Chapter 12
Guide to the Instruction Set

This chapter presents the DSP56300 instruction format as well as partial encodings for use in instruction encoding. The alphabetical instruction descriptions are presented in **Appendix B**, I*nstruction Set*. The complete range of instruction capabilities combined with the flexible DSP56300 addressing modes provide a very powerful assembly language for implementing DSP algorithms. The instruction set allows efficient coding for DSP high-level language compilers, such as the C Compiler. Hardware looping capabilities, an instruction pipeline, and parallel moves minimize execution time.

12.1 Instruction Formats and Syntax

The DSP56300 core instructions consist of one or two 24-bit words—an operation word and an optional extension word. This extension word can be either an effective address extension word or an immediate data extension word. While the extension word occupies the full 24-bit width of the program memory, only the sixteen Least Significant Bits (LSBs) are relevant for effective address extension or for immediate data. Therefore, the extension word is effectively sixteen bits wide. **Figure 12-1** shows the general formats of the instruction word. Most instructions specify data movement on the X Data Bus (XDB), Y Data Bus (YDB), and Data ALU operations in the same operation word. The DSP56300 core performs each of these operations in parallel.

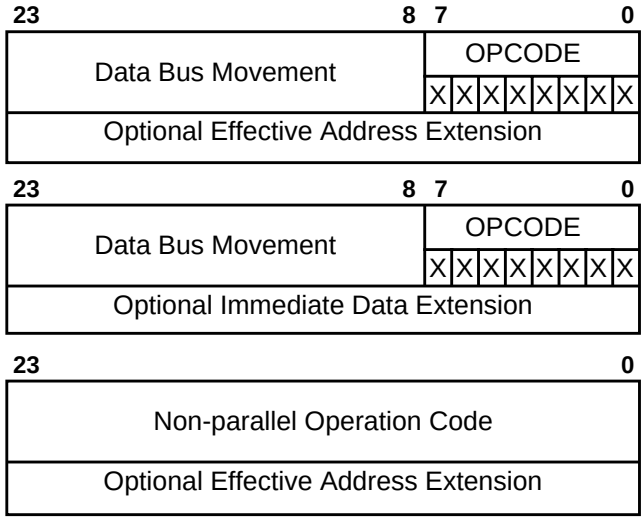


Figure 12-1. General Formats of an Instruction Word

The Data Bus Movement field provides the operand reference type, which 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/or YDB. This field may require additional information to fully specify the operand for certain addressing modes. An extension word following the operation word is used to provide immediate data, absolute address or address displacement, if required. Examples of operations that may include the extension word include move operation such as MOVE X:$100,X0.

The Opcode field of the operation word specifies the Data ALU operation or the Program Control Unit (PCU) operation to be performed.

The instruction syntax has two formats—parallel and non-parallel, as **Table 12-1** and **Table 12-2** show. A parallel instruction is organized into five columns: opcode, operands, two optional parallel-move fields, and an optional condition field. The condition field disables the execution of the opcode if the condition is not true, and it cannot be used in conjunction with the parallel move fields.

Table 12-1. Parallel Instruction Format

	Opcode	Operands	XDB	YDB	Condition
Example 1	MAC	X0,Y0,A	X:(R0)+,X0	Y:(R4)+,Y0	
Example 2	MOVE		X:-(R1),X1		
Example 3	MAC	X1,Y1,B			
Example 4	MPY	X0,Y0,A			IFeq

Assembly-language source codes for some typical one-word instructions are shown in **Table 12-1**. Because of the multiple bus structure and the parallelism of the DSP56300 core, as many as three data transfers can be specified in the instruction word—one on the XDB, one on the YDB, and one within the Data ALU. These transfers are explicitly specified. A fourth data transfer is implied and occurs in the PCU (instruction word prefetch, program looping control, etc.). The opcode column indicates the Data ALU operation to be performed, but may be excluded if only a MOVE operation is needed. 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 YDB and the associated addressing modes. The address space qualifiers (X:, Y:, and L:) indicate which address space is being referenced.

A non-parallel instruction is organized into two columns: opcode and operands. Assembly-language source codes for some typical one-word instructions are shown in **Table 12-2**. Non-parallel instructions include all the program control, looping, and peripherals read/write instructions. They also include some Data ALU instructions that are impossible to encode in the Opcode field of the parallel format.

Table 12-2. Non-Parallel Instruction Format

	Opcode	Operands
Example 1:	JEQ	(R5)
Example 2:	MOVEP	#data,X:ipr
Example 3:	RTS	

12.2 Operand Lengths

Operand lengths are defined as follows: a byte is 8 bits, a word is 16 bits, a long word is 48 bits, and an accumulator is 56 bits, as shown in **Figure 12-2**. The operand size for each instruction is either explicitly encoded in the instruction or implicitly defined by the instruction operation.

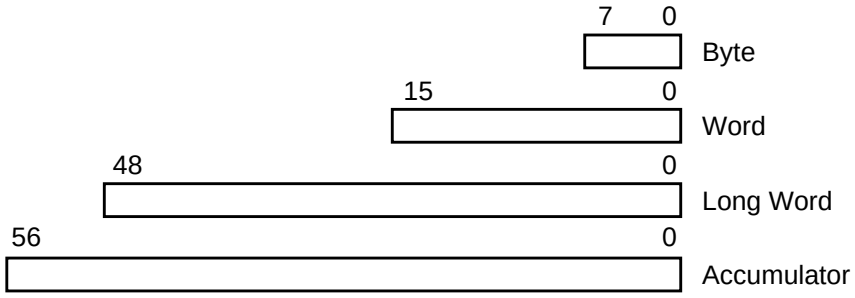


Figure 12-2. Operand Lengths

In Sixteen-Bit Arithmetic mode the operand lengths are as follows: a byte is 8 bits, a word is 16 bits, a long word is 32 bits, and an accumulator is 40 bits.

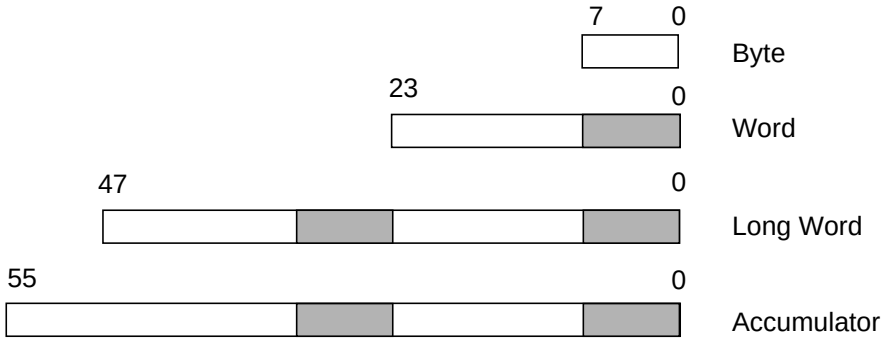


Figure 12-3. Operand Lengths in Sixteen-Bit Mode

Table 12-3 shows the operand lengths supported by the registers of the DSP56300 core.

Table 12-3. Register Operand Lengths

Registers	Number of Registers	Operand Lengths Supported	Sixteen-Bit Mode
ALU	10	8- or 24-bit data With concatenation: 48- or 56-bit data	16-bit data With concatenation: 32- or 40-bit data
AGU address registers	8	24-bit address or data	No
AGU offset registers	8	24-bit offsets or 24-bit address or data	No
AGU modifier registers	8	24-bit modifiers or 24-bit address or data	No
Program Counter (PC)	1	24-bit address	No
Status Register (SR)	1	8- or 24-bit data	16-bit data
Operating Mode Register (OMR)	1	8- or 24-bit data	16-bit data
Loop Counter (LC)	1	24-bit address	No
Loop Address (LA)	1	24-bit address	No

12.2.1 Data ALU Registers

The eight main data 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). In Sixteen-Bit mode, the LSB is bit 8 and bits 24 to 31 are ignored for long-word operands. The MSB is the leftmost bit.

The two accumulator extension registers are 8 bits wide. When an accumulator extension register is a source operand, it occupies the low-order portion (bits 0–7) of the word; the high-order portion (bits 8–23) is sign-extended (see **Figure 12-5**). 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 (e.g., A2:A1:A0 or B2:B1:B0). The LSB is the right-most bit (bit 0 in 24-bit mode and bit 8 for 16-bit mode), and the MSB is the leftmost bit (bit 55).

When a 56-bit accumulator (A or B) is specified as a *source* operand S, the accumulator value is optionally shifted according to the Scaling mode bits S0 and S1 in the Mode Register (MR). If the data out of the shifter indicates that the accumulator extension register is in use and the data is to be moved into a 24-bit destination, the value stored in the destination is limited to a maximum positive or negative saturation constant to minimize truncation error. Limiting does not occur if an individual 24-bit accumulator register (A1, A0, B1, or B0) is specified as a source operand instead of the full 56-bit

accumulator (A or B). This limiting feature allows block floating-point operations to be performed with error detection since the L bit in the Condition Code Register (CCR) is latched.

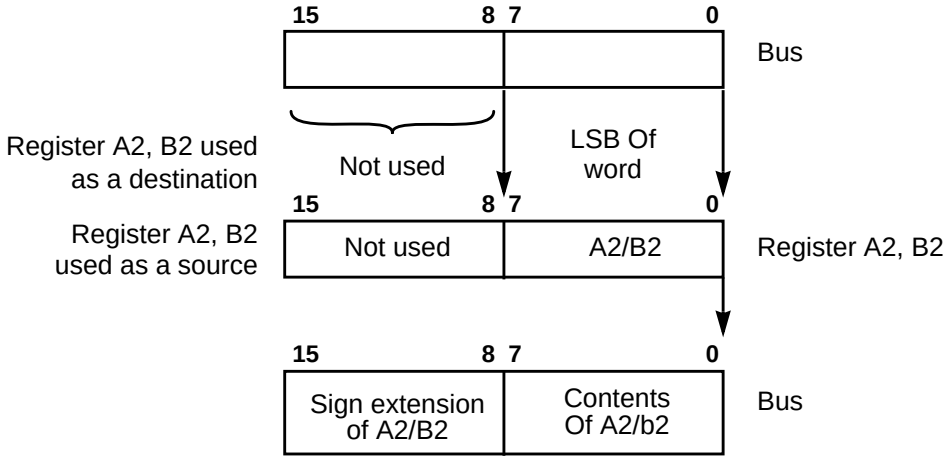


Figure 12-4. Reading and Writing the ALU Extension Registers

When a 56-bit accumulator (A or B) is specified as a *destination* operand D, any 24-bit source data to be moved into that accumulator is automatically extended to 56 bits by sign-extending the MSB of the source operand (Bit 23) and appending the source operand with twenty-four 0s in the LSBs. For 24-bit source operands, both the automatic sign extension and zeroing features can be disabled by specifying the destination register to be one of the individual 24-bit accumulator registers (A1 or B1).

12.2.2 AGU Registers

The twenty-four 24-bit AGU registers can be accessed as word operands for address, address offset, address modifier, and data storage. The Rn notation designates one of the eight address registers, R0–R7. The Nn notation designates one of the eight address offset registers, N0–N7. The Mn notation designates one of the eight address modifier registers, M0–M7.

12.2.3 Program Control Registers

Within the 24-bit Operating Mode Register (OMR), the Chip Operating Mode (COM) register occupies the low-order 8 bits, the Extended chip Operating Mode (EOM) register occupies the middle-order 8 bits, and the System Stack Control Status (SCS) register occupies the high-order 8 bits. The OMR and the Vector Base Address (VBA) are accessed as word operands; however, not all of their bits are defined. Reserved bits are read as zero and should be written with zero for future compatibility.

Within the 24-bit SR, the user condition code register (CCR) occupies the low-order 8 bits, the system Mode Register (MR) occupies the middle-order 8 bits, and the Extended Mode Register (EMR) occupies the high-order 8 bits. The SR can be accessed as a word operand. The MR and CCR can be accessed individually as word operands (see **Figure 12-5**). The Loop Counter (LC), Loop Last Address (LA), stack Size (SZ), System Stack High (SSH), and System Stack Low (SSL) registers are 24 bits wide and are accessed as word operands. The system Stack Pointer (SP) is a 24-bit register that is accessed as a word operand. The PC, a special 24-bit-wide Program Counter register, is generally referenced implicitly as a word operand, but it can also be referenced explicitly (by all PC-relative operation codes) as a word operand.(see **Figure 12-5**).

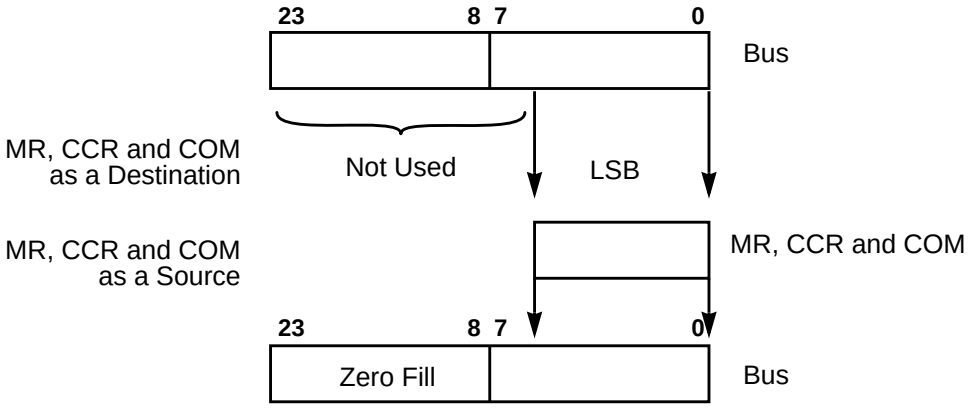


Figure 12-5. Reading and Writing Control Registers

12.2.4 Data Organization in Memory

The 24-bit program memory can store both 24-bit instruction words and instruction extension words. The 48-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 16-bit-wide X and Y memories can store word and byte operands. 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.

12.3 Instruction Groups

The instruction set is divided into the following groups:

- Arithmetic
- Logical

- Bit Manipulation
- Loop
- Move
- Program Control

Each instruction group is described in the following paragraphs.

12.3.1 Arithmetic Instructions

The arithmetic instructions perform all of the arithmetic operations within the Data ALU. 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 results calculated in previous instructions to be stored. The move operation that can be specified in parallel to the instruction marked is one of the parallel instructions listed in **Table 12-8**, “Move Instructions,” on page 12-12. Arithmetic instructions can be executed conditionally, based on the condition codes generated by the previous instructions. Conditional arithmetic instructions do not allow parallel data movement over the various data buses. **Table 12-4** lists the arithmetic instructions.

Table 12-4. Arithmetic Instructions

Mnemonic	Description	Parallel Instruction*
* A √ in the “Parallel Instruction” column means that the instruction is a parallel instruction. A blank table cell indicates that the instruction is not a parallel instruction.		
ABS	Absolute Value	√
ADC	Add Long with Carry	√
ADD	Add	√
ADD (imm.)	Add (immediate operand)	
ADDL	Shift Left and Add	√
ADDR	Shift Right and Add	√
ASL	Arithmetic Shift Left	√
ASL (mb.)	Arithmetic Shift Left (multi-bit)	
ASL (mb., imm.)	Arithmetic Shift Left (multi-bit, immediate operand)	

Table 12-4. Arithmetic Instructions (Continued)

Mnemonic	Description	Parallel Instruction*
* A √ in the “Parallel Instruction” column means that the instruction is a parallel instruction. A blank table cell indicates that the instruction is not a parallel instruction.		
ASR	Arithmetic Shift Right	√
ASR (mb.)	Arithmetic Shift Right (multi-bit)	
ASR (mb., imm.)	Arithmetic Shift Right (multi-bit, immediate operand)	
CLR	Clear an Operand	√
CMP	Compare	√
CMP (imm.)	Compare (immediate operand)	
CMPM	Compare Magnitude	√
CMPU	Compare Unsigned	
DEC	Decrement Accumulator	
DIV	Divide Iteration	
DMAC	Double Precision Multiply-Accumulate	
INC	Increment Accumulator	
MAC	Signed Multiply-Accumulate	√
MAC (su,uu)	Mixed Multiply-Accumulate	
MACI	Signed Multiply-Accumulate (immediate operand)	
MACR	Signed Multiply-Accumulate and Round	√
MACRI	Signed Multiply-Accumulate and Round (immediate operand)	
MAX	Transfer By Signed Value	√
MAXM	Transfer By Magnitude	√
MPY	Signed Multiply	√
MPY (su,uu)	Mixed Multiply	
MPYI	Signed Multiply (immediate operand)	
MPYR	Signed Multiply and Round	√
MPYRI	Signed Multiply and Round (immediate operand)	
NEG	Negate Accumulator	√
NORMF	Fast Accumulator Normalize	

Table 12-4. Arithmetic Instructions (Continued)

Mnemonic	Description	Parallel Instruction*
* A √ in the "Parallel Instruction" column means that the instruction is a parallel instruction. A blank table cell indicates that the instruction is not a parallel instruction.		
RND	Round	√
SBC	Subtract Long with Carry	√
SUB	Subtract	√
SUB (imm.)	Subtract (immediate operand)	
SUBL	Shift Left and Subtract	√
SUBR	Shift Right and Subtract	√
Tcc	Transfer Conditionally	
TFR	Transfer Data ALU Register	√
TST	Test an Operand	√

12.3.2 Logical Instructions

The logical instructions execute in one instruction cycle and perform all logical operations within the Data ALU (except ANDI and ORI). They can affect all of the CCR bits and, like the arithmetic instructions, are register-based. Optional data transfers can 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 results calculated in previous instructions to be stored.The move operation that can be specified in parallel to the instruction marked is one of the parallel instructions listed in **Table 12-8**, "Move Instructions," on page 12-12. **Table 12-5** lists the logical instructions.

Table 12-5. Logical Instructions

Mnemonic	Description	Parallel Instruction*
* A √ in the "Parallel Instruction" column means that the instruction is a parallel instruction. A blank table cell indicates that the instruction is not a parallel instruction.		
AND	Logical AND	√
AND (imm.)	Logical AND (immediate operand)	
ANDI	AND Immediate to Control Register	
CLB	Count Leading Bits	

Table 12-5. Logical Instructions (Continued) (Continued)

Mnemonic	Description	Parallel Instruction*
* A √ in the “Parallel Instruction” column means that the instruction is a parallel instruction. A blank table cell indicates that the instruction is not a parallel instruction.		
EOR	Logical Exclusive OR	√
EOR (imm.)	Logical Exclusive OR (immediate operand)	
EXTRACT	Extract Bit Field	
EXTRACT (imm.)	Extract Bit Field (immediate operand)	
EXTRACTU	Extract Unsigned Bit Field	
EXTRACTU (imm.)	Extract Unsigned Bit Field (immediate operand)	
INSERT	INSERT Bit Field	
INSERT (imm.)	INSERT Bit Field (immediate operand)	
LSL	Logical Shift Left	√
LSL (mb.)	Logical Shift Left (multi-bit)	
LSL (mb., imm.)	Logical Shift Left (multi-bit, immediate operand)	
LSR	Logical Shift Right	√
LSR (mb.)	Logical Shift Right (multi-bit)	
LSR (mb.,imm.)	Logical Shift Right (multi-bit, immediate operand)	
MERGE	Merge Two Half Words	
NOT	Logical Complement	√
OR	Logical Inclusive OR	√
OR (imm.)	Logical Inclusive OR (immediate operand)	
ORI	OR Immediate to Control Register	
ROL	Rotate Left	√
ROR	Rotate Right	√

12.3.3 Bit Manipulation Instructions

The bit manipulation instructions test the state of any single bit in a memory location and then optionally set, clear, or invert the bit. The carry bit of the CCR contains the result of the bit test. **Table 12-6** lists the bit manipulation instructions.

Table 12-6. Bit Manipulation Instructions

Mnemonic	Description	Parallel Instruction*
* A √ in the “Parallel Instruction” column means that the instruction is a parallel instruction. A blank table cell indicates that the instruction is not a parallel instruction.		
BCHG	Bit Test and Change	
BCLR	Bit Test and Clear	
BSET	Bit Test and Set	
BTST	Bit Test	

12.3.4 Loop Instructions

The hardware DO loop executes with no overhead cycles—that is, it runs as fast as straight-line code. Replacing straight-line code with DO loops can significantly reduce program memory usage. The loop instructions control hardware looping either by initiating a program loop and establishing looping parameters or by 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 nest The address of the first instruction in a program loop is also saved to allow no-overhead looping. The ENDDO instruction is not used for normal termination of a DO loop; it terminates a DO loop before the LC is decremented to 1. **Table 12-7** lists the loop instructions.

Table 12-7. Loop Instructions

Mnemonic	Description	Parallel Instruction*
* A √ in the “Parallel Instruction” column means that the instruction is a parallel instruction. A blank table cell indicates that the instruction is not a parallel instruction.		
BRKcc	Conditionally Break the current Hardware Loop	
DO	Start Hardware Loop	
DO FOREVER	Start Forever Hardware Loop	
ENDDO	Abort and Exit from Hardware Loop	

12.3.5 Move Instructions

The move instructions perform data movement over the XDB and YDB or over the GDB. Move instructions, most of which allow Data ALU opcode in parallel, do not affect the CCR, except the limit bit L, if limiting is performed when reading a Data ALU accumulator register. **Table 12-8** lists the move instructions.

Table 12-8. Move Instructions

Mnemonic	Description	Parallel Instruction
* A √ in the “Parallel Instruction” column means that the instruction is a parallel instruction. A blank table cell indicates that the instruction is not a parallel instruction.		
LUA	Load Updated Address	
LRA	Load PC-Relative Address	
MOVE	Move Data Register	√
MOVEC	Move Control Register	
MOVEM	Move Program Memory	
MOVEP	Move Peripheral Data	
U MOVE	Update Move	√
VSL	Viterbi Shift Left	

12.3.6 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. **Table 12-9** lists the program control instructions.

Table 12-9. Program Control Instructions

Mnemonic	Description	Parallel Instruction*
* A √ in the “Parallel Instruction” column means that the instruction is a parallel instruction. A blank table cell indicates that the instruction is not a parallel instruction.		
IFcc.U	Execute Conditionally and Update CCR	
IFcc	Execute Conditionally	
Bcc	Branch Conditionally	
BRA	Branch Always	
BScc	Branch to Subroutine Conditionally	
BSR	Branch to Subroutine Always	
DEBUGcc	Enter into the Debug Mode Conditionally	
DEBUG	Enter into the Debug Mode Always	

Table 12-9. Program Control Instructions (Continued)

Mnemonic	Description	Parallel Instruction*
* A √ in the "Parallel Instruction" column means that the instruction is a parallel instruction. A blank table cell indicates that the instruction is not a parallel instruction.		
Jcc	Jump Conditionally	
JMP	Jump Always	
JCLR	Jump if Bit Clear	
JSET	Jump if Bit Set	
JScc	Jump to Subroutine Conditionally	
JSR	Jump to Subroutine Always	
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)	
TRAPcc	Trap Conditionally	
TRAP	Trap Always	
WAIT	Wait for Interrupt (Low-Power Standby)	

12.4 Guide to Instruction Descriptions

The following information is included in each instruction description:

- *Name and Mnemonic:* Highlighted in **bold** type for easy reference.
- *Assembler Syntax and Operation:* The syntax line for each instruction symbolically describes the corresponding operation. If several operations are indicated on a single line in the operation field, those operations may not occur in the order shown, but are generally assumed to occur in parallel. Any parallel data move is indicated in parentheses in both the assembler syntax and operation fields. An optional letter in the mnemonic appears in parentheses in the assembler syntax field.

- *Description:* Includes any special cases and/or condition code anomalies.
- *Condition Codes:* The Status Register (SR) is depicted with the condition code bits that can be affected by the instruction. Not all bits in the SR are used. Reserved bits are indicated with gray boxes.
- *Instruction Format:* The instruction fields, the instruction opcode, and the instruction extension word are specified in the instruction syntax. Optional extension words are so indicated. The values that can be assumed by each of the variables in the various instruction fields are shown under the instruction field heading.

12.4.1 Notation

Each instruction description contains symbols to abbreviate certain operands and operations. **Table 12-10** lists the symbols and their respective meanings. Depending on the context, registers refer either to the register itself or to the contents of the register.

Table 12-10. Instruction Description Notation

Symbol	Meaning
Data ALU Registers Operands	
Xn	Input Register X1 or X0 (24 bits)
Yn	Input Register Y1 or Y0 (24 bits)
An	Accumulator Registers A2, A1, A0 (A2—8 bits, A1 and A0—24 bits)
Bn	Accumulator Registers B2, B1, B0 (B2—8 bits, B1 and 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)
BA	Accumulators B and A = B1: A1 (48 bits)
A10	Accumulator A = A1: A0 (48 bits)
B10	Accumulator B = B1:B0 (48 bits)
Program Control Unit Registers Operands	
PC	Program Counter Register (24 bits)
MR	Mode Register (8 bits)

Table 12-10. Instruction Description Notation (Continued)

Symbol	Meaning
CCR	Condition Code Register (8 bits)
SR	Status Register = EMR:MR:CCR (24 bits)
EOM	Extended Chip Operating Mode Register (8 bits)
COM	Chip Operating Mode Register (8 bits)
OMR	Operating Mode Register = EOM:COM (24 bits)
SZ	System Stack Size Register (24 bits)
SC	System Stack Counter Register (5 bits)
VBA	Vector Base Address (24 bits, eight set to 0)
LA	Hardware Loop Address Register (24 bits)
LC	Hardware Loop Counter Register (24 bits)
SP	System Stack Pointer Register (24 bits)
SSH	Upper Portion of the Current Top of the Stack (24 bits)
SSL	Lower Portion of the Current Top of the Stack (24 bits)
SS	System Stack RAM = SSH: SSL (16 locations by 32 bits)
Address Operands	
ea	Effective Address
eax	Effective Address for X Bus
eay	Effective Address for Y Bus
xxxxxx	Absolute or Long Displacement Address (24 bits)
xxx	Short or Short Displacement Jump Address (12 bits)
xxx	Short Displacement Jump Address (9 bits)
aaa	Short Displacement Address (7 bits, sign-extended)
aa	Absolute Short Address (6 bits, zero-extended)
pp	High I/O Short Address (6 bits, ones-extended)
qq	Low I/O Short Address (6 bits)
<. . .>	Specifies the Contents of the Specified Address
X:	X Memory Reference
Y:	Y Memory Reference

Table 12-10. Instruction Description Notation (Continued)

Symbol	Meaning
L:	Long Memory Reference = X Concatenated with Y
P:	Program Memory Reference
Miscellaneous Operands	
S, Sn	Source Operand Register
D, Dn	Destination Operand Register
D [n]	Bit n of D Destination Operand Register
#n	Immediate Short Data (5 bits)
#xx	Immediate Short Data (8 bits)
#xxx	Immediate Short Data (12 bits)
#xxxxxx	Immediate Data (24 bits)
r	Rounding Constant
#bbbbb	Operand Bit Select (5 bits)
Unary Operands	
–	Negation Operator
—	Logical NOT Operator (Overbar)
PUSH	Push Specified Value onto the System Stack (SS) Operator
PULL	Pull Specified Value from the SS Operator
READ	Read the Top of the SS Operator
PURGE	Delete the Top Value on the SS Operator
||	Absolute Value Operator
Binary Operands	
+	Addition Operator
–	Subtraction Operator
*	Multiplication Operator
÷, /	Division Operator
+	Logical Inclusive OR Operator
•	Logical AND Operator
⊕	Logical Exclusive OR Operator

Table 12-10. Instruction Description Notation (Continued)

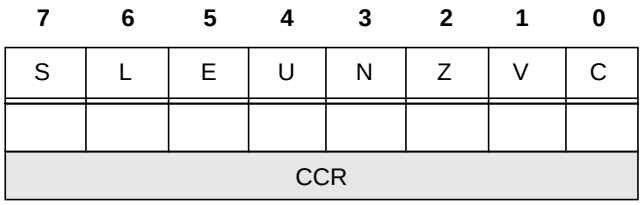
Symbol	Meaning
⇒	"Is Transferred To" Operator
:	Concatenation Operator
Addressing Mode Operators	
<<	I/O Short Addressing Mode Force Operator
<	Short Addressing Mode Force Operator
>	Long Addressing Mode Force Operator
#	Immediate Addressing Mode Operator
#>	Immediate Long Addressing Mode Force Operator
#<	Immediate Short Addressing Mode Force Operator
Mode Register Symbols	
LF	Loop Flag Bit Indicating When a DO Loop is in Progress
DM	Double-Precision Multiply bit indicating whether the chip is in Double-Precision Multiply mode
SB	Sixteen-Bit Arithmetic Mode
RM	Rounding Mode
S1, S0	Scaling Mode Bits Indicating the Current Scaling Mode
I1, I0	Interrupt Mask Bits Indicating the Current Interrupt Priority Level
Condition Code Register (CCR) Symbols	
S	Block Floating Point Scaling Bit Indicating Data Growth Detection
L	Limit Bit Indicating Arithmetic Overflow and/or Data Shifting/Limiting
E	Extension Bit Indicating if the Integer Portion of Data ALU result is in Use
U	Unnormalized Bit Indicating if the Data ALU Result is Unnormalized
N	Negative Bit Indicating if Bit 55 of the Data ALU Result is Set
Z	Zero Bit Indicating if the Data ALU Result Equals Zero
V	Overflow Bit Indicating whether Arithmetic Overflow occurred in Data ALU
C	Carry Bit Indicating if a Carry or Borrow occurred in Data ALU Result
( )	Optional Letter, Operand, or Operation
(. . .)	Any Arithmetic or Logical Instruction that Allows Parallel Moves

Table 12-10. Instruction Description Notation (Continued)

Symbol	Meaning
EXT	Extension Register Portion of an Accumulator (A2 or B2)
LS	Least Significant
LSP	Least Significant Portion of an Accumulator (A0 or B0)
MS	Most Significant
MSP	Most Significant Portion of a n Accumulator (A1 or B1)
S/L	Shifting and/or Limiting on a Data ALU Register
Sign Ext	Sign Extension of a Data ALU Register
Zero	Zeroing of a Data ALU Register
Address ALU Registers Operands	
Rn	Address Registers R0–R7 (24 bits)
Nn	Address Offset Registers N0–N7 (24 bits)
Mn	Address Modifier Registers M0–M7 (24 bits)

12.4.2 Condition Code Computation

The Condition Code Register (CCR) portion of the Status Register (SR) consists of eight bits (see **Figure 12-6**). The E, U, N, Z, V, and C bits are true condition code bits that reflect the condition of the result of a Data ALU operation. These condition code bits are not “sticky” and are not affected by Address ALU calculations or by data transfers over the XDB, YDB, or GDB. The L bit is a “sticky” overflow bit that indicates an overflow in the Data ALU or data limiting when the contents of the A and/or B accumulators are moved. The S bit is a “sticky” bit used in block floating-point operations to indicate the need to scale the number in A or B.



S — Scaling bit
L — Limit bit
E — Extension bit
U — Unnormalized bit
N — Negative bit
Z — Zero bit
V — Overflow bit
C — Carry bit

Figure 12-6. Condition Code Register (CCR)

Every instruction contains an illustration showing how the instruction affects the various condition codes. An instruction can affect a condition code according to three different rules, as described in **Table 12-11**.

Table 12-11. Instruction Effect on Condition Code

Standard Mark	Effect on the Condition Code
—	This bit is unchanged by the instruction.
√	This bit is changed by the instruction, according to the standard definition of the condition code.
*	This bit is changed by the instruction, according to a special definition of the condition code depicted as part of the instruction description.

Table 12-12. Condition Code Register (CCR) Bit Definitions

<table>
<tr><th>Bit Number</th><th>Bit Name</th><th>Reset Value</th><th>Description</th></tr>
<tr><td>7</td><td>S</td><td>0</td><td>Scaling
Computed, according to the logical equations shown here when an instruction or a parallel move reads the contents of accumulator A or B to XDB or YDB. The S bit is a “sticky” bit, cleared only by an instruction that specifically clears it or by hardware reset.
<table>
<tr><th>S0</th><th>S1</th><th>Scaling Mode</th><th>S Bit Equation</th></tr>
<tr><td>0</td><td>0</td><td>No scaling</td><td>S = (A46 XOR A45) OR (B46 XOR B45) OR S (previous)</td></tr>
<tr><td>0</td><td>1</td><td>Scale up</td><td>S = (A47 XOR A46) OR (B47 XOR B46) OR S (previous)</td></tr>
<tr><td>1</td><td>0</td><td>Scale down</td><td>S = (A45 XOR A44) OR (B45 XOR B44) OR S (previous)</td></tr>
<tr><td>1</td><td>1</td><td>Reserved</td><td>S undefined</td></tr>
</table></td></tr>
<tr><td>7 cont.</td><td>S</td><td>0</td><td>Scaling cont.
The S bit detects data growth, which is required in Block Floating-Point FFT operation. The S bit is set if the absolute value in the accumulator, before scaling, is greater than or equal to 0.25 and smaller than 0.75. Typically, the bit is tested after each pass of a radix 2 decimation-in-time FFT and, if it is set, the appropriate scaling mode should be activated in the next pass. The Block Floating-Point FFT algorithm is described in the Motorola application note APR4/D, Implementation of Fast Fourier Transforms on Motorola’s DSP56000/DSP56001 and DSP96002 Digital Signal Processors.</td></tr>
</table>

Table 12-12. Condition Code Register (CCR) Bit Definitions (Continued)

<table>
<tr><th>Bit Number</th><th>Bit Name</th><th>Reset Value</th><th>Description</th></tr>
<tr><td>6</td><td>L</td><td>0</td><td>Limit
Set if the Overflow bit (V) is set or if an instruction or a parallel move causes the data shifter/limiters to perform a limiting operation while reading the contents of accumulator A or B to the XDB or YDB bus. In Arithmetic Saturation mode, the limit bit is also set when an arithmetic saturation occurs in the Data ALU result. Not affected otherwise. The L bit is “sticky” and must be cleared only by an instruction that specifically clears it or by hardware reset.</td></tr>
<tr><td>5</td><td>E</td><td>0</td><td>Extension
Cleared if all the bits of the signed integer portion of the Data ALU result are the same (i.e., the bit patterns are either 00. . . 00 or 11. . . 11). Otherwise, this bit is set. The signed integer portion is defined by the scaling mode, as shown here.
<table>
<tr><th>S0</th><th>S1</th><th>Scaling Mode</th><th>S Bit Equation</th></tr>
<tr><td>0</td><td>0</td><td>No scaling</td><td>Bits 55, 54.....48, 47</td></tr>
<tr><td>0</td><td>1</td><td>Scale down</td><td>Bits 55, 54.....49, 48</td></tr>
<tr><td>1</td><td>0</td><td>Scale up</td><td>Bits 55, 54.....47.46</td></tr>
</table>
The signed integer portion of an accumulator is not necessarily the same as its extension register portion. It consists of the most significant 8, 9, or 10 bits of that accumulator, depending on the scaling mode. The extension register portion of an accumulator (A2 or B2) is always the eight Most Significant Bits of that accumulator. The E bit refers to the signed integer portion of an accumulator and not the extension register portion of that accumulator. For example, if the current scaling mode is set for no scaling (S1 = S0 = 0), the signed integer portion of the A or B accumulator consists of bits 47 through 55. If the A accumulator contained the signed 56-bit value $00:800000:000000 as a result of a Data ALU operation, the E bit would be set (E = 1) since the 9 Most Significant Bits of that accumulator are not all the same (i.e., neither 00...00 nor 11...11). Thus, data limiting occurs if that 56-bit value is specified as a source operand in a move-type operation. This limiting operation results in either a positive or negative 24-bit or 48-bit saturation constant stored in the specified destination. The signed integer portion of an accumulator and the extension register portion of an accumulator are the same only in the “Scale Down” scaling mode (i.e., S1 = 0 and S0 = 1).</td></tr>
</table>

Table 12-12. Condition Code Register (CCR) Bit Definitions (Continued)

<table>
<tr><th>Bit Number</th><th>Bit Name</th><th>Reset Value</th><th>Description</th></tr>
<tr><td>4</td><td>U</td><td>0</td><td>Unnormalized
Set if the two Most Significant Bits of the Most Significant Portion (MSP) of the Data ALU result are the same. This bit is cleared otherwise. The MSP is defined by the scaling mode. The U bit is computed as shown here. The result of calculating the U bit in this fashion is that the definition of a positive normalized number p is 0.5 ≤ p < 1.0 and the definition of negative normalized number n is –1.0 ≤ n < –0.5.
<table>
<tr><th>S1</th><th>S0</th><th>Scaling Mode</th><th>U Bit Computation</th></tr>
<tr><td>0</td><td>0</td><td>No Scaling</td><td>U = $\overline{\text{(Bit 47 xor Bit 46)}}$</td></tr>
<tr><td>0</td><td>1</td><td>Scale Down</td><td>U = $\overline{\text{(Bit 48 xor Bit 47)}}$</td></tr>
<tr><td>1</td><td>0</td><td>Scale Up</td><td>U = $\overline{\text{(Bit 46 xor Bit 45)}}$</td></tr>
</table></td></tr>
<tr><td>3</td><td>N</td><td>0</td><td>Negative
Set if the MS bit (Bit 55 in arithmetic instructions or Bit 47 in logical instructions) of the Data ALU result is set. Otherwise, this bit is cleared.</td></tr>
<tr><td>2</td><td>Z</td><td>0</td><td>Zero
Set if the Data ALU result equals 0. Otherwise, this bit is cleared.</td></tr>
<tr><td>1</td><td>V</td><td>0</td><td>Overflow
Set if an arithmetic overflow occurs in the 56-bit Data ALU result. Otherwise, this bit is cleared. This indicates that the result cannot be represented in the 56-bit accumulator, so the accumulator overflows. In Arithmetic Saturation mode, an arithmetic overflow occurs if the Data ALU result is not representable in the accumulator without the extension part (i.e., 48-bit accumulator; 32-bit in the Sixteen Bit mode).</td></tr>
<tr><td>0</td><td>C</td><td>0</td><td>Carry
Set if a carry is generated out of the MSB of the Data ALU result of an addition or if a borrow is generated out of the MSB of the Data ALU result of a subtraction. Otherwise, this bit is cleared. The carry or borrow is generated out of Bit 55 of the Data ALU result. The C bit is also affected by bit manipulation, rotate, shift, and compare instructions. The C bit is not affected by Arithmetic Saturation mode.</td></tr>
</table>

12.5 Instruction Partial Encoding

This section gives the encodings for the following:

- Various groupings of registers used in the instruction encodings
- Condition Code combinations
- Addressing
- Addressing modes

The symbols used in decoding the various fields of an instruction are identical to those used in the Opcode section of the individual instruction descriptions.

12.5.1 Partial Encodings for Use in Instruction Encoding

Table 12-13. Partial Encodings for Use in Instruction Encoding

Destination Accumulator Encoding		**Data ALU Operands Encoding 1**		**Data ALU Source Operands Encoding**	
D/S	**d/S/D**	**S**	**J**	**S**	**JJ**
A	0	X	0	X0	00
B	1	Y	1	Y0	01
				X1	10
				Y1	11
Program Control Unit Register Encoding		**Data ALU Operands Encoding 2**		**Effective Addressing Mode Encoding 1**	
Register	**EE**	**S**	**JJJ**	(Rn)–Nn	0 0 0 r r r
MR	00	B/A*	0 0 1	(Rn)+Nn	0 0 1 r r r
CCR	01	X	0 1 0	(Rn)–	0 1 0 r r r
COM	10	Y	0 1 1	(Rn)+	0 1 1 r r r
EOM	11	X0	1 0 0	(Rn)	1 0 0 r r r
		Y0	1 0 1	(Rn+Nn)	1 0 1 r r r
		X1	1 1 0	–(Rn)	1 1 1 r r r
		Y1	1 1 1	Absolute address	1 1 0 0 0 0
		* The source accumulator is B if the destination accumulator (selected by the **d** bit in the opcode) is A, or A if the destination accumulator is B.		Immediate data	1 1 0 1 0 0
				“r r r” refers to an address register R0–R7	

Table 12-13. Partial Encodings for Use in Instruction Encoding

Data ALU Operands Encoding 3

SSS/sss	S,D	qqq	S,D	ggg	S,D
000	reserved	000	reserved	000	B/A*
001	reserved	001	reserved	001	reserved
010	A1	010	A0	010	reserved
011	B1	011	B0	011	reserved
100	X0	100	X0	100	X0
101	Y0	101	Y0	101	Y0
110	X1	110	X1	110	X1
111	Y1	111	Y1	111	Y1

* The selected accumulator is B if the source two accumulator (selected by the **d** bit in the opcode) is A, or A if the source two accumulator is B.

Memory/Peripheral Space		Effective Addressing Mode Encoding 2		Effective Addressing Mode Encoding 3	
Space	**S**	**Mode**	**MMMRRR**	**Mode**	**MMMRRR**
X Memory	0	(Rn)–Nn	0 0 0 r r r	(Rn)–Nn	0 0 0 r r r
Y Memory	1	(Rn)+Nn	0 0 1 r r r	(Rn)+Nn	0 0 1 r r r
		(Rn)–	0 1 0 r r r	(Rn)–	0 1 0 r r r
		(Rn)+	0 1 1 r r r	(Rn)+	0 1 1 r r r
		(Rn)	1 0 0 r r r	(Rn)	1 0 0 r r r
		(Rn+Nn)	1 0 1 r r r	(Rn+Nn)	1 0 1 r r r
		–(Rn)	1 1 1 r r r	–(Rn)	1 1 1 r r r
		Absolute address	1 10 0 0 0		
		"r r r" refers to an address register R0–R7			

Effective Addressing Mode Encoding 4		Six-Bit Encoding for All On-Chip Registers		
Mode	**MMRRR**	**Destination Register**	**D D D D D D / d d d d d d**	
(Rn)–Nn	0 0 r r r	4 registers in Data ALU	0 0 0 1 D D	
(Rn)+Nn	0 1 r r r	8 accumulators in Data ALU	0 0 1 D D D	
(Rn)–	1 0 r r r	8 address registers in AGU	0 1 0 T T T	
(Rn)+	1 1 r r r	8 address offset registers in AGU	0 1 1 N N N	
"r r r" refers to an address register R0–R7		8 address modifier registers in AGU	1 0 0 F F F	
		1 address register in AGU	1 0 1 E E E	
		2 program controller register	1 1 0 V V V	
		8 program controller registers	1 1 1 G G G	
		See **Table 12-14** for the specific encodings.		

Table 12-14. Triple-Bit Register Encoding

Code	1DD	DDD	TTT	NNN	FFF	EEE	VVV	GGG
000	—	A0	R0	N0	M0	—	VBA	SZ
001	—	B0	R1	N1	M1	—	SC	SR
010	—	A2	R2	N2	M2	EP	—	OMR
011	—	B2	R3	N3	M3	—	—	SP
100	X0	A1	R4	N4	M4	—	—	SSH
101	X1	B1	R5	N5	M5	—	—	SSL
110	Y0	A	R6	N6	M6	—	—	LA
111	Y1	B	R7	N7	M7	—	—	LC

Table 12-15. Long Move Register Encoding

S	S1	S2	S S/L	D	D1	D2	D Sign Ext	D Zero	LLL
A10	A1	A0	no	A10	A1	A0	no	no	0 0 0
B10	B1	B0	no	B10	B1	B0	no	no	0 0 1
X	X1	X0	no	X	X1	X0	no	no	0 1 0
Y	Y1	Y0	no	Y	Y1	Y0	no	no	0 1 1
A	A1	A0	yes	A	A1	A0	A2	no	1 0 0
B	B1	B0	yes	B	B1	B0	B2	no	1 0 1
AB	A	B	yes	AB	A	B	A2,B2	A0,B0	1 1 0
BA	B	A	yes	BA	B	A	B2,A2	B0,A0	1 1 1

Table 12-16. Partial Encodings for Use in Instructions Encoding, 2

Data ALU Source Registers Encoding		AGU Address and Offset Registers Encoding		
S	**JJJ**	**Destination Address Register D**	**dddd**	
B/A*	000	R0-R7	onnn	
X0	100	N0-N7	1nnn	
Y0	101			
X1	110			
Y1	111			

Table 12-16. Partial Encodings for Use in Instructions Encoding, 2

Data ALU Multiply Operands Encoding 1				Data ALU Multiply Operands Encoding 2	
S1 * S2	**Q Q Q**	**S1 * S2**	**Q Q Q**	**S**	**Q Q**
X0,X0	0 0 0	X0,Y1	1 0 0	Y1	0 0
Y0,Y0	0 0 1	Y0,X0	1 0 1	X0	0 1
X1,X0	0 1 0	X1,Y0	1 1 0	Y0	1 0
Y1,Y0	0 1 1	Y1,X1	1 1 1	X1	1 1
NOTE: Only the indicated S1 * S2 combinations are valid. X1 * X1 and Y1 * Y1 are not valid.					

Data ALU Multiply Operands Encoding 3		Data ALU Multiply Operands Encoding 4			
S	**qq**	**S1*S2**	**Q Q Q Q**	**S1*S2**	**Q Q Q Q**
X0	0 0	X0,X0	0 0 0 0	X0,Y1	0 1 0 0
Y0	0 1	Y0,Y0	0 0 0 1	Y0,X0	0 1 0 1
X1	1 0	X1,X0	0 0 1 0	X1,Y0	0 1 1 0
Y1	1 1	Y1,Y0	0 0 1 1	Y1,X1	0 1 1 1
Data ALU Multiply Sign Encoding		X1,X1	1 0 0 0	Y1,X0	1 1 0 0
Sign	**k**	Y1,Y1	1 0 0 1	X0,Y0	1 1 0 1
+	0	X0,X1	1 0 1 0	Y0,X1	1 1 1 0
–	1	Y0,Y1	1 0 1 1	X1,Y1	1 1 1 1

Five-Bit Register Encoding 1				Write Control Encoding	
D/S	**ddddd / eeeee**	**D/S**	**ddddd / eeeee**	**Operation**	**W**
X0	0 0 1 0 0	B2	0 1 0 1 1	Read Register or Peripheral	0
X1	0 0 1 0 1	A1	0 1 1 0 0	Write Register or Peripheral	1
Y0	0 0 1 1 0	B1	0 1 1 0 1	**ALU Registers Encoding**	
Y1	0 0 1 1 1	A	0 1 1 1 0	**Destination Register**	**D D D D**
A0	0 1 0 0 0	B	0 1 1 1 1	4 registers in Data ALU	0 1 D D
B0	0 1 0 0 1	R0-R7	1 0 r r r	8 accumulators in Data ALU	1 D D D
A2	0 1 0 1 0	N0-N7	1 1 n n n	See **Table 12-14**, “Triple-Bit Register Encoding,” on page 12-24 for the specific encodings.	
“r r r” = Rn number, “n n n” = Nn number					

Table 12-16. Partial Encodings for Use in Instructions Encoding, 2

Immediate Data ALU Operand Encoding			Write Control Encoding	
n	**ssss**	**constant**	**Operation**	**W**
1	00001	010000000000000000000000	Read Register or Peripheral	0
2	00010	001000000000000000000000	Write Register or Peripheral	1
3	00011	000100000000000000000000	**ALU Registers Encoding**	
4	00100	000010000000000000000000	**Destination Register**	**D D D D**
5	00101	000001000000000000000000	4 registers in Data ALU	0 1 D D
6	00110	000000100000000000000000	8 accumulators in Data ALU	1 D D D
7	00111	000000010000000000000000	See **Table 12-14** on page 12-24 for the specific encodings.	
8	01000	000000001000000000000000	**X:Y: Move Operands Encoding**	
9	01001	000000000100000000000000	**X Effective Addressing Mode**	**MMRRR**
10	01010	000000000010000000000000	(Rn)+Nn	0 1 s s s
11	01011	000000000001000000000000	(Rn)–	1 0 s s s
12	01100	000000000000100000000000	(Rn)+	1 1 s s s
13	01101	000000000000010000000000	(Rn)	0 0 s s s
14	01110	000000000000001000000000	**Y Effective Addressing Mode**	**mmrr**
15	01111	000000000000000010000000000	(Rn)+Nn	0 1 t t
16	10000	000000000000000001000000000	(Rn)–	1 0 t t
17	10001	000000000000000001000000	(Rn)+	1 1 t t
18	10010	000000000000000000100000	(Rn)	0 0 t t
19	10011	000000000000000000010000	where the following apply: “s s s” refers to an address register R0–R7 and “t t” refers to an address register R4–R7 or R0–R3 in the opposite address register bank from that used in the X effective address	
20	10100	000000000000000000001000		
21	10101	000000000000000000000100		
22	10110	000000000000000000000010		

Table 12-16. Partial Encodings for Use in Instructions Encoding, 2

X:R Operand Registers Encoding				**Signed/Unsigned Partial Encoding 1**	
S1,D1	**f f**	**D2**	**F**	**ss/su/uu**	**ss**
X0	0 0	Y0	0	ss	00
X1	0 1	Y1	1	su	10
A	1 0			uu	11
B	1 1			(Reserved)	01
R:Y Operand Registers Encoding				**Signed/Unsigned Partial Encoding 2**	
D1	e	S2,D2	f f	**su/uu**	**s**
X0	0	Y0	0 0	su	0
X1	1	Y1	0 1	uu	1
		A	1 0		
		B	1 1		
Single-Bit Special Register Encoding				**Five-Bit Register Encoding 2**	
d	**X:R Class II Opcode**	**R:Y Class II Opcode**		**S1,D1**	**ddddd**
0	A → X:<ea> , X0 → A	Y0 → A , A → Y:<ea>		M0-M7	00nnn
1	B → X:<ea> , X0 → B	Y0 → B , B → Y:<ea>		EP	01010
Move Operand Encoding				VBA	10000
S1,D1	e e	S2,D2	f f	SC	10001
X0	0 0	Y0	0 0	SZ	11000
X1	0 1	Y1	0 1	SR	11001
A	1 0	A	1 0	OMR	11010
B	1 1	B	1 1	SP	11011
				SSH	11100
				SSL	11101
				LA	11110
				LC	11111
				where “n n n” = Mn number (M0–M7)	

Table 12-17. Condition Code Computation Equation

	"cc" Mnemonic	Condition
CC(HS)	Carry Clear (higher or same)	C = 0
CS(LO)	Carry Set (lower)	C = 1
EC	Extension Clear	E = 0
EQ	Equal	Z = 1
ES	Extension Set	E=1
GE	Greater than or Equal	$N \oplus V=0$
GT	Greater Than	$Z+(N \oplus V)=0$
LC	Limit Clear	L=0
LE	Less than or Equal	$Z+(N \oplus V)=1$
LS	Limit Set	L=1
LT	Less Than	$N \oplus V=1$
MI	Minus	N=1
NE	Not Equal	Z=0
NR	Normalized	$Z+(\overline{U} \bullet \overline{E})=1$
PL	Plus	N=0
NN	Not Normalized	$Z+(\overline{U} \bullet \overline{E})=0$

NOTES:
$\overline{U}$ denotes the logical complement of U.
+ denotes the logical OR operator.
● denotes the logical AND operator.
$\oplus$ denotes the logical Exclusive OR operator.

Table 12-18. Condition Codes Encoding

Mnemonic	C C C C	Mnemonic	C C C C
CC(HS)	0 0 0 0	CS(LO)	1 0 0 0
GE	0 0 0 1	LT	1 0 0 1
NE	0 0 1 0	EQ	1 0 1 0
PL	0 0 1 1	MI	1 0 1 1

Table 12-18. Condition Codes Encoding (Continued)

Mnemonic	C C C C	Mnemonic	C C C C
NN	0 1 0 0	NR	1 1 0 0
EC	0 1 0 1	ES	1 1 0 1
LC	0 1 1 0	LS	1 1 1 0
GT	0 1 1 1	LE	1 1 1 1
The condition code computation equations are listed in **Table 12-17.** on page 12-28.			

12.5.2 Parallel Instruction Encoding of the Operation Code

The operation code encoding for the instructions that allow parallel moves is divided into the multiply and non-multiply instruction encodings shown in the following subsections.

12.5.2.1 Multiply Instruction Encoding

The 8-bit operation code for multiply instructions allowing parallel moves has different fields than the non-multiply instruction operation code. The 8-bit operation code = **1QQQ dkkk** where

- QQQ =selects the inputs to the multiplier (see **Table 12-17**, “Condition Code Computation Equation,” on page 12-28)
- kkk = three unencoded bits k2, k1, k0
- d = destination accumulator
  d = 0 → A
  d = 1 → B

Table 12-19. Operation Code K0–2 Decode

Code	k2	k1	k0
0	positive	mpy only	don’t round
1	negative	mpy and acc	round

12.5.2.2 Non-Multiply Instruction Encoding

The 8-bit operation code for instructions allowing parallel moves contains two 3-bit fields defining which instruction the operation code represents and one bit defining the destination accumulator register. The 8-bit operation code = **0 J J J D k k k w**here

- J J J = 1/2 instruction number
- k k k = 1/2 instruction number
- D = 0 → A
  D = 1 → B

Table 12-20. Non-Multiply Instruction Encoding

J J J	D = 0 Src Oper	D = 1 Src Oper	k k k							
			0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
0 0 0	B	A	MOVE[1]	TFR	ADDR	TST	*	CMP	SUBR	CMPM
0 0 1	B	A	ADD	RND	ADDL	CLR	SUB	*	SUBL	NOT
0 1 0	B	A	—	—	ASR	LSR	—	—	ABS	ROR
0 1 1	B	A	—	—	ASL	LSL	—	—	NEG	ROL
0 1 0	X1 X0	X1 X0	ADD	ADC	—	—	SUB	SBC	—	—
0 1 1	Y1 Y0	Y1 Y0	ADD	ADC	—	—	SUB	SBC	—	—
1 0 0	X0_0	X0_0	ADD	TFR	OR	EOR	SUB	CMP	AND	CMPM
1 0 1	Y0_0	Y0_0	ADD	TFR	OR	EOR	SUB	CMP	AND	CMPM
1 1 0	X1_0	X1_0	ADD	TFR	OR	EOR	SUB	CMP	AND	CMPM
1 1 1	Y1_0	Y1_0	ADD	TFR	OR	EOR	SUB	CMP	AND	CMPM

NOTES:
1. Special case 1.
2. * = Reserved

Table 12-21. Special Case1

O P E R C O D E	Operation
0 0 0 0 0 0 0 0	MOVE
0 0 0 0 1 0 0 0	reserved