

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.

66-8 cop. 2

STORE.

CCP-171
9/23/66 diuj
Jay Earley

X

*1

INTERNAL SPECIFICATIONS

Jay Earley

field contents; B1 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 B0 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:

<u>Macro</u>	The name of the macro.
<u>Called by</u>	A simple list of the macros which call the macro.
<u>Inputs</u>	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.
<u>Macros called</u>	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 <u>Action</u> .
<u>Outputs</u>	A list similar to <u>Inputs</u> .
<u>Code*</u>	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 <u>Action</u> . The use of any globals shown here or in a listing of the macros may be found by consulting the list of global descriptions.
<u>Action</u>	This explains what the macro accomplishes and how.

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.

<u>Macro</u>	ACCESS	
<u>Called by</u>	LINK, RIGHT	
<u>Inputs</u>	<u>Globals</u>	
	OLD	Register pointing to block of field to be accessed.
	NEW	Register to get field.
	C	The name of the field being accessed.
<u>Macros called</u>	SETUP	To set up F and G to represent to word of the field.
	<u>Globals</u>	
<u>Outputs</u>	NEW	The register into which the fields has been accessed.
<u>Action</u>	Code:	
	L NEW, FG	
	N NEW, C1	

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

* Affected by Meta-language

<u>Macro</u>	ADDD	
<u>Called by</u>	BINOP	
<u>Inputs</u>	None	
<u>Macros called</u>	EASY	with parameter "A", to produce the code to do addition.
<u>Outputs</u>	None	
<u>Action</u>	Code:	
	EASY	A
<u>Macro</u>	ADDX	
<u>Called by</u>	BSTORE	
<u>Inputs</u>	<u>Parameters</u>	
	RO	A register to which RX may be added.
<u>Macros called</u>	None	
<u>Outputs</u>	None	
<u>Action</u>	If RX = 0, we do nothing. Otherwise we code	
	AR R0, 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, R0.	

<u>Macro</u>	AND
<u>Called by</u>	BINOP
<u>Inputs</u>	<u>Globals</u> RR, EX, F, G
<u>Macros called</u>	SHIF, STOR
<u>Outputs</u>	None
<u>Action</u>	Code: SHIF O 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.

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

<u>Macro</u>	BASFIELD	
<u>Called by</u>	Source Language	
<u>Inputs</u>	<u>Parameters</u>	
	BUG	The name of the base field.
	LOC	Its location.
<u>Macros called</u>	None	
<u>Outputs</u>	BUGS	This is the column of the bug table which contains the bug names.
<u>Globals</u>	<u>GBLA</u>	
	B	Index to most recent storage location for base fields.
	TB	Index to bug table.
<u>Action</u>	BUG is set equal to LOC if it is passed or to the storage location (indexed by B) if it is not.	

Macro

BINDEC

Called by

BINOP

InputsParameters

F2 The field on the right which contains a binary number.

F1 The field on the left which will contain decimal digits.

Macros called

RIGHT, TABLEF

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

SETUP To set up F for UNPK.

Code

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)

Action

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.

Macro

BINOP

Called by

OP, TEST

InputsParameters

SEQ The sequence on the left.

OP The operator.

QUAN The quantity on the right.

Macros calledBINDEC, DECBIN If the operator is $D \leftarrow B$ or $B \leftarrow D$
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 calcu-
lates 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, MULT, ADDD, ORR, AND, XOR, GT, LT,
EQ, BEG Then depending on the operator, the
appropriate macro is called to exe-
cute the operation or test indicated.Outputs

None

Action

Codes the operation or test passed to it.

<u>Macro</u>	BLINK
<u>Called by</u>	SEQ, RIGHT, LEFT
<u>Inputs</u>	<u>Globals</u>
	<p>C The block through which we are linking.</p>
<u>Macros called</u>	TABLEBK, RIGHT, RET
<u>Outputs</u>	See Action
<u>Action</u>	<ol style="list-style-type: none"> 1. If C is a static base block we code <p style="text-align: center;">LA NEW, C.</p> 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, <ol style="list-style-type: none"> 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

BLOCK

Called by

Source Language, BASBLOCK

InputsParameters

B The name of the block.

BOUNDS The lower bounds and size of the block.

Globals

BASE = 1 if its a base block, = 0 otherwise.

TESTSEQ To test the lower bound to see whether it is dynamic.

Macros calledOutputs

BLKS, LB, DLB, SIZ 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.

Action

If it is a static base block, the block name is set equal to the lower bound.

Macro

BRAKTEST

Called by

SEQ

InputsParameters

STRING A sequence which is being linked through.

Globals

I The index of a character in the sequence.

C That character.

OutputsGlobals

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 "]".

<u>Macro</u>	BSTORE
<u>Called by</u>	BINOP
<u>Inputs</u>	None
<u>Macros called</u>	TABLEBK, ADDX, GPAIR, SETG, CLEAR
<u>Outputs</u>	None
<u>Code</u>	If the right side is a quantity, <pre> ST RR, INC.G CLEAR (INC=INC+4, SIZE=size of left block-4) </pre> <p>If the right side is a Z word block,</p> <pre> LM OLD, OLD+1, INCR(RR) STM OLD, OLD+1, INC(OLD) CLEAR (INC=INC+8, SIZE=size of left block-8) </pre> <p>Otherwise</p> <pre> MVC INC(L, OLD), INCR(RR) CLEAR (INC=INC+L, SIZE=size of left block-L) </pre> <p>where L = size of right block.</p>

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.

<u>Macro</u>	CALL	
<u>Called by</u>	UNOP	
<u>Inputs</u>	<u>Parameters</u>	
	DEST	A quantity which supplies the address of the subroutine.
<u>Macros called</u>	DO, RIGHT	
<u>Outputs</u>	None	
<u>Code</u>	<u>if DEST is a sequence</u>	<u>if DEST is a label</u>
	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, N₁ and N₂ 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

CHANGE

Called by

Source Language, PRIORITY

InputsParameters

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

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

<u>Macro</u>	COMP
<u>Called by</u>	UNOP
<u>Inputs</u>	SEQ The sequence defining the field to be complemented.
<u>Macros called</u>	LEFT, GREB, STOR
<u>Outputs</u>	None
<u>Action:</u>	Code:
	LEFT
	L R, FG
	X R, C1
	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.

<u>Macro</u>	DECBIN
<u>Called by</u>	BINOP
<u>Inputs</u>	<u>Parameters</u>
	F2 The field on the right which contains decimal digits.
	F1 The field on the left which will get a binary number.
<u>Macros called</u>	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.
<u>Code</u>	SEQ F2
	PAGE 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.

<u>Macro</u>		DEF	
<u>Called by</u>		INPUT	
<u>Inputs</u>		<u>Parameters</u>	
		BLK	The name of the I/O block.
<u>Macros called</u>		DTFCD, DTFPR	To define the files for the blocks.
		OPEN	To open the I/O area.
<u>Output</u>			BLK is entered in IOBL, the label table for I/O labels. The index of this entry is then the integer which is added to LAB to produce the label of the DTF.
<u>Code</u>	For input:	B	ON
	LAB	DTFCD	TYPEFLE = INPUT, RECFORM = FIXUNB, BLKSIZE = 80 DEVICE = 1442 DEVADDR = SYSRDR, EOFADDR = $\begin{cases} EA \\ \text{BLK}(2) \text{ if specified} \end{cases}$ IOAREA1 = BLK
	ON	OPEN	LAB
	For output:		
	LAB	DTFPR	RECFORM = FIXUNB, BLKSIZE = 120, DEVADDR = SYSLST, DEVICE = 1443, IOAREA1 = BLK
	ON	OPEN	LAB

Action: The indicated I/O block is defined and opened.

<u>Macro</u>	DIV	
<u>Called by</u>	BINOP	
<u>Inputs</u>	<u>Parameters</u>	
	OP	=/ if quotient is wanted or = MOD if remainder.
<u>Macros called</u>	SHIF, GREG, STOR	
	GPAIR	To get a pair of registers for the division.
<u>Outputs</u>	None	
<u>Code:</u>		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, C1

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.

<u>Macro</u>	DO	
<u>Called by</u>	Source Language, BINOP, CALL, RETURN	
<u>Inputs</u>	<u>Parameters</u>	
	P1,...,P8	Parenthesized operations of source language
<u>Macros called</u>	OP	Called once with each operation as a parameter.
<u>Outputs</u>	None	
<u>Action:</u>	Codes each operation.	
<u>Macro</u>	EASY	
<u>Called by</u>	ADDD, SUB, ORR, XOR	
<u>Inputs</u>	<u>Parameters</u>	
	OP	A letter indicating the operation to be performed (A,S,O,X)
<u>Macros called</u>	SHIF, STOR	
<u>Outputs</u>	None	
<u>Code</u>	SHIF	
	OP RR, FG	
	STOR (NEW=RR, EX=0)	
<u>Action:</u>	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.	

<u>Macro</u>	ENTER	
<u>Called by</u>	CHANGE	
<u>Inputs</u>	<u>Parameters</u>	
	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.
	<u>Globals</u>	
	T1	The index of the entries which are to be made.
<u>Macros called</u>	None	
<u>Outputs</u>		The above entries are made for columns LINK, STOR, STORE, or ACCESS
<u>Macro</u>	ENTERB	
		Same as ENTER except that the column names all have B's after them.

<u>Macro</u>	EQ	
<u>Called by</u>	BINOP	
<u>Inputs</u>	<u>Parameters</u>	
	OP	The relation (=, =0, ≠, ≠0)
<u>Macros called</u>	REL	With parameter E for equality test.
	NOTANY	With parameter OP, to set up K and N for REL.
<u>Outputs</u>	None	
<u>Action:</u>	Produces code to make the test.	

Macro

FIELD

Called by

Source Language

InputsParameters

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.

Macros called

GREG, RRESS To get and return registers used
 for computing the mask.

Outputs

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.

Code

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	} This produces a mask with 1's in the bits of the field in R.
LR	R1, R	
SRL	R, LD	
SLL	R1, RD	

Macro FIELD - cont'd.

NR	R, R1	
LR	R1, R	} This puts its complement in R1
X	R1, ONES	

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.

<u>Macro</u>	FINAL		
<u>Called by</u>	Source Language		
<u>Inputs</u>	None		
<u>Macros Called</u>	CLOSE		To close each of the I/O blocks.
	EOJ		To end the run.
<u>Outputs</u>	None		
<u>Code</u>	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.

<u>Macro</u>	GOTO		
<u>Called by</u>	UNOP		
<u>Inputs</u>	<u>Parameters</u>		
	DEST		The quantity which specifies the address to go to.
<u>Macros called</u>	RIGHT		
<u>Outputs</u>	None		
<u>Code</u>	If a quantity		If a sequence
	B DEST		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.

<u>Macro</u>	GPAIR	
<u>Called by</u>	MULT, DIV, BSTORE	
<u>Inputs</u>	None	
<u>Macros called</u>	None	
<u>Outputs</u>	<u>GBLA</u>	
	R	Number of the first register of the pair.
<u>Globals</u>	REG	Register column.
<u>Action:</u>	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.	

<u>Macro</u>	GREG	
<u>Called by</u>	Many things	
<u>Inputs</u>	None	
<u>Macros called</u>	None	
<u>Outputs</u>	<u>Globals</u>	
	R	Number of register found
<u>Action:</u>	The first unused register ($REG(R) = 0$) is found and its entry is set to 1. showing that it is now in use.	

<u>Macro</u>	GT	
	Same as EQ except parameter is H for high.	

<u>Macro</u>	IF	
<u>Called by</u>	Source Language, IFANY	
<u>Inputs</u>	<u>Parameters</u>	
	P1,...,P8	A number of tests followed by THEN followed by a number of operations.
	<u>GBLB</u>	
	ANY	=0 if called from source language =1 if called by IFANY
<u>Macros called</u>	TEST	Called with each test as parameter.
	OP	Called with each operation as parameter.
	RREGS	Called after each call of TEST or OP to mark. The registers they had used as unused.
<u>Outputs</u>	None	

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

	<u>IF</u>		<u>IF ANY</u>
	Test		Test
	Branch on False to OUT		Branch on true to IN
	Test		Test
	Branch on False to OUT		Branch on true to IN
	Operation		Branch to OUT
	Operation	IN	Operation
OUT	NOPR		Operation
		OUT	NOPR

<u>Macro</u>	IFANY	
<u>Called by</u>	Source Language	
<u>Inputs</u>	<u>Parameters</u>	
	P1,...,P8	A number of tests followed by THEN followed by a number of operations.
<u>Macros called</u>	IF	With same parameters as IFANY, except that ANY = 1.
<u>Outputs</u>	None	
<u>Action:</u>	Produces code to execute the operation if any of the tests are true.	
<u>Macro</u>	IN	
<u>Called by</u>	Source Language	
<u>Inputs</u>	<u>Parameters</u>	
	BLK	The name of the block into which a card is to be read.
<u>Macros called</u>	TABLEL	To look up the block name and set its DTF label.
	GET	To input the card.
<u>Outputs</u>	None	
<u>Action:</u>	The next card is read into block BLK.	

<u>Macro</u>	INITIAL
<u>Called by</u>	Source Language
<u>Inputs</u>	None
<u>Macros called</u>	BASFIELD, FIELD
<u>Outputs</u>	None
<u>Code</u>	BEGIN BALR 2,0 USING *,2 B ON ONES DC 12C' DOUB DS 1D ON

and others

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

<u>Macro</u>	INPUT
<u>Called by</u>	Source Language, OUTPUT
<u>Inputs</u>	<u>Parameters</u> P1,...,P8 Input blocks.
<u>Macros called</u>	DEF With each of the parameters in turn.
<u>Output</u>	<u>Globals</u> OUT Is reset to False

Action: DEF compiles the DTF for each input block.

<u>Macro</u>	LEFT	
<u>Called by</u>	BINOP, SHIFT, COMP	
<u>Inputs</u>	<u>Parameters</u>	
	SEQ	The sequence to be linked through in preparation for an operation or test.
<u>Macros called</u>	GREG, TABLEF, INBLEB	
	SEQ	This links through all the names of the sequence except the last.
	SETUP	To set up the right characters in F and G so that the proper field can be accessed in macros which follow.
<u>Outputs</u>	<u>GBLB</u>	
	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.

<u>Macro</u>	LINK*	
<u>Called by</u>	SEQ	
<u>Inputs</u>	<u>Parameters</u>	None
	<u>Globals</u>	
	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.
<u>Macros called</u>	ACCESS	
<u>Outputs</u>	<u>Globals</u>	
	OLD	The register which points to the new block and will now be OLD for the next link.
<u>Code</u>	ACCESS	
	SRL	NEW, CS

We get the word of the field into NEW, extract it, and shift it.

Then we set OLD to NEW.

<u>Macro</u>	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(T1) is not null, we call CHANGES with parameter MAC(T1). 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.

<u>Macro</u>	MULT
<u>Called by</u>	BINOP
<u>Inputs</u>	None
<u>Macros called</u>	SHIF, GREG, STOR
	GPAIR To get a pair of registers for the multiplications
<u>Outputs</u>	None
<u>Code</u>	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, C1
	MR 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.

<u>Macro</u>	NOTANY	
<u>Called by</u>	TEST, EQ, GT, LT	
<u>Inputs</u>	<u>Parameters</u>	
	OP	An operator which may or may not start with \neg .
<u>Macros called</u>	None	
<u>Outputs</u>	<u>Globals</u>	
	LAB	=OUT if called from IF, = IN otherwise, the label is followed by an integer (IO) to make it unique from other statements.
	K	=2 if OP starts with " \neg ", =1 otherwise.
<u>Action:</u>	Sets LAB and K.	
<u>Macro</u>	OP	
<u>Called by</u>	DO, IF	
<u>Inputs</u>	<u>Parameters</u>	
	OPER	A parenthesized operation.
<u>Macros called</u>	RETURN	With no parameter in case the operation has only one member.
	UNOP	With the 2 members of the operation as parameters if the operation has 2 members.
	BINOP	As with UNOP if the operation has 3 members.
<u>Outputs</u>	None	
<u>Action:</u>	Codes the operation.	

Macro

ORR

Same as ADDD, except that the parameter is 0.

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 PRIORITY

Called by Source Language

Inputs Parameters

BUG The name of the bug which is to be made a priority bug.

SW =OFF if the priority of the bug is to be turned off rather than on.

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 PRIO column of the bug table is set to the number of the register that the bug is loaded into.

Code

```

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.

<u>Macro</u>	PRIORS
<u>Called by</u>	CHANGES
<u>Inputs</u>	<u>Parameters</u>
	TAG The tag that was passed to CHANGES.
	PRIO(TI) The register which the prioritybug now resides in.

<u>Macros called</u>	RET
<u>Outputs</u>	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.

<u>Macro</u>	RREGS
<u>Called by</u>	DO, IF
<u>Inputs</u>	None
<u>Macros called</u>	None
<u>Outputs</u>	None

Action: Sets all entries to 0 which are 1 in the register column. It is marking these registers unused.

<u>Macro</u>	REL
<u>Called by</u>	EQ, GT, LT
<u>Inputs</u>	<u>Parameters</u>
	OP =E for equality, = H for >, =L for <
<u>Macros called</u>	ACCESS
<u>Outputs</u>	None
<u>Code</u>	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).

<u>Macro</u>	RETURN
<u>Called by</u>	OP
<u>Input</u>	None
<u>Macros called</u>	DO
<u>Outputs</u>	None
<u>Code</u>	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.

Macro

RIGHT

Called by

BINOP, CALL, BINDEC, GOTO, SHIFT, TEST, SEQ

InputsParameters

QUAN The quantity to be evaluated.

Macros called

GREG, TABLEF, TABLEB

SEQ The link through the sequence if there is one, except for the last element.

ACCESS To access the last element of the sequence.

TESTSEQ To test whether QUAN is a sequence.

OutputsGlobals

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.

Code

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.

<u>Macro</u>	SAVE
<u>Called by</u>	PRIORITY
<u>Input</u>	<u>Parameters</u>
	R11 The number of a register
<u>Macros called</u>	None
<u>Outputs</u>	None

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.

<u>Macro</u>	SEQ	
<u>Called by</u>	RIGHT, LEFT, DECBIN	
<u>Inputs</u>	<u>Parameters</u>	
	STRING	A string of names to be linked through.
<u>Macros called</u>	GREG, TABLEB, TABLEBK	
	BRACKETEST	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.
<u>Outputs</u>	<u>Globals</u>	
	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.

<u>Macro</u>	SETG	
<u>Called by</u>	SETUP, BSTORE	
<u>Inputs</u>	None	
<u>Macros called</u>	None	
<u>Outputs</u>	<u>Globals</u>	
	G	Sets G as described in SETUP.

Action: Sets G and zeroes RX.

<u>Macro</u>	SETUP	
<u>Called by</u>	LEFT, ACCESS, BINDEC, DECBIN	
<u>Inputs</u>	None	
<u>Macros called</u>	SETG	To set G, see below.
<u>Outputs</u>	<u>Globals</u>	
	F,G	These hold characters which represent how to access the current field. F has the displacement, G has (R) or (R1, R2) where the R's are registers. So we can later write L R, FG and the like.

Action: Besides setting F and G, it zeroes INC and RX after using them to set up F and G.

Macro

SHIF

Called by

STORE, EASY, AND, MULT, DIV, BLINK

InputsGlobals

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.

Outputs

None

Action:

Code is produced to shift RR to the position of
SHIF(TI)

<u>Macro</u>	SHIFT
<u>Called by</u>	UNOP
<u>Inputs</u>	<u>Parameters</u>
	SIZE A quantity representing the amount of the shift.
	SEQ A field sequence which is to be shifted.
	<u>Globals</u>
	DIR =R if the shift is right, = L otherwise.
<u>Macros called</u>	RIGHT, LEFT, GREG, STOR
<u>Outputs</u>	None
<u>Code</u>	RIGHT SIZE
	SRL RR, POS
	LEFT SEQ
	L R1, FG
	LR R2, R1
	N R1, C1
	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.

<u>Macro</u>	STOR*
<u>Called by</u>	EASY, MULT, DIV, AND, COMP, STORE, SHIFT
<u>Inputs</u>	<u>Globals</u>
	<p>OLD A register containing the word which contains the field which is to be stored into. This may be undefined if EX = FASE.</p> <p>NEW A register containing the quantity to be stored.</p>
<u>Macros called</u>	None
<u>Outputs</u>	None
<u>Code</u>	<p>N OLD, CO</p> <p>OR NEW, OLD</p> <p>ST NEW, FG</p>

Action: We destroy the contents of the field in OLD. Then we OR in 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.

<u>Macro</u>	STORE*	
<u>Called by</u>	BINOP	
<u>Inputs</u>	<u>Parameters</u>	None
	<u>Globals</u>	
	F, G	The characters which access the word of the field to be stored into. (i.e. for a field which is in the 2nd word of its block, which is pointed to by register 3, F=2, G=(3))
	RR	The register containing the quantity to be stored.
	ONLY	=TRUE if we are storing into a basefield.
<u>Macros called</u>	GREG, SHIF, STOR	
<u>Outputs</u>	None	
<u>Code</u>	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)	

Action: In the first case we are just storing, so the quantity in RR is shifted to match the position of the field in F. Then we load R with the word containing the old copy of F, and we store. In the second case we are making the left field point to the right block, so we load RR with the address of the beginning of that block and recall store.

* 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.

Macro

TABLEF

Called by

Many things

InputsParameter

FD

The name of the field to be looked up.

Macros called

None

OutputsGlobals

FOUND

=1 if it finds the name in the table.

T1

= the index of the name in the 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.

<u>Macro</u>	TEST	
<u>Called by</u>	IF	
<u>Inputs</u>	<u>Parameter</u>	
	OPER	A parenthesized test to be coded.
<u>Macros called</u>	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 \neg preceding the sequence.
	RIGHT	Is called with the sequence as parameter.
<u>Code</u>	After calling these macros, let R be a register, then	
	SR	R, R
	CLR	RR, R
	B(N)Z	LAB
		} only produced if no extractor was used to get the quantity in RR.
	(where N depends on \neg 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).

<u>Macro</u>	TESTSEQ
<u>Called by</u>	BLOCK, RIGHT
<u>Inputs</u>	<u>Parameters</u>
	STRING A string which may be a sequence.
<u>Macros called</u>	BRACKET, TABLEF, TABLEB, TABLEBK
<u>Outputs</u>	<u>Globals</u>
	FOUND =TRUE if string is a sequence.

Action: If the first name in the string is a block or basefield, and every other name is a block or field, then FOUND ← TRUE else FOUND ← FALSE.

<u>Macro</u>	UNOP
<u>Called by</u>	OP
<u>Inputs</u>	OP The unary operator
	THING Its operand.
<u>Macros called</u>	COMP With parameter THING if the operator is "┌" to perform the complement operation.
	GOTO With parameter THING, if the operator is "GOTO".
	SHIFT With parameters
	(1) The rest of OP, excluding the
	→ or ←
	(2) THING
	and with DIR=R for →, L for ←.
	CALL With parameter THING if operator is "CALL".
<u>Outputs</u>	None

Action: 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.
REG	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 lable in the lable table
 for I/O.

WID See FIELD.

WORD See FIELD.

GBLB

ANY See IF.

BASE See BASBLOCK.

BKL = TRUE if the left sequence is a block sequence.

EX See LEFT.

EXT See FIELD.

FOUND = TRUE if the thing looked for in TABLEF, TABLEB, TABLEBK, or
TESTSEQ was found.

ONLY = 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.