STACK STRUCTURES FOR THE TMS9900

A TEXAS INSTRUMENTS APPLICATION SHEET

TEXAS INSTRUMENTS
INCORPORATED

INTRODUCTION

Data stacks are a clean, efficient method by which complex data can be manipulated. And while it may not seem immediately obvious to a programmer unfamiliar with the TI 990 family of computers, stack structures can be implemented using the TMS9900 workspace system.

Workspace techniques, however, are not difficult to learn, and most stack concepts can be applied directly to the 990 family of computers. This report describes a method for simple automatic workspace allocation, and shows how familiar stack structures may be easily implemented on the TMS9900.

The advanced architecture of the TMS9900 does not make extensive use of internal registers, instead memory is used to provide general purpose register space. In fact, only three registers in the TMS9900, the Workspace Pointer, the Program Counter, and the Status Register, are accessible to the user. The Workspace Pointer (WP), a 16-bit register, is used to keep the address of a 16-word contiguous block of memory which is being used for the 16 general purpose registers. The Program Counter (PC) contains the address of the next instruction to be executed. The Status Register (ST) contains information about the existent state of processor operation.

A context switch occurs when the processor changes from one operating environment (context) to another. When an interrupt, subroutine call (BLWP), or Extended Operation (XOP) is executed, a context switch is said to have happened. A new operating context can be affected by loading new values into the Program Counter and Workspace Pointer Registers. These two values, WP and PC, are called the "TRAP VECTOR". The execution of a context switch will store to present CPU environment, WP, PC, and ST in the new context workspace registers R13, R14, and R15 respectively. The reverse operation occurs when an "RTWP" instruction is executed. The contents of R13, R14, and R15 are loaded into the WP, PC, and ST respectively

WORKSPACE STACKS

Nesting of "BLWP" subroutine calls is a common programming practice and it is possible for a programmer to inadvertently allocate the same workspace to routines which call each other. To avoid requiring the programmer to manually determine the workspace for each subroutine, it is preferable to use a method by which the workspace is selected automatically at execution time.

Figure 1 shows an XOP routine which performs this allocation by simply "stacking" the workspace down through memory. Rather than by calling a subroutine as:

```
BLWP @VECTOR
```

The subroutine is called through the XOP instruction which, for convenience, has been renamed by a DXOR directive as "CALL":

```
CALL @SUBRT
```

where SUBRT is the program counter (PC) for the subroutine. The XOP routine automatically assigns the subroutine workspace to the next lowest 16 words of memory. This routine can be further enhanced to provide a means by which the XOP routine can check for stack underflow and overflow.

DATA STACKS

The TMS9900 can easily accommodate data stacking with very simple instruction sequences. For ease of programming the data stack, examples are implemented as MACRO instructions. The structures can be manually generated since the longest consists of only two instructions.

Two general purpose stack MACRO instructions — PUSH and POP — are defined. The PUSH instruction stores the operand data onto a stack, and advances the stack pointer to the next available stack location. The POP sets the stack pointer to the most recently stored data, and transfers the data to the operand address.

Figure 1

A TRANSPARENT STACK STRUCTURE FOR THE 9900

```
*********************************************************************
*  ABSTRACT                                                         *
*    TRANSPARENT STACKING OF WORKSPACES IS ACHIEVED BY              *
*   CALLING ALL SUBROUTINES THROUGH AN XOP NAMED 'CALL.'            *
*   RETURN FROM ANY SUBROUTINE IS VIA A NORMAL RTWP.                *
*   ARGUMENTS MAY BE PASSED BY STANDARD REGISTER                    *
*   CONVENTIONS. THE STACK BUILDS DOWN THROUGH MEMORY               *
*   AND WILL BE [N+32]BYTES DEEP, WHERE N IS THE                    *
*   NESTING LEVEL. THE CALL XOP IS NOT REENTRANT, AND               *
*   THEREFORE PROTECTS AGAINST INTERRUPTS. ANY                      *
*   NUMBER OF WORKSPACE STACKS MAY COEXIST IN MEMORY.               *
*********************************************************************
*EXAMPLE OF USE     (XOP 14 IS ASSUMED TO BE THE CALL XOP.)
*
*
                      ....
*                     DXOP CALL,14              ;DEFINE THE XOP NAME, NUMBER
                    PRESET THE XOP VECTOR
0078                  AORG >78                  ;XOP VECTOR SPACE
0078  FF00            DATA XOPWP                ;WORKSPACE FOR XOP
007A  0094            DATA CALLPC               ;CALL XOP ENTRY POINT
*                     ....
*                     ....
*                   XOP USEAGE EXAMPLE FOLLOWS
0080                  AORG >80                  ;ARBITRARY START ADDRESS
0080  02E0  MAIN      LWPI TPSTCK               ;SET TOP OF WSPACE STACK
0082  FEC0
*                     ....
*                     ....
0084  0200            LI   RO,ARG1              ;PREPARE SUBR. ARGUMENT
0086  0000
0088  2FA0 *          CALL @SUBR                ;USE XOP TO CALL SUBR
008A  008C
*                     ....
*                     ....
*                     ....
*                     ....
008C  C05D  SUBR      MOV   >R13,R1             ;GET RO OF CALLING ROUTINE
*                     ....                      ;PROCESS ARGUMENTS
*                     ....
008E  CB41 *          MOV   R1,@2(R13)          ;RESULT TO R1 OF CALLER
0090  0002
0092  0380            RTWP                      ;NORMAL SUBROUTINE RETURN
*                     ....
*                     ....
*                     ....
*********************************************************************
*     THE SYMBOLS 'XOPWP', 'TPSTACK', AND 'ARG1' ALL HAVE           *
*     ARBITRARY VALUES FOR EXAMPLE USE ONLY.                        *
*********************************************************************
```

The most common application of these routines is to save the return address of a subroutine called by a "BL" instruction. Use of the PUSH and POP MACRO instructions, along with an example of return address stacking, are shown in Figure 2.

The sequence:

```
LABEL POP R11
      RT
```

will normally be repeated in many modules. This sequence can be assigned a label and shared by many subroutines. The result is that a program will generally have more PUSH than POP instructions. For this reason the stack was implemented as a build-up structure, allowing the shorter auto-increment instruction to be used in the PUSH MACRO.
In the example, R10 was chosen as a stack pointer. There is no restriction on which register can be used as the stack pointer, and multiple stacks and stack pointers may coexist in a system.

Although the TMS9900 uses a unique and powerful workspace architecture, traditional stack structures may be used both to augment workspace allocation and to provide additional data handling capabilities. The richness of TMS9900 addressing modes, and the minicomputer simplicity of the TMS9900 family instruction set allows the mixture of workspace and stack structures.

Figure 1 (Continued)

A TRANSPARENT STACK STRUCTURE FOR THE 9900

```
                 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 * CALL XOP                                                  *
                 *  THIS XOP ROUTINE AUTOMATICALLY STACKS                    *
                 *  WORKSPACES DOWN THROUGH MEMORY. THE                      *
                 *  NORMAL SUBROUTINE 'RTWP' WILL RETURN                     *
                 *  TO THE CALLER WITH THE OLD WORKSPACE,                    *
                 *  EFFECTIVELY "POPPING" THE STACK.                         *
                 *   CALLING SEQUENCE:                                       *
                 *    CALL @OPERAND   OR   XOP @OPERAND,N (N=CALL XOP)       *
                 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0094    0300     CALLPC LIMI 0                  ;PROTECT AGAINST RE-ENTRY
0096    0000
0098    CB4D            MOV  R13,@-6(R13)       ;MOVE RETURN WP
009A    FFFA
009C    CB4E            MOV  R14,@-4(R13)       ;MOVE RETURN PC
009E    FFFC
00A0    CB4F            MOV  R15,@-2(R13)       ;MOVE RETURN STATUS
00A2    FFFE
00A4    C38B            MOV  R11,R14            ;SUBROUTINE PC
00A6    022D            AI   R13,-32            :MOVE WORKSPACE DOWN IN RAM
00A8    FFE0     *
00AA    0380     *      RTWP                    ;CALL THE SUBROUTINE
                 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 *                                                           *
                 * THIS XOP REQUIRES 168 CYCLES TO EXECUTE.                  *
                 * AT 3MHZ THIS WOULD BE 56 MICROSECONDS                     *
                 * OF EXECUTION TIME.                                        *
                 *                                                           *
                 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 *
                 *
```

Figure 2

A TRANSPARENT STACK STRUCTURE FOR THE 9900

```
                ********************************************************
                * PUSH AND POP MACRO'S                                 *
                *      THESE MACRO INSTRUCTIONS IMPLEMENT THE TWO      *
                *      TRADITIONAL STACK OPERATIONS; PUSH AND POP.     *
                *      REGISTER 10 OF THE CURRENT WORKSPACE IS USED AS A*
                *      STACK POINTER. THE MACRO OPERANDS MAY BE ANY VALID*
                *      'MOV' INSTRUCTION OPERAND. MULTIPLE STACKS MAY  *
                *      EXIST IN MEMORY. THE STACK IN THIS SPACE IS     *
                *      DESIGNED TO BUILD UPWARDS THROUGH MEMORY TO ALLOW*
                *      MAXIMUM CODE EFFICIENCY.                        *
                ********************************************************
                PUSH    $MACRO  OP              ;DEFINE THE MACRO NAME, OPERAND
                        MOV  :OP.S:,*R10+       ;MOVE THE DATA ONTO STACK, BUMP(SP)
                        $END PUSH               ;END OF PUSH DEFINITION
                *
                POP     $MACRO  OP              ;DEFINE THE MACRO NAME, OPERAND
                        DECT R10                ;BACKUP STACK POINTER
                        MOV  *R10,:OP.S:        ;GET THE DATA FROM THE STACK
                        $END POP                ;END OF POP DEFINITION
                ********************************************************
                *EXAMPLE OF USE FOR PUSH AND POP MACRO'S               *
                *  THE "M" OPTION MAY BE USED TO SUPPRESS MACRO EXPANSION*
                *  IN THE LISTING. NOTE THAT THE MACRO EXPANSION IS FLAGGED*
                *  BY AN ASTRISK TO THE LEFT OF THE LINE NUMBER        *
                ********************************************************
00AC  020A      MAINP   LI   R10,STACK          ;ASSIGN BOTTOM OF DATA STACK
00AE  F000
                *       ....
00B0  06A0              BL   @SUBR1             ;CALL FIRST LEVEL SUBROUTINE
00B2  00B4
                *       ....
                SUBR1   PUSH R11                ;SAVE THE RETURN PC
00B4  CE8B              MOV  R11,*R10+  ;MOVE THE DATA ONTO STACK, BUMP(SP)
                *       ....
00B6  06A0              BL   @SUBR2             ;CALL SECOND LEVEL SUBROUTINE
00B8  00C0
                *       ....
                        POP  R11                ;RESTORE THE RETURN PC
00BA  064A              DECT R10                ;BACKUP STACK POINTER
00BC  C2DA              MOV  *R10,R11   ;GET THE DATA FROM THE STACK
00BE  045B      RT      RT                      ;RETURN TO CALLER
                *       ....
                *       ....
00C0  A081      SUBR2   A    R1,R2              ;PROCESS OPERANDS
                *       ....
00C2  045B              RT                      ;RETURN TO FIRST LEVEL
                ********************************************************
                * THE SYMBOL 'STACK' HAS AN ARBITRARY VALUE FOR EXAMPLE ONLY *
                ********************************************************
```