

New Technical Notes

Macintosh



Developer Support

Setting and Restoring A5

Overview

M.OV.A5

Revised by: Jim Reekes

June 1989

Written by: Andrew Shebanow

August 1988

The routines `SetupA5` and `RestoreA5` do not work properly when used with some optimizing Pascal and C compilers. Two new routines, `SetCurrentA5` and `SetA5`, are available in MPW 3.0, and they should work with any compiler.

Changes since December 1988: Removed the sample code and expanded the explanation of these two routines. The sample code in `M.TB.MultifinderMisc` reflects these new A5 routines.

Introduction

The in-line glue routines `SetupA5` and `RestoreA5` are often used by completion routines, VBL tasks, and interrupt handlers written in C and Pascal to gain access to an application's global variables. Unfortunately, these routines play fast and loose with the stack pointer.

Newer, more sophisticated, optimizing compilers (e.g., MPW C 3.0) will often leave function parameters on the stack across multiple function calls, removing the arguments for several functions with a single instruction. This significantly reduces code size and execution time, at the expense of a small amount of additional stack usage. As a side effect, this optimization breaks the `SetupA5` and `RestoreA5` glue.

This Technical Note describes a pair of in-line glue routines which have more functionality than `SetupA5` and `RestoreA5`, without making assumptions about a compiler's stack handling. These routines are provided as a standard part of the MPW Pascal 3.0 and MPW C 3.0 packages, in the files `OSUtils.p` and `OSUtils.h`, respectively.

The Old Way

The in-line code for `SetupA5` was:

```
MOVE.L    A5, -(A7)           ; leave old A5 on stack: Danger Will Robinson!
MOVE.L    CurrentA5, A5       ; set current A5
```

The in-line code for RestoreA5 was:

```
MOVE.L    (A7)+,A5        ; pop old A5 off stack
```

The problem is that `SetupA5` leaves the old value of A5 on the stack, and `RestoreA5` assumes that the stack pointer is still valid. If the programmer mistakenly calls `RestoreA5` within a called subroutine, the value that is popped off the stack and stored in A5 will be garbage. Of course, the “garbage” could be something moderately useful, like a return address.

The New, Totally Cool Way

The solution to this distressing problem is provided by two new functions in the MPW 3.0 libraries, `SetCurrentA5` and `SetA5`. The idea behind using these two routines is to get the application’s A5, pass it to some interrupt routine, which will use some globals, and then restore it to the original A5. Before you can call `SetA5`, you’ll have to have the application’s real A5. This is obtained by calling `SetCurrentA5` at non-interrupt time. Here’s how it should work.

An application is about to create an interrupt routine such as a VBL task, or a completion routine to be called from some asynchronous operation. To get the current A5, the application will use:

```
ourA5:= SetCurrentA5;
```

This call performs two functions. It will get the application’s A5, save it in a variable, and set A5 to the application’s low memory global, `CurrentA5`. The application will pass the result of `SetCurrentA5` to the interrupt routine.

The first instruction that the interrupt routine executes is to set A5 to the application’s A5. To do this the interrupt routine calls:

```
oldA5:= SetA5(ourA5);           {set A5 to the app's real A5}
                                {perform the interrupt task}
ignoreResult:= SetA5(oldA5);    {restore A5 to the original A5}
```

The call to `SetA5` performs two tasks. It will set A5 to the address specified and return the actual address in A5. You’ll use the original A5 address to call `SetA5` when the interrupt routine is about to exit. This action will restore A5. It’s also a good idea to read `M.TB.MultifinderMisc`, which demonstrates this technique in detail.

The Interfaces

The interfaces for `SetCurrentA5` and `SetA5`, along with their corresponding implementation as subroutine calls is:

MPW Pascal

```
FUNCTION SetCurrentA5 : LongInt;
    INLINE $2E8D, $2A78, $0904;

FUNCTION SetA5 (newA5 : LongInt) : LongInt;
    INLINE $2F4D, $0004, $2A5F;
```

MPW C

```
pascal long SetCurrentA5(void);
    { 0x2E8D, 0x2A78, 0x0904 };

pascal long SetA5 (long newA5) =
    { 0x2F4D, 0x0004, 0x2A5F };
```

Assembly Language

The following assembly-language version is for those of you who are not using a compiler capable of handling multiple word in-line functions, such as MPW C 2.0.2.

```
SetCurrentA5      MOVE.L    A5,4(A7)          ; store old A5 as function result
                  MOVE.L    currentA5,A5      ; set A5 to low memory global
                  RTS

SetA5             MOVE.L    (A7)+,A0          ; save return address
                  MOVE.L    A5,4(A7)          ; store old A5 as function result
                  MOVE.L    (A7)+,A5          ; set A5 to passed value
                  JMP      (A0)
```

A Special Note

Many optimizing compilers, including the MPW C and Pascal compilers, may put the address of a global variable (i.e., a large array or record) into a register **before** your call to SetCurrentA5 or SetA5. If this happens, incorrect references to your global data may be generated. To avoid this problem, you can divide your completion routine into two separate routines: one to set up and restore A5 and one to do the actual completion work. Refer to M.TB.MultifinderMisc and the following code sample for getting the application's A5 while in an interrupt routine. The routines below will be called at interrupt time and must all be in the same code segment; otherwise, jump table references, which use A5, are required.

```
void DoCompletionRoutine(ParmBlkPtr pb, OSErr result)
{
    aGlobal = -1;                /* do some work          */
}

pascal void MyCompletionRoutine
/* This is the actual completion routine that will be called.          */
/* There are two assembly routines not shown, GetOurA5 and              */
/* GetPBPtr. The application's A5 had been saved in front of            */
/* the ParmBlk, which is pointed to in A0. The assembly                  */
/* routine GetOurA5 obtains A5 that was stored in front of the pb        */
/* by the application. This technique is used in M.TB.MultifinderMisc.   */
/* All of these routines must be within the same code segment.          */
{
    long oldA5;

    oldA5 = SetA5(GetOurA5);      /* set to our A5          */
    DoCompletionRoutine(GetPBPtr,result); /* pbPtr is in A0        */
    (void) SetA5(oldA5);          /* restore previous A5    */
}
```

We recommend that you switch over to the new routines as soon as possible, no matter what development system you use.

Further Reference:

- *Inside Macintosh*, Volume V-1, Compatibility Guidelines
- M.PT.Customs
- M.TB.MultifinderMisc