1966

*1: internal specifications

Earley

_Carnegie Mellon University_

Follow this and additional works at: http://repository.cmu.edu/comptsci
NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making
of photocopies or other reproductions of copyrighted material. Any copying of this
document without permission of its author may be prohibited by law.
INTERNAL SPECIFICATIONS

Jay Earley
This paper describes the internal specifications of *1. It is intended that it be detailed enough so that it could be used by someone learning to maintain the system, but organized in such a way that someone who wants to know only enough to use some part of the meta-language can extract what he wants without too much trouble.

The language consists entirely of a set of macros. It is therefore coded completely in the 360 macro language . A knowledge of this language may or may not be necessary, depending on one's use of these specifications. This paper consists of descriptions of the function of each macro and descriptions of the use of each global SET symbol.

First we shall present one *1 statement and the code it produces as an example. Assume that the following definitions have been made:

```
BASFIELD M
BASFIELD N
FIELD A, 1, (16, 31)
FIELD B, 2, (0, 15)
```

Then this statement

```
DO (MB,<r <s NBA)
```

produces the following code:

```
Code       Macro Produced By
1. L 3, N  LINK
2. L 3, 8(3)  SEQ
3. N 3, B1  RIGHT
4. SRL 3, BS
5. L 3, 4(3)  BINOP
6. N 3, A1  OP
7. L 4, M
8. SLL 3, 16  STORE
9. L 5, 8(4)
10. N 5, BO
11. O 3, 5
12. ST 3, 8(4)
```

Explanation: (1) loads the contents of base field N into register 3; (2) loads the word containing the B-field of the block pointed to by N; (3) extracts the

* IBM System /360 Basic Operating System Language Specifications, Assembler (16K Disc/Tape)
field contents; Bl is a mask which has 1's in the bit positions of field B and 0's elsewhere; (4) shifts it over so that it can be used as a pointer; BS is a constant which is the number of bits that the right end of field B is away from the right end of its word. These 3 have accomplished linking through that field. (5) and (6) access the A-field of the block presently pointed to, but do no shifting. This will come later if necessary. (7) loads base field M. (8) now realizes that the quantity NBA must be at the same position in its register as a B-field, so it shifts register 3 over to get it to that position. (9) loads the word containing the B-field of the block pointed to by M into a new register. Register 4 is needed for later use. Since we are storing into this B-field, we must first destroy its old contents. (10) does this since BO is a mask with 0's in the bit positions of field B and 1's elsewhere. We can now OR in the quantity we want to store with (11) and store it back in memory from whence it came with (12).
MACRO DESCRIPTIONS

Each macro of the *1 system is described in the following format:

<table>
<thead>
<tr>
<th>Macro</th>
<th>The name of the macro.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called by</td>
<td>A simple list of the macros which call the macro.</td>
</tr>
<tr>
<td>Inputs</td>
<td>All the parameters of the macro will be listed with explanations of what they are. Some of the globals used in the macro which function explicitly as inputs will also be explained. The absence of a global from this part or from output doesn't indicate that it definitely could not be considered an input or output of the macro.</td>
</tr>
<tr>
<td>Macros called</td>
<td>A list of all macros called by the macro. Some will have explanations beside them. The absence of an explanation means that either the use of the macro can be understood from reading its description or it will be explained below under Action.</td>
</tr>
<tr>
<td>Outputs</td>
<td>A list similar to Inputs.</td>
</tr>
<tr>
<td>Code*</td>
<td>A look at the code produced and macros called by the macro in order and with parameters. Some explanation will be given here, but usually the code and the reason for it is explained under Action. The use of any globals shown here or in a listing of the macros may be found by consulting the list of global descriptions.</td>
</tr>
<tr>
<td>Action</td>
<td>This explains what the macro accomplishes and how.</td>
</tr>
</tbody>
</table>

Since each of the macros which is affected by the meta-language is affected in the same way, this explanation is made separately of these macros. It is under the heading META-LANGUAGE even though this is not a macro.

*May not appear in some descriptions.
<table>
<thead>
<tr>
<th>Macro</th>
<th>ACCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called by</td>
<td>LINK, RIGHT</td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LINK, RIGHT</td>
</tr>
<tr>
<td>Globals</td>
<td>OLD</td>
</tr>
<tr>
<td></td>
<td>NEW</td>
</tr>
<tr>
<td></td>
<td>C</td>
</tr>
<tr>
<td>Macros called</td>
<td>SETUP</td>
</tr>
<tr>
<td>Outputs</td>
<td>NEW</td>
</tr>
<tr>
<td>Action</td>
<td>Code:</td>
</tr>
<tr>
<td></td>
<td>L NEW, FG</td>
</tr>
<tr>
<td></td>
<td>N NEW, CI</td>
</tr>
</tbody>
</table>

It accesses the field contents, but does not shift to the right side of their register.

* Affected by Meta-language
Macro
Called by ADDD
Inputs BINOP
Macros called None
Outputs EASY with parameter "A", to produce the code to do addition.
Action

Macro
Called by ADDX
Inputs BSTORE
Macros called None
Parameters RO A register to which RX may be added.
Outputs None
Action If RX = 0, we do nothing. Otherwise we code
AR RO, RX and we set RX to 0. This is because we are about to use an instruction which cannot name double indexing, so if we have an index register, we must add it to the one base register we are using, RO.
Macro
Called by AND
BINOP
Inputs Globals
RR,EX,F,G
Macros called SHIF,STOR
Outputs None
Action Code:
SHIF
0 RR,CO
N RR,FG
STOR (with NEW=RR,EX=FALSE)

First we place 1's all around the quantity and then we AND it directly into the word containing the field. Then we put it back in memory.

Macro BASBLOCK
Called by Source Language
Inputs Parameters
B The name of the block
BOUNDS The lower bound and size of the block.
Macros called BLOCK Block is called with the same parameters, except that BASE=1.
Outputs None
Action Sets up compile-time information about the block.
Macro BASFIELD
Called by Source Language
Inputs Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUG</td>
<td>The name of the base field.</td>
</tr>
<tr>
<td>LOC</td>
<td>Its location.</td>
</tr>
</tbody>
</table>

Macros called None

Outputs BUGS This is the column of the bug table which contains the bug names.

Globals CBLA

<table>
<thead>
<tr>
<th>Global</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Index to most recent storage location for base fields.</td>
</tr>
<tr>
<td>TB</td>
<td>Index to bug table.</td>
</tr>
</tbody>
</table>

Action BUG is set equal to LOC if it is passed or to the storage location (indexed by B) if it is not.
Macro

Called by

Inputs

Macros called

Parameters

Code

Action

<table>
<thead>
<tr>
<th>Macro</th>
<th>BINDEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called by</td>
<td>BINOP</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The field on the right which contains a binary number.

The field on the left which will contain decimal digits.

RIGHT, TABLEF

ADDX

To check for adding RX to OLD since UNPK cannot use double indexing.

SETUP

To set up F for UNPK.

RIGHT, F2

SRL, RR, POS

CVD, RR, DOUB

SEQ, F1

UNPK, INC(L, OLD), DOUB

(where INC has been set to access the first byte of F2 and L = length of F1 in bytes)

We calculate F2 right justified in RR. We convert it to decimal into DOUB. Then we calculate F1 and unpack DOUB into the field that it points to.
### BINOP

**Called by**: OP, TEST

**Parameters**
- **SEQ**: The sequence on the left.
- **OP**: The operator.
- **QUAN**: The quantity on the right.

**Macros called**
- **BINDEC, DECBIN**: If the operator is D ← B or B ← D, it calls those immediately.
- **DO**: If the operation is push or pop, it calls DO with the appropriate arrangement of operands to perform the operation.
- **RIGHT**: Otherwise RIGHT is called with QUAN as a parameter. This calculates the quantity on the right.
- **LEFT**: Then LEFT is called to calculate the sequence on the left except for its last element.
- **STORE, BSTORE, DIV, MUL, ADD, ORR, AND, XOR, GT, LT, EQ, BEQ**: Then depending on the operator, the appropriate macro is called to execute the operation or test indicated.

**Outputs**: None

**Action**: Codes the operation or test passed to it.
Macro
Called by
SEQ, RIGHT, LEFT
BLINK
Inputs
Globals
C
The block through which we are linking.
Macros called
TABLEBK, RIGHT, RET
Outputs
See Action
Action
1. If C is a static base block we code
   LA NEW, C.
2. If C is a static block we increment INC by
   C's lower bound.
3. If C is a dynamic block, we push down a number
   of globals used in RIGHT, then we call RIGHT
   on the lower bound of C. This leaves the
   increment we want in RR. Then,
   a. If C is a base block we just set OLD to
      be RR so it will link through that next.
   b. If RX is zero, we set RX to RR so that
      RR will be used as an index register
      in the next access.
   c. If RX already has an index register, we
      code AR RX, RR.
Then we pop back all the quantities that we pushed.
Macro
Called by
Inputs
Parameters

<table>
<thead>
<tr>
<th>Macro called</th>
<th>Outputs</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>BLOCK</th>
<th>Source Language, BASEBLOCK</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>The name of the block.</td>
</tr>
<tr>
<td>BOUNDS</td>
<td>The lower bounds and size of the block.</td>
</tr>
<tr>
<td>BASE</td>
<td>= 1 if its a base block, = 0 otherwise.</td>
</tr>
<tr>
<td>TESTSEQ</td>
<td>To test the lower bound to see whether it is dynamic.</td>
</tr>
<tr>
<td>BLKS, LB, DLB, SIZ</td>
<td>Each of these is a column in the table of blocks. BLKS contains the block name, LB the lower bound if it is a static field and not a base field, DLB the lower-bound if it is dynamic, and SIZ the size.</td>
</tr>
</tbody>
</table>

Action
If it is a static base block, the block name is set equal to the lower bound.
Macro  BRAKTEST
Called by  SEQ
Inputs
Parameters
STRING  A sequence which is being linked through.

Globals
I  The index of a character in the sequence.
C  That character.

Outputs
Globals
I  (The index of the next name in the sequence) - 1.
C  The name of the sequence which begins with the input C.

Action
If C is not "]", it does nothing. If it is, it stores into C all the characters between the "[" and the next "]" and sets I to be the index of the "]".
Macro: BSTORE
Called by: BINOP
Inputs: None
Macros called: TABLEBK, ADDX, GPAIR, SETG, CLEAR
Outputs: None
Code:

If the right side is a quantity,

\[
\text{ST} \quad \text{RR,INC,G}
\]
\[
\text{CLEAR} \quad (\text{INC} = \text{INC} + 4, \text{SIZE} = \text{size of left block} - 4)
\]

If the right side is a Z word block,

\[
\text{LM} \quad \text{OLD, OLD+1, INC(RR)}
\]
\[
\text{STM} \quad \text{OLD, OLD+1, INC(OLD)}
\]
\[
\text{CLEAR} \quad (\text{INC} = \text{INC} + 8, \text{SIZE} = \text{size of left block} - 8)
\]

Otherwise

\[
\text{MVC} \quad \text{INC(L, OLD), INC(RR)}
\]
\[
\text{CLEAR} \quad (\text{INC} = \text{INC} + L, \text{SIZE} = \text{size of left block} - L)
\]

where \( L = \text{size of right block} \).

Action: We do the store in one of three ways; then we set up the inputs for CLEAR, which zeroes the rest of the left block.
Macro

Called by

Inputs

Parameters

Macros called

Outputs

Code

CALL
UNOP

DEST  A quantity which supplies the address of the subroutine.

DO, RIGHT

None

DO, RIGHT

DO ([MS], +, 1)
DO ([MS], +, 1)

DO ([MS][FO], +0, *+N_1)
RIGHT DEST

B DEST

DO ([MS][FO], +0, *+N_2)
BR RR

(where MS is the pointer to the mark stack for subroutines, FO is the 0th full word in its block, N_1 and N_2 are constants adjusted so that the address of the next command after the branch is stored in the mark stack)

Action: We push down the mark stack, put in it the return address and go to the beginning of the subroutine.
Macro
Called by
Source Language, PRIORITY

Inputs
Parameters
MACROS A list of macros to be substituted for when called.

THINGS A list of fields, blocks, etc. for which the above macros are altered.

TAG A string designating this particular call on CHANGE.

Macros called
ENTER, ENTERB To make entries in the change table for fields, and for base fields.

Outputs
For each combination of a macro and a field or block, TAG is entered in the spot in the change table corresponding to that field or block and macro. i.e. If LINK and Field A, where A is the 3rd field, then LINK(3) <- TAG. If ACCESS and base field B, where B is the 2nd base field, then ACCESSB(2) <- TAG. If "FIELDS" or "BFIELDS" occurs, all entries in that column are made.

Action
See Output
Macro
Called by
UNOP
Inputs
SEQ
The sequence defining the field to be complemented.
Macros called
LEFT, GREB, STOR
Outputs
None
Action:
We get the word containing the field into R and exclusive or with a mask containing 1's in the bit positions of the field.

---

Macro
Called by
BSTORE
Inputs
SIZE
The size of the block to be cleared.
Macros called
GREG
Outputs
None
Code:
SR R, R
ST R, INC, G
ST R, [INC+4], G
Action: First we set R to zero and then store it into each word of the block with a succession of ST statements.

---

Macro
Called by
UNOP
Inputs
SEQ
The sequence defining the field to be complemented.
Macros called
LEFT, GREB, STOR
Outputs
None
Action:
Code:
LEFT
L R, FG
X R, CI
STOR (with NEW=R, EX=0)
We get the word containing the field into R and exclusive or with a mask containing 1's in the bit positions of the field.
Macro

Called by

Inputs

Parameters

F2  The field on the right which contains decimal digits.
F1  The field on the left which will get a binary number.

Macros called

SEQ, TABLEF, GREG, STORE
SETUP  To set up F for the PACK instruction.
ADDX  To check for adding RX to OLD since PACK cannot use double indexing.

Code

SEQ  F2
PACE DOUB, INC(L, OLD)
CVB  RR, DOUB
LEFT  F1  (pos = 0)
STORE
(where INC has been set up to access the first byte of F2 and L = length of right field in bytes)

Action: We calculate F2 and pack it in a double word DOUB. We then convert it to binary into register RR, and from there store it with the left field.
Macro

Called by

Inputs

Macros called

Output

Code For input:

LAB

B

ON

LAB

DIFCD

TYPEFILE = INPUT,

RECFORM = FIXUNB,

BLKSIZE = 80

DEVICE = 1442

DEVADDR = SYSRDR,

EOFADDR = EA

IOAREAI = BLK

For output:

LAB

DIFPR

RECFORM = FIXUNB,

BLKSIZE = 120,

DEVADDR = SYSLST,

DEVICE = 1443,

IOAREAI = BLK

Action: The indicated I/O block is defined and opened.
Macro
Called by
Inputs
Macros called
Parameters
OP =/ if quotient is wanted or = MOD if remainder.
Outputs
Code:

Let R1 and R2 are a pair of contiguous registers, R1 is even, and R is a third register

SHIF
L R2, FG
SR R1, R1
LR R, R1
N R2, CI
DR R1, RR MOD
SLL R2, CS
STOR (OLD = R, NEW = R1)
STOR (OLD = R, NEW = R2)

Action: We shift the quantity in RR to match the position of the left quantity. We load the word containing the left field into R2 and zero R1. We copy the word into R. Then we extract the field contents in R2, and divide the pair by RR. This leaves the quotient in R2 and the remainder in R1. The quotient will be right justified, however, so in that case we shift it left to match the field position and then store it. The remainder is already in the correct position, so in case of "MOD", we just store it.
Macro
Called by Source Language, BINOP, CALL, RETURN
Inputs Parameters
Pl,...,P8 Parenthesized operations of
source language
Macros called OF Called once with each operation
as a parameter.
Outputs None
Action: Codes each operation.

Macro EASY
Called by ADDD, SUB, ORR, XOR
Inputs Parameters
OP A letter indicating the operation
to be performed (A,S,O,X)
Macros called SHIF, STOR
Outputs None
Code
OP RR, FG
STOR (NEW=RR, EX=0)
Action: The quantity on the right is shifted over so that it matches the
right side of the left field. Then it is added (or subtracted,
etc.) right into the word containing the field. The result is
then stored back into memory.
Macro

Called by

Inputs

Parameters

MACROS  A list of macros in whose columns, entries are to be made.
TAG     A string which is the entry which is to be made.

Globals

T1     The index of the entries which are to be made.

Macros called

Outputs

None

The above entries are made for columns LINK, STOR, STORE, or ACCESS

Macro

ENTERB

Same as ENTER except that the column names all have B's after them.
<table>
<thead>
<tr>
<th>Macro</th>
<th>EQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called by</td>
<td>BINOP</td>
</tr>
<tr>
<td>Inputs</td>
<td>Parameters</td>
</tr>
<tr>
<td></td>
<td>OP</td>
</tr>
<tr>
<td>Macros called</td>
<td>REL</td>
</tr>
<tr>
<td></td>
<td>NOTANY</td>
</tr>
<tr>
<td>Outputs</td>
<td>None</td>
</tr>
</tbody>
</table>

**Action:** Produces code to make the test.
Called by
Inputs
Macros called
Output
Code

Macro

FIELD
Source Language
Parameters
FD The name of the field.
N The word of the block in which it resides.
BITS The beginning and ending bits of the field.
GREG,RRESS To get and return registers used for computing the mask.
FLDS, WORD, WID, SHIF, EXT

Each of these is a column in the table of fields. FLDS contains the field name, WORD the number of bytes to be incremented to access the correct word of the block, WID the width of the field in bits, SHIF the number of bits the field is from the right side of its word, EXT=1 if the field is not a full word so that extracting is necessary.

Let R and R1 be two registers, LD and RD be the distance of the field from the left and right hand side of its word in bytes. ONES=X'FFFFFFFF'.

The code:

L R, ONES
LR R1, R
SRL R, LD
SLL R1, RD

This produces a mask with 1's in the bits of the field in R.
Macro FIELD - cont'd.

```
NR  R, R1
LR  R1, R
X   R1, ONES
```

This puts its complement in R1

**Action:** This code computes the two marks.

They then are stored in the next two available storage locations for masks (indexed by M).

Let F be the field name, F1 = the ones mask, FO = the zeroes mask.

Also FS = the entry in SHIF.
### Macro: FINAL

**Called by:** Source Language

**Inputs:** None

**Macros Called:**
- CLOSE: To close each of the I/O blocks.
- EOJ: To end the run.

**Outputs:** None

**Code:**
```
CLOSE  LAB  (For each I/O block)
EA     EOJ
```

**Action:** EA is the end-at-file address for I/O blocks for which the programmer does not specify one.

---

### Macro: GOTO

**Called by:** UNOP

**Inputs:**
- PARAMETERS
  - DEST: The quantity which specifies the address to go to.

**Macros Called:** RIGHT

**Outputs:** None

**Code:**
```
If a quantity
B  DEST
If a sequence
RIGHT  DEST
BR   RR
```

**Action:** We compile a branch to the given destination. If it is a sequence we get it into RR and then branch to it.
Macro GPAIR

Called by MULT, DIV, BSTORE

Inputs None

Macros called None

Outputs GBLA

R Number of the first register of the pair.

Globals REG Register column.

Action: Finds the first pair of contiguous unused registers starting with an even register. They are both set to 1 to show that they are in use.

Macro GREG

Called by Many things

Inputs None

Macros called None

Outputs R Number of register found

Action: The first unused register (REG(R) = 0) is found and its entry is set to 1, showing that it is now in use.

Macro GT

Same as EQ except parameter is H for high.
<table>
<thead>
<tr>
<th>Macro</th>
<th>IF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called by</td>
<td>Source Language, IFANY</td>
</tr>
<tr>
<td>Inputs</td>
<td>Parameters</td>
</tr>
<tr>
<td>P1,...,P8</td>
<td>A number of tests followed by THEN followed by a number of operations.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GBLB</th>
<th>ANY</th>
</tr>
</thead>
<tbody>
<tr>
<td>=0 if called from source language</td>
<td></td>
</tr>
<tr>
<td>=1 if called by IFANY</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Macros called</th>
<th>TEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called with each test as parameter.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Macros called</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called with each operation as parameter.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Macros called</th>
<th>RREGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called after each call of TEST or OP to mark. The registers they had used as unused.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th>None</th>
</tr>
</thead>
</table>

**Action:** First it produces the code for the tests. These will produce branches to OUT on false if ANY = 1 and branches to IN on true if ANY = 0. Then it produces the code for the operations with the appropriate labels. For example, with 2 tests and 2 operations we get

```
IF
  Test
  Branch on False to OUT
  Test
  Branch on False to OUT
  Operation
  Operation
  IN
OUT
NOR
```

```
IF ANY
  Test
  Branch on true to IN
  Test
  Branch on true to IN
  Branch to OUT
  Operation
  Operation
  IN
OUT
NOR
```
Macro
Called by
Inputs
Macros called
Outputs
Action:

IFANY
Source Language
Parameters
P1,...,P8
A number of tests followed by
THEN followed by a number of
operations.

IF
With same parameters as IFANY,
except that ANY = 1.

None

Produces code to execute the operation if any of the tests are true.

IN
Source Language
Parameters
BLK
The name of the block into which
a card is to be read.

TABLEL
To look up the block name and
set its DTF label.

GET
To input the card.

None

The next card is read into block BLK.
Macro
Called by
Inputs
Macros called
Outputs
Code
Action:

INITIAL

Source Language

None

BASFIELD, FIELD

None

BEGIN
BALR 2,0
USING *,2
B ON
ONES DC 12C'
DOUB DS 1D
ON

and others

Initial declares certain storage areas, standard constants, and standard fields and/or base fields.

Macro
Called by
Inputs
Macros called
Output
Action:

INPUT

Source Language, OUTPUT

Parameters

P1,...,P8 Input blocks.

DEF With each of the parameters in turn.

Globals

OUT Is reset to False

DEF compiles the DTF for each input block.
### Macro: LEFT

**Called by:** BINOP, SHIFT, COMP

**Inputs:**

**Parameters**

SEQ: The sequence to be linked through in preparation for an operation or test.

**Macros called:**

GREG, TABLEF, INBLEB

**Outputs:**

GBLB

EX: =1 if the program should extract in working with the last field.

F and G: Contain the first and second parts of the character string needed to access the last field. See Setup.

BKL: =True if the last name in the sequence is a block.

C: has the last name in the sequence.

**Action:** Links through the sequence and leaves the outputs described above.
Macro | LINK*, SEQ
Called by | None
Inputs | Parameters
Globals | None

- **OLD**: A register pointing to a block.
- **NEW**: A register which will point to a new block obtained by linking through the C field of the old block.

**Macros called** | ACCESS
**Globals** | None

**Outputs** | ACCESS
**Globals** | None

**Code** | SRL
**NEW, CS**

We get the word of the field into NEW, extract it, and shift it. Then we set OLD to NEW.

**Macro** | LT

Same as EQ except parameter is L for LOW.

*Affected by Meta-Language*
META-LANGUAGE

All macros which are affected by the meta-language are affected in the following way:

Let the macro be MAC. If MAC(TI) is not null, we call CHANGES with parameter MAC(TI). Otherwise the macro is processed normally. MACB is used if the macro is processing a base field instead of a field.

However, before this check is made, we check CH. If this is true, it means that we have already called CHANGES from this macro and that CHANGES has called it back again. In this case we set CH to False and then process the macro normally. Thus, if the meta-language programmer wants CHANGES to call the same macro which called it, he must set CH to true first.
Macro
Called by BINOP
Inputs None
Macros called SHIF, GREG, STOR
GPAIR To get a pair of registers for the multiplications
Outputs None
Code
Let R1 and R2 are a contiguous pair of registers, R1 is even, and R is a third register.

```
SHIF
L R2, FG
LR R, R2
N R2, CI
NR R1, RR
SRDL R1, CS
STOR (OLD=R, NEW=R2)
```

Action: We shift the quantity in RR to match the position of the left quantity. We load the word containing the left field into R2 and R. We extract the field contents in R2 and multiply the pair by RR. Because it is multiplication, the result will be shifted over to the left twice as far as it should be for the position of the field. So we shift it back to the right. The shift is double since it might extend over both registers. We then store it back into R and put that back in memory.
Macro
Called by
Inputs
Parameters
Macros called
Outputs
Globals
Action

<table>
<thead>
<tr>
<th>Macro</th>
<th>NOTANY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called by</td>
<td>TEST, EQ, GT, LT</td>
</tr>
<tr>
<td>Inputs</td>
<td>OP</td>
</tr>
<tr>
<td>Macros called</td>
<td>None</td>
</tr>
<tr>
<td>Outputs</td>
<td>LAB</td>
</tr>
<tr>
<td>Action: Sets LAB and K.</td>
<td></td>
</tr>
</tbody>
</table>

An operator which may or may not start with -.

- OUT if called from IF, = IN otherwise, the label is followed by an integer (IO) to make it unique from other statements.

= 2 if OP starts with "-", = 1 otherwise.

Macro
Called by
Inputs
Parameters
Macros called
Outputs
Action

<table>
<thead>
<tr>
<th>Macro</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called by</td>
<td>DO, IF</td>
</tr>
<tr>
<td>Inputs</td>
<td>OPER</td>
</tr>
<tr>
<td>Macros called</td>
<td>RETURN</td>
</tr>
<tr>
<td>Outputs</td>
<td>None</td>
</tr>
<tr>
<td>Action: Codes the operation.</td>
<td></td>
</tr>
</tbody>
</table>

A parenthesized operation.

With no parameter in case the operation has only one member.

With the 2 members of the operation as parameters if the operation has 2 members.

As with UNOP if the operation has 3 members.
Macro OUT
Same as IN except that PUT is called to print a line.

Macro OUTPUT
Called by Source Language
Inputs P1,...,P8 Output Blocks.
Macros called Input
With same parameters except that OUT is set True.

Action: The DTF's for the blocks are compiled.
**Macro**

**Called by**

**Source Language**

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUG</td>
<td>The name of the bug which is to be made a priority bug.</td>
</tr>
<tr>
<td>SW</td>
<td>OFF if the priority of the bug is to be turned off rather than on.</td>
</tr>
</tbody>
</table>

**Macros called**

CHANGE, GREG, TABLEB

SAVE To save the register which will hold the bug from being returned by RREGS.

RET To return this register when we are finished with it.

**Outputs**

The entry for the bug in the PRIOR column of the bug table is set to the number of the register that the bug is loaded into.

**Code**

```plaintext```
if SW ≠ OFF
L R, BUG
CHANGE (ACCESS), (BUG), A
CHANGE (LINK), (BUG), L
CHANGE (STORE), (BUG), ST

if SW = OFF
ST R, BUG
CHANGE (ACCESS, LINK, STORE), (BUG), OFF
```

**Action:** The changes which one setup here must then be carried out by calling the macro "PRIORS" from CHANGES.
Macro: PRIORS
Called by: CHANGES
Inputs: Parameters
TAG: The tag that was passed to CHANGES.
PRIO(TI): The register which the priority bug now resides in.

Macros called: RET
Outputs: See Action

Action: If TAG = A, we return the register R which was just gotten, and set NEW = PRIO(TI), because NEW is the output register for an access. If TAG = L, we just set OLD = PRIO(TI) because OLD is the output register for a link.
If TAG = ST, we code
LR PRIO(TI), RR
This stores the quantity into the bug.

Macro: RREGS
Called by: DO, IF
Inputs: None
Macros called: None
Outputs: None
Action: Sets all entries to 0 which are 1 in the register column. It is marking these registers unused.
Macro REL
Called by EQ, GT, LT
Inputs Parameters OP =E for equality, =H for >, =L for <
Macros called ACCESS
Outputs None
Code ACCESS
SHIF CR NEW, RR
B(N)OP LAB
Action: Right has already left a quantity in RR. We access the contents of the field at the end of the left sequence. We shift the right one over to match it. We compare them. We branch depending on various things to the appropriate label (See NOTANY, IF).

Macro RETURN
Called by OP
Input None
Macros called DO
Outputs None
Code DO ([MS], -, 1)
DO (GOTO, [MS][F1])
(where MS is a pointer to the mark stack and F1 is the 1st (not 0th) full word in its block).
Action: We pop the mark stack, and then go to the address we just popped.
<table>
<thead>
<tr>
<th>Macro called</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Called by</strong></td>
<td><strong>RIGHT</strong></td>
</tr>
<tr>
<td><strong>Inputs</strong></td>
<td><strong>BINOP, CALL, BINDEC, GOTO, SHIFT, TEST, SEQ</strong></td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td><strong>QUAN</strong> The quantity to be evaluated. <strong>GREG, TABLEF, TABLEB</strong></td>
</tr>
<tr>
<td><strong>Macros called</strong></td>
<td><strong>SEQ</strong> The link through the sequence if there is one, except for the last element. <strong>ACCESS</strong> To access the last element of the sequence. <strong>TESTSEQ</strong> To test whether QUAN is a sequence. <strong>Globals</strong></td>
</tr>
</tbody>
</table>
| **Outputs** | **RR** If the parameter is a quantity or field sequence, it is left in register RR. If a block sequence, RR points to the block which contains the last block; If a base block, RR = 0. **POS** This is the number of bits from the right side of the register that this quantity lies. **INCR** This is the current increment to be used in displacements for accessing the block on the right. It is only used if we have a block sequence. **RXR** The current index register to be used in accessing the block on the right if it is a block sequence. **BKR** Contains the block name if QUAN is a block sequence.
Macro RIGHT - cont'd.

For a three name field sequence

    LINK
    LINK
    ACCESS

Action: If the quantity is an assembler expression, that is loaded in a register. If it is a base field, that is loaded in a register. If it is a sequence, it links through the sequence and then if it ends in a field the final value goes in a register. If it ends in a block, RR will contain a pointer to the containing block and BKR contains the ending block name. Otherwise BKR will be null.
<table>
<thead>
<tr>
<th>Macros called</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>R11</td>
</tr>
<tr>
<td></td>
<td>The number of a register</td>
</tr>
</tbody>
</table>

**Action:** That entry corresponding to the register in the register column is set to 2, so that it will not be set to 0 by RREGS.
Macro

**SEQ**

**Called by**

RIGHT, LEFT, DECBIN

**Inputs**

**Parameters**

STRING

A string of names to be linked through.

**Macros called**

GREG, TABLEB, TABLEBK

BRAKTEST

It checks if the next name in the sequence is bracketed, and if so, puts the name in C.

LINK

Links from the block pointed to by OLD through field C and points to the new block with register NEW.

BLINK

To link through a block.

**Outputs**

**Globals**

ONLY

=TRUE if the sequence consisted of a base field only.

C

The last name of the sequence.

**Action:** It links through all names in the sequence except the last. For each field or base field name it calls LINK, and for each block or base block name it calls BLINK.
Macro: \text{SETG}

Called by: SETUP, BSTORE

Inputs: None

Macros called: None

Outputs: None

Globals: G

Action: Sets G as described in SETUP.

---

Macro: \text{SETUP}

Called by: LEFT, ACCESS, BINDEC, DECBIN

Inputs: None

Macros called: \text{SETG}

Outputs: F, G

Globals: To set G, see below.

Action: Besides setting F and G, it zeroes INC and RX after using them to set up F and G.
<table>
<thead>
<tr>
<th>Macro</th>
<th>SHIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called by</td>
<td>STORE, EASY, AND, MULT, DIV, BLINK</td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td>None</td>
</tr>
<tr>
<td>Action:</td>
<td>Code is produced to shift RR to the position of SHIF(TI)</td>
</tr>
</tbody>
</table>

**Globals**

- **POS**  The position of the quantity on the right in RR.
- **ONLY**  =TRUE if the quantity came from a base field.
- **SHIF(TI)**  The position of the quantity on the left in its word.
Macro

Called by

Parameters

SIZE A quantity representing the amount of the shift.
SEQ A field sequence which is to be shifted.

Globals

DIR =R if the shift is right, = L otherwise.

Macros called

RIGHT, LEFT, GREG, STOR

Outputs

None

Code

RIGHT SIZE
SRL RR, POS
LEFT SEQ
L R1, FC
LR R2, R1
N R1, CI
S(DIR)L R1, 0(RR)
STOR (OLD = R2, NEW = R1)

Action: The size is calculated and put right justified in RR. Then the word containing the field to be shifted is put into R1 and R2. The field contents are extracted and shifted using R1 and then they are stored into the copy in R2 and put back in memory.
**Macro**

**Called by**

**Inputs**

**Macros called**

**Outputs**

**Code**

**Globals**

**Action:** We destroy the contents of the field in OLD. Then we OR the quantity and store it back into memory. IF EX = FALSE only the store is done, so this is used often to do just the store.
Macro

Called by

Inputs

Actions

Output

Code

Macros called

GREG, SHIF, STOR

Outputs

None

If the right side is a quantity,

SHIF

L   R, FG

STOR (OLD=R, NEW=RR)

If the right side is a block,

LA   RR, INCR(RR,RX) [RX may be absent]

STORE (POS=0)

* Affected by meta-language
### Macro: SUB

Same as ADDD, except that the parameter is \( S \).

### Macro: TABLEB

Same as TABLEF except it works on the bug table.

### Macro: TABLEBK

Same as TABLEF except it works on the block table.

<table>
<thead>
<tr>
<th>Macro</th>
<th>TABLEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called by</td>
<td>Many things</td>
</tr>
<tr>
<td>Inputs</td>
<td>Parameter</td>
</tr>
<tr>
<td></td>
<td>FD</td>
</tr>
<tr>
<td>Macros called</td>
<td>None</td>
</tr>
<tr>
<td>Outputs</td>
<td>Globals</td>
</tr>
<tr>
<td></td>
<td>FOUND</td>
</tr>
<tr>
<td></td>
<td>Tl</td>
</tr>
</tbody>
</table>

**Action:** Looks up the name in the table.

### Macro: TABLEL

Same as TABLEF except that it works on the label table for I/O.
Macro
Called by
Inputs
Macros called

TEST
IF

Parameter
OPER A parenthesized test to be coded.
BINOP If the test is binary, then BINOP is called with the 3 parts of the test as its parameters.
NOTANY Is called in the case of a test on just a sequence. This puts the correct label in LAB and sets K to 1 or 2 depending on whether there is a \( \sim \) preceding the sequence.
RIGHT Is called with the sequence as parameter.

Code
After calling these macros, let R be a register, then
\[
\begin{align*}
\text{SR} & \quad R, R \\
\text{CLR} & \quad RR, R
\end{align*}
\]
only produced if no extractor was used to get the quantity in RR.
\[
B(N)Z \quad \text{LAB}
\]
(where N depends on \( \sim \) and on whether called from IF or IFANY and LAB depends on latter)

Action: We set the contents of the field into RR, if necessary we compare it with zero, and then we branch according to conditions (see IF).
Macro
Called by
Inputs
Macros called
Outputs
Action:

TESTSEQ
BLOCK, RIGHT

UNOP

STRING
A string which may be a sequence.

OP

BRAKTEST, TABLEF, TABLEB, TABLEBK

THING

COMP
With parameter THING if the operator
is "-1" to perform the complement
operation.

GOTO
With parameter THING, if the operator
is "GOTO".

SHIFT
With parameters
(1) The rest of OP, excluding the
\rightarrow or \leftarrow
(2) THING
and with DIR=\rightarrow for \rightarrow, L for \leftarrow.

CALL
With parameter THING if operator is
"CALL".

None

Perform the unary operation.
Macro XOR

Same as ADDD, except that the parameter is X.
GLOBAL DESCRIPTIONS

GBLA

I

See BRAKTEST.

INC

The displacement which is to be used in the next access of a field within a particular sequence.

IO

A counter used to get unique names for the IN and OUT labels used in IF and IFANY.

LB

See BLOCK.

M

The index of the address of the most recently stored mask.

NEW,OLD

See LINK, ACCESS, or STOR.

POS

See RIGHT.

R

The output register of GREG and GPAIR.

R1, R2

Registers.

REC

The column of registers. It contains a 0 in the entry with index of a register that is not in use. It has a 1 for registers temporarily in use, and a 2 for those permanently in use.

RR

See RIGHT.

RX

The register which is to be used as an index in the next access of a field within a particular sequence. If RX=0, there is no index register needed.

SHIF

See FIELD

SIZ

See BLOCK.

TB

The index of the most recently entered base field in the bug table.

TBK

The index of the most recently entered block in the block table.

TF

The index of the most recently entered field in the field table.

TI

The index of the most recently looked-up entry in the field, base field, or block tables.
TL  The index of the most recently entered label in the table for I/O.

WID  See FIELD.

WORD  See FIELD.
GBLB

ANY
BASE
BKL
EX
EXT
FOUND
ONLY

See IF.
See BASEBLOCK.
TRUE if the left sequence is a block sequence.
See LEFT.
See FIELD.
TRUE if the thing looked for in TABLEF, TABLEB, TABLEBK, or TESTSEQ was found.
TRUE if the object we are processing is a base field or base block as opposed to a field or block.
GBLC

BLKS  See BLOCK.

C      The current or most recently looked-at name which is from a sequence.

DLB   See BLOCK.

F, G   See SETUP.