Published: 08/16/68

Identification

Specific POPS
B. P. Goldberg, I. B. Goldberg

# Chapter 2

# POPS

## A. WORK STACK POPS

### Pops

POP: CLOAD Load into current work

FORMAT: cload(Y)

FUNCTION: Set C(W0) = C(Y)

EXAMPLE: See CEAW

POP: CEAW Effective address to current work

FORMAT: ceaw(Y)

FUNCTION: Set C(W0) 0-17 = Y

Set C(W0) 18-35 = 0

EXAMPLE:

Assume C(VARSIZ) =

| 000007 | 000000 |
|---|---|
| 0 | 18 35 |

The following two pops are equivalent: cload(varsiz) and ceaw(7)

W0 - W5 Before

| | | |
|---|---|---|
| W5 | 000105 | 000000 |
| W4 | 000104 | 000000 |
| W3 | 000103 | 000000 |
| W2 | 000102 | 000000 |
| W1 | 000101 | 000000 |
| W0 | 000100 | 000000 |
| | 0 | 18 35 |

W0 - W5 After

| | | |
|---|---|---|
| W5 | 000105 | 000000 |
| W4 | 000104 | 000000 |
| W3 | 000103 | 000000 |
| W2 | 000102 | 000000 |
| W1 | 000101 | 000000 |
| W0 | 000007 | 000000 |
| | 0 | 18 35 |

POP: LOAD Load

FORMAT: load(Y)

FUNCTION: 1. Add 1 to work counter

2. Set C(W0) = C(Y)

EXAMPLE: See EAW

POP: EAW Effective address to work

FORMAT: eaw(Y)

FUNCTION: 1. Add 1 to work counter

2. Set C(W0) 0-17 = Y

Set C(W0) 18-35 = 0

EXAMPLE:

Assume C(VARSIZ) =

| 000007 | 000000 |
|---|---|
| 0 | 18 35 |

The following two pops are equivalent: load(varsiz) and eaw(7)

W0 - W5 Before

| | | |
|---|---|---|
| W5 | 000105 | 000000 |
| W4 | 000104 | 000000 |
| W3 | 000103 | 000000 |
| W2 | 000102 | 000000 |
| W1 | 000101 | 000000 |
| W0 | 000100 | 000000 |
| | 0 | 18 35 |

W0 - W5 After

| | | |
|---|---|---|
| W5 | 000104 | 000000 |
| W4 | 000103 | 000000 |
| W3 | 000102 | 000000 |
| W2 | 000101 | 000000 |
| W1 | 000100 | 000000 |
| W0 | 000007 | 000000 |
| | 0 | 18 35 |

POP: STOR Store

FORMAT: stor(Y)

FUNCTION: Set C(Y) = C(W0)

POP: STORP Store and prune

FORMAT: storp(Y)

FUNCTION: 1. Set C(Y) = C(W0)

2. Prune W0

POP: STU Store upper

FORMAT: stu(Y)

FUNCTION: Set C(Y) 0-17 = C(W0) 0-17

Do not change C(Y) 18-35

POP: STUP Store upper, and prune

FORMAT: stup(Y)

FUNCTION: 1. Set C(Y) 0-17 = C(W0) 0-17
Do not change C(Y) 18-35

2. Prune W0

POP: PRW Prune work

FORMAT: prw(Y)

FUNCTION: Subtract C(Y) 0-17 from work counter; i.e., prune work stack by C(Y) 0-17 words

EXAMPLE: See PWCT

POP: PWCT Prune work to count

FORMAT: pwct(Y)

FUNCTION: Prune work to size C(Y) 0-17

EXAMPLE:

Assume C(c2) =

| 000002 | 000000 |
|---|---|
| 0 | 18 35 |

C(c3) =

| 000003 | 000000 |
|---|---|
| 0 | 18 35 |

The following pops are equivalent: prw(c2) and pwct (c3)

Work Stack Before

| | | |
|---|---|---|
| W5 | 0 | 0 |
| W4 | 100 | 0 |
| W3 | 200 | 0 |
| W2 | 300 | 0 |
| W1 | 400 | 0 |
| W0 | 500 | 0 |
| | 0 | 18 35 |

Work Stack After

| | | |
|---|---|---|
| W3 | 0 | 0 |
| W2 | 100 | 0 |
| W1 | 200 | 0 |
| W0 | 300 | 0 |
| | 0 | 18 35 |

POP: PRWX Prune work for exit

FORMAT: prwx( )

FUNCTION: Prune work to its size prior to the last executed JSB pop

COMMENT: The function of this pop is to prune work, not to extend it

B. MISCELLANEOUS POPS

Pops

POP: POPNOP No operation

FORMAT: popnop( )

FUNCTION: Go to the next pop

POP: ORKEY Or symbol key

FORMAT: orkey(Y)

FUNCTION: Set C(SYMKEY) 18-35 = C(SYMKEY) 18-35 .or. C(Y) 18-35

SYMKEY is a one-word register in the data segment. (The interpreter ignores C(SYMKEY) 0-17)

POP: MRK Set MRKER

FORMAT: mrk(Y)

FUNCTION: Set C(MRKER) 0-17 = Y

MRKER is a one-word register in the data segment. (The interpreter ignores C(MRKER) 18-35)

POP: FACT Fact

FORMAT: fact(Y)

Y may be an even or odd location

FUNCTION: 1. Bump bottom of roll 3 (fact roll) by two words

2. Set C(word 1) = C(Y)

3. Set C(word 2) = C(Y+1)

COMMENT: fact(Y) is equivalent to
```
load(Y)
pob(3)
cload(Y+1)
pobp(3)
```

C. ARITHMETIC AND LOGICAL POPS

Most of the arithmetic and logical pops are in one of eight categories, as shown in Figure 1. The pops that are not in one of these categories are NGT and NOT.

1. Arithmetic Pops

Arithmetic pops perform the following types of operations: addition, subtraction, multiplication, division, and negation. These pops work on full words. However, they are frequently used to perform simple operations in which only the upper half of each word is of interest to the user. In these cases, the user should make sure that the lower halves of the words are cleared before the pops are executed. Otherwise, the results might be incorrect.

EXAMPLE:

The pop add(Y) sets C(WO) = C(WO) + C(Y)

| | Before | | | After | | |
|---|---|---|---|---|---|---|
| Y | 000003 | 777777 | | Y | 000003 | 777777 |
| | 0 | 18 | 35 | | 0 | 18 | 35 |
| WO | 000002 | 777777 | | WO | 000006 | 777776 |
| | 0 | 18 | 35 | | 0 | 18 | 35 |

The upper half of the result is 6, not 5, becuase of carry from the lower half.

Pops

POP: ADD Add

FORMAT: add(Y)

FUNCTION: Set C(WO) = C(WO) + C(Y)

| Category | Interpreter Action | Operation and Corresponding Pops | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | + | - | * | / | .and. | .or. | .ext. | .eor. | .ins. |
| 1. | Set C(WO) = C(WO) operation C(Y) | ADD | SUB | MLT | DVD | AND | OR | EXT | EOR | INS |
| 2. | Set C(Y) = C(Y) operation C(WO) | ADS | SBS | MLTS | | ANS | ORS | | ERS | |
| 3. | Set C(Y) = C(Y) operation C(WO) and prune WO | ADSP | SBSP | MLTSP | | ANSP | ORSP | | ERSP | |
| 4. | Set C(RP(WO)) = C(RP(WO))operation C(Y) | ADDI | | | | ANDI | | EXTI | | INSI |
| 5. | Set C(RP(WO)) = C(RP(WO))operation C(Y) and prune WO | ADDIP | | | | ANDIP | | EXTIP | | INSIP |
| 6. | Set C(B) = C(B) operation C(Y) B is a location on roll N (See ERB for further explanation) | | | | | | | | ERB | INSB |
| 7. | Set C(W1) = C(W1) operation C(Y) | | | | | | | | | INS1 |
| 8 | Set C(W2) = C(W2) operation C(Y) | | | | | | | | | INS2 |

Notes: a. The operations are described in the text

b. For representative examples of the categories, see EXT(categories 1,7, and 8), ORS (categories 2 and 3), ADDI (categories 4 and 5), and ERB (category 6).

Figure 1. Categories of Arithmetic and Logical Pops

POP: ADS Add to storage

FORMAT: ads(Y)

FUNCTION: Set C(Y) = C(Y) + C(WO)

POP: ADSP Add to storage, and prune

FORMAT: adsp(Y)

FUNCTION: 1. Set C(Y) = C(Y) + C(WO)

2. Prune WO

POP: ADDI Add indirect

FORMAT: addi(Y)

FUNCTION: Set C(RP(WO)) = C(RP(WO)) + C(Y)

EXAMPLE:

addi(alpha)

ALPHA

| 000002 | 000000 |
|---|---|
| 0 | 18 35 |

C(TOP+6) 0-17 = 100000

WO

| 5 | 0 | 6 |
|---|---|---|
| 0 | 18 30 | 35 |

RP(WO) = 100005

100005 before

| 000004 | 000000 |
|---|---|
| 0 | 18 35 |

100005 after

| 000006 | 000000 |
|---|---|
| 0 | 18 35 |

POP: ADDIP Add indirect, and prune

FORMAT: addip(Y)

FUNCTION: 1. Set C(RP(WO)) = C(RP(WO)) + C(Y)

2. Prune WO

POP: SUB Subtract

FORMAT: sub(Y)

FUNCTION: Set C(WO) = C(WO) - C(Y)

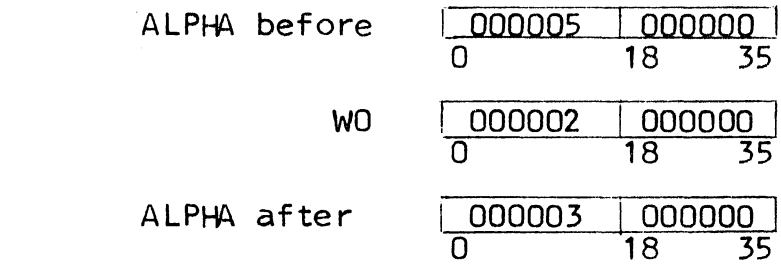
POP: SBS Subtract from storage

FORMAT: sbs(Y)

FUNCTION: Set C(Y) = C(Y) - C(WO)

EXAMPLE:

sbs(alpha)




POP: SBSP Subtract from storage, and prune

FORMAT: sbsp(Y)

FUNCTION: 1. Set C(Y) = C(Y) - C(WO)

2. Prune WO

POP: MLT Multiply

FORMAT: mlt(Y)

FUNCTION: Set C(WO) 0-17 = C(WO) 0-17* C(Y) 0-17, assuming that C(WO) 18-35 = 0 and C(Y) 18-35 = 0

COMMENT: The operands and the product are 18-bit upper-half integers. If necessary, the product is truncated on the left to 18 bits.

POP: MLTS Multiply to storage

FORMAT: mlts(Y)

FUNCTION: Set C(Y) 0-17 = C(Y) 0-17* C(WO) 0-17, assuming that C(Y) 18-35 = 0 and C(WO) 18-35 = 0

EXAMPLE:

```
mlts(alpha)
```

| | | |
|---|---|---|
| WO | 000003 | 000000 |
| | 0 18 | 35 |
| ALPHA before | 000002 | 000000 |
| | 0 18 | 35 |
| ALPHA after | 000006 | 000000 |
| | 0 18 | 35 |

COMMENT: The operands and the product are 18-bit upper-half integers. If necessary, the product is truncated on the left to 18 bits.

POP: MLTSP Multiply to storage, and prune

FORMAT: mltsp(Y)

FUNCTION: 1. Set C(Y) 0-17 = C(Y) 0-17* C(WO) 0-17 assuming that C(Y) 18-35 = 0 and C(WO) 18-35 = 0

2. Prune WO

COMMENT: The operands and the product are 18-bit upper-half integers. If necessary, the product is truncated on the left to 18 bits.

POP: DVD Divide

FORMAT: dvd(Y)

FUNCTION: Set C(WO) 0-17 = C(WO) 0-17/C(Y) 0-17, assuming that C(WO) 18-35 = 0, C(Y) 18-35 = 0, and C(Y) 0-17 ≠ 0

EXAMPLE:

```
dvd(alpha)
```

| | | |
|---|---|---|
| WO before | 000007 | 000000 |
| | 0 18 | 35 |
| ALPHA | 000003 | 000000 |
| | 0 18 | 35 |
| WO | 000002 | 000000 |
| | 0 18 | 35 |

COMMENT: The operands and the quotient are 18-bit upper-half integers. The remainder is ignored.

POP: NGT Negate (two's complement)

FORMAT: ngt( )

FUNCTION: Set C(WO) = -C(WO)

COMMENT: 1. Two negates will always return the original state of a number

2. Negation means taking the two's complement of a number. It does not mean changing bit 0 (the sign bit). The following code changes C(ALPHA) 0:

```
eaw(octal(400000))
ersp(alpha)
```

This code does not negate ALPHA, unless C(ALPHA) = 200000000000 or 600000000000.

POP: NGTS Negate storage (two's complement)

FORMAT: ngts(Y)

FUNCTION: Set C(Y) = -C(Y)

EXAMPLES:

In each of the following examples, the pop is ngts(alpha)

ALPHA before | 000001 | 000000 | (0, 18, 35) +1 ALPHA after | 777777 | 000000 | (0, 18, 35) -1

ALPHA before | 000000 | 000000 | (0, 18, 35) 0 ALPHA after | 000000 | 000000 | (0, 18, 35) 0

(There is no such thing as -0)

## 2. Logical Pops

### a. Simple Operations

The following truth table summarizes the simple logical operations:

| xk | yk | zk | | | |
|---|---|---|---|---|---|
| | | .and. | .ext. | .or. | .eor. |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 |

.and. = and
.ext. = extract
.or. = or
.eor. = exclusive or

The general format of pops performing these operations is:

Set zk = xk .logical. yk (for k = 0,1,..., and 35)

where .logical. represents the operation; and x,y,and z are 36-bit quantities whose k'th bits are xk, yk, and zk, respectively. The pops work on all bits of x,y, and z in parallel.

Pops

POP: AND And

FORMAT: and(Y)

FUNCTION: Set C(WO) = C(WO) .and. C(Y)

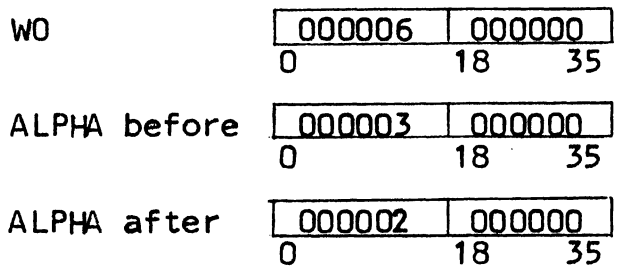
POP: ANS And to storage

FORMAT: ans(Y)

FUNCTION: Set C(Y) = C(Y) .and. C(WO)

EXAMPLE:

ans(alpha)




POP: ANSP And to storage, and prune

FORMAT: ansp(Y)

FUNCTION: 1. Set C(Y) = C(Y) .and. C(WO)

2. Prune WO

POP: ANDI And indirect

FORMAT: andi(Y)

FUNCTION: Set C(RP(WO)) = C(RP(WO)) .and. C(Y)

POP: ANDIP And indirect, and prune

FORMAT: andip(Y)

FUNCTION: 1. Set C(RP(WO)) = C(RP(WO)) .and. C(Y)

2. Prune WO

POP: OR Or

FORMAT: or(Y)

FUNCTION: Set C(WO) = C(WO) .or. C(Y)

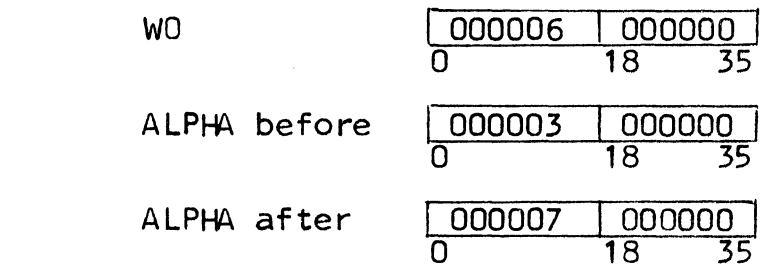
POP: ORS Or to storage

FORMAT: ors(Y)

FUNCTION: Set C(Y) = C(Y) .or. C(WO)

EXAMPLE:

ors(alpha)




POP: ORSP Or to storage, and prune

FORMAT: orsp(Y)

FUNCTION: 1. Set C(Y) = C(Y) .or. C(WO)

2. Prune WO

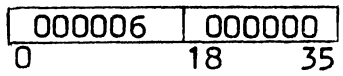
POP: EXT Extract

FORMAT: ext(Y)

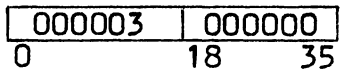
FUNCTION: Set C(WO) = C(WO) .ext. C(Y)

EXAMPLE: ext(alpha)

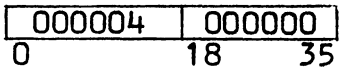
WO before




ALPHA




WO after




POP: EXTI Extract indirect

FORMAT: exti(Y)

FUNCTION: Set C(RP(WO)) = C(RP(WO)) .ext. C(Y)

POP: EXTIP Extract indirect, and prune

FORMAT: extip(Y)

FUNCTION: 1. Set C(RP(WO)) = C(RP(WO)) .ext. C(Y)

2. Prune WO

POP: EOR Exclusive or

FORMAT: eor(Y)

FUNCTION: Set C(WO) = C(WO) .eor. C(Y)

POP: ERS Exclusive or to storage

FORMAT: ers(Y)

FUNCTION: Set C(Y) = C(Y) .eor. C(WO)

POP: ERSP Exclusive or to storage, and prune

FORMAT: ersp(Y)

FUNCTION: 1. Set C(Y) = C(Y) .eor. C(WO)

2. Prune WO

POP: ERB Exclusive or to bottom

FORMAT: erb(Y)

FUNCTION: Set C(B) = C(B) .eor. C(Y)

B is a location on roll N, where N = C(MRKER) 0-17

For fixed-size groups, B = C(BOTTOM+N) 0-17
- C(GRPSIZ+N) 0-17

For variable-size groups, B = C(BOTTOM+N) 0-17
- C(VARSIZ) 0-17

EXAMPLE:

erb(alpha)

ALPHA | 000001 | 000000 | C(BOTTOM+6) 0-17 = 100100
0 18 35 B = 100100 - 100 = 100000

MRKER | 000006 | 000000 | C(GRPSIZ+6) 0-17 = 100
0 18 35

100000 before | 000007 | 000000 |
0 18 35

100000 after | 000006 | 000000 |
0 18 35

COMMENT: If N consists of fixed-size groups, B is the first word of the last group before the bottom. However, the user may set VARSIZ to any number; thus, if N consists of variable-size groups, B may be any word in (or even above) the last group.

b. Insert Operation

The following truth table summarizes the insert operation:

| xk | yk | zk before | zk after |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Pops in this category insert xk into zk when yk = 0. These pops work on all bits of x, y, and z in parallel. The operand Y is always an even location, with C(Y) = x and C(Y+1) = y. The location of z is different for each insert pop.

Pops

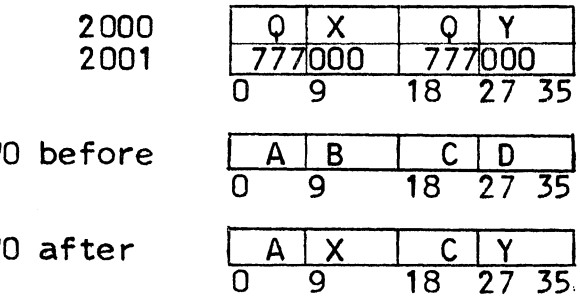
POP: INS Insert

FORMAT: ins(Y)
Y is an even address

FUNCTION: Set C(WO) = C(WO) .ins. C(Y)

EXAMPLE:

ins(2000)




POP: INSI Insert indirect

FORMAT: insi(Y)
Y is an even address

FUNCTION: Set C(RP(WO)) = C(RP(WO)) .ins. C(Y)

POP: INSIP Insert indirect, and prune

FORMAT: insip(Y)
Y is an even address

FUNCTION: 1. Set C(RP(WO)) = C(RP(WO)) .ins. C(Y)

2. Prune WO

POP: INSB Insert into bottom
(See ERB.)

FORMAT: insb(Y)
Y is an even address

FUNCTION: Set C(B) = C(B) .ins. C(Y)

POP: INS1 Insert into W1

FORMAT: ins1(Y)
Y is an even address

FUNCTION: Set C(W1) = C(W1) .ins. C(Y)

POP: INS2 Insert into W2

FORMAT: ins2(Y)
Y is an even address

FUNCTION: Set C(W2) = C(W2) .ins. C(Y)

C. Not Operation

Pop

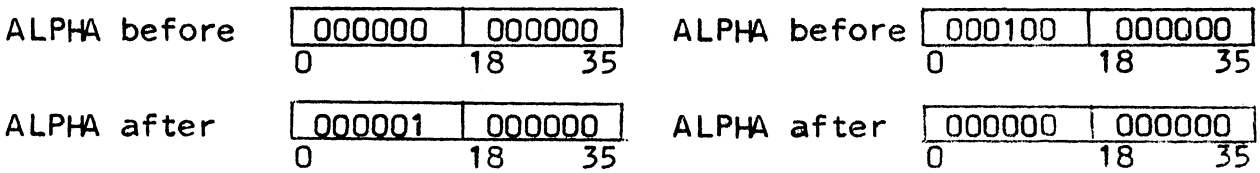
POP: NOT Not

FORMAT: not(Y)

FUNCTION: If C(Y) = 0, set C(Y) 0-17 = 1
If C(Y) ≠ 0, set C(Y) = 0

EXAMPLES:

not(alpha)




COMMENT: The purpose of this pop is to invert 000000 | 000000 (0, 18, 35) and 000001 | 000000 (0, 18, 35). The interpreter treats any operand that is non-zero as if it were 000001 | 000000 (0, 18, 35).

D. ROLL MANIPULATION POPS

1. Normal Roll Manipulation

Pops

POP: POB Put on bottom

FORMAT: pob(N)

FUNCTION: 1. Bump bottom of roll N by 1 word

2. Set C(word 1) = C(WO)

POP: POBP Put on bottom, and prune

FORMAT: pobp(N)

FUNCTION: 1. Bump bottom of roll N by 1 word

2. Set C(word 1) = C(WO)

3. Prune WO

POP: POBS Put on bottom from storage

FORMAT: pobs(Y)

FUNCTION: 1. Bump bottom of roll M by 1 word, where M=C(MRKER) 0-17

2. Set C(word 1) = C(Y)

POP: GOB Get off bottom

FORMAT: gob(N)

FUNCTION: 1. Set false, and go to next pop if C(TOP+N) 0-17 = C(BOTTOM+N) 0-17. Otherwise, set true and perform steps 2-3.

2. Add 1 to work counter

3. Set C(WO) = C(C(BOTTOM+N) 0-17 - 1); i.e., load the word immediately above the bottom of roll N

POP: GOBP Get off bottom, and prune

FORMAT: gobp(N)

FUNCTION: 1. Set false, and go to next pop if C(TOP+N) 0-17 = C(BOTTOM+N) 0-17. Otherwise, set true and perform steps 2-4.

2. Add 1 to work counter

3. Set C(WO) = C(C(BOTTOM+N) 0-17 - 1); i.e., load the word immediately above the bottom of roll N

4. Set C(BOTTOM+N) 0-17 = C(BOTTOM+N) 0-17 - 1; i.e., prune one word from the bottom of roll N

POP: CNT Count roll

FORMAT: cnt(N)

FUNCTION: 1. Add 1 to work counter

2. Set C(WO) = C(BOTTOM+N) - C(TOP+N) This calculation gives the number of words between the top and the bottom of the roll; i.e., the number of unreserved words in use.

POP: CNTG Count group

FORMAT: cntg(N)

FUNCTION: Count the last variable size group on roll N.

1. Assume that RP(ROLPTR+N) is the location of the VSW of the last group on roll N. Count the group (w words)
w = C(BOTTOM+N) 0-17 -RP(ROLPTR+N)-1

2. Set C(VSW) 0-17 = w

EXAMPLE:

cntg(6)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TOP+6 | 100000 | 000000 | | BOTTOM+6 | 100107 | 000000 | |
| | 0 | 18 | 35 | | 0 | 18 | 35 |
| ROLPTR+6 | 100 | x | 6 | RP(ROLPTR+6) = 100100 | | | |
| | 0 | 18 | 30 35 | | | | |
| VSW after | 000006 | unchanged | | (100107-100100-1 = 6) | | | |
| | 0 | 18 | 35 | | | | |

Illustration

Roll 6

| | | | |
|---|---|---|---|
| 100000 | | | |
| 100100 | 000006 | 000000 | Variable size word |
| 100101 | first word | | |
| | | | |
| 100107 | | | Bottom |
| | 0 | 18 35 | |

COMMENT: In executing this pop, the interpreter ignores C(ROLPTR+N) 30-35.

POP: PRU Prune roll

FORMAT: pru(N)

FUNCTION: Set C(BOTTOM+N) 0-17 = C(TOP+N) 0-17

POP: PTP Prune to pointer

FORMAT: ptp(Y)

FUNCTION: Set C(BOTTOM+C(Y) 30-35) 0-17 = RP(Y)

POP: PTPP Prune to pointer in work and prune

FORMAT: ptpp( )

FUNCTION: 1. Set C(BOTTOM+C(W0) 30-35) 0-17 = RP(W0)

2. Prune W0

COMMENT: Assume C(C1) = 000001 000000
0 18 35

In this case, ptpp( ) is equivalent to ptp(w0)
prw(c1)

POP: PBCT Prune by count

FORMAT: pbct(N)

FUNCTION: Set C(BOTTOM+N) 0-17 = C(BOTTOM+N) 0-17 - C(W0) 0-17; i.e., prune C(W0) 0-17 words from the bottom of roll N

POP: PBCTP Prune by count, and prune

FORMAT: pbctp(N)

FUNCTION: 1. Set C(BOTTOM+N) 0-17 = C(BOTTOM+N) 0-17 - C(W0) 0-17

2. Prune W0

POP: PTCT Prune to count

FORMAT: ptct(N)

FUNCTION: Set C(BOTTOM+N) 0-17 = C(TOP+N) 0-17 + C(W0) 0-17; i.e., prune roll N, so that there are C(W0) 0-17 words from top to bottom

POP: PTCTP Prune to count, and prune

FORMAT: ptctp(N)

FUNCTION: 1. Set C(BOTTOM+N) 0-17 = C(TOP+N) 0-17 + C(W0) 0-17

2. Prune W0

POP: PLG Prune last group

FORMAT: plg(N)

FUNCTION: Set C(BOTTOM+N) 0-17 = C(BOTTOM+N) 0-17 -K, if the difference $\geq$ C(TOP+N) 0-17. Otherwise, do not prune group.

K = C(GRPSIZ+N) 0-17, if C(GRPSIZ+N) 0-17 is non-zero

K = C(VARSIZ) 0-17 + 1 if C(GRPSIZ+N) 0-17 is zero.

POP: REMOV Remove

FORMAT: remov(N)

FUNCTION: Make all but one of the words from anchor to floor of roll N available to other rolls, and put roll N on the list of removed rolls.

Set C(BOTTOM+N) = 0, and set C(TOP+N) = information to be used by the interpreter.

POP: OPN Open

FORMAT: opn(N)

FUNCTION: Case 1: C(BOTTOM+N) $\neq$ 0; i.e., the roll is already open

Go to next pop

Case 2: C(BOTTOM+N) = 0

1. Take roll N off the list of removed rolls
2. Set C(BOTTOM+N) and C(TOP+N) both equal to C(ANCHOR+N)

NOTE: If all of the available words on the removed roll were used, C(ANCHOR+N) points to the remaining word on roll N. Otherwise, C(ANCHOR+N) points to the first unused word on roll N.

POP: RSV Reserve

FORMAT: rsv(N)

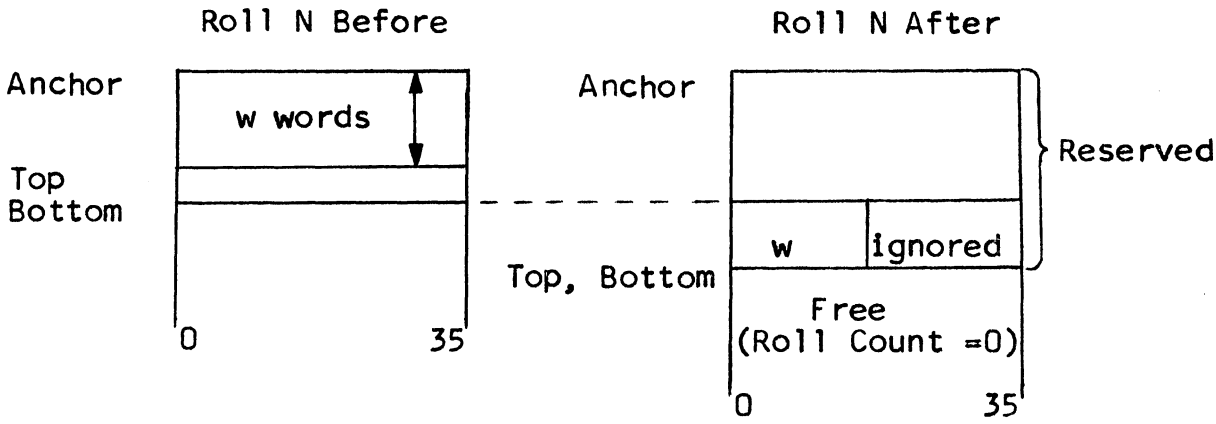
FUNCTION: 1. Compute the number of words currently reserved (w words) w = C(TOP+N) 0-17 - C(ANCHOR+N) 0-17

2. Bump the bottom of roll N by 1 word

3. Set C(word 1) 0-17 = w. Ignore the lower half of word 1.

4. Set C(TOP+N) 0-17 = C(BOTTOM+N) 0-17

This is illustrated below:




POP: REL Release

FORMAT: rel(N)

FUNCTION: Case 1: C(TOP+N) 0-17 = C(ANCHOR+N) 0-17; nothing is reserved

Set C(BOTTOM+N) 0-17 = C(TOP+N) 0-17

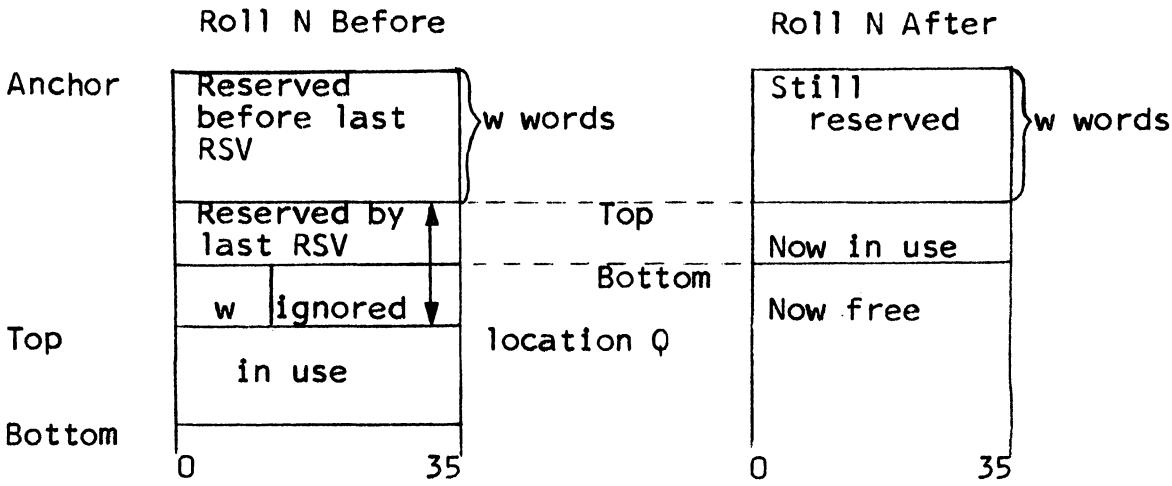
Case 2: Roll N contains one or more reserved words

1. Set C(BOTTOM+N) 0-17 = Q-1, where Q = C(TOP+N) 0-17

2. Recover w, C(Q-1) 0-17 (See RSV pop.)

3. Set C(TOP+N) 0-17 = C(ANCHOR+N) 0-17 + w

This is illustrated below:




POP: RSVM Reserve and mark

FORMAT: rsvm(N)

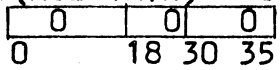
FUNCTION: 1. Execute mrk(N)

2. Execute rsv(N)

POP: DNX Down next word

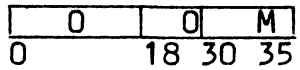
FORMAT: dnx(N)

FUNCTION: 1. Case 1: C(ROLPTR+N) = 0; i.e., C(ROLPTR+N) =



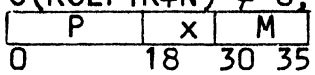

(Points to bottom.)

Set C(ROLPTR+N) =



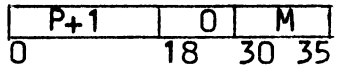

, where M = N. (Points to top.)

Case 2: C(ROLPTR+N) ≠ 0; i.e., C(ROLPTR+N) =




, where M is usually N.

Set C(ROLPTR+N) =



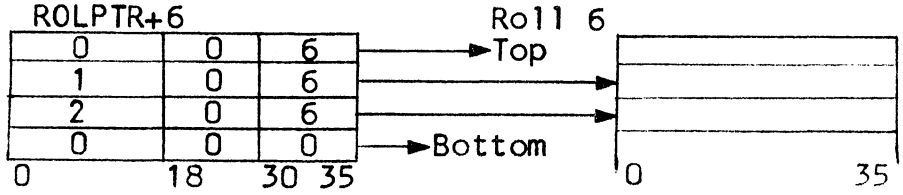

(Points one word down.)

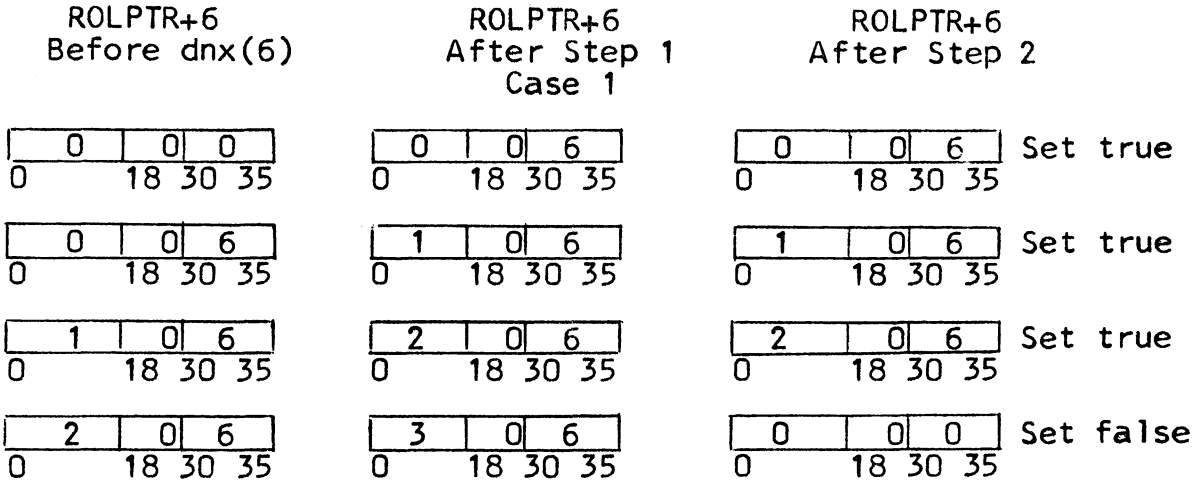
2. If roll M is removed, set C(ROLPTR+N) = 0, and set false

   If RP(ROLPTR+N) ≥ C(BOTTOM+M) 0-17, set C(ROLPTR+N) = 0, and set false*

   If RP(ROLPTR+N) < C(BOTTOM+M) 0-17, set true

EXAMPLE: Assume initial C(ROLPTR+6) = 0




*If roll M is the current read-spill roll (see Paragraph D.2.a), the interpreter does the following:

1. Determine whether there are more groups in the current read-spill segment

2. If there are no more groups, set C(ROLPTR+N) = 0, and set false

   If there are more groups, set C(ROLPTR+N) 0-17 = C(BOTTOM+M) 0-17 - C(TOP+M) 0-17, append next section of groups to bottom of roll M, and set true

POP: UNX Up next word

FORMAT: unx(N)

FUNCTION: Let Q = C(BOTTOM+N) 0-17 - C(TOP+N) 0-17

1. Case 1: C(ROLPTR+N) = 0; i.e.,
C(ROLPTR+N) = | 0 | 0 | 0 |
0 18 30 35

Set C(ROLPTR+N) = | P | 0 | N |, where P = Q
0 18 30 35

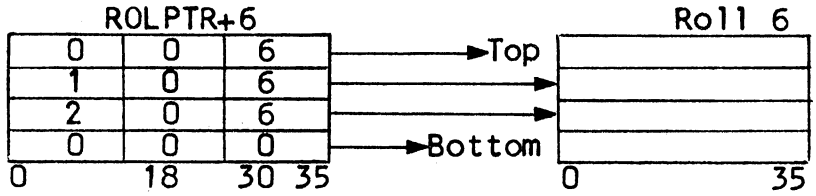
Case 2: C(ROLPTR+N) ≠ 0; i.e.,
C(ROLPTR+N) = | P | ignored |
0 18 35

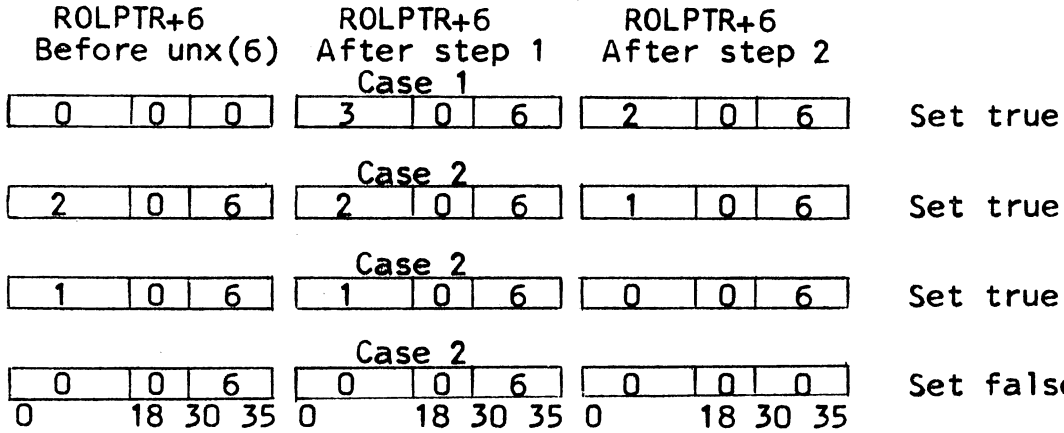
Do not change C(ROLPTR+N)

2. If P = 0, then set C(ROLPTR+N) = | 0 | 0 | 0 |,
and set false
0 18 30 35

If P ≠ 0, then set C(ROLPTR+N) = | P-1 | unchanged |,
and set true
0 18 35

EXAMPLE: Assume initial C(ROLPTR+6) = 0




COMMENTS: 1. The interpreter assumes that roll N is not removed

2. N overrides C(ROLPTR+N) 30-35

3. No special action is taken for the read-spill roll.

POP: DLOAD Down and Load

FORMAT: dload(N)

FUNCTION: 1. Execute the pop dnx(N)

2. If true condition was set, add 1 to work counter; and set C(W0) = C(RP(ROLPTR+N)) -- i.e., load the word pointed to
If false condition was set, do not change work

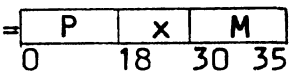
POP: DNG Down next group

FORMAT: dng(N)

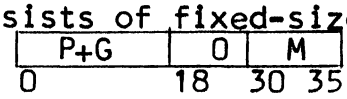
FUNCTION: Let G = C(GRPSIZ+N) 0-17 and let
V = C(RP(ROLPTR+N)) 0-17

Execute the pop dnx(N), with the following exceptions:

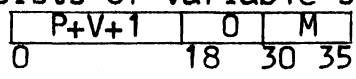
In step 1, case 2; i.e., C(ROLPTR+N) =




If roll N consists of fixed-size groups, set C(ROLPTR+N) =




If roll N consists of variable-size groups, set C(ROLPTR+N) =




POP: ULOAD Up and load

FORMAT: uload(N)

FUNCTION: 1. Execute the pop unx(N)

2. If true condition was set, add 1 to work counter; and set C(W0) = C(RP(ROLPTR+N)) -- load the word pointed to

If false condition was set, do not change work

POP: UNG Up next group

FORMAT: ung(N)

FUNCTION: Let G = C(GRPSIZ+N) 0-17. (Here, G must be non-zero)
Execute the pop unx(N) with the following exception:

In step 2, if P ≠ 0, set
C(ROLPTR+N) =

| P - G | unchanged |
|---|---|
| 0 | 18 35 |

COMMENT: This pop required fixed-size groups.

POP: CPY Copy

FORMAT: cpy(N)
M = C(MRKER) 0-17

FUNCTION: 1. Count roll N, to determine the number of words between the top and bottom of the roll.
w = number of words counted

2. If w = 0, take no further action.

If w ≠ 0, bump the bottom of roll M by w words; and copy w words (top to bottom) from roll N into this w-word area of roll M. (Words are copied sequentially.)

EXAMPLE: mrk(5)
cpy(6)

| | | |
|---|---|---|
| TOP+5 | 100000 | 000000 |
| TOP+6 | 101000 | 000000 |

0 18 35

BOTTOM+6

| | |
|---|---|
| 101002 | 000000 |

0 18 35

BOTTOM+5 before

| | |
|---|---|
| 100003 | 000000 |

0 18 35

BOTTOM+5 after

| | |
|---|---|
| 100005 | 000000 |

0 18 35

Illustration

Before pop

| | | | |
|---|---|---|---|
| 100000 | 2 | 0 | Top roll 5 |
| | 100 | 0 | |
| | 200 | 0 | |
| | | | Bottom |
| 101000 | 1 | 0 | Top roll 6 |
| | 1000 | 0 | |
| | | | Bottom |

0 18 35

After pop

| | | | |
|---|---|---|---|
| 100000 | 2 | 0 | Top roll 5 |
| | 100 | 0 | |
| | 200 | 0 | |
| | 1 | 0 | |
| | 1000 | 0 | |
| | | | Bottom |
| 101000 | 1 | 0 | Top roll 6 |
| | 1000 | 0 | |
| | | | Bottom |

0 18 35

POP: CPYR Copy and release

FORMAT: cpyr(N)

FUNCTION: 1. Execute cpy(N)

2. Execute rel(N)

POP: CPYP Copy and prune

FORMAT: cpyp(N)

FUNCTION: 1. Execute cpy(N)

2. Set C(BOTTOM+N) 0-17 = C(TOP+N) 0-17

POP: CPYG Copy group

FORMAT: cpyg(Y)
X and Y are locations of roll pointers
X = C(MRKER) 0-17

FUNCTION: Copy the group starting at location RP(Y) (group A) to the group starting at location RP(X) (group B). Either group may be fixed-size or variable-size. (If variable-size, the roll pointer points to the VSW.) Group sizes need not be the same.

1. Determine group size of each group, as follows: G = C(GRPSIZ+M) 0-17 , where M = roll number

   If G ≠ 0, group size equals G

   If G = 0, group size equals C(VSW) 0-17

2. Copy group A into group B, according to the following rules:

   a. If group A is a variable size group, do not copy the VSW. Begin by copying the word following the VSW.

   b. If group B is a variable size group, do not change the VSW. Copy the first word from group A into the location following the VSW.

   c. Copy words sequentially

d. If the group size of group A = N words, and the group size of group B > N words, then copy group A into the first N words of group B and set the remaining words of group B to zero.

e. If the group size of group B = N words, and the group size of group A ≥ N words, then copy the first N words of group A into group B.

EXAMPLE: mrk(rolptr+6)
cpyg(rolptr+5)

Assume RP(ROLPTR+5) is the location of the first word of the following fixed-size group on roll 5:

| word 1 |
|---|
| word 2 |
| word 3 |
| word 4 |

The following illustrations show how this group would be copied into different types of groups on roll 6:

| 3-word Fixed size | 5-word Fixed size | 4-word Fixed size | 3-word Variable size | |
|---|---|---|---|---|
| word 1 | word 1 | word 1 | VSW | unchanged |
| word 2 | word 2 | word 2 | word 1 | |
| word 3 | word 3 | word 3 | word 2 | |
| | word 4 | word 4 | word 3 | |
| | zeros | | | |

POP: CPYGB Copy variable-size group from Y pointer to bottom of marked roll

FORMAT: cpygb(Y)
Y is the location of a roll pointer
M = C(MRKER) 0-17

FUNCTION: In the following discussion, RP(Y) is the location of the VSW of the group to be copied, and C(VSW) 0-17 = V

1. Set C(VARSIZ) 0-17 = V

2. Bump bottom of roll M by V+1 words

3. Set C(word 1) 0-17 = V
Set C(word 1) 18-35 = 0

4. Copy remaining words (if any) into words 2 through V+1

EXAMPLE: cpygb(rolptr+5)

GRPSIZ+5 000000 000000
GRPSIZ+6 000000 000000
0 18 35

TOP+5 100000 000000
TOP+6 101000 000000
0 18 35

ROLPTR+5 10 0 5
0 18 30 35

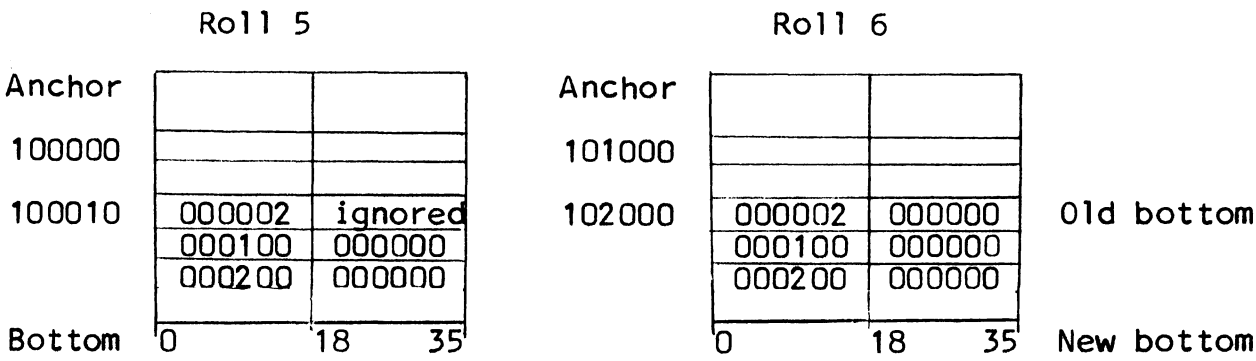
BOTTOM+6 102000 000000
0 18 35

MRKER 000006 000000
0 18 35

VARSIZ after 000002 unchanged
0 18 35

BOTTOM+6 after 102003 000000
0 18 35

Illustration




POP: CPYX Copy expression

FORMAT: cpyx(N)
M = C(MRKER) 0-17

FUNCTION: Case 1: Roll N is empty

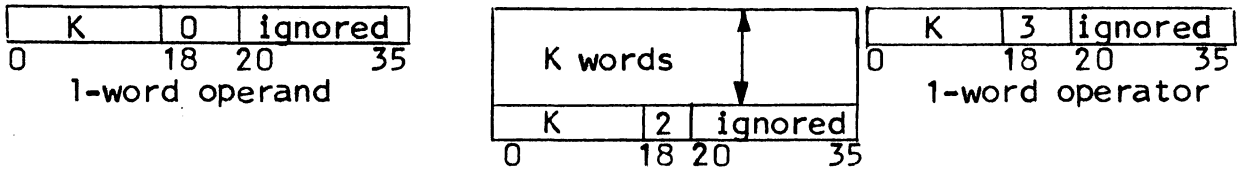
Set false

Case 2: Roll N is not empty

1. Set true

2. Determine the length of the expression to be copied, using the following rules:

a. The elements of the expression are either operands or operators:




where: $K \geq 1$

b. Unless the expression consists only of one operand, its first element is an operator

c. Elements are stored backward (from bottom to top). The first element appears immediately above the bottom, but the last element does not necessarily end at the top.

d. The interpreter scans the elements from bottom to top. It initially sets a counter to 1. Each operand subtracts 1 from the counter. Each operator adds K-1 to the counter. When the count reaches 0, the expression ends.

3. Bump bottom of marked roll by number of words to copy

4. Copy expression onto the marked roll.

EXAMPLE:

Z = A + B FORTRAN expression
= Z + A B Polish expression

Value f counter, after scanning of each element

| Location | Representation on Roll | | | |
|---|---|---|---|---|
| B 100000 | Offset of B in roll 0 | 0 | anything | 0 |
| A 100001 | Offset of A in roll 0 | 0 | anything | 1 |
| + 100002 | 2 | 3 | code for + | 2 |
| Z 100002 | Offset of Z in roll 0 | 0 | anything | 1 |
| = 100004 | 2 | 3 | code for = | 2 |
| | 0 | 18 20 | 35 | 1 (initial value) |

B, A, and Z are on links in roll 0. Here, the offsets are the locations of the VSW's for these links.

POP: CPYXP Copy expression, and prune

FORMAT: cpyxp(N)

FUNCTION: 1. Execute cpyx(N)

2. If true, set C(BOTTOM+N) 0-17 = location of last word in the expression; i.e., the word nearest the top of the roll.

POP: ZBG Zero bottom group

FUNCTION: Let G = C(GRPSIZ+N) 0-17 and V = C(VARSIZ) 0-17

Case 1: $G \neq 0$

1. Make ROLPTR+N point to the bottom of roll N
2. Bump bottom of roll N by G words
3. Set C(word 1) = 0, ..., C(word G) = 0

Case 2: G = 0

1. Make ROLPTR+N point to the bottom of roll N
2. Bump bottom of roll N by V+1 words
3. Set C(word 1) 0-17 = V

   Set C(word 1) 18-35 = 0
4. Set C(word 2) = 0,..., C(word V+1) = 0, if V>0

POP: SORTR Sort roll

FORMAT: sortr(N)
Roll N consists of 2-word groups; the first word of each group is the key.

FUNCTION: sort the groups of roll N (from top to bottom) so that the keys are in ascending logical order; i.e., the keys are interpreted as signless 36-bit numbers.

EXAMPLE:

sortr(5)

Roll 5 Before

| | | |
|---|---|---|
| top | 12 | 0 |
| | 4 | 0 |
| | 1 | 0 |
| | 5 | 0 |
| | 1003 | 0 |
| | 6 | 0 |
| bottom | | |

0 18 35

Roll 5 After

| | | |
|---|---|---|
| top | 1 | 0 |
| | 5 | 0 |
| | 12 | 0 |
| | 4 | 0 |
| | 1003 | 0 |
| | 6 | 0 |
| bottom | | |

0 18 35

COMMENT: This pop is used to sort addresses or one-word symbols. It is not used for sorting signed numbers; in this case, the positive numbers would precede the negative numbers.

2. Manipulation of Spill Rolls

a. Spill Rolls

The following discussion covers the use of spill rolls by a two-pass compiler or assembler. These rolls cannot be used by a one-pass procedure.

Assume N-1 is the last roll used, where N = C(OPNERS) 0-17. Rolls N-1 and N-2 are used as read or write spill rolls, as described below:

Pass 1:

N-1 -- Write-spill roll
The interpreter writes data on this roll

N-2 -- Not used

Pass 2:

N-1 -- Read spill roll
The interpreter reads data from this roll

N-2 -- Write-spill roll (binary roll)
The interpreter writes data on this roll

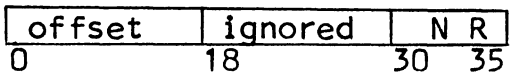
Pass 3: WBIN (See WBIN pop)

N-1 -- Not used

N-2 -- Read spill (binary roll)
The interpreter reads data from this roll, to produce text, linkage, and symbol segments during the execution of WBIN

RSPTR and WSPTR are data segment registers denoting offsets on the current read- and write-spill rolls, respectively:

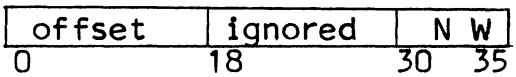
RSPTR




N R - Number of current read-spill roll

NOTE: If C(RSPTR) = 0, there is no current read-spill roll

WSPTR




N W - Number of current write-spill roll

NOTE: If C(WSPTR) = 0, there is no current write-spill roll

Words above the location denoted by RSPTR may be discarded. Words above the location denoted by WSPTR may be written on an auxiliary data segment, called a write-spill segment. During pass 1, the write-spill segment is spill segment 1; any data written on spill segment 1 is read during pass 2. During pass 2, the write-spill segment is spill segment 2; any data written on spill segment 2 is read during pass 3.

The first section of data from the appropriate spill segment is read via a RWND pop; remaining sections are read via DNG pops (see the discussions of these pops for details).

The read- and write-spill rolls consist of variable-size groups. Figure 2 illustrates the setup of these rolls during pass 2:

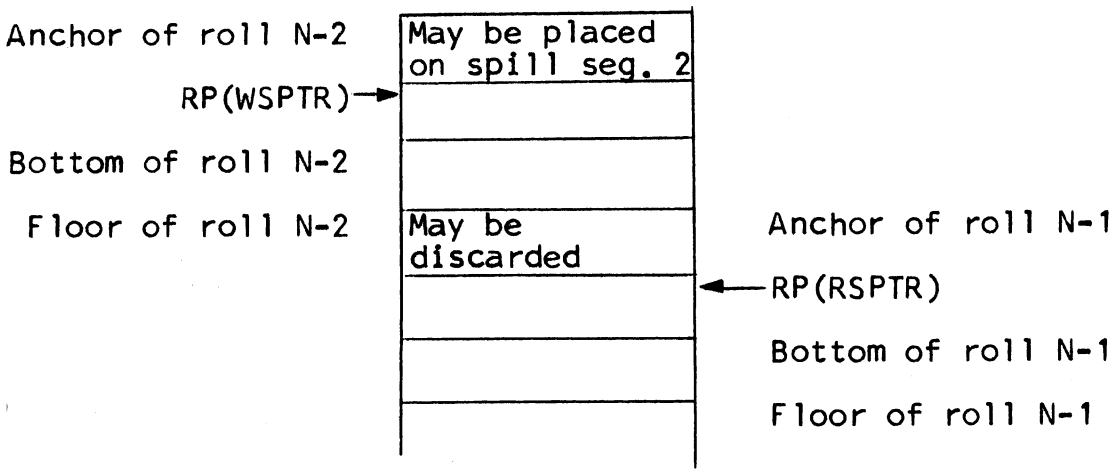



Figure 2. Setup of Read-Spill and Write-Spill Rolls During Pass 2

The following information is true for both the read-spill and the write-spill rolls:

1. Initially, anchor, top, and bottom are at the same location.
2. Thereafter, anchor is the location of the first word in use; bottom is one word after the last word in use, and the distance between top and anchor = total number of words released.
3. The only significance of top is for roll pointer addressing; i.e., the offset of any word on either of these rolls is fixed, even if information has been released above it.

b. Manipulation of Space

Manipulation of spill rolls is summarized below:

Pass 1 -- N W = N-1

Assume: 1. Interpreter tried to bump bottom of some roll, but there was insufficient space in the data segment.

2. C(WSPTR) ≠ 0

3. RP(WSPTR) > C(ANCHOR+N W) 0-17; i.e., there is available space to release, starting with the anchor and ending with RP(WSPTR) -1.

Action: 1. Copy words from available space to spill segment 1 (starting with next free location in spill segment 1).

2. Release the space on roll N W

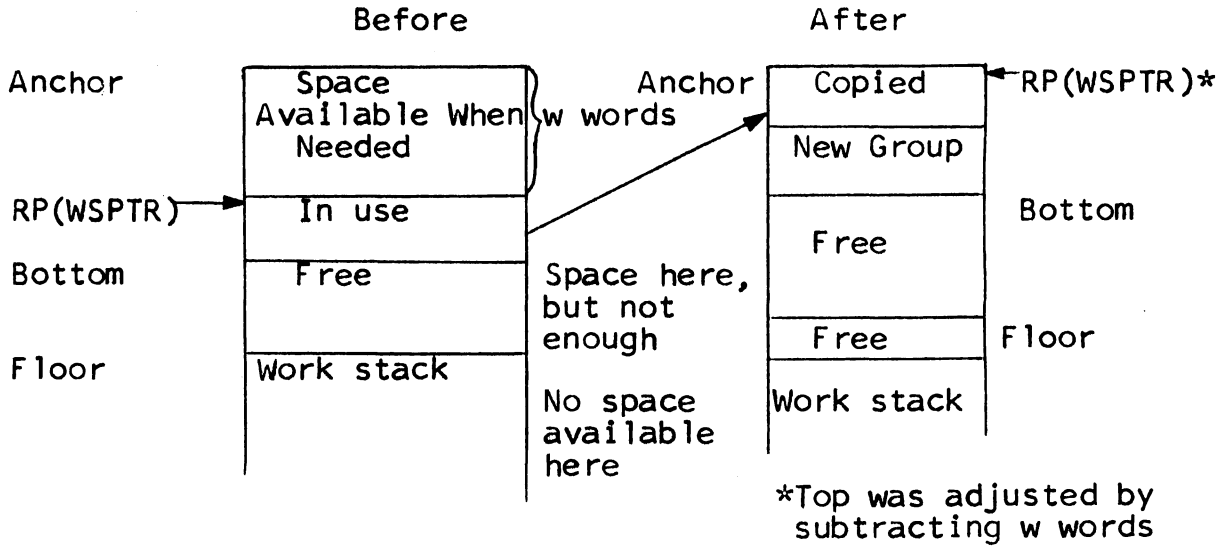
3. Set C(ANCHOR+N W) 0-17 = RP(WSPTR)

4. Perform any necessary roll movements

Result: C(ANCHOR+N W) 0-17 - C(TOP+N W) 0-17 = cumulative number of words written on spill segment 1

Example:

zbg(N W) Here, the write-spill roll is the expanding roll



Pass 2 -- N R = N-1

Assume: 1. Interpreter tried to bump bottom of some roll, but there was insufficient space in the data segment.

2. C(RSPTR) ≠ 0

3. RP(RSPTR) > C(ANCHOR+N R) 0-17; i.e., there is available space to release, starting with the anchor and ending with RP(RSPTR) -1.

Action: 1. Release space

2. Set C(ANCHOR+N R) 0-17 = RP(RSPTR)

3. Perform any necessary roll movements

Result: C(ANCHOR+N R) 0-17 - C(TOP+N R) 0-17 = Cumulative number of words released

Comment: In pass 2, roll N-2 may obtain space from roll N-1, without moving any words. The interpreter merely adjusts the floor of roll N-2, which is always the anchor or roll N-1. The interpreter releases only the required number of words from roll N-1, and saves the remaining releasable words for future allocation. During pass 2, usually more words are read than written. Therefore, the read-spill roll (N-1) should make space available to the write-spill roll (N-2) fast enough to eliminate the need for moving any words. In pass 2, the write-spill roll is manipulated in the manner described for pass 1. In this case, however, the write-spill roll is N-2, and released words are copied into spill segment 2.

RWND Pop

POP: RWND Rewind

FORMAT: rwnd(N)
N is the number of the current read-spill or write-spill roll

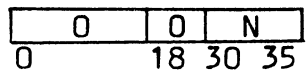
FUNCTION: Case 1: If C(RSPTR) 30-35 = N, then rewind the read-spill roll

1. Set C(TOP+N) = C(ANCHOR+N)
   Set C(BOTTOM+N) = C(ANCHOR+N)

2. Set C(RSPTR) = 0

Case 2: If C(RSPTR) 30-35 ≠ N, then rewind the write-spill roll

1. Set C(RSPTR) =




   Set C(WSPTR) = 0

2. If current write-spill segment is empty, go to next pop
   Otherwise, perform steps a, b, and c

a. Copy remaining words from roll N (anchor to bottom) to write-spill segment. (This segment now becomes the read-spill segment.)

b. Set C(TOP+N) = C(ANCHOR+N)

   Set C(BOTTOM+N) = C(ANCHOR+N)

c. Get first section of groups (implementation dependent) from new read-spill segment and put them on the bottom of roll N; i.e., bump the bottom of roll N by the appropriate number of words, and copy the words from the read-spill segment to roll N.

COMMENT: The RWND pop should be used as follows:

At end of pass 1 -- rwnd(N-1)

At end of pass 2 -- rwnd(N-1)
rwnd(N-2)

E. CONTROL POPS

Pops

POP: JMP Jump

FORMAT: jmp(Y)
Y is a location in the procedure segment

FUNCTION: 1. Set pop counter equal to Y

2. Execute the pop at location Y

POP: JMPP Prune and jump

FORMAT: jmpp(Y)
Y is a location in the procedure segment

FUNCTION: 1. Prune W0

2. Set pop counter equal to Y

3. Execute the pop at location Y

POP: JNX Jump on no index

FORMAT: jnx(Y)
Y is a location in the procedure segment

FUNCTION: If C(W0) 7-17 = 0, then execute jmpp(Y)

If C(W0) 7-17 ≠ 0, then set C(W0) 7-17 = C(W0) 7-17 - 1

NOTE: C(W0) 7-17 is called the count field.

EXAMPLE:

In this example, control passes to jmp(loop) three times; then, W0 is pruned, and control passes to done.

```
      eaw(3)
loop:pop
       .
       .
       .
     jnx(done)
     jmp(loop)
```

POP: DOML Do machine language

FORMAT: doml(Y)

Y is a location in the procedure segment. It is the location of the first word of a machine language program

FUNCTION: Execute the GE-645 instruction tra(Y)

COMMENT: Assume that ALPHA is the current C(X1), and that BETA is the location of the next pop to be interpreted after leaving the machine language program. At any location GAMMA, the machine language program may return to the interpreter as follows:

| Case | Instruction(s) |
|---|---|
| BETA = ALPHA | gamma: tra(popset,ri) |
| BETA = ALPHA+1 | gamma: tra(next,ri) |
| BETA = GAMMA+1 | gamma: trx(x1,popset,ri) |
| Otherwise | gamma: ldx(x1,beta,du)<br>tra(popset,ri) |

POPSET and NEXT are locations of ITS pairs in the data segment.

Each ITS pair points to a location in the interpreter segment.

POP: EXEC Execute

FORMAT: exec(Y)
Y is a location in the procedure segment

FUNCTION: Execute the pop at location Y, using the current value of the pop counter; i.e., do not set the pop counter to location Y.

EXAMPLE:

The following is part of a conversion routine to handle

principal part, decimal scale, and binary scale.

Assume C(W0) 0-17 = 0 if principal part is to be converted

1 if decimal scale is to be converted

2 if binary scale is to be converted

The following code appears in the procedure segment:

```
table: cona(char)
       conda(char)
       conba(char)
```

The code below could be used to convert a character

```
xntv(w0)
exec(table)    "execute one of the above three instructions"
jmp(endn)      "go here if end-of-line"
next pop       "go here otherwise"
```

COMMENT: If the executed pop is a JSB pop (see Paragraph F), then the exit stack records the location following the EXEC pop (not the location following the JSB pop).

F. SUBROUTINE POPS

Pops

POP: JSB Jump to subroutine

FORMAT: jsb(Y)
Y is a location in the procedure segment. It is the location of an entry word in a subroutine

FUNCTION: 1. Add 2 to exit counter

2. Set C(word 1) 0-17 = L + 1, where L is the location of the current JSB pop

Set C(word 1) 18-35 = current work size

Set C(word 2) = 0 (i.e., set false)

3. Execute jmp(Y)

POP: EXIT Exit

FORMAT: exit(N)
N is usually 0.

FUNCTION: 1. Prune two words from the exit buffer

2. Jump to location L + N + 1, where L is the location of the last executed JSB pop.

3. Restore true/false status that existed before the last JSB was executed.

POP: EXITP Prune and exit

FORMAT: exitp(N)

FUNCTION: 1. Execute PRWX pop

2. Execute exit(N) pop

COMMENT: exitp(N) is equivalent to prwx( )
exit(N)

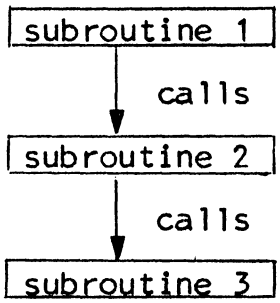
POP: PRE Prune exit

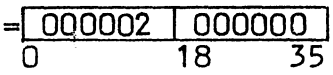
FORMAT: pre(Y)

FUNCTION: Subtract 2 * C(Y) 0-17 from exit counter; i.e., prune exit stack by C(Y) 0-17 pairs of words

EXAMPLE:

Assume a program contains the following subroutine calls:




Assume C(C2) = 


To return directly from a location in subroutine 3 to location ALPHA in subroutine 1, the user would write:

```
pre(c2)
jmp(alpha)
```

G. ADDRESS SUBSTITUTION POPS

The locations W0 through W5, and D0 through D5 are treated as pops when they appear as operands with other pops; e.g., add(w0). Another group of pops, A0 through A5, are also in this category (see Argument Pops below).

Pops

POP: The work pops: W0, W1, W2, W3, W4, and W5

FORMAT: pop(Wn)

n = 0, 1, 2, 3, 4, or 5

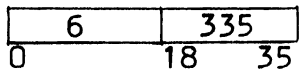
FUNCTION: 1. Form a new pop whose left half is the location of Wn and whose right half is "pop"

2. Execute this new pop

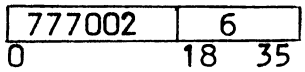
EXAMPLE:

Assume that the work counter is set at 777005.

add(w3) look like this:




The interpreter converts this to:




This means: Set C(W0) = C(W0) + C(777002)

COMMENT: If a work pop has a true or false tag, the tag is added to Wn, and the test is made before step 1.

POP: The dummy pops: D0, D1, D2, D3, D4, and D5

FORMAT: pop(Dn)

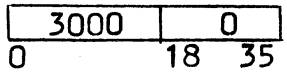
n = 0, 1, 2, 3, 4, or 5

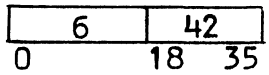
FUNCTION: 1. Form a new pop whose left half is C(Dn) 0-17 and whose right half is "pop"

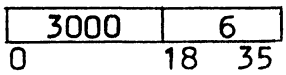
2. Execute this new pop

EXAMPLE:

Assume that the dummy counter is set at 777405

777404 


add(d1) looks like this: 


The interpreter converts this to: 


This means: Set C(W0) = C(W0) + C(3000)

COMMENT: If a dummy pop has a true or false tag, the tag is added to Dn, and the test is made before step 1.

POP: The argument pops: A0, A1, A2, A3, A4, and A5

FORMAT: pop(An)

n = 0, 1, 2, 3, 4, or 5

FUNCTION: Get the argument of "pop" from the argument list of the last executed JSB pop.

Assume the last executed JSB pop is at location L.

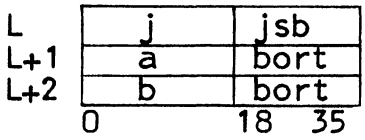
| Location | Pop |
|---|---|
| L+0 | jsb(Y0) |
| L+1 | bort(Y1) |
| L+2 | bort(Y2) |
| L+3 | bort(Y3) |
| L+4 | bort(Y4) |
| L+5 | bort(Y5) |

NOTE: The subroutine should be terminated by the pop exit(5), so that none of the BORT pops will be executed.

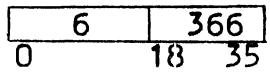
1. Form a new pop whose left half is Yn and whose right half is "pop"
2. Execute this new pop

EXAMPLE:

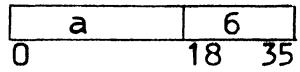
Assume that the last JSB pop is at location L and that locations L through L+2 contain the following data:




add(a1) looks like this:




The interpreter converts this to:




This means: Set C(WO) = C(WO) + C(a)

COMMENTS:

1. If an argument pop has a true or false tag, the tag is added to An, and the test is made before step 1.
2. The use of BORT pops in forming the argument list insures against a possible bug in the pops procedure. If control accidentally reaches one of these pops, the interpreter will abort the procedure.
3. The number of BORT pops depends on the number of arguments. If there are more than 5 arguments, then an XNDW pop must precede any reference to the sixth (or succeeding) argument.
4. The argument indices and the exit indices are off by one; e.g., termination of a subroutine by exit(0) would set the pop counter to location L+1.

H. INDEX POPS

Each index pop is followed by another pop, and it modifies the operand of this following pop. This modification affects only the execution of the modified pop; it does not change the representation of the pop in the procedure segment.

Pops

POP: XNTF Index next table by fixed

FORMAT: xntf(Y)
pop(Z)

FUNCTION: Use Y + Z instead of Z

EXAMPLE: See Figure 3

COMMENT: The following coding illustrates the use of the XNTF pop:

```
l:  xntf(3)
m: stor(table)
     .
     .
     .
   jmp(l) "To store in TABLE+3"
   jmp(m) "To store in TABLE"
```

POP: XNTV Index next table by variable

FORMAT: xntv(Y)
pop(Z)

FUNCTION: Use C(Y) 0-17 + Z instead of Z

EXAMPLE: See Figure 3

POP: XNIF Index next indirect by fixed

FORMAT: xnif(Y)
pop(Z)

FUNCTION: Use Y + C(Z) 0-17 instead of Z

EXAMPLE: See Figure 3

POP: XNIV Index next indirect by variable

FORMAT: xniv(Y)
pop(Z)

FUNCTION: Use C(Y) 0-17 + C(Z) 0-17 instead of Z

EXAMPLE: See Figure 3

POP: XNPF Index next pointer by fixed

FORMAT: xnpf(Y)
pop(Z)

FUNCTION: Use Y + RP(Z) instead of Z

EXAMPLE: See Figure 3

POP: XNFP Index next fixed by pointer

FORMAT: xnfp(Y)
pop(Z)

FUNCTION: Use RP(Y) + Z instead of Z

EXAMPLE: See Figure 3

POP: XNPV Index next pointer by variable

FORMAT: xnpv(Y)
pop(Z)

FUNCTION: Use C(Y) 0-17 + RP(Z) instead of Z

EXAMPLE: See Figure 3

POP: XNVP Index next variable by pointer

FORMAT: xnvp(Y)
pop(Z)

FUNCTION: Use RP(Y) + C(Z) 0-17 instead of Z

EXAMPLE: See Figure 3

POP: XNDW Index next work, dummy, or argument

FORMAT: xndw(Y)
pop(Z)

FUNCTION: Add C(Y) 0-17 to work, dummy, or argument address

EXAMPLE: See Figure 3

Data Segment Setup

Location

| | | |
|---|---|---|
| 100 | 5 | ignored |
| | | |
| 200 | 700 | ignored |
| | | |
| 300 | 2 | ignored |

0 18 35

Some characteristics of roll 5

C(TOP+5) 0-17 = 120000

ROLPTR+5 =

| | | |
|---|---|---|
| 1 | x | 5 |

0 18 30 35

RP(ROLPTR+5) = 120001

EXAMPLES:

```
xntf(3)
          = stor(403)
stor(400)

xnif(3)
          = stor(703)
stor(200)

xnpf(3)
          = stor(120004)
stor(rolptr+5)

xnpv(100)
          = stor(120006)
stor(rolptr+5)

xndw(300)
          = stor(w1)
stor(w3)

xndw(300)
          = stor(d1)
stor(d3)

xndw(300)
          = stor(a5)
stor(a3)
```

```
xntv(100)
          = stor(405)
stor(400)

xniv(100)
          = stor(705)
stor(200)

xnfp(rolptr+5)
          = stor(120004)
stor(3)

xnvp(rolptr+5)
          = stor(120006)
stor(100)
```

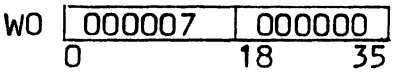
Figure 3. Examples of the Index Pops

General Comments

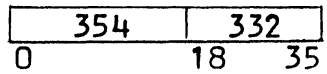
1. XNDW is the only indexing pop that can be followed by an address substitution pop.
2. XNFP and XNVP are faster than XNPF and XNPV, respectively.
3. The following example illustrates a complex use of the XNTV pop:

```
xntv(w0)
stor(400)
```

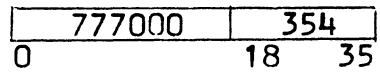
Assume W0 is at location 777000 in the data segment, and that it has the following format:




xntv(w0) looks like this:




The interpreter converts this to:




Thus, the modified pop is: stor(407)

I. MOVE POPS

Pops

POP: MOV Move

FORMAT: mov(Y)
to(Z)

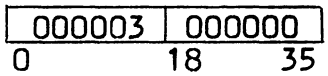
FUNCTION: Set C(Z) = C(Y) and skip the pop to (Z)

NOTE: The interpreter ignores the number of the pop following the MOV pop

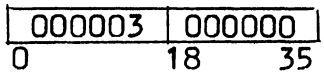
EXAMPLE:

```
mov(c3)
to(varsiz)
```

C3




VARSIZ after




COMMENTS: See TO

POP: TO

FORMAT: mov(Y)
to(Z)

FUNCTION: See MOV

COMMENTS: 1. The TO pop should not be executed alone. However, the user may allow a certain number of illegal executions of the TO pop to occur before an abort, by setting TOCNT, a one-word register in the data segment.

When a TO pop is executed alone, the interpreter adds 1 to C(TOCNT). If C(TOCNT) is 0 or positive, an abort occurs. If C(TOCNT) is negative, an error message is printed (e.g., EXECUTED TO POP AT 015610); and the interpreter executes the next pop.

2. The following rules apply to the MOV and TO pops:

   a. The TO pop must have the same tag (true, false, or none) as the MOV pop has

   b. The TO pop cannot be indexed

   c. The operand of the TO pop cannot be an address substitution pop

NOTE: The following code is permitted:

| Indexed Move | Move with Address Substitution Pop |
|---|---|
| xnpf(5)<br>mov(rolptr+6)<br>to(alpha) | mov(w3)<br>to(alpha) |

3. The following comparative code shows the advantage of the MOV pop:

| Fast | Slower | Even Slower |
|---|---|---|
| mov(Y)<br>to(Z) | cload(Y)<br>stor(Z) | load(Y)<br>storp(Z) |
| 2 pops in one | | Pushes down and pops up work |

POP: MOVF Move from

FORMAT: movf(Y)

FUNCTION: Set C(FROM) 0-17 = Y

FROM is a one-word register in the data segment. (The interpreter ignores C(FROM) 18-35)

EXAMPLE: See MOVT

COMMENTS: See MOVT

POP: MOVT Move to

FORMAT: movt(Y)

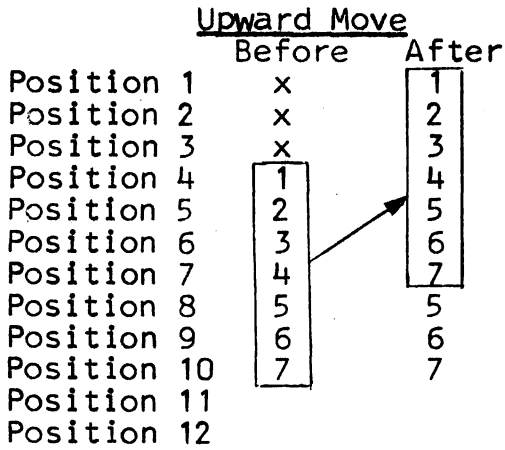
FUNCTION: Move n consecutive words from starting location C(FROM) 0-17 to starting location Y.
n = C(WO) 0-17
FROM is a one-word register in the data segment.
(The interpreter ignores C(FROM) 18-35.)

EXAMPLE: In this example, the words at locations ALPHA -- ALPHA+7 are moved to locations BETA -- BETA+7.

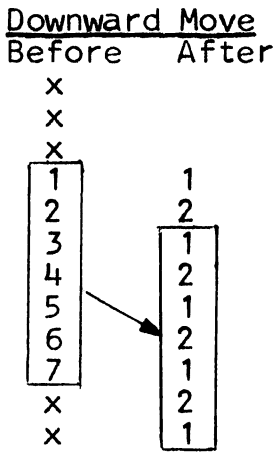
```
ceaw(7)
movf(alpha)
movt(beta)
```

COMMENTS:

1. Each MOVF overrides the preceding MOVF
2. Any number of MOVT pops may follow a MOVF pop
3. Any number of pops may be executed between a MOVF pop and a MOVT pop
4. There are no restrictions concerning true/false tags, or the use of address substitution pops
5. Words are moved in a forward sequence; thus, the upward move on the left works and the downward move on the right does not work:




Here, the numbers in the left column were moved up 3 positions to form the right column. The boxed numbers were moved.




Here, the numbers in the left column were moved down 2 to form the right column. The boxed numbers were moved.

J. INPUT POPS

1. Character Input

The source of character input is an input stream. There are two types of input streams: the source procedure, and strings. The source procedure is the ASCII text in the input segment; e.g., FORTRAN source procedure. String input consists of portions of type-1, type-2, or type-3 strings, as summarized below:

| string | Input Stream |
|---|---|
| Type-1 | Excludes count-character and characters not included in the count |
| Type-2 | Excludes end-of-file character and any characters that follow it |
| Type-3 | Excludes the two control words at the beginning of each group of the string |

Characters excluded from a string direct the interpreter in delimiting the string.

The source procedure comes from a file that was previously created on the console. Type-1 and type-2 strings come from SYMBUF or rolls. Type-3 strings come from rolls.

There is only one input stream at a time; this is called the current input stream. Initially, the current input stream is the source procedure.

The input stream can be changed by a SWIP or a SWAP pop. The SWIP pop nests input streams; this is similar to the nesting of subroutines by JSB and EXIT: swip(0) corresponds to the EXIT pop; and swip(Y), where Y $\neq$ 0, corresponds to the JSB pop. On the other hand, the SWAP pop is similar to the JMP pop, in that it changes input streams without changing nesting.

## 2. Input Registers

The input registers reflect the status of the current input stream. The status may be one of the following:

Initial -- Signifies beginning of line (applicable only to the source procedure)

Normal -- Signifies middle of line or string

End-of-file -- Signifies no more lines, if current input stream is the source procedure
Signifies no more characters, if current input stream is a string

The input registers are one-word registers in the data segment:

TLYIN -- GE-645 tally word. Points to next available character in the current input stream.

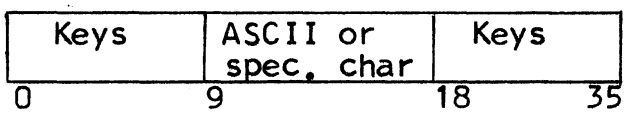
NOTE: TLYIN points to a location in the data segment. If the current input stream is the source procedure, this location is in a buffer containing information which the interpreter copies from the input segment.

MODES -- C(MODES) 0-17 = -1 if current input stream is the source procedure
Location of NXST file, if current input stream is a string (See NXST.)

C(MODES) 18-35 = 0 if current status is normal
-2 if current status in initial
-6 if current status is end-of-file

CHARC -- C(CHARC) 0-17 = Column number of current input character. (On a source procedure line, the leftmost character occupies column 1.)

CHAR -- Representation of current character in the input stream, as it appears in the TRANS table (See Chapter 1, Paragraph G.2.b.)



CRDNUM -- C(CRDNUM) 0-17 - Must be zero

C(CRDNUM) 18-26 = Number of last group processed in current type-3 string

C(CRDNUM) 27-35 - Must be zero

CRDNUM+1 -- C(CRDNUM+1) 0-17 - Must be zero

C(CRDNUM+1) 18-26 = Number of groups in current type-3 string

C(CRDNUM+1) 27-35 - Must be zero

SWIP and SWAP initialize all necessary input registers for a new current input stream. The character input subroutine shared by the next character and next string pops updates the input registers each time it fetches a character from the current input stream.

3. Character Input Pops

Pops

POP: NXCH Get next character

FORMAT: a: nxch( )

FUNCTION: 1. If status is end-of-file, execute the pop in location 2 in the procedure segment. This will usually be a JMP or a JSB pop.

If status is initial, then change it to normal; and get next character, if any.

If status is normal, then get next character, if any.

2. Case 1: If successful in getting another character

a. Add 1 to column number

b. If this character is to be skipped, get next character, if any, and go back to beginning of step 2.

c. If this character is octal 201, then set status to end-of-file and jump to location a+1 in the procedure segment.

If this character is octal 000, octal 001, . . ., or octal 177, then store the character and its keys in CHAR, and jump to location a+2.

Case 2: If unsuccessful in getting another character, and input is from the console

Try to get the next line:

If unsuccessful, set status to end-of-file, and jump to location a+1

If successful, set status to initial, set column number to 0, add 1 to alter number, and jump to location a+1

Case 3: If unsuccessful in getting another character, and input is from a roll

If this is the end of a type-1 or type-3 string, set end-of-file condition, and jump to location a+1

If this is a type-2 string, the unsuccessful condition cannot occur, since these strings are terminated with an end-of-file character (octal 201).

EXAMPLE: See PAKA, Paragraph K

POP: NXST Next string

FORMAT: a: nxst(Y)

Y is the location of a 3-word file:

| | | |
|---|---|---|
| Y | location of key word | roll # if non-0 SYMBUF if 0 |
| Y+1 | type of string 1, 2, or 3 | Pack-from option 0 = off non-0 = on |
| Y+2 | Used only by NXSTCS, SWIP, and SWAP | |

0 18 35

C(Y) 0-17 -- Location of word with keys in bits 0-8 and 18-35

C(Y) 18-35 -- Roll into which string is to be packed (or SYMBUF if 0)

C(Y+1) 0-17 -- Type of string to be formed

C(Y+1) 18-35 -- Option determining whether C(CHAR) 9-17 is to be packed into the string

FUNCTION: Pack the characters from the current input stream into SYMBUF or on a roll, forming a type-1, type-2, or type-3 string. Pack characters from left to right. If the string is being formed on a roll, start forming the string at the bottom of the roll. If the string is being formed in SYMBUF, assume that an RSYM pop was executed, and start forming the string in the first location in SYMBUF.

1. Initialize string according to its type

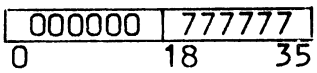
2. If the pack-from option is on, then pack C(CHAR) 9-17 into the string

3. Simulate NXCH pop, with exceptions noted below

4. Case 1: If successful, compare keys in CHAR with keys specified in key word. If one or more bits match, terminate the string according to type, and jump to location a+2 in the procedure segment. (Also perform Case 2.c) Otherwise, pack C(CHAR) 9-17 into the string, and go back to step 3.

   Case 2: If unsuccessful, perform the following steps:

   a. Terminate string according to type

   b. Set C(CHAR) = 


   c. If the string was formed on a roll, then make the corresponding roll pointer point to the location of the first word in string

   d. Jump to location a+1 in the procedure segment

COMMENT: On end-of-line, the interpreter goes to the next pop (same as NXCH).

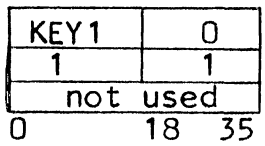
POP: NXSTC Next string continued

FORMAT: a: nxstc(Y)

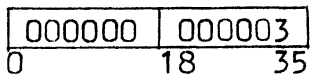
FUNCTION: Continue forming a string (with an option to pack the current character).
Execute NXST pop, eliminating step 1.

EXAMPLE: In this example, the interpreter forms a string in SYMBUF, beginning with a left parenthesis and ending with a right parenthesis.

FILE1




KEY1




Assume the keys have the following meaning:

Bit 34 -- 1 if ~ (escape character)
Bit 35 -- 1 if )

Assume that the current character is a left parenthesis.

Code:

```
     rsym( )
     nxst(file1)              "pack initial left parenthesis and
                               string up to break"
     jmp(error)               "special case if end of line"
more:scha(trans+octal(176))   "advance if escape"
     jmp(error)               "special case if end of line"
     jmp(done,f)              "right parenthesis encountered
                               instead"
     nxstc(file1)             "continue string with escaped
                               character"
     jmp(error)               "special case if end of line"
     jmp(more)                "test break character"
        .
        .
        .
done:pak(char)                "pack terminal right parenthesis"
     cnts( )                  "count the string"
```

COMMENT: On end-of-line, the interpreter goes to the next pop (same as NXCH).

POP: NXSTCS Next string continued and save

FORMAT: a: nxstcs(Y)
Y is the location of a file with the same format as the file for NXST. (See NXST.)

FUNCTION: Save next available position in type-3 string, and then continue forming string.

1. Set C(Y+2) = coded information describing current input character and next available position in string being formed

2. Execute nxstc(Y)

COMMENTS: 1. NXSTCS may not be used to form type-1 or type-2 strings

2. The pops swip(Y) and swap(Y) use the information in Y+2 to determine whether to switch to the beginning of an input stream or to a saved position in the stream.
If C(Y+2) = 0, swip(Y) or swap(Y) switches to the beginning of the stream. The pops procedure is responsible for clearing C(Y+2), whenever necessary.

3. On end-of-line the interpreter goes to the next pop (same as NXCH).

POP: NXICH Get next initial character

FORMAT: nxich( )

FUNCTION: If the current status is normal, get the first character of the next line
If the current status is end-of-file, execute the pop in location 2 of the procedure segment
If the current status in initial, get the first character of the current line

COMMENT: 1. This pop can only be used with the source procedure. It cannot be used with string input.

2. On end-of-line, the interpreter goes to the next pop (same as NXCH).

## 4. Changing Input Streams

### Pops

POP: SWAP Swap input streams

FORMAT: swap(Y)

Y is the location of a file with the same format as the file for NXST. (See NXST.)

If C(Y) 18-35 = 0, the new input stream is in SYMBUF

If C(Y) 18-35 = N, the new input stream is on roll N and RP(ROLPTR+N) is the location of the first word

FUNCTION: Change current input stream, as directed by the file at location Y.

If C(Y+2) = 0, set C(CHARC) = 0

Otherwise, set CHAR and CHARC according to coded information in C(Y+2) (See NXSTCS.)

COMMENTS:
1. If Y = 0, then the interpreter aborts the pops procedure
2. SWAP can only change the input stream to SYMBUF or a roll

POP: SWIP Switch input streams recursively

FORMAT: swip(Y), where Y ≠ 0 (Here, Y is the location of a file with the same format as the file for NXST.)
or
swip(0)

FUNCTION: Case 1: The operand is Y; Y≠0

1. Bump bottom of roll 4 (swip roll) by three words
2. Use these three words to record current position in current input stream for later use by a swip(0)
3. Execute swap(Y); i.e., change input streams as directed by the file at location Y

Case 2: The operand is 0

1. Get three words off the bottom of roll 4
2. Use these words to recover current position in a previous input stream
3. Change to previous input stream, as directed by these words
4. Prune roll 4 by three words

COMMENT: 1. It is illegal to change the input stream to a type-3 string using a swip(Y) pop, if a type-3 string is already nested

2. The swip(Y) pop can only change the input stream to SYMBUF or a roll
3. The input stream can be changed to the source procedure only by the appropriate number of swip(0) pops.

5. Changing Mode

Pops

POP: MODB Mode blank

FORMAT: modb( )

FUNCTION: Cause NXCH to accept any character; i.e., after this pop, NXCH will accept any character, until the next MODNB pop says otherwise.

POP: MODNB Mode non-blank

FORMAT: modnb(Y)
Y is usually a location the the TRANS table.
C(Y) 9-17 = character to be skipped
modnb( ) is equivalent to modnb(trans+octal (40)) -- blank is 040

FUNCTION: Cause NXCH to skip the character whose code matches C(Y) 9-17
If Y = 0, then NXCH will skip blanks.

EXAMPLE: In this example, NXCH gets the next non-blank character

```
modnb( )
  .
  .
  .
nxch( )
jmp(end1n)
```

## K. STRING MANIPULATION POPS

Pops

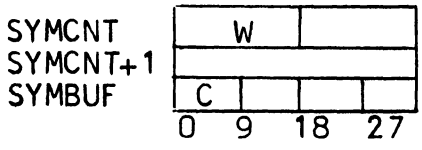
POP: CNTS Count symbol

FORMAT: cnts(Y)

FUNCTION: Assume SYMBUF contains a type-1 string

1. Count the number of characters and the number of words in type-1 string in SYMBUF.
2. Set C(SYMBUF) 0-8 = number of characters in string.
   Set C(SYMCNT) 0-17 = number of words in string.
3. If Y is a non-zero, then set C(Y) 0-17 = number of words in string.
   Otherwise, do not change C(Y) 0-17.

This is illustrated below:




The interpreter would change W and C. If Y ≠ 0, C(Y) 0-17 would be set to W. NOTE: W = C/4 + 1 (ignoring remainder)

POP: RSYM Reset symbol buffer

FORMAT: rsym( )

FUNCTION: Assume SYMBUF contains a type-1 string

1. Count the type-1 string currently in SYMBUF (i.e., count the number of words and the number of characters in SYMBUF).
2. Change the count field of the string to zero.
3. Change the remaining characters in the string to blanks
4. Set C(SYMCNT) = 0
   Set C(SYMCNT+1) = 0

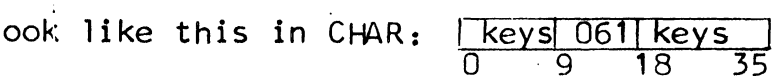
POP: PAK Pack

FORMAT: pak(Y)

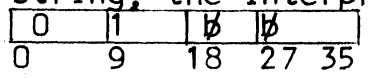
FUNCTION: Insert C(Y) 9-17 immediately to the right of the last character inserted into SYMBUF
C(Y) 9-17 is a 9-bit ASCII character

EXAMPLE: pak(char)

The character 1 would look like this in CHAR:




If SYMBUF contains the null string, the interpreter sets C(SYMBUF) as follows:




POP: PAKR Reset and pack

FORMAT: pakr(Y)

FUNCTION: 1. Execute RSYM pop
2. Execute pak(Y) pop

POP: PAKA Pack and advance

FORMAT: paka(Y)

FUNCTION: 1. Execute pak(Y) pop
2. Execute NXCH pop

EXAMPLE: The following coding causes the interpreter to keep packing characters into SYMBUF, until it receives a blank:

```
        modb( )                 "accept blanks"
        rsym( )                 "reset symbol to blanks"
loop;   nxch( )                 "get next character"
        jmp(endln)              "special case if end of line"
        sch(trans+octal(40))    "octal 40 is blank"
        jmp(blk,t)              "jump if blank"
        paka(char)              "otherwise, pack in symbol,
                                 and advance"
        jmp(endln)              "special case if end of line"
        jmp(loop+2)             "continue"
             .
             .
             .
 blk:   cnts( )                 "count the symbol"
```

COMMENT: On end-of-line, the interpreter goes to the next pop (same as NXCH).

POP: PAKAR Reset, pack, and advance

FORMAT: pakar(Y)

FUNCTION: 1. Execute RSYM pop

2. Execute pak(Y) pop

3. Execute NXCH pop

COMMENT: On end-of-line, the interpreter goes to the next pop (same as NXCH).

POP: PLXP Plex put

FORMAT: plxp(N)

FUNCTION: Given a type-1 string in SYMBUF, put a plex on the bottom of roll N. The string need not have been counted.

1. Count the string in SYMBUF (i.e., count the number of words and the number of characters)
   w = number of words

2. Bump bottom of roll N by w+1 words

3. Move type-1 string from SYMBUF to word 1 -- word w (inclusive)

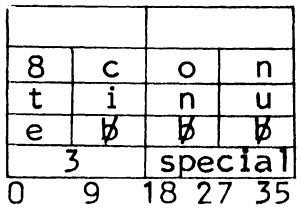
4. Set C(word w+1) 0-17 = w (the lower half of this word is used by the interpreter.)

EXAMPLE:

plxp(5)

Assume the type-1 string "continue" is in SYMBUF.

Roll 5 after




Old bottom

New bottom

POP: PLXG Plex get

FORMAT: plxg(N)

FUNCTION: Given a plex on the bottom of roll N, form a type-1 string in SYMBUF.

1. Recover number of words in string (w words) from the last word of the plex (word w+1) w = C(word w+1) 0-17 (See PLXP)
2. Move word 1 -- word w (inclusive) to SYMBUF -- SYMBUF + w - 1 (inclusive)
3. Set C(SYMCNT) 0-17 = w
4. Set C(SYMCNT) 18-35 = 0
5. Set C(SYMCNT+1) = C(word w+1)
6. If the previous string in SYMBUF had more than w words, then fill these words with ASCII blanks.

EXAMPLE:

plxg(5)

Assume roll 5 contains the plex shown in the example for PLXP.

plxg(5)

SYMBUF after

| | | | | |
|---|---|---|---|---|
| SYMCNT | 3 | | 0 | |
| SYMCNT+1 | 3 | | special | |
| SYMBUF | 8 | c | o | n |
| | t | i | n | u |
| | e | ∅ | ∅ | ∅ |
| | 0 | 9 | 18 | 27 35 |

POP: PLXM Plex make

FORMAT: plxm(N)

FUNCTION: Convert a type-1 string into a plex. Assume the following:

RP(ROLPTR+N) is the location of the VSW of the last group on roll N. This group contains a type-1 string. The bottom of roll N follows the last word of the string
The string has been counted (i.e., the number of words and the number of characters have been counted)

NOTE: The group need not have been counted

Let w = number of words in the string

1. Bump bottom of roll N by 1 word

2. Set C(word 1) 0-17 = w (The lower half of this word is used by the interpreter)

3. Set C(VSW) 0-17 = w+1

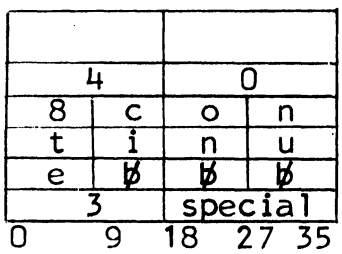
   Set C(VSW) 18-35 = 0

EXAMPLE:

plxm(5)

Assume the last group on roll 5 is the string, "continue".

Roll 5 after



POP: CCAT Concatenate

FORMAT: ccat(N)

FUNCTION: Concatenate a type-1 string in SYMBUF to a type-1 string in roll N.

Assume the following:

RP(ROLPTR+N) is the location of the first word of the second type-1 string

The bottom of roll N immediately follows the last word of the second type-1 string

Both type-1 strings have been counted (i.e., the number of words and the number of characters have been counted)

1. Bump bottom of roll N if the concatenated string requires more words.

2. Add the character count of the first string to the character count of the second string.

3. Concatenate the first string to the second string.

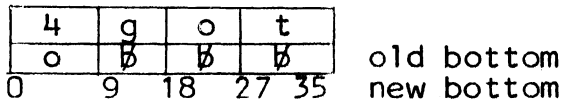
4. Insert trailing blanks into the last word, if necessary.

EXAMPLE:

ccat(5)

Assume SYMBUF contains the string, "to", and RP(ROLPTR+5) is the location of the string, "go".

Resulting string in roll 5:



L. SYNTAX POPS

Pops

POP: FEX Set FEXIT

FORMAT: fex(Y)

FUNCTION: Set C(FEXIT) 0-17 = Y

FEXIT is a one-word register in the data segment. (The interpreter ignores C(FEXIT) 18-35)

POP: PSAV Position save

FORMAT: psav( )

FUNCTION: 1. Bump bottom of roll 2 (save roll) by four words

2. In these four words, record the current status of the input stream.

EXAMPLE: See PRES

POP: PRES Position restore

FORMAT: pres( )

FUNCTION: If C(BOTTOM+2) 0-17 ≠ C(TOP+2) 0-17, go back to last saved position; and remove four words from roll 2.

Otherwise, take no action

EXAMPLE:

Input stream: ABCDEF

| Pop | Character read |
|---|---|
| nxch( ) | A |
| popnop( ) | |
| psav( ) | |
| nxch( ) | B |
| popnop( ) | |
| nxch( ) | C |
| popnop( ) | |
| pres( ) | |
| nxch( ) | B |
| popnop( ) | |

POP: PDES Position destroy

FORMAT: pdes( )

FUNCTION: If C(BOTTOM+2) 0-17 ≠ C(TOP+2) 0-17, prune four words from the bottom of roll 2

M. COUNTING POPS

Pops

POP: ZER Zero

FORMAT: zer(Y)

FUNCTION: Set C(Y) = 0

POP: ZERD Zero double

FORMAT: zerd(Y) Y is an even address

FUNCTION: Set DP(Y) = 0

POP: ONE One

FORMAT: one(Y)

FUNCTION: Set C(Y) 0-17 = 1
Set C(Y) 18-35 = 0

POP: INC Increment

FORMAT: inc(Y)

FUNCTION: Set C(Y) 0-17 = C(Y) 0-17 +1

POP: DCR Decrement

FORMAT: dcr(Y)

FUNCTION: 1. If C(Y) 0-17 = 0, set false
If C(Y) 0-17 ≠ 0, set true

2. If true, set C(Y) 0-17 = C(Y) 0-17 -1

EXAMPLE:

In this example, control passes to loop three times; then control passes to the pop after jmp(loop,t).

```
        ceaw(3)
        stor(alpha)
loop:   pop
         .
         .
         .
        dcr(alpha)
        jmp(loop,t)
```

## N. CONVERSION POPS

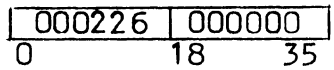
The following number will be used to demonstrate the use of the conversion pops:

1.5e2b17

This number is equivalent to the fraction:

$$\frac{1.5 * 10**2}{2**17}$$

It is represented in octal as follows:




(226 octal = 150 decimal)

The number consists of three parts:

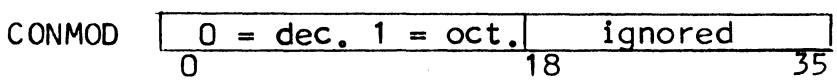
1.5 -- Principal part. If a sign preceded this number, the interpreter would not consider the sign to be included in the principal part.

e2 -- Decimal scale. The decimal scale may be positive or negative.

b17 -- Binary scale. The binary scale may be positive or negative.

### 1. Setting Mode

The interpreter uses the data-segment register CONMOD, to determine whether a number to be converted to binary format is to be treated as a decimal number or an octal number. It has the following format:



Pops That Set CONMOD

POP: MODD Decimal mode

FORMAT: modd( )

FUNCTION: Set C(CONMOD) 0-17 = 0; i.e., cause decimal conversions

POP: MODO Octal mode

FORMAT: modo( )

FUNCTION: Set C(CONMOD) 0-17 = 1; i.e., cause octal conversions

## 2. The Number Buffer

Number conversions occur in the following group of contiguous data-segment registers, the number buffer:

| | | |
|---|---|---|
| CONBUF<br>CONBUF+1 | Principal part (ignoring the decimal point) changed to binary. A two-word signless integer. | |
| FCNT | # of digits to right of dec. pt. | Set to 0 and ignored |
| SIGN | Sign of number. 0 = +<br>non-zero = - | |
| DSCALE | # after e<br>Pos. or neg. | Set to 0 and ignored |
| BSCALE | # after b<br>Pos. or neg. | Set to 0 and ignored |
| DSIGN | Sign of dec. scale<br>0=+, nonzero = - | Set to 0 and ignored |
| BSIGN | Sign of bin. scale<br>0 = +, nonzero = - | 0=no bin. scale #<br>1=bin. scale # |

0 7|8 18 35

The user sets the following registers in the number buffer: FCNT, SIGN, DSIGN, and the upper half of BSIGN. The following coding is recommended for this purpose:

To signify a negative number -- one(sign)

To signify a negative decimal or binary scale -- one(dsign) or one(bsign)

To count each digit to the right of the decimal point -- inc(fcnt)

The interpreter sets all the other registers (including the lower half of BSIGN) as directed by the conversion pops.

EXAMPLE:

The number buffer would be set as follows for the number 1.5e2b17. Assume decimal conversion throughout.

| | | |
|---|---|---|
| CONBUF | 0 | 0 |
| CONBUF+1 | 0 | 15 |
| FCNT | 1 | 0 |
| SIGN | 0 | |
| DSCALE | 2 | 0 |
| BSCALE | 17 | 0 |
| DSIGN | 0 | 0 |
| BSIGN | 0 | 1 |

0 8 18 35

Pops That Set Registers in the Number Buffer

POP: RNUM Reset number

FORMAT: rnum( )

FUNCTION: Set the 8 words in the number buffer to zero

POP: CON Convert principal part

FORMAT: con(Y)
Y is usually CHAR

FUNCTION: Let integer = DP(CONBUF), and digit = C(Y) 14-17. (Digit is a number from 0 through 9.)

Octal conversion - Set integer = 8*integer + digit

Decimal conversion - Set integer = 10*integer + digit

These formulas convert the principal part from left to right, digit by digit.

Following conversion, the entire principal part = DP(CONBUF), ignoring any decimal point or sign.

POP: CONA Convert and advance

FORMAT: cona(Y)
Y is usually CHAR

FUNCTION: 1. Execute con(Y) pop

2. Execute NXCH pop

COMMENT: On end-of-line, the interpreter goes to the next pop (same as NXCH)

POP: CONR Reset and convert

FORMAT: conr(Y)

FUNCTION: 1. Execute RNUM pop

2. Execute con(Y) pop

POP: CONAR Reset, convert, and advance

FORMAT: conar(Y)

FUNCTION: 1. Execute RNUM pop

2. Execute con(Y) pop

3. Execute NXCH pop

COMMENT: On end-of-line, the interpreter goes to the next pop (same as NXCH).

POP: CONBA Convert binary scale, and advance

FORMAT: conba(char)
CHAR is always assumed to be the operand for this pop, regardless of the operand.

FUNCTION: 1. Let integer = C(BSCALE) 0-7, and digit = C(CHAR) 14-17. (Digit is a number from 0 - 9)

Octal conversion - Replace integer by 8* integer ± digit
Decimal conversion - Replaces integer by 10* integer ± digit

NOTE: This pop is normally preceded by a MODD pop.

The formula converts the binary scale from left to right, digit by digit.

Following conversion, the entire signed binary scale = C(BSCALE) 0-7.

2. Set C(BSIGN) 18-35 = 1.

3. Execute NXCH pop

COMMENT: On end-of-line, the interpreter goes to the next pop (same as NXCH).

POP: CONDA Convert decimal scale, and advance

FORMAT: conda(char)
CHAR is always assumed to be the operand for this pop, regardless of the operand

FUNCTION: 1. Let integer = C(DSCALE) 0-17, and digit = C(CHAR) 14-17. (Digit is a number from 0 - 9)

Octal conversion - Replace integer by 8* integer ± digit.

Decimal conversion - Replace integer by 10* integer ± digit.

NOTE: This pop is normally preceded by a MODD pop.

The formula converts the decimal scale from left to right, digit by digit.

Following conversion, the entire signed decimal scale = C(DSCALE) 0-17.

2. Execute NXCH pop.

COMMENT: On end-of-line, the interpreter goes to the next pop (same as NXCH).

## 3. Conversion and Storage of Binary Numbers

The following pops convert binary numbers in the number buffer to fixed-point or floating-point numbers of single or double precision.

### Pops

POP: FXDS Convert to fixed-point, single-precision

FORMAT: fxds(Y)
Y is the location of a single word in the data segment

FUNCTION: Convert the number in the number buffer to a fixed-point, single-precision number, and store the number in Y.

EXAMPLE: See FLTD

POP: FXDD Convert to fixed-point, double-precision

FORMAT: fxdd(Y)
Y and Y+1 are the locations of a pair of words in the data segment. Y is an even location.

FUNCTION: Convert the number in the number buffer to a fixed-point, double-precision number, and store the number in Y and Y+1. The high-order bits are in Y, and the low-order bits are in Y+1.

EXAMPLE: See FLTD

POP: FLTS Convert to floating-point, single-precision

FORMAT: flts(Y)
Y is a location of a single word in the data segment

FUNCTION: Convert the number in the number buffer to a floating-point, single-precision number, and store the number in Y.

EXAMPLE: See FLTD

POP: FLTD Convert to floating-point, double-precision

FORMAT: fltd(Y)
Y and Y+1 are the locations of a pair of words in the data segment
Y is an even location.

FUNCTION: Convert the number in the number buffer to a floating-point, double-precision number, and store the number in Y and Y+1. The high-order bits are in Y, and the low-order bits are in Y+1.

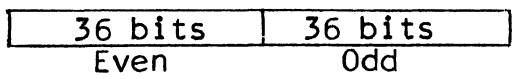
EXAMPLES:

Assume that ALPHA is an even location in the data segment, and that the number buffer is set as shown in the example on page 74.

| Pop | ALPHA after | | ALPHA+1 after | |
|---|---|---|---|---|
| fxds(alpha) | 000226 | 000000 | unchanged | |
| fxdd(alpha) | 000226 | 000000 | 000000 | 000000 |
| flts(alpha) | 020454 | 000000 | unchanged | |
| fltd(alpha) | 020454 | 000000 | 000000 | 000000 |
| | 0 18 35 | | 0 18 35 | |

# 0. PRECISION ARITHMETIC POPS

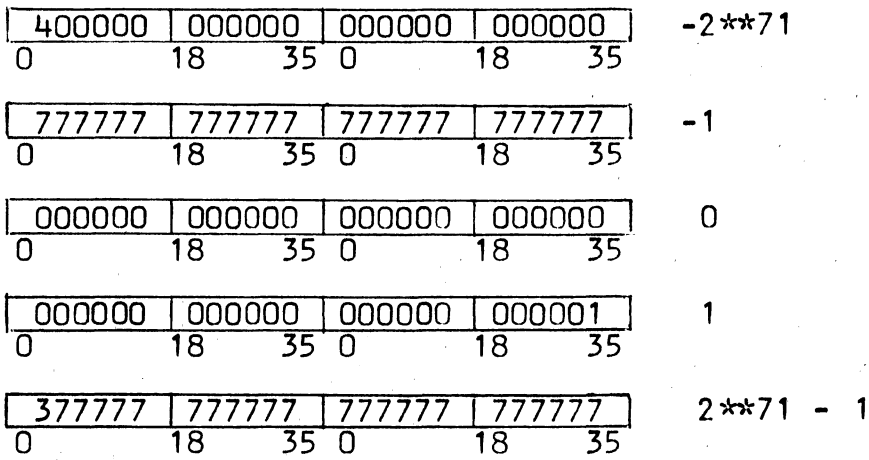
The precision arithmetic pops use a two-word register, located in the data segment:




## 1. Fixed-Point Operations

Double-precision fixed-point numbers are integers ranging from -2**71 through 2**71 - 1.

EXAMPLES:




### Pops

Each of the following pops requires a single-precision operand. However, several of the pops require extension of C(Y). Here, extension means that the interpreter prefixes C(Y) with 36 bits, each of which is a copy of C(Y) 0. (See -1 and 1 in the examples above.)

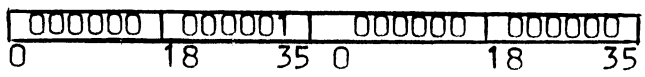
POP: PADD Precision add

FORMAT: padd(Y)

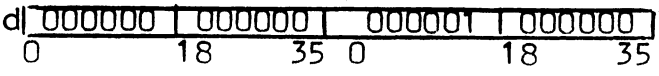
FUNCTION: Set DP(MPAC) = DP(MPAC) + C(Y) where C(Y) is extended to double precision

EXAMPLES:

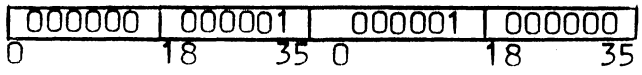
padd(alpha)

MPAC before 

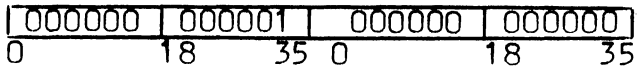
2**36

ALPHA extended 

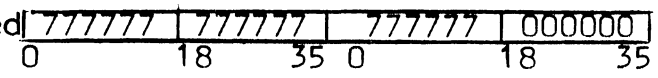
2**18 (extended)

MPAC after 

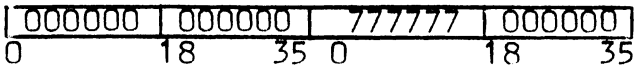

padd(alpha)

MPAC before 

2**36

ALPHA extended 

-2**18 (extended)

MPAC after 

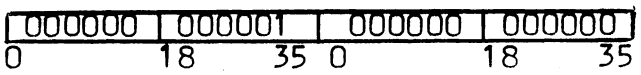

POP: PSUB Precision subtract

FORMAT: psub(Y)

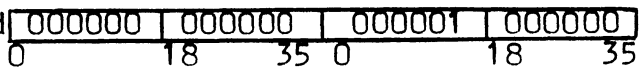
FUNCTION: Set DP(MPAC) = DP(MPAC) - C(Y) where C(Y) is extended to double precision

EXAMPLES:

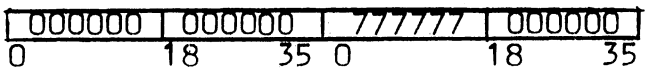
psub(alpha)

MPAC before 

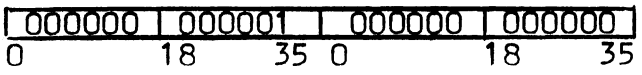
2**36

ALPHA extended 

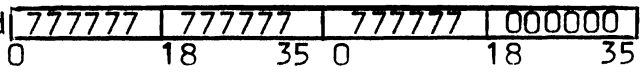
2**18 (extended)

MPAC after 

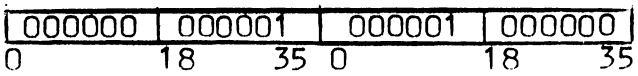

psub(alpha)

MPAC before 

2**36

ALPHA extended 

-2**18 (extended)

MPAC after 

POP: PMLT Precision multiply

FORMAT: pmlt(Y)

FUNCTION: Set DP(MPAC) = C(MPAC+1) * C(Y)

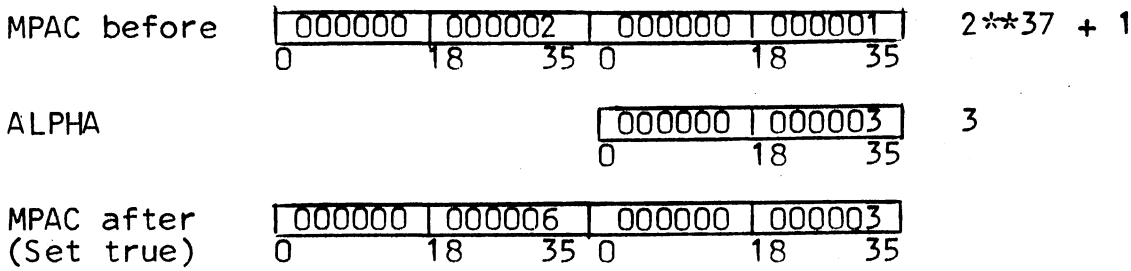
POP: PMLTD Precision multiply double

FORMAT: pmltd(Y)

FUNCTION: 1. Set DP(MPAC) = DP(MPAC) * C(Y)
2. Truncate product to 72 bits, if necessary.
If truncation is necessary, set false
If no truncation is necessary, set true

EXAMPLE:

pmltd(alpha)



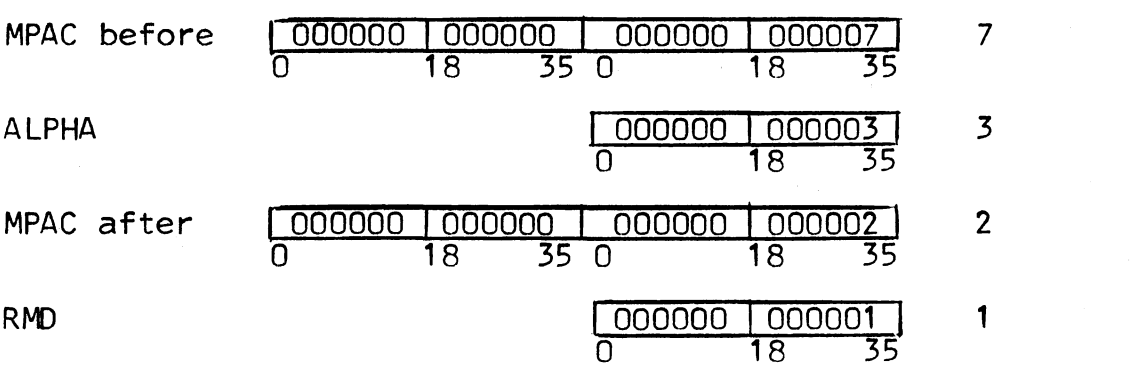

POP: PDVD Precision divide

FORMAT: pdvd(Y)

FUNCTION: 1. Set DP(MPAC) = DP(MPAC) / C(Y)
Extend the quotient to double precision
Note: If the quotient falls outside the range from -2**35 to 2**35 - 1, a divide check error occurs.
2. Set C(RMD) = remainder
RMD is a one-word register in the data segment

EXAMPLE:

pdvd(alpha)



## 2. Floating-Point Operations

Double-precision floating-point numbers consist of an exponent, e, which is an integer ranging from -128 to +127 in steps of 1; and a mantissa, m, which is a fraction ranging from -1 to 1 -2**-63 in steps of 2**-63. (For further information, see the GE-635 Programmers' Reference Manual.) These numbers are subject to the following restrictions:

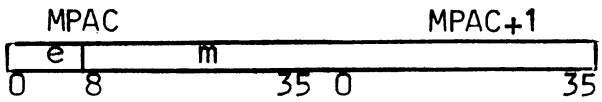
For negative numbers, $-1 \leq m < -1/2$

For zero, m = 0 and e = -128

For positive numbers, $1/2 \leq m < 1$

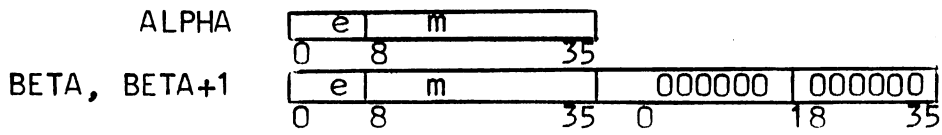
Both m and e are in two's complement form

The floating-point number m * 2**e is represented in the MPAC register as follows:




### Pops

Each of the following pops requires a double-precision operand. The user can extend a single-precision number to double precision, by moving it to an even location and setting the following odd location to zero:




where C(ALPHA) is extended via the following pops:

```
mov(alpha)
to(beta)    "BETA must be an even location"
zer(beta+1)
```

CAUTION: Overflow may occur in each of the following operations!

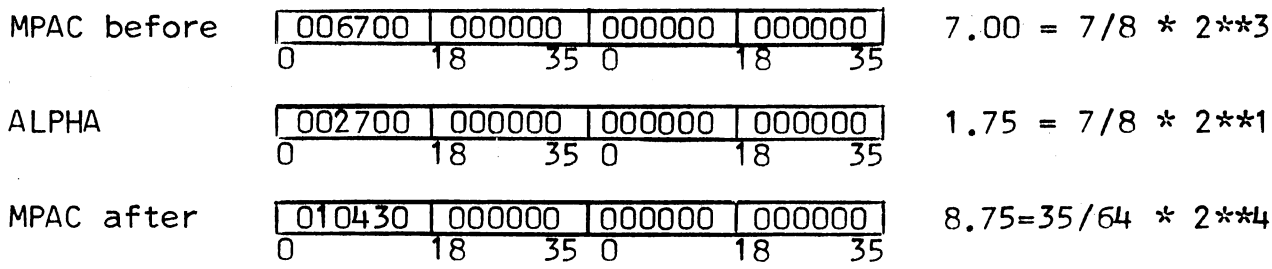
POP: PADDF Precision add floating

FORMAT: paddf(Y)

FUNCTION: Set DP(MPAC) = DP(MPAC) + DP(Y)

EXAMPLE:

paddf(alpha)




POP: PSUBF Precision subtract floating

FORMAT: psubf(Y)

FUNCTION: Set DP(MPAC) = DP(MPAC) - DP(Y)

POP: PMLTF Precision multiply floating

FORMAT: pmltf(Y)

FUNCTION: Set DP(MPAC) = DP(MPAC) * DP(Y)

POP: PDVDF Precision divide floating

FORMAT: pdvdf(Y)

FUNCTION: Set DP(MPAC) = DP(MPAC) / DP(Y)

## 3. Conversion Operations

### Pops

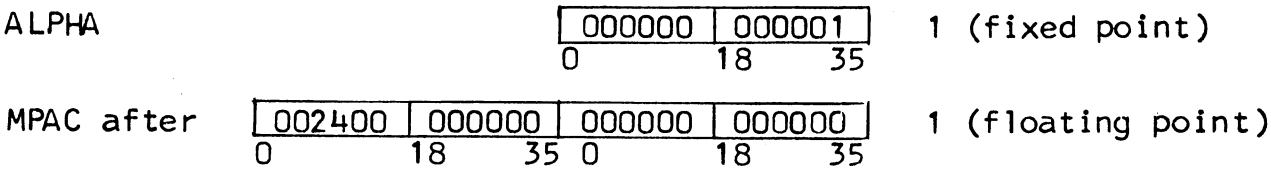
POP: FLT Float

FORMAT: flt(Y)
C(Y) is a single-precision integer

FUNCTION: Set DP(MPAC) = C(Y) converted to a double-precision floating-point number

EXAMPLE:

flt(alpha)




POP: FIXS Fix Single

FORMAT: fixs(Y)
C(Y) is a single-precision floating-point number
Y may be an even or odd address

FUNCTION: Set DP(CONBUF) = C(Y) converted to a double-precision fixed-point integer.

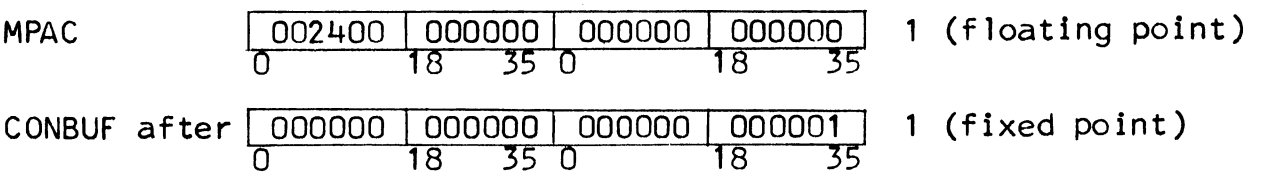
POP: FIXD Fix Double

FORMAT: fixd(Y)
DP(Y) is a double-precision floating-point number
Y must be an even address

FUNCTION: Set DP(CONBUF) = DP(Y) converted to a double-precision fixed-point integer

EXAMPLE:

fixd(mpac)



P. SET POPS

The set pops compare two items to determine whether a certain condition is met. If the condition is met, they set the true indicator. If the condition is not met, they set the false indicator.

1. Algebraic Comparisons

Algebraic comparisons involve 36-bit signed integers. The largest number is octal 377777777777 (2**35 - 1). The smallest number is octal 400000000000 (-2**35).

Pops

POP: SGT Set on greater than

FORMAT: sgt(Y)

FUNCTION: Set true if C(WO) > C(Y) algebraically
Otherwise, set false

POP: SGTP Set on greater than, and prune

FORMAT: sgtp(Y)

FUNCTION: 1. Set true if C(WO) > C(Y) algebraically
Otherwise, set false

2. Prune WO

POP: SEQ Set on equal

FORMAT: seq(Y)

FUNCTION: Set true if C(WO) = C(Y)
Otherwise, set false

POP: SEQP Set on equal, and prune

FORMAT: seqp(Y)

FUNCTION: 1. Set true if C(WO) = C(Y)
Otherwise, set false
2. Prune WO

POP: SLT Set on less than

FORMAT: slt(Y)

FUNCTION: Set true if C(WO) < C(Y) algebraically
Otherwise, set false

POP: SLTP Set on less than, and prune

FORMAT: sltp(Y)

FUNCTION: 1. Set true if C(WO) < C(Y) algebraically
Otherwise, set false

2. Prune WO

2. Masked Comparisons

Pops

POP: SME Set on masked equality

FORMAT: sme(Y)
Y is an even address

FUNCTION: Set true if C(WO) k = C(Y) k, whenever C(Y+1) k = 0
(ignoring all other bits)
Otherwise, set false.

EXAMPLE:

sme(alpha)

| | | |
|---|---|---|
| WO | 112233 | 445566 |
| ALPHA | 716253 | 443526 |
| ALPHA+1 | 707070 | 707070 |
| | 0 18 | 35 |

In this case, set true

POP: SMEP Set on masked equality, and prune

FORMAT: smep(Y)
Y is an even address

FUNCTION: 1. Set true if C(WO) k = C(Y) k, whenever C(Y+1) k = 0
(ignoring all other bits)
Otherwise, set false

2. Prune WO

POP: SME1 Set on masked equality in W1

FORMAT: sme1(Y)
Y is an even address

FUNCTION: Set true if C(W1) k = C(Y) k , whenever C(Y+1) k = 0
(ignoring all other bits)
Otherwise, set false

POP: SME2 Set on masked equality in W2

FORMAT: sme2(Y)
Y is an even address

FUNCTION: Set true if C(W2) k = C(Y) k, whenever C(Y+1) k = 0
(ignoring all other bits)
Otherwise, set false

POP: SMEI Set on masked equality indirect

FORMAT: smei(Y)
Y is an even address

FUNCTION: Set true if C(RP(WO)) k = C(Y) k , whenever
C(Y+1) k = 0 (ignoring all other bits)
Otherwise, set false

POP: SMEIP Set on masked equality indirect, and prune

FORMAT: smeip(Y)
Y is an even address

FUNCTION: 1. Set true if C(RP(WO)) k = C(Y) k , whenever
C(Y+1) k = 0 (ignoring all other bits)
Otherwise, set false
2. Prune WO

POP: SMEB Set on masked equality on bottom
(See ERB.)

FORMAT: smeb(Y)
Y is an even address.

FUNCTION: Set true if C(B) k = C(Y) k , whenever
C(Y+1) k = 0 (ignoring all other bits)

Otherwise, set false

3. Bit Comparisons

Pops

POP: SEV Set on even

FORMAT: sev( )

FUNCTION: Set true if C(WO) 17 = 0
Otherwise, set false

POP: SEVS Set on even in storage

FORMAT: sevs(Y)

FUNCTION: Set true if C(Y) 17 = 0
Otherwise, set false

POP: SCA Set on comparative and

FORMAT: sca(Y)

FUNCTION: Set true if C(WO) .and. C(Y) ≠ 0
Otherwise, set false

EXAMPLE:

The following is a test to determine whether bit 7 or bit 8 of CHAR is 1:

```
ceaw(octal(3000))
sca(char)
jmp(bothoff,f)
```

The CEAW pop puts 1 in bits 7 and 8 of WO, and 0 in each of the other 34 bits.

POP: SCAP Set on comparative and, and prune

FORMAT: scap(Y)

FUNCTION: 1. Set true if C(WO) .and. C(Y) ≠ 0
Otherwise, set false

2. Prune WO

POP: SNZ Set on non-zero

FORMAT: snz( )

FUNCTION: Set true if C(WO) ≠ 0
Otherwise, set false

POP: SNZS Set on non-zero in storage

FORMAT: snzs(Y)

FUNCTION: Set true if C(Y) ≠ 0
Otherwise, set false

POP: SCNT Set on roll count

FORMAT: scnt(N)

FUNCTION: Set true if C(TOP+N) 0-17 ≠ C(BOTTOM+N) 0-17
Otherwise, set false

4. Character Comparisons

Pops

POP: SCH Set on character equal

FORMAT: sch(Y)

FUNCTION: Set true if C(CHAR) 0-17 = C(Y) 0-17
Otherwise, set false

POP: SCHA Set on character equal, and advance

FORMAT: scha(Y)

FUNCTION:
1. Set true if C(CHAR) 0-17 = C(Y) 0-17
Otherwise, set false
2. If true, execute NXCH pop
If false, skip a pop

EXAMPLE: The following two pops would cause a blank to be suppressed:

```
a: scha(trans+octal(40))
   jmp(endln)
```

The following pops would suppress all leading blanks:

```
a: scha(trans+octal(40))
   jmp(endln)
   jmp(a,t)
```

COMMENT: On end-of-line, the interpreter goes to the next pop (same as NXCH)

POP: SCKY Set on character keys

FORMAT: scky(Y)

FUNCTION: Set true if C(CHAR) 18-35 .and. C(Y) 18-35 ≠ 0
Otherwise, set false

POP: SCKYA Set on character keys, and advance

FORMAT: sckya(Y)

FUNCTION: 1. Set true if C(CHAR) 18-35 .and. C(Y) 18-35 ≠ 0
Otherwise, set false
2. If true, execute NXCH pop
If false, skip a pop

EXAMPLE:

sckya(alpha)

Assume the TRANS table entries for comma and semicolon are:

TRANS+ octal 54 | 000054 | 002000 | -- comma
0 9 18 35

TRANS+ octal 73 | 000073 | 010000 | -- semicolon
0 9 18 35

In this example, a 1 in bit 23 signifies a semicolon, and a 1 in bit 25 signifies a comma.

| | C(ALPHA) | Result |
|---|---|---|
| Case 1: | 0 \| 2000 | Set true and execute NXCH pop if , |
| Case 2: | 0 \| 10000 | Set true and execute NXCH pop if ; |
| Case 3: | 0 \| 12000 (0 18 35) | Set true and execute NXCH pop if , or ; |

COMMENT: On end-of-line, the interpreter goes to the next pop (same as NXCH)

5. Null Comparisons

The following pops make no comparisons, but simply set true or false.

Pops

POP: STRU Set true

FORMAT: stru( )

FUNCTION: Set true

POP: SFAL Set false

FORMAT: sfal( )

FUNCTION: Set false

6. Symbol Comparisons

Pops

POP: SSKY Set on symbol key

FORMAT: ssky(Y)

FUNCTION: Set true if C(Y) .and. C(SYMKEY) ≠ 0
Otherwise, set false

COMMENT: The user must be sure that C(Y) 0-17 = 0, since this is a full-word comparison. (See ORKEY.)

POP: SSKYA Set on symbol key, and advance

FORMAT: sskya(Y)

FUNCTION:
1. Set true if C(Y) .and. C(SYMKEY) ≠ 0
   Otherwise, set false
2. If true, execute the pop at location 1 of the procedure segment
   If false, go to the next pop

POP: SSY Set on symbol

FORMAT: ssy(Y)

Y is the location of the first word of a type-1 string

FUNCTION: Assume SYMBUF contains a type-1 string

1. Count the type-1 string in SYMBUF.
2. Compare the string in SYMBUF with the string starting at location Y. This is a word-by-word comparison.
3. If all words match, set true
   Otherwise, set false

POP: SSYA Set on symbol, and advance to next symbol if equal

FORMAT: ssya(Y)

Y is the location of the first word of a type-1 string

FUNCTION: Assume SYMBUF contains a type-1 string

1. Count the type-1 string in SYMBUF.
2. Compare the string in SYMBUF with the string starting at location Y. This is a word-by-word comparison.
3. If all words match, set true
   Otherwise, set false
4. If true, execute the pop at location 1 in the procedure segment
   If false, go to the next pop

COMMENTS: 1. The pop at location 1 is usually a JMP or a JSB

2. ssya(symbol) is equivalent to ssy(symbol)
   exec(1,t)

Q. REQUIRE POPS

The require pops compare two items to determine whether a certain condition is met. If the condition is met, the interpreter goes to the next pop (or advances). If the condition is not met, the interpreter executes the syntax fail routine. The syntax fail routine executes a PRES pop and jumps to the location specified by FEXIT (see FEX).

1. Symbol Comparisons

Pops

POP: RSY Require on symbol

FORMAT: rsy(Y)

FUNCTION: Assume SYMBUF contains a type-1 string

1. Count the type-1 string in SYMBUF
2. Compare the string in SYMBUF with the string starting at location Y. This is a word-by-word comparison.
3. If all words match, go to next pop.

   Otherwise, execute syntax fail routine.

POP: RSYA Require on symbol, and advance.

FORMAT: rsya(Y)

FUNCTION: Assume SYMBUF contains a type-1 string

1. Count the type-1 string in SYMBUF.
2. Compare the string in SYMBUF with the string starting at location Y. This is a word-by-word comparison.
3. If all words match, execute the pop at location 1 of the procedure segment.

   Otherwise, execute syntax fail routine.

POP: RSKY Require on symbol key

FORMAT: rsky(Y)

FUNCTION: Go to next pop if C(Y) .and. C(SYMKEY) ≠ 0

Otherwise, execute syntax fail routine.

POP: RSKYA Require on symbol key, and advance

FORMAT: rskya(Y)

FUNCTION: If C(Y) .and. C(SYMKEY) ≠ 0, execute the pop at location 1 of the procedure segment.

Otherwise, execute syntax fail routine.

2. Character Comparisons

Pops

POP: RCH Require on character equal

FORMAT: rch(Y)

FUNCTION: Go to next pop if C(CHAR) 0-17 = C(Y) 0-17

Otherwise, execute syntax fail routine

POP: RCHA Require on character equal, and advance

FORMAT: rcha(Y)

FUNCTION: Execute NXCH pop if C(CHAR) 0-17 = C(Y) 0-17

Otherwise, execute syntax fail routine

COMMENT: On end-of-line, the interpreter goes to the next pop (same as NXCH)

POP: RCKY Require on character keys

FORMAT: rcky(Y)

FUNCTION: Go to next pop if C(CHAR) 18-35 .and. C(Y) 18-35 ≠ 0

Otherwise, execute syntax fail routine

POP: RCKYA Require on character keys, and advance

FORMAT: rckya(Y)

FUNCTION: Execute NXCH pop if C(CHAR) 18-35 .and. C(Y) 18-35 ≠ 0
Otherwise, execute syntax fail routine.

COMMENT: On end-of-line, the interpreter goes to the next pop (same as NXCH)

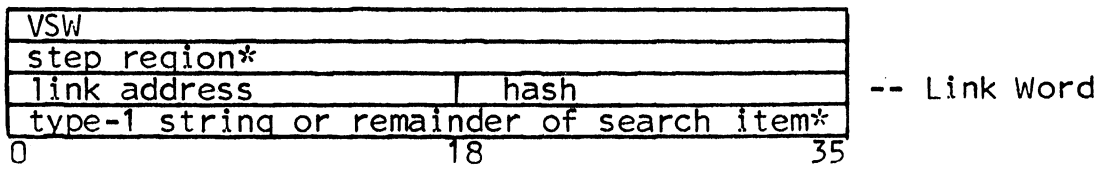
R. SEARCH POPS

1. General

The search pops search a roll for a search item matching a given item called a clue.

Two types of rolls may be searched: linked and non-linked.

A linked roll consists of variable-size groups that are linked on threads via link words. Each link word contains the address of the next logical link word on the thread. A linked group is set up as follows:



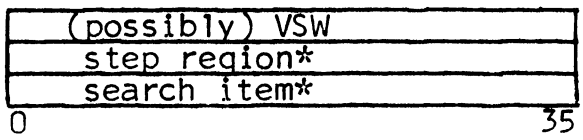

* May be any number of words

The link address is 0 if no more links follow; otherwise, it is the location of the next link word. For roll 0 it is an absolute location; for rolls 1-63 it is relative to the top.

If the clue is a type-1 string, then the interpreter computes the hash; in this case, the hash is not included in the clue or in the search item. If the clue is not a type-1 string, the interpreter uses the right half of the first word of the clue as the hash; in this case, the hash is included in the clue and in the search item.

The step region contains any pertinent information describing the search item.

A non-linked roll may consist of fixed or variable-size groups. It is set up as follows:




* May be any number of words

Roll 0 must be set up as a linked roll. Rolls 1-63 may be linked or non-linked.

2. Types of Threads

Two types of threads are used for linked groups, depending on the roll number and on the type of clue to be matched:

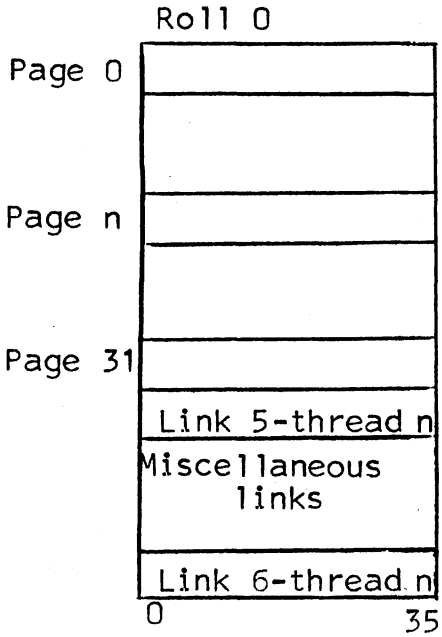
| Thread Type | Clue | Roll Number |
|---|---|---|
| 1 | Type-1 string | 0 |
| 2 | Type-1 string | 1-63 |
| | Anything else | 0-63 |

There are 32 type-1 threads. There may be any number of type-2 threads.

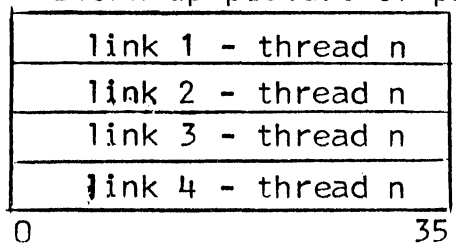
The first 32K of roll 0 (after the first eight words) consists of one page for each type-1 thread. The links in this area are contiguous. If the total requirement for a type-1 thread exceeds 1024 words, the excess words are stored somewhere in the 32K area after page 31.

EXAMPLE:

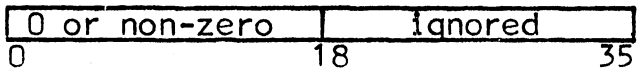
Assume that the first four links for thread n (a type-1 thread) take up 1024 words:




Blown up picture of page n



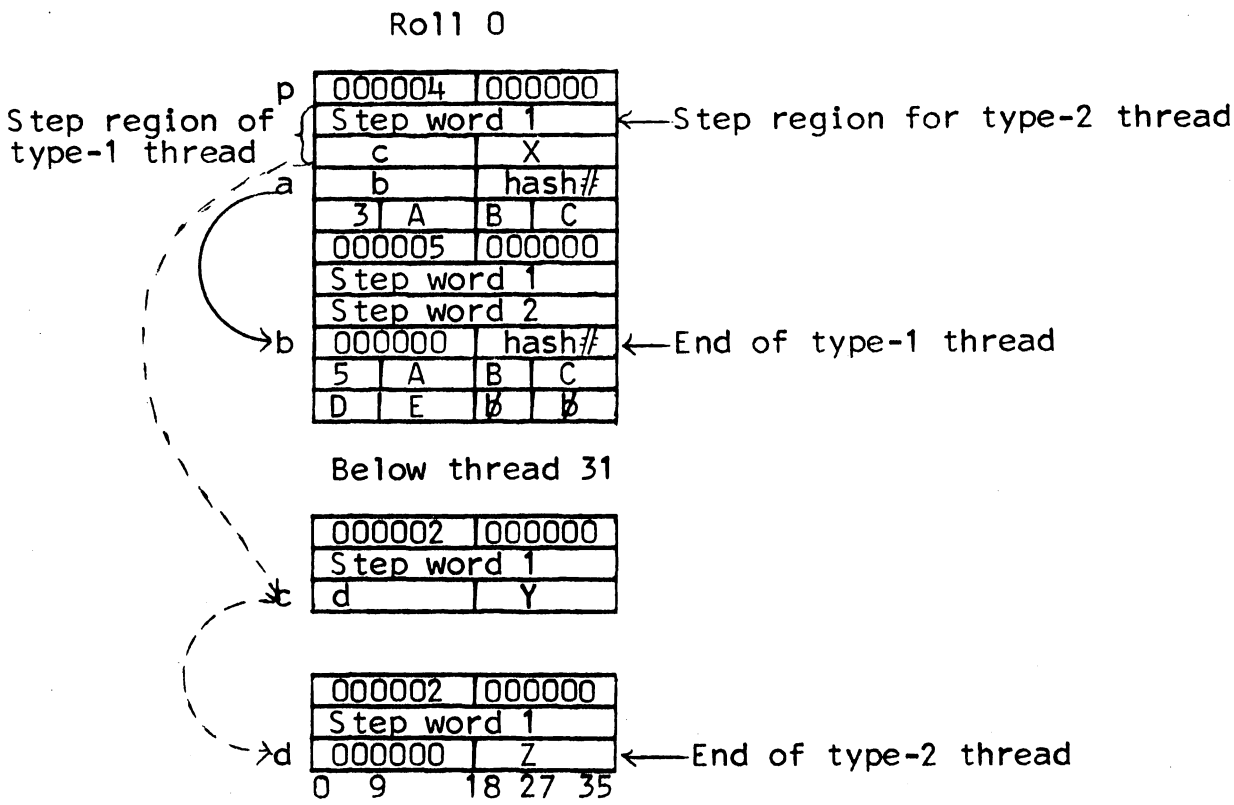

The thread table for roll 0 contains an entry for each type-1 thread:




Here 0 means that the thread is empty. A non-zero value gives the absolute location of the first link word on the thread. The thread table THREAD is located in the data segment.

Links for type-2 threads in roll 0 are placed after page 31 by the interpreter. The user may start such a thread in the first 7-word group on roll 0, provided that the step size is 0-6 and C(ROLPTR) = 0. Otherwise, these type-2 threads must emanate from the step region of a type-1 thread. In each case, all subsequent links are placed after page 31.

EXAMPLE:




NOTE: If the type-1 thread is thread T, C(THREAD+T) 0-17 = a

The type-1 thread is connected by a solid arrow. The type-2 thread is connected by dashed arrows. X, Y and Z are 18-bit constants, most likely offsets on some other roll (pointing to other information).

3. The Search File

The search file for srch(Y), srchp(Y), linkn(Y), and linkp(Y) has the following format:

| | | |
|---|---|---|
| Y | roll number (N) | 0 - not rel.links<br>non-0 - rel.links |
| Y+1 | step size | 0 - non-masked<br>non-zero - masked |
| Y+2 | # of words in clue | ignored |
| Y+3 | Tells whether clue is type-1 string | location of clue |
| Y+4 | mask (if any) -# of bits in mask must match # of bits in clue * | |

0 18 35

* May be any number of words

The items in the search file have the following functions:

C(Y) 0-17 - Number of the roll to be searched

C(Y) 18-35 - In a search file for roll 0, C(Y) 18-35 is ignored

In a search file for another roll containing links, C(Y) 18-35 = non-zero

In a search file for another roll not containing links, C(Y) 18-35 = 0

C(Y+1) 0-17 - Number of words in step region

C(Y+2) 0-17 - Number of words in clue (computed by interpreter for type-1 strings).

C(Y+3) 0 - If 0, the clue is not a type-1 string

If 1, the clue is a type-1 string

Bits 1-17 are ignored.

C(Y+3) 18-35 - A location in the data segment, usually SYMBUF

If this field contains -1, C(MRKER) 0-17 is the location of a roll pointer which points to clue.

If this field contains -2, C(MRKER) 0-17 specifies the location of the clue.

This permits the clue to be in the work stack.

Mask - The mask is used, only if selected portions of the clue are to be compared. In the mask, a 1-bit means ignore and a 0-bit means compare. The number of words in the mask matches the number of words in the clue: if there is no mask, the search file is 4-words; if there is a mask, the search file is 4-words plus the number of words in the clue.

Masks may be used only for non-linked searches.

The following search file is for step size 2 and type-1 strings in roll 0:

| | |
|---|---|
| 0 | 0 |
| 2 | 0 |
| 0 | 0 |
| -1 | SYMBUF |

0 18 35

4. Linked Searches

If there are no links on the roll to be searched, the search is unsuccessful. Otherwise, the interpreter begins a linked search by determining the proper thread to search and the location of the first link on that thread. It uses one of the following methods:

Type-1 thread -- The interpreter computes the thread number and looks in the thread table to obtain the location of the first link word in the thread.

Type-2 thread -- In this case, the user must make ROLPTR+N point to the VSW of the first link on the thread. The interpreter computes the absolute location of the VSW, and adds the number of words in the step region plus one to determine the location of the link word.

Next, the interpreter computes the hash number of the clue. (See Paragraph R.1.)

The interpreter then starts the search by comparing the hash in the first link word with the hash of the clue. If they match, and the clue is not a type-1 string, the search is successful. If they match, and the clue is a type-1 string, the interpreter compares the clue words with the search item words; if they do not match (very rare), it looks for another occurrence of the same hash. If the two hashes do not match, the interpreter uses the link word to determine the address of the next link word. The interpreter continues this process, until it finds a match or reaches the end of the thread.

5. Non-Linked Searches

If there are no groups on the roll to be searched, then the search is unsuccessful. Otherwise, the interpreter assumes that RP(ROLPTR+N) = location of the first word (or VSW) of a group; it locates the first item according to one of the following equations:

For fixed-size groups -- Search item location = RP(ROLPTR+N) + step size

For variable-size groups -- Search item location = RP(ROLPTR+N) + step size +1

Next, the interpreter compares the search item with the clue. If the comparison is unsuccessful, the interpreter simulates the pop dng(N) to find the first word of the next group.

The interpreter continues its search, until it is successful, or until it reaches the bottom of the roll.

Pops

POP: SRCH Search

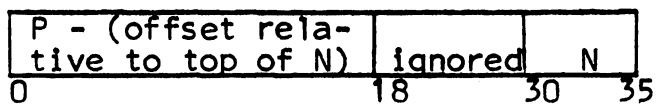
FORMAT: srch(Y)

Y is the location of a search file (See Paragraph R.3.)

FUNCTION: 1. Perform the search as directed by the search file

2. If successful, make CURPTR and ROLPTR+N point to the location of the first word of the group containing the clue, and set true.

CURPTR is a one-word register in the data segment with the following format:




CURPTR is a roll pointer; it points to P + C(TOP+N) 0-17

If unsuccessful, set C(ROLPTR+N) = 0, and set false

EXAMPLE:

In this example, the user scans for a left parenthesis and then searches for a symbol.

```
        modnb( )                "ignore blanks"
        rsym( )                 "reset symbol to blanks"
   loop: nxch( )                "get next character"
        jmp(endln)              "special case if end of line"
        sch(trans+octal(50))    "octal 50 is left parenthesis"
        jmp(loctr+3,t)          "jump if left parenthesis"
        pak(char)               "otherwise, pack in symbol"
        jmp(loop)               "...and continue"
        cnts( )                 "count the symbol"
        srch(sfile)             "search for it"
        jmp(absent,f)           "jump if unsuccessful"
```

POP: SRCHP Search Put

FORMAT: srchp(Y)

Y is the location of a search file (See Paragraph R.3.)

FUNCTION:

1. Perform the search as directed by the search file
2. If successful, perform actions indicated for SRCH pop. If unsuccessful, perform steps 3-10.
3. Set false
4. Determine number of words to create (n words)
5. Allocate space for group, as follows:

   Roll 0, type-1 thread - If there is room on the page corresponding to the thread, allocate space at the bottom of the page

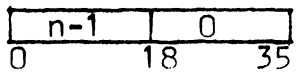
   If there is no room on the page, allocate space on the bottom of the roll

NOTE: The first time the interpreter determines that there is no space available on a page, it closes the page; even if a subsequent link will fit on the page, it is not put there.

Roll 0, type-2 thread -- Allocate next available space on the bottom of the roll

Roll 1-63 -- Allocate next available space at the bottom of the roll

6. If this is a variable size group, create VSW, as follows:




7. Fill the step region with zeros.

8. Create link word, if necessary. (The link address of this word is 0.)

   Set the link address of the previous link word (if any) in the thread to the absolute or relative location of the newly created link word.

9. Copy the clue.

10. Make CURPTR and ROLPTR+N point to the location of the first word of the newly created group.

EXAMPLES:

Assume that the user wants to form a type-2 thread on roll 6. This is the only thread on roll 6. To accomplish this, he writes the following code:

```
pru(6)
srchp(file1) "The search will fail, and the clue
              will be put on roll 6."
```

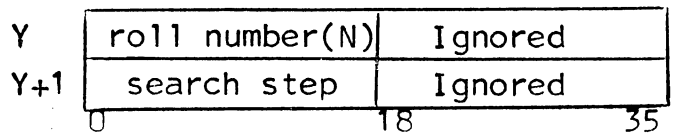
To put a new link on the same thread, the user writes:

```
zer(rolptr+6)
srchp(file1)
```

POP: LINKN Link next

FORMAT: linkn(Y)

Y is the location of a search file (see Paragraph R.3.). Here, only the first two words are applicable:



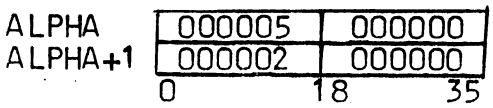

FUNCTION: Assume RP(ROLPTR+N) is the location of the VSW of a linked group on roll N.

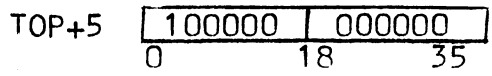
If C(link word) 0-17 = 0, set false

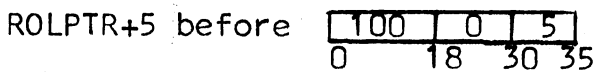
Otherwise, make ROLPTR+N point to the VSW of the next link in the thread; and set true.

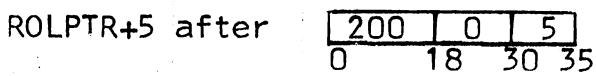
EXAMPLE:

linkn(alpha)



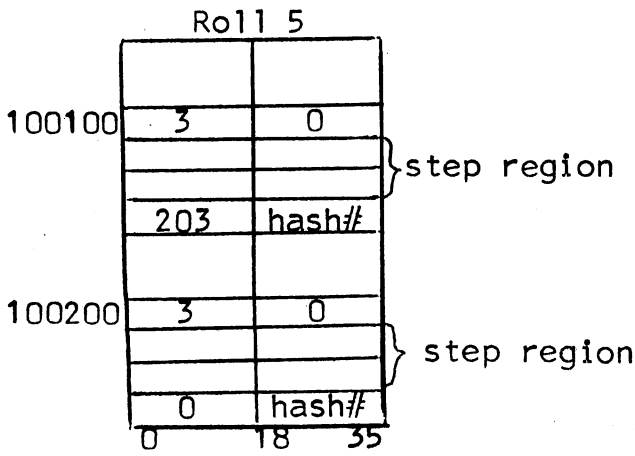




The true condition would be set.

Illustration



Comments: 1. This pop can use a search file set up for a search pop. In this case, it simply ignores all information except C(Y) 0-17 and C(Y+1) 0-17.

2. The interpreter gets N from the search file. It assumes C(ROLPTR+N) 30-35 = N.

POP: LINKP Link put

FORMAT: linkp(Y)

Y is the location of a search file (see Paragraph R.3.).

FUNCTION: Assume RP(ROLPTR+N) is the location of the VSW of the last link on roll N. Also assume that the first word of the clue is special: the left half is ignored, and the right half is to be used as hash

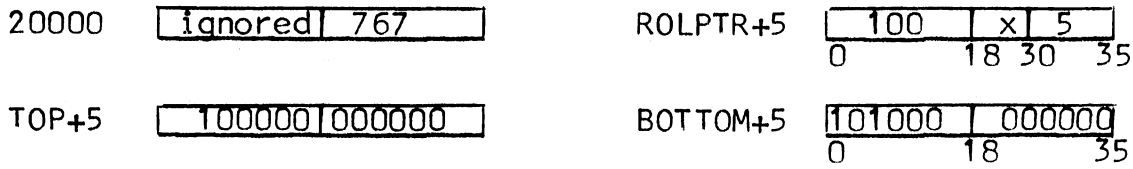
Place a link on the bottom of roll N, according to the specifications of the search file:

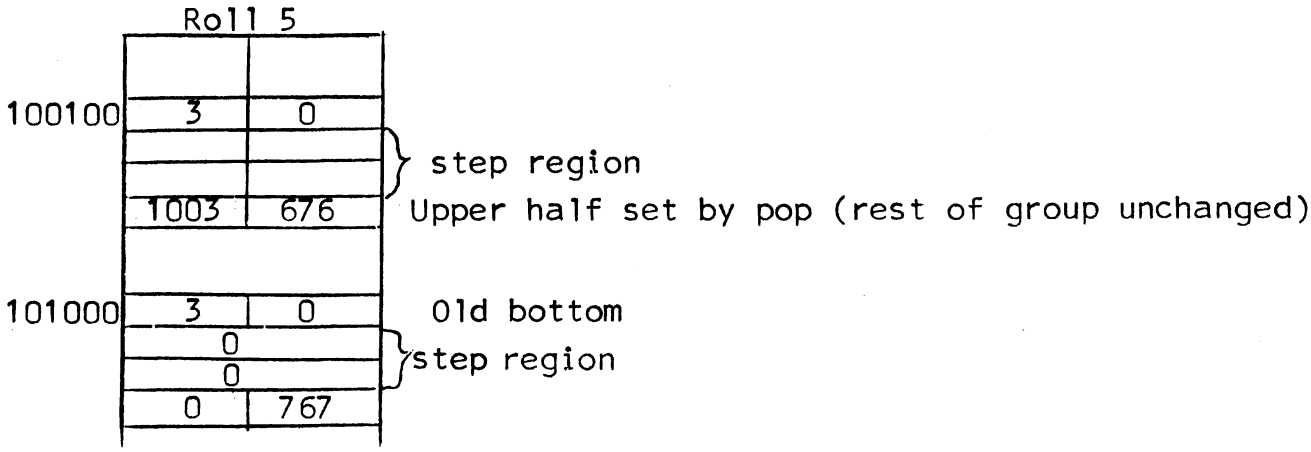
1. Determine number of words to create
2. Allocate space for link
3. Create VSW
4. Fill the step region with zeros
5. Create link word: zero in left half, followed by right half of first word of clue
6. Make the link word in the previous link point to the newly created link word.
7. Copy the remainder of the clue (if any)
8. Make CURPTR and ROLPTR+N point to the location of the first word of the newly created link

EXAMPLE:

linkp (alpha)

| | | |
|---|---|---|
| ALPHA | 5 | 1 |
| ALPHA+1 | 2 | 0 |
| ALPHA+2 | 1 | 0 |
| ALPHA+3 | 0 | 20000 |
| | 0 | 18 35 |



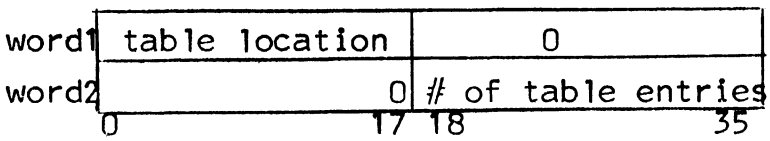

POP: TSRCH Table search

FORMAT: tsrch(Y)

If Y ≠ 0, then Y is the even location of a 2-word search file

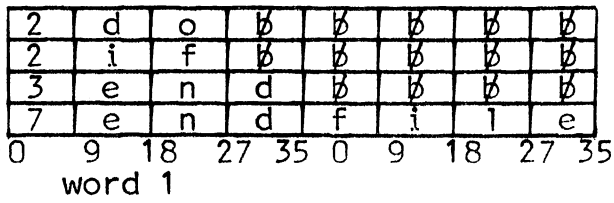
Otherwise, FTAB, a two-word register in the data segment, is the location of the search file

Search File

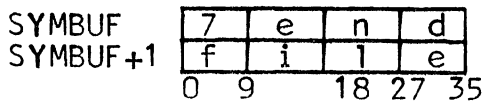



C(word 1) 0-17 -- Location of table consisting of 4-word entries. The first two words of each entry are the key, and the keys are in ascending logical order.

The following illustration shows how the keys are sorted:




C(SYMBUF,SYMBUF+1) = clue. A typical clue is shown below:




FUNCTION:

1. Search the table for a key that matches the clue

2. If there is no match, set false

   Otherwise, set true, and C(MRKER) 0-17 = location immediately following the matching key; i.e., word 3 of the entry satisfying the test.

EXAMPLE:

In this example, if the search is successful, the interpreter will load words 3 and 4 of the entry satisfying the test.

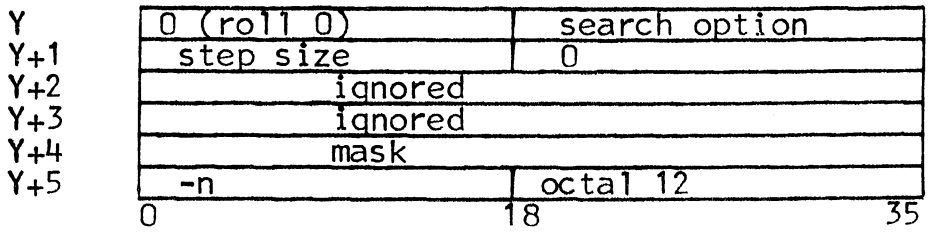
```
cnts( )

tsrch( )

jmp(absent,f)

xntv(mrker)

load(1)          "load word 4"

xntv(mrker)

load(0)          "load word 3"
```

POP: SRCHK Search keys

FORMAT: srchk(Y)

Y is the location of a 6-word search file

Search File




C(Y+5) 0-17 -- Offset relative to link word in any link on roll 0. This is the offset of a selected word in the step region. Therefore, n must be $\leq$ step size.

C(Y+4) -- 36-bit mask. The 1's in this mask define bits to be tested in the selected word. These bits are called the keys.

C(Y) 18-35 -- 0 if a test is to be made for all keys off

1 if a test is to be made for any keys on

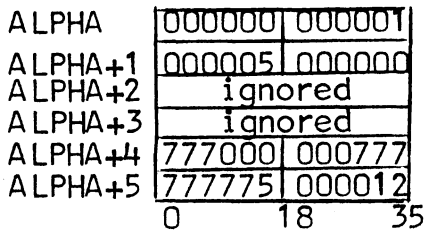
FUNCTION: 1. Test keys in each link on roll 0, according to the search option, until successful or until there are no more links.

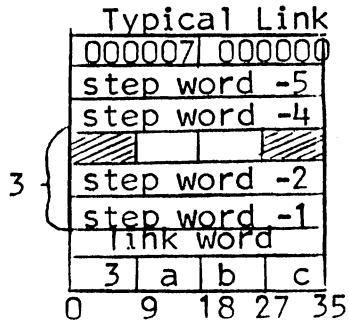
2. If successful, set true; and make ROLPTR+0 point to the VSW of the link satisfying the test.

If unsuccessful, set false; and set C(ROLPTR+0) = 0.

EXAMPLE:

srchk(alpha)




NOTE: 777775 is -3

Here, the interpreter tests each link for any 1 bits in the shaded areas in step word -3.

COMMENT: The interpreter performs the search as follows: It first searches thread 31, then thread 30, then thread 29, etc. However, it performs a forward search on each thread.

POP: SRCHKC Search keys continued

FORMAT: srchkc(Y)

Y is the location of the 6-word search file used by the last executed SRCHK or SRCHKC pop. (See SRCHK for format.)

FUNCTION:

1. Determine whether the last executed SRCHK or SRCHKC pop was successful

2. If the pop was not successful, then set false and set C(ROLPTR+0) = 0.

   If the pop was successful, execute SRCHK pop with the following modification: RP(ROLPTR+0) is pointing to the VSW of the link satisfying the test. If this link is the last on thread N, then start search at the first link of thread N-1. If this link is not the last link on thread N, then start search at the next link of thread N.

S. SHIFT POPS

Pops

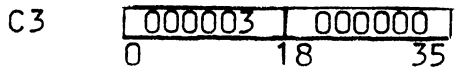
POP: WRKL Shift work left

FORMAT: wrkl(Y)

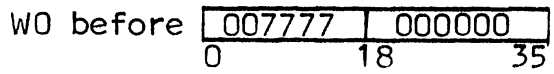
FUNCTION: Shift work left by C(Y) 0-17 bit positions. Fill with zeros on the right of W0.

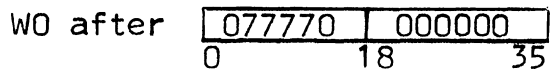
EXAMPLES:

wrkl (C3)

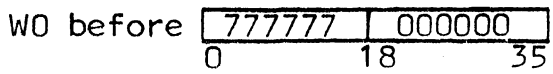


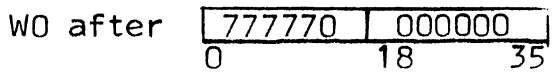

Case 1




Case 2




COMMENT: WRKL is equivalent to multiplication by a power of 2.

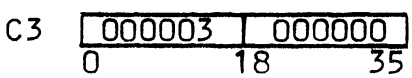
POP: WRKR Shift work right

FORMAT: wrkr(Y)

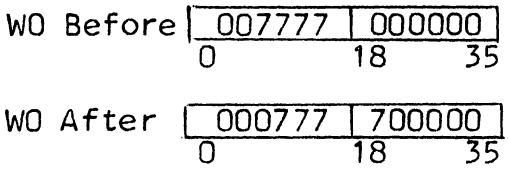
FUNCTION: Shift work right by C(Y) 0-17 bit positions. Fill with zeros on the left of W0.

EXAMPLES:

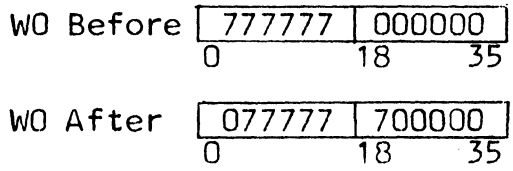
wrkr (C3)



Case 1




Case 2




COMMENT: WRKR is a logical shift, not an arithmetic shift. It should not be used to divide a negative number by a power of 2.

T. EXCHANGE POPS

Pops

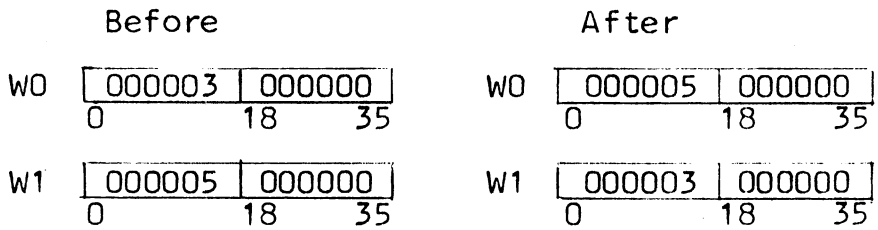
POP: XCH Exchange work and storage

FORMAT: xch(Y)

FUNCTION: Exchange C(W0) and C(Y)

EXAMPLE:

xch(w1)




POP: XLR Exchange left and right halves of work

FORMAT: xlr( )

FUNCTION: Exchange C(W0) 0-17 and C(W0) 18-35

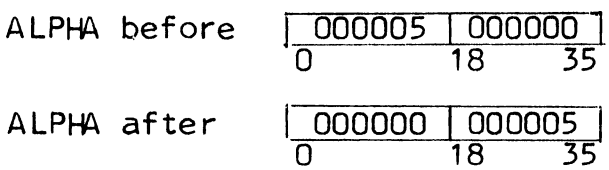
POP: XLRS Exchange left and right halves of storage

FORMAT: xlrs(Y)

FUNCTION: Exchange C(Y) 0-17 and C(Y) 18-35

EXAMPLE:

xlrs(alpha)




COMMENT: xlr( ) is equivalent to xlrs(w0)

U. DUMMY POPS

Pops

POP: DMY Load Dummy

FORMAT: dmy(Y)

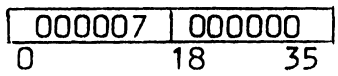
FUNCTION: 1. Add 1 to dummy counter

2. Set C(DO) 0-17 = Y

Set C(DO) 18-35 = 0

EXAMPLE:

dmy(7) appends the following word to the dummy stack:




POP: PRD Prune Dummy

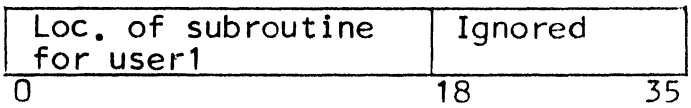
FORMAT: prd(Y)

FUNCTION: Subtract C(Y) 0-17 from dummy counter;

i.e., prune dummy stack by C(Y) 0-17 words

V. USER POPS

There are five user pops. Each corresponds to a GE-645 machine language subroutine in the procedure segment. The starting locations of these subroutines are stored in the upper halves of locations 3-7 in the procedure segment. For example:




Each of these starting locations is somewhere after the reserved locations (octal 0-12).

In writing the subroutines constituting the user pops, the user must observe the following conventions:

1. The base pairs are all reserved:

| Used by Pops Interpreter | Used by Multics |
|---|---|
| AP - Procedure segment | SP - Stack segment |
| BP - Data segment | LP - Linkage segment |

2. The following registers are reserved:

X1 - Pop counter - Relative to AP
X3 - Work
X4 - Exit } - Relative to BP
X5 - Dummy

3. The other registers may be changed.

4. The user routines return to the interpreter in the manner described for DOML (see DOML).

Pops

POP: USER1, USER2, USER3, USER4, and USER5

FORMAT: userN(Y)

N = 1,2,3,4, or 5
Y is a location in the data segment

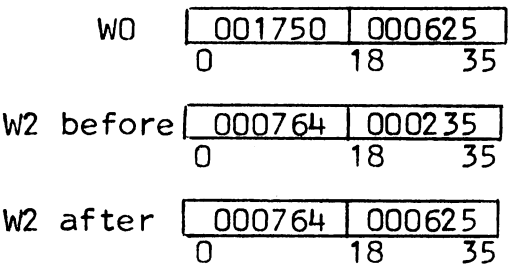
FUNCTION: Execute the corresponding user routine

EXAMPLE:

user3(w2)

Assume user3 is a store lower pop: The operand is Y, a location in the data segment.

The routine sets C(Y) 18-35 = C(W0) 18-35.




COMMENTS:

1. True/false tags and address modification pops may be used with the user pops; i.e., the pop user4(w2,f) is permitted.

2. The user pops are similar to DOML; however, they are more versatile, since the operand of a user pop may specify data. The operand of the DOML pop simply specifies the location of the machine language subroutine. (See DOML.)

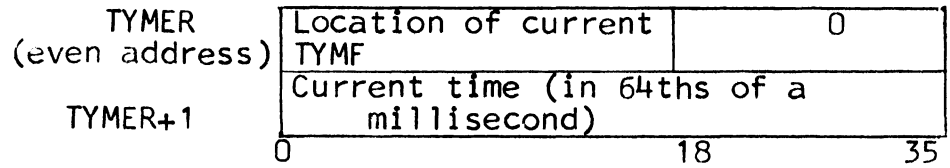
W. TIMING POPS

Pops

POP: TYMF Time from

FORMAT: tymf(N)
N is normally roll 3 (fact roll)

FUNCTION: Record time by setting TYMER and TYMER+1. These are contiguous one-word registers in the data segment, with the following formats:

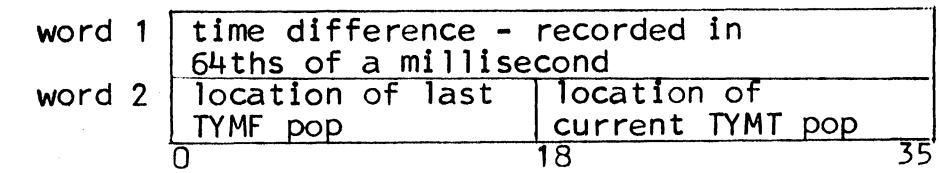



POP: TYMT Time to

FORMAT: tymt(N)
N is normally roll 3

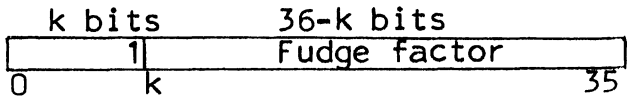
FUNCTION: 1. Compute elapsed time between last time recorded (TYMF) and current time (TYMT)

2. Bump bottom of roll N by two words, and put the following information in these words:




Time difference = current time - C(TYMER+1) - C(TYMASK)

TYMASK is a 36-bit cell in the data segment, with the following format:




The k-bit field may be used to set a flag in word 1, indicating that timing information follows. If the k-bit field contains 1, then the first k-bits of word 1 are set to 1's. The user should choose k small enough, so that 36-k bits are sufficient for recording the time difference. If k=0, then there will be no flag in word 1.

The fudge factor is set equal to the time spent in executing the TYMF and TYMT pops, so that this may be omitted from a critical time calculation. This time may be determined by executing a pair of successive TYMF and TYMT pops and using the elapsed time. The current estimate is 25 (milliseconds/64).

General Comments

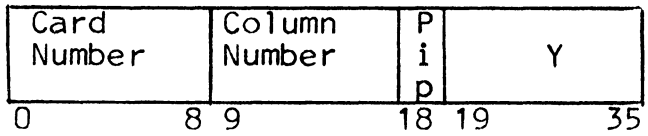
1. Any number of pops may appear between a TYMF pop and a TYMT pop.

2. If there are several TYMF pops, each TYMF overrides the preceding TYMF.

CAUTION: The current implementation of the interpreter does not include a Multics timer interface. Therefore, until further notice, all times will be 0. The timer feature was originally provided for a GECOS environment, where the timings would be more meaningful than in a Multics environment.

X. ERROR POPS

Each of the error pops bumps the bottom of roll 1 (error roll) by one word and sets C(word 1) as follows:




Card number -- C(CRDNUM) 18-26.

Column number -- The number of the column in which the error occurred. This is either C(WO) 0-17 or C(CHARC) 0-17, depending on the pop. The first 9 bits in this number must be zero.

Pip -- A character printed as part of the error list (see Printing the Error List, paragraph Y.1.c). If C(word 1) 18 = 0, the pip is a ^. This appears under the erroneous character. If C(word 1) 18 = 1, the pop is a `. This appears under a character after the erroneous character.

Y -- The operand of the pop. This is the location of the first word of the error message. This location must be in the lower half of the data segment; i.e., Y must be < 2**17 The error message is a type-1 string

EXAMPLE:

| | | | | |
|---|---|---|---|---|
| Y | 17 | I | L | L |
| Y+1 | E | G | A | L |
| Y+2 | ø | C | H | A |
| Y+3 | R | A | C | T |
| Y+4 | E | R | ø | ø |

0 9 18 27 35

Pops

POP: EROR Error on work

FORMAT: eror(Y)
Y is the location of the error message. (See description above).

FUNCTION: 1. Bump bottom of roll 1 by one word

2. Set C(word 1) as follows:

C(word 1) 0-8 = C(CRDNUM) 18-26
C(word 1) 9-17 = C(WO) 9-17. (Here, C(WO) 0-8 must be zero.)

C(word 1) 18 = 0
C(word 1) 19-35 = Y

POP: ERRP Error on work, and prune

FORMAT: errp(Y)
Y is the location of the error message.
(See description above.)

FUNCTION: 1. Execute eror(Y)

2. Prune WO

POP: ERRCC Error on current column

FORMAT: errcc(Y)
Y is the location of the error message.
(See description above.)

FUNCTION: 1. Bump bottom of roll 1 by one word

2. Set C(word 1) as follows:

C(word 1) 0-8 = C(CRDNUM) 18-26
C(word 1) 9-17 = C(CHARC) 9-17. (Here, C(CHARC) 0-8 must be zero.)
C(word 1) 18 = 0
C(word 1) 19-35 = Y

COMMENT: errcc(Y) is equivalent to load(charc)
errp(Y)

POP: ERRLC Error on last column

FORMAT: errlc(Y)
Y is the location of the error message.
(See description above.)

FUNCTION: 1. Bump bottom of roll 1 by one word

2. Set C(word 1) as follows:

C(word 1) 0-8 = C(CRDNUM) 18-26
C(word 1) 9-17 = C(CHARC) 9-17. (Here, C(CHARC) 0-8 must be zero.)
C(word 1) 18 = 1
C(word 1) 19-35 = Y

# Y. OUTPUT POPS

## 1. Print Pops

The print pops (PRNT and PRNTC) prepare data for printing. They specify the format of the data and the order in which it is to be printed.

Each print line may contain up to 132 ASCII characters. The characters constituting the print line are first placed in PLINE, a 37-word buffer in the data segment. PLINE has the following format:

```
PLINE+0  -- Print positions 1-4
PLINE+1  -- Print positions 5-8
PLINE+2  -- Print positions 9-12
          .
          .
          .
PLINE+32 -- Print positions 129-132
PLINE+33
PLINE+34    Extra words, which may be needed for control
PLINE+35              characters.
PLINE+36
```

(The locations PLINE-3, PLINE-2, and PLINE-1 provide a backstop in case any integer extends to the left of print position 1.)

When a line is terminated, the interpreter moves the line to the list segment or the error segment.

### a. The Parameter File

The operand of a print pop is the location of the parameter file in the data segment.* Six types of words may be used in this file:

* This is true if Y ≠ 0. (See Printing the Error List, paragraph Y.1.c.)

ATH - ASCII to ASCII conversion

HTH - Hollerith to ASCII conversion

FID, LID, RID, FIO, LIO, and RIO - Numeric to ASCII conversion

Skip - Skip a certain number of words in the parameter file

Leave - Go to the next pop, without terminating the print line

PRT - Terminate the print line, and go to the next pop.

Each file word consists of five fields:

Field 1 (bits 0-2) - Code - This field identifies the type of file word, as follows:

0 - SKIP
1 - ATH
3 - HTH
4 - FID, LID, RID, FIO, LIO, and RIO
5 - PRT
2,6, and 7 - not used

The code field is not applicable to the leave file word.

Field 2 (bits 3-7) - Word Count (W) - This field tells the number of words to be moved to PLINE (ATH, HTH) or the type of integer conversion (FID, LID, RID, FIO, LIO, RIO). Otherwise, it is ignored.

Field 3 (bits 8-14) - Print Position (P) - The meaning of this field depends on the type of file word.

Field 4 (bits 15-17) and Field 5 (bits 18-35) - Index (I) and Address (A) - These fields give an effective address, which is computed as follows:

| Index | Effective Address |
|---|---|
| I = 0 | A |
| I = 1 | A+C(X1) |
| I = 2 | A+C(X2) |
| I = 3 | A+C(X3) |
| I = 4 | A+C(OUTAG4) 0-17 |
| I = 5 | A+C(OUTAG5) 0-17 |
| I = 6 | A+C(OUTAG6) 0-17 |
| I = 7 | (See LEAVE, page 126) |

Each index corresponds to a GE-645 index register. Therefore, this feature can be used only by the interpreter.

OUTAG4, OUTAG5, and OUTAG6 are cells in the data segment.

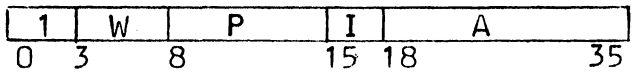
The meaning of the effective address depends on the type of file word.

b. Description of File Words

1. ATH - ASCII to ASCII

FORMAT: ath(A,I,P,W)

REPRESENTATION IN DATA SEGMENT:




MEANING OF OPERANDS:

A and I -- Effective address is location of first word in string

P -- Print position for first ASCII character (1-127) ASCII characters are moved into PLINE left to right

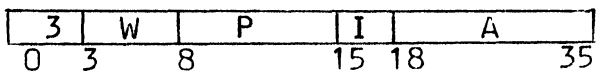
W -- Word count. If W = 0, a type-1 string is to be moved to PLINE
If W = 1,2,..., or 31, 4W characters are to be moved to PLINE

NOTE: The first character in a type-1 string tells how many characters are to be moved to PLINE, but this character itself is never moved.

2. HTH - Hollerith to ASCII

FORMAT: hth(A,I,P,W)

REPRESENTATION IN DATA SEGMENT:



MEANING OF OPERANDS:

A, I, and P -- Same as for ATH

The ASCII equivalents of the Hollerith characters are moved into PLINE left to right.

W -- Word count. If W = 0, a type-1 Hollerith string is to be moved to PLINE

If W = 1,2,..., or 31, 6W characters are to be moved to PLINE.

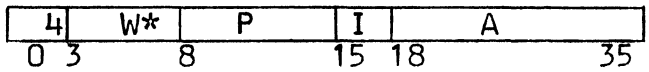
COMMENT: The difference between an ASCII type-1 string and a Hollerith type-1 string is illustrated below:

| ASCII | 3 | A | B | C | Hollerith | 5 | A | B | C | D | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 003 | 101 | 102 | 103 | | 05 | 21 | 22 | 23 | 24 | 25 |

3. Integer to ASCII (macros shown below)

FORMAT: macro(A,I,P)

REPRESENTATION IN DATA SEGMENT:




*The macro determines W. (See MEANING OF OPERANDS.)

MEANING OF OPERANDS:

A and I -- Effective address is location of integer

P -- Print position for least significant digit (1-127) Numeric data is moved into PLINE right to left.

W -- Conversion type - The value of W for each of the macros is shown in the chart below:

| Macro | Meaning | Value of W | Number of characters |
|---|---|---|---|
| FID | Full integer decimal | 0 | Number of significant digits (Plus one, if the sign is negative) |
| LID | Left integer decimal | 1 | |
| RID | Right integer decimal | 2 | |
| FIO | Full integer octal | 8 | 12 |
| LIO | Left integer octal | 9 | 6 |
| RIO | Right integer octal | 10 | 6 |

EXAMPLE:

C(ALPHA) =

| 777777 | 000001 |
|---|---|
| 0 | 18 35 |

Print Positions

| | |
|---|---|
| -262143 | fid(alpha,0,20) |
| -1 | lid(alpha,0,20) |
| 1 | rid(alpha,0,20) |
| 777777000001 | fio(alpha,0,20) |
| 777777 | lio(alpha,0,20) |
| 000001 | rio(alpha,0,20) |
| 9 20 | |

COMMENTS: 1. For negative decimal numbers, the leftmost character is the sign. Otherwise, the leftmost character is the most significant digit.

2. Leading zeros are printed with octal numbers. No leading zeros are printed with decimal numbers.

3. Zero is represented as shown below:

```
FID, LID, RID              0
     LIO, RIO         000000
          FIO 000000000000
```

4) LEAVE - Leave

FORMAT: leave( )

REPRESENTATION IN DATA SEGMENT

| 7 7 7 7 7 | 7 | 7 7 7 7 7 7 |
|---|---|---|
| 0 | 15 18 | 35 |

The macro determines the entire word

The 7 in bits 15-17 indicates that the file word is LEAVE
All other bits are ignored

5) SKIP - Skip

FORMAT: skip(A,I) or skip(A)

REPRESENTATION IN DATA SEGMENT:

| 0 | 0 | 0 | I | A |
|---|---|---|---|---|
| 0 3 | 8 | 15 | 18 | 35 |

MEANING OF OPERANDS:

A and I -- Effective address is location of next parameter word relative to current parameter word.

COMMENT:

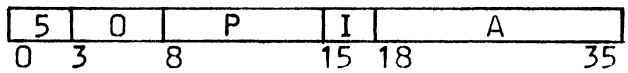
The user may specify a backward skip, but may not specify a skip of 0 words:

skip(0) is illegal (terminates line, with error message)
skip(1) goes to next word, as usual
skip(2) skips one word
skip(-3) goes back three words

6) PRT - Print

FORMAT: prt(P,I,A)

REPRESENTATION IN DATA SEGMENT:




MEANING OF OPERANDS:

A and I -- An effective address of 1-15 decimal specifies the number of lines to space after printing: 1 means single space, 2 means double space, etc.

An effective address of 20 decimal specifies that a new page should begin after the line is printed.

P -- The report code:

2, if the line should be included in the list segment

Any other number, if the line should be included in the error segment

Pops

POP: PRNT Print

FORMAT: prnt(Y)
If Y ≠ 0, it is the first word of the parameter file (See Printing the Error List, paragraph Y.I.c, for a description of what happens when Y = 0.)

FUNCTION: 1. Clear PLINE -- PLINE+36 to ASCII blanks

2. Proceed as directed by parameter file

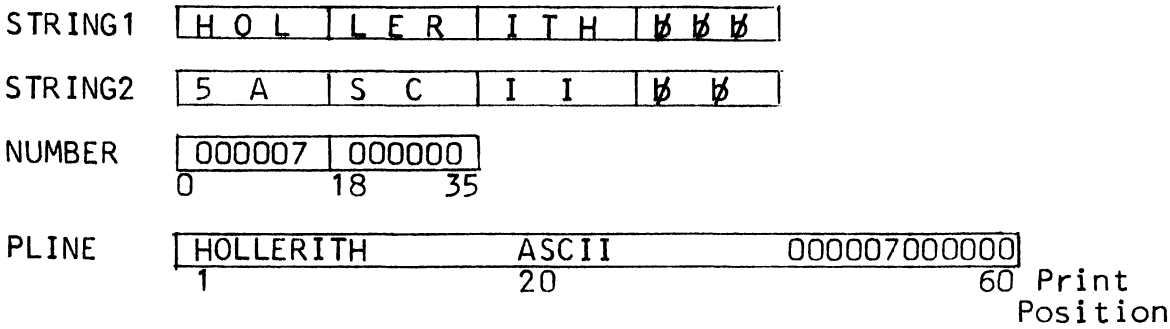
EXAMPLE:

Procedure Segment

```
prnt(list)
```

Data Segment

```
list:  hth (string1,0,1,2)
       ath (string2,0,20,0)
       fio(number,0,60)
       prt(1,0,3)
```




COMMENTS: 1. The interpreter will truncate an ASCII or Hollerith string on the right, if necessary, so that it will not exceed to the right of print position 132.

2. Data may be placed in PLINE in any order; e.g., positions 22-35 may be filled before positions 3-10. Data may also be superimposed on other data

3. CAUTION: Always use a PRNT or PRNTC pop to place data in PLINE -- do not move any strings directly into the buffer.

POP: PRNTC Print continue

FORMAT: prntc(Y)

FUNCTION: Proceed as directed by parameter file
NOTE: This pop does not clear PLINE buffer

COMMENTS: See PRNT

c. Printing the Error List

POP: PRNT Print

FORMAT: prnt(0)

FUNCTION: Case 1: Input Stream on Cards -- Roll 1 (error roll) empty
Go to next pop

Case 2: Input Stream on Cards -- Roll 1 non-empty

1. Scan each word on error roll, and print each error message
2. Prune error roll

Case 3: Input Stream on Roll N -- Roll 1 empty

Set C(BOTTOM+N) 0-17 = RP(ROLPTR+N)

Case 4: Input Stream on Roll N -- Roll 1 non-empty

1. Sort error roll in ascending order
2. Place the following information in the error segment: each message on the error roll, with the corresponding FORTRAN statement (including pips). (See Error Pops, Paragraph X.) (The input stream must be a type-3 string).
3. Prune error roll
4. Set C(BOTTOM+N) 0-17 = RP(ROLPTR+N)

EXAMPLE:

The following messages are part of a FORTRAN error list:

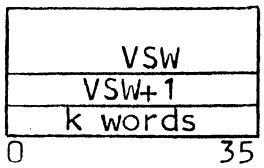
```
  38      IF(L(J)-MC(K)) 299,50,299
* * *     OPERAND IN WRONG MODE.
  67      999 FORMAT (15H1 ***SUB-LATINS )
* * *     MULTIPLY DEFINED EFN.
```

2. WBIN Pop

WBIN uses information on the binary and relbit rolls to produce text, linkage, and symbol segments.

a. Binary Roll Format

Each group on the binary roll defines a section of consecutive words in the text, linkage, or symbol segment. The binary roll contains variable-size groups with the following format:




C(VSW) 0-17 - Size of group

C(VSW) 18-35 - 0 if not labeled COMMON
Otherwise, offset of word on SYMREF roll
Note: This field is used only by FORTRAN IV, not by the interpreter.

C(VSW+1) 0-7 - Number of words in section (k)

k = 0,1,2,...,or 255

C(VSW+1) 8-11 - Segment type (in octal)

0-7 -- Illegal (does not apply here)
10 -- Object procedure (text segment)
11 -- Absolute (does not apply here)
12 -- Linkage (linkage segment)
13 -- Blank COMMON (does not apply here)
14 -- Stack (does not apply here)
15 -- Definitions section (in text or linkage segment)
16 -- Symbol segment
17 -- Illegal (does not apply here)

Note: Groups with types labeled "does not apply here" are ignored by the interpreter.

C(VSW+1) 12-17 - Ignored by the interpreter

C(VSW+1) 18-35 - Loading origin

k words -- Section of code to be moved to specified segment

b. Setup of Text and Linkage Segments

The interpreter uses the following data segment registers to determine the setup of the text and linkage segments:

TXTL

| # of words in text for obj. proc. | Ignored |
|---|---|
| 0 18 | 35 |

LNKL

| # of words in linkage for obj.proc. | Ignored |
|---|---|
| 0 18 | 35 |

PUTDEF

| 0 - Def. section follows text in text segment non-0 - Def. sec. follows linkage in linkage segment | Ignored |
|---|---|
| 0 18 | 35 |

DFSL

| # of words in definitions sect. | Ignored |
|---|---|
| 0 18 | 35 |

c. Loading Origin

The loading origin is an offset in the text, linkage, or symbol segment. It tells the interpreter where to begin loading the k-word section. For the definitions section, the loading origin is relative to the location of the first definition word. Otherwise, it is relative to the first word of the segment.

d. Relbit Roll Format

As each new group is being formed on the binary roll, relocation codes (if any) for that group are stored on the relbit roll. A 6-bit relocation code is stored for each half-word in the k-word section. These codes are stored contiguously, starting at the leftmost position in the relbit roll. Figure 4 shows the setup of the relbit roll.

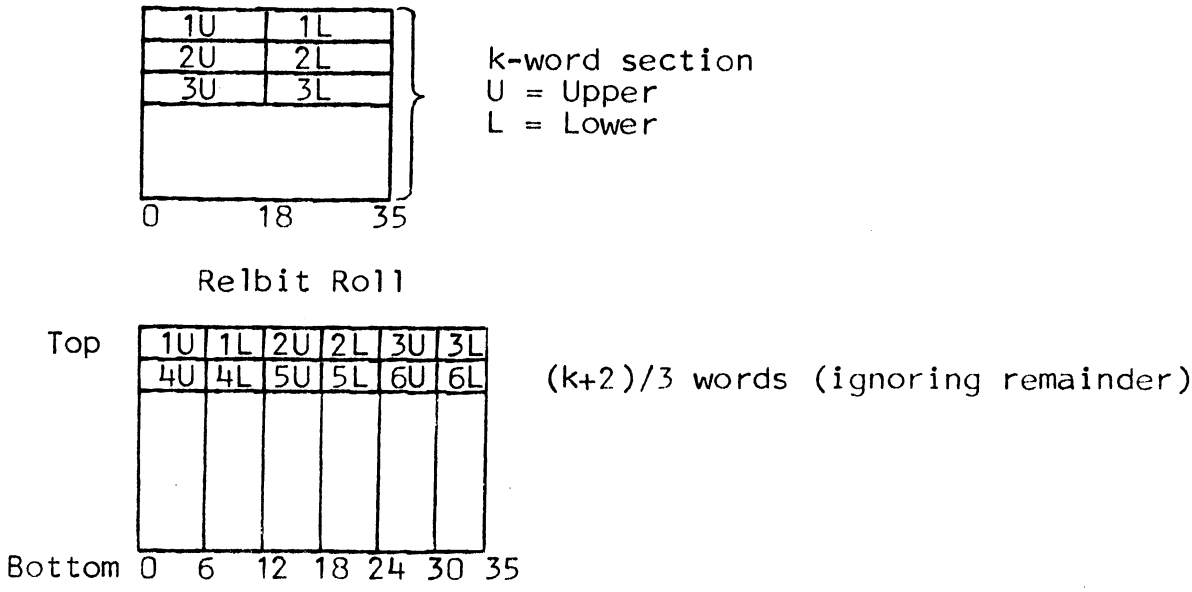



Figure 4. Setup of Relocation Information for k-word Section

The relocation codes have the following meanings:

```
   00 -- Multics code 0 (later squeezed to 1 bit)
01-17 -- Not used
20-37 -- Standard Multics codes (later squeezed to 5 bits)
40-77 -- Not used
```

e. Collection of Relbits

The interpreter uses the text and linkage segments to collect relbits. After all relbits have been collected, the interpreter squeezes the 6-bit relocation codes into 1-bit and 5-bit Multics relocation codes, and places this information into the symbol segment:

| Before | After | |
|---|---|---|
| 000000 | 0 | 1-bit string |
| 01xxxx | 1xxxx | 5-bit string |

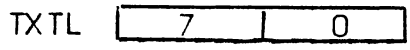
NOTE: These strings are concatenated; thus, 5-bit codes may overlap words.

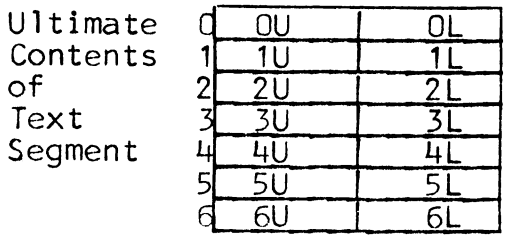
After all relbits have been moved to the symbol segment, the interpreter moves text to the text segment, linkage to the linkage segment, and symbols to the symbol segment.

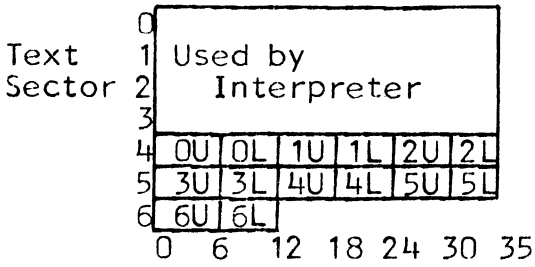
The relbits are collected in the text and linkage segments as follows: There are four classes of relbits: text, linkage, definitions, and symbol. The text and linkage segments are each ...ed into sectors. Each sector is approximately 2**18/3 words. The second sector of the text segment contains text relbits, the third sector of the text segment contains definitions relbits, the second sector of the linkage segment contains linkage relbits, and the third sector of the linkage segment contains symbol relbits. The first sector in each segment is not used in the collection of relbits. Since Multics preclears each unused page on first reference, the interpreter does not preclear the sectors before collecting relbits.

The sectors containing relbits have the following setup: The first four words of each sector are used by the interpreter to store pointers. The remaining words in the sector contain relocation codes in the same format as they would appear on the relbit roll.

EXAMPLE:



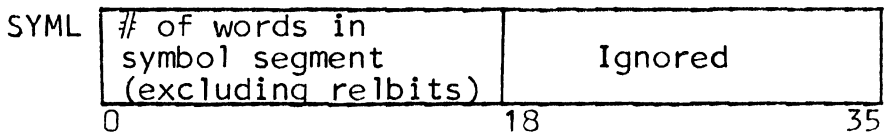




f. Symbol Segment

The first portion of the symbol segment contains symbols which are moved from the binary roll. Following this are three sections of relbits for the text, linkage, and symbol segments, respectively.

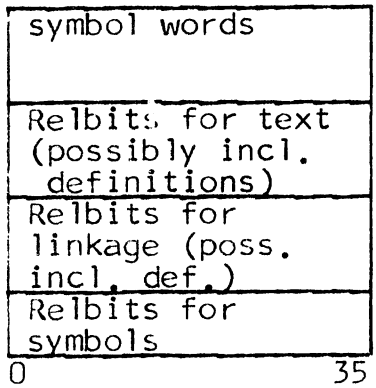
The interpreter uses the contents of the SYML register in the data segment to determine the setup of the symbol segment:



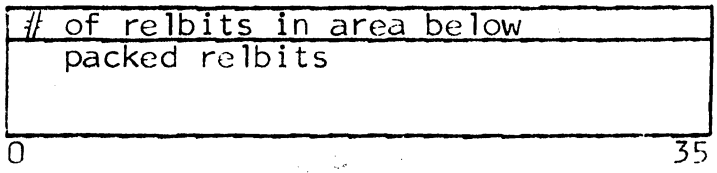

Relbits for the definitions section are concatenated with those for the text segment or for the linkage segment, depending on the value of PUTDEF.

Relbits for the symbol segment pertain only to the symbol words stored in the first portion.

The setup of the symbol segment is shown below:

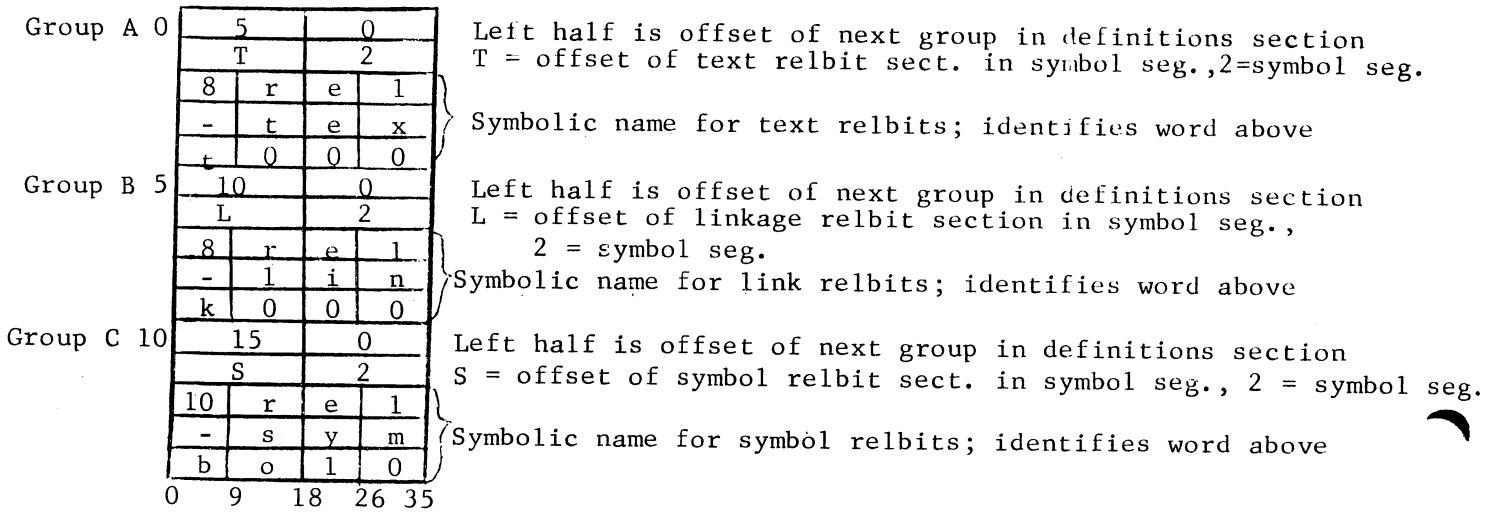



The following illustrates the format of a typical relbits section:



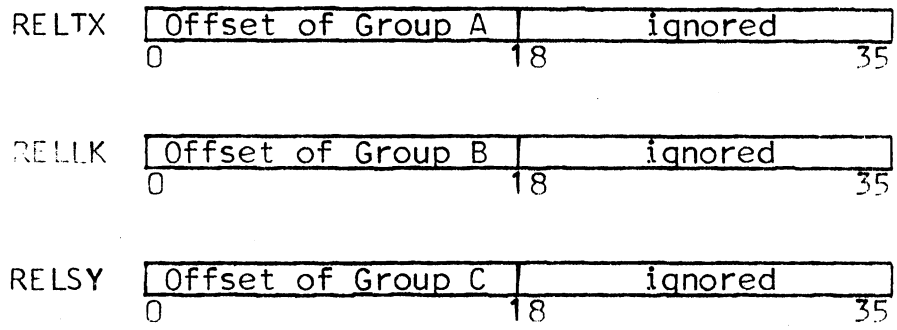

g. Definitions Section

The definitions section should begin with the following words:



The second word in each group is set by the interpreter. All other words are set by the user.

The offsets of these three groups in the definitions section are contained in three data segment registers set by the user:




If the definitions section is set up as described above, C(RELTX) 0-17 = 0, C(RELLK) 0-17 = 5, and C(RELSY) 0-17 = 10. This feature allows the user to place these groups anywhere in the definitions section.

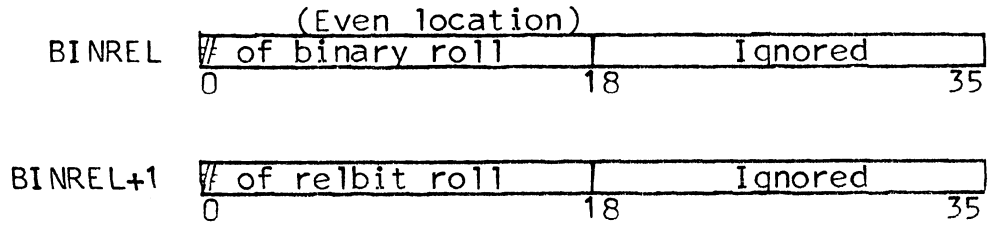
Pop

POP: WBIN Write Binary

FORMAT: wbin(0)
or
wbin(N) where N is the number of the binary roll

FUNCTION: Case 1: The operand is 0

This pop determines the numbers of the binary and relbit rolls from two data segment registers:



1. If relbit roll is empty, go to next pop.

   Otherwise, move 6-bit characters from relbit roll to proper sector in text or linkage segment.

2. Prune relbit roll

Case 2: The operand is N

1. If C(LNKL) 0-17 = 0, then set C(LNKL) 0-17 = 8, and create dummy 8-word linkage header:

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 2 | 0 | 0 |
| 3 | 0 | 0 |
| 4 | 0 | 0 |
| 5 | 0 | 0 |
| 6 | 0 | 8 |
| 7 | 0 | 8 |
| | 0 | 18 35 |

2. Squeeze relbits and move them to the symbol segment

3. Set the three relbit pointers in the definitions section.

4. Scan the binary roll and move the text, linkage, and symbol words to the proper segments.

COMMENTS:

1. The wbin(0) pop should be executed as each group is put on the binary roll (if the group contains relocatable information).

2. The wbin(N) pop should be executed only after the last group has been put on the binary roll.

3. If the operand of the WBIN pop is N, N overrides C(BINREL) 0-17

# Z. EXECUTIVE AND TERMINATION POPS

## 1. Snap Pops

There are two snap pops:

SNAPC -- Snap core

SNAP -- Snap panel, stacks, and rolls

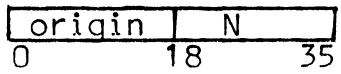
The format of each of these pops is pop(Y), where Y is the location of a file in the data segment.

All snap output appears in the error segment.

### a. Core Dumps

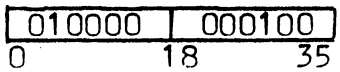
#### 1) File Format

Each word of the file for a SNAPC pop has the following format:




If origin ≠ 777777, the file word directs the interpreter to snap N words in the data segment, starting at origin. The file may contain any number of words of this type.

EXAMPLE:




This word causes the interpreter to snap data segment locations 10000 -- 10077.

If origin = 777777, the file word signifies the end of the file. Every file must include a word of this type.

A typical file is:

| | |
|---|---|
| 010000 | 000100 |
| 034000 | 000040 |
| 777777 | 000000 |

0 18 35

2) Snap Format

SNAPC produces the following output:

1. The first line is:

   SNAP LOC nnnnnn

   where nnnnnn is the octal location of the SNAPC pop in the procedure segment.

2. Following this message, all snaps requested in the file appear in single-spaced format. Each line begins with the octal address of the first word snapped in the line. This is followed by eight octal words, in each line except the last. For example, 013100 indicates that the line snaps data segment locations 013100 -- 013107. The last line snaps one to eight words, depending on the number of words remaining; i.e., N = 8I + J (I lines of 8 words and 1 line of J words).

   A star appearing to the right of one of the octal addresses indicates that one or more lines were deleted. A deletion occurs whenever an 8-word pattern is repeated. Most deletions occur because of a block of zeros.

EXAMPLE:

```
000600
002020*
```

Here, the lines starting with the following addresses were deleted: 000610 -- 002010;

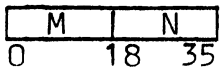
i.e., C(600) = C(610) = . . . = C(2010)
C(601) = C(611) = . . . = C(2011)
.
.
.
C(607) = C(617) = . . . = C(2017)

The last line of a snap is never deleted.

b. Panel, Stack, and Roll Dumps

1) File Format

Each word of the file for a SNAP pop has the following format:



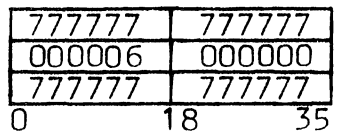

The significance of these fields is summarized in the following chart:

| Sign-bit of M | Sign-bit of N | Meaning in First Word | Meaning After First Word |
|---|---|---|---|
| 0 | 0 | Snap roll M. If N≠0, it is the location of a message to precede the snap; the message is a type-1 string (generally the name of the roll) in the data segment. | Same as in first word |
| 0 | 1 | N is a negative two's complement number. Snap \|N\| rolls starting at roll M: i.e., snap rolls M through M+\|N\| - 1<br>EXAMPLE: 000010 \| 777774 (0, 18, 35)<br>says snap rolls 10,11, 12, and 13 (octal). (777774=-4) | Same as in first word |
| 1 | 0 | Snap stacks. | End of file |
| 1 | 1 | Snap stacks and panel. | End of file |

If the sign bit of the first word = 1, then the first word is a special word; its interpretation is different from that of the following words in the file. There are three possible types of setups:

| | | |
|---|---|---|
| special word | special word | roll words |
| roll words | end word | end word |
| end word | | |

The sign bit of the end word = 1. A typical file is:




Snap stacks and panel
Snap roll 6 (no message)
End of file

2) Panel Snaps

A panel snap displays the following registers in the data segment: SYMTLY, SYMCNT, SYMBUF-1, FEXIT, SYMKEY, CHAR, CHARC, MODES, CONMOD, VARSIZ, MRKER, FCNT, DSCALE, BSCALE, DSIGN, BSIGN, SYMBUF, CURPTR, RMD, MPAC, TLYIN, CONBUF, and ALTER. The identification of the register appears above its contents.

3) Stack Snaps

A stack snap displays the currently used words in the work, dummy, and exit stacks. A word in the left margin identifies the stack: WORK, DMY, or EXIT. The format is similar to that of a core snap. However, there is no suppression of lines; and the addresses are relative to the beginning of each stack. The appearance of addresses anywhere within the range 777000 -- 777770 indicates that the stack was over-pruned. In this case, the snap displays the over-pruned area.

EXAMPLE:

```
(-100)  777700
(- 70)  777710
          .          Indicates octal 100(i.e.,64) words
          .          over-pruned
          .
        777770
```

The stack snap is single spaced.

4) Roll Snaps

A roll snap may be preceded by a message (see the chart on page 140). The format of the snap depends on the characteristics of the roll:

1. If the roll is empty or over-pruned, the following message appears: n EMPTY, where n is the roll number in decimal. There is nothing else in the snap.

2. If the roll is not open, the following message appears: n NOT OPEN, where n is the roll number in decimal. There is nothing else in the snap.

3. Otherwise, the snap displays the roll from the anchor up to (but not including) the bottom.

The first two items on each line are: the roll number (in decimal), and the location in octal of the first word in the line relative to the anchor. These are followed by eight octal words in each line except the last. If the number of words from anchor to bottom is a multiple

of eight, then the last octal line contains simply the roll number and the offset of the bottom from the anchor. Otherwise, the last octal line also displays one to seven words. These octal lines are double spaced.

The following subtitles appear below the octal words to which they apply:

^ANCHOR
^TOP
^ANCHOR,TOP

One of the following subtitles appears below the empty space corresponding to the bottom:

^BOTTOM
^TOP,BOTTOM

EXAMPLE:

```
003726000000
             ^BOTTOM
```

If the last line of the roll dump contains no items of the roll, then ^BOTTOM or ^TOP,BOTTOM appears below the first empty space.

Suppression is possible. However, only the lines under which no subtitles appear may be suppressed. The first and the last lines are always printed.

The following information appears below the indication of the bottom:

n1 TO FLOOR VAR GRPSIZ(or n2 GRPSIZ) n3 GUESS n4 ROLPTR

where: n1 = number of words from bottom to floor

n2 = number of words in fixed-size groups
(VAR indicates variable-size groups)

n3 = number of words in the initial guess

n4 = C(ROLPTR+N) 0-17. If no number appears here then C(ROLPTR+N) = 0.

These are all decimal numbers.

For roll 0 only, the snap concludes with a table showing the start of each of the 32 threads in the roll, the number of links in each thread, and the number of references made to each thread (by SRCH or SRCHP).

Pops

POP: SNAPC Snap core

FORMAT: snapc(Y)

Y is the location of the first word of a file (See Paragraph Z.1.a.)

FUNCTION: Produce a snap, as directed by the file

POP: SNAP Snap panel, stacks and rolls

FORMAT: snap(Y)

Y is the location of the first word of a file (See Paragraph Z.1.b.)

FUNCTION: Produce a snap, as directed by the file

2. Termination Pops

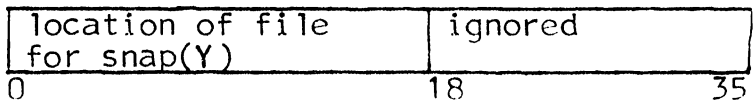
Pops

POP: BORT Abort procedure

FORMAT: bort( )

FUNCTION: 1. Snap data segment locations 0 to location preceding top of roll 0

2. Execute a snap(Y) pop, using C(ROLDMP) 0-17 as the file location. ROLDMP is a register in the data segment with the following format:




3. Terminate the procedure (See Chapter 1, Paragraph F)

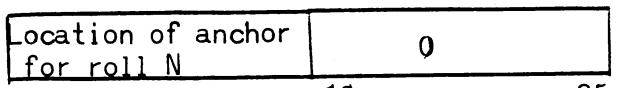
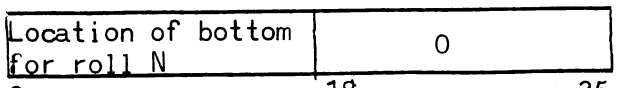
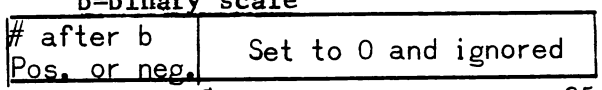
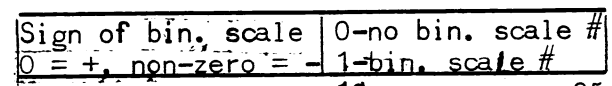
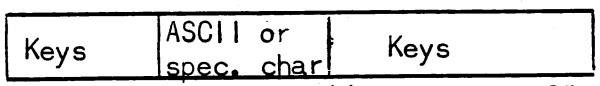
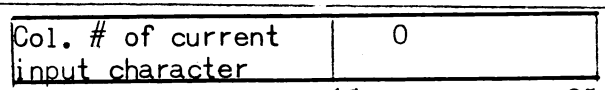
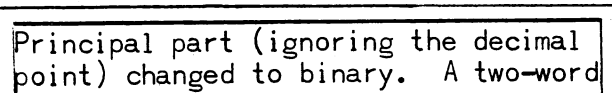
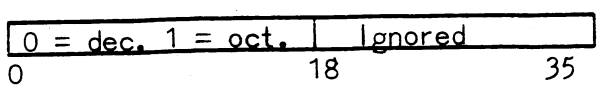
POP: FIN Finish procedure

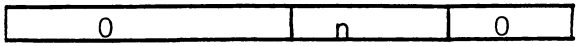
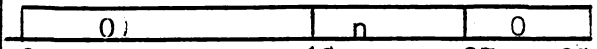
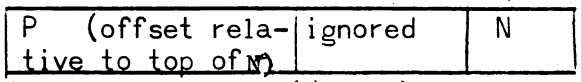
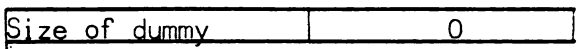
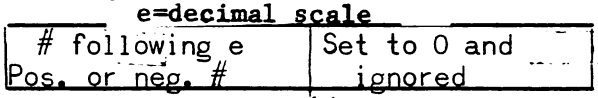
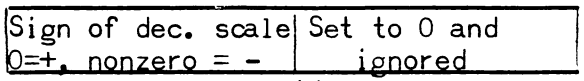
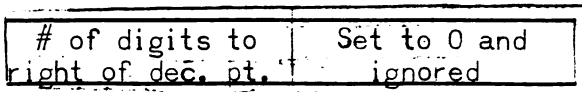
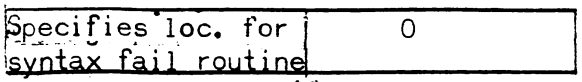
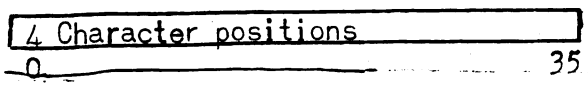
FORMAT: fin( )

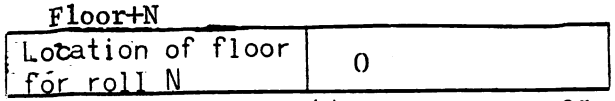
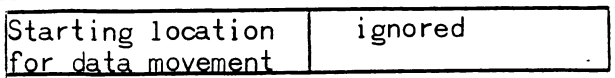
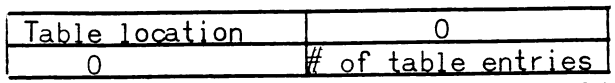
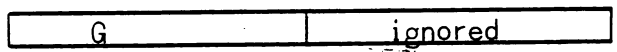
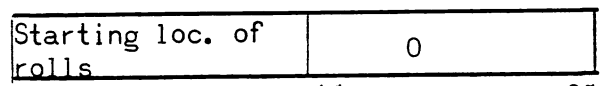
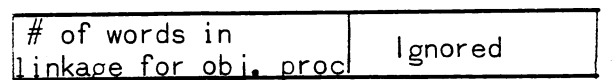
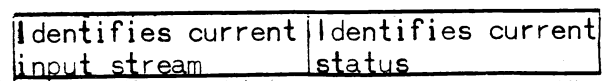
FUNCTION: Terminate the procedure (See Chapter 1, Paragraph F.)

# Appendix A

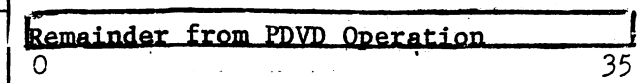
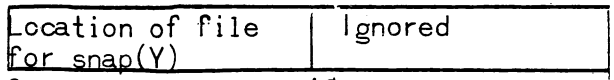
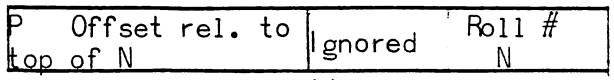
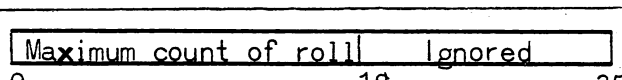
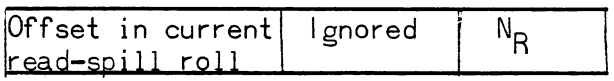
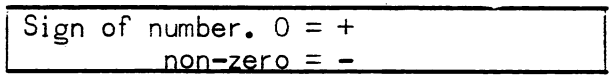
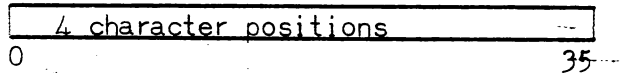
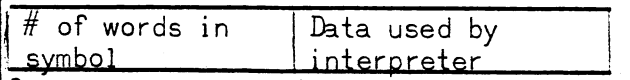
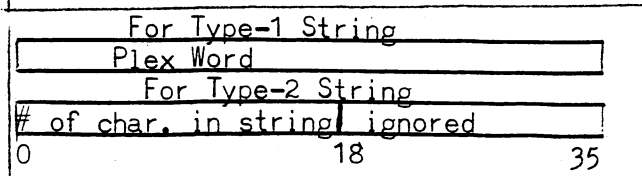
## Summary of Commonly Used Areas in the Data Segment

| Data Area | Illustration | Set By | Used By | Cleared By |
|---|---|---|---|---|
| Anchor table | ANCHOR+N<br>  | Initialization routine<br>Roll expansion routine and pops that put information on the bottom of a roll | Pops that refer to rolls<br>Roll expansion routine<br>OPN, REL, REMOV, RSV<br>RSVM, RWND, SNAP | Lower half cleared by initialization routine |
| Bottom Table | BOTTOM+N<br>  | Initialization routine<br>Roll expansion routine<br>CPYP, CPYR, GOBP, OPN, PBCT, PBCTP, PLG, PRNT(0), PRU, PTCT, PTCTP, PTP, PTPP, REL, REMOV, RSV, RSVM, RWND | Pops that refer to rolls<br>Roll expansion routine<br>CNT, CNTG, CPY, CPYP, CPYR, CPYX, CPYXP, DLOAD, DNG, DNX, ERB, GOB, GOBP, INSB, OPN, PBCT, PBCTP, PDES, PLG, PLXG, PRES, PSAV, RSV, RSVM, SCNT, SNAP, SMEB, SORTR, ULOAD, UNG, UNX | Lower half cleared by initialization routine |
| BSCALE | b=binary scale<br>  | CONBA | FXDD, FXDS, SNAP | CONR, CONARR, RNUM |
| BSIGN |  | User (upper)<br>CONBA(lower) | FXDD, FXDS, SNAP | CONR, CONARR, RNUM |
| CHAR |  | Character input subrt.<br>SWIP, SWAP | Character input subrt.<br>CONBA, CONDA, RCH, RCHA, RCKY, RCKYA, SCH, SCHA, SCKY, SCKYA, SNAP | |
| CHARC |  | Character input subrt.<br>SWIP, SWAP | Character input subrt.<br>ERRCC, ERRLC, SNAP | |
| CONBUF &CONBUF+1 |  | CON, CONA, CONAR, CONR, FIXD, FIXS | FLTD, FLTS, FXDS, FXDD, SNAP | CONR, CONARR, RNUM |
| CONMOD |  | MODD,MODO | CON,CONA,CONAR,CONBA, CONDA,CONR,SNAP | |

| Data Area | Illustration | Set By | Used By | Cleared By |
|---|---|---|---|---|
| CRDNUM |  where n = # of last group processed in current type-3 string | Character input subrt. SWAP, SWIP | Character input subrt. EROR, ERRCC, ERRLC, ERRP, PRNT | |
| CRDNUM+1 |  where n = number of groups in current type-3 string | Character input subrt. SWAP, SWIP | Character input subrt. EROR, ERRCC, ERRLC, ERRP, PRNT | |
| CURPTR |  | LINKP, SRCH, SRCHP | SNAP | |
| DFSL |  | User | WBIN | |
| DMYSIZ |  | User | Initialization routine | |
| DSCALE |  | CONDA | FLTD, FLTS, FXDD, FXDS, SNAP | CONR, CONAR, RNUM |
| DSIGN |  | User | FLTD, FLTS, FXDD, FXDS, SNAP | CONR, CONAR, RNUM |
| FCNT |  | User | FLTD, FLTS, FXDD, FXDS, SNAP | CONR, CONAR, RNUM |
| FEXIT |  | FEX | RCH, RCHA, RCKY, RCKYA, RSKY, RSKYA, RSY, RSYA | |
| FUNBUF | FUNBUF+n where n = offset of word in FUNBUF<br>  | Interpreter | User | |

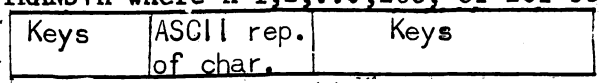
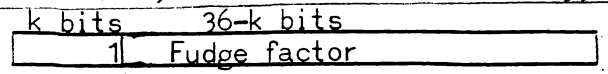
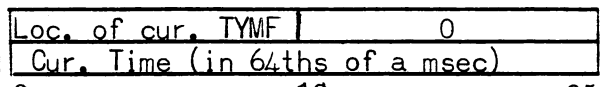
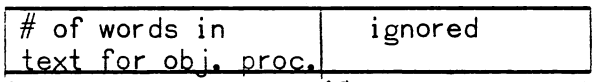
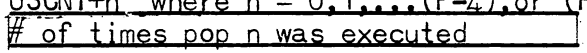
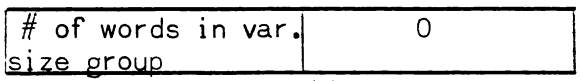
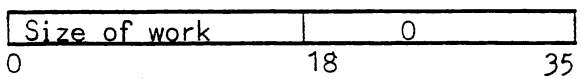
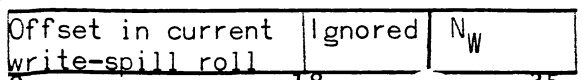
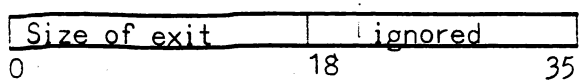
| Data Area | Illustration | Set By | Used By | Cleared By |
|---|---|---|---|---|
| Floor Table | Floor+N<br>  <br> | See Anchor Table | See Anchor Table | See Anchor Table |
| FROM |  <br> | MOVF | MOVT | |
| FTAB<br>FTAB+1 |  <br> | User | TSRCH | |
| GRPSIZ | GRPSIZ+N<br>  <br><br>G≠0 - Roll N contains G-word groups<br>G=0 - Roll N contains var. size groups | User | CPYG, DNG, ERB, INSB, PLG, SMEB, SNAP, SRCH, SRCHP, UNG, ZBG | |
| GUESS | GUESS+N<br>  <br> | User | Initialization routine SNAP | |
| LAST |  <br> | User | Initialization routine | |
| LNKL |  <br> | User | WBIN | |
| MODES |  <br> | Character input subrt. SWAP, SWIP | SNAP | |

| Data Area | Illustration | Set By | Used By | Cleared |
|---|---|---|---|---|
| MPAC<br>MPAC+1 | Used for storage of fixed-point and floating-point numbers<br>0 35 | FLT,PADD,PADDF PDVD, PDVDF,PMLT, PMLTD, PMLTF,PSUB, PSUBF | PADD,PADDF,PDVD,PDVDF PMLT,PMLTD,PMLTF,PSUB, PSUBF,SNAP | |
| MRKER | Roll number \| ignored<br>0 18 35 | MRK,RSVM,TSRCH | CPY,CPYG,CPYGB,CPYP,CPYR, CPYX,CPYXP,ERB,INSB,LINKP,POBS SMEB,SNAP, SRCH,SRCHP | |
| MTEST | **0-Roll movement statistics given if debug version<br>non-0-Roll movement statistics never given**<br>0 35 | User | Roll expansion routine in debug version. Ignored in production version. | |
| OPNERS | # of rolls to be opened \| 0<br>0 18 35 | User | Initialization routine<br>Roll expansion routine | |
| OUTAG4 | Address used to det. effective address for PRNT and PRNTC \| 0<br>0 18 35 | User | PRNT(Y), PRNTC(Y) | |
| OUTAG5 | Same as OUTAG4 | Same as OUTAG4 | Same as OUTAG4 | |
| OUTAG6 | Same as OUTAG4 | Same as OUTAG4 | Same as OUTAG4 | |
| PLINE Buffer | PLINE+n where n = 0,1,...,35,or 36<br>Four print positions<br>0 35 | PRNT(Y), PRNTC(Y) | | PRNT(Y) |
| PUTDEF | 0 - Def section follows text in text segment<br>non-0 - Def. sec. follows linkage in linkage segment \| Ignored<br>0 18 35 | User | WBIN | |

| Data Area | Illustration | Set By | Used By | Cleared By |
|---|---|---|---|---|
| RMD |  | PDVD | SNAP | |
| ROLDMP |  | User | BORT | |
| ROLPTR |  | DLOAD, DNG, DNX, LINKN LINKP, SRCH, SRCHK, SRCHKC, SRCHP, ULOAD, UNG, UNX, ZBG | CCAT, CNTG, CPYG, DLOAD, DNG, DNX, LINKN, LINKP, PLXM PRNT(U), SRCH, SRCHKC, SRCHP, SWAP, SWIP, SNAP, ULOAD, UNG, UNX | DLOAD, DNG, DNX, ULOAD, UNG, UNX (Also by user prior to one of above pops.) |
| RSIZE |  | Roll expansion routine (only in debug version) | Termination routine (only in debug version) | Initialization routine (only in debug version) |
| RSPTR |  where $N_R$ = Read-Spill roll | User RWND | Spill routine | RWND |
| SIGN |  | User | FLTD, FLTS, FXDD, FXDS | CONR, CONAR, RNUM |
| SYMBUF | SYMBUF+n where n = 0,1,...128 or 129<br>  | CNTS, NXST, NXSTC, NXSTCS PAK, PAKA, PAKAR, PAKR, PLXG, PLXP, RSY, RSYA, RSYM, SSY, SSYA | CCAT, CNTS, LINKR, PAK, PAKA, PAKAR, PAKR, PLXP, RSY, RSYA, RSYM, SNAP, SRCH, SRCHP, SSY, SSYA, SWAP, SWIP, TSRCH | PAKR, PAKAR, RSYM |
| SYMCNT |  | CNTS, PAKAR, PAKR, PLXG, PLXP, RSY, RSYA, RSYM, SSY, SSYA | PLXG, PLXP, RSY, RSYA, SNAP, SSY, SSYA | PAKAR, PAKR, RSYM |
| SYMCNT+1 |  | PLXG<br>Character Input Subrt. | SNAP<br>Character input subrt. | RSYM, PAKAR, PAKR |

| Data Area | Illustration | Set By | Used By | Cleared By |
|---|---|---|---|---|
| STAR1F | 0 - Special function buffer not used<br>non-0 - Special function buffer used<br>0 35 | User | Character input subroutine | |
| STRMTY | FUNBUF \| 4n = 1 \| 41(oct)<br>0 18 30 35<br>FUNBUF - Location of first word in function buffer<br>n - Number of words in function buffer | User | Character input subroutine | |
| ESCAPE | 0 - Ignore NL character and its keys<br>non-0 - Pack NL character into FUNBUF and examine keys<br>0 35 | User | Character input subroutine | |
| TOGOOD | 1- Funct. Buff. full<br>2- Roll 0 thread table to be written \| Ignore<br>0 18 35 | User | Executed to(-1) pop<br>(Any other value means that to(-1) is illegal.) | |
| BINREL | # of binary roll \| Ignored<br>0 18 35 | User | WBIN | |
| BINREL+1 | # of rel bit roll \| Ignored<br>0 18 35 | User | WBIN | |
| RELTX | Offset of link word for **rel_text** in definitions section \| Ignored<br>0 18 35 | User | WBIN | |
| RELLK | Offset of link word for **rel_link** in def. section \| Ignored<br>0 18 35 | User | WBIN | |
| RELSY | Offset of link word for **rel_symbol** in definitions section \| Ignored<br>0 18 35 | User | WBIN | |

| Data Area | Illustration | Set By | Used By | Cleared |
|---|---|---|---|---|
| SYMKEY | Ignored (0–18) \| Value used in symbol comparison (18–35) | ORKEY | RSKY, RSKYA, SNAP, SSKY, SSKYA | User |
| SYML | Offset in symbol segment (0–18) \| Ignored (18–35) | User | WBIN | |
| THREAD | C(THREAD+n) where n = 1,2,...31,or32<br>0 - empty thread<br>non-0 - abs. loc. of 1st link word on thread (0–18) \| Ignored (18–35) | SRCHP | SRCH,SRCHK,SRCHKC, SRCHP | Initialization |
| TLYIN | GE-645 Tally Word - Points to next avail. char. in the current input strm (0–35) | Character input subrt. SWAP, SWIP | SNAP | |
| TOCNT | # of illegal exect. of TO pop that may occur before an abort (0–35) | User<br>TO | TO | |
| TOP TABLE | TOP+N<br>Location of top for roll N (0–18) \| 0 (18–35) | Any pop that puts info. on the bottom of a roll<br>CPYR,OPN,REL,REMOV,RSV RSVM,RWND | All pops that refer to rolls | Lower half cleared by initialization routine |
| PKFRSW | Index (0–18) \| Ignored (18–35)<br>Index Meaning<br>0 Get pack-from option from NXST file (as usual)<br>1 No pack-from this time; set C(PKFRSW) = 0 for next time<br>2 Pack-from this time; set C(PKFRSW) = 0 for next time<br>3 No pack from<br>4 Pack-from | User | Character Input subroutine | |

| Data Area | Illustration | Set By | Used By | Cleared by |
|---|---|---|---|---|
| TRANS | TRANS+n where n=1,2,...,200, or 201 octal<br>  | User | Character input subrt. | |
| TYMASK |  <br>k-bit field - flag set here, to indicate timing information follows<br>fudge factor - equals time spent in execut. timing pops; this time may be omitted from time calculation | User | TYMT | |
| TYMER<br>TYMER+1 |  | TYMF | TYMT | |
| TXTL |  | User | WBIN | |
| USCNT | USCNT+n where n = 0,1,...,(F-4),or (F-5)<br>  <br>Note: F = current value of false tag | Execution of each pop (Debug version only) | Termination routine (Debug version only) | Initialization routine (Debug version only) |
| VARSIZ |  | User<br>CPYGB | ERB,INSB,PLG,SMEB, PLG<br>SNAP, ZBG | User |
| WRKSIZ |  | User | Initialization routine | |
| WSPTR |  <br>where Nw=write-spill roll | User | Spill routine | |
| XITSIZ |  | User | Initialization routine | |

Appendix B

SPECIAL FEATURES

## Function Buffer

Characters from the current input stream may be packed in a special function buffer, located in the data segment. This feature is currently used only by FL/I.

The interpreter examines STAR1F, a register in the data segment, to determine whether to pack characters into the function buffer:

STAR1F

| 0(FORTRAN console input or any string input stream) - buffer not used<br>non-0 (FL/I console input stream only)- buffer used |
|---|

0 35

If C(STAR1F) ≠ 0, then STRMTY, another data-segment register, is interpreted as a GE-645 tally word. The pops procedure is responsible for setting STRMTY to its initial value; FL/I initializes this register as follows:

STRMTY

| FUNBUF | 4n - 1 | 41(oct) |
|---|---|---|

0 18 30 35

FUNBUF - Location of first word in function buffer.
n - Number of words in function buffer

The interpreter packs all non-skipped characters into the function buffer. On end-of-line, the interpreter continues with the first character from the next line; if there are no more lines, it goes to the next pop. Its treatment of the (NL) character depends on the contents of the ESCAPE register:

ESCAPE

| 0 - Ignore (NL) character and its keys<br>non-0 - Pack (NL) character into FUNBUF<br>and examine keys |
|---|

0 35

If C(STAR1F) = 0, the interpreter ignores C(ESCAPE).

When the function buffer is full, the interpreter sets the data segment register TOGOOD, as follows:

| 000001 | 000000 |
|---|---|

0 18 35

The interpreter then executes the pop at location 8 (decimal) in the procedure segment. This pop may be a jump to a routine that copies the contents of the function buffer onto a roll and reinitializes the buffer. The last executed pop in the routine should be to(-1):

| 777777 | 000020 |
|---|---|

0 18 35

Special Version of TO Pop -- to(-1)

POP: TO

FORMAT: to(-1)

FUNCTION: If C(TOGOOD) 0-17 = 1, return to point in character input routine at which function buffer overflow occurred, and proceed as if no overflow.

If C(TOGOOD) 0-17 = 2, then call the symout entry in the FL/I command to write out roll 0 and the thread table. This feature allows the user to predefine symbols. FL/I uses this feature to save the symbol table for initialization on future assemblies. (See MSPM BX.7.01, The fl/1 Command.)

Otherwise, perform function of ordinary TO pop.

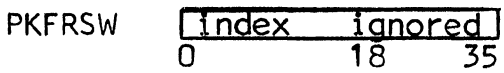
COMMENT: to(-1) clears C(TOGOOD) after testing it.

CCAT Pop -- Abnormal Case

If the total number of characters in a concatenated string would be > 511, then the interpreter sets C(TOGOOD) 0-17 = 3, and executes the pop at location 10(decimal) in the procedure segment. This pop may be a jump to a routine that starts a new string instead. This routine should not contain a to(-1) pop.

Pack-From Switch

The pack-from switch, PKFRSW, regulates the use of the pack-from option (see NXST). This register, located in the data segment, has the following format:




| Value of Index | Meaning |
|---|---|
| 0 | Get pack-from option from NXST file (as usual) |
| 1 | No pack-from this time; set C(PKFRSW) = 0 for next time |
| 2 | Pack-from this time; set C(PKFRSW) = 0 for next time |
| 3 | No pack-from |
| 4 | Pack-from |