A; Stack x; Adjust Duplicate<br>Byte 1; Add; Assign-Value | |

| Instruction: | Eval | | |
|---|---|---|---|
| Effect: | The value described by TOS is protected against being altered as the side-effect of alterations to any variables which make up that value. | | |
| Notes: | Commonly this instruction loads the value of TOS into a machine register. | | |
| Error: | 1. The stack is empty. | | |
| Example: | a = b + c++ | Stack a; Stack b; Stack c; Add; Eval<br>Stack c; Stack c; Byte 1; Add; Assign-Value<br>Assign-Value | |

| Instruction: | Eval-Addr | | |
|---|---|---|---|
| Effect: | The address of the object described by TOS is protected against alteration. | | |
| Notes: | Commonly this instruction loads the address of the object referred to be TOS into a machine register. | | |
| Error: | 1. The stack is empty.<br>2. TOS is not a reference. | | |
| Example: | int *p; *p++ = 10; | Stack p; Eval-Addr; Stack p; Duplicate<br>Byte 1; Adjust; Assign-Value<br>Byte 10; Assign-Value | |

| Instruction: | Float | | |
|---|---|---|---|
| Effect: | The value described by TOS is converted into a real value if necessary. | | |
| Notes: | If TOS is already a real this operation is a no-op. | | |
| Error: | 1. The stack is empty.<br>2. TOS is neither integer nor real. | | |
| Example: | | | |

| Instruction: | Include <string> | | |
|---|---|---|---|
| Effect: | This instruction marks the start (or end) of the code generated from source contained in an include file.<br>If <string> is null it marks the end of an include file. | | |
| Notes: | | | |
| Error: | | | |
| Example: | | | |

| Instruction: | Init-Type <n> | | |
|---|---|---|---|
| Effect: | The type of the init-variable (see Init) is <n> for the purposes of subsequent 'Init' instructions. <n> encodes the type as in the type field of 'Define'. | | |
| Notes: | | | |
| Error: | 1. <n> does not correspond to a valid type. | | |
| Example: | | | |

| Instruction: | Int | | |
|---|---|---|---|
| Effect: | TOS is replaced by Int(TOS), that is, the nearest integer to the value of TOS. The type of the new TOS is integer. | | |
| Notes: | See the IMP Library Definition for a discussion of the details of the INT function. | | |
| Error: | 1. The stack is empty.<br>2. TOS is neither integer nor real. | | |
| Example: | I = Int(R+0.3) | Stack I; Stack R; Real <0.3>; Add<br>Int<br>Assign-Value | |

| | | | |
|---|---|---|---|
| Instruction: | Intpt | | |
| Effect: | TOS is replaced by Intpt(TOS), that is, the integer part of TOS. The type of the new TOS is integer. | | |
| Notes: | See the IMP Library Definition for a discussion of the details of the INTPT function. | | |
| Error: | 1. The stack is empty.<br>2. TOS is neither integer nor real value. | | |
| Example: | I = Intpt(R-S) | Stack I; Stack R; Stack S; Sub<br>Intpt<br>Assign-Value | |

| | | | |
|---|---|---|---|
| Instruction: | Localise | | |
| Effect: | The area pointed at by <a> is copied into the local stack frame and <a> is updated to point at the new area. | | |
| Notes: | If the first byte of the new area is at X, <a> is updated to the address X-<c>. | | |
| Error: | 1. The stack contains less than three items.<br>2. <a> is not a reference.<br>3. <b> and <c> are not integer values. | | |
| Example: | | | |

| | | | |
|---|---|---|---|
| Instruction: | Null-Set | | |
| Effect: | A descriptor of a null set is stacked. | | |
| Notes: | | | |
| Error: | | | |
| Example: | SetA := []; | Stack SetA; Null-Set; Assign-Value | |

| | | | |
|---|---|---|---|
| Instruction: | Pop | | |
| Effect: | TOS is removed from the stack. | | |
| Notes: | | | |
| Error: | 1. The stack is empty. | | |
| Example: | X := Y := 0 | Stack X; Duplicate<br>Stack Y; Duplicate<br>Byte 0; Assign-Value; Assign-Value<br>Pop | |

| | | | |
|---|---|---|---|
| Instruction: | Reference <n> | | |
| Effect: | TOS is replaced by a reference to an object of type <n> at the address given by the original TOS.<br>The value of <n> is encoded in the same way as the type information, <a>, in the Define instruction. | | |
| Notes: | | | |
| Error: | 1. The stack is empty.<br>2. TOS is not an integer (address) value. | | |
| Example: | P = Integer(Q) | Stack P; Stack Q; Reference 1<br>Assign-Value | |

| | | | |
|---|---|---|---|
| Instruction: | Remainder | | |
| Effect: | TOS and SOS are removed from the stack and are replaced by the value REM(SOS, TOS). The exact definition of REM is given in the IMP Library Definition. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is not an integer value.<br>3. SOS is not an integer value. | | |
| Example: | Digit = Rem(N, 10) | Stack Digit; Stack N; Byte 10; Remainder<br>Assign-Value | |

| | | | |
|---|---|---|---|
| Instruction: | Round | | |
| Effect: | TOS is replaced by the value ROUND(TOS) where ROUND is as defined in section 6.6.6.3 of the Pascal standard BS 6192:1982, with the extension that ROUND returns the value of its parameter if that value is already an integer. | | |
| Notes: | | | |
| Error: | 1. The stack is empty.<br>2. TOS is neither integer nor real. | | |
| Example: | i := Round(r+0.1); | Stack i; Stack r; Real <0.1>; Add<br>Round; Assign-Value | |

| | | | |
|---|---|---|---|
| Instruction: | Size-Of | | |
| Effect: | TOS is replaced by the integer value giving the number of bytes in the object referenced by the original TOS. | | |
| Notes: | | | |
| Error: | 1. The stack is empty.<br>2. TOS is not a reference to a data object. | | |
| Example: | P = Sizeof(R) | Stack P; Stack R; Size-Of; Assign-Value | |

| | | | |
|---|---|---|---|
| Instruction: | Stack-Condition <b> | | |
| Effect: | The values of SOS and TOS are compared as in Compare-Values but instead of the condition-code being set, TOS and SOS are replaced by the constant 1 (true) or 0 (false) depending on whether the condition specified by <b> is true or false. The values of <b> are the encodings of the instructions:<br>BEQ, BNE, BLT, BLE, BGT, BGE, BT, BF | | |
| Notes: | The comparison is signed when integers are concerned. | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS and SOS cannot be compared.<br>3. <b> is not a valid condition. | | |
| Example: | B := (X=Y); | Stack B<br>Stack X; Stack Y; Stack-Condition BEQ<br>Assign-Value | |

| | | | |
|---|---|---|---|
| Instruction: | Stack-In | | |
| Effect: | TOS and SOS are removed from the stack and are replaced by an integer value which is 1 (true) if SOS is IN the set TOS or 0 (false) if SOS is not IN the set TOS. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS is not a set.<br>3. SOS is not an integer. | | |
| Example: | B := (x IN s); | Stack B<br>Stack x; Stack s; Stack-In<br>Assign-Value | |

| | | | |
|---|---|---|---|
| Instruction: | Stack-Unsigned-Condition <b> | | |
| Effect: | The values of SOS and TOS are compared as in Compare-Unsigned-Values but instead of the condition-code being set TOS and SOS are replaced by the constant 1 (true) or 0 (false) depending on whether the condition specified by <b> is true or false. The values of <b> are the encodings of the instructions:<br>BEQ, BNE, BLT, BLE, BGT, BGE, BT, BF | | |
| Notes: | The comparison is unsigned when integers are concerned. | | |
| Error: | 1. The stack contains less than two items.<br>2. TOS and SOS cannot be compared.<br>3. <b> is not a valid condition. | | |
| Example: | B := (Ux < Uy); | Stack B<br>Stack Ux; Stack Uy; Stack-Unsigned-Condition BLT<br>Assign-Value | |

| | | | |
|---|---|---|---|
| Instruction: | Swop | | |
| Effect: | The top two items on the stack are reversed. That is, TOS becomes the new SOS, and SOS becomes the new TOS. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two items. | | |
| Example: | X = Y | Stack Y; Stack X; Swop; Assign-Value | |

| | | | |
|---|---|---|---|
| Instruction: | Test-Boolean | | |
| Effect: | The condition-code is set to 'false' or 'true' depending on whether TOS is 0 (false) or 1 (true). TOS is removed from the stack. | | |
| Notes: | | | |
| Error: | 1. The stack is empty.<br>2. TOS is not a boolean value. | | |
| Example: | If B Then DoItNow; | Stack B; Test-Boolean; BF 43<br>Stack DoItNow; Call<br>Label 43 | |

| | | | |
|---|---|---|---|
| Instruction: | Test-In | | |
| Effect: | The value of SOS is tested for inclusion within the set TOS. The condition-code is set 'true' or 'false' accordingly. Both TOS and SOS are removed from the stack. | | |
| Notes: | | | |
| Error: | 1. The stack contains less than two items.<br>2. SOS is not an integer value.<br>3. TOS is not a set value. | | |
| Example: | If NOT x IN s Then x := 0; | Stack x; Stack s; Test-In<br>Bt 31<br>Stack x; Byte 0; Assign-Value<br>Label 31 | |

| | | | |
|---|---|---|---|
| Instruction: | Test-Nil | | |
| Effect: | A check is performed to ensure that TOS is not NIL. An event is signalled if it is, or if TOS points to a heap item which has been returned to the heap (disposed). | | |
| Notes: | This test can also perform an unassigned variable check. | | |
| Error: | 1. The stack is empty.<br>2. TOS is not a pointer variable. | | |
| Example: | P^ := 0; | Stack P; Test-Nil<br>Reference <1><br>Byte 0; Assign-Value | |

| | | | |
|---|---|---|---|
| Instruction: | Test-Range <tag> | | |
| Effect: | The value of TOS is checked to be within the range defined by the tag. If the value is not in the range an event is signalled. | | |
| Notes: | The event is signalled at run-time. | | |
| Error: | 1. The stack is empty.<br>2. TOS is not an integer value.<br>3. <tag> does not define a range. | | |
| Example: | Byteval := Bigval; | Stack Byteval<br>Stack Bigval; Test-Range Byterange<br>Assign-Value | |

| | | | |
|---|---|---|---|
| Instruction: | Trunc | | |
| Effect: | TOS is replaced by the value TRUNC(TOS) where TRUNC is as defined in section 6.6.6.3 of the Pascal standard BS 6192:1982, with the extension that Trunc returns the value of its parameter if that value is already an integer. | | |
| Notes: | | | |
| Error: | 1. The stack is empty.<br>2. TOS is neither integer nor real. | | |
| Example: | i := Trunc(r+0.1); | Stack i; Stack r; Real <0.1>; Add<br>Trunc; Assign-Value | |

| Instruction: | Variable-Call | | |
|---|---|---|---|
| Effect: | The procedure described by TOS is called. This differs from 'Call' in that the procedure may have a variable number of parameters. | | |
| Notes: | This instruction is used to call 'C' procedures and its definition is as woolly as the definition of that language. | | |
| Error: | 1. The stack is empty.<br>2. TOS is not a procedure descriptor. | | |
| Example: | try(1); try(1,2); | Stack try<br>Byte 1; Assign-Parameter<br>Variable-Call<br><br>Stack try;<br>Byte 1; Assign-Parameter<br>Byte 2; Assign-Parameter<br>Variable-Call | |

# Appendix 1. Instructions which set the condition code

Call {predicate}

Compare-Values

Compare-Unsigned-Values

Compare-References

Compare-Repeated-Values

Test-Boolean

Test-In

# Appendix 2. Instructions which test the condition code

JEQ

JZ

JGE

JGT

JLE

JLT

JNE

JNZ