

## EXERCISE 2 - ADDRESSING MODES

### Introduction

This exercise describes the function of the Address Generation Unit (AGU) and its main objective is to illustrate:

- The DSP56300 Special Addressing Modes
- The Address Register Indirect operation
- The function of the Modifier Registers
- Modulo and Reverse Carry Addressing techniques

### Technical Considerations

**AGU Architecture.** The AGU provides all the addressing modes required by DSP algorithms, such as modulo addressing for circular buffer generation and bit-reverse arithmetic for FFT butterflies. The AGU is divided into two halves, each of which has an address arithmetic logic unit (ALU) and three sets of registers. There are a total of eight independent pointer registers (R), eight offset registers (N) and eight modifier registers (M). All addresses are 24-bits supporting 16-Mwords in each memory space. Two address arithmetic units permit computation and generation of two 24-bit data addresses for dual operand access in every clock cycle.

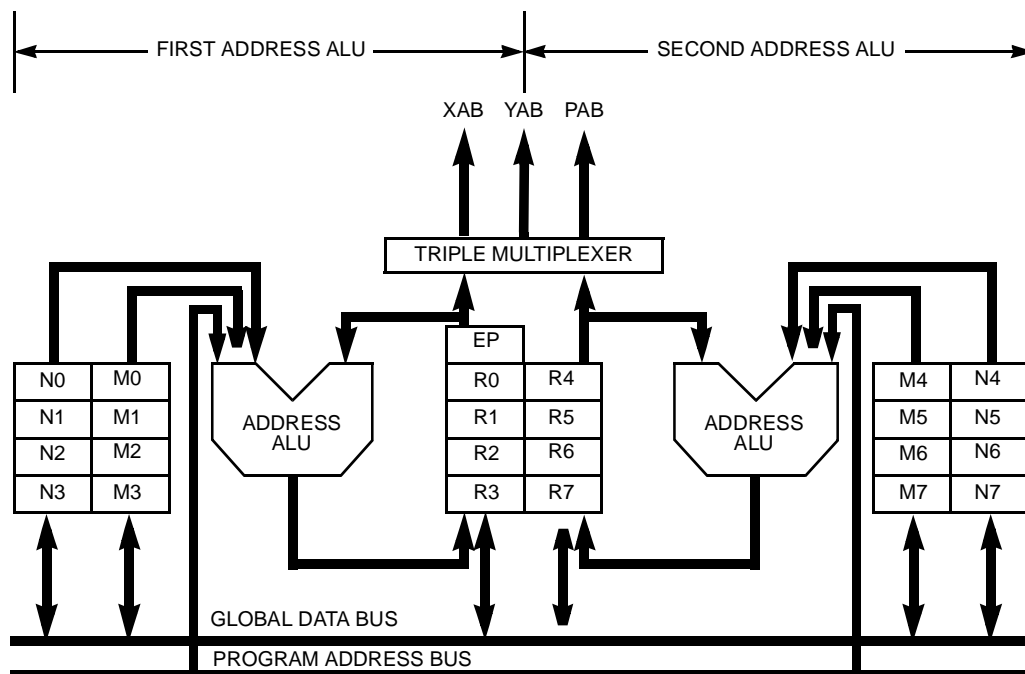
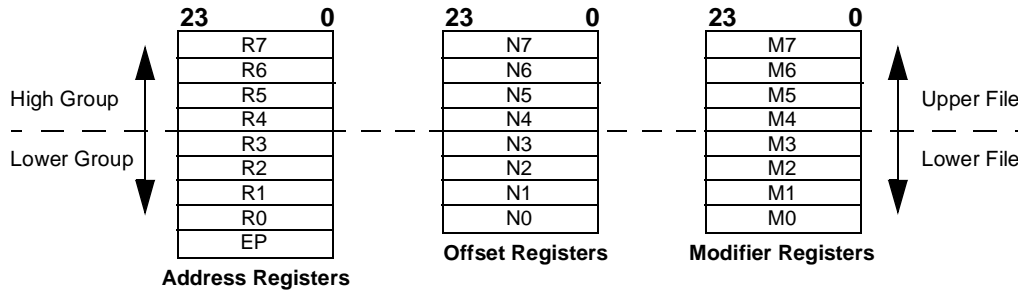


Figure 5: AGU Architecture

For more details please refer to [1], section 4.

**The AGU Programming Model.** The programmer's view of the AGU is three sets of eight registers, which can be used as temporary data registers and indirect memory pointers. Automatic updating is available when using address register indirect addressing.



**Figure 6: Address Registers**

Each address register  $R_n$  has an associated offset register  $N_n$  and an associated modifier register  $M_n$ . The  $R_n$  registers are used as address pointers to locate data operands in memory, and can be programmed for linear addressing, modulo addressing (regular or multiple wrap-around), and bit-reverse addressing. The  $N_n$  registers are used to provide an offset value for offset updating of the address registers. The  $M_n$  registers select the type of address arithmetic to be performed when an address register is updated. The EP register (when enabled) is used to point to the stack extension in data memory and is referenced implicitly by instructions such as DO, JSR, RTI, etc. or directly by the MOVEC instruction. For a more detailed description on how to use the stack extension mode of operation, please refer to Section 6.3.5 in [1].

**Address Modifier Types.** As previously mentioned, the address modifier ( $M_n$ ) defines the type of address arithmetic to be performed, and allows the user to create various data structures in memory, such as FIFOs, delay lines, circular buffers, stacks, and bit-reversed FFT buffers etc. The table below shows how the contents of the modifier register ( $M_n$ ) select the type of address arithmetic to be performed. For more details please refer to [1], Section 4.

Modifier $M_n$	Address Calculation Arithmetic
XX0000	Reverse-Carry (Bit-Reverse)
XX0001	Modulo 2
XX0002	Modulo 3
:	:
XX7FFE	Modulo 32767 ( $2^{15}-1$ )
XX7FFF	Modulo 32768 ( $2^{15}$ )
XX8001	Multiple Wrap-Around Modulo 2
XX8003	Multiple Wrap-Around Modulo 4
XX8007	Multiple Wrap-Around Modulo 8
:	:
XX9FFF	Multiple Wrap-Around Modulo $2^{13}$
XXBFFF	Multiple Wrap-Around Modulo $2^{14}$
XXFFFF	Linear (Modulo $2^{24}$ )

**Notes:                      XX means don't care**

**All other combinations are reserved**

The addressing modes specify whether the operands are in registers and/or memory locations, and provide the specific address of the operands. The DSP56300 core provides four different addressing modes: register direct, address register indirect, special and PC relative.

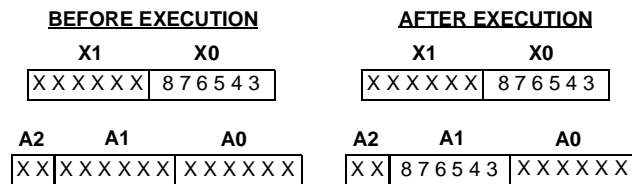
## DSP56300 Programming Exercises

The following sections attempt to illustrate the various addressing modes, it is recommended that the `ex7_main.asm` is made available as you read this document as the code will illustrate the examples discussed.

**Register Direct Mode.** This mode specifies that the operand is in one or more of the 10 data ALU registers (A2,A1,A0,B2,B1,B0,X1,X0,Y1,Y0), 24 address registers (R0-R7,N0-N7,M0-M7) or 7 control registers (OMR,SR,PC,VBA,LA,LC,SP).

**Example:** Move the contents of the 24-bit X0 data input register to the 24-bit A1 accumulator register.

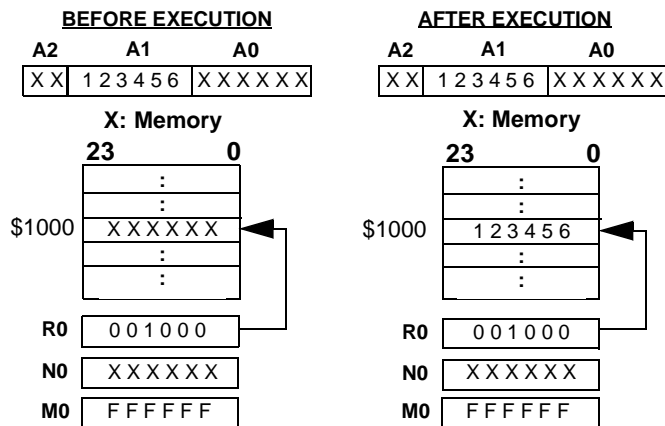
**Figure 7: move x0,a1**



**Address Register Indirect Modes.** These addressing modes specify an address register (Rn) to point to an operand stored in memory. They can also specify an address calculation to be performed either pre or post instruction execution. Each address register Rn is associated with an offset register Nn and a modifier register Mn. The Nn register contains an offset value which can be added to Rn to update its contents. The Mn register specifies the type of address arithmetic to be performed when Rn is updated. Mn is set to \$FFFFFF upon reset to specify linear address arithmetic.

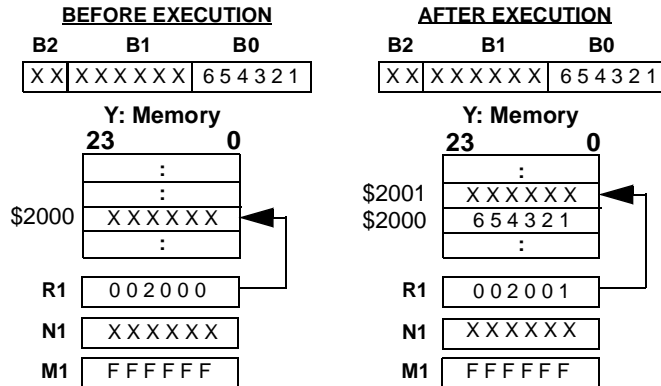
**No Update (Rn) - Example:** Transfer the contents of accumulator register A1 to the X-Memory location pointed to by address register R0.

**Figure 8: move a1,x:(r0)**



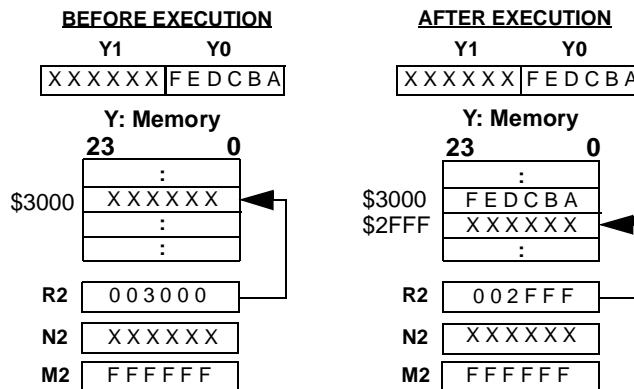
**Postincrement by one (Rn)+ - Example:** Transfer the contents of accumulator register B0 to the Y-Memory location pointed to by address register R1. Once the transfer is complete, R1 is incremented by one.

**Figure 9: move b0,y:(r1)+**



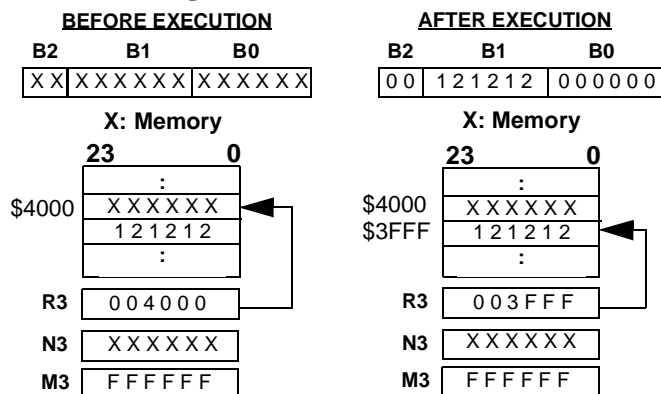
**Postdecrement by one (Rn)- - Example.** Transfer the contents of data register Y0 to the Y-Memory location pointed to by address register R2. Once the transfer is complete, R2 is decremented by one.

**Figure 10: move y0,y:(r2)-**



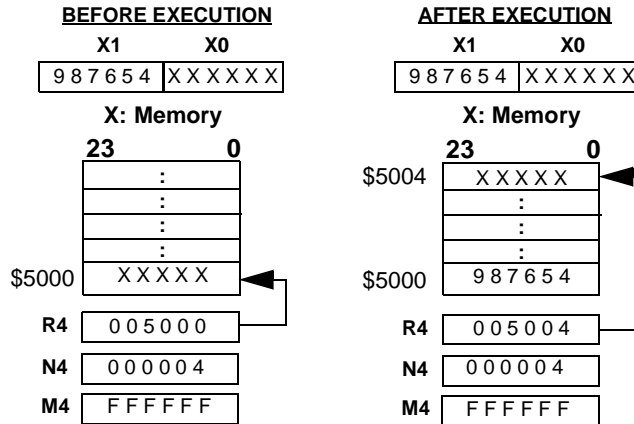
**Predecrement by one -(Rn) - Example.** Predecrement address register R3 by one and transfer the X-Memory location pointed to by the decremented address register R3 to accumulator register B

**Figure 11: move x:-(r3),b**



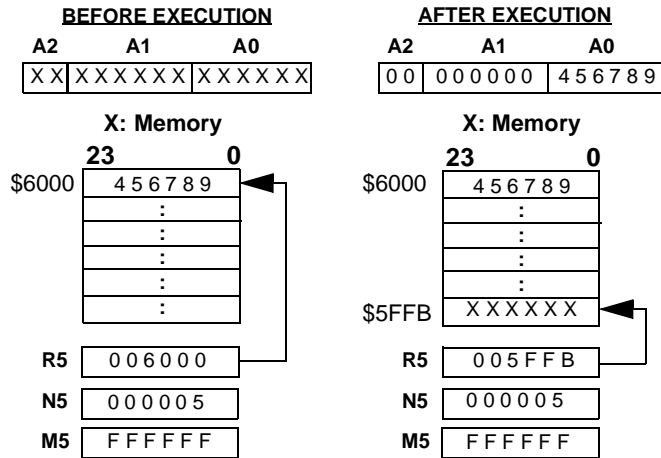
**Postincrement by Offset (Rn)+Nn - Example .** Transfer the contents of data input register X1 to the X-Memory location pointed to by address register R4. Once the transfer is complete, R4 is updated by adding the offset contained in offset register N4 to the contents of R4.

**Figure 12: move x1,x:(r4)+n4**



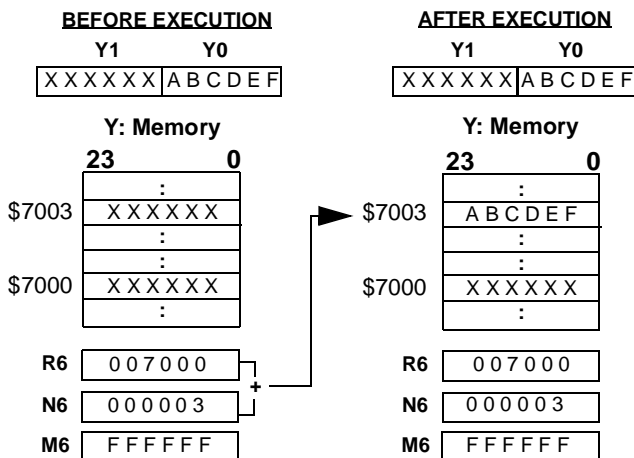
**Postdecrement by Offset (Rn)-Nn - Example.** Transfer the contents of the X-Memory location pointed to by address register R5 to accumulator A0. Once the transfer is complete, R5 is updated by subtracting the offset contained in offset register N5 from the contents of R5.

**Figure 13: move x:(r5)-n5,a0**



**Indexed by Offset (Rn+Nn) - Example.** Transfer the contents of data input register Y0 to the X-Memory location pointed to by the summation of the contents of the address register R6 and the offset register N6. Note that address register R6 is not updated.

**Figure 14: move y0,x:(r6+n6)**

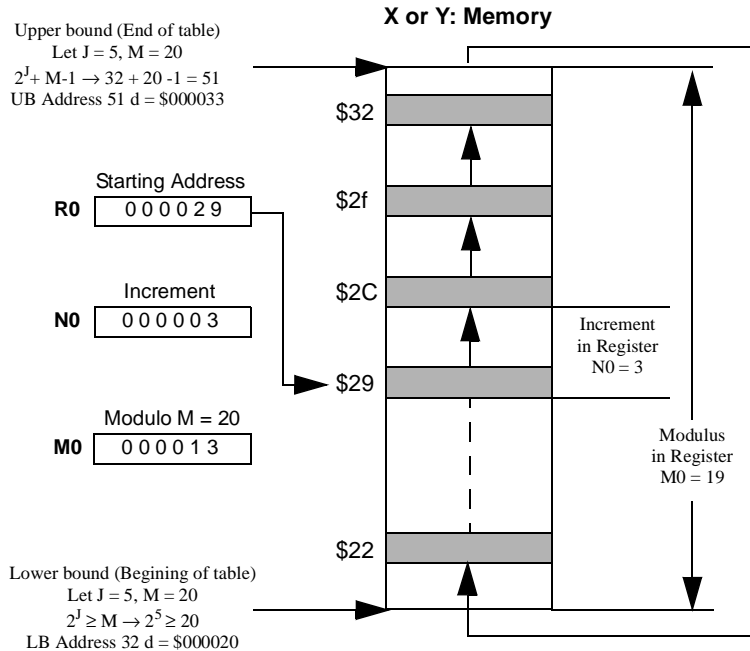


**Modifier Register Usage.** The eight 24-bit modifier registers (M0-M7) specify the type of address arithmetic to be performed for addressing mode calculations, or can be used for general-purpose storage. The address ALU supports linear, modulo and reverse-carry arithmetic for all address register indirect modes.

**Modulo Addressing (Mn = Modulus-1).** For modulo arithmetic, the contents of Mn specifies the modulus i.e. circular buffer/table size. The following diagram illustrates the procedure taken for modulo register set-up, for a buffer/table size of 20 elements and an increment offset (Nn) of 3.

**Modulo Register Setup:**

1. MODULUS = M  
Modifier register Mn = M-1
2. Beginning of Table  
Lower bound =  $2^J \geq M$
3. End of Table  
Upper bound =  $2^J + M - 1$
4. Starting point within Table  
Lower bound  $\leq$  Address register Rn  $\leq$  Upper bound
5. Desired Increment (if any)  
Offset register Nn = Increment  $\leq$  M



**Figure 15: Modulo Addressing**

**Reverse-Carry Addressing (Mn = \$000000).** Reverse carry is selected by setting the modifier register to zero. The address modification is performed in hardware by propagating the carry in the reverse direction i.e. from the MSB to the LSB. Reverse carry is equivalent to bit reversing the contents of Rn (i.e. redefining the MSB as the LSB, the next MSB as bit 1, etc.) and the offset value, Nn, adding normally, and then bit reversing the result. If the + Nn addressing mode is used with this address modifier and Nn contains the value  $2^{(k-1)}$  (a power of two), this addressing modifier is equivalent to bit reversing the k LSBs of Rn, incrementing Rn by 1, and bit reversing the k LSBs of Rn again.

To illustrate the address reordering technique, consider each element of the input sequence labelled 'orgdatar' in the file 'ex2\_main.asm' and its associated binary base address \$40=1000000. For an 8-point (eight-element) input data sequence (i.e. orgdatar) the three least significant bits (LSB) of the associated binary addresses are 000,001,002,...,111 respectively. To reorder the addresses of the data input sequence, the m LSBs ( $2^{(k-1)} = 3$ ) of the address of each sequence element must be 'Bit-Reversed' as shown below:

### Bit Reverse Register Setup:

LeT Modifier Register Mn = \$000000

Let offset register Nn =  $2^{k-1}$

Let the number of points (Data elements/Table size) = 8

Let beginning of table (lower bound) =  $2^k \geq N$ , where N = Table size

Let the end of table (upper bound) =  $2^k - 1$

Let starting point within table = Lower bound  $\leq$  Address register Rn  $\leq$  Upper bound

Original Address Input Sequence	Original Data Input Sequence	Bit-Reversed Address Modification (3 LSBs)	Resulting Input Sequence in Bit-Reverse Order
\$8 -> 1000	#0.0	1000 = \$8 +1	\$8 -> #0.0
\$9 -> 1001	#0.1	1100 = \$C +1	\$9 -> #0.4
\$A -> 1010	#0.2	1010 = \$A +1	\$A -> #0.2
\$B -> 1011	#0.3	1110 = \$E +1	\$B -> #0.6
\$C -> 1100	#0.4	1001 = \$9 +1	\$C -> #0.1
\$D -> 1101	#0.5	1101 = \$D +1	\$D -> #0.5
\$E -> 1110	#0.6	1011 = \$B +1	\$E -> #0.3
\$F -> 1111	#0.7	1111 = \$F	\$F -> #0.7

**Figure 16: Bit-Reverse Addressing**

This address modification is useful for addressing the twiddle factors in  $2^k$ -point FFT addressing and to unscramble  $2^k$ -point FFT data. The range of values for Nn is 0 to + 8M i.e.  $Nn=2^{23}$ , which allows bit-reverse addressing for FFTs up to 16,777,216 points.

**Special Addressing Modes.** They do not use an address register in specifying an effective address. These modes specify the operand or the address of the operand in a field of the instruction or they implicitly reference the operand.

**Immediate Data.** The immediate data addressing mode points to a 24-bit operand located in the extension word of the instruction.

**Immediate Data into a 24-Bit Accumulator - Example.** Transfer the immediate value \$123456 to Accumulator Register A0.

**Figure 17: move #123456,a0**

BEFORE EXECUTION			AFTER EXECUTION		
A2	A1	A0	A2	A1	A0
XX	XXXXXXXX	XXXXXXXX	XX	XXXXXXXX	1 2 3 4 5 6

**Positive Immediate Data into a 56-Bit Accumulator - Example.** Transfer the immediate value \$654321 to accumulator register A. Note that the accumulator register A1 is loaded and that accumulator register A2 is sign-extended from A1 i.e. A2 holds the value \$00 indicating that the value stored in A1 is positive.

**Figure 18: move #654321,a**

BEFORE EXECUTION			AFTER EXECUTION		
A2	A1	A0	A2	A1	A0
XX	XXXXXXXX	XXXXXXXX	00	654321	000000

**Negative Immediate Data into 56-bit Accumulator - Example:** Transfer the immediate value \$876543 to accumulator register B. Note that the accumulator register B1 is loaded and that accumulator register B2 is sign-extended from B1 i.e. B2 holds the value \$FF indicating that the value stored in B1 is negative.

**Figure 19: move #876543,b**

BEFORE EXECUTION			AFTER EXECUTION		
B2	B1	B0	B2	B1	B0
XX	XXXXXXXX	XXXXXXXX	FF	876543	000000

**Immediate Short Data.** The immediate short addressing mode points to an 8-bit or a 12-bit immediate data operand located in the instruction operation word. The immediate data is interpreted as an unsigned integer. if the destination register is one of the following 24-bit registers A2, A1, A0, B2, B1, B0, R0-R7 or N0-N7. The immediate data is transferred into the least significant bits of the destination with the most significant bits zeroed. The immediate data is interpreted as a signed fraction if the destination is one of the following 24-bit registers X1, X0, Y1, Y0, or the 56-bit A and B accumulators. The immediate data is transferred into the most significant bits of the destination with the least significant bits zeroed.

**Immediate Short Data into 24-Bit Register - Example:** Transfer the immediate short data value \$FE to accumulator register A1. Note that the immediate data \$FE is interpreted as an unsigned Integer and is transferred into the least significant bits of A1.

**Figure 20: move #FE,a1**

BEFORE EXECUTION			AFTER EXECUTION		
A2	A1	A0	A2	A1	A0
XX	XXXXXXXX	XXXXXXXX	XX	0000FE	XXXXXXXX

**Example:** Transfer the immediate short data value \$FE to data input register Y1. Note that the immediate data \$FE is interpreted as a signed fraction and is transferred into the most significant bits of Y1.

**Figure 21: move #FE,y1**

BEFORE EXECUTION		AFTER EXECUTION	
Y1	Y0	Y1	Y0
XXXXXXXX	XXXXXXXX	FE0000	XXXXXXXX

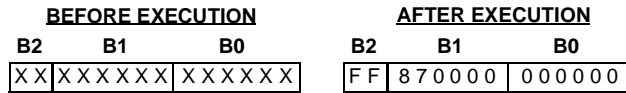
**Immediate Short Data into 56-bit Accumulators - Example:** Transfer the immediate short data value \$34 to the accumulator register A. Note that the immediate data \$34 is interpreted as a signed fraction and is transferred into the most significant bits of the accumulator register A and that accumulator register A2 is sign-extended.

**Figure 22: move #34,a**

BEFORE EXECUTION			AFTER EXECUTION		
A2	A1	A0	A2	A1	A0
XX	XXXXXXXX	XXXXXXXX	00	340000	000000

**Example.** Transfer the immediate short data value \$87 to the accumulator register B. Note that the immediate data \$87 is interpreted as a signed fraction and is transferred into the most significant bits of the accumulator register B and that accumulator register B2 is sign-extended.

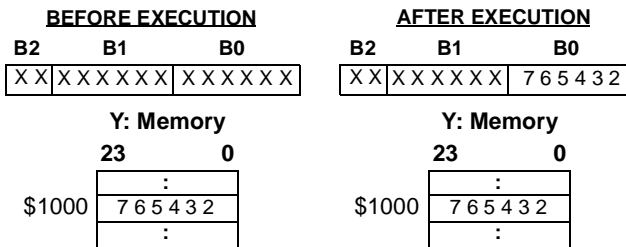
**Figure 23: move #\$87,b**



**Absolute Addressing.** The absolute addressing mode uses the 24-bit address operand located in the instruction extension word as a pointer to the location of the data operand.

**Example:** Transfer the contents of the Y-Memory location pointed to by the instruction extension word to accumulator register B0.

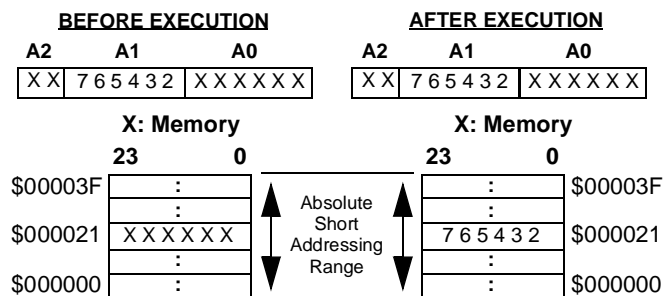
**Figure 24: move y:\$1000,b0**



**Absolute Short Addressing.** This mode uses an immediate 6-bit address operand which is located in the instruction operation word and is zero-extended to form a 24-bit pointer to the data operand. This mode addresses the lowest 64 words (range 0-63 d, \$0-\$3F) of X,Y,L data RAM and interrupt vectors.

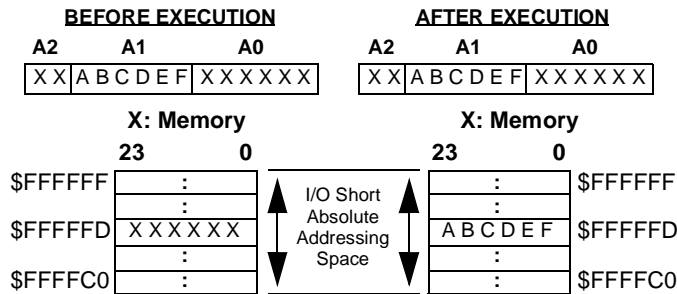
**Example:** Transfer the contents of the accumulator register A1 to the X-Memory location pointed to by the instruction extension word.

**Figure 25: move a1,x:\$21**



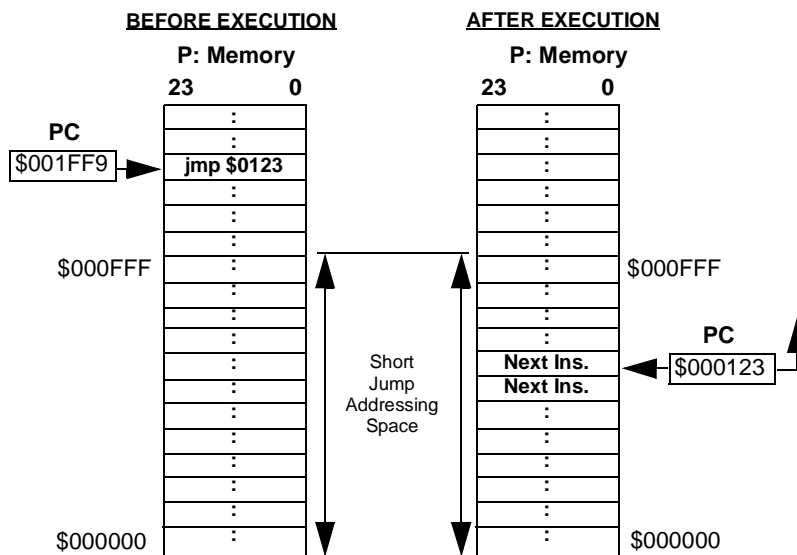
**I/O Short Addressing.** The I/O short addressing is similar to the absolute short addressing mode, it also uses an immediate 6-bit address operand which is located in the instruction operation word but is ones-extended to form a 24-bit pointer to the data operand rather than zero-extended. This mode addresses the highest 64 words (range 16777152-16777215 d, \$FFFC0-\$FFFFFF) of X or Y memory, and is used with the bit manipulation and move peripheral data instructions.

Figure 26: `movep a1,x:$FFFFFFD`



**Short Jump Addressing.** The short jump addressing mode uses a 12-bit immediate jump operand which is located in the instruction operation word and is zero-extended to form a 24-bit “jump to” operand, which is used to replace the contents of the program counter (PC). This mode addresses the lowest 4096 words (range 0-4095 d, \$0-\$000FFF) of program-RAM.

Figure 27: `jmp <$123`



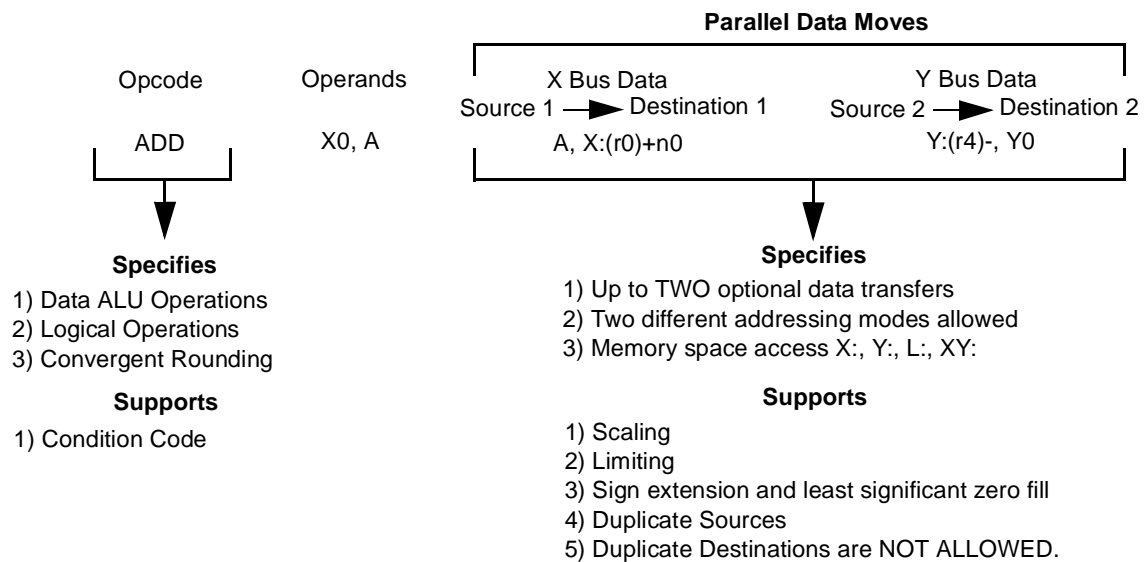
**Program Counter Relative Modes.** In the program counter relative addressing modes, the address of the operand is obtained by adding a displacement, represented in two’s complement format, to the value of the program counter (PC). The PC points to the address of the instruction’s opcode word. The Nn and Mn registers are ignored, and the arithmetic used is always linear.

**Short Displacement PC Relative.** The short displacement occupies 9-bits in the instruction operation word. The displacement is first sign extended to 24 bits and then added to the PC to obtain the address of the operand.

**Long Displacement PC Relative.** This addressing mode requires one word of instruction extension. The address of the operand is the sum of the contents of the PC and the extension word.

**Address Register PC Relative.** The address of the operand is the sum of the contents of the PC and the address register Rn. The Mn and Nn registers are ignored. The contents of the Rn register are unchanged.

**Parallel Data Move Descriptions.** Thirty of the sixty-two DSP56300 core instructions allow an optional parallel data bus movement over the X and/or Y data bus. This allows a data ALU operation to be executed in parallel with up to two data bus moves during one instruction/clock cycle. Ten types of parallel moves are permitted, including register to register moves, register to memory moves, and memory to register moves. For example, the parallel XY memory data move must specify two independent effective addresses (e.g. (opcode/operand (<eax> and <eay>) --> add Y, A A,x:(r1)+n1 y1,y:(r5)+)) where one of the effective addresses must use the lower bank of address registers (R0–R3) while the other effective address must use the upper bank of address registers (R4–R7). However, not all addressing modes are allowed for each type of memory reference. The following paragraphs provide some examples of parallel move operations.



**Figure 28: Parallel Data Move Instruction Syntax**

Parallel Data Move Examples			
	Opcode/Operand	Source 1 → Destination 1	Source 2 → Destination 2
Immediate Short Data Move	ADD A, B	#\$81, A1	
Register to Register Move	ADD X0, A	A1, Y0	
Address Register Update	ADD Y1, A	(r0)+n0	
X or Y Memory Move	ADD X1, B	B, X:\$1000	
X or Y Mem. and Reg. Move	ADD Y0, A	A, X1	Y:(r2)-, Y0
L Memory Move	ADD X, A	A10, L:\$3000	
XY Memory Move	ADD Y, B	B, X:(r3)+n3	Y:(r7)+n7, Y1
<b>Multiply and Accumulate Instructions</b>			
MAC ± S1, S2, D [Parallel Move]	D ± (S1 * S2) -----> D	MAC X0, Y0, A	X:(r0)+, X0 Y:(r4)-, Y0
MACR ± ±S1, S2, D [Parallel Move]	D ± (S1 * S2) + r ----> D	MACR -Y0, Y0, B	X1, X:(r1)+ B, Y:(r5)-

**NOTE: each of these parallel move examples is executed in one single clock/instruction cycle which means each example is executed in 15.2nsec @ 66MHz, or 12.5nec @ 80MHz**

**Figure 29: Examples of Parallel Data Moves**