SHARE PROGRAM LIBRARY AGENCY

PROGRAM NUMBER

068004

University of Miami

1365 MEMORIAL DRIVE - CORAL GABLES, FLORIDA
(305) - 284-6257

HARE PROGRAM LIBRARY SUBMITTAL FORM

SHARE Program Library Agency
Triangle Universities Computation Center
P. O. Box 12076
Research Triangle Park, N. C. 27709

CONTROL NUMBER:

his form should be completed and submitted with the program package to the SHARE Program ibrary Agency at the address shown above. Standards and instructions for submitting programs re in the "SHARE Program Library Standards Manual".

1) Program Number (to be filled in by SPLA). 360D-06.8.004

2) System Type (machine) . . . . . IBM System 360 (OS)

3) Search Key . . . . . IN-CORE STACK MANIPULATION FOR OS/360 ASSEMBLER LANGUAGE PROGRAMS

4) Programming Language . . . . . IBM S/360 OS Assembler

5) Author's Name and Address . . . . . Roger J Chetwynd

6) Direct Inquiries to Name and Address . . . . . Technical Assistance Not Currently Available
(if different than Author)

7) Title of Program . . . . . In-core stack manipulation package for use in an OS/360 Assembler Language environment

8) Submitter's Installation Membership Code. . . . . U Y

9) Submitter's Own Program Identification and Suffix (optional). . . . .

10) Primary Subject Code. 06 8

11) Operating or Monitor System Required OS/360

12) New or Revision Code (if revision, show prior Program Number in Item 1). . . . . new

13) Year Completed 1971

14) Date of Submittal. . . . . 2 August 1973

15) Documentation (number of original pages submitted) 24

16) Abstract (should contain sufficient information for a reader to determine the value of the program). Listed on the reverse side of this form are subjects which may serve as a guide for a descriptive abstract.

SHARE PROGRAM LIBRARY SUBMITTAL FORM

Subject Guide:

a. Purpose
b. Programming Language used
c. Version and modification level or release number
d. Field of application
e. Type of routine (main program, subroutine, etc.)
f. Specific description of machine requirements

ABSTRACT WSUSTACK is a reenterable subprogram which dynamically creates and maintains core-resident stacks in an OS/360 Assembler Language environment. It may be assembled and used on an IBM S/360 under any version of OS since release 14.

Stack lengths are limited only by the main storage available to the task, the size of the stack node may vary from 1 to 256 bytes and is constant for a given stack, and any number of stacks may be maintained concurrently.

As one of the design objectives was optimization of storage and execution time, the calling sequences are non-standard. Accordingly a companion set of macro instructions is provided to generate the proper calling sequences. The functions available, each of which is called by a corresponding macro instruction, are: Allocate and initialize stack, Delete stack, Stack a node, Unstack a node, Reset stack to the empty condition, Index stack (locate n'th node), Search stack (locate a node satisfying given conditions).

(Please attach additional pages if necessary). . . . . Total pages attached 0

Permission to Publish

"I hereby give the SHARE Program Library Agency permission to reprint, reproduce, and distribute this program."

(17) Signature of Submitter and Date R.J. Child 2 August 1973

(18) Signature of Installation Addressee Kathleen Edwards (WY)

Program 360D-06.8.004 NO LABEL

TAPE DISTRIBUTION:

80 character records blocked 8000.
Density as ordered.

CONTENTS:

File 1	WSTK
File 2	STML

WSUSTACK

A utility for handling unlimited-length core-resident stacks in an OS/360 Assembler Language environment.

Programming and Documentation by:

Roger Chetwynd
Computer Science Department
Washington State University
Pullman, Washington 99163

Acknowledgement:

The author is indebted to Washington State University for providing the computing facilities on which this program was developed.

Contents:

I - Introduction

WSUSTACK consists of a single load module with the same name which creates and maintains core-resident stacks by means of calls generated by a companion set of macro instructions. The stacks are limited in length only by the main storage available to the task. The size of the stack node may vary from 1 byte to 256 bytes, and is constant for a given stack. Any number of stacks may be maintained concurrently.

The stacks are maintained internally in the form of a stack descriptor and a number of stack sections, each section having space for a predetermined number of nodes plus four words of linkage and other information. Only as many sections are kept as will contain the current number of nodes; an overflow of the current section will cause a new section to be obtained, and an unstack of the one node remaining in a section will cause that section to be released. GETMAIN and FREEMAIN type R are used, thus an attempt to obtain a new stack section when no more main storage is available will cause the task to be abnormally terminated with a system completion code of 80A. The first node in each section lies on a doubleword boundary, thus node alignment is consistent with node size.

The code of the module WSUSTACK is reentrant, thus no problem of attributes will be encountered in linking WSUSTACK to other modules.

Standard OS linkage is assumed; the calling program must supply the address of an 18-word save area in register 13.

The module and macros have been designed together for efficiency of both storage and time in the management of the stacks, thus it is imperative that the module be accessed only by the macro instructions.

The functions available are summarized in the following table, and are described in detail in section II.

Funetion	Macro Instruction	Registers Altered[1,2]
Allocate and Initialize Stack	GTSTK	14,15,0,1
Delete Stack	DLSTK	14,15,1
Stack a Node	STACK	14,15,0,1
Unstack a Node	UNSTK	14,15,0,1
Reset Stack	CLSTK	14,15,0,1
Index Stack	IXSTK	14,15,0,1,cc
Search Stack	SHSTK	14,15,0,1,FPR0,cc[3]

Note 1: All other registers are unchanged by execution of the macro instruction.

Note 2: The condition code is unchanged by execution of the macro instruction, with the exceptions of IXSTK and SHSTK for which the condition code is set to indicate termination conditions.

Note 3: General Register 0 or Floating Point Register 0 are altered depending on the form of the macro·instruction; see detailed description in part (h).

II - The Macro Instruction Set

(a) General:

All operands of the macro instructions are optional, with default values described under the appropriate instruction.

The operands "stackadr" and "vconadr" which are common to many of the macro instructions are described here:

stackadr - is either: an absolute or relocatable expression specifying the address of a fullword whose low-order three bytes contain the address of the stack descriptor,
- or: an absolute expression enclosed in parentheses specifying one of the general registers 0 through 15 whose low-order three bytes contain the address of the stack descriptor.
- If omitted, the address of the stack descriptor is assumed to be contained in register 1.

vconadr - is either: an absolute or relocatable expression specifying the address of a fullword whose low-order three bytes contain the external address of WSUSTACK,
- or: an absolute expression enclosed in parentheses specifying a general register other than 1 whose low-order three bytes contain the address of WSUSTACK.
- If omitted, the literal =V(WSUSTACK) is generated.

(b) The GTSTK Macro Instruction:

The GTSTK macro instruction causes allocation and initialization of a stack with specified attributes.

Positional operands (in order):

vconadr - see section II(a) - register 0 must not be specified.

Keyword operands:

SIZE= - a number from 1 through 256, specifying the size in bytes of the stack node.
- If omitted, SIZE=4 is assumed.

NODES= - either: a number greater than zero specifying the maximum number of nodes in a stack section (see note 1),
- or: written as NODES=(0), indicating that register 0 contains the proper parameter information as described under the MF=E operand. In this case the SP= operand is ignored.
- If omitted, NODES=60 is assumed.

SP= - a number from 0 through 127, specifying the subpool from which the stack sections obtained by GETMAIN are to be taken.
- If omitted, SP=0 is assumed.
- If NODES=(0) is coded, the SP= operand is ignored.

MF= - specifies the format of the macro expansion.
Standard form: parameter omitted.
If the values of SIZE, NODES, and SP require it, an in-line parameter list is generated along with the executable code.
List form: MF=L
The parameter list alone is generated, which is intended to be used as a remote parameter list in conjunction with the execute form of the GTSTK macro instruction.
Execute form: two forms, MF=(E,adrs) and MF=E
(i) - MF=(E,adrs) - In this case, "adrs" is either: an absolute or relocatable expression specifying the address of a remote parameter list,
or: an absolute expression enclosed in parentheses specifying a general register from 1 through 15 containing the address of a remote parameter list.
(ii) - MF=E - In this case it is assumed that registers 0 and 1 already contain the proper parameter information and the SIZE, NODES, and SP operands are ignored. The registers must have the following form:
Reg 0 - bits 0-7: subpool number
bits 8-31: SIZE*NODES+40
Reg 1 - bits 0-31: SIZE parameter

The GTSTK macro instruction generates a call to an entry point in the module WSUSTACK, which obtains core for and initializes the 24-byte stack descriptor and the first stack section. The first node in the stack is designated the null node and is cleared to zero. If this entry point in WSUSTACK receives either in register 1 a number not in the range 1 to 256, or in register 0 an amount of storage greater than 32791 or a subpool number greater than 127, the task will be abnormally terminated with a user code of 104 and a core dump.

On return, the registers and condition code are as follows:

Reg 14 - address of instruction following macro expansion
Reg 15 - address of the cleared null node.
Reg 0 - index of the null node (=1).
Reg 1 - address of the stack descriptor for the new stack.
Reg 2-13 - as before execution of the macro.
cc - as before execution of the macro.

Note 1: The value of NODES should be chosen so as to utilize both storage and time most efficiently. If it is too small, the many resulting stack section overflows will consume execution time; if it is too large, there will be considerable wasted space. It should be borne in mind that the size of each stack section is (NODES*SIZE+16) bytes and that this quantity must not exceed 32767.

(c) The DLSTK Macro Instruction:

The DLSTK macro instruction releases the main storage occupied by a stack initially obtained by use of the GTSTK macro instruction.

Positional operands (in order):

stackadr - see section II(a).

vconadr - see section II(a).

Keyword operands:

SP= - written as either: SP=X, indicating that the subpool assigned to the specified stack is owned exclusively by that stack and thus the stack can be released much more quickly by a FREEMAIN of the entire subpool,

- or: SP=S, indicating that the subpool assigned to the specified stack is shared with other requests, preventing the stack from being released by a FREEMAIN of the subpool.
- If omitted, SP=S is assumed.
- The operand is meaningless and should be omitted in the case of subpool zero.

The DLSTK macro instruction generates a call to an entry point in the module WSUSTACK, which releases all main storage obtained for stack sections and the stack descriptor for the specified stack.

On return, the registers and condition code are as follows:

Reg 14 - address of instruction following macro expansion.
Reg 15 - not useful (an address internal to WSUSTACK).
Reg 0 - as before execution of the macro.
Reg 1 - not useful (the address of the deleted stack descriptor).
Reg 2-13 - as before execution of the macro.
cc - as before execution of the macro.

Programming note: A FREEMAIN of an entire subpool will also delete any stacks assigned to that subpool.

(d) The STACK Macro Instruction:

The STACK macro instruction adds a new node to the top of the specified stack.

Positional operands (in order):

stackadr - see section II(a).

vconadr - see section II(a).

Keyword operands: none.

The STACK macro instruction generates a call to an entry point in the module WSUSTACK, which adds a new node to the top of the stack identified by the first parameter. If a stack section overflows, a new one is obtained and initialized. The new node is not cleared to zero or initialized in any way.

On return, the registers and condition code are as follows:

Reg 14 - address of instruction following macro expansion.
Reg 15 - address of new top node.
Reg 0 - index of new top node, = number of nodes in stack.
Reg 1 - address of the stack descriptor.
Reg 2-13 - as before execution of the macro.
cc - as before execution of the macro.

Programming note: Obtain a new node at the top of the stack before transferring any data to it.

(e) The UNSTK Macro Instruction:

The UNSTK macro instruction removes a node from the top of the specified stack.

Positional operands (in order):

stackadr - see section II(a).

vconadr - see section II(a).

Keyword operands: none.

The UNSTK macro instruction generates a call to an entry point in the module WSUSTACK, which removes the node at the top of the stack identified by the first parameter. If a stack section becomes empty, its storage is released. An attempt to unstack the null node will result in abnormal termination of the current task with a user code of 100 and a core dump.

On return, the registers and condition code are as follows:

Reg 14 - address of instruction following macro expansion.
Reg 15 - address of node appearing at top of stack after the unstack operation.
Reg 0 - index of top node, = number of nodes in stack, after the unstack operation..
Reg 1 - address of the stack descriptor.
Reg 2-13 - as before execution of the macro.
cc - as before execution of the macro.

Programming note: Retrieve the contents of the top node before unstacking it.

(f) The CLSTK Macro Instruction:

The CLSTK macro instruction resets the specified stack to the null condition.

Positional operands (in order):

stackadr - see section II(a).

vconadr - see section II(a).

Keyword operands: none.

The CLSTK macro instruction generates a call to an entry point in the module WSUSTACK, which releases storage occupied by all stack sections except the first, resets the top-of-stack pointer to point to the null node, and clears the null node to zero. The stack then appears just as it did immediately after being created with the GTSTK macro instruction.

On return, the registers and condition code are as follows:

Reg 14 - address of instruction following macro expansion.
Reg 15 - address of the cleared null node.
Reg 0 - index of the null node (=1).
Reg 1 - address of the stack descriptor.
Reg 2-13 - as before execution of the macro.
cc - as before execution of the macro.

(g) The IXSTK Macro Instruction:

The IXSTK macro instruction finds the n'th node in the specified stack for a parameter n.

Positional operands (in order):

stackadr - see section II(a).

index - is either: an absolute or relocatable expression with a displacement value less than 4096 specifying the index n,
- or: an absolute expression enclosed in parentheses specifying a general register other than 1 which contains the index n.
- If omitted or zero, the address and index of the current top-of-stack is returned.

vconadr - see section II(a). Register 0 must not be specified. This parameter is ignored if the second parameter is omitted.

Keyword operands: none.

The IXSTK macro instruction generates a call to an entry point in the module WSUSTACK, which uses the input parameter n to index the stack and returns the address of the n'th node. When the second parameter is missing, however, no call is generated; the address and index of the current top node are returned.

On return, the registers and condition code are as follows:

Case (i) - second parameter present and non-zero:
Reg 14 - address of instruction following macro expansion.
Reg 15 - successful index: address of indexed node.
- unsuccessful index: address of WSUSTACK.
Reg 0 - index of node.
Reg 1 - address of stack descriptor.
Reg 2-13 - as before execution of the macro.
cc - successful index: 0.
- unsuccessful index (index out of range): 2.

Case (ii) - second parameter present and zero:
Reg 14 - address of instruction following macro expansion.
Reg 15 - address of node at top of stack.
Reg 0 - index of top node, = number of nodes in stack.
Reg 1 - address of the stack descriptor.
Reg 2-13 - as before execution of the macro.
cc - 1.

Case (iii) - second parameter omitted:
Reg 1-14 - as before execution of the macro.
Reg 15 - address of current top-of-stack.
Reg 0 - index of current top-of-stack.
cc - as before execution of the macro.

Note: The statement

```
IXSTK STACK,0
```

returns the same information as the statement

```
IXSTK STACK
```

However, omitting the second parameter destroys fewer registers and omits the call to WSUSTACK, thus the second statement is recommended when the address and index of the current top-of-stack is desired.

(h) - The SHSTK Macro Instruction:

The SHSTK macro instruction causes a search of the nodes in the specified stack for a match under given conditions with a given key.

Positional operands (in order):

stackadr - see section II(a).

vconadr - see section II(a). Register 0 must not be specified in the cases of MODE=CLC,C,CL,CH.

Keyword operands:

MODE= - the instruction and type of key to be used in the search.

MODE=CLC - the key is a sequence of bytes in core.
MODE=CLI - the key is a single byte of immediate data.
MODE=C - the key is a 4-byte fixed-point signed quantity.
MODE=CL - the key is a 4-byte fixed-point unsigned quantity.
MODE=CH - the key is a 2-byte fixed-point signed quantity.
MODE=CE - the key is a single-precision (4-byte) floating-point number.
MODE=CD - the key is a double-precision (8-byte) floating-point number.
Note: Compare Decimal (CP) is not supported at present.

KEY= - the location or value of the key.

For MODE=CLC, the operand is
either: an absolute or relocatable expression specifying the address of the beginning of the key,
or: an absolute expression enclosed in parentheses specifying a register other than 1 which contains the address of the key.
If omitted, the address of the key is assumed to be in register 0.

For MODE=CLI, the operand is
either: an absolute expression specifying the 8-bit quantity to be used as key,
or: an absolute expression enclosed in parentheses specifying a register other than 1 whose low-order byte contains the key.
If omitted, the key is assumed to be in the low-order byte of register 0.

For all other modes, the operand is
either: an absolute or relocatable expression specifying the address of the properly aligned storage location containing the key,
or: an absolute expression enclosed in parentheses specifying a register other than 1 (floating-point register in the cases of MODE=CE or CD) which contains the key. Note; If MODE=CH, the halfword quantity in the register must have its sign bit properly extended.
If omitted, the key is assumed to be in register 0 (floating-point reg 0 for MODE=CE or CD).

COND= - the condition of compare for successful search. The search is successful if the node satisfies the given relation to the key. Permitted relations are:

EQ - search on equal,
NE - search on not equal,
GT - search on greater than,
LT - search on less than,
GE - search on greater than or equal to,
LE - search on less than or equal to.

If omitted, COND=EQ is assumed.

DISPL= - an absolute expression specifying the displacement from the beginning of the node to the beginning of the field to be examined. If omitted, DISPL=0 is assumed.

LEN= - an absolute expression specifying the length of the key. This operand is examined only in the case of MODE=CLC; in all other cases the length of the key is implied by the mode chosen (1 for CLI, 2 for CH, 8 for CD, 4 for all others). If omitted in the case of CLC, LEN defaults to the size of the stack node (the SIZE= parameter in the associated GTSTK macro instruction).

The SHSTK macro instruction generates a call to an entry point in the module WSUSTACK, which performs the requested compare node by node from the top of the stack down until the conditions are satisfied or the null node is reached. If the parameters received by this entry point are such that an alignment error may occur or a compare is requested on a field outside the node, the task will be abnormally terminated with a user code of 108 and a core dump.

On return, the registers and condition code are as follows:

Reg 14 - address of instruction following macro expansion.
Reg 15 - successful search: address of node satisfying search.
- unsuccessful search: address of null node.
Reg 0 - successful search: index of node satisfying search.
- unsuccessful search: zero.
Reg 1 - address of the stack descriptor.
Reg 2-13 - as before execution of the macro.
cc - successful search: 0.
- unsuccessful search: 2.
FPR 0 - the key, if MODE=CE or CD, otherwise as before execution of the macro.

Notes:

1. It is possible for the null node to satisfy the search.
2. To avoid a possible alignment error, in cases other than MODE=CLC the key length must be a factor of the node size.
3. To avoid a possible out-of-range error, the explicit or implied values of DISPL and LEN must sum to a value which is not larger than the size of the node.

Appendix A - Abnormal Terminations

In order to obtain the core dump, a SYSUDUMP or SYSABEND DD card must be included.

U 100 - from UNSTK - indicates an attempt to unstack the null node.

Contents of the registers and condition code at time of abend are:

Reg 14 - address of instruction following macro expansion
Reg 15 - address of stack descriptor
Reg 0 - as before execution of macro
Reg 1 - abend codes
Reg 2-13 - as before execution of macro
cc - as before execution of macro

U 104 - from GTSTK - indicates invalid parameter information.

Contents of the registers and condition code at time of abend are:

Reg 14 - address of instruction following macro expansion
Reg 15 - node size
Reg 0 - subpool number (1 byte), GETMAIN amount (3 bytes)
Reg 1 - abend codes
Reg 2-13 - as before execution of macro
cc: 0 - node size = 0
1 - node size less than zero
- or subpool number greater than 127
2 - node size greater than 256
- or reg 0 ≠ NODES*SIZE+40
- or NODES*SIZE+40 greater than 32791

U 108 - from SHSTK - indicates invalid parameter information.

Contents of the registers and condition code at time of abend are:

Reg 14 - address of instruction following macro expansion
Reg 15 - address of stack descriptor
Reg 0 - address of key (MODE=CLC)
- or key (MODE=C,CH,CL)
- or as before execution of macro (others)
Reg 1 - abend codes
Reg 2-13 - as before execution of macro
FPR 0 - key (MODE=CE,CD)
- or as before execution of macro (others)
cc: 1 - alignment error
2 - field to be examined lies partly outside of node (out-of-range error)

Appendix B - Examples

(a) Create a stack, node size = 4, 60 nodes per stack section, subpool zero:

```
          .
          .
          GTSTK
          ST     1,STACKADR
          .
          .
          .
STACKADR  DS     A
          .
          .
```

(b) Create the same stack as in example (a), but with the module WSUSTACK loaded dynamically, and using a remote parameter list for GTSTK:

```
          .
          .
          LOAD   EP=WSUSTACK
          ST     0,ASTAKER
          .
          .
          GTSTK ASTAKER,MF=(E,LIST)
          ST     1,STACKADR
          .
          .
STACKADR  DS     A
ASTAKER   DS     A
LIST      GTSTK MF=L
          .
          .
```

(c) Stack the contents of register 3 on the stack created in example (b) and save the address and index of that node for later retrieval:

```
          .
          .
          STACK STACKADR,ASTAKER
          ST     3,0(,15)
          STM    15,0,INDXSV
          .
          .
INDXSV    DS     2F
          .
          .
```

(d) Unstack the two top nodes on the stack created in example (b) and load the contents of those nodes into registers 8 and 11 respectively. The address of the current top-of-stack is unknown.

```
          .
          .
          IXSTK STACKADR
          L     8,0(,15)
          UNSTK STACKADR,ASTAKER
          L     11,0(,15)
          UNSTK (1),ASTAKER
          .
          .
```

(e) Replace the contents of the node stacked in example (c) with the contents of register 4.

```
          .
          .
          L     15,INDXSV
          ST    4,0(,15)
          .
          .
```

or:

```
          .
          .
          L     0,INDXSV+4
          IXSTK STACKADR,(0),ASTAKER
          ST    4,0(,15)
          .
          .
```

(f) Search the stack created in example (b) for a compare logical equal with the contents of register 6.

```
          .
          .
          SHSTK STACKADR,ASTAKER,MODE=CL,COND=EQ,KEY=(6)
          BNZ   NOTFOUND
          ST    15,SAVEADR
          .
          .
NOTFOUND  routine if search unsuccessful
          .
          .
SAVEADR   DS    A
          .
          .
```

(g) Retrieve the contents of the second node from the top of the stack created in example (b).

```
.
.
IXSTK STACKADR
S     0,=F'1'
IXSTK STACKADR,(0),ASTAKER
L     5,0(,15)
.
.
```

(h) Check before unstacking that the null node will not be unstacked.

```
.
.
IXSTK STACKADR
BCT   0,UNSTACK
routine for attempt to unstack null node
.
.
UNSTACK   UNSTK STACKADR,ASTAKER
.
.
```

(i) Clear the stack created in example (b).

```
.
.
CLSTK STACKADR,ASTAKER
.
.
```

(j) Delete the stack created in example (b).

```
.
.
DLSTK STACKADR,ASTAKER
.
.
```

Appendix C - Internal Data Structure

(a) Stack Descriptor and First Section:

Displacement (descriptor)	Displacement (first section)	Length	Contents
0	-	6	Compare instruction for SHSTK
6	-	2	BCR instruction for SHSTK
8	-	1	Subpool number
9	-	3	Size of stack section
12	-	4	Node size (bytes)
16	-	4	Address of node at top of stack
20	-	4	Number of nodes in stack (including null node)
24	0	4	Address of highest section
28	4	4	Address of highest available node in this section
32	8	4	Number of meaningful nodes in this section
36	12	4	Address of next higher section
40	16	?	First (null) and subsequent nodes

(b) Subsequent sections:

Displacement	Length	Contents
0	4	Address of next lower section
4	4	Address of highest available node in this section
8	4	Number of meaningful nodes in this section
12	4	Address of next higher section
16	?	Nodes in this section

(c) Details of bytes 0-7 in stack descriptor on entry to SHSTK:

a_1	a_2	b_1	b_2	2	0	c_1	c_2
5	0	d_1	d_2	0	7	c	F

Byte 0: Operation code for compare -

MODE= :	CLC	CLI	C	CL	CH	CE	CD
a_1a_2= :	D5	95	59	55	49	79	69

Byte 1: L field (CLC); I_2 field (CLI); R1/X2 fields (others) -

MODE=	b_1b_2
CLC, LEN given	LEN-1
CLC, LEN not given	unspecified
CLI	key
C, CL, CH	50
CE, CD	00

Byte 3: Displacement of field in node (DISPL parameter)

Byte 5: LEN specification flag (MODE=CLC only);
eventually set to zero as displacement of key -

LEN given: d_1d_2=00

LEN not given: d_1d_2=FF

Byte 7: Condition and branch register -

COND=:	EQ	NE	GT	LT	GE	LE
e (MODE=CLC or CLI):	6	8	C	A	4	2
e (MODE= others):	6	8	A	C	2	4

Appendix D - Implementation Guide

1. Environment considerations:

WSUSTACK will operate under any version of OS/360 which uses the current (since release 14) calling sequences for R-type GETMAIN and FREEMAIN (SVC 10) and for ABEND (SVC 13). To optimize the code, the IBM macros are not used; the parameter structures for the SVC's are maintained implicitly or calculated as needed.

Both the source module WSUSTACK and code incorporating macro calls to WSUSTACK will assemble correctly under IBM's Assembler level F.

The assembled source module may be linkedited with the RENT and REUS attributes.

2. Testing history:

WSUSTACK was dveloped and tested at Washington State University on an IBM System 360 Model 67-I, under OS/MVT. It has been used extensively in compiler-writing classes at Washington State University, and no errors or problems have been reported.

3. Deck key:

The program package supplied consists of an assembler source program and nine macro definitions, in two EBCDIC card decks -

Deck #1 - assembler source deck
- identification = WSUSTACK
- 337 cards, numbered WSTK0001 to WSTK0337 in columns 73 to 80
- deck contains one assembler source program, without control cards (JCL)

3. Deck key (continued):

Deck #2 - macro library

- identification = WSUSTKML
- 229 cards, numbered STML0001 to STML0229 in columns 73 to 80
- deck contains nine (9) macros in alphabetical order, each preceded by one IEBUPDTE utility control statement in the following form:

```
/.  ADD  LIST=ALL,LEVEL=00,SOURCE=0,
                          NAME=macroname
```

The nine macros supplied are the following:

CALSTK (inner macro only)	- 20 cards
CLSTK	- 5 cards
DLSTK	- 12 cards
GTSTK	- 62 cards
IXSTK	- 18 cards
RJCRG (inner macro only)	- 12 cards
SHSTK	- 81 cards
STACK	- 5 cards
UNSTK	- 5 cards

4. Local implementation changes

(a) Different macro names -

If different names for the outer macros are desired, all that is necessary is to change the macro names on the corresponding prototype statements (as well as including the macros in the macro library under the desired names), and altering the documentation accordingly.

If different names are desired for the inner macros (CALSTK, RJCRG), the above must be done, and in addition all inner macro calls done by the outer macros must be updated to reflect the alteration.

4. Local implementation changes (continued):

(b) Different module name -

If it is desired to change the name WSUSTACK to some other name, two things must be done before assembling the source and creating the macro library. First, change the second statement in WSUSTACK from

```
WSUSTACK CSECT
```

to

```
newname  CSECT
WSUSTACK EQU   *
```

and second, change the literal =V(WSUSTACK) in the CALSTK macro to =V(newname) .

5. Some estimated timings:

Timings are given in microseconds, and do not include the time for executing instructions in the macro expansion. Data pertains to the IBM S/360 model 67-I, and is extracted from IBM System 360 Model 67 Functional Characteristics, form number GA27-2719-1.

STACK - no section overflow: 17.84
- section overflow: 29.42 + time for GETMAIN

UNSTK - no section underflow: 17.34 + .4b
- section underflow: 23.51 + time for FREEMAIN

IXSTK - index in range: 21.76 + 2.9k + .8b
- index = 0: 9.53 + .4b
- index out of range: 3.6

- where: b = 1 if save area (reg 13) lies on a doubleword boundary,
0 if save area lies on a fullword not a doubleword boundary;

k = number of section boundaries passed to access the node (if the node is in the first section, k = 0).