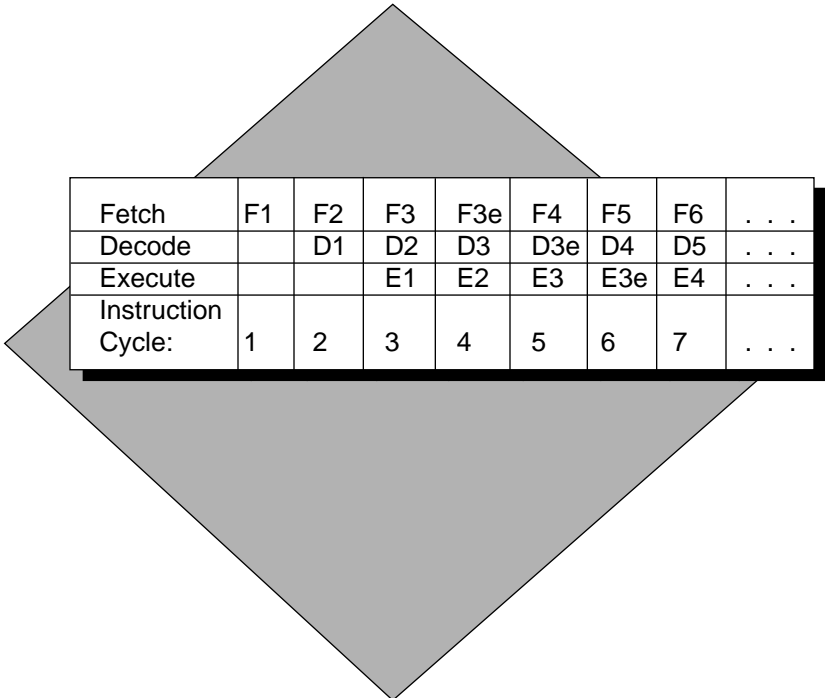

SECTION 6

INSTRUCTION SET INTRODUCTION



Fetch	F1	F2	F3	F3e	F4	F5	F6	...
Decode		D1	D2	D3	D3e	D4	D5	...
Execute			E1	E2	E3	E3e	E4	...
Instruction Cycle:	1	2	3	4	5	6	7	...

SECTION CONTENTS

SECTION 6.1 INSTRUCTION SET INTRODUCTION	3
SECTION 6.2 SYNTAX	3
SECTION 6.3 INSTRUCTION FORMATS	3
6.3.1 Operand Sizes	5
6.3.2 Data Organization in Registers	6
6.3.2.1 Data ALU Registers	6
6.3.2.2 AGU Registers	7
6.3.2.3 Program Control Registers	8
6.3.3 Data Organization in Memory	9
6.3.4 Operand References	11
6.3.4.1 Program References	11
6.3.4.2 Stack References	11
6.3.4.3 Register References	11
6.3.4.4 Memory References	11
6.3.4.4.1 X Memory References	11
6.3.4.4.2 Y Memory References	12
6.3.4.4.3 L Memory References	12
6.3.4.4.4 YX Memory References	12
6.3.5 Addressing Modes	12
6.3.5.1 Register Direct Modes	13
6.3.5.1.1 Data or Control Register Direct	13
6.3.5.1.2 Address Register Direct	13
6.3.5.2 Address Register Indirect Modes	13
6.3.5.3 Special Addressing Modes	14
6.3.5.3.1 Immediate Data	14
6.3.5.3.2 Absolute Address	14
6.3.5.3.3 Immediate Short	14
6.3.5.3.4 Short Jump Address	14
6.3.5.3.5 Absolute Short	14
6.3.5.3.6 I/O Short	16
6.3.5.3.7 Implicit Reference	16
6.3.5.4 Addressing Modes Summary	20
SECTION 6.4 INSTRUCTION GROUPS	20
6.4.1 Arithmetic Instructions	22
6.4.2 Logical Instructions	23
6.4.3 Bit Manipulation Instructions	24
6.4.4 Loop Instructions	24
6.4.5 Move Instructions	26
6.4.6 Program Control Instructions	27

6.1 INSTRUCTION SET INTRODUCTION

The programming model shown in Figure 6-1 suggests that the DSP56K central processing module architecture can be viewed as three functional units which operate in parallel: data arithmetic logic unit (data ALU), address generation unit (AGU), and program control unit (PCU). The instruction set keeps each of these units busy throughout each instruction cycle, achieving maximal speed and maintaining minimal program size.

This section introduces the DSP56K instruction set and instruction format. The complete range of instruction capabilities combined with the flexible addressing modes used in this processor provide a very powerful assembly language for implementing digital signal processing (DSP) algorithms. The instruction set has been designed to allow efficient coding for DSP high-level language compilers such as the C compiler. Execution time is minimized by the hardware looping capabilities, use of an instruction pipeline, and parallel moves.

6.2 SYNTAX

The instruction syntax is organized into four columns: opcode, operands, and two parallel-move fields. The assembly-language source code for a typical one-word instruction is shown in the following illustration. Because of the multiple bus structure and the parallelism of the DSP, up to three data transfers can be specified in the instruction word – one on the X data bus (XDB), one on the Y data bus (YDB), and one within the data ALU. These transfers are explicitly specified. A fourth data transfer is implied and occurs in the program control unit (instruction word prefetch, program looping control, etc.). Each data transfer involves a source and a destination.

Opcode	Operands	XDB	YDB
MAC	X0,Y0,A	X:(R0)+,X0	Y:(R4)+,Y0

The opcode column indicates the data ALU, AGU, or program control unit operation to be performed and must always be included in the source code. The operands column specifies the operands to be used by the opcode. The XDB and YDB columns specify optional data transfers over the XDB and/or YDB and the associated addressing modes. The address space qualifiers (X:, Y:, and L:) indicate which address space is being referenced. Parallel moves are allowed in 30 of the 62 instructions. Additional information is presented in APPENDIX A - INSTRUCTION SET DETAILS.

6.3 INSTRUCTION FORMATS

The DSP56K instructions consist of one or two 24-bit words – an operation word and an optional effective address extension word. The general format of the operation word is

INSTRUCTION FORMATS

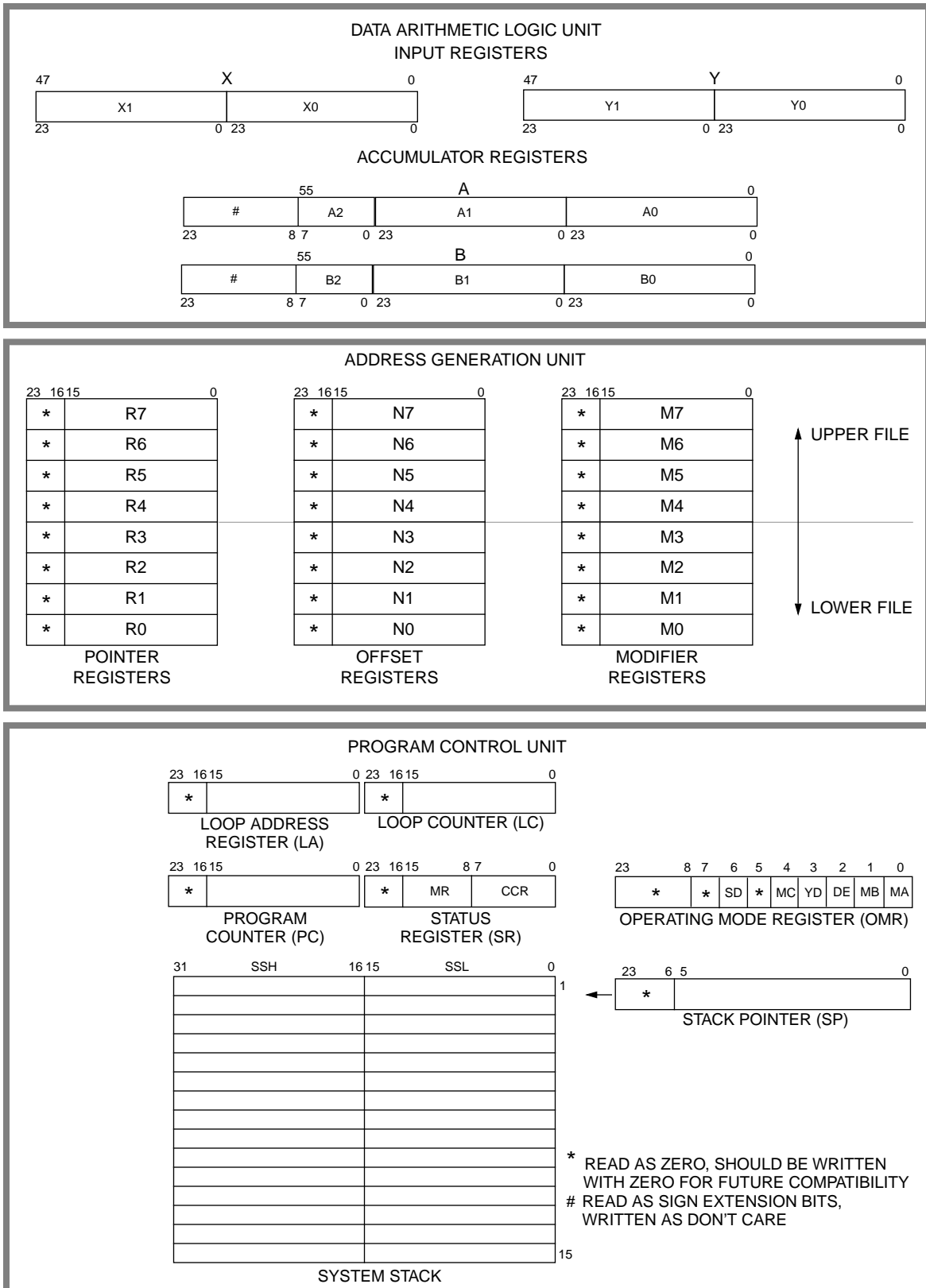


Figure 6-1 DSP56K Central Processing Module Programming Model

shown in Figure 6-2. Most instructions specify data movement on the XDB, YDB, and data ALU operations in the same operation word. The DSP56K performs each of these operations in parallel.

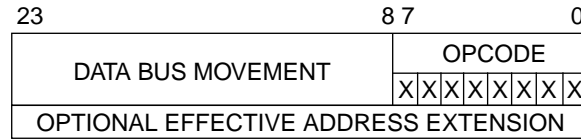


Figure 6-2 General Format of an Instruction Operation Word

The data bus movement field provides the operand reference type. It selects the type of memory or register reference to be made, the direction of transfer, and the effective address(es) for data movement on the XDB and YDB. This field may require additional information to fully specify the operand for certain addressing modes. An effective address extension word following the operation word provides an immediate data address or an absolute address if required (see Section 6.3.5.3 for the description of special addressing modes). Examples of operations that may include the extension word include the move operations X:, X:R, Y:, R:Y, and L:. Additional information is presented in APPENDIX A - INSTRUCTION SET DETAILS.

The opcode field of the operation word specifies the data ALU operation or the program control unit operation to be performed, and any additional operands required by the instruction. Only those data ALU and program control unit operations that can accompany data bus movement will be specified in the opcode field of the instruction. Other data ALU, program control unit, and all address ALU operations will be specified in an instruction word with a different format. These formats include operation words which contain short immediate data or short absolute addresses (see Section 6.3.5.3 for the description of special addressing modes).

6.3.1 Operand Sizes

Operand sizes are defined as follows: a byte is 8 bits long, a short word is 16 bits long, a word is 24 bits long, a long word is 48 bits long, and an accumulator is 56 bits long (see Figure 6-3). The operand size for each instruction is either explicitly encoded in the instruction or implicitly defined by the instruction operation. Implicit instructions support some subset of the five sizes shown in Figure 6-3.

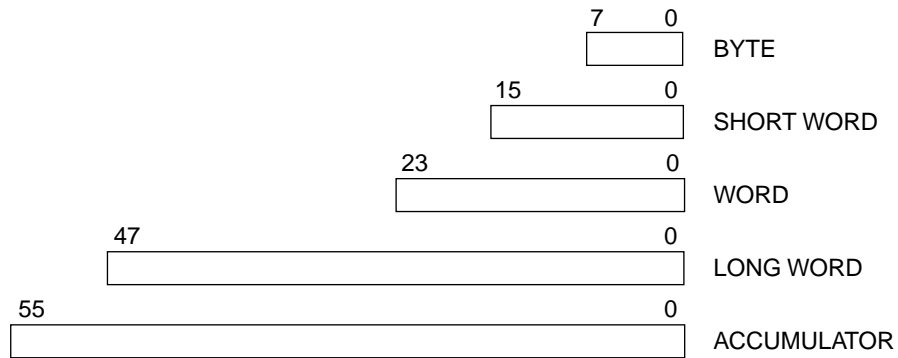


Figure 6-3 Operand Sizes

6.3.2 Data Organization in Registers

The ten data ALU registers support 8- or 24-bit data operands. Instructions also support 48- or 56-bit data operands by concatenating groups of specific data ALU registers. The eight address registers in the AGU support 16-bit address or data operands. The eight AGU offset registers support 16-bit offsets or may support 16-bit address or data operands. The eight AGU modifier registers support 16-bit modifiers or may support 16-bit address or data operands. The program counter (PC) supports 16-bit address operands. The status register (SR) and operating mode register (OMR) support 8- or 16-bit data operands. Both the loop counter (LC) and loop address (LA) registers support 16-bit address operands.

6.3.2.1 Data ALU Registers

The eight main data ALU registers are 24 bits wide. Word operands occupy one register; long-word operands occupy two concatenated registers. The least significant bit (LSB) is the right-most bit (bit 0) and the most significant bit (MSB) is the left-most bit (bit 23 for word operands and bit 47 for long-word operands). The two accumulator extension registers are eight bits wide.

When an accumulator extension register acts as a source operand, it occupies the low-order portion (bits 0–7) of the word and the high-order portion (bits 8–23) is sign extended (see Figure 6-4). When used as a destination operand, this register receives the low-order portion of the word, and the high-order portion is not used. Accumulator operands occupy an entire group of three registers (i.e., A2:A1:A0 or B2:B1:B0). The LSB is the right-most bit (bit 0), and the MSB is the left-most bit (bit 55).

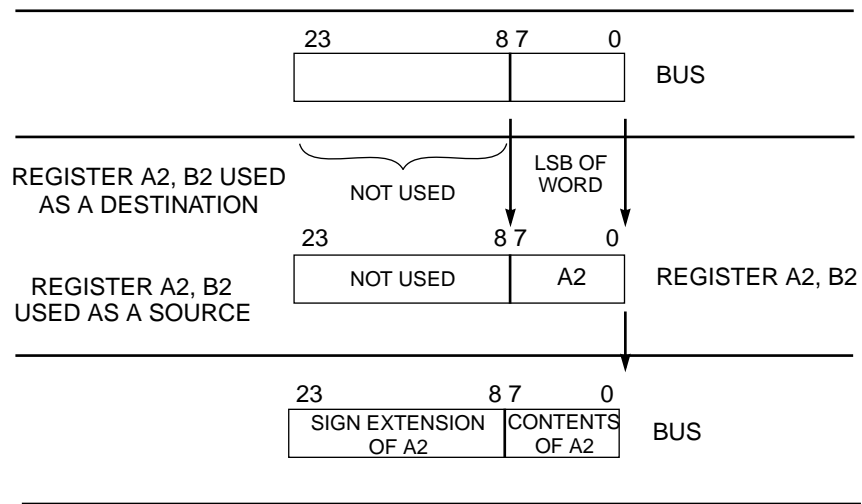


Figure 6-4 Reading and Writing the ALU Extension Registers

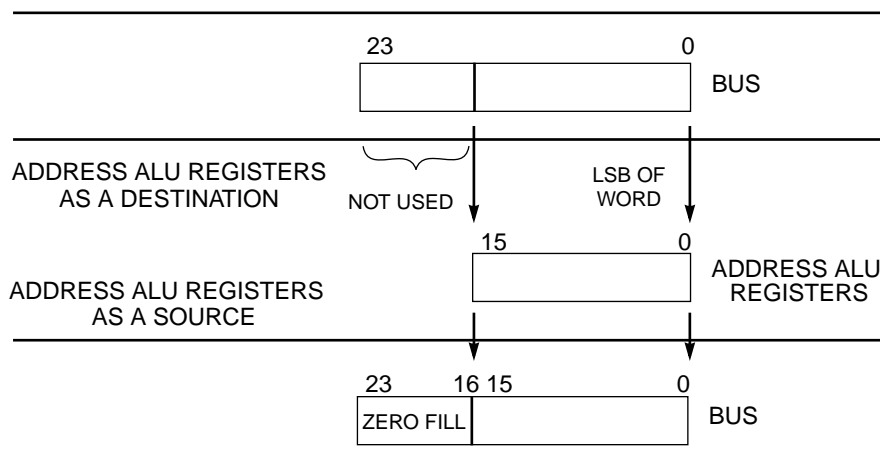
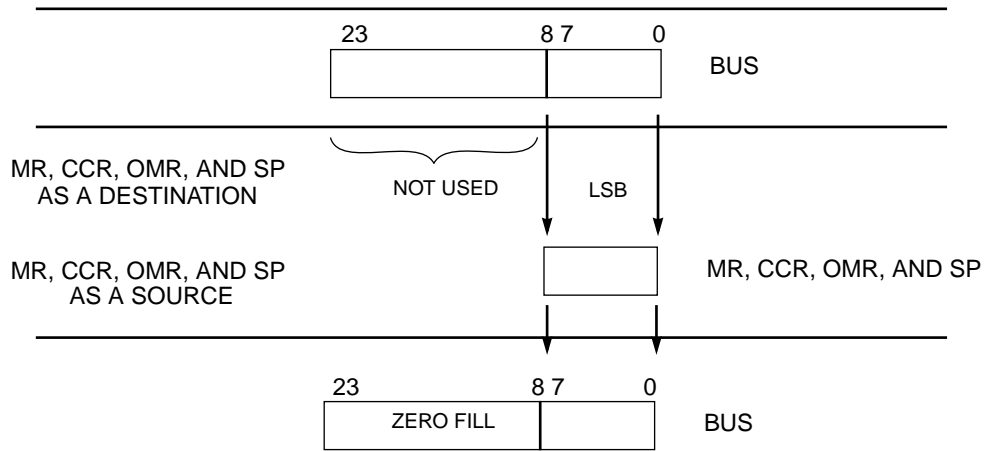


Figure 6-5 Reading and Writing the Address ALU Registers

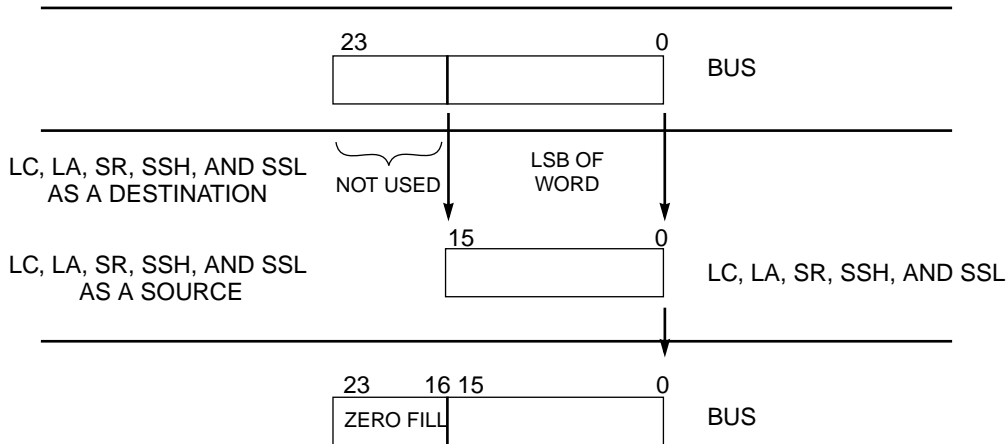
6.3.2.2 AGU Registers

The 24 AGU registers are 16 bits wide. They may be accessed as word operands for address, address modifier, and data storage. When used as a source operand, these registers occupy the low-order portion of the 24-bit word; the high-order portion is read as zeros (see Figure 6-5). When used as a destination operand, these registers receive the low-order portion of the word; the high-order portion is not used. The notation “Rn” designates one of the eight address registers, R0–R7; the notation “Nn” designates one of the eight address offset registers, N0–N7; and the notation “Mn” designates one of the eight

address modifier registers, M0–M7.



(a) 16 Bit



(b) 8 Bit

Figure 6-6 Reading and Writing Control Registers

6.3.2.3 Program Control Registers

The 8-bit operating mode register (OMR) may be accessed as a word operand. However, not all eight bits are defined, and those that are defined will vary depending on the DSP56K family member. In general, undefined bits are written as “don’t care” and read as zero.

The 16-bit SR has the system mode register (MR) occupying the high-order eight bits and

the user condition code register (CCR) occupying the low-order eight bits. The SR may be accessed as a word operand.

The MR and CCR may be accessed individually as word operands (see Figure 6-6(b)). The LC, LA, system stack high (SSH), and system stack low (SSL) registers are 16 bits wide and may be accessed as word operands (see Figure 6-6(a)). When used as a source operand, these registers occupy the low-order portion of the 24-bit word; the high-order portion is zero. When used as a destination operand, they receive the low-order portion of the 24-bit word; the high-order portion is not used. The system stack pointer (SP) is a 6-bit register that may be accessed as a word operand.

The PC, a special 16-bit-wide program control register, is always referenced implicitly as a short-word operand.

6.3.3 Data Organization in Memory

The 24-bit program memory can store both 24-bit instruction words and instruction extension words. The 32-bit system stack (SS) can store the concatenated PC and SR registers (PC:SR) for subroutine calls, interrupts, and program looping. The SS also supports the concatenated LA and LC registers (LA:LC) for program looping. The 24-bit-wide X and Y memories can store word, short-word, and byte operands. Short-word and byte operands, which usually occupy the low-order portion of the X or Y memory word, are either zero extended or sign extended on the XDB or YDB.

The symbols used to abbreviate the various operands and operations in each instruction and their respective meanings are shown in the following list:

Data ALU

X _n	Input Registers X1, X0 (24 Bits)
Y _n	Input Registers Y1, Y0 (24 Bits)
A _n	Accumulator Registers A2 (8 Bits), A1, A0 (24 Bits)
B _n	Accumulator Registers B2 (8 Bits), B1, B0 (24 Bits)
X	Input Register X (X1:X0, 48 Bits)
Y	Input Register Y (Y1:Y0, 48 Bits)
A	Accumulator A (A2:A1:A0, 56 Bits)*
B	Accumulator B (B2:B1:B0, 56 Bits)*
AB	Accumulators A and B (A1:B1, 48 Bits)*

*Data Move Operations: when specified as a source operand, shifting and limiting are performed. When specified as a destination operand, sign extension and zero filling are performed.

BA	Accumulators B and A (B1:A1, 48 Bits)*
A10	Accumulator A (A1:A0, 48 Bits)
B10	Accumulator B (B1:B0, 48 Bits)

Address ALU

Rn	Address Registers R0–R7 (16 Bits)
Nn	Address Offset Registers N0–N7 (16 Bits)
Mn	Address Modifier Registers M0–M7 (16 Bits)

Program Control Unit

PC	Program Counter (16 Bits)
MR	Mode Register (8 Bits)
CCR	Condition Code Register (8 Bits)
SR	Status Register (MR:CCR, 16 Bits)
OMR	Operating Mode Register (8 Bits)
LA	Hardware Loop Address Register (16 Bits)
LC	Hardware Loop Counter (16 Bits)
SP	System Stack Pointer (6 Bits)
SS	System Stack RAM (15X32 Bits)
SSH	Upper 16 Bits of the Contents of the Current Top of Stack
SSL	Lower 16 Bits of the Contents of the Current Top of Stack

Addresses

ea	Effective Address
xxxx	Absolute Address (16 Bits)
xxx	Short Jump Address (12 Bits)
aa	Absolute Short Address (6 Bits Zero Extended)
pp	I/O Short Address (6 Bits Ones Extended)
< . . . >	Contents of the Specified Address
X:	X Memory Reference
Y:	Y Memory Reference
L:	Long Memory Reference – X Concatenated with Y
P:	Program Memory Reference

Miscellaneous

#xx	Immediate Short Data (8 Bits)
#xxx	Immediate Short Data (12 Bits)
#xxxxxx	Immediate Data (24 Bits)
#n	Immediate Short Data (5 Bits)
S,Sn	Source Operand Register
D,Dn	Destination Operand Register

D[n]	Bit n of D Affected
r	Rounding Constant
I1,I0	Interrupt Priority Level in SR
LF	Loop Flag in SR

6.3.4 Operand References

The DSP separates operand references into four classes: program, stack, register, and memory references. The type of operand reference(s) required for an instruction is specified by both the opcode field and the data bus movement field of the instruction. However, not all operand reference types can be used with all instructions. The operand size for each instruction is either explicitly encoded in the instruction or implicitly defined by the instruction operation. Implicit instructions support some subset of the five operand sizes.

6.3.4.1 Program References

Program (P) references, which are references to 24-bit-wide program memory space, are usually instruction reads. Instructions or data operands may be read from or written to program memory space using the move program memory (MOVEM) and move peripheral data (MOVEP) instructions. Depending on the address and the chip operating mode, program references may be internal or external memory references.

6.3.4.2 Stack References

Stack (S) references, which are references to the System Stack (SS), a separate 32-bit-wide internal memory space, are used implicitly to store the PC and SR for subroutine calls, interrupts, and returns. In addition to the PC and SR, the LA and LC registers are stored on the stack when a program loop is initiated. S references are always implied by the instruction. Data is written to the stack memory to save the processor state and is read from the stack memory to restore the processor state. In contrast to S references, references to SSL and SSH are always explicit.

6.3.4.3 Register References

Register (R) references are references to the data ALU, AGU, and program control unit registers. Data can be read from one register and written into another register.

6.3.4.4 Memory References

Memory references, which are references to the 24-bit-wide X or Y memory spaces, can be internal or external memory references, depending on the effective address of the operand in the data bus movement field of the instruction. Data can be read or written from any address in either memory space.

6.3.4.4.1 X Memory References

The operand, which is in X memory space, is a word reference. Data can be transferred from memory to a register or from a register to memory.

6.3.4.4.2 Y Memory References

The operand, a word reference, is in Y memory space. Data can be transferred from memory to a register or from a register to memory.

6.3.4.4.3 L Memory References

Long (L) memory space references both X and Y memory spaces with one operand address. The data operand is a long-word reference developed by concatenating the X and Y memory spaces (X:Y). The high-order word of the operand is in the X memory; the low-order word of the operand is in the Y memory. Data can be read from memory to concatenated registers X1:X0, A1:A0, etc. or from concatenated registers to memory.

6.3.4.4.4 YX Memory References

XY memory space references both X and Y memory spaces with two operand addresses. Two independent addresses are used to access two word operands – one word operand is in X memory space, and one word operand is in Y memory space. Two effective addresses in the instruction are used to derive two independent operand addresses – one operand address may reference either X or Y memory space and the other operand address must reference the other memory space. One of these two effective addresses specified in the instruction must reference one of the address registers, R0–R3, and the other effective address must reference one of the address registers, R4–R7. Addressing modes are restricted to no-update and post-update by +1, –1, and +N addressing modes. Each effective address provides independent read/write control for its memory space. Data may be read from memory to a register or from a register to memory.

6.3.5 Addressing Modes

The DSP instruction set contains a full set of operand addressing modes. To minimize execution time and loop overhead, all address calculations are performed concurrently in the address ALU.

Addressing modes specify whether the operand(s) is in a register or in memory, and provide the specific address of the operand(s). An effective address in an instruction will specify an addressing mode, and, for some addressing modes, the effective address will further specify an address register. In addition, address register indirect modes require additional address modifier information that is not encoded in the instruction. The address modifier information is specified in the selected address modifier register(s). All indirect memory references require one address modifier, and the XY memory reference requires two address modifiers. The definition of certain instructions implies the use of specific registers and addressing modes.

Some address register indirect modes require an offset and a modifier register for use in address calculations. These registers are implied by the address register specified in an effective address in the instruction word. Each offset register (Nn) and each modifier register (Mn) is assigned to an address register (Rn) having the same register number (n). Thus, the assigned register triplets are R0;N0;M0, R1;N1;M1, R2;N2;M2, R3;N3;M3, R4;N4;M4, R5;N5;M5, R6;N6;M6, and R7;N7;M7. Rn is used as the address register; Nn is used to specify an optional offset; and Mn is used to specify the type of arithmetic used to update the Rn.

The addressing modes are grouped into three categories: register direct, address register indirect, and special. These addressing modes are described in the following paragraphs. Refer to Table 6-1 for a summary of the addressing modes and allowed operand references.

6.3.5.1 Register Direct Modes

These effective addressing modes specify that the operand source or destination is one of the data, control, or address registers in the programming model.

6.3.5.1.1 Data or Control Register Direct

The operand is in one, two, or three data ALU register(s) as specified in a portion of the data bus movement field in the instruction. Classified as a register reference, this addressing mode is also used to specify a control register operand for special instructions such as OR immediate to control registers (ORI) and AND immediate to control registers (ANDI).

6.3.5.1.2 Address Register Direct

Classified as a register reference, the operand is in one of the 24 address registers (Rn, Nn, or Mn) specified by an effective address in the instruction.

Note: Due to instruction pipelining, if an address register (Mn, Nn, or Rn) is changed with a MOVE instruction, the new contents will not be available for use as a pointer until the second following instruction.

6.3.5.2 Address Register Indirect Modes

The address register indirect mode description is presented in SECTION 4 - ADDRESS GENERATION UNIT.

6.3.5.3 Special Addressing Modes

The special addressing modes do not use specific registers to specify an effective address. These modes specify the operand or the operand address in a field of the instruction, or they implicitly reference an operand. Figure examples are given for each of the special addressing modes discussed in the following paragraphs.

6.3.5.3.1 Immediate Data

Classified as a program reference, this addressing mode requires one word of instruction extension containing the immediate data. Figure 6-7 shows three examples. Example A moves immediate data to register A0 without affecting A1 or A2. Examples B and C zero fill register A0 and sign extend register A2.

6.3.5.3.2 Absolute Address

This addressing mode requires one word of instruction extension containing the absolute address. Figure 6-8 shows that `MOVE Y:$5432,B0` copies the contents of address \$5432 into B0 without changing memory location \$5432, register B1, or register B2. This addressing mode is classified as both a memory reference and program reference. The 16-bit absolute address is stored in the 16 LSBs of the extension word; the eight MSBs are zero filled.

6.3.5.3.3 Immediate Short

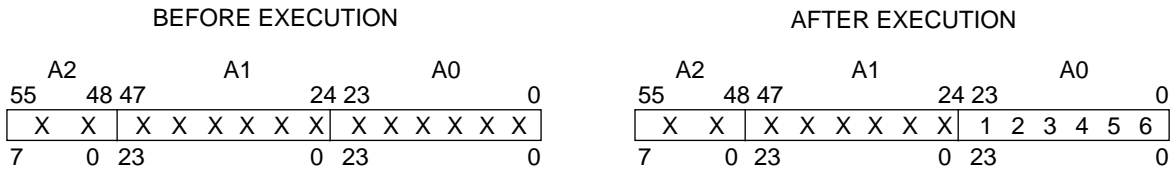
The 8- or 12-bit operand, which is in the instruction operation word, is classified as a program reference. The immediate data is interpreted as an unsigned integer (low-order portion) or signed fraction (high-order portion), depending on the destination register. Figure 6-9 shows the use of immediate short addressing in four examples.

6.3.5.3.4 Short Jump Address

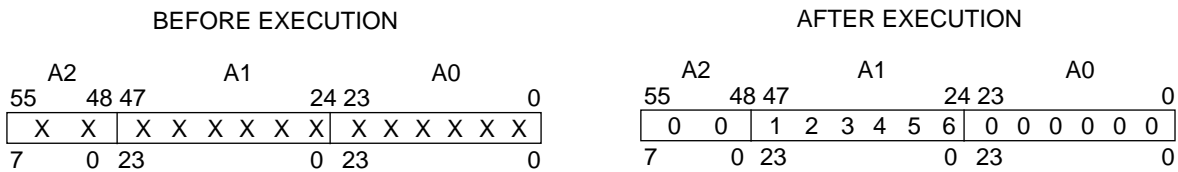
The operand occupies 12 bits in the instruction operation word, which allows addresses \$0000–\$0FFF to be accessed (see Figure 6-10). The address is zero extended to 16 bits

INSTRUCTION FORMATS

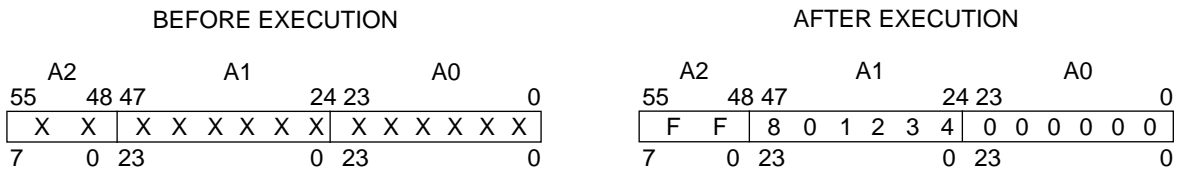
EXAMPLE A: IMMEDIATE INTO 24-BIT REGISTER
(MOVE #123456,A0)



EXAMPLE B: POSITIVE IMMEDIATE INTO 56-BIT REGISTER
(MOVE #123456,A)



EXAMPLE C: NEGATIVE IMMEDIATE INTO 56-BIT REGISTER
(MOVE #801234,A)

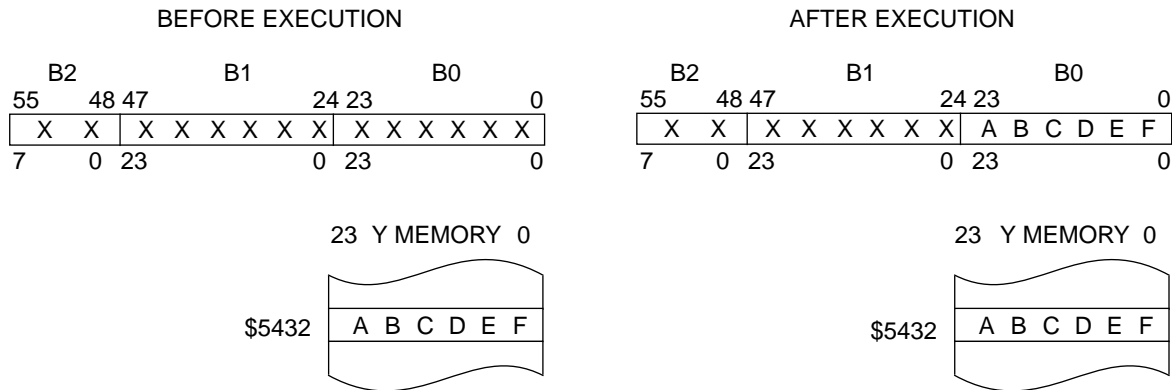


Assembler Syntax: #XXXXXX
Memory Spaces: P:
Additional Instruction Execution Time (Clocks): 2
Additional Effective Address Words: 1

Figure 6-7 Special Addressing – Immediate Data

when used to address program memory. This addressing mode is classified as a program reference.

EXAMPLE: MOVE Y:\$5432,B0



Assembler Syntax: XXXX or aa
 Memory Spaces: P:
 Additional Instruction Execution Time (Clocks): 2
 Additional Effective Address Words: 1

Figure 6-8 Special Addressing – Absolute Addressing

6.3.5.3.5 Absolute Short

The address of the operand occupies six bits in the instruction operation word, allowing addresses \$0000–\$003F to be accessed (see Figure 6-11). Classified as both a memory reference and program reference, the address is zero extended to 16 bits when used to address an operand or program memory.

6.3.5.3.6 I/O Short

Classified as a memory reference, the I/O short addressing mode is similar to absolute short addressing. The address of the operand occupies six bits in the instruction operation word. I/O short is used with the bit manipulation and MOVEP instructions. The I/O short address is ones extended to 16 bits to address the I/O portion of X and Y memory (addresses \$FFC0–\$FFFF – see Figure 6-12).

6.3.5.3.7 Implicit Reference

Some instructions make implicit reference to PC, SS, LA, LC, or SR. For example, the jump instruction (JMP) implicitly references the PC; whereas, the repeat next instruction (REP) implicitly references LC. The registers implied and their uses are defined by the individual instruction descriptions (see APPENDIX A - INSTRUCTION SET DETAILS).

6.3.5.4 Addressing Modes Summary

INSTRUCTION FORMATS

EXAMPLE A: IMMEDIATE SHORT INTO A0, A1, A2, B0, B1, B2, Rn, Nn
(MOVE #FF,A1)

BEFORE EXECUTION

55	A2	48	47	A1	24	23	A0	0
X	X	X	X	X	X	X	X	X
7	0	23	0	23	0	23	0	0

AFTER EXECUTION

55	A2	48	47	A1	24	23	A0	0
X	X	0	0	0	0	F	F	X
7	0	23	0	23	0	23	0	0

EXAMPLE B: POSITIVE IMMEDIATE SHORT INTO X0, X1, Y0, Y1, A, B
(MOVE #1F, Y1)

BEFORE EXECUTION

47	Y1	24	23	Y0	0
X	X	X	X	X	X
23	0	23	0	23	0

AFTER EXECUTION

47	Y1	24	23	Y0	0
1	F	0	0	0	0
23	0	23	0	23	0

EXAMPLE C: POSITIVE IMMEDIATE SHORT INTO X, Y, A, B
(MOVE #1F, A)

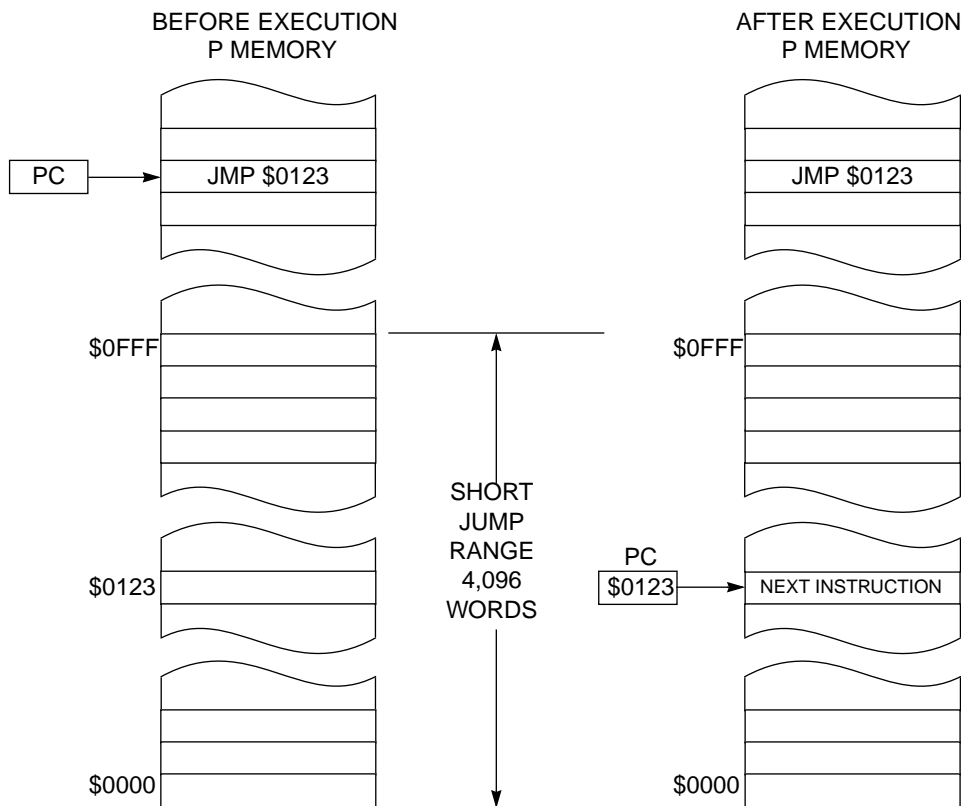
BEFORE EXECUTION

55	A2	48	47	A1	24	23	A0	0
X	X	X	X	X	X	X	X	X
7	0	23	0	23	0	23	0	0

AFTER EXECUTION

55	A2	48	47	A1	24	23	A0	0
0	0	1	F	0	0	0	0	0
7	0	23	0	23	0	23	0	0

EXAMPLE: JMP \$123



Assembler Syntax: XXX
 Memory Spaces: P:
 Additional Instruction Execution Time (Clocks): 0
 Additional Effective Address Words: 0

Figure 6-10 Special Addressing – Short Jump Address

6.4 INSTRUCTION GROUPS

The instruction set is divided into the following groups:

- Arithmetic
- Bit Manipulation
- Move
- Logical
- Loop
- Program Control

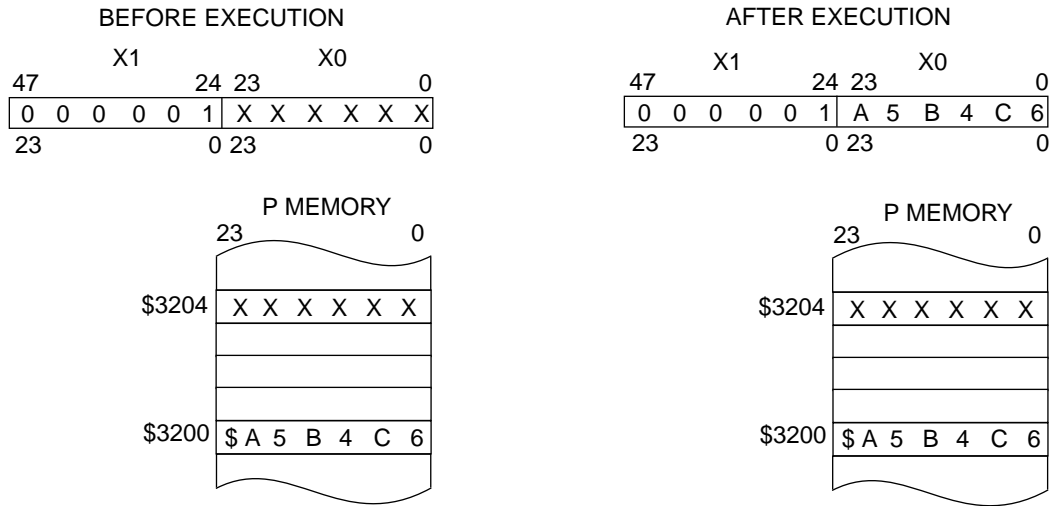
Each instruction group is described in the following paragraphs; detailed information on each instruction is given in APPENDIX A - INSTRUCTION SET DETAILS.

6.4.1 Arithmetic Instructions

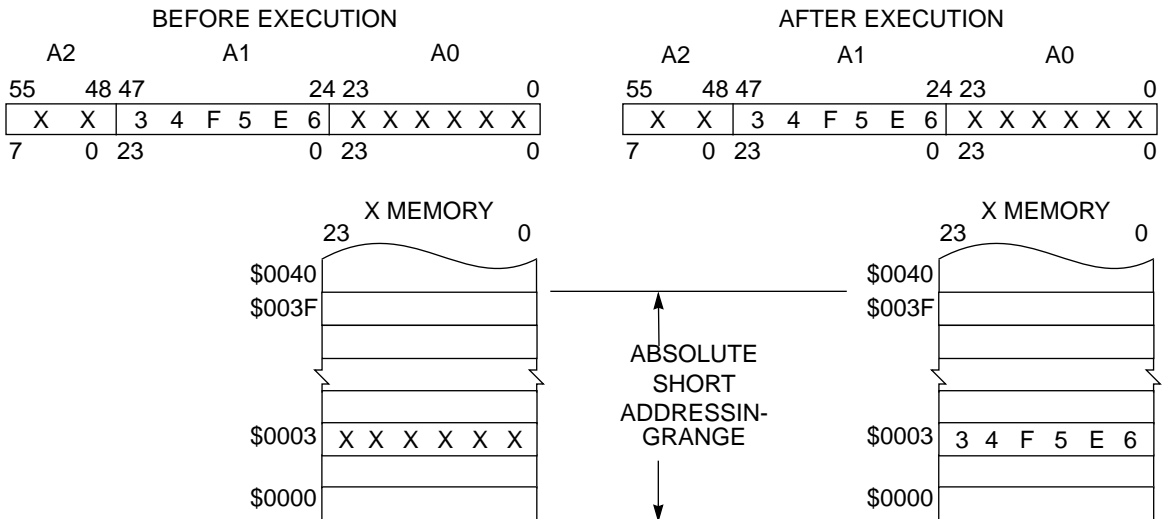
The arithmetic instructions, which perform all of the arithmetic operations within the data

INSTRUCTION GROUPS

EXAMPLE A: MOVE P: \$3200,X0



EXAMPLE B: MOVE A1, X: \$3

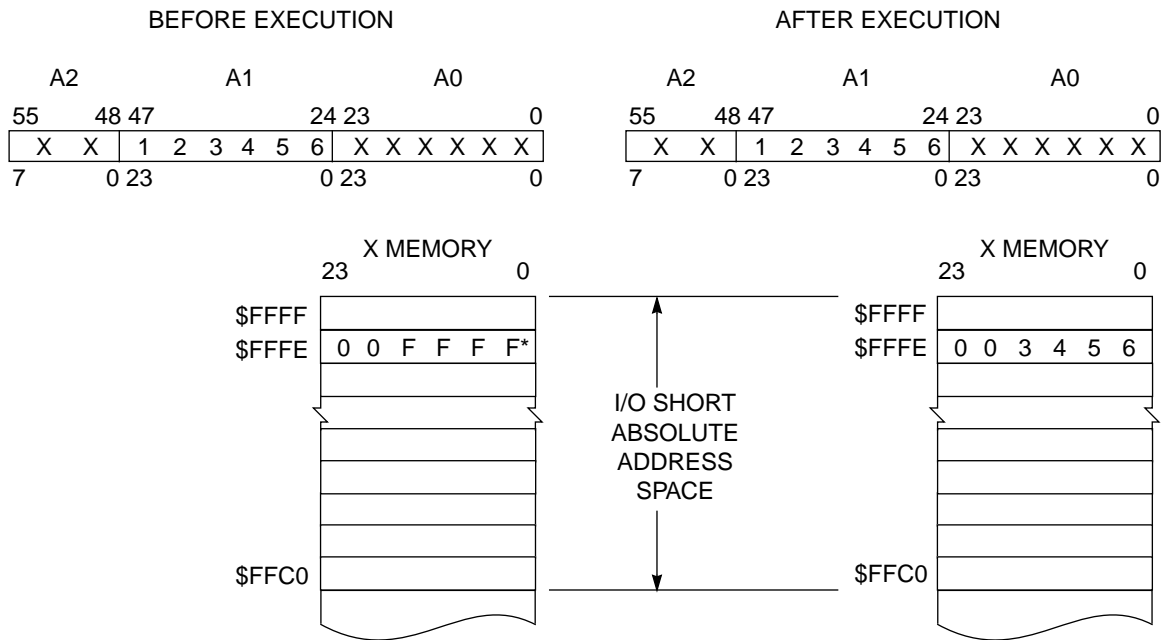


Assembler Syntax: aa
 Memory Spaces: P:, X:, Y:, L:
 Additional Instruction Execution Time (Clocks): 0
 Additional Effective Address Words: 0

Figure 6-11 Special Addressing - Absolute Short Address

INSTRUCTION GROUPS

EXAMPLE: MOVEP A1, X:<<\$FFFE



*Contents of Bus Control Register (X:\$FFFE) After Reset

Assembler Syntax: pp
 Operands Referenced: X, Y Memories
 Additional Instruction Execution Time (Clocks): 0
 Additional Effective Address Words: 0

Figure 6-12 Special Addressing – I/O Short Address

ALU, execute in one instruction cycle. These instructions may affect all of the CCR bits. Arithmetic instructions are register based (register direct addressing modes used for operands) so that the data ALU operation indicated by the instruction does not use the XDB, the YDB, or the global data bus (GDB). Optional data transfers may be specified with most arithmetic instructions, which allows for parallel data movement over the XDB and YDB or over the GDB during a data ALU operation. This parallel movement allows new data to be prefetched for use in subsequent instructions and allows results calculated in previous instructions to be stored. The following list contains the arithmetic instructions:

Table 6-1 Addressing Modes Summary

Addressing Mode	Modifier MMMM	Operand Reference								
		P	S	C	D	A	X	Y	L	XY
Register Direct										
Data or Control Register	No			X	X					
Address Register	No					X				
Address Modifier Register	No					X				
Address Offset Register	No					X				
Address Register Indirect										
No Update	No	X					X	X	X	X
Postincrement by 1	Yes	X					X	X	X	X
Postdecrement by 1	Yes	X					X	X	X	X
Postincrement by Offset Nn	Yes	X					X	X	X	X
Where: MMMM = Address Modifier	Yes	X					X	X	X	
P = Program Reference		X					X	X	X	
S = Stack Reference		X					X	X	X	

C = Program Control Unit Register Reference
 D = Data ALU Register Reference
 A = AGU Register Reference
 X = X Memory Reference
 Y = Y Memory Reference
 L = L Memory Reference
 XY = XY Memory Reference

ABS	Absolute Value
ADC	Add Long with Carry
ADD	Addition
ADDL	Shift Left and Add
ADDR	Shift Right and Add
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right
CLR	Clear an Operand
CMP	Compare
CMPM	Compare Magnitude
DEC*	Decrement by One
DIV*	Divide Iteration
INC*	Increment by One
MAC	Signed Multiply-Accumulate**
MACR	Signed Multiply-Accumulate and Round**
MPY	Signed Multiply**
MPYR	Signed Multiply and Round**
NEG	Negate Accumulator
NORM*	Normalize
RND	Round
SBC	Subtract Long with Carry
SUB	Subtract
SUBL	Shift Left and Subtract
SUBR	Shift Right and Subtract
Tcc*	Transfer Conditionally
TFR	Transfer Data ALU Register
TST	Test an Operand

6.4.2 Logical Instructions

The logical instructions execute in one instruction cycle and perform all of the logical operations within the data ALU (except ANDI and ORI). They may affect all of the CCR bits and, like the arithmetic instructions, are register based.

Logical instructions are the only instructions that allow apparent duplicate destinations, such as:

```
AND X0,A      X:(R0):A0
```

A logical instruction uses only the MSP portion of the A and B registers (A1 and B1).

*These instructions do not allow parallel data moves.

**Certain applications of these instructions do not allow parallel data moves.

Therefore, the instruction actually ignores what appears to be a duplicate destination and logically ANDs the value in the X0 register with the bits in the A1 portion (bits 47-24) of the A accumulator. The parallel move shown above can simultaneously write to either of the other two portions of the A or the B accumulator without conflict. Avoid confusion by explicitly stating A1 or B1 in the original instruction.

Optional data transfers may be specified with most logical instructions, allowing parallel data movement over the XDB and YDB or over the GDB during a data ALU operation. This parallel movement allows new data to be prefetched for use in subsequent instructions and allows results calculated in previous instructions to be stored. The following list includes the logical instructions:

AND	Logical AND
ANDI*	AND Immediate to Control Register
EOR	Logical Exclusive OR
LSL	Logical Shift Left
LSR	Logical Shift Right
NOT	Logical Complement
OR	Logical Inclusive OR
ORI*	OR Immediate to Control Register
ROL	Rotate Left
ROR	Rotate Right

*These instructions do not allow parallel data moves.

6.4.3 Bit Manipulation Instructions

The bit manipulation instructions test the state of any single bit in a memory location or a register and then optionally set, clear, or invert the bit. The carry bit of the CCR will contain the result of the bit test. The following list defines the bit manipulation instructions:

BCLR	Bit Test and Clear
BSET	Bit Test and Set
BCHG	Bit Test and Change
BTST	Bit Test on Memory and Registers

6.4.4 Loop Instructions

The hardware DO loop executes with no overhead cycles after the DO instruction itself has been executed— i.e., it runs as fast as straight-line code. Replacing straight-line code with DO loops can significantly reduce program memory. The loop instructions control hardware looping by 1) initiating a program loop and establishing looping parameters or by 2) restoring the registers by pulling the SS when terminating a loop. Initialization includes saving registers used by a program loop (LA and LC) on the SS so that program loops can be nested. The address of the first instruction in a program loop is also saved to allow no-overhead looping. The loop instructions are as follows:

DO	Start Hardware Loop
ENDDO	Exit from Hardware Loop

Both static and dynamic loop counts are supported in the following forms:

DO	#xxx,Expr	; (Static)
DO	S,Expr	; (Dynamic)

Expr is an assembler expression or absolute address, and S is a directly addressable register such as X0.

The operation of a DO loop is shown in Figure 6-13. When a program loop is initiated with the execution of a DO instruction, the following events occur:

1. The stack is pushed.
 - A. The SP is incremented.
 - B. The current 16-bit LA and 16-bit LC registers are pushed onto the SS to allow nested loops.
 - C. The LC register is initiated with the loop count value specified in the DO instruction.

START OF LOOP

1)SP+1 ↗ SP; LA ↗ SSH; LC ↗ SSL; #xxx ↗ LC
 2)SP+1 ↗ SP; PC ↗ SSH; SR ↗ SSL; Expr-1 ↗ LA
 3)1 ↗ LF

END OF LOOP

1)SSL(LF) ↗ SR
 2)SP-1 ↗ SP; SSH ↗ LA; SSL ↗ LC; SP-1 ↗ SP
 3)PC + 1 ↗ PC

NOTE:
 #xxx=Loop Count Number
 Expr=Expression

Figure 6-13 Hardware DO Loop

2. The stack is pushed again.
 - A. The SP is incremented.
 - B. The address of the first instruction in the program loop (PC) and the current SR contents are pushed onto the SS.
 - C. The LA register is initialized with the value specified in the DO instruction decremented by one.
3. The LF bit in the SR is set. The LF bit is set when a program loop is in progress and enables the end-of-loop detection.

The program loop continues execution until the program address fetched equals the LA register contents (last address of program loop). The contents of the LC are then tested for one. If the LC is not one, it is decremented, and the top location in the stack RAM is read (but not pulled) into the PC to return to the start of the loop. If the LC is one, the program loop is terminated by the following sequence:

1. Reading the previous LF bit from the top location in the SS into the SR
2. Purging the SS (pulling the top location and discarding the contents), pulling the LA and LC registers off the SS, and restoring the respective registers
3. Incrementing the PC

The LF bit (pulled from the SS when a loop is terminated) indicates if the terminated loop was a nested loop. Figure 6-14 shows two DO loops, one nested inside the other. If the stack is managed to prevent a stack overflow, DO loops can be stacked indefinitely.

The ENDDO instruction is not used for normal termination of a DO loop; it is only used to terminate a DO loop before the LC has been decremented to one.

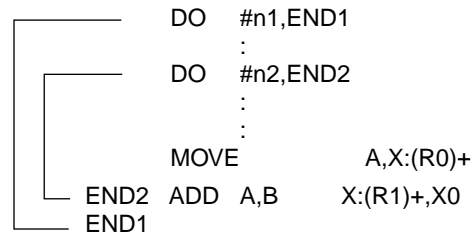


Figure 6-14 Nested DO Loops

6.4.5 Move Instructions

The move instructions perform data movement over the XDB and YDB or over the GDB. Move instructions only affect the CCR bits S and L. The S bit is affected if data growth is detected when the A or B registers are moved onto the bus. The L bit is affected if limiting is performed when reading a data ALU accumulator register. An address ALU instruction (LUA) is also included in the following move instructions. The MOVE instruction is the parallel move with a data ALU no-operation (NOP).

- LUA Load Updated Address
- MOVE Move Data Register
- MOVEC Move Control Register
- MOVEM Move Program Memory
- MOVEP Move Peripheral Data

Note: Due to instruction pipelining, if an AGU register (Mn, Nn, or Rn) is directly changed with a MOVE-type instruction, the new contents may not be available for use until the second following instruction. See the restrictions discussed in SECTION 7 - PROCESSING STATES on page 7-10.

There are nine classifications of parallel data moves between registers and memory. Figure 6-15 shows seven parallel moves. The source of the data to be moved and the destination are separated by a comma.

Examples of the other two classifications, XY and long (L) moves, are shown in Figure 6-16. Example A illustrates the following steps: 1) register X0 is added to register A and the result is placed in register A; 2) register X0 is moved to the X memory register location pointed to by R3, and R3 is incremented; and 3) the contents of the Y memory location pointed to by R7 is moved to the B register, and R7 is decremented.

Example B depicts the following sequence: 1) register X0 is added to register A and the result is placed in register A; and 2) registers A and B are moved, respectively, to the loca-

	OPCODE/OPERANDS	PARALLEL MOVE EXAMPLES
IMMEDIATE SHORT DATA	ADD X0,A	#\$05,Y1
ADDRESS REGISTER UPDATE	ADD X0,A	(R0)+N0
REGISTER TO REGISTER	ADD X0,A	A1,Y0
X MEMORY	ADD X0,A	X0,X:(R3)+
X MEMORY PLUS REGISTER	ADD X0,A	X:(R4)-,X1 A,Y0
Y MEMORY	ADD X0,A	Y:(R6)+N6,X0
Y MEMORY PLUS REGISTER	ADD X0,A	A,X0 B,Y:(R0)

NOTE: Parallel Move Syntax—Source(Src), Destination(Dst)

Figure 6-15 Classifications of Parallel Data Moves

contents of the 56-bit registers A and B were rounded to 24 bits before moving to the 24-bit memory registers.

The DSP offers parallel processing of the data ALU, AGU, and program control unit. For the instruction word above, the DSP will perform the designated operation (data ALU), the data transfers specified with address register updates (AGU), and will decode the next instruction and fetch an instruction from program memory (program control unit) all in one instruction cycle. When an instruction is more than one word in length, an additional instruction execution cycle is required. Most instructions involving the data ALU are register based (all operands are in data ALU registers), thereby allowing the programmer to keep each parallel processing unit busy. An instruction that is memory oriented (such as a bit manipulation instruction) or that causes a control-flow change (such as a JMP) prevents the use of parallel processing resources during its execution.

6.4.6 Program Control Instructions

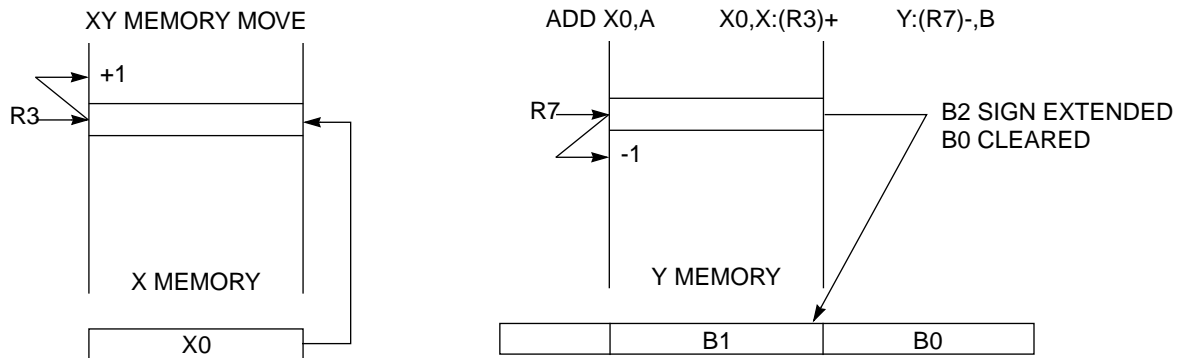
The program control instructions include jumps, conditional jumps, and other instructions affecting the PC and SS. Program control instructions may affect the CCR bits as specified in the instruction. Optional data transfers over the XDB and YDB may be specified in some of the program control instructions. The following list contains the program control instructions:

DEBUG	Enter Debug Mode
DEBUGcc	Enter Debug Mode Conditionally
Ill	Illegal Instruction
Jcc	Jump Conditionally
JMP	Jump

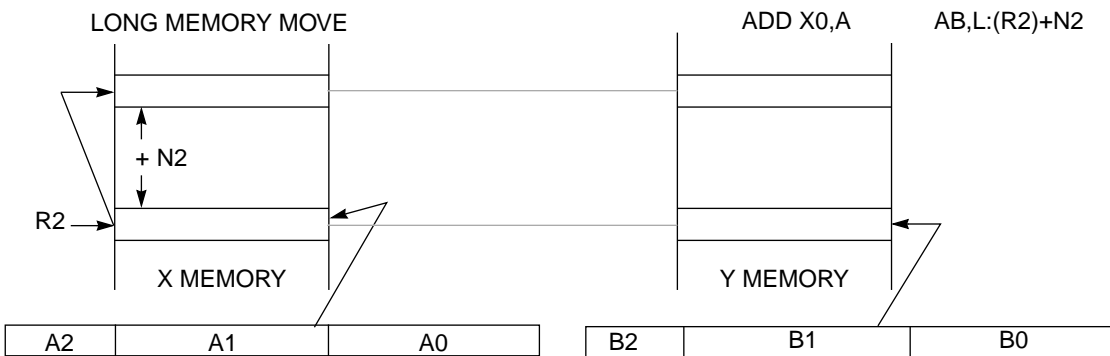
INSTRUCTION GROUPS

JCLR	Jump if Bit Clear
JSET	Jump if Bit Set
JScc	Jump to Subroutine Conditionally
JSR	Jump to Subroutine
JSCLR	Jump to Subroutine if Bit Clear
JSSET	Jump to Subroutine if Bit Set
NOP	No Operation
REP	Repeat Next Instruction
RESET	Reset On-Chip Peripheral Devices
RTI	Return from Interrupt
RTS	Return from Subroutine
STOP	Stop Processing (Low-Power Standby)
SWI	Software Interrupt
WAIT	Wait for Interrupt (Low-Power Standby)

Example A



Example B



A,B ARE SHIFTED AND LIMITED

Figure 6-16 Parallel Move Examples

INSTRUCTION GROUPS
