

# APPLICATION NOTE

## **AN458**

### Dual data pointers for '51 family

Author: Peious Yoseph

1996 Dec 18

IC20 Data Handbook

## Dual data pointers for '51 family

AN458

*Author: Peious Yoseph*

### INTRODUCTION

In the course of redesign for manufacturing process upgrade, members of the '51 family have been enhanced with the addition of a second 'data pointer' (DPTR). Software can take advantage of the additional data pointer to both increase speed and reduce code size.

This application note starts by reviewing the operation of the original data pointer and then shows how the new data pointer fits in. Next, physical and software identification of dual data pointer parts is explained. Finally, various software strategies (including assembly language, interrupts and 'C') for taking advantage of the dual data pointers are covered.

### DATA POINTER (DPTR) REVIEW

In the original 8051 architecture, the 16-bit DPTR register provides the means for addressing various portions of the memory space. The instructions which use DPTR to address memory are as follows:

```
MOVX @DPTR,A ;move A to data memory byte addressed by DPTR
MOVX A,@DPTR ;move data memory byte addressed by DPTR to A
```

The 8051 architecture supports separate code (read-only) and data memories with the  $\overline{\text{PSEN}}$  and  $\overline{\text{RD}}$  lines respectively. The MOVX instructions access data memory (i.e.,  $\overline{\text{RD}}$  pin) which is typically RAM or memory-mapped I/O.

```
MOVC A,@DPTR ;move code memory byte addressed by DPTR to A
```

To support read-only constants (ex: data tables, strings) the MOVC instruction accesses code memory (i.e.,  $\overline{\text{PSEN}}$  pin) which is typically ROM or EPROM.

```
JMP @A+DPTR ;branch to address DPTR+A
```

The PC is set equal to DPTR + A (neither DPTR or A is changed). This is often used to implement a 'jump table' in which DPTR points to the base of a table of jump instructions indexed by A.

Instructions which manipulate DPTR are as follows:

```
INC DPTR      ;increment DPTR by 1
MOV DPTR,#16 ;load DPTR with a 16-bit immediate
```

However, DPTR can also be manipulated by virtue of the fact that it, like other CPU registers such as A, B and the PSW, is also accessible as a Special Function Register (SFR). All SFRs are accessible as bytes by a variety of instructions (such as PUSH and MOV) using the 'direct' addressing mode.

The 16-bits of DPTR are mapped into two SFR byte addresses, referred to as DPH (high-byte, address 83H) and DPL (low-byte, address 82H). For example, the instruction:

```
MOV DPTR,#1234H ;load DPTR with 1234H
```

is functionally equivalent (but not timing or code size equivalent) to the sequence:

```
MOV DPH,#12H ;load high-byte of DPTR (83H) with 12H
MOV DPL,#34H ;load low-byte of DPTR (82H) with 34H
```

# Dual data pointers for '51 family

AN458

## DUAL DATA POINTERS

The following '51 family CPUs are being upgraded with dual data pointers:

80C51, 80C52, 80C54, 80C58, 83C51FA, 83C51FB, 83C51FC, 83C575

Contact Philips Semiconductors to determine the dual data pointer conversion status of these and future products.

As shown in Figure 1, these upgraded CPUs still have one logical DPTR, but now two physical data pointers. References to the single logical data pointer (i.e., via instructions that contain 'DPTR' as an operand) are mapped to one of the physical data pointers according to the state of the DPS bit in the AUXR1 SFR (address A2H). Similarly, references to SFR addresses DPH (83H) and DPL (82H) access the currently selected data pointer.

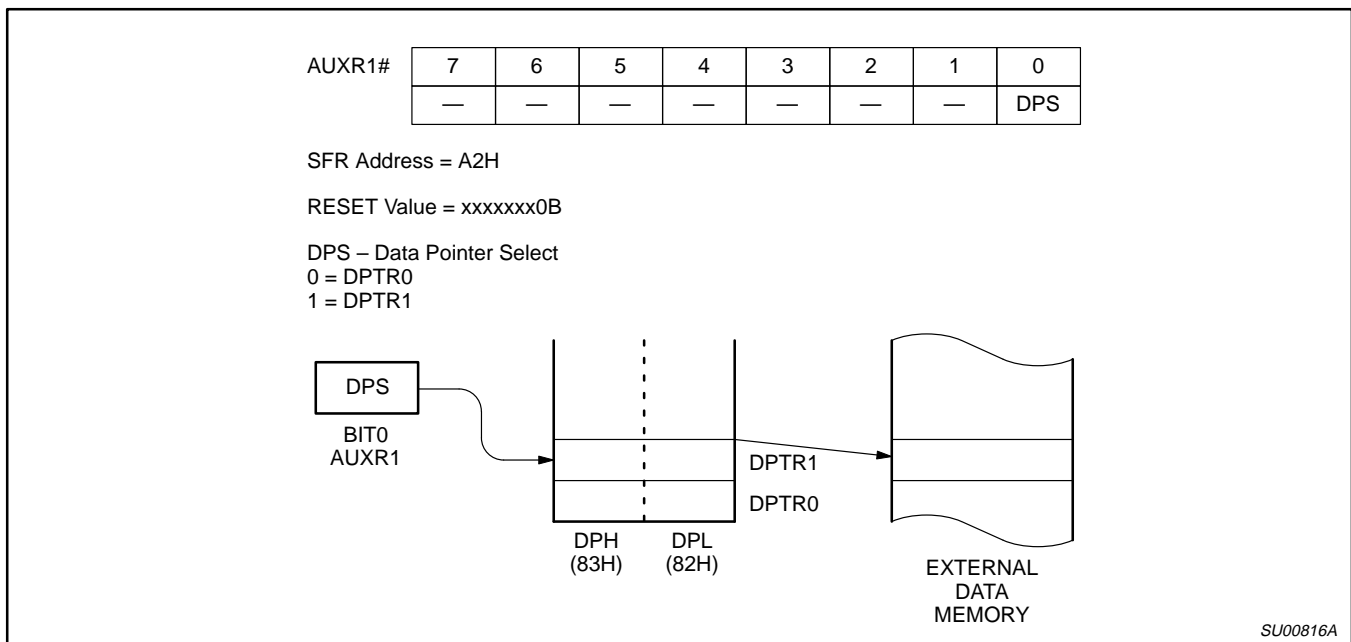


Figure 1. AUXR1# Definition

At RESET, DPS is set to '0'. To allow easy manipulation of DPS, undefined bits in AUXR1 can be written with any value, but always read as 1. Whether a CPU has a single or dual data pointers can be determined at runtime by checking for the presence of the DPS bit.

A special note for 80C51 (and **only** 80C51) users. The new 80C51 with dual data pointers has a slightly different format for AUXR1. As shown in Figure 2, bit 3 serves as a WUPD (Wake-Up from Power Down) mode selection bit. WUPD (along with DPS) is set to '0' at RESET. Applications which use the CPU power-down mode can set WUPD to '1' after RESET to enable the wakeup feature if desired. Bit 2 is defined as '0' to allow easy manipulation of the DPS bit as described in the next section.

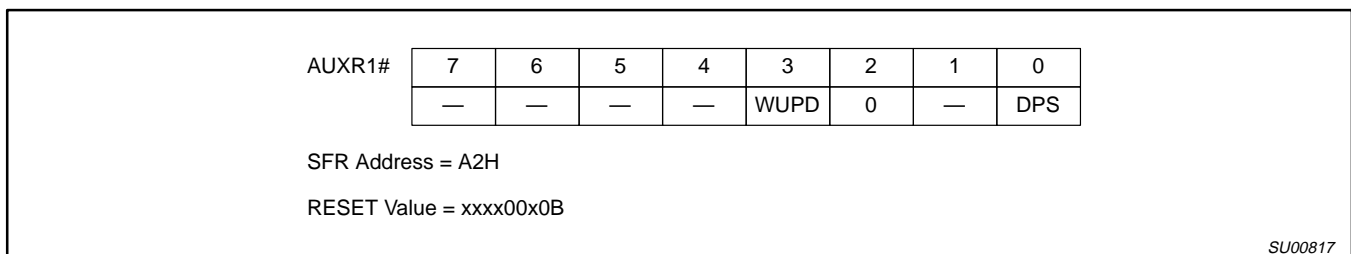


Figure 2. AUXR1# Definition (80C51)

## Dual data pointers for '51 family

AN458

**ASSEMBLY LANGUAGE**

The additional data pointer can be used to speed code execution and reduce code size in a number of ways. For example, many common 'block' operations (such as copy, compare, search, etc.) are well served by using one data pointer as a 'source' pointer and the other as a 'destination' pointer.

```

; Block move using dual data pointers
; Destroys DPTR0, DPTR1, A and PSW
; note: DPS exits opposite of entry state
; unless an extra INC AUXR1 is added
;
00A2          AUXR1 EQU 0A2H
;
0000 909000   MOV  DPTR,#SOURCE   ; address of SOURCE
0003 05A2     INC  AUXR1         ; switch data pointers
0005 90A000   MOV  DPTR,#DEST    ; address of DEST
0008          LOOP:
0008 05A2     INC  AUXR1         ; switch data pointers
000A E0       MOVX A,@DPTR      ; get a byte from SOURCE
000B A3       INC  DPTR         ; increment SOURCE address
000C 05A2     INC  AUXR1         ; switch data pointers
000E F0       MOVX @DPTR,A      ; write the byte to DEST
000F A3       INC  DPTR         ; increment DEST address
0010 70F6     JNZ  LOOP         ; check for 0 terminator
0012 05A2     INC  AUXR1         ; (optional) restore DPS

```

INC is a short (2 bytes) and fast (12 clocks) way to manipulate the DPS bit in the AUXR1 SFR. However, note that the INC instruction does not directly force the DPS bit to a particular state, but simply toggles it.

In simple routines, such as the block move example, only the fact that DPS is toggled in the proper sequence matters, not its actual value. In other words, the block move routine works the same whether DPS is '0' or '1' on entry. Observe that without the last instruction (INC AUXR1), the routine will exit with DPS in the opposite state.

For the 80C51, bit 2 is defined to always read as '0'. Thus, repeated INCs will not propagate past bit 2 and cannot affect the WUPD bit.

In certain situations (such as fault recovery or interrupts) toggling DPS may not be sufficient and DPS must be set to a known value without depending on the current state.

```

53A27E      ANL  AUXR1,#7EH      ; DPS=0
43A201      ORL  AUXR1,#1        ; DPS=1

```

Each of these instructions requires 3 bytes and 24 clocks.

## Dual data pointers for '51 family

## AN458

**INTERRUPTS**

Another way to exploit the additional data pointer is to dedicate it to one or more interrupt handlers. Traditionally (i.e., single data pointer), the handler must save the current value of DPTR on entry (typically to the registers or the stack), load DPTR with the handlers value and then reverse the process on exit.

Now, in a typical foreground/background interrupt scheme, each handler can be given exclusive use of a data pointer. Switching between data pointers using DPS speeds the interrupt response, cuts interrupt overhead and reduces code size by eliminating the instructions needed to share a single data pointer.

More general cases include those in which there are more than two interrupt handlers and/or it is desired to use both data pointers in more than one handler. In these cases, data pointers can be shared using the traditional single data-pointer approach of saving and restoring DPTR. Note that this is typically accomplished via physical SFR access so any instruction that supports the 'direct' addressing mode (such as PUSH or MOV) can be used.

In some cases it may be necessary to save the state of DPS as well. This occurs anytime a routine that can corrupt DPS is nested within (i.e., either by interrupt or subroutine call) a routine that uses DPTR. Should it be necessary, the AUXR1 SFR (containing DPS) can be saved and restored using any instruction that supports the 'direct' addressing mode (such as PUSH or MOV).

Applying these techniques to the block move example, the contents of both data pointers is saved and restored. Since this version of the block move routine happens to exit with DPS equal to the value on entry, the state of DPS (i.e., contents of AUXR1) need not be explicitly saved.

```

;
;
; Block move using dual data pointers
; This version saves & restores data pointer state
; Destroys only A & PSW
;
00A2          AUXR1 EQU 0A2H
;
0000 C083     PUSH DPH           ; save first
0002 C082     PUSH DPL           ; data pointer
0004 909000   MOV DPTR,#SOURCE   ; use it for SOURCE address
0007 05A2     INC AUXR1          ; switch data pointers
0009 C083     PUSH DPH           ; save second
000B C082     PUSH DPL           ; data pointer
000D 90A000   MOV DPTR,#DEST    ; use it for DEST address
0010          LOOP:
0010 05A2     INC AUXR1          ; switch data pointers
0012 E0       MOVX A,@DPTR       ; get a byte from SOURCE
0013 A3       INC DPTR           ; increment SOURCE address
0014 05A2     INC AUXR1          ; switch data pointers
0016 F0       MOVX @DPTR,A       ; write the byte to DEST
0017 A3       INC DPTR           ; increment DEST address
0018 70F6     JNZ LOOP           ; check for 0 terminator
001A D082     POP DPL            ; restore second
001C D083     POP DPH            ; data pointer
001E 05A2     INCV AUXR1         ; switch data pointers
0020 D082     POP DPL            ; restore first
0022 D083     POP DPH            ; data pointer

```

# Dual data pointers for '51 family

AN458

## 'C' COMPILER

The operation of existing 'C' compilers is not affected by the additional data pointer. Assuming the DPS bit is not changed following RESET, existing 'C' programs will continue to use a single data pointer.

In many cases, programs consist of a mixture of 'C' and assembly language. Using the previously mentioned techniques, the assembly language portion (ex: custom library entry, interrupt handler, etc.) can be upgraded to take advantage of the second data pointer without affecting the 'C' compiler's use of the first.

Applications written in 'C' benefit most by upgrading to a compiler that exploits the dual data pointers. As shown in Figure 3, noticeable ( $\approx 15\text{--}30\%$ ) speed-up is obtained simply by modifying the block oriented (copy and move) library routines. Further optimization of code generators and parameter passing conventions offers the potential for even greater performance improvement.

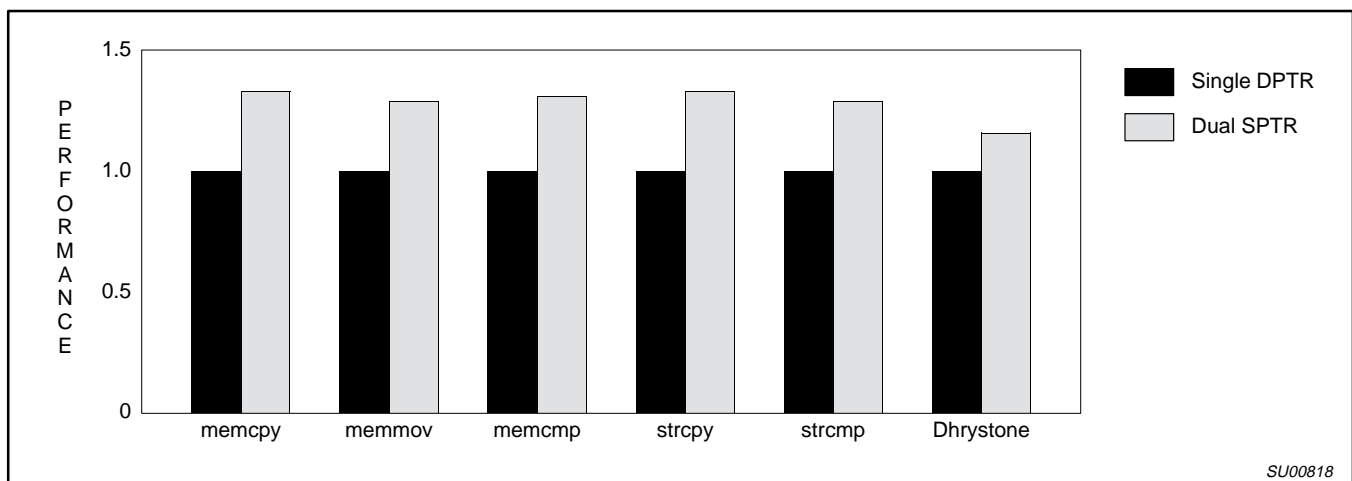


Figure 3. Dual DPTR 'C' Performance Improvement

---

# Dual data pointers for '51 family

AN458

---

## NOTES

---

## Dual data pointers for '51 family

---

AN458

---

Philips Semiconductors and Philips Electronics North America Corporation reserve the right to make changes, without notice, in the products, including circuits, standard cells, and/or software, described or contained herein in order to improve design and/or performance. Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no license or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified. Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

### LIFE SUPPORT APPLICATIONS

Philips Semiconductors and Philips Electronics North America Corporation Products are not designed for use in life support appliances, devices, or systems where malfunction of a Philips Semiconductors and Philips Electronics North America Corporation Product can reasonably be expected to result in a personal injury. Philips Semiconductors and Philips Electronics North America Corporation customers using or selling Philips Semiconductors and Philips Electronics North America Corporation Products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors and Philips Electronics North America Corporation for any damages resulting from such improper use or sale.

---

**Philips Semiconductors**  
**811 East Arques Avenue**  
**P.O. Box 3409**  
**Sunnyvale, California 94088-3409**  
**Telephone 800-234-7381**

Philips Semiconductors and Philips Electronics North America Corporation register eligible circuits under the Semiconductor Chip Protection Act.  
© Copyright Philips Electronics North America Corporation 1996  
All rights reserved. Printed in U.S.A.

*Let's make things better.*

