Published: 08/16/68

Identification

Implementation
B. P. Goldberg, I. B. Goldberg

Chapter 1.

INTRODUCTION

A. PSEUDO-COMPUTER

Pops are machine instructions for a pseudo-computer simulated on the GE-645. This pseudo-computer was designed specifically for writing compilers and assemblers. Each pop corresponds to a GE-645 machine language subroutine. In order to execute a pops language program, an interpreter interprets each pop and calls the proper subroutine.

This process involves the following segments:

Procedure segment (pure) -- Contains the compiler or assembler logic written in pops language

Interpreter segment (pure) -- Contains the interpreter

Data segment (impure) -- Contains the memory and registers of the pseudo-computer

Input segment (pure) -- Contains the source procedure to compile or assemble

List segment (impure) -- Contains object listing

Error segment (impure) -- Contains error messages

Text segment (impure)
Linkage segment (impure) } For object procedure
Symbol segment (impure)

Each segment contains up to 262144 36-bit words. The parenthetical remarks "pure" and "impure" apply only when the pops interpreter is simulating the pops procedure.

Figure 1 illustrates the relationship between the interpreter, procedure, and data segments:

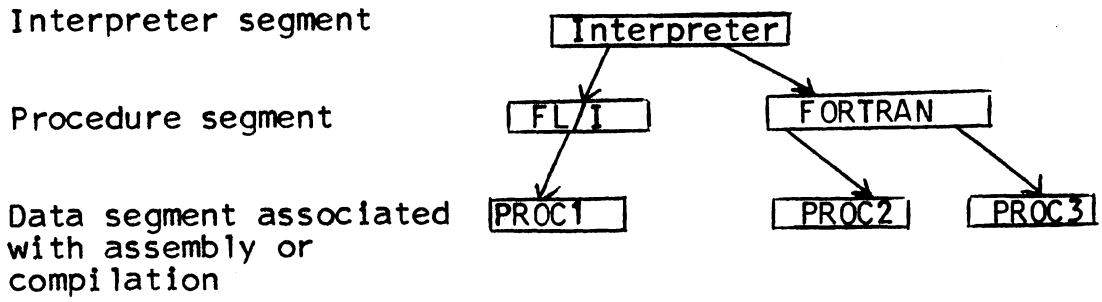


Figure 1. Relationships Between Segments in Pseudo-Computer

B. POPS LANGUAGE

1. Pop Format

A pop is a 36-bit word consisting of two 18-bit halves, the operand and the pop number.

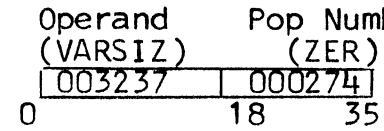
In the FL/I syntax used in this manual, the pop name is a function, and its operand is the first argument. Arguments are enclosed by parentheses and separated by commas. (See Paragraph B.2 for an example of the use of two arguments.) Comments are enclosed in quotes.

CAUTION: A semicolon, quote, or tilde in a comment must be preceded by a tilde (octal 176). For simplicity, this manual does not use these three characters in comments.

EXAMPLE:

POP: zer(varsiz)

Assume that VARSIZ is at location 3237 in the data segment.



(These are octal numbers)

Each pop number represents a pop name. The 250 pop names are listed in Figure 2. The pop number of each pop is determined by an octal addition of the appropriate row and column number; e.g., ZER = 270+4.

TABLE OF POP NUMBERS -- F = 256 -- I = 512.

+	0	1	2	3	4	5	6	7
000	BORT	USER1	USER2	USER3	USER4	USER5	ADD	ADDI
010	ADDIP	ADS	ADSP	AND	ANDI	ANDIP	ANS	ANSP
020	TO	CCAT	CEAW	CLOAD	CNT	CNTG	CNTS	CON
030	CONA	CONBA	CONBA	CPY	CPYP	CPYG	CPYR	CPYX
040	CPYXP	D0	D1	D2	D3	D4	D5	DCR
050	DLOAD	DNG	DNX	DNY	DOML	DVD	EAW	EOR
060	ERB	INSB	SMEB	EROR	ERRP	ERRCC	ERRLC	ERS
070	ERSP	EXEC	EXIT	EXT	EXTI	EXTIP	FACT	FEX
100	FIN	FIXD	FIXS	FLTD	FLTS	FXDD	FXDS	GOB
110	GOBP	INC	INS	INS1	INS2	ZERD	INSI	INSIP
120	JMP	JMPP	JNX	JSB	LINKN	LOAD	MLT	MLTS
130	MLTSP	MODB	MODNB	MODD	MODO	MOV	MOVF	MOVT
140	MRK	NGT	NGTS	NOT	NXCH	NXICH	NXST	NXSTC
150	NXSTCS	ONE	OPN	OR	ORKEY	ORS	ORSP	PADD
160	PADDF	PAK	PAKA	PBCT	PBCTP	PDES	PDVD	PDVDF
170	PLG	PLXS	PLXM	PLXP	PMLT	PMLTD	PMLTP	POB
200	POBP	POBS	POPNOP	PRD	PRE	PRES	PRNT	PRNTC
210	CPYGB	PRU	PRW	PRWX	PSAV	PSUB	PSUBT	PTCT
220	PTCTP	PTP	PTPP	PWCT	RCH	RCHA	RCKY	RCKYA
230	REL	REMOV	RNUM	CONR	CONAR	RSKY	RSKYA	RSV
240	RSVM	RSY	RSYA	RSYM	PAKR	PAKAR	RWND	SBS
250	SBSP	SCA	SCAP	SCH	SCHA	SCKY	SCKYA	SCNT
260	SEQ	SEQP	SEV	SEVS	SFAL	SGT	SGTP	SLT
270	SLTP	SME	SME1	SME2	ZER	SMEI	SMEIP	SMEP
300	SNAP	SNAPC	SNZ	SNZS	SORTR	SRCH	SRCHP	LINKP
310	SRCHK	SRCHKC	SSKY	SSKYA	SSY	SSYA	STRU	STOR
320	STORP	SUB	SWAP	SWIP	TSRCH	TYMF	TYMT	ULOAD
330	UNG	UNX	W0	W1	W2	W3	W4	W5
340	WBIN	WRKL	WRKR	XCH	XLR	XLRS	XNDW	XNIP
350	XNIV	XNPF	XNPV	XNTF	XNTV	ZBG	XNFP	XNVP
360	FLT	EXITP	A5	A4	A3	A2	A1	A0
370	STU	STUP						
+	0	1	2	3	4	5	6	7

Figure 2

Table of Pop Numbers

The operand is interpreted in one of the following ways:

1. As the offset of another pop (relative to location 0 in the procedure segment)
2. As the offset of a data word (relative to location 0 in the data segment)
3. As a literal number
4. As another pop number (See Chapter 2, Paragraph G).

The operand may also be ignored. See the example of NXCH below for format.

The pop number determines how the operand is to be interpreted. This is illustrated in the following chart:

Pop Number	Pop Name	Interpretation of Operand	Example
274	ZER	Address in data segment	zer(varsiz)
120	JMP	Address in procedure segment	jmp(init)
144	NXCH	Ignored	nxch()
022	CEAW	Literal number	ceaw(4096)

Note: nxch() is equivalent to nxch(0)

2. True/False Indicator

The pseudo-computer has a true/false indicator. There is an option to execute a pop only if the indicator is set true or only if the indicator is set false. In source language, the T/F tag specifies this option. T and F have values that are added to the pop number. These values are derived as follows:

F = 6 + the number of pops

T = 2 * F

Since there are currently 250 pops, the current values of T and F are:

F = 256 decimal = 400 octal

T = 512 decimal = 1000 octal

The T/F tag must always be preceded by a comma. This is true even if the operand field is null, e.g., nxch(,t).

Note: nxch(,t) is equivalent to nxch(0,t)

EXAMPLES:

Pop: zer(sign)	Pop: zer(sign,t)	Pop: zer(sign,f)
Pop number: 274	Pop number: 1274	Pop number: 674

C. PROCEDURE SEGMENT

1. Reserved Locations

Locations 0 to 10 (decimal) in the procedure segment are reserved for the following purposes:

Dec. location	
0	Contains the pop jmp(fstpop), where FSTPOP is the offset of the first pop to be executed

Dec. location	
1	Contains the pop executed on "advance to next symbol" (See SSYA and RSYA pops)
2	Contains the pop executed on end-of-file (See NXCH pop)
3	Upper half contains offset of USER1 routine; lower half contains 0 (See Chapter 2, Paragraph X.)
4	Upper half contains offset of USER2 routine; lower half contains 0
5	Upper half contains offset of USER3 routine; lower half contains 0
6	Upper half contains offset of USER4 routine; lower half contains 0
7	Upper half contains offset of USER5 routine; lower half contains 0
8	Contains pop executed on function buffer overflow (See Appendix B)
9	Currently not used, but reserved for possible future use
10	Contains pop executed on concatenation overflow (See Appendix B)

Each offset above is relative to location 0 in the procedure segment.

2. Order of Execution

The interpreter first executes the pop at location FSTPOP. It then executes pops in sequence, unless a pop causes change of control (e.g., JMP) or reads a character from the input stream (e.g., NXCH). (See Chapter 2, Paragraphs E and J for details)

D. POP COUNTER

The pop counter is an 18-bit counter which points to the pop currently being executed. Therefore, it is the instruction counter of the pops machine. This counter is initially set to FSTPOP. It is normally incremented by 1 after the sequential execution of a pop.

If a pop causes change of control, then the interpreter sets the pop counter equal to the operand of the executed pop.

Any pop that reads a character or string from the current input stream causes the pop counter to be set as follows:

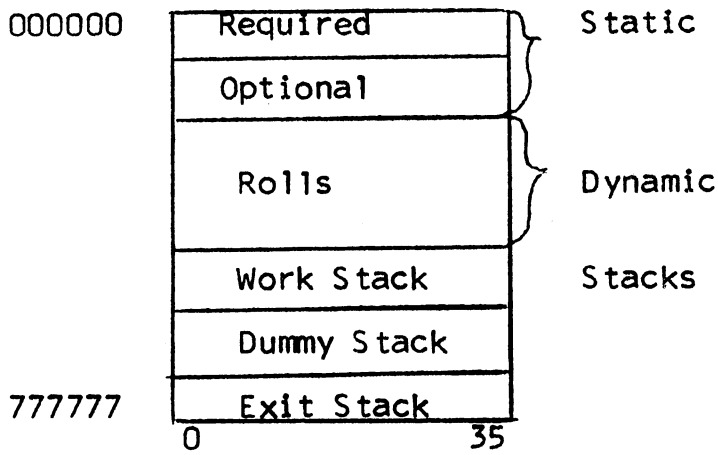
1. If end-of-line or end-of-stream has not been reached, then the pop counter is incremented by 2

2. Otherwise, the pop counter is incremented by 1.

The pop counter is simulated by the GE-645 index register 1.

E. DATA SEGMENT

The data segment has the following format:



1. Static Storage

The items in static storage are of fixed size and are in fixed locations.

The format of required storage is the same for each pops procedure. (See the FL/I assembly listing for a list of the items in required storage.) The optional storage differs for each procedure; i.e., optional storage for FL/I has a different format from that for FORTRAN. The optional storage contains constants, variables, and fixed-size tables that are used by the procedure.

The interpreter refers to locations in required storage directly; it refers to a location in optional storage either by a pointer in required storage or by using the operand of a pop (the latter method is used most frequently).

Three locations are fixed in every data segment:

000000 -- The first location

(See FL/I assembly listing) -- The start of optional storage

777777 -- The last location

The user specifies the following starting locations in one-word registers in the data segment:

Starting location of rolls -- Specified in C(LAST) 0-17

Starting locations of work, dummy, and exit -- Specified indirectly in C(WRKSIZ) 0-17, C(DMYSIZ) 0-17, and C(XITSIZ) 0-17 (See Paragraph E.3.)

Each of these registers should contain zero in the lower half.

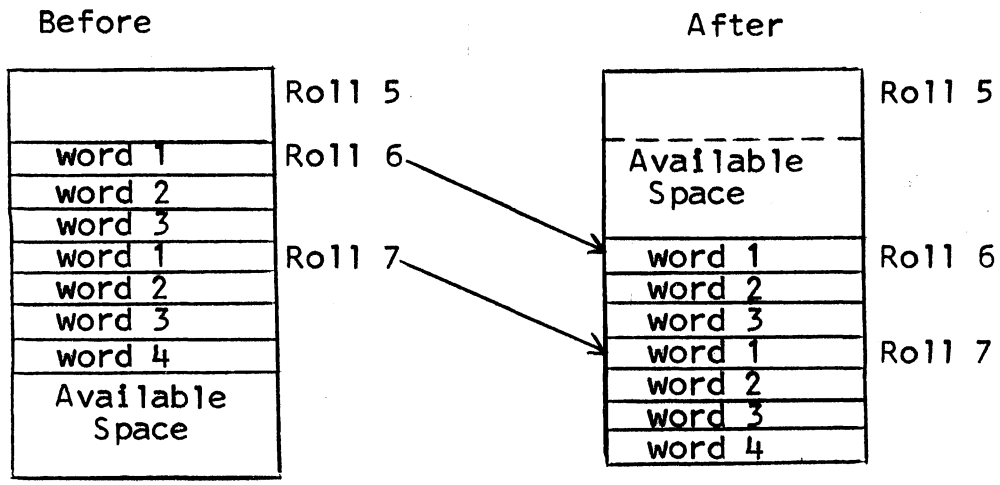
Most registers and tables storing half-word information use the upper half. (Exceptions are noted in this manual, whenever they occur.) For safety, the lower half of each of these words should contain zero, even though the interpreter may sometime ignore its contents.

2. Rolls

A roll is like a table in function; however, its size and location in core storage are dynamically variable during pops interpretation. Rolls are in the data segment, and there are a maximum of 64 rolls; the first is roll 0, the second, roll 1, etc. The notation Roll N is used, where N is 0,1,2,...63. Rolls are stored in sequential order by roll number, and words on rolls are stored in sequential locations.

The interpreter allocates an initial amount of space for each roll. If, during interpretation, available space in a particular roll is exceeded, the interpreter may allocate more space for that roll. Because storage remains sequential, this reallocation may cause other rolls to be moved. In this case, each word in the affected rolls is moved by a fixed amount; however, the order of the rolls does not change, nor does the relative position of data within a roll.

EXAMPLE:



Roll 0 is never moved; however, it may be expanded. All other rolls may be moved and expanded.

There are four significant locations on each roll:

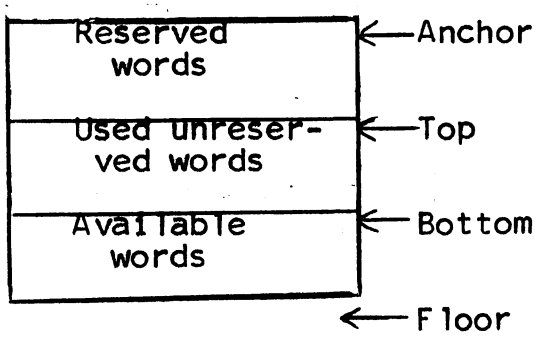
The anchor of a roll is the current location of the first word allocated for the roll.

The floor of a roll is one more than the current location of the last word allocated for the roll.

The top of a toll is the current location of the first unreserved word on the roll. (See RSV and REL pops.)

The bottom of a roll is one more than the current location of the last word used on that roll.

Each of these locations is relative to location 0 in the data segment. If there are no reserved words on a roll, then the top and the anchor are the same location. If no unreserved words are used, then the top and the bottom are the same location. If the roll contains no words at all, then the anchor, top, and bottom are the same location. The four locations are shown below:



The floor of each roll, except the last allocated roll, is the anchor of the next roll. Thus, the floor of the last allocated roll is the first location available for roll expansion. The user may make even more space available by removing rolls not in current use. (See OPN and REMOV pops.)

A roll expands downward from the anchor towards the floor. When the bottom reaches the floor, the interpreter must allocate more space to expand the roll. The interpreter appends a group of n consecutive words to a roll, as follows:

1. The interpreter bumps the bottom by n; i.e., moves the bottom down n words towards the floor.

2. If the bottom has reached (or passed) the floor, the interpreter allocates more words and adjusts anchor, top, bottom, and floor, if necessary.

3. The interpreter moves words 1 to n into the n locations above the new bottom.

Four tables in the data segment, each containing 64 entries, give the current anchor, top, bottom, and floor for each roll:

C(ANCHOR+N) 0-17 = anchor for roll N

C(TOP+N) 0-17 = top for roll N

C(BOTTOM+N) 0-17 = bottom for roll N

C(ANCHOR+N+1) 0-17 = floor for roll N

The lower half of each of these entries contains 0. This portion is initially cleared by the interpreter.

The anchor and floor tables are interwoven, as shown below:

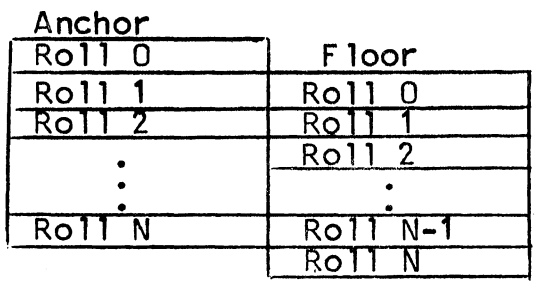
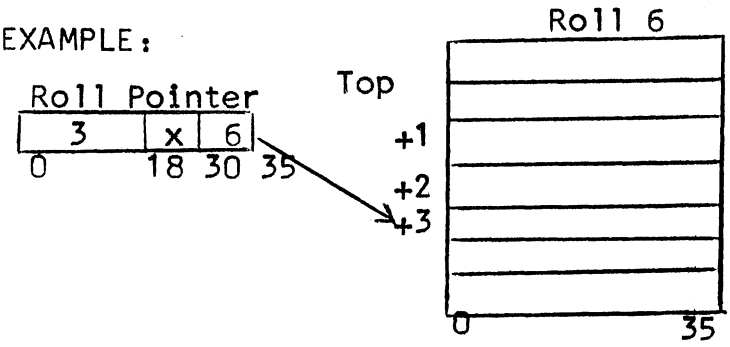


Figure 3 shows the relationships between entries in the ANCHOR, TOP, and BOTTOM tables and the roll locations they describe.

a. Roll Pointers

A roll pointer is a word that contains an offset P in bits 0-17 and a roll number N in bits 30-35. It points to the location that is P words after the top of roll N.

EXAMPLE:



If Y is a location containing a roll pointer, then RP(Y) is defined as the location to which the roll pointer points; i.e., RP(Y) = P+C(TOP+N) 0-17.

Roll pointers are useful, since the distance from the top of a roll is constant, even though the roll may have been moved.

While the contents of any location may be used as a roll pointer, each roll has a particular roll pointer assigned to it in the ROLPTR table, located in the data segment. For example, ROLPTR+5 is the roll pointer assigned to roll 5. Many pops use the ROLPTR table to record the current position on a roll.

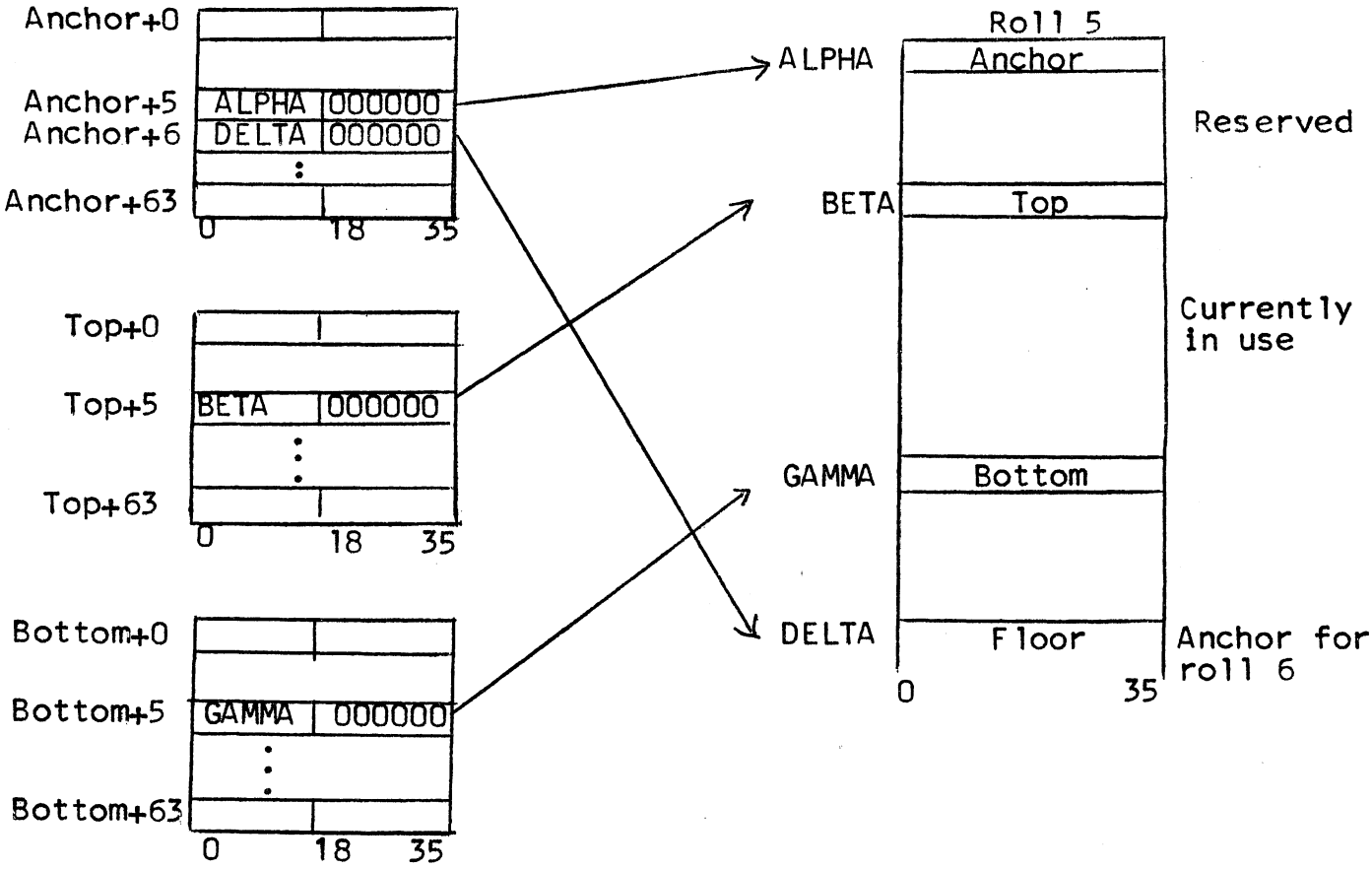


Figure 3: Illustration of Rolls and their Corresponding Tables

b. Guess Table

The GUESS table in the data segment specifies the initial allocation of space for rolls and the extra amount of storage to allocate in case this initial storage is exceeded. There are 64 entries in the GUESS table. The following chart illustrates how the interpreter uses the GUESS table to allocate space for roll N: (I and J are unsigned integers.)

C (GUESS+N) 0-17	18-35	Number of words to allocate: On initiation of pops procedure	On first ref. to roll	On subsequent allocations
I	J	J	0	I
I	-J	J/100	0	(I*J)/10000
-I	J	1	J	I
-I	-J	1	J/100	(I*J)/10000

The numbers in the last two columns represent minimum guesses. If more words are needed than are specified in these columns, the interpreter will allocate the required number of words.

The upper half of OPNERS, a one-word register in the data segment, specifies the number of rolls to be opened. The interpreter allocates space only to these rolls. If OPNERS specifies N rolls, then the interpreter allocates space to rolls 0 through N-1 (inclusive).

c. Composition of Rolls

1) General

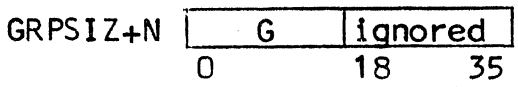
A roll may consist of single words, groups, or plexes (See Paragraph H.4 for a discussion of plexes.)

2) Groups

A group consists of 0, 1, or a consecutive series of words; e.g., a 10-word group consists of 10 consecutive words on a roll.

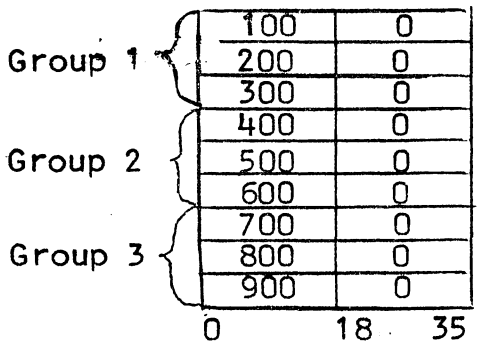
There are two types of groups: fixed-size groups and variable-size groups. A roll containing fixed-size groups may not contain variable-sized groups, and vice-versa.

The GRPSIZ table in the data segment indicates the type of groups on each roll. This table contains 64 entries, one for each roll:

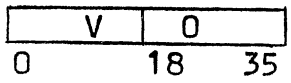


If G≠0, roll N consists of G-word fixed-size groups. These groups are stored contiguously.

EXAMPLE:

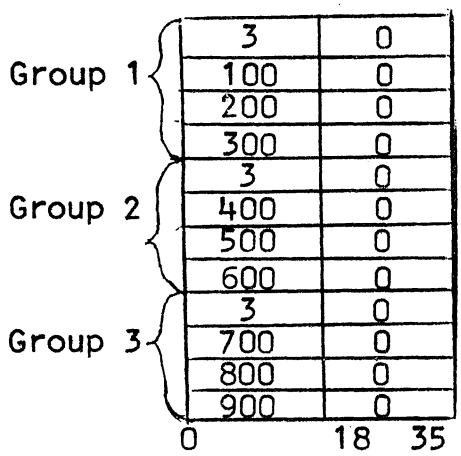


If G = 0, then roll N consists of variable-size groups. Each variable-size group is preceded by a variable-size word (VSW) with the following format:



where V = the number of words in the group; i.e., the number of words following the VSW. Variable-size groups are also stored contiguously.

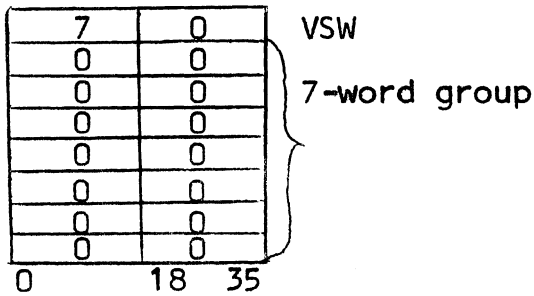
EXAMPLE:



Certain pops use the upper half of VARSIZ, a one-word register in the data segment, to determine the number of words in a variable-size group (excluding the VSW).

d. Special Rolls

Roll 0 must be used for the symbol table. The first eight words on roll 0 have the following format:



There are two reasons for this:

1. To avoid having a word in roll 0 at an offset of 0 from the top

2. To provide an additional area for starting a type-2 thread for a search (See Chapter 2, Paragraph R.)

The symbol table starts at location 8 in roll 0.

The user must reserve the following rolls for their special purposes, only if he uses the pops that refer to these rolls:

Roll #	Purpose	Used By
1	Error roll	EROR, ERRCC, ERRLC, ERRP, PRNT, PRNTC
2	Save roll	PRES, PSAV, PDES
3	Fact roll	FACT, TYMF, TYMT
4	Swip roll	SWIP
N-2* where N = C(OPNERS) 0-17	Binary roll	WBIN, RWND
N-1 and N-2 where N = C(OPNERS) 0-17	Spill rolls	RWND, DNX, DNG, DLOAD
M where M = C(BINREL+1) 0-17 (See Paragraph Y.2 in MSPM BZ.7.02.)	Relbit roll	WBIN

The uses of these rolls are covered in the descriptions of the pops.

---

*The interpreter actually uses C(BINREL) 0-17 or the operand of the WBIN pop to determine the number of the binary roll. N-2 is the binary roll in a two-pass assembler or compiler.

3. Buffers

Three buffers appear below the last roll in the following order:

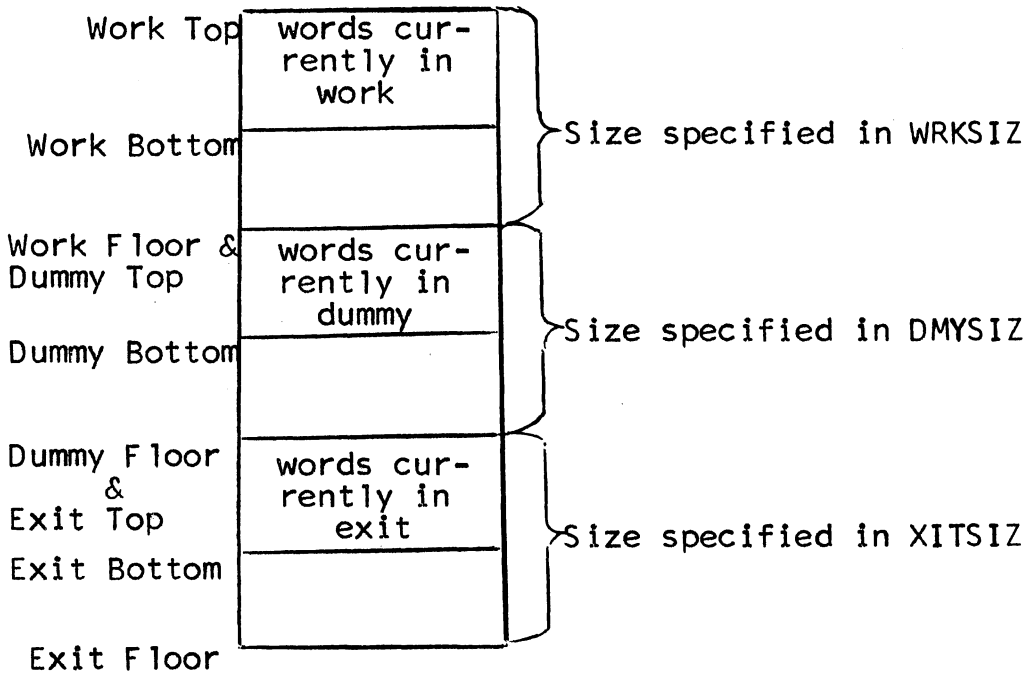
WRKBUF -- work buffer or work

DMYBUF -- dummy buffer or dummy

XITBUF -- exit buffer or exit

These buffers are fixed-size stacks. Their sizes are specified in the lower halves of three one-word registers in the data segment: WRKSIZ, DMYSIZ, and XITSIZ.

The buffers may be pictured as follows:



Three 18-bit counters point to the current positions in the three buffers: WRKCTR, DMYCTR, and XITCTR. These counters are simulated by GE-645 index registers 3, 5, and 4, respectively. WRKCTR and DMYCTR point one word above the bottom of work and dummy, respectively. XITCTR points two words above the bottom of exit.

a. Work Buffer

The work buffer is a push-down accumulator, used for storing data.

New words may be loaded into work one word at a time. In this case, the interpreter bumps the bottom of work by 1 word, adds 1 to C(WRKCTR) and moves the appropriate data into word 1.

EXAMPLE:

load(Y) loads C(Y) into word 1

A word may be loaded into current work. In this case, the interpreter does not adjust the bottom of work.

EXAMPLE:

cload(Y) loads C(Y) into current work

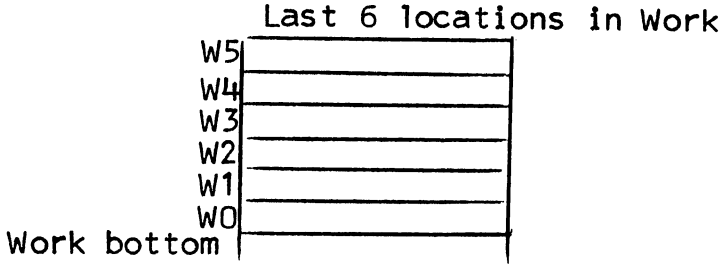
Any number of words (including 0) may be removed from work. This is called pruning work. In this case, the interpreter subtracts the appropriate number of words from WRKCTR and adjusts the bottom of work accordingly.

EXAMPLE:

prw(Y) prunes C(Y) 0-17 words from work

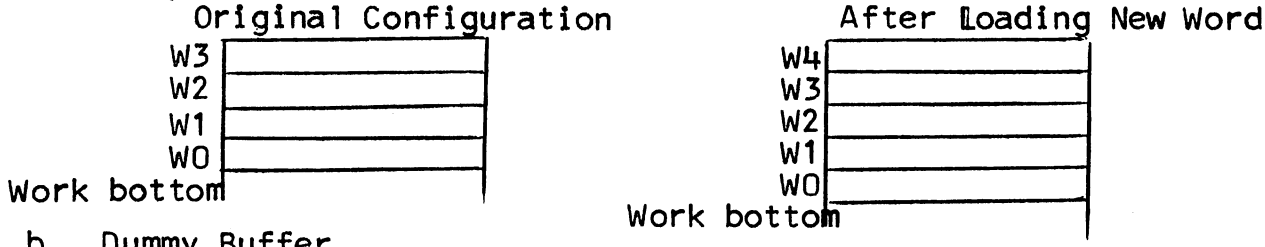
Initially, there is one word in the work stack. This word should never be pruned. However, the user may load data into this word. The term work size denotes the number of words appended to work; e.g., if work contains two words, the work size is one. N-1 words may be appended to work, where N = C(WRKSIZ) 18-35.

W0 is the symbolic location for current work. The five locations preceding W0 (if present) are referred to respectively as W1, W2, W3, W4, and W5:



These locations are adjusted appropriately, whenever the bottom of work changes.

EXAMPLE:



b. Dummy Buffer

The dummy buffer provides a method of indirect addressing. It is used to store addresses rather than data. The difference may be illustrated as follows:

stor(w1) stores current work into previous work

stor(d1) stores current work into the location specified in the upper half of the previous dummy

New words may be loaded into dummy one word at a time. In this case, the interpreter bumps the bottom of dummy by 1 word, adds 1 to C(DMYCTR) and moves the appropriate data into word 1.

A word may not be loaded into current dummy.

The dummy may be pruned by any number of words (including 0). In this case, the interpreter subtracts the appropriate number of words from DMYCTR and adjusts the bottom of dummy accordingly.

EXAMPLE:

prd(Y) prunes C(Y) 0-17 words from dummy

Initially, there is one word in the dummy stack. This word should never be pruned. The user should not load data into this word. The term dummy size denotes the number of words appended to dummy; e.g., if dummy contains two words, the dummy size is one. N-1 words may be appended to dummy, where N = C(DMYSIZ) 18-35.

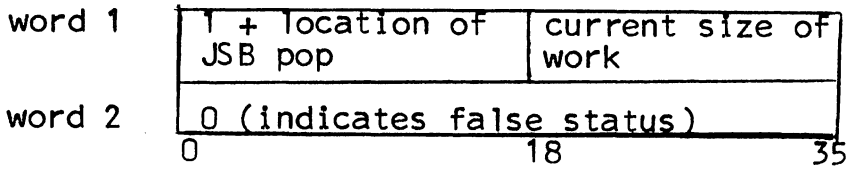
D0 is the symbolic location for current dummy. The five locations preceding D0 (if present) are referred to respectively as D1, D2, D3, D4, and D5.

C. Exit Buffer

The interpreter adjusts the exit buffer, upon entering and leaving a subroutine. This buffer contains two-word entries.

The JSB pop is used to transfer to a subroutine. Before entering the subroutine, the interpreter adjusts the exit buffer as follows:

1. It adds 2 to XITCTR
2. It bumps the bottom of the exit buffer by 2, and puts the following information in words 1 and 2:



Word 2 is the current true/false indicator for the pseudo-computer. If C(word 2) = 0, the current status is false; if C(word 2) ≠ 0, the current status for the pseudo-computer is true. This word is set by pops in the procedure segment; e.g., the search pops (See Chapter 2, Paragraph R).

Upon leaving a subroutine, the interpreter prunes two words from the exit buffer.

The user cannot load words into the exit buffer. However, he may prune the buffer by any number of word pairs (including 0).

EXAMPLE:

pre(Y) prunes exit by C(Y) 0-17 pairs of words

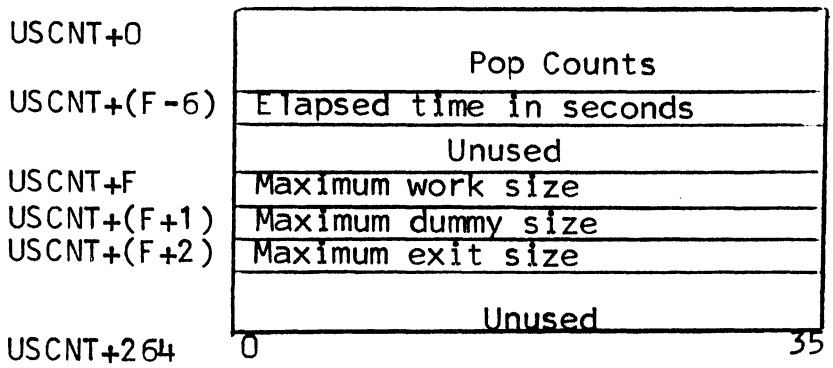
Initially, there are two words in the exit stack. The interpreter sets C(word 2) = 0; C(word 1) is indeterminate. These words should never be pruned. The term exit size denotes the number of words appended to exit; e.g., if exit contains four words, the exit size is two. N-2 words may be appended to exit, where N = C(XITSIZ) 18-35.

F. PRODUCTION AND DEBUG VERSIONS OF THE INTERPRETER SEGMENT

There are two versions of the interpreter segment: production and debug. These can both run with any data or procedure segments. The debug version is 50% slower than the production version. However, it provides important statistics.

The following tables (located in the data segment) are set during the debug version and are dumped upon termination of a procedure (they are ignored during the production version):

USCNT -- 264 word table. The first F-6 locations correspond to each of the F-6 pops, where F is the value of the false tag. Each of these locations records the number of times the corresponding pop was executed. The functions of the other locations are illustrated on the following chart:



RSIZE -- Table containing one word for each roll. The upper half of each entry is set to the maximum count of each roll (maximum number of words from top to bottom). The lower halves are ignored.

MTEST is a one-word register in the data segment. If MTEST is set to 0, the debug version records statistics about each roll movement and dumps this information upon termination of a procedure:

- Number of roll moved
- Number of words appended to roll
- Number of words originally required by roll
- *Method* of obtaining words
- Elapsed time to perform move (in seconds)

(For each roll movement)

The debug version also dumps the following general statistics:

- Total number of pops executed
- Elapsed time in seconds
- Number of roll moves
- Elapsed time in doing roll moves (in seconds)

G. DATA REPRESENTATION

1. Number Representation

The pops interpreter uses standard GE-645 two's complement arithmetic for addresses, fixed-point numbers, and floating-point numbers; e.g., address -1 is equivalent to octal 777777. (See GE-635 Reference Manual for complete description). Chapter 2, Paragraph 0, covers the representation of fixed-point and floating-point numbers in the pseudo-computer.

2. Text Representation

a. Character Set

Most text for the pseudo-computer is written in the ASCII character set (See Chapter 2, Paragraph Y.1 for an exception.) Figure 4 shows the 128 ASCII characters and their corresponding octal codes, 000-177. The interpreter also uses two special characters internally.

octal 200 - Skip character

octal 201 - End-of-file character

Each of these 130 characters is represented in a 9-bit field; e.g., A (octal 101) is represented as 001000001. Therefore, a word in the pseudo-computer may contain up to four characters.

b. TRANS Table

The TRANS table in the data segment contains one word for each of the 130 characters mentioned above. Bits 9-17 of each word contain a copy of the ASCII or special character. This field coincides with the offset of the word from the first word in the TRANS table; i.e., C(TRANS+N) 9-17 = N. The remaining bits

are called keys and may be used to store any information pertaining to the characters. Each bit may correspond to a special property of one or more characters; e.g., alphabetic or numeric. The TRANS table is illustrated below:

TRANS+000	K	000	K
+001	e	001	e
•	y	•	y
•	s	•	s
+177		177	
+200	777200		000000
+201	777201		000000

The user provides information for TRANS through TRANS+177 8. The interpreter sets TRANS+200 8 and TRANS+201 8.

ASCII Character Set on Multics

	0	1	2	3	4	5	6	7
000								BEL
010	BS	HT	NL	VT	NP		RRS	BRS
020			HLF		HLR			
030								
040	Space	!	"	#	$	%	&	'
050	(	)	*	+	,	-	.	/
060	0	1	2	3	4	5	6	7
070	8	9	:	;	<	=	>	?
100	@	A	B	C	D	E	F	G
110	H	I	J	K	L	M	N	O
120	P	Q	R	S	T	U	V	W
130	X	Y	Z	[	\	]	^	_
140	`	a	b	c	d	e	f	g
150	h	i	j	k	l	m	n	o
160	p	q	r	s	t	u	v	w
170	x	y	z	{	¦	}	~	

Multics Definitions:

NL	New Line (carriage return and line feed)
HLF	Half-Line Forward Feed
HLR	Half-Line Reverse Feed
RRS	Red Ribbon Shift
BRS	Black Ribbon Shift
NP	New Page (carriage return and form feed)

Figure 4

H. STRINGS

The interpreter may combine ASCII characters into three types of strings:

Type 1 -- Counted

Type 2 -- Non-counted

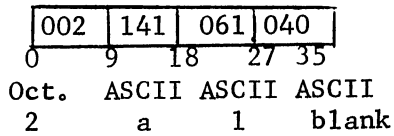
Type 3 -- Special

1. Type-1 Strings

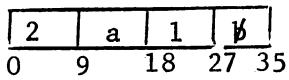
The first character in a type-1 string indicates the number of characters (0-511) that follow in the string. If the last word in a type-1 string is not full, the remainder of the word is filled with ASCII blanks; these blanks are not part of the string.

EXAMPLE:

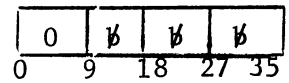
Type-1 String "a1"



Future examples use the following abbreviation:



The null type-1 string is shown below:

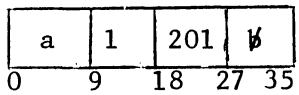


2. Type-2 Strings

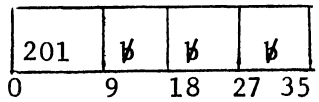
A type-2 string has no initial counter character; instead, the last character in the string is the end-of-file character, 201. If the last word is not full, the remainder of the word is filled with ASCII blanks; these blanks are not part of the string. A type-2 string may contain 0-512 characters, excluding the end-of-file character.

EXAMPLE:

Type-2 String "a1"



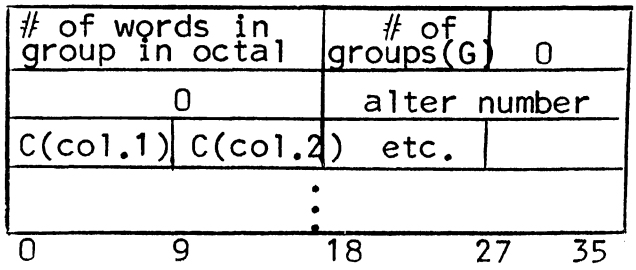
The null type-2 string is shown below:



3. Type-3 Strings

A type-3 string represents a complete FORTRAN source statement. It contains one variable-size group for each line in the statement. The first two words of each group are control words. The remaining words represent the line (4 columns per word). If the last word in a group is not full, the remainder of the word is filled with skip characters (octal 200). The maximum number of characters in a group is 163.

Each group in the type-3 string has the following format:



Each line of a FORTRAN source statement ends with a new-line character (NL). In forming a type-3 string from console input, the interpreter interprets the new-line character as follows:

1. The sequence %(NL) means ignore both characters, and consider the next line as continuation.

2. (NL) preceded by a character other than % means ignore (NL), and terminate the type-3 string.

The % and (NL) characters are not included as part of the string.

Comments may not appear in type-3 strings. Thus, the user must obey the following rules:

1. A comment may not appear in the middle of a FORTRAN statement: a comment may not follow %(NL) in a FORTRAN statement.

2. A comment may occupy only one line

3. A comment may begin only with a star; the procedure segment is responsible for recognizing the star.

Figure 5 shows a FORTRAN statement and its representation as a type-3 string.

FORTRAN Statement	Alter Number
12345 a=b%	30
+c%	31
+d	32

Representation as type-3 string

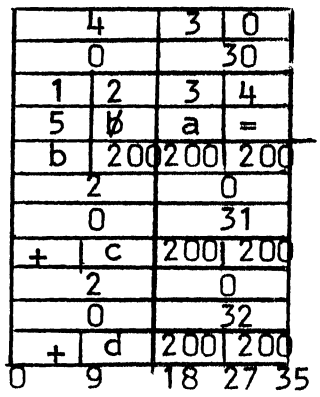
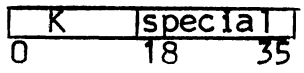


Figure 5. Typical Type-3 String

4. Plexes

The interpreter may convert a type-1 string to a plex, by appending a word:



(Bits 18-35 have special meaning to the interpreter.)

where K is the number of words in the string. K is derived as follows:

Let C = the first character in the type-1 string

K = C/4 + 1 (ignoring any remainder)

5. SYMBUF

SYMBUF is a region in the data segment used for forming type-1 or type-2 strings. The maximum capacity of SYMBUF is 511 characters for a type-1 string and 512 characters for a type-2 string. The interpreter will abort the pops procedure if either limit is exceeded. The best size for SYMBUF is 129 words, since this is one more word than necessary for the largest string. SYMBUF is preceded by two words, SYMCNT, and SYMCNT+1. These words have the following format:

C(SYMCNT) 0-17 -- Number of words in symbol

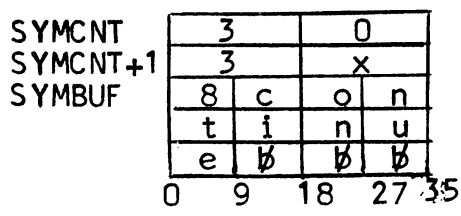
C(SYMCNT) 18-35 -- Data used by the interpreter

C(SYMCNT+1) -- For type-1 string, plex word

For type-2 string, upper half contains count of number of characters in the string (excluding the end-of-file character), lower half is ignored

EXAMPLE:

A plex in SYMBUF



I. NOTATION USED IN THIS MANUAL

1. Coding Examples

FL/1 syntax is used in all coding examples.

2. Capitalization

The following names are capitalized:

1. Names of pops, variables, and registers, outside of coding examples

2. Generic names in coding examples

Otherwise, all names are written in lower case.

Examples:

```
NXCH pop
MRKER register
add(Y), where Y signifies "any operand"
add(alpha), where ALPHA is the name of a particular operand
```

3. Number Representation

Numbers are represented in octal, unless otherwise indicated. All number fields are right-justified; e.g., 6 and 000006 are equivalent in an 18-bit field. Since FL/1 syntax accepts decimal mode as the normal default mode, small numbers (0-7) are used in the examples whenever possible. Symbolic addresses are used in some examples in place of large octal addresses, for simplicity.

4. Terms

work size - Number of words appended to work stack. Initially, work contains one word; this word is not included in the work size.

dummy size - Number of words appended to dummy stack. Initially, dummy contains one word; this word is not included in the dummy size.

exit size - Number of words appended to exit stack. Initially, the exit stack contains two words; these words are not included in the exit size.

prune - To remove zero or more words from a stack or roll

word k - The kth word in a group of contiguous words; i.e., in a group of contiguous words, the first word is word 1 (unless otherwise indicated)

bumping bottom - Moving bottom of a roll or stack down by a specified number of words, to allocate more words for current use.

key - Bit describing some property of a character or string.

counting a string - Computing the number of words and the number of characters in a string.

file - One or more consecutive words in the data segment, used by a pop for information. The operand of a pop using a file is the location of the first word of the file.

loctr - FL/1 term for the current location (like * in many other assembly languages)

alter number - Number associated with each line of a source procedure, starting with 1 for the first line.

go to next pop - Interpret next pop

5. Symbols

Y - Operand of the current pop

Z - Operand of the next pop

C(Y) - Contents of bits 0-35 of operand Y

DP(Y) - C(Y,Y+1) in double precision, where Y must be an even address

C(Y) k - Contents of bit k of operand Y, where k = 0,1,2,..., or 35

C(Y) 0-17 - Contents of upper half of operand Y

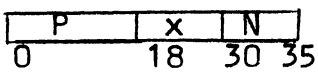
C(Y) 18-35 - Contents of lower half of operand Y

N - A number. Unless otherwise specified, N is a roll number.

M - Roll number. Used when description refers to two rolls

x in diagram - Ignore (unless otherwise specified)

RP(Y) - Y is a location containing a roll pointer:



, where P is an offset from the top of roll N

RP(Y) = location to which roll pointer points; i.e., P + C(TOP+N) 0-17

VSW - Variable size word

.logical. - A logical operation; e.g., .and. = AND

t - True tag

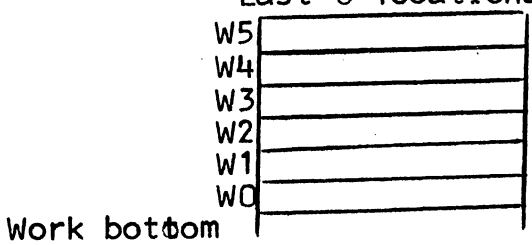
f - False tag

ϕ - Blank

(NL) - New line character

W0 - Current work

W1, W2, W3, W4, and W5 - The five locations preceding W0:



D0 - Current dummy

D1, D2, D3, D4, and D5 - The five locations preceding current dummy

A0, A1, A2, A3, A4, A5 - See Chapter 2, Paragraph G