SECTION 6: MEMORY ACCESS

The High–Speed Microcontroller follows the memory interface convention established for the industry standard 80C51/80C31. Products in the family may vary, so refer to the specific product data sheet for any potential differences. Like the 8051 series, the High–Speed Microcontroller uses two memory segments. These are program memory and data memory. Program memory is read–only and is usually implemented in ROM or EPROM. Data memory is read/write and is commonly implemented in SRAM.

Memory areas can be implemented as either on–chip, off–chip, or a combination. When using devices without internal program memory, or if the maximum address of on–chip program or data memory is exceeded, the device will perform an external memory access using the Expanded memory bus on ports 0 and 2. While serving as a memory bus, port 0 and port 2 do not function as I/O ports, following the standard 8051 convention of addressing external memory. The $\overline{PSEN}$ signal will go active low to serve as a chip enable or output enable when performing a code fetch from external memory. Products with no on–chip program memory such as the DS80C320 will always use the Expanded bus. These devices have no Port 0 latch since the port is dedicated for memory operations. Devices which incorporate on–chip MOVX data memory operate in a similar fashion, except that the $\overline{RD}$ and $\overline{WR}$ signals serve as chip enables when accessing an external SRAM.

Program execution begins at address 0000h. This is the reset vector. Any reset will cause the next program fetch to begin at this location. Subsequent branches and interrupts determine how the memory fetch deviates from sequential addressing. Since all programs begin at 0000h, this will be the beginning address of all program execution. If on–chip program memory is present, program execution will begin at internal location 0000h, otherwise external program memory will be used.

INTERNAL PROGRAM MEMORY

Some members of the High–Speed Microcontroller family incorporate internal EPROM or ROM for program storage. On–chip program memory begins at address 0000h and is contiguous through the amount of on–chip memory. Exceeding the maximum address of on–chip memory will cause the device to perform an external memory access using the Expanded memory bus on ports 0 and 2. For example, if the on–chip program memory is 16KB, then it lies between 0000h and 3FFFh in a contiguous area. Thus a fetch at program memory location 4000h would be directed to the Expanded bus. Restricting memory operations within the on–chip memory allows ports 0 and 2 to be used for general purpose I/O. For more information concerning memory size for a specific device, consult the specific data sheet.

The High–Speed Microcontroller family was designed to be compatible with industry standard 87C51FB programming tools. A number of third–party device programmers are available which support Dallas Semiconductor products. In addition, Dallas Semiconductor manufactures the DS87000 Microcontroller Programmer, specifically designed for EPROM–based members of the High–Speed Microcontroller family.

INTERNAL DATA MEMORY

Some members of the High–Speed Microcontroller family incorporate internal SRAM for additional data storage. This memory is addressed via MOVX commands, and is in addition to the 256 bytes of scratchpad memory. On–chip data memory begins at address 0000h and is contiguous through the amount of on–chip memory. Exceeding the maximum address of on–chip memory will cause the device to perform an external memory access using the Expanded memory bus on ports 0 and 2. For example, if the on–chip program memory is 1KB, then it lies between 0000h and 03FFh in a contiguous area. Thus a MOVX instruction affecting memory location 0400h would be directed to the Expanded bus.

Another advantage of internal data memory is that it guarantees a 2 machine cycle data memory access. This data can be made nonvolatile on the DS87C530 through the use of an external battery. Restricting memory operations within the on–chip memory allows ports 0 and 2 to be used for general purpose I/O. For more information concerning memory size for a specific device, consult the corresponding data sheet.

Upon a power–on reset, the internal data memory area is disabled and transparent to the system map. Any memory access between 0000h and FFFFh will be directed to the Expanded bus. This allows the device to remain drop–in compatible with existing 87C52 designs. To enable the internal SRAM area, software must configure the Data Memory Enable bits DME1, DME0 (PMR.1–0). The three memory configurations shown in Table 6–1 are supported to allow either external data memory access via the expanded bus, internal data

memory access, or read–only access to the EPROM System Control Byte. Note that these bits are cleared after a reset, so access to the internal data memory is prohibited until these bits are modified. The contents of internal data memory are not affected by the changing of the Data Memory Enable bits.

DATA MEMORY ACCESS CONTROL Table 6–1

DME1	DME0	DATA MEMORY ADDRESS RANGE	DATA MEMORY LOCATION
0	0	0000h–FFFFh	External Data Memory (default)
0	1	0000h–03FFh 0400h–FFFFh	Internal Data Memory External Data Memory
1	0	Reserved	Reserved
1	1	0000h–03FFh 0400h–FFFBh FFFCh FFFDh–FFFFh	Internal Data Memory Reserved System Control Byte (Read only) Reserved

ROMSIZE FEATURE

Members of the High–Speed Microcontroller family which incorporate internal program memory allow the system to dynamically vary the on–chip memory size. This permits the device to reconfigure the upper limit of on–chip memory, allowing a portion of the memory to be mapped off–chip. The size of on–chip memory can vary from 0KB to the full range of memory, allowing the device to behave like a device with less on–chip memory.

This feature has two primary uses. In the first instance, it allows the device to act as a bootstrap loader for a Flash memory or nonvolatile SRAM (NVSRAM). The internal program memory can contain a bootstrap loader, which can program the external memory device. Secondly, this method can be used to increase the amount of available program memory from 64KB to 80KB without bank–switching.

The maximum amount of on–chip memory is selected by configuring the ROM Size Select register bits RMS2, RMS1, RMS0 (ROMSIZE.2–0). The modification of the ROMSIZE register must be followed by a 2 machine cycle delay, such as executing two NOP instructions, before jumping to the new address range. Interrupts must be disabled during this operation, because a jump to the interrupt vector during the changing of the memory map can cause erratic results. In addition, modification of the ROMSIZE register must be done from a location that will be valid both before and after the on–chip memory configuration. If off–chip memory access is planned, it is recommended that ports 0 and 2 not be used as general purpose I/O, as their state will be disturbed by the memory operations. The settings for the ROM Size Select register are shown in Table 6–2. Note that the memory configurations shown are not available on all devices.

ROMSIZE REGISTER SETTINGS Table 6–2

RMS2	RMS1	RMS0	Max. On–chip ROM
0	0	0	0KB
0	0	1	1KB
0	1	0	2KB
0	1	1	4KB
1	0	0	8KB
1	0	1	16KB
1	1	0	32KB
1	1	1	64KB

After reset, a device with internal program memory will reset the ROMSIZE bits to their default setting. This will be the maximum amount of on–chip memory for that device. The procedure to reconfigure the amount of on–chip memory is as follows:

1. Jump to a location in program memory that will be unaffected by the change,
2. Disable interrupts by clearing the EA bit (IE.7),
3. Write AAh to the Timed Access Register (TA;C7h),
4. Write 55h to the Timed Access Register (TA;C7h),
5. Modify the ROM Size Select bits (RMS2–RMS0),

6. Delay 2 machine cycles (2 NOP instructions),
7. Enable interrupts by setting the EA bit (IE.7).

If the 0KB of internal program memory setting is selected, extra precautions must be taken. In this case, it will be necessary to duplicate the interrupt vector table in external program memory. This is because the interrupt vector table is located in the lower 1KB of memory, and the device will automatically redirect any fetches from the interrupt vector table to external memory. Care must be exercised when assembling or compiling the program so that all the modules are located at the correct starting address, including the interrupt vector table.

PROGRAM MEMORY INTERCONNECT

The program memory interconnect scheme for the High–Speed Microcontroller family is shown in Figure 6–1. This example uses the DS80C320 and one 32K x 8 EPROM. The Program Store Enable ($\overline{PSEN}$) signal is used to provide an output enable to the EPROM. It can also be used to provide a chip enable, but this produces less favorable timing. The address LSB and data are multiplexed on port 0, and the address MSB is provided on port 2. An external latch, shown in the diagram as a 74F373, is used to latch the lower byte of the address to the memory device. The Address Latch Enable (ALE) signal controls the timing of the latch so that the operation is performed in the proper sequence. The signals and relative timing for a program access are shown in Figure 6–2.

When implementing a high speed memory interface, the F series (or faster) logic should be used. HC logic will have worst case propagation delays that are too long. Specifications for all devices should be checked. More information on memory interface timing can be found in Application Note 57, DS80C320 Memory Interface Timing, and Application Note 85, High Speed Microcontroller Interface Timing.

The first product in the family, the DS80C320, provides an extremely high speed interface to external ROM or EPROM. This assures that the user can use the slowest, and least expensive, memory device for a given crystal speed. The DS80C320 provides very fast slew rates, but controls ringing and overshoot. Fast slew rates allow the maximum possible time for memory access. In most cases, however, these aspects will be transparent to the user. Refer to the electrical specifications for exact timing of each product.

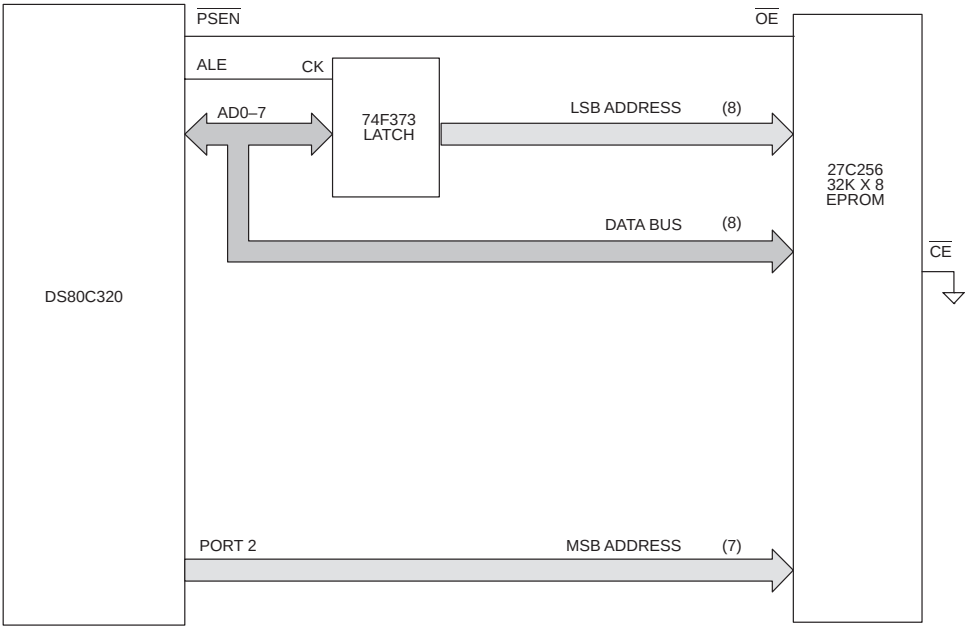
DATA MEMORY INTERCONNECT

As described in Section 4, the High–Speed Microcontroller provides a small amount of RAM mapped as registers for on–chip direct access. This is not considered data memory and does not fall into the memory map. Systems that require more RAM or memory mapped peripherals must use the data memory area. This segment is a 64KB space located between 0000h and FFFFh. It is reached using the MOVX instruction. Any use of this instruction automatically accesses the data area. Although, the original 8051 convention placed all data memory off–chip, many members of the High–Speed Microcontroller family contain some data memory on–chip.

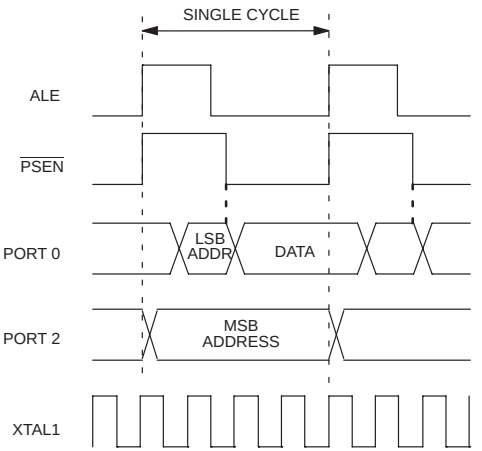
From a software standpoint, the physical location of the data area is not relevant because the same instructions are used. Like the program segment, if software accesses a data address that is above the on–chip data area, this access will automatically be routed to the Expanded bus. Thus data or peripherals that are off–chip can be used in conjunction with on–chip memory by selecting addresses that do not overlap. As an example, if the microcontroller has 1KB of on–chip data memory, then a MOVX instruction at location 0400h will be directed off–chip via the Expanded bus.

The physical connection of off–chip data memory is shown in Figure 6–3. This illustrates a DS80C320 with interfaced with an 8K SRAM. The data memory map begins at address 0000h since the DS80C320 has no on–chip data memory. A similar interconnection scheme would be implemented if a device with internal data memory, such as the DS87C520 would be used. Note that any external memory that overlapped the range of on–chip data memory would not be used.

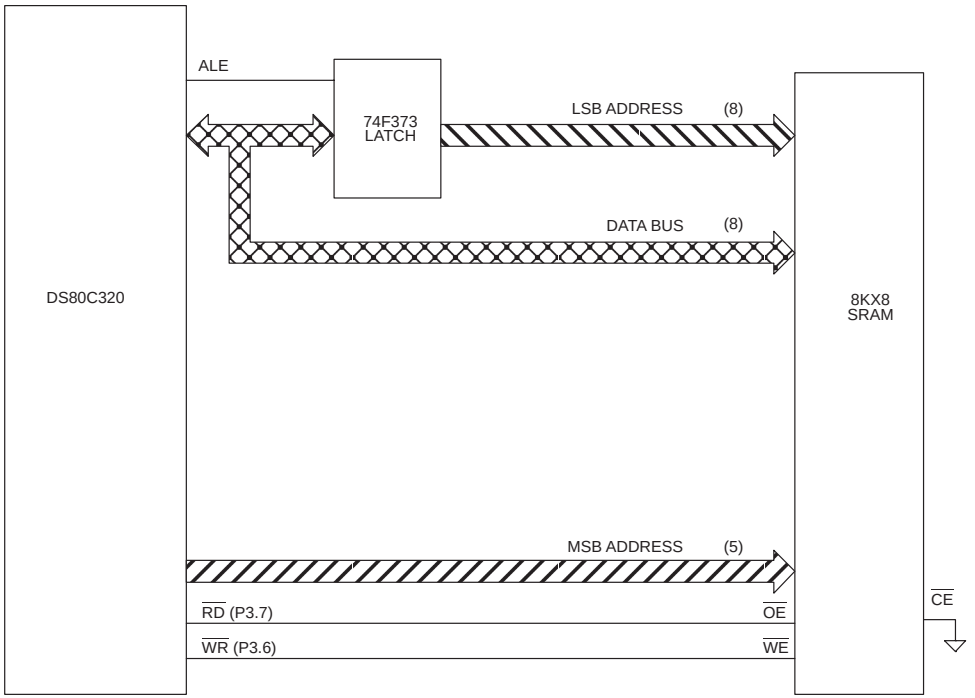
PROGRAM MEMORY INTERFACE Figure 6–1



PROGRAM MEMORY SIGNALS Figure 6–2



DATA MEMORY INTERFACE Figure 6–3



DATA MEMORY ACCESS

As mentioned above, the High–Speed Microcontroller uses the MOVX instruction for data memory access. This includes off–chip RAM and memory mapped peripherals needing read/write access. Several aspects of the MOVX operation have been enhanced as compared to the original 8051. The principal improvements are in the areas of the MOVX timing and the Data Pointer.

The MOVX instruction is used to generate read/write access to off–chip address locations. It has several addressing modes. The MOVX can be used to reach a 256 byte block by using the MOVX @ Ri command. This instruction uses the value in the designated working register to address one of 256 locations. The upper byte of the address is supplied by the value in the Port 2 latch. A second way to access data is the Data Pointer (DPTR). This 16–bit register provides an absolute address for data memory access. 16–bits cover the entire 64KB area. Thus the DPTR serves as a pointer to memory. Using the DPTR, the relevant instruction is MOVX @DPTR.

The original 8051 contained one DPTR. While this provides access to the entire memory area, it is difficult to move data from one address to another. The High–Speed Microcontroller provides two Data Pointers. Thus software can load both a source and a destination address. The MOVX instruction will use the active pointer to direct the off–chip address. The active pointer is selected by the Data Pointer Select (DPS).

The Data Pointers are called DPTR0 and DPTR1. DPTR0 is located at SFR addresses 82h and 83h. These are the locations used by the original 8051. No modification of standard code is needed to use DPTR0. The new DPTR is located at SFR 84h and 85h. The Data Pointer Select bit (DPS) chooses the active pointer and is located at the LSb of the SFR location 86h. No other bits in register 86h have any effect and are set to 0. When DPS is set to 0, the DPTR0 is active. When set to 1, DPTR1 is used.

The user switches between data pointers by toggling the DPS bit (LSb of register 86h). The INC instruction is the fastest way to accomplish this. All DPTR–related instructions use the currently selected DPTR for any activity. Therefore only one instruction is required to switch from a source to a destination address. Using the Dual Data Pointer saves code from needing to save source and destination addresses when doing a block move. Once loaded , the software simply switches between DPTR0 and DPTR1. Sample code listed below illustrates the saving from using the dual DPTR. The relevant register locations are summarized as follows.

DPL	82h	Low byte original DPTR
DPH	83h	High byte original DPTR
DPL1	84h	Low byte new DPTR
DPH1	85h	High byte new DPTR
DPS	86h	DPTR Select (LSb)

The example program listed below was original code written for an 8051 and requires a total of 1869 machine cycles on the DS80C320. This takes 299 μs to execute at 25 MHz. The new code using the Dual DPTR requires only 1097 machine cycles taking 175.5 μs. The Dual DPTR saves 772 machine cycles or 123.5 μs for a 64 byte block move. Since each pass through the loop saves 12 machine cycles when compared to the single DPTR approach, larger blocks gain more efficiency using this feature.

A typical application of the Dual Data Pointer is moving data from an external RAM to a memory mapped display. Another application would be to retrieve data from a stored table, process it using a software algorithm, then store the result in a new table.

64 BYTE BLOCK MOVE WITH DUAL DATA POINTER

```
; SH and SL are high and low byte source address.
; DH and DL are high and low byte of destination address.
; DPS is the data pointer select. Reset condition is DPS=0, DPTR0 is selected.
                                                                    # CYCLES
DPS     EQU 86h             ; TELL ASSEMBLER ABOUT DPS
MOV     R5, #64             ; NUMBER OF BYTES TO MOVE           2
MOV     DPTR, #DHDL         ; LOAD DESTINATION ADDRESS          3
INC     DPS                 ; CHANGE ACTIVE DPTR                2
MOV     DPTR, #SHSL         ; LOAD SOURCE ADDRESS               2

MOVE:
; THIS LOOP IS PERFORMED THE NUMBER OF TIMES LOADED INTO R5, IN THIS EXAMPLE 64
MOVX    A, @DPTR            ; READ SOURCE DATA BYTE             2
INC     DPS                 ; CHANGE DPTR TO DESTINATION        2
MOVX    @DPTR, A            ; WRITE DATA TO DESTINATION         2
INC     DPTR                ; NEXT DESTINATION ADDRESS          3
INC     DPS                 ; CHANGE DATA POINTER TO SOURCE     2
INC     DPTR                ; NEXT SOURCE ADDRESS               3
DJNZ    R5, MOVE            ; FINISHED WITH TABLE?              3
```

64 BYTE BLOCK MOVE WITHOUT DUAL DATA POINTER

```
; SH and SL are high and low byte source address.
; DH and DL are high and low byte of destination address.
                                                                    # CYCLES
MOV     R5, #64d            ; NUMBER OF BYTES TO MOVE           2
MOV     DPTR, #SHSL         ; LOAD SOURCE ADDRESS               3
MOV     R1, #SL             ; SAVE LOW BYTE OF SOURCE           2
MOV     R2, #SH             ; SAVE HIGH BYTE OF SOURCE          2
MOV     R3, #DL             ; SAVE LOW BYTE OF DESTINATION      2
MOV     R4, #DH             ; SAVE HIGH BYTE OF DESTINATION     2

MOVE:
; THIS LOOP IS PERFORMED THE NUMBER OF TIMES LOADED INTO R5, IN THIS EXAMPLE 64
MOVX    A, @DPTR            ; READ SOURCE DATA BYTE             2
MOV     R1, DPL             ; SAVE NEW SOURCE POINTER           2
MOV     R2, DPH             ;                                   2
MOV     DPL, R3             ; LOAD NEW DESTINATION              2
MOV     DPH, R4             ;                                   2
MOVX    @DPTR, A            ; WRITE DATA TO DESTINATION         2
INC     DPTR                ; NEXT DESTINATION ADDRESS          3
MOV     R3, DPL             ; SAVE NEW DESTINATION POINTER      2
MOV     R4, DPH             ;                                   2
MOV     DPL, R1             ; GET NEW SOURCE POINTER            2
MOV     DPH, R2             ;                                   2
INC     DPTR                ; NEXT SOURCE ADDRESS               3
DJNZ    R5, MOVE            ; FINISHED WITH TABLE?              3
```

DATA MEMORY TIMING

Data memory timing refers to the execution of the MOVX instruction. This instruction includes a program fetch memory access, then a read or write memory access. The program fetch for a MOVX instruction is no different from any other instruction. The unique timing occurs for the second memory operation when data is accessed.

As described in Section 5, the High–Speed Microcontroller uses four oscillator clocks for each machine cycle. A machine cycle involves one memory access. Generally, an instruction using two memory accesses would be a two machine cycle instruction (except for branches, MUL, DIV, INC DPTR, MOVC, and MOVX). The MOVX instruction is unique in that the user determines the time allowed for a data memory access. This feature is called the Stretch MOVX instruction.

The High–Speed Microcontroller allows the application software to adjust the speed of data memory access. The microcontroller is capable of performing the MOVX in as little as two machine cycles. Since one machine cycle is used for the program fetch, this leaves one machine cycle to perform the actual data memory access. However, this value can be adjusted as needed so that both fast memory and slow memory or peripherals can be accessed with no glue logic. Even in high–speed systems, it may not be necessary to perform data memory access at full speed. In addition, there are a variety of slower memory mapped peripherals such as LCD displays or UARTs.

When using a MOVX instruction, the user controls the time for which a read or write strobe is kept active. Setup and hold times are also adjusted. Thus the Stretch value will be selected to provide a long enough memory strobe to satisfy the access time of the target device.

The Stretch MOVX is controlled by a value in a special function register described below. This allows the user to select a stretch value between zero and seven. A Stretch of zero will result in a two machine cycle MOVX. This leaves one machine cycle to actually read or write data. A Stretch of seven will result in a MOVX of nine cycles. The time is added to the middle of the memory strobe, creating a very long read or write cycle. The Stretch value can be changed dynamically under software control depending on the type of memory or peripheral to be accessed.

On reset, the Stretch value will default to a one, resulting in a three cycle MOVX. Therefore, data memory access will not be performed at full speed. This is a convenience to existing designs that may not have fast RAM in place. When maximum speed is desired, the software should select a Stretch value of zero. Note that faster RAMs will be needed. When using very slow RAM or peripherals, a larger stretch value can be selected. Note that this affects data memory only and the only way to slow program memory (ROM) access is to use a slower crystal.

Using a Stretch value between one and seven results in a wider read/write strobe allowing more time for memory/peripherals to respond. The microcontroller stretches the read/write strobe and all related timing. The full speed access is shown in Figure 6–4. Note that this is not the reset default case. A three cycle MOVX is shown in Figure 6–5A. This is the reset default condition. To modify the MOVX timing, the Stretch value in the Clock Control register described below must be changed. Figure 6–5B shows the timing for a four cycle MOVX (Stretch=2).

Table 6–1 below shows the resulting strobe widths for each Stretch value. The memory stretch is implemented using the Clock Control Special Function Register at SFR location 8Eh. The stretch value is selected using bits CKCON.2–0. In the table, these bits are referred to as M2 through M0. Note that the Stretch time can be dynamically varied, allowing fast RAMs but slow peripherals. The first stretch allows the use of common 120 ns or 150 ns RAMs without dramatically lengthening the memory access.

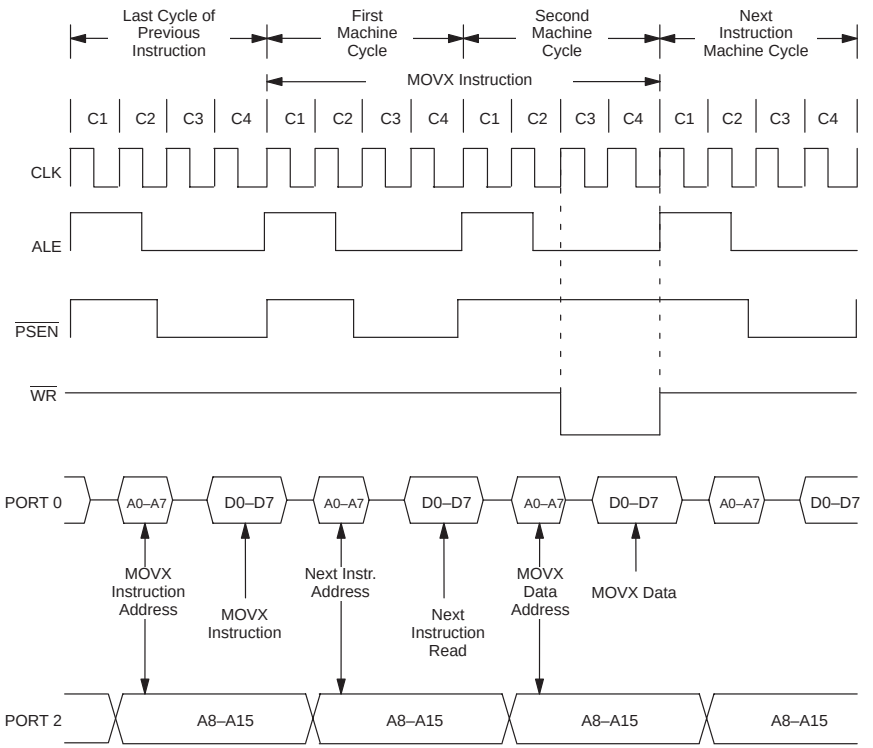
Note that the first Stretch value does not follow the pattern of adding four clocks to the strobe. This is because the first Stretch uses one clock to create additional set–up and one clock to create additional hold time. Systems using a Stretch cycle of zero are presumed to be fast enough or to be running at a slower clock speed. Since the Stretch is based on crystal timing, the resulting pulse widths must be viewed on the basis of the real system timing.

DATA MEMORY CYCLE STRETCH VALUES Table 6–1

CKCON.2–0			MEMORY CYCLES	$\overline{RD}$ OR $\overline{WR}$ STROBE WIDTH		
M2	M1	M0		IN CLOCKS	t @ 25 MHz	t @ 12 MHz
0	0	0	2	2	80 ns	167 ns
0	0	1	3 (default)	4	160 ns	333 ns
0	1	0	4	8	320 ns	667 ns
0	1	1	5	12	480 ns	1000 ns
1	0	0	6	16	640 ns	1333 ns
1	0	1	7	20	800 ns	1667 ns
1	1	0	8	24	960 ns	2000 ns
1	1	1	9	28	1120 ns	2333 ns

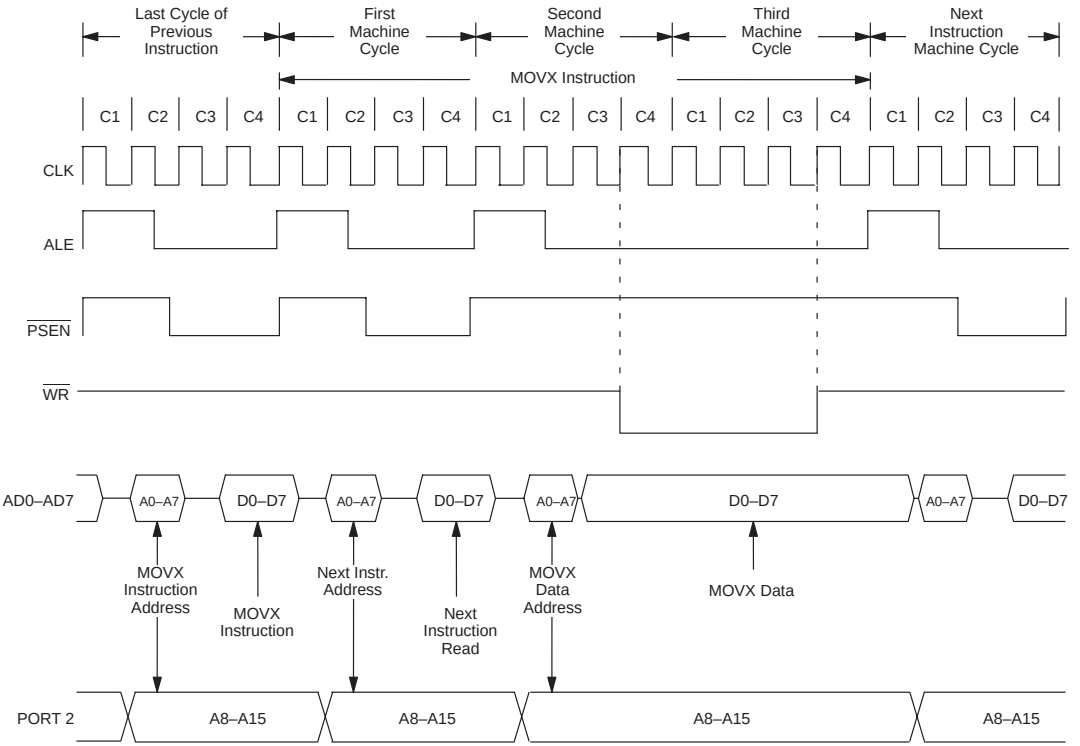
Note: These numbers represent nominal values. Actual timing may vary slightly.

FULL SPEED MOVX INSTRUCTION Figure 6–4



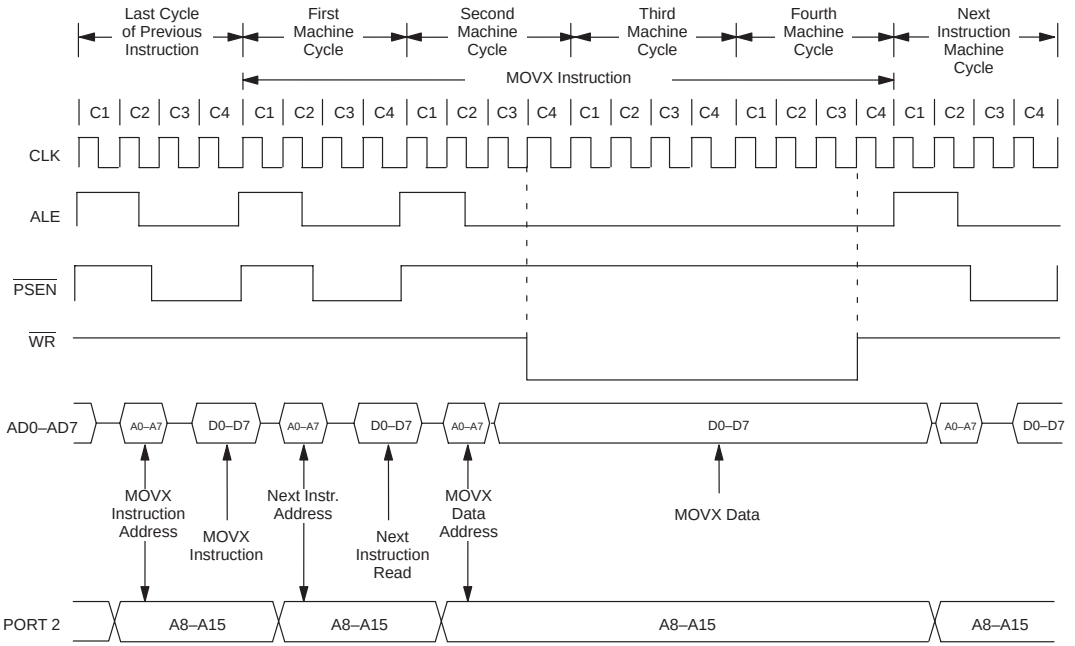
DATA MEMORY WRITE (2 CYCLE, STRETCH=0)

THREE CYCLE MOVX INSTRUCTION Figure 6–5a



THREE CYCLE DATA MEMRORY WRITE (RESET DEFAULT)
STRETCH VALUE=1

FOUR CYCLE MOVX INSTRUCTION Figure 6–5b



FOUR CYCLE DATA MEMORY WRITE
STRETCH VALUE=2