Advanced Features of Macro Instructions

This lecture will focus on some of the advanced features of the macro language as implemented by the IBM/System 360 assembler.

We shall focus on our stack handling macros.

Some of the features to be covered by this lecture include.

1. The use of concatenation to generate type–specific instructions.
2. Some standard system variable symbols.
3. The use of one system variable symbol to solve the branch problem.
4. Conditional assembly.
5. The use of conditional assembly as a help in writing **STKPOP**.

7. The **ABEND** macro and its use in signaling run–time errors.
8. A completed version of our stack macros.

Concatenation: Building Operations

In a model statement, it is possible to concatenate two strings of characters.

Consider the macro prototype to load a register from one of several sources.
Note the use of the string "**&NAME**" to allow this to be a branch target.

```
         MACRO
&NAME    LOAD &REG,&TYPE,&ARG
&NAME    L&TYPE &REG,&ARG
         MEND
```

Consider a number of invocations.

LOAD R7,R,R6	becomes	**LR R7,R6**
LOAD R7,H,HW	becomes	**LH R7,HW**
LOAD R7,,FW	becomes	**L R7,FW**

Note here: the second argument is empty. The empty string is concatenated to **"F"**.

We shall now extend the stack operations to push and pop contents of
half–words and full–words, as well as registers.

Pushing from Various Sources

We look first at the handling of our **STKPUSH**. The only restriction on the stack is that every value pushed be treated as a 32–bit fullword.

As a result, a 16–bit halfword will be sign–extended to a 32–bit fullword before being pushed onto the stack. This is similar to the function of the **LH** instruction, which loads a register from a halfword.

The key instruction in the original **STKPUSH** macro is the following.

```
    ST  &R,0(3,2)     STORE THE ITEM INTO THE STACK
```

In this case, the item to be placed on the stack is found in the register indicated by the symbolic parameter **&R**.

The way to extend this instruction to all data types is as follows.

1. Select a register to be a fixed source for the word on the stack, and
2. Construct instructions to load that fixed register from the source.

What Shall Be Stored on the Stack?

At this point, we have a decision to make. What data types to store?

The size restriction on the stack limits the simple choices to addresses and the contents of registers, halfwords, and fullwords.

We must select a working register for the new macro. I select R4.
The "key code" becomes as follows.

Stacking an address

```
LA R4,&ARG        Load address into R4.
ST R4,&R,0(3,2)
```

Stacking a halfword

```
LH R4,&ARG       Load halfword into R4.
ST R4,&R,0(3,2)
```

Stacking a fullword

```
L  R4,&ARG       Load fullword into R4.
ST R4,&R,0(3,2)
```

Stacking a register

```
LR R4,&ARG       Load value from source
                 register into R4.
ST R4,&R,0(3,2)
```

Passing the Type in a Macro Invocation

The solution adopted to the problem above is to pass the type in the macro call and use concatenation to build the load operator.

Here is some code taken from a macro definition that has been run and tested. First, we show the macro prototype.

```
&L2      STKPUSH &ARG,&TYP
```

Next we show the "key instruction" in the macro body.

```
         L&TYP R4,&ARG
```

Here are four typical invocations of the macro.

```
          STKPUSH R7,R      PUSH VALUE IN REGISTER.
          STKPUSH HHW,H     PUSH A HALFWORD VALUE.
          STKPUSH FFW,A     PUSH AN ADDRESS.
          STKPUSH FFW       PUSH A FULLWORD.
```

Note that the last invocation lacks a second argument. In the expansion, this causes **&TYP** to be set to **' '**, a blank; "**L&TYP**" becomes "**L** ".

The Macro Definition

Here is the definition for the macro at this stage of its development.

```
         MACRO
&L2      STKPUSH &ARG,&TYP
&L2      LH    R3,STKCOUNT
         SLA   R3,2
         LA    R2,THESTACK
         L&TYP R4,&ARG
         ST    R4,0(3,2)
         LH    R3,STKCOUNT
         AH    R3,=H'1'
         STH   3,STKCOUNT
         MEND
```

Again, the "**&L2**" allows the macro invocation to be a branch target.

Some Invocations of this Macro

```
91          STKPUSH R7,R
92+         LH     R3,STKCOUNT
93+         SLA    R3,2
94+         LA     R2,THESTACK
95+         LR     R4,R7
96+         ST     R4,0(3,2)
97+         LH     R3,STKCOUNT
98+         AH     R3,=H'1'
99+         STH    3,STKCOUNT

100          STKPUSH HHW,H
101+         LH     R3,STKCOUNT
102+         SLA    R3,2
103+         LA     R2,THESTACK
104+         LH     R4,HHW
105+         ST     R4,0(3,2)
106+         LH     R3,STKCOUNT
107+         AH     R3,=H'1'
108+         STH    3,STKCOUNT
```

More Invocations of this Macro

```
109          STKPUSH FFW
110+         LH     R3,STKCOUNT
111+         SLA    R3,2
112+         LA     R2,THESTACK
113+         L      R4,FFW
114+         ST     R4,0(3,2)
115+         LH     R3,STKCOUNT
116+         AH     R3,=H'1'
117+         STH    3,STKCOUNT

118          STKPUSH FFW,A
119+         LH     R3,STKCOUNT
120+         SLA    R3,2
121+         LA     R2,THESTACK
122+         LA     R4,FFW
123+         ST     R4,0(3,2)
124+         LH     R3,STKCOUNT
125+         AH     R3,=H'1'
126+         STH    3,STKCOUNT
```

NOTE: The originals of the program listing are found at the end of the slides.

Saving the Work Registers

As written, this macro has the side effect of changing the values of three registers: R2, R3, & R4. The value of R4 is preserved only if it is being pushed.

We should write macros so that they operate without side effects. The only way to do this is to save and restore the values of the work registers.

There are many ways to do this. The simplest is to alter the stack data structure. Here is the new version.

```
STKCOUNT DC H‘0’     NUMBER OF ITEMS STORED ON STACK
STKSIZE  DC H‘64’    MAXIMUM STACK CAPACITY
STKSAV2  DC F‘0’     SAVES CONTENTS OF R2
STKSAV3  DC F‘0’     SAVES CONTENTS OF R3
STKSAV4  DC F‘0’     SAVES CONTENTS OF R4
THESTACK DC 64F‘0’   THE STACK HOLDS 64 FULLWORDS
```

This new definition does not alter the **STKINIT** macro. It does affect the other two macros: **STKPOP** and **STKPUSH**. We illustrate the latter.

The First Revision of STKPUSH

Here is the revision that allows the work registers to be saved.

```
         MACRO
&L2      STKPUSH &ARG,&TYP
&L2      ST    R2,STKSAV2   THE ORDER OF SAVING
         ST    R3,STKSAV3   IS NOT IMPORTANT.
         ST    R4,STKSAV4
         LH    R3,STKCOUNT
         SLA   R3,2
         LA    R2,THESTACK
         L&TYP R4,&ARG
         ST    R4,0(3,2)
         LH    R3,STKCOUNT
         AH    R3,=H'1'
         STH   R3,STKCOUNT
         L     R4,STKSAV4   THE ORDER OF RESTORATION
         L     R3,STKSAV3   IS NOT IMPORTANT EITHER.
         L     R2,STKSAV2
         MEND
```

The Status of the Macros at This Point

There are a few issues to be addressed at this point.

The only macro that will not change is the initialization macro, **STKINIT**.

1. We have not yet dealt with generalizing the **STKPOP** macro.
2. We have not yet dealt with either the stack empty problem or that of the stack being full. Each has to be addressed.

Each of these issues demands the use of techniques we have not yet discussed.

Consider the first problem. We shall want to pop the following from the stack: register values, halfwords, and fullwords. The type for the argument refers to the destination; an address can be popped into either a register or fullword.

In order to see the problem for **STKPOP**, consider the "key instruction".

Halfword: **STH R4,&ARG**

Fullword: **ST R4,&ARG**

Register: **LR &ARG,R4 No STR for store register.**

We could write a **STR** macro, but I want to use another solution.

Some System Variable Symbols

There are a number of system variable symbols. I mention three.

&SYSDATE The system date, in the 8 character form "**MM/DD/YY**".
Use in the form of a declaration of initialized storage, as in
TODAY DC C'&SYSDATE'

&SYSTIME The system time of day, in the five character form "**HH.MM**".
Also used in the form of a declaration, as in
NOW DC C'&SYSTIME'

&SYSNDX The macro expansion index. For the first macro expansion, the Assembler initializes **&SYSNDX** to the string **"0001"**.
Each macro invocation increases the value represented by 1, giving rise to the sequence **"0001"**, **"0002"**, **"0003"**, etc.

The **&SYSNDX** system variable symbol can prevent a macro from generating duplicate labels. The system symbol is concatenated to a leading character, which begins the label and must be unique within the macro definition.

More on the Macro Expansion Index

First consider the following string, used as a label in a macro definition.

```
L&SYSNDX L R4,STKSAV4
```

Note that the string "**L&SYSNDX**", as written, contains eight characters: the initial character "**L**" followed by the 7 character sequence "**&SYSNDX**".

On expansion, this will be converted to labels such as "**L0001**", "**L0002**", etc.

In the macro definition, this takes the maximum eight characters allowed for a properly formatted listing. For this reason, I suggest that the better form for the label in the macro definition is **Single_Letter&SYSNDX**.

In actual fact, the requirement for the leading characters, to which the **&SYSNDX** is to be appended can be any sequence of one to four characters, provided only that the first character is a letter. Thus the following are valid.

```
A12&SYSNDX ...    This label might become A120003.
WXYZ&SYSNDX ...   This might become WXYZ0117.
```

I suggest use of a single leading letter, this allows 26 labels per macro.

A Simple Example of Label Generation

Consider the simple macro used for packed division in the previous lecture. We adapt it to prevent division by zero.

```
         MACRO
&LABEL   DIVID &QUOT,&DIVIDEND,&DIVISOR
&LABEL   ZAP   &QOUT,&DIVIDEND
         CP    &DIVISOR,=P‘0’  IS IT ZERO
         BNE   A&SYSNDX        NO, DIVISION IS OK
         ZAP   &QUOT,=P‘0’     YES, SET QUOTIENT TO 0
         B     B&SYSNDX
A&SYSNDX DP    &QUOT,&DIVISOR
B&SYSNDX NOPR  R3              DO NOTHING
         MEND
```

Note that the format of the **NOPR** instruction requires a register number (here **R3**), even though the instruction does nothing.

Sample Expansion of the Macro

With the above definition, consider the following expansions.

```
 A10START DIVID X,Y,Z
+A10START ZAP   X,Y
+         CP    Z,=P'0'      IS IT ZERO
+         BNE   A0001        NO, DIVISION IS OK
+         ZAP   X,=P'0'      YES, SET QUOTIENT TO 0
+         B     B0001
+A0001    DP    X,Z
+B0001    NOPR  R3           DO NOTHING

 A20DOIT  DIVID A,B,C
+A20DOIT  ZAP   A,B
+         CP    C,=P'0'      IS IT ZERO
+         BNE   A0002        NO, DIVISION IS OK
+         ZAP   X,=P'0'      YES, SET QUOTIENT TO 0
+         B     B0002
+A0002    DP    A,C
+B0002    NOPR  R3           DO NOTHING
```

Note that each invocation has distinct labels. This removes the name clashes.

Another Design Strategy for DIVID

In this variant, a zero divisor will cause the program to terminate abnormally.

```
         MACRO
&LABEL   DIVID &QUOT,&DIVIDEND,&DIVISOR
&LABEL   ZAP   &QOUT,&DIVIDEND
         CP    &DIVISOR,=P‘0’  IS IT ZERO
         BNE   A&SYSNDX        NO, DIVISION IS OK
         ABEND                 INVOKE THE MACRO TO
                               TERMINATE EXECUTION.
A&SYSNDX DP    &QUOT,&DIVISOR
         MEND
```

The First Revision of STKINIT

Here is a revision of the STKINIT code that allows initialization of its size.

```
         MACRO
&L1      STKINIT &SIZE
&L1      ST R3,STKSAV3
         SR R3,R3
         STH R3,STKCOUNT
         L  R3,STKSAV3
         B  L&SYSNDX
STKCOUNT DC H'0'
STKSIZE  DC H'&SIZE'
STKSAV2  DC F'0'
STKSAV3  DC F'0'
STKSAV4  DC F'0'
THESTACK DC &SIZE.F'0'
L&SYSNDX SLA R3,0
         MEND
```

Note the **“.”** in the definition of **THESTACK**. This concatenates the value of the symbolic parameter with “**F‘0’**”, as in “**128F‘0’**”

The Second Revision of STKPUSH

```
         MACRO
&L2      STKPUSH &ARG,&TYP
&L2      ST    R3,STKSAV3
         LH    R3,STKCOUNT  GET COUNT OF ITEMS ON THE STACK
         CH    R3,STKSIZE   IS THE STACK FULL?
         BNL   Z&SYSNDX     YES, DO NOT ADD ANOTHER.
         ST    R4,STKSAV4   NO, WE CAN PUSH ANOTHER ITEM.
         ST    R2,STKSAV2   START BY SAVING THE OTHER 2 REGISTERS
         SLA   R3,2         MULTIPLY THE INDEX BY 4.
         LA    R2,THESTACK
         L&TYP R4,&ARG      FORM THE ADDRESS
         ST    R4,0(3,2)    STORE THE ITEM
         LH    R3,STKCOUNT  GET THE OLD COUNT OF ITEMS
         AH    R3,=H'1'     INCREMENT THE COUNT BY 1
         STH   R3,STKCOUNT  STORE THE CURRENT COUNT
         L     R4,STKSAV4   RESTORE THE REGISTERS.
         L     R2,STKSAV2
Z&SYSNDX L     R3,STKSAV3
         MEND
```

This is the final version of the **STKPUSH** macro.

We must discuss another basic topic before addressing **STKPOP**.

Conditional Assembly

We have already seen how concatenation can be used to construct different instructions in a macro expansion.

We now investigate conditional assembly, in which the expansion of a macro can lead to a number of distinct code sequences.

Conditional assembly permits the testing of attributes such as data format, data value, or field length, and to use the results of such testing to generate source code that is specific to the case in question.

This lecture will focus on five specific conditional assembly instructions.

AGO	an unconditional branch
AIF	a conditional branch. This means "Ask If".
ANOP	A NOP that can be the branch target for either **AGO** or **AIF**.
MNOTE	print a programmer defined message at assembly time
MEXIT	exit the macro definition.

Attributes for Use by Conditional Assembly

The assembler can generate code specified by certain attributes of the arguments to the macro definition at the time it is expanded.

There are six types of attributes that can be associated with a parameter. Here are three if the more useful attributes.

L’	Length	The length of the symbolic parameter
I’	Integer	The integer attribute of a fixed–point, floating–point, or packed decimal number.
T’	Type	The type of the parameter, as specified by the DC or DS declaration with which it is defined.

Some types for the T’ attribute are as follows.

A	Address	C	Character	H	Halfword	P	Packed Decimal
B	Binary	F	Fullword	I	Instruction	X	Hexadecimal

The Ask If (AIF) Instruction

The **AIF** instruction has two parts.

1. A logical expression in parentheses, and
2. A sequence symbol immediately following, which serves as the branch target.

The **AIF** logical expression may use the following relational operators, which are quite similar to those seen in early versions of the FORTRAN language.

EQ	Equal To	**NE**	Not Equal To
LT	Less Than	**GE**	Greater Than or Equal To
GT	Greater Than	**LE**	Less Than or Equal To

If the type of **&AMT** is packed, go to **.B23PACK**

```
AIF(T'&AMT EQ 'P').B23PACK
```

If the type of **&LINK** is not an instruction, go to **.R30ERROR**

```
AIF(T'&LINK NE 'I').R30ERROR
```

Testing the Value of a Symbolic Parameter

What we want for the STKPOP instruction is a conditional assembly based on the value of the second parameter.

The prototype will be something like

```
&L1      STKPOP &ARG,&TYP
```

What we want to issue is an **AIF** statement such as

```
AIF (&TYP EQ ‘R’).ISREG
```

There is a well–known peculiarity in assembler language, not just in the IBM Assembler, that disallows this straightforward construct.

We must put the symbolic parameter in single quotes. The statement is thus:

```
AIF (‘&TYP’ EQ ‘R’).ISREG
```

If **&TYP** is the character R, the logical expression becomes **(‘R’ EQ ‘R’)**, which immediately evaluates to True, and the branch is taken.

Reference

Page 384, High Level Assembler for z/OS & z/VM & z/VSE Language Reference Manual, Release 6 (July 2008), SC26–4940–05

Targets for Use by Conditional Assembly

Each of the **AGO** and **AIF** instructions is a branch instruction that takes effect at assembly time. Neither persists into the assembly source code.

It should be expected that the targets for either of these conditional assembly branch instructions should be of a distinct type.

The targets for these are called **sequence symbols**.

The format of a sequence symbol is as follows.
A **sequence symbol** begins with a period (.) followed by one to seven letters or digits, the first of which must be a letter.

Unlike the symbols created by use of the **&SYSNDX** system symbol, sequence symbols do not persist into assembly time, and thus cannot generate a name conflict for the assembler.

A Sample of Conditional Assembly

Here is the DIVID macro, with conditional assembly instructions to insure that it is expanded only for parameters that are packed decimal.

```
         MACRO
&LABEL   DIVID &QUOT,&DIVIDEND,&DIVISOR
         AIF   (T’&QUOT NE ‘P’).NOTPACK
         AIF   (T’&DIVIDEND NE T’&QUOT).NOTPACK
         AIF   (T’&DIVISOR NE T’&QUOT).NOTPACK
         AGO   .DOIT
.NOTPAK  MNOTE  ‘ONE PARAMETER IS NOT PACKED DECIMAL’
         MEXIT
.DOIT    ANOP
&LABEL   ZAP   &QOUT,&DIVIDEND
         CP    &DIVISOR,=P‘0’  IS IT ZERO
         BNE   A&SYSNDX        NO, DIVISION IS OK
         ZAP   &QUOT,=P‘0’     YES, SET QUOTIENT TO 0
         B     B&SYSNDX
A&SYSNDX DP    &QUOT,&DIVISOR
B&SYSNDX NOPR  R3              DO NOTHING
         MEND
```

Some Examples of the Conditional Assembly Divide Macro

In the following, assume that each of **X**, **Y**, and **Z** is defined by a DC statement as packed decimal, but that **A**, **B**, and **C** are defined as halfwords.

Here are some possible expansions.

```
 F10DOIT  DIVID X,Y,Z
+F10DOIT  ZAP   X,Y
+         CP    Z,=P‘0’          IS IT ZERO
+         BNE   A0032            NO, DIVISION IS OK
+         ZAP   X,=P‘0’          YES, SET QUOTIENT TO 0
+         B     B0032
+A0032    DP    X,Z
+B0032    NOPR  R3               DO NOTHING

 F25NODO  DIVID A,B,C
+ONE PARAMETER IS NOT PACKED DECIMAL
```

The Original Definition of Macro STKPOP

We now begin our redefinition of the **STKPOP** macro.
We begin with the original definition, which popped a value into a register.

```
*STKPOP
         MACRO
&L3      STKPOP &R
&L3      LH  3,STKCOUNT   GET THE STACK COUNT
         SH  3,=H’1’      SUBTRACT 1 WORD OFFSET OF TOP
         STH 3,STKCOUNT   STORE AS NEW SIZE
         SLA 3,2          BYTE OFFSET OF STACK TOP
         LA  2,THESTACK   ADDRESS OF STACK BASE
         L   &R,0(3,2)    LOAD ITEM INTO THE REGISTER.
         MEND
*
```

Again, this macro has one symbolic parameter: **&R**. Again, a register number.

We want to expand this definition in a number of ways.

We begin by introducing the type **&TYP**.

At this point, it will become necessary to have another work register.

Mechanics of the Revised STKPOP

The new design will use register R4 to transfer the value at the top of the stack.

The new prototype will be as follows.

```
&L3      STKPOP &ARG,&TYP
```

Each type of instruction will include the following as the first statement in the "key code" – that which actually places the value into the destination.

```
         L   R4,0(3,2)    LOAD ITEM INTO REGISTER R4.
```

The second statement of the "key code" depends on the type of the destination.

```
&TYP == H        STH R4,&ARG

&TYP == F        ST  R4,&ARG

&TYP == A        ST  R4,&ARG  (SAME AS FULLWORD)

&TYP == R        LR &ARG,R4   COPY R4 INTO REGISTER
```

Again, I could define a STR macro and avoid the use of conditional assembly. For a number of reasons, I have chosen not to do so.

The Key Code as Reflected in Conditional Assembly

Again, the new prototype will be as follows.

```
&L3      STKPOP &ARG,&TYP
```

Here is the key code section, with the conditional assembly.

The first statement is common to all types.

```
         L   R4,0(3,2)    LOAD ITEM INTO REGISTER R4.
         AIF (‘&TYPE’ EQ ‘R’).ISREG
         ST&TYP R4,&ARG
         AGO .CONT
.ISREG   LR &ARG,R4
.CONT    The next statement.
```

STKPOP: Revision 2

Here I am going to add some code to save and restore the work registers.

```
         MACRO
&L3      STKPOP &ARG,&TYP
&L3      ST  R2,STKSAV2
         ST  R3,STKSAV3
         ST  R4,STKSAV4
         LH  R3,STKCOUNT   GET THE STACK COUNT
         SH  R3,=H’1’      SUBTRACT 1 WORD OFFSET OF TOP
         STH R3,STKCOUNT   STORE AS NEW SIZE
         SLA R3,2          BYTE OFFSET OF STACK TOP
         LA  R2,THESTACK   ADDRESS OF STACK BASE
         L   R4,0(3,2)    LOAD ITEM INTO REGISTER R4.
         AIF (‘&TYPE’ EQ ‘R’).ISREG
         ST&TYP R4,&ARG
         AGO .CONT
.ISREG   LR &ARG,R4
.CONT    L   R4,STKSAV4
         L   R3,STKSAV3
         L   R2,STKSAV2
         MEND
```

STKPOP: The Complete Version

```
         MACRO
&L3      STKPOP &ARG,&TYP
&L3      ST   R3,STKSAV3
         LH   R3,STKCOUNT   GET THE STACK COUNT
         CH   R3,=H‘0’      IS THE COUNT POSITIVE
         BNH  Z&SYSNDX      NO, WE CANNOT POP.
         SH   R3,=H’1’      SUBTRACT 1 WORD OFFSET OF TOP
         STH  R3,STKCOUNT   STORE AS NEW SIZE
         SLA  R3,2          BYTE OFFSET OF STACK TOP
         ST   R2,STKSAV2    SAVE REGISTER R2
         ST   R4,STKSAV4    SAVE REGISTER R4
         LA   R2,THESTACK   ADDRESS OF STACK BASE
         L    R4,0(3,2)    LOAD ITEM INTO REGISTER R4.
         AIF (‘&TYPE’ EQ ‘R’).ISREG
         ST&TYP R4,&ARG
         AGO .CONT
.ISREG   LR &ARG,R4
.CONT    L    R4,STKSAV4
         L    R2,STKSAV2
Z&SYSNDX L    R3,STKSAV3
         MEND
```

Original Code for the Macro Expansions

```
                                        33 *          MACRO DEFINITIONS
                                        34 *
                                        35            MACRO
                                        36 &L2        STKPUSH &ARG,&TYP
                                        37 &L2        LH    R3,STKCOUNT
                                        38            SLA   R3,2
                                        39            LA    R2,THESTACK
                                        40            L&TYP R4,&ARG
                                        41            ST    R4,0(3,2)
                                        42            LH    R3,STKCOUNT
                                        43            AH    R3,=H'1'
                                        44            STH   3,STKCOUNT
                                        45            MEND
                                        46 *
                                        89 *          SOME MACRO INVOCATIONS
                                        90 *
                                        91            STKPUSH R7,R
00004A 4830 C0C6              000CC     92+           LH    R3,STKCOUNT
00004E 8B30 0002              00002     93+           SLA   R3,2
000052 4120 C0CA              000D0     94+           LA    R2,THESTACK
000056 1847                             95+           LR    R4,R7
000058 5043 2000              00000     96+           ST    R4,0(3,2)
00005C 4830 C0C6              000CC     97+           LH    R3,STKCOUNT
000060 4A30 C43A              00440     98+           AH    R3,=H'1'
000064 4030 C0C6              000CC     99+           STH   3,STKCOUNT
```

Original Macro Definitions

```
                                           100         STKPUSH HHW,H
000068 4830 C0C6              000CC        101+        LH    R3,STKCOUNT
00006C 8B30 0002              00002        102+        SLA   R3,2
000070 4120 C0CA              000D0        103+        LA    R2,THESTACK
000074 4840 C1CE              001D4        104+        LH    R4,HHW
000078 5043 2000              00000        105+        ST    R4,0(3,2)
00007C 4830 C0C6              000CC        106+        LH    R3,STKCOUNT
000080 4A30 C43A              00440        107+        AH    R3,=H'1'
000084 4030 C0C6              000CC        108+        STH   3,STKCOUNT
                                           109         STKPUSH FFW
000088 4830 C0C6              000CC        110+        LH    R3,STKCOUNT
00008C 8B30 0002              00002        111+        SLA   R3,2
000090 4120 C0CA              000D0        112+        LA    R2,THESTACK
000094 5840 C1CA              001D0        113+        L     R4,FFW
000098 5043 2000              00000        114+        ST    R4,0(3,2)
00009C 4830 C0C6              000CC        115+        LH    R3,STKCOUNT
0000A0 4A30 C43A              00440        116+        AH    R3,=H'1'
0000A4 4030 C0C6              000CC        117+        STH   3,STKCOUNT
                                           118         STKPUSH FFW,A
0000A8 4830 C0E6              000EC        119+        LH    R3,STKCOUNT
0000AC 8B30 0002              00002        120+        SLA   R3,2
0000B0 4120 C0EA              000F0        121+        LA    R2,THESTACK
0000B4 4140 C1EA              001F0        122+        LA    R4,FFW
0000B8 5043 2000              00000        123+        ST    R4,0(3,2)
0000BC 4830 C0E6              000EC        124+        LH    R3,STKCOUNT
0000C0 4A30 C45A              00460        125+        AH    R3,=H'1'
0000C4 4030 C0E6              000EC        126+        STH   3,STKCOUNT
                                           127 *
                                           136 **********************************
```

Original Macro Definitions

Revised Code for the Macros

The next few pages show the listing of the final forms of the macros, as actually coded and tested. These are followed by listings of the expanded macros.

```
*
         MACRO
&L1      STKINIT
&L1      ST R3,STKSAV3
         SR R3,R3
         STH R3,STKCOUNT           CLEAR THE COUNT
         L  R3,STKSAV3
         MEND
*
```

```
003000          MACRO
003100 &L2      STKPUSH &ARG,&TYP
003110 &L2      ST    R3,STKSAV3              SAVE REGISTER R3
003200          LH    R3,STKCOUNT             GET THE CURRENT SIZE
003210          CH    R3,STKSIZE              IS THE STACK FULL?
003220          BNL   Z&SYSNDX                YES, DO NOT PUSH
003230          ST    R4,STKSAV4              OK, SAVE R2 AND R4
003240          ST    R2,STKSAV2
003300          SLA   R3,2                    MULTIPLY BY FOUR
003310          LA    R2,THESTACK             ADDRESS OF STACK START
003320          L&TYP R4,&ARG                 LOAD R4 WITH VALUE
003330          ST    R4,0(3,2)               STORE INTO THE STACK
003331          LH    R3,STKCOUNT
003332          AH    R3,=H'1'
003333          STH   3,STKCOUNT
003334          L     R4,STKSAV4
003335          L     R2,STKSAV2
003336 Z&SYSNDX L     R3,STKSAV3
003337          MEND
003338 *
003339 *
```

Revised Macro Definitions

```
         MACRO
&L3      STKPOP &ARG,&TYP
&L3      ST    R3,STKSAV3
         LH    R3,STKCOUNT      GET THE STACK COUNT
         CH    R3,=H'0'         IS THE COUNT POSITIVE?
         BNH   Z&SYSNDX         NO, WE CANNOT POP
         SH    R3,=H'1'         SUBTRACT 1 WORD OFFSET
         STH   R3,STKCOUNT      STORE THE NEW SIZE
         SLA   R3,2             BYTE OFFSET OF STACK TOP
         ST    R2,STKSAV2       SAVE REGISTER R2
         ST    R4,STKSAV4       SAVE REGISTER R4
         LA    R2,THESTACK      ADDRESS OF STACK BASE
         L     R4,0(3,2)        LOAD ITEM INTO R4
         AIF   ('&TYP' EQ 'R').ISREG
         ST&TYP R4,&ARG
         AGO .CONT
.ISREG   LR &ARG,R4
.CONT    L   R4,STKSAV4
         L   R2,STKSAV2
Z&SYSNDX L   R3,STKSAV3
         MEND
*
```

Revised Code for the Macro Expansions

```
                                   128 *         SOME MACRO INVOCATIONS
                                   129 *
                                   130           STKINIT
00004A 5030 C22E           00234   131+          ST R3,STKSAV3
00004E 1B33                        132+          SR R3,R3
000050 4030 C226           0022C   133+          STH R3,STKCOUNT
000054 5830 C22E           00234   134+          L  R3,STKSAV3
                                   135 *
                                   136           STKPUSH R7,R
000058 5030 C22E           00234   137+          ST     R3,STKSAV3
00005C 4830 C226           0022C   138+          LH     R3,STKCOUNT
000060 4930 C228           0022E   139+          CH     R3,STKSIZE
000064 47B0 C08C           00092   140+          BNL    Z0010
000068 5040 C232           00238   141+          ST     R4,STKSAV4
00006C 5020 C22A           00230   142+          ST     R2,STKSAV2
000070 8B30 0002           00002   143+          SLA    R3,2
000074 4120 C236           0023C   144+          LA     R2,THESTACK
000078 1847                        145+          LR     R4,R7
00007A 5043 2000           00000   146+          ST     R4,0(3,2)
00007E 4830 C226           0022C   147+          LH     R3,STKCOUNT
000082 4A30 C5A2           005A8   148+          AH     R3,=H'1'
000086 4030 C226           0022C   149+          STH    3,STKCOUNT
00008A 5840 C232           00238   150+          L      R4,STKSAV4
00008E 5820 C22A           00230   151+          L      R2,STKSAV2
000092 5830 C22E           00234   152+Z0010     L      R3,STKSAV3
```

```
                                        153         STKPUSH HHW,H
000096 5030 C22E            00234       154+        ST     R3,STKSAV3
00009A 4830 C226            0022C       155+        LH     R3,STKCOUNT
00009E 4930 C228            0022E       156+        CH     R3,STKSIZE
0000A2 47B0 C0CC            000D2       157+        BNL    Z0011
0000A6 5040 C232            00238       158+        ST     R4,STKSAV4
0000AA 5020 C22A            00230       159+        ST     R2,STKSAV2
0000AE 8B30 0002            00002       160+        SLA    R3,2
0000B2 4120 C236            0023C       161+        LA     R2,THESTACK
0000B6 4840 C33A            00340       162+        LH     R4,HHW
0000BA 5043 2000            00000       163+        ST     R4,0(3,2)
0000BE 4830 C226            0022C       164+        LH     R3,STKCOUNT
0000C2 4A30 C5A2            005A8       165+        AH     R3,=H'1'
0000C6 4030 C226            0022C       166+        STH    3,STKCOUNT
0000CA 5840 C232            00238       167+        L      R4,STKSAV4
0000CE 5820 C22A            00230       168+        L      R2,STKSAV2
0000D2 5830 C22E            00234       169+Z0011   L      R3,STKSAV3
```

```
                                   170            STKPUSH FFW
0000D6 5030 C22E          00234    171+           ST     R3,STKSAV3
0000DA 4830 C226          0022C    172+           LH     R3,STKCOUNT
0000DE 4930 C228          0022E    173+           CH     R3,STKSIZE
0000E2 47B0 C10C          00112    174+           BNL    Z0012
0000E6 5040 C232          00238    175+           ST     R4,STKSAV4
0000EA 5020 C22A          00230    176+           ST     R2,STKSAV2
0000EE 8B30 0002          00002    177+           SLA    R3,2
0000F2 4120 C236          0023C    178+           LA     R2,THESTACK
0000F6 5840 C336          0033C    179+           L      R4,FFW
0000FA 5043 2000          00000    180+           ST     R4,0(3,2)
0000FE 4830 C226          0022C    181+           LH     R3,STKCOUNT
000102 4A30 C5A2          005A8    182+           AH     R3,=H'1'
000106 4030 C226          0022C    183+           STH    3,STKCOUNT
00010A 5840 C232          00238    184+           L      R4,STKSAV4
00010E 5820 C22A          00230    185+           L      R2,STKSAV2
000112 5830 C22E          00234    186+Z0012      L      R3,STKSAV3
```

```
                                    187         STKPUSH FFW,A
000116 5030 C22E           00234    188+        ST     R3,STKSAV3
00011A 4830 C226           0022C    189+        LH     R3,STKCOUNT
00011E 4930 C228           0022E    190+        CH     R3,STKSIZE
000122 47B0 C14C           00152    191+        BNL    Z0013
000126 5040 C232           00238    192+        ST     R4,STKSAV4
00012A 5020 C22A           00230    193+        ST     R2,STKSAV2
00012E 8B30 0002           00002    194+        SLA    R3,2
000132 4120 C236           0023C    195+        LA     R2,THESTACK
000136 4140 C336           0033C    196+        LA     R4,FFW
00013A 5043 2000           00000    197+        ST     R4,0(3,2)
00013E 4830 C226           0022C    198+        LH     R3,STKCOUNT
000142 4A30 C5A2           005A8    199+        AH     R3,=H'1'
000146 4030 C226           0022C    200+        STH    3,STKCOUNT
00014A 5840 C232           00238    201+        L      R4,STKSAV4
00014E 5820 C22A           00230    202+        L      R2,STKSAV2
000152 5830 C22E           00234    203+Z0013   L      R3,STKSAV3
```

```
                                           204 *
                                           205          STKPOP   R8,R
000156 5030 C22E             00234         206+         ST       R3,STKSAV3
00015A 4830 C226             0022C         207+         LH       R3,STKCOUNT
00015E 4930 C5A4             005AA         208+         CH       R3,=H'0'
000162 47D0 C186             0018C         209+         BNH      Z0014
000166 4B30 C5A2             005A8         210+         SH       R3,=H'1'
00016A 4030 C226             0022C         211+         STH      R3,STKCOUNT
00016E 8B30 0002             00002         212+         SLA      R3,2
000172 5020 C22A             00230         213+         ST       R2,STKSAV2
000176 5040 C232             00238         214+         ST       R4,STKSAV4
00017A 4120 C236             0023C         215+         LA       R2,THESTACK
00017E 5843 2000             00000         216+         L        R4,0(3,2)
000182 1884                                217+         LR R8,R4
000184 5840 C232             00238         218+         L      R4,STKSAV4
000188 5820 C22A             00230         219+         L      R2,STKSAV2
00018C 5830 C22E             00234         220+Z0014    L      R3,STKSAV3
```

```
                                        221          STKPOP  FFW
000190 5030 C22E            00234       222+         ST      R3,STKSAV3
000194 4830 C226            0022C       223+         LH      R3,STKCOUNT
000198 4930 C5A4            005AA       224+         CH      R3,=H'0'
00019C 47D0 C1C2            001C8       225+         BNH     Z0015
0001A0 4B30 C5A2            005A8       226+         SH      R3,=H'1'
0001A4 4030 C226            0022C       227+         STH     R3,STKCOUNT
0001A8 8B30 0002            00002       228+         SLA     R3,2
0001AC 5020 C22A            00230       229+         ST      R2,STKSAV2
0001B0 5040 C232            00238       230+         ST      R4,STKSAV4
0001B4 4120 C236            0023C       231+         LA      R2,THESTACK
0001B8 5843 2000            00000       232+         L       R4,0(3,2)
0001BC 5040 C336            0033C       233+         ST       R4,FFW
0001C0 5840 C232            00238       234+         L     R4,STKSAV4
0001C4 5820 C22A            00230       235+         L     R2,STKSAV2
0001C8 5830 C22E            00234       236+Z0015    L     R3,STKSAV3
```

```
                                      237        STKPOP  HHW,H
0001CC 5030 C22E            00234     238+       ST      R3,STKSAV3
0001D0 4830 C226            0022C     239+       LH      R3,STKCOUNT
0001D4 4930 C5A4            005AA     240+       CH      R3,=H'0'
0001D8 47D0 C1FE            00204     241+       BNH     Z0016
0001DC 4B30 C5A2            005A8     242+       SH      R3,=H'1'
0001E0 4030 C226            0022C     243+       STH     R3,STKCOUNT
0001E4 8B30 0002            00002     244+       SLA     R3,2
0001E8 5020 C22A            00230     245+       ST      R2,STKSAV2
0001EC 5040 C232            00238     246+       ST      R4,STKSAV4
0001F0 4120 C236            0023C     247+       LA      R2,THESTACK
0001F4 5843 2000            00000     248+       L       R4,0(3,2)
0001F8 4040 C33A            00340     249+       STH      R4,HHW
0001FC 5840 C232            00238     250+       L     R4,STKSAV4
000200 5820 C22A            00230     251+       L     R2,STKSAV2
000204 5830 C22E            00234     252+Z0016  L     R3,STKSAV3
                                      253 *
```

Revised Code for the Macro STKINIT

Here is an expansion of the newer definition of STKINIT, which allows the stack size to be specified.

```
                                      138          STKINIT 128
00004A 5030 C05E            00064     139+         ST R3,STKSAV3
00004E 1B33                           140+         SR R3,R3
000050 4030 C056            0005C     141+         STH R3,STKCOUNT
000054 5830 C05E            00064     142+         L  R3,STKSAV3
000058 47F0 C266            0026C     143+         B  L0009
00005C 0000                           144+STKCOUNT DC H'0'
00005E 0080                           145+STKSIZE  DC H'128'
000060 00000000                       146+STKSAV2  DC F'0'
000064 00000000                       147+STKSAV3  DC F'0'
000068 00000000                       148+STKSAV4  DC F'0'
00006C 0000000000000000               149+THESTACK DC 128F'0'
00026C 8B30 0000            00000     150+L0009    SLA R3,0
```