Specification: EVAL Microprogram

1.0 GENERAL. The EVAL operation performs interpretive evaluation of expressions which are written in a modified Polish-string notation. In addition to initiating arithmetic operations as a part of the string evaluation, the operation performs all necessary housekeeping for control of a pushdown operand stack. Once the EVAL operation is initiated, it maintains control until certain escape characters appear in the Polish string, or until a data fetch is attempted which requires software intervention. At these times, control goes to the next machine instruction.

1.1 FORMAT. EVAL is an RS-format instruction with the hexidecimal operation code AE. The fields within the RS-format are interpreted as described below.

1.1.1 FIELD DEFINITION.

R1 points to the string which is being evaluated.

R3 identifies a register which EVAL will use to return clue information to the using program.

B2,D2 is the base address for an indirect addressing table.

1.2 ADDRESSING CONVENTIONS. All information which is communicated via general registers is in the form of 24-bit absolute addresses. All addresses contained in the indirect addressing table is in the form of 12-bit base-displacement addresses.

1.3 *REGISTER USAGE. R1, R3, and R(0) are used to communicate information to and from the EVAL operation.

1.3.1 GENERAL REGISTER R1. R1 may be any of the general reigsters other than R(0). It should be distinct from R3, since both R1 and R3 are updated by EVAL. The left-most byte of R1 is altered by EVAL; a programmer should not depend upon information contained in this byte.

s points to the next string character to be interpreted. The user must initialize R1 to the first character in his string before invoking EVAL. R1 is updated by EVAL and will be current upon return from the operation.

contents of this byte are unpredictable.

---

* NOTE: Through this specification, R1 and R3 refer to general registers specified in the operation code. R(0) refers to general register 0.

Specification: EVAL Microprogram

1.3.2 GENERAL REGISTER R3. R3 may be any of the general registers except R(0). It should, however, be distinct from R1. The contents of R3 provide clue information upon return from execution of the EVAL operation. The specific information contained in R3 depends on the manner in which EVAL completed its execution.

1.3.2.1 R3 CONTENTS ON FUNCTION ESCAPE: Two of the EVAL operators cause an escape, after an address mapping is performed. The contents of R3 are similar in both cases, differing only in byte 0 of the register.

Function escape

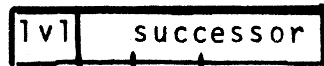
Left-variable escape

pointer is a 24-bit address which results from mapping the R-number which follows the operators mentioned in an EVAL-string.

1.3.2.2 R3 CONTENTS ON OTHER ESCAPES. The remaining escape operators set up R3 as shown below:

esc is the string character which caused escape from EVAL

1.3.2.3 R3 CONTENTS ON ADDRESS INTERCEPT. There is a third escape from EVAL, which occurs when the address mapping process is "intercepted". For this case, R3 will contain:



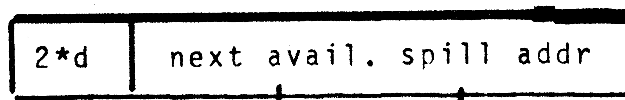
lvl is the number of levels of indirect addressing which occurred before interception.

successor is a 24-bit pointer to the location which would have been accessed if interception had not occurred.

1.3.3 GENERAL REGISTER 0. This register points to a core area into which the operand stack may overflow. Before EVAL is invoked, the user should insure that R(0) is pointing to a memory address which is:

a) On a double word boundary
b) available for storage of the operand stack
c) at least 64 double-words long

EVAL updates R(0) before escape so that it will contain the following information:



d — the number of items currently in the operand stack.

next addr — is the address into which an additional operand would be placed.

Note that the spill base may be computed from:

NAØ = NA - 2*(2*d)

where
NAØ = spill base
NA = next available spill address
d = current stack depth.

1.4 CONDITION CODES. The condition register is set by EVAL to reflect the reason for terminating the operation.

cc	explanation
ØØ	Normal completion. EVAL has detected an EOX (End of Expression) operator in the string.
Ø1	Function Call. A function operator or an odd-address interception has occurred. These cases are consistent in that a function operator is inserted in the Polish-string whenever it can be determined from context that a function must be invoked. The odd-address intercept is forced as a result of declarations which specifiy that an identifier is, in fact, the name of a function. Function codes take precedence over odd-address intercepts in the sense that odd addresses are ignored while mapping the R-number associated with a function.
1Ø	Escape. Any of the pure escape characters other than EOX set the condition code to two.

Specification: EVAL Microprogram

1.4 CONDITION CODES (continued)

cc	explanation
11	Program Interruption. The only abnormal exits from EVAL occur as the result of a program interrupt (e.g. invalid address, specification error, arithmetic error). If an interrupt occurs the appropriate interruption routine is executed, then control passes to the instruction following EVAL.

1.5 MACHINE STATE ON RETURN. When execution of EVAL is completed the machine state is as described in Table 1 . The operand stack has been unloaded and resides in core. R1, R3 and R(Ø) have been updated to the values discussed in earlier paragraphs, and the appropriate condition code has been set. Upon return from EVAL, the user is free to use any of the machine registers (general or floating point) except that, of course, he should preserve any pertinent information in R1,R3, or R(Ø).

Operator	Action	CC	C[R3]
ØØ Expon.	Checks for two operands, then escapes. Sets ØØ in low byte of R3.	1Ø	ØØ ØØ ØØ ØØ
Ø6 LVAR	Left-replace variable. Does R-mapping on the byte which follows the LVAR Code. Sets CØ into byte Ø of R3.	Ø1	CØ rr rr rr
Ø7 Function	Does R-mapping on the byte which follows the FUNCTION Code. Sets 8Ø into byte Ø of R3.	Ø1	8Ø rr rr rr
ØB Left-Rep.	Sets ØB into byte 3 of R3.	1Ø	ØØ ØØ ØØ ØB
ØF EOX	End of expression string. Sets ØF into byte 3 of R3.	ØØ	ØØ ØØ ØØ ØF

TABLE 1. Machine State on Return from EVAL operations

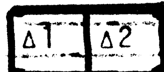
Specification: EVAL Microprogram

2.0 STRING NOTATION. The string which is interpreted by EVAL must be consistent with coding conventions which are designed into the operation code. The following paragraphs describe the string conventions and notation which must be adhered to.

2.1 BYTE CLASSIFICATION. Each byte in the string is an operator which causes some action with respect to a push-down stack of operands. By convention, the four high order bits distinguish a "Load Stack" operator from the other operators in the set. Specifically, if the four high order bits are zero, an operator is encoded in the four low-order bits; otherwise a Load Stack operation is implied.

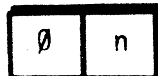
2.1.1 LOAD STACK OPERATOR. A load stack operation is implied whenever any of the four high-order bits are non-zero. The Load Stack operator uses the byte contents to accomplish a two-level indirect address mapping. Detailed interpretation is discussed in Section 3 of this specification.

Load Stack



where $\Delta 1 \neq \emptyset$

2.1.2 OTHER STACK OPERATORS. If the high-order four digits of a byte are zero, the four low-order bits specify one of the 16 permissible EVAL operators. The EVAL operators fall into three general classes; unary operators, binary operators, and process control operators. An operator byte has the following configuration:



where n= one of the sixteen possible four-bit combinations.

2.2 OPERATOR LENGTH. As a rule, operators require only a single byte in the Polish-string. In certain specific cases, the byte following an operator contains additional information which the operator must use.

2.2.1 COMPARE OPERATOR. The operator byte for compare is immediately followed by a byte which indicates the condition(s) for which the comparison is "True".

2.2.2 FUNCTION OPERATOR. The operator byte for a function is immediately followed by a Load Stack Operator. This second byte provides the internal mapping (R-number) of the function to be evaluated.

2.2.3 LEFT-VARIABLE OPERATOR. The operator byte for a left-variable

2.2.3 LEFT-VARIABLE OPERATOR(continued)

is immediately followed by a Load Stack Operator. This second byte provides the internal mapping (R-number) of the Left-hand variable.

2.4 OPERATOR CODES. The 16 EVAL operators are described in this section of the specification. Arithmetic, logical and relational operators have an interpretation which is consistent with the PL/I programming language. Escape and no-operation codes have no source-language counterpart; their interpretation is tailored to the needs of the system for which EVAL was developed. Operator actions, and machine state at the end of operator interpretation are summarized in Table 2.

2.4.1 UNARY OPERATORS. The unary operators operate on the top member of the operand stack. The result of their operation replaces the top member of the stack.

2.4.1.1 PREFIX MINUS (Code Ø2). The value at the top of the operand stack is negated; that is, it is replaced by minus it's value.

$$A \leftarrow -A$$

2.4.1.2 LOGICAL NEGATION (Code ØE). The value at the top of stack is replaced by Ø if it was "True"; by 1 if it was "False". "True" and "False" values are consistent with the PL/I definition.

$$A \leftarrow 0 \quad \text{if} \quad |A| \geq 1$$

$$A \leftarrow 1 \quad \text{if} \quad |A| < 1$$

2.4.2 BINARY OPERATORS. The binary operators work with the top two members of the operand stack. The general operation is

$$A \leftarrow A \,\delta\, B$$

where δ is one of the permitted binary operators.
B is the value at the top of the operand stack
A is the second value in the operand stack.

Upon completion of the operation, A is replaced by a result and B is deleted from the top of stack.

Spec **Specification: EVAL Microprogram**

2.4.2 BINARY OPERATORS (continued)

2.4.2.1 ARITHMETIC OPERATORS.

There are four binary arithmetic operators. The values on which they operate are A-BC Floating Decimal Numbers. The specific operations are defined as follows:

Code	Operation	Process
Ø8	addition	A ← A+B
Ø9	subtraction	A ← A-B
ØC	multiplication	A ← A*B
ØD	division	A ← A/B

2.4.2.2 LOGICAL OPERATORS.

The two binary logical operators follow the PL/I convention for the Truth or falsity of a value.

T if $|A| \geq 1$; F if $|A| < 1$

The logical operators satisfy the following truth tables:

Logical And (&)

B \ A	F	T
F	Ø	Ø
T	Ø	1

Logical Or (|)

B \ A	F	T
F	Ø	1
T	1	1

The stack operations are:

Code Ø5 A ← A&B
Code Ø1 A ← A|B

2.4.2.3 COMPARISON OPERATOR(Code Ø4).

The compare operator tests the two top stack operands. The results of its comparison are tested against the relation specified by a byte which immediately follows the compare operator. If the specified relationship is true, a value of one is placed on the stack; if false, a value of zero is placed on the stack. In general

A ← A ρ B

where ρ is one of the permitted relationals.

2.4.2.3 COMPARISON OPERATOR(continued)

The four low-order bits of an information byte are used to specify the desired relationship. The relational codes are identical to those used in an S/360 conditional branch instruction. However, there are repeated here for completeness. In the table which follows both the compare operator (Code Ø4) and the relational mask are shown.

String Code	Conditions Tested				Remarks
	=	<	>	?	
Ø4 ØØ	Ø	Ø	Ø	Ø	False
Ø4 Ø2	Ø	Ø	1	Ø	A greater than B
Ø4 Ø4	Ø	1	Ø	Ø	A less than B
Ø4 Ø6	Ø	1	1	Ø	A not equal B
Ø4 Ø8	1	Ø	Ø	Ø	A equal B
Ø4 ØA	1	Ø	1	Ø	A greater than or equal B
Ø4 ØC	1	1	Ø	Ø	A less than or equal to B
Ø4 ØE	1	1	1	Ø	True

2.4.3 CONTROL OPERATORS. The remaining EVAL operators are used to control processing of the string. They fall into two classes, escape operators, and no-operators.

2.4.3.1 NO-OPERATORS. No actual computation results from the appearance of these operators in the Polish-string. They are included in the operators set to simplify the problems of recomposing the Polish-string into a source statement.

Code	Name	Action
Ø3	Skip	This byte and the byte which immediately follows it are ignored by EVAL.
ØA	Nop	This byte only is ignored by EVAL.

2.4.3.2 ESCAPE OPERATORS. There are five operators which terminate execution of an EVAL instruction. Two of these operators are followed by an information byte which is acted upon before escape. The other three are single-byte operators.

2.4.3.2 ESCAPE OPERATORS(continued)

Code	Name	Action
ØØ	Exponentiation	Operand stack is checked to make sure that there are two operands available. No actual computation is performed.
Ø6	Left-Variable	The Byte which follows the Ø6 Code is mapped into a 24-bit address. The string pointer is positioned to point two characters beyond the Ø6 and an escape is initiated.
Ø7	Function	Action is similar to that for Code Ø6.
ØB	Left-Replace	An escape is initiated, with the proper information in R3.
ØF	End-of-expression	An escape is initiated with the proper condition code setting.

Specification: EVAL Microprogram

3.0 VARIABLE MAPPING. Variables in an EVAL String are represented by one-byte internal codes. When EVAL detects a variable in the string, it is treated as a Load Stack Operator. The microprogram attempts to fetch the current data value via a two-level indirect addressing scheme. In the case of a scalar variable, the fetch is completed successfully, and the data value is placed on top of the operand stack. For array variables, and certain other cases, the attempt to fetch data is "intercepted" and EVAL relinquishes control to software.

3.1 LOAD STACK INTERPRETATION. The internal code for a variable is treated as two four-bit offsets to be used in the address mapping process.

LOAD STACK

The addressing algorithm is:

$$V = C[L + 2*\Delta^2]$$

with

$$L = C[R + 2*\Delta^1]$$

in the above

R = Base of a 16 half-word Segment Directory
L = Base of a 16 half-word Line Directory
Δ^1, Δ^2 = Row numbers in their respective directories.
V = Value sought

3.1.1 SEGMENT DIRECTORY. The segment directory consists of a sixteen half-word table which starts on a half-word boundary. Each entry in the table consists of the base-displacement address of a Line Directory. Segment Ø is not used because the internal codes Øn are reserved for operators.

3.1.2 LINE DIRECTORIES. Each line directory cosists of sixteen half-words, starting on a half-word boundary. The entries in a Line Directory consist of base-displacement addresses which point to data values or to data attribute tables.

3.1.3 DATA WORD BOUNDARIES. By convention, data values must be stored on ~~double word~~ appropriate boundaries, and data attribute tables on full-word boundaries. Thus the base-displacement (BD) addresses in Line- or Segment-Directories normally give 24-bit effective addresses which are multiples of 4.

3.2 FETCH INTERCEPTION. As stated in 3.1.3, normal addresses in a directory will be even. EVAL is designed so that an odd effective address produces the following:

a) The effective address is placed in R3
b) R1 and R(Ø) are updated, and the operand stack is spilled to core.
c) EVAL execution is terminated.

3.2 FETCH INTERCEPTION. (Continued)

The instruction sequence which follows EVAL should test for this termination condition, and then use the information which is pointed to by R3 to complete the data fetch.

3.3 SHORT PRECISION DATA. The normal data value is a double precision Allen-Babcock Decimal Number. However, bit 30 in an effective address is used to signal the existence of a short (full-word) data value. If this bit is on, then only the high order portion of the data value is fetched from memory. Zeros are forced into the low-order portion of the operand stack, and processing continues as for normal values.

FUNCTIONAL CHARACTERISTICS: EVAL MICROPROGRAM

5.0 INTERNAL OPERATION OF EVAL. This section provides detailed information about the organization and the inner workings of the EVAL operation. A knowledge of microprogramming, and the availability of appropriate Control Logic Diagrams is assumed.

5.1 OVERALL ORGANIZATION. As far as possible, the EVAL Microprogram was divided into functional pieces which are on seperate Control Logic Diagrams (CLD's). The following table summarizes the processes which are defined by each of the CLD's.

CLD	FUNCTIONS PERFORMED
QZ489	Post I-fetch Initialization.
QZ490	Setup on entry to EVAL. Refetch Stack operand on Re-entry to EVAL. Pick up next string character.
QZ491	Perform R-mapping for Load Stack Operations. Classify other operations (Unary,Binary,Control) Re-fetch 2nd operand for binary operators.
QZ492	Branch on EVAL operator; link to arithmetic. Treat skip and nop operators. Treat unary minus.
QZ493	Treat AND,OR, and NOT logical operators. Convert result of COMP into TRUE or FALSE.
QZ494	Move Data from Memory to operand stack.
QZ495	Treat Function and LVAR Operators.
QZ496	Clean up and housekeeping before exit from EVAL. Post-arithmetic cleanup.
QZ497	Spill operand stack to memory.
QZ498	.....Not presently used....
QZ499	Treat error returns from arithmetic operations.

The relationships among the above CLD'S are shown schematically on Figure 5.0. Normal flow of control is shown by double lines; error paths are single lines.

FUNCTIONAL CHARACTERISTICS: EVAL MICROPROGRAM

5.0 INTERNAL OPERATION (Continued)

5.2 PROGRAM SIZE. The EVAL Microprogram requires approximately 191 words of Read-only Storage (ROS). It is located in Extended ROS with the following distribution across ROS planes:

Plane Number	Number of ROS Words	CLD Reference
1	1	QZ489 JA
4	22	QZ493
	19	QZ494 except RA,TA
5	5	QZ489 except JA
	23	QZ490
	33	QZ491
	23	QZ492
	2	QZ494 RA and TA
	17	QZ495
	15	QZ496
	24	QZ497
	7	QZ499
	191	

5.3 DETAILED DESCRIPTION. A detailed narrative description of EVAL will be written at a later date.

FUNCTIONAL CHARACTERISTICS: EVAL MICROPROGRAM

6.0 ERROR CONDITIONS. A variety of error conditions are detected within EVAL. Specific tests, the actions taken, and possible causes are detailed below.

ROS Location		Data	Error Type	Action
QZ490	CE	Rotor	Invalid Addr	IC= 25
	EH	Rotor	Not dble-word	26
	CH	String	Invalid Addr	25
	JJ	Depth	Odd	26
QZ491	LM	Spill	Invalid Addr	25
	MC	Depth	Stack Underflow	27
	SF	R-SEG	Invalid Addr	25
QZ493	LD	String	Invalid Addr on Compare Mask	25
QZ494	SG	Stack	Overflow	23
	TG	Data	Off dble-word bound	26
	VH	Spill	Invalid Addr	25
QZ495	ED	String	Function-Inv Addr	25
QZ497	NI	Spill	Invalid Addr	25
	QI	Spill	Invalid Addr	26
QZ499	EJ	Math	Exponent Overflow	2C
			Exponent Underflow	2D *
			Significance	2E *
			Divide Check	2F

* May be masked

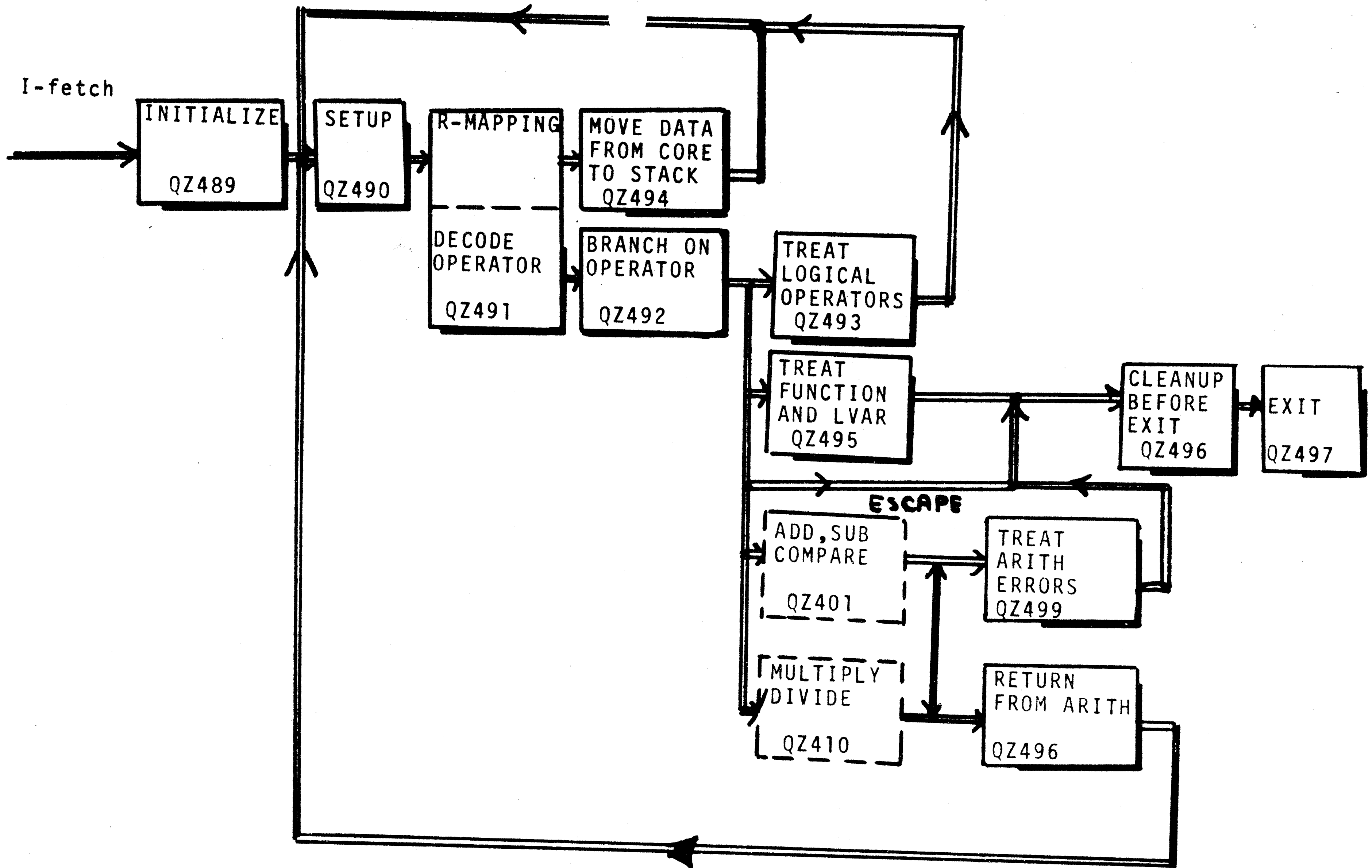


FIGURE 5.0. EVAL CLD SCHEMATIC

String	Operator	Process	Stack depth	string Control	R3	CC
ØØ ..	**	escape	d	S+1	ØØ ØØ ØØ ØØ	1Ø
Ø1 ..	or	A←A|B	d-1	S+1	-- -- -- --	--
Ø2 ..	pfx -	A← -A	d	s+1	-- -- -- --	--
Ø3 ..	skip	nop	----	s+2	-- -- -- --	--
Ø4 Øm	cmpar	A←AρB	d-1	s+2	-- -- -- --	--
Ø5 ..	and	A←A&B	d-1	s+1	-- -- -- --	--
Ø6 rr	LVAR	R[rr]→→R3; escape	d	s+2	CØ rr rr rr	Ø1
Ø7 rr	Function	R[rr]→→R3; escape	d	s+2	8Ø rr rr rr	Ø1
Ø8 ..	add	A←A+B	d-1	s+1	-- -- -- --	--
Ø9 ..	subtract	A←A-B	d-1	s+1	-- -- -- --	--
ØA ..	nop	nop	d	s+1	-- -- -- --	--
ØB ..		escape	d	s+1	ØØ ØØ ØØ ØB	1Ø
ØC ..	mult.	A←A*B	d-1	s+1	-- -- -- --	--
ØD ..	divide	A←A/B	d-1	s+1	-- -- -- --	--
ØE ..	negate	A← ∿A	d	s+1	-- -- -- --	--
ØF ..	EOX	escape	d	s+1	ØØ ØØ ØØ ØF	ØØ

Table 2. Summary of EVAL Operation Codes.