Appendix A INSTRUCTION SET

A-1 INTRODUCTION

The programming model indicates that the DSP56300 Core central processor architecture can be viewed as three functional units operating in parallel: data arithmetic logic unit (ALU), address generation unit (AGU), and program control unit. The goal of the instruction set is to provide the capability to keep each of these units busy each instruction cycle, achieving maximum speed and minimum program size.

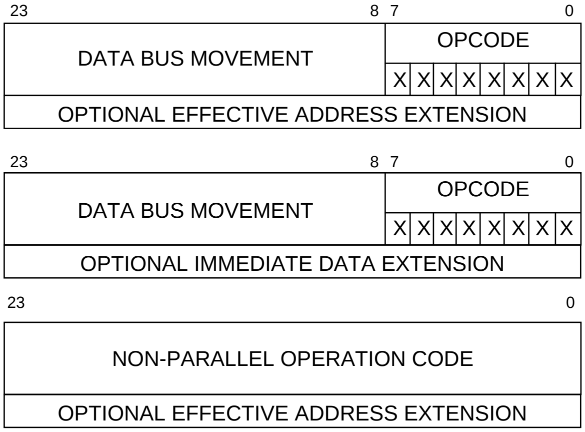
This section introduces the DSP56300 Core instruction set and instruction format. The complete range of instruction capabilities combined with the flexible addressing modes used in this processor provide a very powerful assembly language for implementing digital signal processing (DSP) algorithms. The 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.

A-2 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 effective address extension word or an immediate data extension word. General formats of the instruction word are shown in Figure A-1 Most instructions specify data movement on the XDB, YDB, and data ALU operations in the same operation word. The DSP56300 Core is designed to perform each of these operations in parallel.

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

Figure A-1. General Formats of an Instruction Word


23
8
7
0
DATA BUS MOVEMENT
OPCODE
X X X X X X X X
OPTIONAL EFFECTIVE ADDRESS EXTENSION
23
8
7
0
DATA BUS MOVEMENT
OPCODE
X X X X X X X X
OPTIONAL IMMEDIATE DATA EXTENSION
23
0
NON-PARALLEL OPERATION CODE
OPTIONAL EFFECTIVE ADDRESS EXTENSION

The opcode field of the operation word specifies the data ALU operation or the program control unit operation to be performed.

The operation codes form a very versatile microcontroller unit (MCU) style instruction set, providing highly parallel operations in most programming situations.

The instruction syntax has two formats - Parallel and NonParallel, as shown in Table A-1 and Table A-2. Parallel instruction is organized into five columns: opcode, operands, and two parallel-move fields, each of them is optional, and an optional condition field. The condition field is used to disable the execution of the opcode if the condition is not true, and cannot be used in conjunction with the parallel move fields. Assembly-language source codes for some typical one-word instructions are shown in Table A-1. Because of the multiple bus structure and the parallelism of the DSP56300 Core, 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.). 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/or YDB and the associated addressing modes. The address space qualifiers (X:, Y:, and L:) indicate which address space is being referenced.

Table A-1. Parallel Instructions 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			IF eq

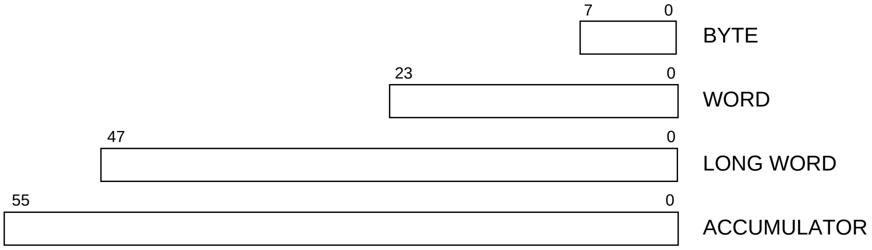
NonParallel instruction is basically organized into two columns: opcode and operands. Assembly-language source codes for some typical one-word instructions are shown in Table A-2. Nonparallel instructions include all the program control, looping and peripherals read/write instructions. They also include some Data ALU instructions that are impossible to be encoded in the opcode field of the Parallel format.

Table A-2. NonParallel Instructions Format

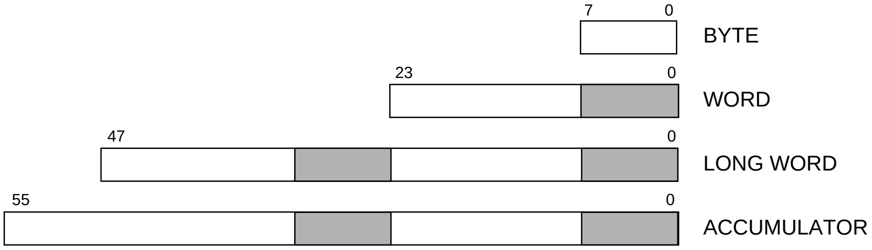
	Opcode	Operands
Example 1:	JEQ	(R5)
Example 2:	MOVEP	#data,X:ipr
Example 3:	RTS	

A-2.1 Operand Sizes

Operand sizes are defined as follows: a byte is 8 bits long, a word is 24 bits long, a long word is 48 bits long, and an accumulator is 56 bits long (see following diagram). The operand size for each instruction is either explicitly encoded in the instruction or implicitly defined by the instruction operation.



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



A-2.2 Data Organization in Registers

The ten data ALU registers support 8- or 24-bit data operands, and 16-bit data in Sixteen Bit mode. Instructions also support 48- or 56-bit data operands (32- or 40-bit in Sixteen Bit mode) by concatenating groups of specific data ALU registers. The eight address registers in the AGU support 24-bit address or data operands. The eight AGU offset registers support 24-bit offsets or may support 24-bit address or data operands. The eight AGU modifier registers support 24-bit modifiers or may support 24-bit address or data operands. The program counter (PC) supports 24-bit address operands. The status register (SR) and operating mode register (OMR) support 8,16 or 24-bit data operands. Both the loop counter (LC) and loop address (LA) registers support 24-bit address operands.

A-2.3 Data ALU Registers

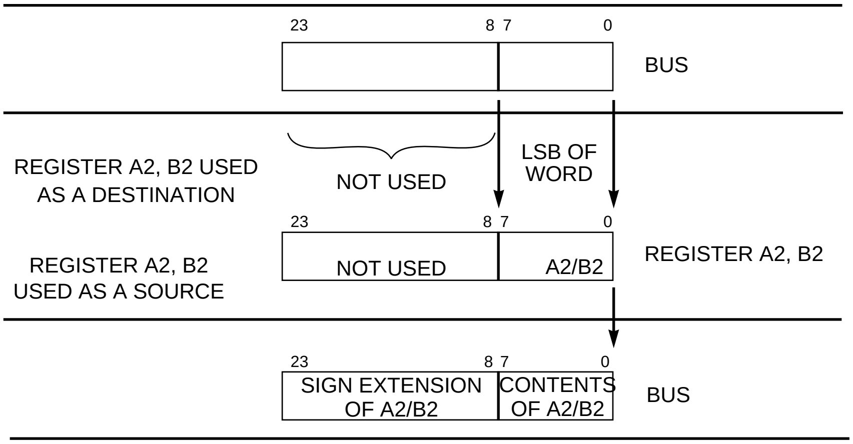
The eight main data registers are 24 bits wide. Word operands occupy one register; long-word operands occupy two concatenated registers. The least significant bit (LSB) is the right-most bit (bit 0); whereas, the most significant bit (MSB) is the left-most bit (bit 23 for word operands and bit 47 for long-word operands). When in Sixteen Bit mode, the least significant bit (LSB) is bit 8; bits 24 to 31 are ignored for long-word operands; whereas, the most significant bit (MSB) is the left-most bit.

The two accumulator extension registers are eight bits wide. When an accumulator extension register is used as 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 Table A-2). 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 for 24 bit mode and bit 8 for Sixteen Bit mode), and the MSB is the left-most 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 system status register (SR). 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 is latched.

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 MS bit of the source operand (bit 23) and appending the source operand with 24 LS zeros. Note that for 24-bit source operands both the automatic sign-extension and zeroing features may be disabled by specifying the destination register to be one of the individual 24-bit accumulator registers (A1 or B1).

Figure A-2. Reading and Writing the ALU Extension Registers



When in Sixteen Bit mode, the move operations associated with Data ALU registers are altered. For further details refer to Section 3.4.1.

A-2.4 AGU Registers

The 24 AGU registers, which are 24 bits wide, may be accessed as word operands for address, address offset, address modifier, and data storage. The notation Rn is used to designate one of the eight address registers, R0–R7; the notation Nn is used to designate one of the eight address offset registers, N0–N7; and the notation Mn is used to designate one of the eight address modifier registers, M0–M7.

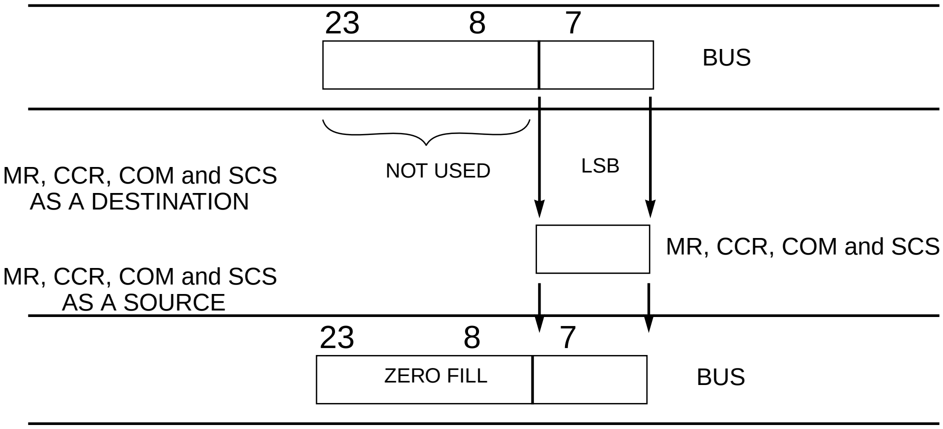
A-2.5 Program Control Registers

The 24-bit OMR has the chip operating mode register (COM) occupying the low-order eight bits and the extended chip operating mode register (EOM) occupying the middle-order eight bits and the system stack control status register (SCS) occupying the high-order eight bits. The Operating Mode Register (OMR) and the Vector Base Address (VBA) are accessed as word operands; however, not all of their bits are defined. These bits read as zero and should be written with zero for future compatibility. The 24-bit SR has the user condition code register (CCR) occupying the low-order eight bits and the system mode register (MR) occupying the middle-order eight bits and the extended mode register (EMR) occupying the high-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 A-3). 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 may also be referenced explicitly (by all PC-relative operation codes) also as a word operand.

A-2.6 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 24-bit-wide X and Y memories can store word and byte operands. When in Sixteen Bit Arithmetic mode the X and Y memories can store 16-bit words, that occupy the low-portion of the memory word. 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.

Figure A-3. Reading and Writing Control Registers



A-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.

A-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 allows results calculated in previous instructions to be stored.A ✔ sign in a table cell in the “Parallel Instruction” column indicates that the corresponding instruction is a parallel instruction, while a blank table cell indicates that the instruction is not a parallel instruction. The move operation that can be specified in parallel to the instruction marked is one of the parallel instructions listed in Table A-7. Arithmetic instructions can be executed conditionally, based on the condition codes generated by the previous instructions. Conditional arithmetic instructions don’t allow parallel data movement over the various data busses. Table A-3 lists the arithmetic instructions.

Table A-3. Arithmetic Instructions

Mnemonic	Description	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)	
ASR	Arithmetic Shift Right	✔
ASR (mb.)	Arithmetic Shift Right (multi-bit)	

Mnemonic	Description	Parallel Instruction
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	✔
NORM	Normalize	
NORMF	Fast Accumulator Normalize	

Mnemonic	Description	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	✔

A-3.2 Logical Instructions

The logical instructions, which execute in one instruction cycle, 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. 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. A ✔ sign in a table cell in the “Parallel Instruction” column indicates that the corresponding instruction is a parallel instruction, while a blank table cell indicates that the instruction is not a parallel instruction. The move operation that can be specified in parallel to the instruction marked is one of the parallel instructions listed in Table A-7. Table A-4 lists the logical instructions.

Table A-4. Logical Instructions

Mnemonic	Description	Parallel Instruction
AND	Logical AND	✔
AND (imm.)	Logical AND (immediate operand)	
ANDI	AND Immediate to Control Register	
CLB	Count Leading Bits	
EOR	Logical Exclusive OR	✔
EOR (imm.)	Logical Exclusive OR (immediate operand)	

Mnemonic	Description	Parallel Instruction
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	✔

A-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 will contain the result of the bit test. Table A-5 lists the bit manipulation instructions.

Table A-5. Bit Manipulation Instructions

Mnemonic	Description	Parallel Instruction
BCHG	Bit Test and Change	
BCLR	Bit Test and Clear	
BSET	Bit Test and Set	
BTST	Bit Test	

A-3.4 Loop Instructions

The hardware DO loop executes with no overhead cycles – 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. Table A-6 lists the loop instructions.

Table A-6. Loop Instructions

Mnemonic	Description	Parallel Instruction
BRKcc	Conditionally Break the current Hardware Loop	
DO	Start Hardware Loop	
DOR	Start Hardware Loop to PC-Related End-Of-Loop Location	
DO FOREVER	Start Forever Hardware Loop	
DOR FOREVER	Start Forever Hardware Loop to PC-Related End-Of-Loop Location	
ENDDO	Abort and Exit from Hardware Loop	

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.

A-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 A-7 lists the move instructions.

Table A-7. Move Instructions

Mnemonic	Description	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	✔

A-3.6 Program Control Instructions

The program control instructions include jumps, conditional jumps, and other instructions affecting the PC, SS and the program Cache. 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 A-8 lists the program control instructions.

Table A-8. Program Control Instructions

Mnemonic	Description	Parallel Instruction
IFcc.U	Execute Conditionally and Update CCR	
IFcc	Execute Conditionally	
Bcc	Branch Conditionally	
BRA	Branch Always	
BRCLR	Branch if Bit Clear	
BRSET	Branch if Bit Set	
BScc	Branch to Subroutine Conditionally	
BSR	Branch to Subroutine Always	
BSCLR	Branch to Subroutine if Bit Clear	
BSSET	Branch to Subroutine if Bit Set	

Mnemonic	Description	Parallel Instruction
DEBUGcc	Enter into the Debug Mode Conditionally	
DEBUG	Enter into the Debug Mode Always	
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	
PLOCK	Lock Program Cache Sector	
PUNLOCK	Unlock Program Cache Sector	
PLOCKR	Lock PC-Related Program Cache Sector	
PUNLOCKR	Unlock PC-Related Program Cache Sector	
PFREE	Unlock all Program Cache Locked Sectors	
PFLUSH	Reset Program Cache State	
PFLUSHUN	Reset Program Cache State to all Unlocked Sectors	
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)	

A-4 INSTRUCTION GUIDE

The following information is included in each instruction description:

1. **Name and Mnemonic:** The mnemonic is highlighted in **bold** type for easy reference.
2. **Assembler Syntax and Operation:** For each instruction syntax, the corresponding operation is symbolically described. If there are several operations indicated on a single line in the operation field, those operations do not necessarily occur in the order shown but are generally assumed to occur in parallel. If a parallel data move is allowed, it will be indicated in parenthesis in both the assembler syntax and operation fields. If a letter in the mnemonic is optional, it will be shown in parenthesis in the assembler syntax field.
3. **Description:** A complete text description of the instruction is given together with any special cases and/or condition code anomalies of which the user should be aware when using that instruction.
4. **Condition Codes:** The status register is depicted with the condition code bits which can be affected by the instruction. Not all bits in the status register are used. Those which are reserved are indicated with a gray box covering its area.
5. **Instruction Format:** The instruction fields, the instruction opcode, and the instruction extension word are specified for each instruction syntax. When the extension word is optional, it is so indicated. The values which can be assumed by each of the variables in the various instruction fields are shown under the instruction field's heading.

A-4.1 NOTATION

Each instruction description contains symbols used to abbreviate certain operands and operations. Table A-9 lists the symbols used and their respective meanings. Depending on the context, registers refer to either the register itself or the contents of the register.

Table A-9. Instruction Description Notation

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)

Data ALU Registers Operands

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)
EMR	Extended Mode Register (8 Bits)
MR	Mode Register (8 Bits)
CCR	Condition Code Register (8 Bits)
SR	Status Register = EMR:MR:CCR (24 Bits)
SCS	System Stack Control Status Register (8 Bits)
EOM	Extended Chip Operating Mode Register (8 Bits)
COM	Chip Operating Mode Register (8 Bits)
OMR	Operating Mode Register = SCS:EOM:COM (24 Bits)
SZ	System Stack Size Register (24 Bits)
SC	System Stack Counter Register (5 Bits)
VBA	Vector Base Address (24 Bits, 8 of them are always zero)
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
xxxx	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
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

Unary Operands

PULL	Pull Specified Value from the System Stack (SS) Operator
READ	Read the Top of the System Stack (SS) Operator
PURGE	Delete the Top Value on the System Stack (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
→	“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 if the Chip is in Double Precision Multiply Mode
SB	Sixteen Bit Arithmetic Mode
RM	Rounding Mode

Mode Register Symbols

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 if Arithmetic Overflow has 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 Which Allows Parallel Moves
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)

A-5 CONDITION CODE COMPUTATION

The condition code register (CCR) portion of the status register (SR) consists of eight defined bits.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
CCR							

S — Scaling Bit

L — Limit Bit

E — Extension Bit

U — Unnormalized Bit

N — Negative Bit

Z — Zero Bit

V — Overflow Bit

C — Carry Bit

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 X, Y, or global data buses. The L bit is a **"sticky" overflow bit** which indicates that an overflow has occurred in the data ALU or that data limiting has occurred when moving the contents of the A and/or B accumulators. 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.

The full description of 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:

standard mark	The affect on the condition code
×	Unchanged by the instruction
✔	Changed by the instruction, according to the standard definition of the condition code
●	Changed by the instruction, according to a special definition of the condition code, depicted as part of the instruction full description

The standard definition of the condition code bits follows.

S (Scaling Bit) This bit is computed, according to the logical equations in the table below, when an instruction or a parallel move reads the contents of accumulator A or B to XDB or YDB. It is a “sticky” bit, cleared only by an instruction that specifically clears it, or hardware reset.

S0	S1	Scaling Mode	S bit equation
0	0	No scaling	S = (A46 XOR A45) OR (B46 XOR B45) OR S (previous)
0	1	Scale down	S = (A47 XOR A46) OR (B47 XOR B46) OR S (previous)
1	0	Scale up	S = (A45 XOR A44) OR (B45 XOR B44) OR S (previous)
1	1	Reserved	S undefined

The scaling bit (S) is used to detect data growth, which is required in Block Floating Point FFT operation. The scaling bit will be set if the absolute value in the accumulator, before scaling, was greater 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.”

L (Limit Bit) 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 XDB or YDB. In Arithmetic Saturation Mode, the limit bit is also set when an arithmetic saturation occurs in the Data ALU result. Not affected otherwise. This bit is “sticky” and must be cleared only by an instruction that specifically clears it, or hardware reset.

E (Extension Bit) 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. Set otherwise.

The signed integer portion is defined by the scaling mode as shown in the following table:

S1	S0	Scaling Mode	Integer Portion
0	0	No Scaling	Bits 55,54..............48,47
0	1	Scale Down	Bits 55,54..............49,48
1	0	Scale Up	Bits 55,54..............47,46

Note that the **signed integer portion** of an accumulator **IS NOT** necessarily the same as the **extension register portion** of that accumulator. The signed integer portion of an accumulator consists of the MS 8, 9, or 10 bits of that accumulator, depending on the scaling mode being used. The extension register portion of an accumulator (A2 or B2) is always the MS 8 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 (i.e., 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** MS bits of that accumulator were not all the same (i.e., neither 00.. 00 nor 11.. 11). This means that data limiting **will** occur if that 56-bit value is specified as a **source** operand in a move-type operation. This limiting operation will result in either a positive or negative, 24-bit or 48-bit saturation constant being stored in the specified destination. The **only** situation in which the signed integer portion of an accumulator and the extension register portion of an accumulator are the same is in the “Scale Down” scaling mode (i.e., S1=0 and S0=1).

U (Unnormalized Bit) Set if the two MS bits of the MSP portion of the Data ALU result are the same. Cleared otherwise. The MSP portion is defined by the scaling mode. The U bit is computed as follows:

S1	S0	Scaling Mode	U Bit Computation
0	0	No Scaling	$U = \overline{(\text{Bit 47 xor Bit 46})}$
0	1	Scale Down	$U = \overline{(\text{Bit 48 xor Bit 47})}$
1	0	Scale Up	$U = \overline{(\text{Bit 46 xor Bit 45})}$

The result of calculating the U bit in this fashion is that the definition of positive normalized number, p, is $0.5 \le p < 1.0$ and the definition of negative normalized number, n, is $-1.0 \le n < -0.5$.

N (Negative Bit) Set if the MS bit (bit 55 in arithmetic instructions or bit 47 in logical instructions) of the Data ALU result is set. Cleared otherwise.

Z (Zero Bit) Set if the Data ALU result equals zero. Cleared otherwise.

V (Overflow Bit) Set if an arithmetic overflow occurs in the 56-bit Data ALU result (40-bit result in Sixteen Bit mode). Cleared otherwise. This indicates that the result cannot be represented in the 56-bit (40-bit) accumulator; thus, the accumulator has overflowed.
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 Sixteen Bit Mode).

C (Carry Bit) Set if a carry is generated out of the MS bit of the Data ALU result of an addition or if a borrow is generated out of the MS bit of the Data ALU result of a subtraction. Cleared otherwise. The carry or borrow is generated out of bit 55 of the Data ALU result. The carry bit is also affected by bit manipulation, rotate, shift and compare instructions. The carry bit is not affected by the Arithmetic Saturation Mode.

A-6 INSTRUCTIONS DESCRIPTIONS

The following section describes each instruction in the DSP56300 Core instruction set in complete detail. Instructions which allow parallel moves include the notation “(parallel move)” in both the **Assembler Syntax** and the **Operation** fields. The MOVE instruction is equivalent to a NOP with parallel moves. Therefore, a detailed description of each parallel move is given with the MOVE instruction details.

Whenever an instruction uses an accumulator as both a destination operand for data ALU operation and as a source for a parallel move operation, the parallel move operation will use the value in the accumulator prior to execution of any data ALU operation.

A-6.1 Absolute Value (ABS)

ABS ABS

Absolute Value

Operation:	**Assembler Syntax:**
| D | → D (parallel move)	ABS D (parallel move)

Description: Take the absolute value of the destination operand D and store the result in the destination accumulator.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	✔	✔	✔	✔	✔	×
CCR							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23 16 15 8	7	0
ABS D	DATA BUS MOVE FIELD	0 0 1 0	d 1 1 0
	OPTIONAL EFFECTIVE ADDRESS EXTENSION		

Instruction Fields:

{D}	**d**	Destination accumulator [A,B] (see Table A-10 on page A-239)

A-6.2 Add Long with Carry (ADC)

ADC ADC

Add Long with Carry

Operation:

S+C+D ➞ D (parallel move)

Assembler Syntax:

ADC S,D (parallel move)

Description: Add the source operand S and the carry bit C of the condition code register to the destination operand D and store the result in the destination accumulator. Long words (48 bits) may be added to the (56-bit) destination accumulator.

Note: The carry bit is set correctly for multiple precision arithmetic using long-word operands if the extension register of the destination accumulator (A2 or B2) is the sign extension of bit 47 of the destination accumulator (A or B).

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	✔	✔	✔	✔	✔	✔
CCR							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

ADC S,D

23 16 15 8	7							0
DATA BUS MOVE FIELD	0	0	1	J	d	0	0	1
OPTIONAL EFFECTIVE ADDRESS EXTENSION								

Instruction Fields:

{S} **J** Source register [X,Y] (see Table A-11 on page A-239)
{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

A-6.3 Add (ADD)

ADD ADD

Add

Operation:	Assembler Syntax:
S+D➞D (parallel move)	ADD S,D (parallel move)
#xx+D➞D	ADD #xx,D
#xxxxxx+D➞D	ADD #xxxxxx,D

Description: Add the source operand S to the destination operand D and store the result in the destination accumulator. The source can be a register (word - 24 bits, long word - 48 bits or accumulator - 56 bits), short immediate (6 bits) or long immediate (24 bits).

When using 6-bit immediate data, the data is interpreted as an unsigned integer. That is, the 6 bits will be right aligned and the remaining bits will be zeroed to form a 24-bit source operand.

Note: The carry bit is set correctly using word or long-word source operands if the extension register of the destination accumulator (A2 or B2) is the sign extension of bit 47 of the destination accumulator (A or B). Thus, the carry bit is always set correctly using accumulator source operands, but can be set incorrectly if A1, B1, A10, B10 or immediate operand are used as source operands and A2 and B2 are not replicas of bit 47.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	✔	✔	✔	✔	✔	✔
CCR							

✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Instruction Formats and opcodes:

ADD S,D

23							16	15							8	7							0
DATA BUS MOVE FIELD																0	J	J	J	d	0	0	0
OPTIONAL EFFECTIVE ADDRESS EXTENSION																							

ADD #xx,D

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	1	i	i	i	i	i	i	1	0	0	0	d	0	0	0

ADD #xxxxxx,D

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	1	0	0	d	0	0	0
IMMEDIATE DATA EXTENSION																							

Instruction Fields:

{S}	**JJJ**	Source register [B/A,X,Y,X0,Y0,X1,Y1] (see Table A-14 on page A-240)
{D}	**d**	Destination accumulator [A/B] (see Table A-10 on page A-239)
{#xx}	**iiiiii**	6-bit Immediate Short Data
{#xxxxxx}		24-bit Immediate Long Data extension word

ADDL ADDL

Shift Left and Add Accumulators

Operation:

S+2∗D→D (parallel move)

Assembler Syntax:

ADDL S,D (parallel move)

Description: Add the source operand S to two times the destination operand D and store the result in the destination accumulator. The destination operand D is arithmetically shifted one bit to the left, and a zero is shifted into the LS bit of D prior to the addition operation. The carry bit is set correctly if the source operand does not overflow as a result of the left shift operation. The overflow bit may be set as a result of either the shifting or addition operation (or both). This instruction is useful for efficient divide and decimation in time (DIT) FFT algorithms.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✓	✓	✓	✓	✓	✓	●	✓
CCR							

- ● V Set if overflow has occurred in A or B result or the MS bit of the destination operand is changed as a result of the instruction’s left shift
- ✓ This bit is changed according to the standard definition
- × This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23 16 15 8	7							0
ADDL S,D	DATA BUS MOVE FIELD	0	0	0	1	d	0	1	0
	OPTIONAL EFFECTIVE ADDRESS EXTENSION								

Instruction Fields:

{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

{S} 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

A-6.5 Shift Right and Add Accumulators (ADDR)

ADDR ADDR

Shift Right and Add Accumulators

Operation:

S+D / 2→D (parallel move)

Assembler Syntax:

ADDR S,D (parallel move)

Description: Add the source operand S to one-half the destination operand D and store the result in the destination accumulator. The destination operand D is arithmetically shifted one bit to the right while the MS bit of D is held constant prior to the addition operation. In contrast to the ADDL instruction, the carry bit is always set correctly, and the overflow bit can only be set by the addition operation and not by an overflow due to the initial shifting operation. This instruction is useful for efficient divide and decimation in time (DIT) FFT algorithms.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	✔	✔	✔	✔	✔	✔
CCR							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23 16	15 8	7							0
ADDR S,D	DATA BUS MOVE FIELD		0	0	0	0	d	0	1	0
	OPTIONAL EFFECTIVE ADDRESS EXTENSION									

Instruction Fields:

{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

{S} 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

A-6.6 Logical AND (AND)

AND AND

Logical AND

Operation:	**Assembler Syntax:**
S • D[47:24]→D[47:24] (parallel move)	AND S,D (parallel move)
#xx • D[47:24]→D[47:24]	AND #xx,D
#xxxxxx • D[47:24]→D[47:24]	AND #xxxxxx,D

where •denotes the logical AND operator

Description: Logically AND the source operand S with bits 47-24 of the destination operand D and store the result in bits 47-24 of the destination accumulator. The source can be a 24-bit register, 6-bit short immediate or 24-bit long immediate. This instruction is a 24-bit operation. The remaining bits of the destination operand D are not affected.

When using 6-bit immediate data, the data is interpreted as an unsigned integer. That is, the 6 bits will be right aligned and the remaining bits will be zeroed to form a 24-bit source operand.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	×	×	×	●	●	●	×
CCR							

● N Set if bit 47 of the result is set
● Z Set if bits 47-24 of the result are zero
● V Always cleared
✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Instruction Formats and opcodes:

AND S,D

<table>
<tr><td>23</td><td></td><td></td><td></td><td></td><td></td><td></td><td>16</td><td>15</td><td></td><td></td><td></td><td></td><td></td><td></td><td>8</td><td>7</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr>
<tr><td colspan="16">DATA BUS MOVE FIELD</td><td>0</td><td>1</td><td>J</td><td>J</td><td>d</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td colspan="24">OPTIONAL EFFECTIVE ADDRESS EXTENSION</td></tr>
</table>

AND #xx,D

<table>
<tr><td>23</td><td></td><td></td><td></td><td></td><td></td><td></td><td>16</td><td>15</td><td></td><td></td><td></td><td></td><td></td><td></td><td>8</td><td>7</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>i</td><td>i</td><td>i</td><td>i</td><td>i</td><td>i</td><td>1</td><td>0</td><td>0</td><td>0</td><td>d</td><td>1</td><td>1</td><td>0</td></tr>
</table>

AND #xxxxxx,D

<table>
<tr><td>23</td><td></td><td></td><td></td><td></td><td></td><td></td><td>16</td><td>15</td><td></td><td></td><td></td><td></td><td></td><td></td><td>8</td><td>7</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>d</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td colspan="24">IMMEDIATE DATA EXTENSION</td></tr>
</table>

Instruction Fields:

{S}	**JJ**	Source input register [X0,X1,Y0,Y1] (see Table A-12 on page A-239)
{D}	**d**	Destination accumulator [A/B] (see Table A-10 on page A-239)
{#xx}	**iiiiii**	6-bit Immediate Short Data
{#xxxxxx}		24-bit Immediate Long Data extension word

A-6.7 AND Immediate with Control Register (ANDI)

ANDI ANDI

AND Immediate with Control Register

Operation:
#xx • D➞D
where •denotes the logical AND operator

Assembler Syntax:
AND(I) #xx,D

Description: Logically AND the 8-bit immediate operand (#xx) with the contents of the destination control register D and store the result in the destination control register. The condition codes are affected only when the condition code register (CCR) is specified as the destination operand.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
●	●	●	●	●	●	●	●
CCR							

For CCR Operand:

- S Cleared if bit 7 of the immediate operand is cleared
- L Cleared if bit 6 of the immediate operand is cleared
- E Cleared if bit 5 of the immediate operand is cleared
- U Cleared if bit 4 of the immediate operand is cleared
- N Cleared if bit 3 of the immediate operand is cleared
- Z Cleared if bit 2 of the immediate operand is cleared
- V Cleared if bit 1 of the immediate operand is cleared
- C Cleared if bit 0 of the immediate operand is cleared

For MR and OMR Operands: The condition codes are not affected using these operands.

Instruction Formats and opcodes:

	23							16	15							8	7							0
AND(I) #xx,D	0	0	0	0	0	0	0	0	i	i	i	i	i	i	i	i	1	0	1	1	1	0	E	E

Instruction fields:

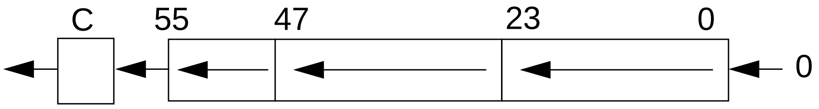
{D}	**EE**	Program Controller register [MR,CCR,COM,EOM] (see Table A-13 on page A-239)
{#xx}	**iiiiiiii**	Immediate Short Data

A-6.8 Arithmetic Shift Accumulator Left (ASL)

ASL ASL

Arithmetic Shift Accumulator Left

Operation:



Assembler Syntax:

ASL D (parallel move)

ASL #ii,S2,D

ASL S1,S2,D

Description:

Single bit shift:

Arithmetically shift the destination accumulator D one bit to the left and store the result in the destination accumulator. The MS bit of D prior to instruction execution is shifted into the carry bit C and a zero is shifted into the LS bit of the destination accumulator D.

Multi-bit shift:

The contents of the source accumulator S2 are shifted left #ii bits. Bits shifted out of position 55 are lost, but for the last bit which is latched in the carry bit C. Zeros are supplied to the vacated positions on the right. The result is placed into destination accumulator D. The number of bits to shift is determined by the 6-bit immediate field in the instruction, or by the 6-bit unsigned integer located in the 6 LSBs of the control register S1. If a zero shift count is specified, the carry bit is cleared. The difference between ASL and LSL is that ASL operates on the entire 56 bits of the accumulator and therefore sets the V bit if the number overflowed.

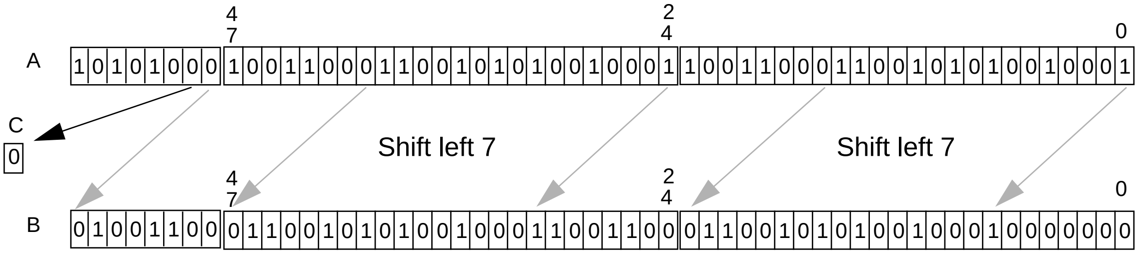
This is a 56 bit operation.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	✔	✔	✔	✔	●	●
CCR							

- ● V Set if bit 55 is changed any time during the shift operation. Cleared otherwise.
- ● C Set if the last bit shifted out of the operand is set. Cleared otherwise. Cleared for a shift count of zero.
- × This bit is unchanged by the instruction
- ✔ This bit is changed according to the standard definition

Example: ASL #7,A, B



Instruction Formats and opcodes:

ASL D

23 … 8	7							0
DATA BUS MOVE FIELD	0	0	1	1	d	0	1	0
OPTIONAL EFFECTIVE ADDRESS EXTENSION								

ASL #ii,S2,D

23							16	15							8	7							0
0	0	0	0	1	1	0	0	0	0	0	1	1	1	0	1	S	i	i	i	i	i	i	D

ASL S1,S2,D

23							16	15							8	7							0
0	0	0	0	1	1	0	0	0	0	0	1	1	1	1	0	0	1	0	S	s	s	s	D

Instruction Fields:

{S2}	**S**	Source accumulator [A,B] (see Table A-10 on page A-239)
{D}	**D**	Destination accumulator [A,B] (see Table A-10 on page A-239)
{S1}	**sss**	Control register [X0,X1,Y0,Y1,A1,B1] (see Table A-15 on page A-240)
{#ii}	**iiiiii**	6 bit unsigned integer [0-55] denoting the shift amount

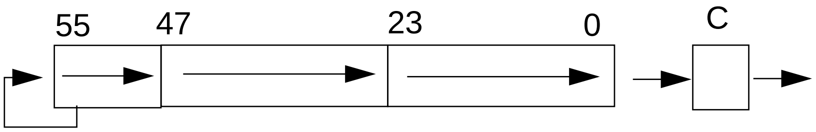
In the **control register** S1: bits 5-0 (LSB) are used as #ii field, and the rest of the register is ignored.

A-6.9 Arithmetic Shift Accumulator Right (ASR)

ASR ASR

Arithmetic Shift Accumulator Right

Operation:



Assembler Syntax:

ASR D (parallel move)

ASR #ii,S2,D

ASR S1,S2,D

Description:

Single bit shift:

Arithmetically shift the destination operand D one bit to the right and store the result in the destination accumulator. The LS bit of D prior to instruction execution is shifted into the carry bit C, and the MS bit of D is held constant.

Multi-bit shift:

The contents of the source accumulator S2 are shifted right #ii bits. Bits shifted out of position 0 are lost, but for the last bit which is latched in the carry bit. Copies of the MSB are supplied to the vacated positions on the left. The result is placed into destination accumulator D.The number of bits to shift is determined by the 6-bit immediate field in the instruction, or by the 6-bit unsigned integer located in the 6 LSBs of the control register S1. If a zero shift count is specified, the carry bit is cleared.

This is a 56- or 40-bit operation, depending on SA bit value in status register.

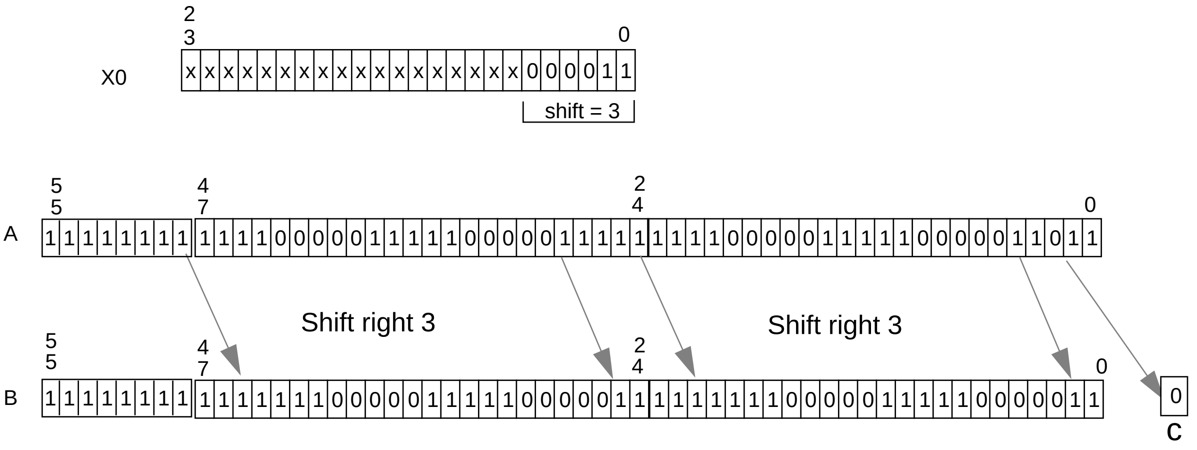
Note: if the number of shifts indicated by the 6 LSBs of the control register or by the immediate field, exceeds the value of 56 (40 in sixteen bit arithmetic mode), then the result would be undefined.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	✔	✔	✔	✔	●	●
CCR							

- ● V Always cleared
- ● C Set if the last bit shifted out of the operand is set. Cleared otherwise. Cleared for a shift count of zero
- × This bit is unchanged by the instruction
- ✔ This bit is changed according to the standard definition

Example: ASR X0,A,B



Instruction Formats and opcodes:

ASR D

23 … 8	7							0
DATA BUS MOVE FIELD	0	0	1	0	d	0	1	0
OPTIONAL EFFECTIVE ADDRESS EXTENSION								

ASR #ii,S2,D

23							16	15							8	7							0
0	0	0	0	1	1	0	0	0	0	0	1	1	1	0	0	S	i	i	i	i	i	i	D

ASR S1,S2,D

23							16	15							8	7							0
0	0	0	0	1	1	0	0	0	0	0	1	1	1	1	0	0	1	1	S	s	s	s	D

Instruction Fields:

{S2}	**S**	Source accumulator [A,B] (see Table A-10 on page A-239)
{D}	**D**	Destination accumulator [A,B] (see Table A-10 on page A-239)
{S1}	**sss**	Control register [X0,X1,Y0,Y1,A1,B1] (see Table A-15 on page A-240)
{#ii}	**iiiiii**	6 bit unsigned integer [0-55] denoting the shift amount

In the **control register** S1: bits 5-0 (LSB) are used as #ii field, and the rest of the register is ignored.

A-6.10 Branch Conditionally (Bcc)

Bcc Bcc

Branch Conditionally

Operation:						Assembler Syntax:	
If	cc,	then	PC+xxxx	→	PC	Bcc	xxxx
		else	PC+1	→	PC		
If	cc,	then	PC+xxx	→	PC	Bcc	xxx
		else	PC+1	→	PC		
If	cc,	then	PC+Rn	→	PC	Bcc	Rn
		else	PC+1	→	PC		

Description: If the specified condition is true, program execution continues at location PC+displacement. If the specified condition is false, the PC is incremented and program execution continues sequentially. The displacement is a 2's complement 24-bit integer that represents the relative distance from the current PC to the destination PC. Short Displacement, Long Displacement and Address Register PC Relative addressing modes may be used. The Short Displacement 9-bit data is sign extended to form the PC relative displacement.

The conditions that the term "**cc**" can specify are listed on Table A-42 on page A-250.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

		23							16	15							8	7							0
Bcc	xxxx	0	0	0	0	1	1	0	1	0	0	0	1	0	0	0	0	0	1	0	0	C	C	C	C
		PC RELATIVE DISPLACEMENT																							

		23							16	15							8	7							0
Bcc	xxx	0	0	0	0	0	1	0	1	C	C	C	C	0	1	a	a	a	a	0	a	a	a	a	a

		23							16	15							8	7							0
Bcc	Rn	0	0	0	0	1	1	0	1	0	0	0	1	1	R	R	R	0	1	0	0	C	C	C	C

Instruction Fields:

{cc}	**CCCC**	Condition code (see Table A-43 on page A-251)
{xxxx}		24-bit PC Relative Long Displacement
{xxx}	**aaaaaaaaa**	Signed PC Relative Short Displacement
{Rn}	**RRR**	Address register [R0-R7]

A-6.11 Bit Test and Change (BCHG)

BCHG BCHG

Bit Test and Change

Operation:		Assembler Syntax:
$D[n] \rightarrow C$	$\overline{D[n]} \rightarrow D[n]$	BCHG #n,[XorY]:ea
$D[n] \rightarrow C$	$\overline{D[n]} \rightarrow D[n]$	BCHG #n,[XorY]:aa
$D[n] \rightarrow C$	$\overline{D[n]} \rightarrow D[n]$	BCHG #n,[XorY]:pp
$D[n] \rightarrow C$	$\overline{D[n]} \rightarrow D[n]$	BCHG #n,[XorY]:qq
$D[n] \rightarrow C$	$\overline{D[n]} \rightarrow D[n]$	BCHG #n,D

Description: Test the n^{th} bit of the destination operand D, complement it, and store the result in the destination location. The state of the n^{th} bit is stored in the carry bit C of the condition code register. The bit to be tested is selected by an immediate bit number from 0–23. This instruction performs a read-modify-write operation on the destination location using two destination accesses before releasing the bus. This instruction provides a test-and-change capability which is useful for synchronizing multiple processors using a shared memory. This instruction can use all memory alterable addressing modes.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
●	●	●	●	●	●	●	●
CCR							

For destination operand SR:

- C Complemented if bit 0 is specified. Not affected otherwise.
- V Complemented if bit 1 is specified. Not affected otherwise.
- Z Complemented if bit 2 is specified. Not affected otherwise.
- N Complemented if bit 3 is specified. Not affected otherwise.
- U Complemented if bit 4 is specified. Not affected otherwise.
- E Complemented if bit 5 is specified. Not affected otherwise.
- L Complemented if bit 6 is specified. Not affected otherwise.
- S Complemented if bit 7 is specified. Not affected otherwise.

For other destination operands:

- C Set if bit tested is set. Cleared otherwise.
- V Not affected.
- Z Not affected.
- N Not affected.
- U Not affected.
- E Not affected.
- L According to the standard definition.
- S According to the standard definition.

MR Status Bits:

For destination operand SR:

- I0 Changed if bit 8 is specified. Not affected otherwise
- I1 Changed if bit 9 is specified. Not affected otherwise
- So Changed if bit 10 is specified. Not affected otherwise
- S1 Changed if bit 11 is specified. Not affected otherwise
- RM Changed if bit 12 is specified. Not affected otherwise
- SB Changed if bit 13 is specified. Not affected otherwise
- DM Changed if bit 14 is specified. Not affected otherwise
- LF Changed if bit 15 is specified. Not affected otherwise

For other destination operands: MR status bits are not affected.

Instruction Formats and opcodes:

BCHG #n,[X or Y]:ea

23							16	15							8	7							0
0	0	0	0	1	0	1	1	0	1	M	M	M	R	R	R	O	S	0	b	b	b	b	b
OPTIONAL EFFECTIVE ADDRESS EXTENSION																							

BCHG #n,[X or Y]:aa

23							16	15							8	7							0
0	0	0	0	1	0	1	1	0	0	a	a	a	a	a	a	0	S	0	b	b	b	b	b

BCHG #n,[X or Y]:pp

23							16	15							8	7							0
0	0	0	0	1	0	1	1	1	0	p	p	p	p	p	p	0	S	0	b	b	b	b	b

BCHG #n,[X or Y]:qq

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	1	q	q	q	q	q	q	0	S	0	b	b	b	b	b

BCHG #n,D

23							16	15							8	7							0
0	0	0	0	1	0	1	1	1	1	D	D	D	D	D	D	0	1	0	b	b	b	b	b

Instruction Fields:

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address (see Table A-16 on page A-241)
{X /Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{aa}	**aaaaaa**	Absolute Address [0-63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0-$FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80-$FFFFBF]
{D}	**DDDDDD**	Destination register [all on-chip registers] (see Table A-22 on page A-243)

A-6.12 Bit Test and Clear (BCLR)

BCLR BCLR

Bit Test and Clear

Operation:		Assembler Syntax:	
D[n] → C	0 → D[n]	BCLR	#n,[XorY]:ea
D[n] → C	0 → D[n]	BCLR	#n,[XorY]:aa
D[n] → C	0 → D[n]	BCLR	#n,[XorY]:pp
D[n] → C	0 → D[n]	BCLR	#n,[XorY]:qq
D[n] → C	0 → D[n]	BCLR	#n,D

Description: Test the nth bit of the destination operand D, clear it and store the result in the destination location. The state of the nth bit is stored in the carry bit C of the condition code register. The bit to be tested is selected by an immediate bit number from 0–23. This instruction performs a read-modify-write operation on the destination location using two destination accesses before releasing the bus. This instruction provides a test-and-clear capability which is useful for synchronizing multiple processors using a shared memory. This instruction can use all memory alterable addressing modes.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
●	●	●	●	●	●	●	●
CCR							

CCR Condition Codes:

For destination operand SR:

- C Cleared if bit 0 is specified. Not affected otherwise.
- V Cleared if bit 1 is specified. Not affected otherwise.
- Z Cleared if bit 2 is specified. Not affected otherwise.
- N Cleared if bit 3 is specified. Not affected otherwise.
- U Cleared if bit 4 is specified. Not affected otherwise.

- E Cleared if bit 5 is specified. Not affected otherwise.
- L Cleared if bit 6 is specified. Not affected otherwise.
- S Cleared if bit 7 is specified. Not affected otherwise.

For other destination operands:

- C Set if bit tested is set. Cleared otherwise.
- V Not affected.
- Z Not affected.
- N Not affected.
- U Not affected.
- E Not affected.
- L According to the standard definition.
- S According to the standard definition.

MR Status Bits:

For destination operand SR:

- I0 Changed if bit 8 is specified. Not affected otherwise
- I1 Changed if bit 9 is specified. Not affected otherwise
- So Changed if bit 10 is specified. Not affected otherwise
- S1 Changed if bit 11 is specified. Not affected otherwise
- RM Changed if bit 12 is specified. Not affected otherwise
- SB Changed if bit 13 is specified. Not affected otherwise
- DM Changed if bit 14 is specified. Not affected otherwise
- LF Changed if bit 15 is specified. Not affected otherwise

Instruction Formats and opcodes:

	23							16	15							8	7							0
BCLR #n,[X or Y]:ea	0	0	0	0	1	0	1	0	0	1	M	M	M	R	R	R	0	S	0	b	b	b	b	b
	OPTIONAL EFFECTIVE ADDRESS EXTENSION																							

	23							16	15							8	7							0
BCLR #n,[X or Y]:aa	0	0	0	0	1	0	1	0	0	0	a	a	a	a	a	a	0	S	0	b	b	b	b	b

	23							16	15							8	7							0
BCLR #n,[X or Y]:pp	0	0	0	0	1	0	1	0	1	0	p	p	p	p	p	p	0	S	0	b	b	b	b	b

	23							16	15							8	7							0
BCLR #n,[X or Y]:qq	0	0	0	0	0	0	0	1	0	0	q	q	q	q	q	q	0	S	0	b	b	b	b	b

	23							16	15							8	7							0
BCLR #n,D	0	0	0	0	1	0	1	0	1	1	D	D	D	D	D	D	0	1	0	b	b	b	b	b

Instruction Fields:

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address (see Table A-16 on page A-241)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{aa}	**aaaaaa**	Absolute Address [0-63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0-$FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80-$FFFFBF]
{D}	**DDDDDD**	Destination register [all on-chip registers] (see Table A-22 on page A-243)

A-6.13 Branch Always (BRA)

BRA BRA

Branch Always

Operation:	**Assembler Syntax:**
PC+xxxx → Pc	BRA xxxx
PC+xxx → Pc	BRA xxx
PC+Rn → Pc	BRA Rn

Description:

Program execution continues at location PC+displacement. The displacement is a 2’s complement 24-bit integer that represents the relative distance from the current PC to the destination PC. Short Displacement, Long Displacement and Address Register PC Relative addressing modes may be used. The Short Displacement 9-bit data is sign extended to form the PC relative displacement.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

BRA xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	1	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0
PC RELATIVE DISPLACEMENT																							

BRA xxx

23							16	15							8	7							0
0	0	0	0	0	1	0	1	0	0	0	0	1	1	a	a	a	a	0	a	a	a	a	a

BRA Rn

23							16	15							8	7							0
0	0	0	0	1	1	0	1	0	0	0	1	1	R	R	R	1	1	0	0	0	0	0	0

Instruction Fields:

{xxxx}		24-bit PC Relative Long Displacement
{xxx}	**aaaaaaaaa**	Signed PC Relative Short Displacement
{Rn}	**RRR**	Address register [R0-R7]

A-6.14 Branch if Bit Clear (BRCLR)

BRCLR BRCLR

Branch if bit Clear

Operation:	**Assembler Syntax:**
If S{n}=0 then PC+xxxx → PC else PC+ 1 → PC	BRCLR #n,[X or Y]:ea,xxxx
If S{n}=0 then PC+xxxx → PC else PC+ 1 → PC	BRCLR #n,[X or Y],aa,xxxx
If S{n}=0 then PC+xxxx → PC else PC+ 1 → PC	BRCLR #n,[X or Y]:pp,xxxx
If S{n}=0 then PC+xxxx → PC else PC+ 1 → PC	BRCLR #n,[X or Y]:qq,xxxx
If S{n}=0 then PC+xxxx → PC else PC+ 1 → PC	BRCLR #n,S,xxxx

Description: The nth bit in the source operand is tested. If the tested bit is cleared, program execution continues at location PC+displacement. If the tested bit is set, the PC is incremented and program execution continues sequentially. However, the address register specified in the effective address field is always updated independently of the condition. The displacement is a 2’s complement 24-bit integer that represents the relative distance from the current PC to the destination PC. The 24-bit displacement is contained in the extension word of the instruction. All memory alterable addressing modes may be used to reference the source operand. Absolute Short, I/O Short and Register Direct addressing modes may also be used. Note that if the specified source operand S is the SSH, the stack pointer register will be decremented by one. The bit to be tested is selected by an immediate bit number 0-23.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	×	×	×	×
CCR							

✔	This bit is changed according to the standard definition
×	This bit is unchanged by the instruction

Instruction Formats and opcodes:

BRCLR #n,[X or Y]:ea,xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	0	1	0	M	M	M	R	R	R	0	S	0	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

BRCLR #n,[X or Y]:aa,xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	0	1	0	a	a	a	a	a	a	1	S	0	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

BRCLR #n,[X or Y]:pp,xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	0	1	1	p	p	p	p	p	p	0	S	0	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

BRCLR #n,[X or Y]:qq,xxxx

23							16	15							8	7							0
0	0	0	0	0	1	0	0	1	0	q	q	q	q	q	q	0	S	0	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

BRCLR #n,S,xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	0	1	1	D	D	D	D	D	D	1	0	0	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

Instruction Fields:

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address (see Table A-19 on page A-242)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{xxxx}		24-bit PC relative displacement
{aa}	**aaaaaa**	Absolute Address [0-63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0-$FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80-$FFFFBF]
{S}	**DDDDDD**	Source register [all on-chip registers] (see Table A-22 on page A-243)

A-6.15 Exit Current Do Loop Conditionally (BRKcc)

BRKcc BRKcc

Exit Current Do Loop Conditionally

Operation:		**Assembler Syntax:**
If cc	LA+1→PC; SSL(LF,FV)→ SR; SP-1→ SP	BRKcc
	SSH → LA; SSL→ LC; SP-1→ SP	
else	PC+1→ PC	

Description: Exit conditionally the current hardware DO loop before the current loop counter (LC) equals one. It also terminates the DO FOREVER (or DOR FOREVER) loop. If the value of the current DO loop counter (LC) is needed, it must be read before the execution of the BRKcc instruction. Initially, the PC is updated from the LA, the loop flag (LF) and the ForeVer flag (FV) are restored and the remaining portion of the status register (SR) is purged from the system stack. The loop address (LA) and the loop counter (LC) registers are then restored from the system stack.

The conditions that the term “**cc**” can specify are listed on Table A-43 on page A-251.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23							16	15							8	7							0
BRKcc	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	C	C	C	C

Instruction Fields:

{cc} **CCCC** Condition code (see Table A-43 on page A-251)

A-6.16 Branch if Bit Set (BRSET)

BRSET BRSET

Branch if bit Set

Operation:						Assembler Syntax:	
If	S{n}=1	then	PC+xxxx	→	PC	BRSET	#n,[X or Y]:ea,xxxx
		else	PC+ 1	→	PC		
If	S{n}=1	then	PC+xxxx	→	PC	BRSET	#n,[X or Y],aa,xxxx
		else	PC+ 1	→	PC		
If	S{n}=1	then	PC+xxxx	→	PC	BRSET	#n,[X or Y]:pp,xxxx
		else	PC+ 1	→	PC		
If	S{n}=1	then	PC+xxxx	→	PC	BRSET	#n,[X or Y]:qq,xxxx
		else	PC+ 1	→	PC		
If	S{n}=1	then	PC+xxxx	→	PC	BRSET	#n,S,xxxx
		else	PC+ 1	→	PC		

Description: The nth bit in the source operand is tested. If the tested bit is set, program execution continues at location PC+displacement. If the tested bit is cleared, the PC is incremented and program execution continues sequentially. However, the address register specified in the effective address field is always updated independently of the condition. The displacement is a 2's complement 24-bit integer that represents the relative distance from the current PC to the destination PC. The 24-bit displacement is contained in the extension word of the instruction. All memory alterable addressing modes may be used to reference the source operand. Absolute Short, I/O Short and Register Direct addressing modes may also be used. Note that if the specified source operand S is the SSH, the stack pointer register will be decremented by one. The bit to be tested is selected by an immediate bit number 0-23.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✓	✓	×	×	×	×	×	×
CCR							

✓ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

BRSET #n,[X or Y]:ea,xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	0	1	0	M	M	M	R	R	R	0	S	1	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

BRSET #n,[X or Y]:aa,xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	0	1	0	a	a	a	a	a	a	1	S	1	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

BRSET #n,[X or Y]:pp,xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	0	1	1	p	p	p	p	p	p	0	S	1	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

BRSET #n,[X or Y]:qq,xxxx

23							16	15							8	7							0
0	0	0	0	0	1	0	0	1	0	q	q	q	q	q	q	0	S	1	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

BRSET #n,S,xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	0	1	1	D	D	D	D	D	D	1	0	1	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

Instruction Fields:

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address (see Table A-19 on page A-242)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{xxxx}		24-bit PC relative displacement
{aa}	**aaaaaa**	Absolute Address [0-63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0-$FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80-$FFFFBF]
{S}	**DDDDDD**	Source register [all on-chip registers] (see Table A-22 on page A-243)

A-6.17 Branch to Subroutine Conditionally (BScc)

BScc BScc

Branch to Subroutine Conditionally

Operation:	**Assembler Syntax:**
If cc, then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BScc xxxx
If cc, then PC →SSH;SR →SSL;PC+xxx →PC else PC+1→PC	BScc xxx
If cc, then PC →SSH;SR →SSL;PC+Rn →PC else PC+1→PC	BScc Rn

Description: If the specified condition is true, the address of the instruction immediately following the BScc instruction and the status register are pushed onto the stack. Program execution then continues at location PC+displacement. If the specified condition is false, the PC is incremented and program execution continues sequentially. The displacement is a 2’s complement 24-bit integer that represents the relative distance from the current PC to the destination PC. Short Displacement, Long Displacement and Address Register PC Relative addressing modes may be used. The Short Displacement 9-bit data is sign extended to form the PC relative displacement.

The conditions that the term “**cc**” can specify are listed on Table A-42 on page A-250.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

BScc xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	C	C	C	C
PC RELATIVE DISPLACEMENT																							

BScc xxx

23							16	15							8	7							0
0	0	0	0	0	1	0	1	C	C	C	C	0	0	a	a	a	a	0	a	a	a	a	a

BScc Rn

23							16	15							8	7							0
0	0	0	0	1	1	0	1	0	0	0	1	1	R	R	R	0	0	0	0	C	C	C	C

Instruction Fields:

{cc}	**CCCC**	Condition code (see Table A-43 on page A-251)
{xxxx}		24-bit PC Relative Long Displacement
{xxx}	**aaaaaaaaa**	Signed PC Relative Short Displacement
{Rn}	**RRR**	Address register [R0-R7]

A-6.18 Branch to Subroutine if Bit Clear (BSCLR)

BSCLR BSCLR

Branch to Subroutine if Bit Clear

Operation:	Assembler Syntax:
If S{n}=0 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSCLR #n,[X or Y]:ea,xxxx
If S{n}=0 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSCLR #n,[X or Y],aa,xxxx
If S{n}=0 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSCLR #n,[X or Y]:pp,xxxx
If S{n}=0 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSCLR #n,[X or Y]:qq,xxxx
If S{n}=0 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSCLR #n,S,xxxx

Description: The nth bit in the source operand is tested. If the tested bit is cleared, the address of the instruction immediately following the BSCLR instruction and the status register are pushed onto the stack. Program execution then continues at location PC+displacement. If the tested bit is set, the PC is incremented and program execution continues sequentially. However, the address register specified in the effective address field is always updated independently of the condition. The displacement is a 2's complement 24-bit integer that represents the relative distance from the current PC to the destination PC. The 24-bit displacement is contained in the extension word of the instruction. All memory alterable addressing modes may be used to reference the source operand. Absolute Short, I/O Short and Register Direct addressing modes may also be used. Note that if the specified source operand S is the SSH, the stack pointer register will be decremented by one; if the condition is true, the push operation will write over the stack level where the SSH value was taken. The bit to be tested is selected by an immediate bit number 0-23.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✓	✓	×	×	×	×	×	×
CCR							

✓ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Instruction Formats and opcodes:

BSCLR #n,[X or Y]:ea,xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	1	1	0	M	M	M	R	R	R	0	S	0	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

BSCLR #n,[X or Y]:aa,xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	1	1	0	a	a	a	a	a	a	1	S	0	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

BSCLR #n,[X or Y]:qq,xxxx

23							16	15							8	7							0
0	0	0	0	0	1	0	0	1	0	q	q	q	q	q	q	1	S	0	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

BSCLR #n,[X or Y]:pp,xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	1	1	1	p	p	p	p	p	p	0	S	0	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

BSCLR #n,S,xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	1	1	1	D	D	D	D	D	D	1	0	0	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

Instruction Fields:

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address (see Table A-19 on page A-242)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{xxxx}		24-bit Relative Long Displacement
{aa}	**aaaaaa**	Absolute Address [0-63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0-$FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80-$FFFFBF]
{S}	**DDDDDD**	Source register [all on-chip registers] (see Table A-22 on page A-243)

A-6.19 Bit Test and Set (BSET)

BSET BSET

Bit Test and Set

Operation:		Assembler Syntax:	
D[n] → C	1→ D[n]	BSET	#n,[XorY]:ea
D[n] → C	1→ D[n]	BSET	#n,[XorY]:aa
D[n] → C	1→ D[n]	BSET	#n,[XorY]:pp
D[n] → C	1→ D[n]	BSET	#n,[XorY]:qq
D[n] → C	1→ D[n]	BSET	#n,D

Description: Test the nth bit of the destination operand D, set it, and store the result in the destination location. The state of the nth bit is stored in the carry bit C of the condition code register. The bit to be tested is selected by an immediate bit number from 0–23. This instruction performs a read-modify-write operation on the destination location using two destination accesses before releasing the bus. This instruction provides a test-and-set capability which is useful for synchronizing multiple processors using a shared memory. This instruction can use all memory alterable addressing modes.

When this instruction performs a bit manipulation/test on either the A or B 56-bit accumulator, it optionally shifts the accumulator value according to scaling mode bits S0 and S1 in the system status register (SR). In the data out of the shifter indicates that the accumulator extension register is in use, the instruction will act on the limited value (limited on the maximum positive or negative saturation constant). In addition the “L” flag in the SR will be set accordingly.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
●	●	●	●	●	●	●	●
CCR							

CCR Condition Codes:

For destination operand SR:

- C Set if bit 0 is specified. Not affected otherwise.
- V Set if bit 1 is specified. Not affected otherwise.
- Z Set if bit 2 is specified. Not affected otherwise.
- N Set if bit 3 is specified. Not affected otherwise.
- U Set if bit 4 is specified. Not affected otherwise.
- E Set if bit 5 is specified. Not affected otherwise.
- L Set if bit 6 is specified. Not affected otherwise.
- S Set if bit 7 is specified. Not affected otherwise.

For other destination operands:

- C Set if bit tested is set. Cleared otherwise.
- V Not affected.
- Z Not affected.
- N Not affected.
- U Not affected.
- E Not affected.
- L According to the standard definition.
- S According to the standard definition.

MR Status Bits:

For destination operand SR:

- I0 Changed if bit 8 is specified. Not affected otherwise
- I1 Changed if bit 9 is specified. Not affected otherwise
- So Changed if bit 10 is specified. Not affected otherwise
- S1 Changed if bit 11 is specified. Not affected otherwise
- RM Changed if bit 12 is specified. Not affected otherwise
- SB Changed if bit 13 is specified. Not affected otherwise
- DM Changed if bit 14 is specified. Not affected otherwise
- LF Changed if bit 15 is specified. Not affected otherwise

For other destination operands: MR status bits are not affected.

Instruction Formats and opcodes:

BSET #n,[X or Y]:ea

23							16	15							8	7							0
0	0	0	0	1	0	1	0	0	1	M	M	M	R	R	R	0	S	1	b	b	b	b	b
OPTIONAL EFFECTIVE ADDRESS EXTENSION																							

BSET #n,[X or Y]:aa

23							16	15							8	7							0
0	0	0	0	1	0	1	0	0	0	a	a	a	a	a	a	0	S	1	b	b	b	b	b

BSET #n,[X or Y]:pp

23							16	15							8	7							0
0	0	0	0	1	0	1	0	1	0	p	p	p	p	p	p	0	S	1	b	b	b	b	b

BSET #n,[X or Y]:qq

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	0	q	q	q	q	q	q	0	S	1	b	b	b	b	b

BSET #n,D

23							16	15							8	7							0
0	0	0	0	1	0	1	0	1	1	D	D	D	D	D	D	0	1	1	b	b	b	b	b

Instruction Fields:

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address (see Table A-16 on page A-241)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{aa}	**aaaaaa**	Absolute Address [0-63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0-$FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80-$FFFFBF]
{D}	**DDDDDD**	Destination register [all on-chip registers] (see Table A-22 on page A-243)

A-6.20 Branch to Subroutine (BSR)

BSR BSR

Branch to Subroutine

Operation:	**Assembler Syntax:**
PC →SSH;SR →SSL;PC+xxxx→PC	BSR xxxx
PC →SSH;SR →SSL;PC+xxx→PC	BSR xxx
PC →SSH;SR →SSL;PC+Rn→PC	BSR Rn

Description: The address of the instruction immediately following the BSR instruction and the status register are pushed onto the stack. Program execution then continues at location PC+displacement. The displacement is a 2’s complement 24-bit integer that represents the relative distance from the current PC to the destination PC. Short Displacement, Long Displacement and Address Register PC Relative addressing modes may be used. The Short Displacement 9-bit data is sign extended to form the PC relative displacement.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

BSR xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0
PC RELATIVE DISPLACEMENT																							

BSR xxx

23							16	15							8	7							0
0	0	0	0	0	1	0	1	0	0	0	0	1	0	a	a	a	a	0	a	a	a	a	a

BSR Rn

23							16	15							8	7							0
0	0	0	0	1	1	0	1	0	0	0	1	1	R	R	R	1	0	0	0	0	0	0	0

Instruction Fields:

{xxxx}		24-bit PC Relative Long Displacement
{xxx}	**aaaaaaaaa**	Signed PC Relative Short Displacement
{Rn}	**RRR**	Address register [R0-R7]

A-6.21 Branch to Subroutine if Bit Set (BSSET)

BSSET BSSET

Branch to Subroutine if Bit Set

Operation:	Assembler Syntax:
If S{n}=1 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSSET #n,[X or Y]:ea,xxxx
If S{n}=1 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSSET #n,[X or Y],aa,xxxx
If S{n}=1 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSSET #n,[X or Y]:pp,xxxx
If S{n}=1 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSSET #n,[X or Y]:qq,xxxx
If S{n}=1 then PC →SSH;SR →SSL;PC+xxxx →PC else PC+1→PC	BSSET #n,S,xxxx

Description: The nth bit in the source operand is tested. If the tested bit is set, the address of the instruction immediately following the BSSET instruction and the status register are pushed onto the stack. Program execution then continues at location PC+displacement. If the tested bit is cleared, the PC is incremented and program execution continues sequentially. However, the address register specified in the effective address field is always updated independently of the condition. The displacement is a 2’s complement 24-bit integer that represents the relative distance from the current PC to the destination PC. The 24-bit displacement is contained in the extension word of the instruction. All memory alterable addressing modes may be used to reference the source operand. Absolute Short, I/O Short and Register Direct addressing modes may also be used. Note that if the specified source operand S is the SSH, the stack pointer register will be decremented by one; if the condition is true, the push operation will write over the stack level where the SSH value was taken. The bit to be tested is selected by an immediate bit number 0-23.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	×	×	×	×
CCR							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

BSSET #n,[X or Y]:ea,xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	1	1	0	M	M	M	R	R	R	0	S	1	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

BSSET #n,[X or Y]:aa,xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	1	1	0	a	a	a	a	a	a	1	S	1	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

BSSET #n,[X or Y]:pp,xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	1	1	1	p	p	p	p	p	p	0	S	1	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

BSSET #n,[X or Y]:qq,xxxx

23							16	15							8	7							0
0	0	0	0	0	1	0	0	1	0	q	q	q	q	q	q	1	S	1	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

BSSET #n,S,xxxx

23							16	15							8	7							0
0	0	0	0	1	1	0	1	1	1	D	D	D	D	D	D	1	0	1	b	b	b	b	b
PC RELATIVE DISPLACEMENT																							

Instruction Fields:

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address (see Table A-19 on page A-242)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{xxxx}		24-bit Relative Long Displacement
{aa}	**aaaaaa**	Absolute Address [0-63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0-$FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80-$FFFFBF]
{S}	**DDDDDD**	Source register [all on-chip registers] (see Table A-22 on page A-243)

A-6.22 Bit Test (BTST)

BTST BTST

Bit Test

Operation:	**Assembler Syntax:**
D[n] → C	BTST #n,[XorY]:ea
D[n] → C	BTST #n,[XorY]:aa
D[n] → C	BTST #n,[XorY]:pp
D[n] → C	BTST #n,[XorY]:qq
D[n] → C	BTST #n,D

Description: Test the nth bit of the destination operand D. The state of the nth bit is stored in the carry bit C of the condition code register. The bit to be tested is selected by an immediate bit number from 0–23. This instruction is useful for performing serial to parallel conversion when used with the appropriate rotate instructions. This instruction can use all memory alterable addressing modes.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✓	✓	×	×	×	×	×	●
CCR							

● C Set if bit tested is set. Cleared otherwise.
✓ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

SP — Stack Pointer:

For destination operand SSH: SP — Decrement by 1.
For other destination operands: Not affected

Instruction Formats and opcodes:

BTST #n,[X or Y]:ea

23							16	15							8	7							0
0	0	0	0	1	0	1	1	0	1	M	M	M	R	R	R	O	S	1	b	b	b	b	b
OPTIONAL EFFECTIVE ADDRESS EXTENSION																							

BTST #n,[X or Y]:aa

23							16	15							8	7							0
0	0	0	0	1	0	1	1	0	0	a	a	a	a	a	a	0	S	1	b	b	b	b	b

BTST #n,[X or Y]:pp

23							16	15							8	7							0
0	0	0	0	1	0	1	1	1	0	p	p	p	p	p	p	0	S	1	b	b	b	b	b

BTST #n,[X or Y]:qq

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	1	q	q	q	q	q	q	0	S	1	b	b	b	b	b

BTST #n,D

23							16	15							8	7							0
0	0	0	0	1	0	1	1	1	1	D	D	D	D	D	D	0	1	1	b	b	b	b	b

Instruction Fields:

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address (see Table A-18 on page A-241)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{aa}	**aaaaaa**	Absolute Address [0-63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0-$FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80-$FFFFBF]
{D}	**DDDDDD**	Destination register [all on-chip registers] (see Table A-22 on page A-243)

A-6.23 Count Leading Bits (CLB)

CLB CLB

Count Leading Bits

Operation: **Assembler Syntax:**

If S[55]=0 then CLB S,D
9 - (Number of consecutive leading zeros in S[55:0]) → D[47:24]
else
9 - (Number of consecutive leading ones in S[55:0]) → D[47:24]

Description: Count leading zeros or ones according to bit 55 of the source accumulator. Scan bits 55-0 of the source accumulator starting from bit 55. The MSP of the destination accumulator is loaded with 9 minus the number of consecutive leading ones or zeros found. The result is a signed integer in MSP whose range of possible values is from +8 to -47. This is a 56-bit operation. The LSP of the destination accumulator D is filled with zeros. The EXP of the destination accumulator D is sign extended.

Notes:

1) If the source accumulator is all zeros then the result will be zero.

2) When in sixteen bit arithmetic mode, the count ignores the unused 8 least significant bits of the MSP and LSP of the source accumulator. Therefore the result is a signed integer whose range of possible values is from +8 to -31.

3) This instruction may be used in conjunction with NORMF instruction, to specify the shift direction and amount needed for normalization.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	●	●	●	×
CCR							

● N Set if bit 47 of the result is set. Cleared otherwise
● Z Set if bits 47-24 of the result are zero.
● V Always cleared
× This bit is unchanged by the instruction

Example: CLB B,A

```
     4                        2                       
     7                        4                       0
B 11111011|111110001100101010010001|100110001100101010010001
  [_____]
5 Leading ones

     4                        2                       
     7                        4                       0
A 00000000|000000000000000000000100|000000000000000000000000
```

Result in A is 9 - 5 = 4

Instruction Formats and opcode:

CLB S,D

23							16	15							8	7							0
0	0	0	0	1	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	S	D

Instruction Fields:

{D}	**D**	Destination accumulator [A,B] (see Table A-10 on page A-239)
{S}	**S**	Source accumulator [A,B] (see Table A-10 on page A-239)

A-6.24 Clear accumulator (CLR)

CLR CLR

Clear Accumulator

Operation:

0→ D (parallel move)

Assembler Syntax:

CLR D (parallel move)

Description: Clear the destination accumulator. This is a 56-bit clear instruction.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	●	●	●	●	●	×
CCR							

- ● E Always cleared
- ● U Always set
- ● N Always cleared
- ● Z Always set
- ● V Always cleared
- ✔ This bit is changed according to the standard definition
- × This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23 16 15 8	7	0
CLR D	DATA BUS MOVE FIELD	0 0 0 1	d 0 1 1
	OPTIONAL EFFECTIVE ADDRESS EXTENSION		

Instruction Fields:

{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

A-6.25 Compare (CMP)

CMP CMP

Compare

Operation:	**Assembler Syntax:**
S2 – S1 (parallel move)	CMP S1, S2 (parallel move)
S2 – #xx	CMP #xx, S2
S2 – #xxxxxx	CMP #xxxxxx , S2

Description: Subtract the source one operand from the source two accumulator, S2, and update the condition code register. The result of the subtraction operation is not stored.

The source one operand can be a register (word - 24 bits or accumulator - 56 bits), short immediate (6 bits) or long immediate (24 bits). When using 6-bit immediate data, the data is interpreted as an unsigned integer. That is, the 6 bits will be right aligned and the remaining bits will be zeroed to form a 24-bit source operand.

Note: This instruction subtracts 56-bit operands. When a word is specified as the source one operand, it is sign extended and zero filled to form a valid 56-bit operand. For the carry to be set correctly as a result of the subtraction, S2 must be properly sign extended. S2 can be improperly sign extended by writing A1 or B1 explicitly prior to executing the compare so that A2 or B2, respectively, may not represent the correct sign extension. This note particularly applies to the case where it is extended to compare 24-bit operands such as X0 with A1.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	✔	✔	✔	✔	✔	✔
CCR							

✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Instruction Formats and opcodes:

CMP S1, S2

23							16	15							8	7							0
DATA BUS MOVE FIELD																0	J	J	J	d	1	0	1
OPTIONAL EFFECTIVE ADDRESS EXTENSION																							

CMP #xx, S2

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	1	i	i	i	i	i	i	1	0	0	0	d	1	0	1

CMP #xxxxxx,S2

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	1	0	0	d	1	0	1
IMMEDIATE DATA EXTENSION																							

Instruction Fields:

{S1}	**JJJ**	Source one register [B/A,X0,Y0,X1,Y1] (see Table A-24 on page A-243)
{S2}	**d**	Source two accumulator [A/B] (see Table A-10 on page A-239)
{#xx}	**iiiiii**	6-bit Immediate Short Data
{#xxxxxx}		24-bit Immediate Long Data extension word

A-6.26 Compare Magnitude (CMPM)

CMPM CMPM

Compare Magnitude

Operation:

|S2| – |S1|(parallel move)

Assembler Syntax:

CMPM S1, S2 (parallel move)

Description: Subtract the absolute value (magnitude) of the source one operand, S1, from the absolute value of the source two accumulator, S2, and update the condition code register. The result of the subtraction operation is not stored.

Note: This instruction subtracts 56-bit operands. When a word is specified as S1, it is sign extended and zero filled to form a valid 56-bit operand. For the carry to be set correctly as a result of the subtraction, S2 must be properly sign extended. S2 can be improperly sign extended by writing A1 or B1 explicitly prior to executing the compare so that A2 or B2, respectively, may not represent the correct sign extension. This note particularly applies to the case where it is extended to compare 24-bit operands such as X0 with A1.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	✔	✔	✔	✔	✔	✔
CCR							

✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23 16 15 8	7	0
CMPM S1, S2	DATA BUS MOVE FIELD	0 J J J	d 1 1 1
	OPTIONAL EFFECTIVE ADDRESS EXTENSION		

Instruction Fields:

{S1} **JJJ** Source one register [B/A,X0,Y0,X1,Y1] (see Table A-24 on page A-243)
{S2} **d** Source two accumulator [A,B] (see Table A-10 on page A-239)

A-6.27 Compare Unsigned (CMPU)

CMPU CMPU

Compare Unsigned

Operation:	**Assembler Syntax:**
S2 – S1	CMPU S1, S2

Description: Subtract the source one operand, S1, from the source two accumulator, S2, and update the condition code register. The result of the subtraction operation is not stored.

Note: This instruction subtracts a 24 or 48-bit unsigned operand from a 48-bit unsigned operand. When a 24-bit word is specified as S1 it is aligned to the left and zero filled to form a valid 48-bit operand. If an accumulator is specified as an operand, the value in the EXP does not affect the operation.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	✓	●	●	✓
CCR							

- ● V Always cleared
- ● Z Set if bits 47-0 of the result are zero
- × This bit is unchanged by the instruction
- ✓ This bit is changed according to the standard definition

Instruction Formats and opcodes:

	23							16	15							8	7							0
CMPU S1, S2	0	0	0	0	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	g	g	g	d

Instruction Fields:

{S1} **ggg** Source one register [A,B,X0,Y0,X1,Y1] (see Table A-15 on page A-240)
{S2} **d** Source two accumulator [A,B] (see Table A-10 on page A-239)

A-6.28 Enter Debug Mode (DEBUG)

DEBUG DEBUG

Enter Debug Mode

Operation: **Assembler Syntax:**

Enter the debug mode DEBUG

Description: Enter the debug mode and wait for OnCE commands.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23							16	15							8	7							0
DEBUG	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Instruction Fields: None

A-6.29 Enter Debug Mode Conditionally (DEBUGcc)

DEBUGcc DEBUGcc

Enter Debug Mode Conditionally

Operation:	**Assembler Syntax:**
If cc, then enter the debug mode	DEBUGcc

Description: If the specified condition is true, enter the debug mode and wait for OnCE commands. If the specified condition is false, continue with the next instruction.

The conditions that the term “cc” can specify are listed on Table A-42 on page A-250.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23							16	15							8	7							0
DEBUGcc	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	C	C	C	C

Instruction Fields:

{cc} **CCCC** Condition code (see Table A-43 on page A-251)

A-6.30 Decrement by One (DEC)

DEC DEC

Decrement by One

Operation:

D - 1➞ D

Assembler Syntax:

DEC D

Description: Decrement by one the specified operand and store the result in the destination accumulator. One is subtracted from the LSB of D.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	✔	✔	✔	✔	✔	✔	✔
CCR							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23							16	15							8	7							0
DEC D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	d

Instruction Fields:

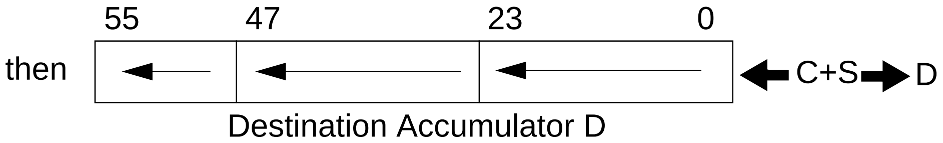
{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

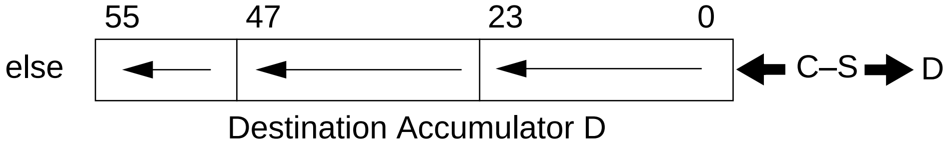
A-6.31 Divide Iteration (DIV)

DIV DIV

Divide Iteration

Operation: If $D[55] \oplus S[23] = 1$,





where $\oplus$ denotes the logical exclusive OR operator

Assembler Syntax: DIV S,D

Description:

Divide the destination operand D by the source operand S and store the result in the destination accumulator D. **The 48-bit dividend must be a positive fraction which has been sign extended to 56-bits and is stored in the full 56-bit destination accumulator D. The 24-bit divisor is a signed fraction and is stored in the source operand S.** Each DIV iteration calculates one quotient bit using a nonrestoring fractional division algorithm (see description on the next page). After the execution of the first DIV instruction, the destination operand holds both the partial remainder and the formed quotient. The partial remainder occupies the high-order portion of the destination accumulator D and is a signed fraction. The formed quotient occupies the low-order portion of the destination accumulator D (A0 or B0) and is a positive fraction. One bit of the formed quotient is shifted into the LS bit of the destination accumulator at the start of each DIV iteration. The formed quotient is the true quotient if the true quotient is positive. If the true quotient is negative, the formed quotient must be negated. **Valid results are obtained only when** $|D| < |S|$ **and the operands are interpreted as fractions.** Note that this condition ensures that the magnitude of the quotient is less than one (i.e., is fractional) and precludes division by zero.

The DIV instruction calculates one quotient bit based on the divisor and the previous partial remainder. To produce an N-bit quotient, the DIV instruction is executed N times where N is the number of bits of precision desired in the quotient, 1;leN;le24. Thus, for a full-precision (24 bit) quotient, 24 DIV iterations are required. In general, executing the DIV instruction N times produces an N-bit quotient and a 48-bit remainder which has (48–N) bits of precision and whose N MS bits are zeros. The partial remainder is not a true remainder and must be corrected due to the nonrestoring nature of the division algorithm before it may be used. Therefore, once the divide is complete, it is necessary to reverse the last DIV operation and restore the remainder to obtain the true remainder.

The DIV instruction uses a nonrestoring fractional division algorithm which consists of the following operations (see the previous **Operation** diagram):

1. **Compare the source and destination operand sign bits:** An exclusive OR operation is performed on bit 55 of the destination operand D and bit 23 of the source operand S;
2. **Shift the partial remainder and the quotient:** The 55-bit destination accumulator D is shifted one bit to the left. The carry bit C is moved into the LS bit (bit 0) of the accumulator;
3. **Calculate the next quotient bit and the new partial remainder:** The 24-bit source operand S (signed divisor) is either added to or subtracted from the MSP portion of the destination accumulator (A1 or B1), and the result is stored back into the MSP portion of that destination accumulator. If the result of the exclusive OR operation previously described was a “1” (i.e., the sign bits were different), the source operand S is added to the accumulator. If the result of the exclusive OR operation was a “0” (i.e., the sign bits were the same), the source operand S is subtracted from the accumulator. Due to the automatic sign extension of the 24-bit signed divisor, the addition or subtraction operation correctly sets the carry bit C of the condition code register with the next quotient bit.

For extended precision division (i.e., for N-bit quotients where N>24), the DIV instruction is no longer applicable, and a user-defined N-bit division routine is required. For further information on division algorithms, refer to pages 524–530 of *Theory and Application of Digital Signal Processing* by Rabiner and Gold (Prentice-Hall, 1975), pages 190–199 of *Computer Architecture and Organization* by John Hayes (McGraw-Hill, 1978), pages 213–223 of *Computer Arithmetic: Principles, Architecture, and Design* by Kai Hwang (John Wiley and Sons, 1979), or other references as required.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	●	×	×	×	×	●	●
CCR							

● L Set if overflow bit V is set

● V Set if the MS bit of the destination operand is changed as a result of the instruction’s left shift operation

● C Set if bit 55 of the result is cleared.

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23							16	15							8	7							0
DIV S,D	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	J	J	d	0	0	0

Instruction Fields:

{S} **JJ** Source input register [X0,X1,Y0,Y1] (see Table A-12 on page A-239)

{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

A-6.32 Double Precision MAC with 24 bit Right Shift (DMAC)

DMAC DMAC

Double (Multi) Precision Multiply Accumulate with Right Shift

Operation:	**Assembler Syntax:**	
[D>>24]±S1∗S2→D (S1 signed, S2 signed)	DMACss	(±)S1,S2,D (no parallel move)
[D>>24]±S1∗S2→D (S1 signed, S2 unsigned)	DMACsu	(±)S2,S1,D (no parallel move)
[D>>24]±S1∗S2→D (S1 unsigned, S2 unsigned)	DMACuu	(±)S2,S1,D (no parallel move)

Description: Multiply the two 24-bit source operands S1 and S2 and add/subtract the product to/from the specified 56-bit destination accumulator D, which has been previously shifted 24 bits to the right. The multiplication can be performed on signed numbers (ss), unsigned numbers (uu), or mixed (unsigned ∗ signed, (su)). The “–” sign option is used to negate the specified product prior to accumulation. The default sign option is “+”. This instruction is optimized for multiprecision multiplication support.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	✔	✔	✔	✔	✔	✔	×
CCR							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23							16	15							8	7							0
DMAC (±)S1,S2,D	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	s	1	s	d	k	Q	Q	Q	Q

Instruction Fields:

{S1,S2}	**QQQQ**	Source registers S1,S2 [all combinations of X0,X1,Y0 and Y1] (see Table A-30 on page A-245)
{D}	**d**	Destination accumulator [A,B] (see Table A-10 on page A-239)
{±±}	**k**	Sign [+,-] (see Table A-29 on page A-244)
{ss,su,uu}	**ss**	[ss,su,uu] (see Table A-39 on page A-248)

A-6.33 Start Hardware Loop (DO)

DO DO

Start Hardware Loop

Operation:	Assembler Syntax:
SP+1 → SP;LA → SSH;LC → SSL;[X or Y]:ea → LC SP+1 → SP;PC → SSH;SR → SSL;expr –1 → LA 1 → LF	DO [Xor Y]:ea,expr
SP+1 → SP;LA → SSH;LC → SSL;[Xor Y]:aa → LC SP+1 → SP;PC → SSH;SR → SSL;expr –1 → LA 1 → LF	DO [Xor Y]:aa,expr
SP+1 → SP;LA → SSH;LC → SSL;#xxx → LC SP+1 → SP;PC → SSH;SR → SSL;expr –1 → LA 1 → LF	DO #xxx,expr
SP+1 → SP;LA → SSH;LC → SSL;S → LC SP+1 → SP;PC → SSH;SR → SSL;expr –1 → LA 1 → LF	DO S,expr

End of Loop:
SSL(LF) → SR;SP–1 → SP
SSH → LA;SSL → LC;SP – 1 → SP

Description: Begin a hardware DO loop that is to be repeated the number of times specified in the instruction’s source operand and whose range of execution is terminated by the destination operand (previously shown as “expr”). No overhead other than the execution of this DO instruction is required to set up this loop. DO loops can be nested and the loop count can be passed as a parameter.

During the first instruction cycle, the current contents of the loop address (LA) and the loop counter (LC) registers are pushed onto the system stack. The DO instruction’s source operand is then loaded into the loop counter (LC) register. The LC register contains the remaining number of times the DO loop will be executed and can be accessed from inside the DO loop subject to certain restrictions. If LC initial value is zero and the 16-bit compatibility mode bit (bit 13, SC, in the Chip Status Register) is cleared, the DO loop is not executed. If LC initial value is zero but SC is set, the DO loop will be executed 65,536 times. All address register indirect addressing modes may be used to generate the effective address of the source operand. If immediate short data is specified, the 12 LS bits of LC are loaded with the 12-bit immediate value, and the 12 MS bits of LC are cleared.

During the second instruction cycle, the current contents of the program counter (PC) register and the status register (SR) are pushed onto the system stack. The stacking of the LA, LC, PC, and SR registers is the mechanism which permits the nesting of DO loops. The DO instruction’s destination operand (shown as “expr”) is then loaded into the loop address (LA) register. This 24 bit operand is located in the instruction’s 24-bit absolute address extension word as shown in the opcode section. The value in the program counter (PC) register pushed onto the system stack is the address of the first instruction following the DO instruction (i.e., the first actual instruction in the DO loop). This value is read (i.e., copied but not pulled) from the top of the system stack to return to the top of the loop for another pass through the loop.

During the third instruction cycle, the loop flag (LF) is set. This results in the PC being repeatedly compared with LA to determine if the last instruction in the loop has been fetched. If LA equals PC, the last instruction in the loop has been fetched and the loop counter (LC) is tested. If LC is not equal to one, it is decremented by one and SSH is loaded into the PC to fetch the first instruction in the loop again. If LC equals one, the “end-of-loop” processing begins.

When executing a DO loop, the instructions are actually fetched each time through the loop. Therefore, a DO loop can be interrupted. DO loops can also be nested. When DO loops are nested, the end-of-loop addresses must also be nested and are not allowed to be equal. The assembler generates an error message when DO loops are improperly nested.

Note: The assembler calculates the end-of-loop address to be loaded into LA (the absolute address extension word) by evaluating the end-of-loop expression “expr” and subtracting one. This is done to accommodate the case where the last word in the DO loop is a two-word instruction. Thus, the end-of-loop expression “expr” in the source code must represent the address of the instruction AFTER the last instruction in the loop.

During the “end-of-loop” processing, the loop flag (LF) from the lower portion (SSL) of SP is written into the status register (SR), the contents of the loop address (LA) register are restored from the upper portion (SSH) of (SP–1), the contents of the loop counter (LC) are restored from the lower portion (SSL) of (SP–1) and the stack pointer (SP) is decremented by two. Instruction fetches now continue at the address of the instruction following the last instruction in the DO loop. Note that LF is the only bit in the status register (SR) that is restored after a hardware DO loop has been exited.

Note: The loop flag (LF) is cleared by a hardware reset.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
●	●	×	×	×	×	×	×
CCR							

● S Set if the instruction sends A/B accumulator contents to XDB or YDB.
● L Set if data limiting occurred [see note 2]
× This bit is unchanged by the instruction

Instruction Formats and opcodes:

DO [X or Y]:ea, expr

23							16	15							8	7							0
0	0	0	0	0	1	1	0	0	1	M	M	M	R	R	R	0	S	0	0	0	0	0	0
ABSOLUTE ADDRESS EXTENSION WORD																							

DO [X or Y]:aa, expr

23							16	15							8	7							0
0	0	0	0	0	1	1	0	0	0	a	a	a	a	a	a	0	S	0	0	0	0	0	0
ABSOLUTE ADDRESS EXTENSION WORD																							

DO #xxx, expr

23							16	15							8	7							0
0	0	0	0	0	1	1	0	i	i	i	i	i	i	i	i	1	0	0	0	h	h	h	h
ABSOLUTE ADDRESS EXTENSION WORD																							

DO S, expr

23							16	15							8	7							0
0	0	0	0	0	1	1	0	1	1	D	D	D	D	D	D	0	0	0	0	0	0	0	0
ABSOLUTE ADDRESS EXTENSION WORD																							

Instruction Fields:

{ea}	**MMMRRR**	Effective Address (see Table A-19 on page A-242)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{expr}		24-bit Absolute Address in 24-bit extension word
{aa}	**aaaaaa**	Absolute Address [0-63]
{#xxx}	**hhhhiiiiiiii**	Immediate Short Data [0-4095]
{S}	**DDDDDD**	Source register [all on-chip registers, **except SSH**] (see Table A-22 on page A-243)

Note:

For DO SP, expr — The actual value that will be loaded into the loop counter (LC) is the value of the stack pointer (SP) before the execution of the DO instruction, incremented by 1.

Thus, if SP=3, the execution of the DO SP,expr instruction will load the loop counter (LC) with the value LC=4.

For DO SSL, expr — The loop counter (LC) will be loaded with its previous value which was saved on the stack by the DO instruction itself.

A-6.34 Start Infinite Loop (DO FOREVER)

DO FOREVER DO FOREVER

Start Infinite Loop

Operation:	**Assembler Syntax:**
SP+1 → SP;LA → SSH;LC → SSL SP+1 → SP;PC → SSH;SR → SSL;expr –1 → LA 1 → LF; 1 →FV	DO FOREVER,expr

Description: Begin a hardware DO loop that is to be repeated for ever and whose range of execution is terminated by the destination operand (shown above as “expr”). No overhead other than the execution of this DO FOREVER instruction is required to set up this loop. DO FOREVER loops can be nested. During the first instruction cycle, the current contents of the Loop Address (LA) and the Loop Counter (LC) registers are pushed onto the system stack. The loop counter (LC) register is pushed onto the stack but is not updated by this instruction.

During the second instruction cycle, the current contents of the Program Counter (PC) register and the Status Register (SR) are pushed onto the system stack. Stacking the LA, LC, PC, and SR registers permits nesting DO FOREVER loops. The DO FOREVER instruction’s destination operand (shown as “expr”) is then loaded into the Loop Address (LA) register . This 24-bit operand is located in the instruction’s 24-bit absolute address extension word as shown in the opcode section. The value in the Program Counter (PC) register pushed onto the system stack is the address of the first instruction following the DO FOREVER instruction (i.e., the first actual instruction in the DO FOREVER loop). This value is read (i.e., copied but not pulled) from the top of the system stack to return to the top of the loop for another pass through the loop.

During the third instruction cycle, the Loop Flag (LF) and the ForeVer flag are set. This results in the PC being repeatedly compared with LA to determine if the last instruction in the loop has been fetched. If LA equals PC, the last instruction in the loop has been fetched and SSH is loaded into the PC to fetch the first instruction in the loop again. The loop counter (LC) register is then decremented by one without being tested. This register can be used by the programer to count the number of loops already executed.

When executing a DO FOREVER loop, the instructions are actually fetched each time through the loop. Therefore, a DO FOREVER loop can be interrupted. DO FOREVER loops can also be nested. When DO FOREVER loops are nested, the end of loop addresses must also be nested and are not allowed to be equal. The assembler generates an error message when DO FOREVER loops are improperly nested.

Note: The assembler calculates the end-of-loop address to be loaded into LA (the absolute address extension word) by evaluating the end-of-loop expression “expr” and subtracting one. This is done to accommodate the case where the last word in the DO loop is a two-word instruction. Thus, the end-of-loop expression “expr” in the source code must represent the address of the instruction AFTER the last instruction in the loop.

The loop counter (LC) register is never tested by the DO FOREVER instruction and the only way of terminating the loop process is to use either the ENDDO or BRKcc instructions. LC is decremented every time PC=LA so that it can be used by the programmer to keep track of the number of times the DO FOREVER loop has been executed. If the programer wants to initialize LC to a particular value before the DO FOREVER, care should be taken to save it before if the DO loop is nested. If so, LC should also be restored immediately after exiting the nested DO FOREVER loop.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23 … 16	15 … 8	7 … 0
DO FOREVER	0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 0	0 0 0 0 0 0 1 1
	ABSOLUTE ADDRESS EXTENSION WORD		

Instruction Fields: None.

A-6.35 Start PC Relative Hardware Loop (DOR)

DOR DOR

Start PC relative Hardware Loop

Operation:	**Assembler Syntax:**
SP+1 → SP;LA → SSH;LC → SSL;[X or Y]:ea → LC SP+1 → SP;PC → SSH;SR → SSL;PC+xxxx → LA 1 → LF	DOR [Xor Y]:ea,label
SP+1 → SP;LA → SSH;LC → SSL;[X or Y]:ea → LC SP+1 → SP;PC → SSH;SR → SSL;PC+xxxx → LA 1 → LF	DOR [Xor Y]:aa,label
SP+1 → SP;LA → SSH;LC → SSL;#xxx → LC SP+1 → SP;PC → SSH;SR → SSL;PC+xxxx → LA 1 → LF	DOR #xxx,label
SP+1 → SP;LA → SSH;LC → SSL;S → LC SP+1 → SP;PC → SSH;SR → SSL;PC+xxxx → LA 1 → LF	DOR S,label

Description:

This instruction initiates the beginning of a PC relative hardware program loop. The current loop address (LA) and loop counter (LC) values are pushed onto the system stack. With proper system stack management, this allows unlimited nested hardware DO loops. The PC and SR are pushed onto the system stack. The PC is added to the 24-bit address displacement extension word and the resulting address is loaded into the loop address register (LA). The effective address specifies the address of the loop count which is loaded into the loop counter (LC). The DO loop is executed LC times. If LC initial value is zero and the 16-bit compatibility mode bit (bit 13, SC, in the Chip Status Register) is cleared, the DO loop is not executed. If LC initial value is zero but SC is set, the DO loop will be executed 65,536 times. All address register indirect addressing modes (less Long Displacement) may be used. Register Direct addressing mode may also be used. If immediate short data is specified, the LC is loaded with the zero extended 12-bit immediate data.

During hardware loop operation, each instruction is fetched each time through the program loop. Therefore, instructions being executed in a hardware loop are interruptible and may be nested. The value of the PC pushed onto the system stack is the location of the first instruction after the DOR instruction. This value is read from the top of the system stack to return to the start of the program loop. When DOR instructions are nested, the end of loop addresses must also be nested and are not allowed to be equal.

The assembler calculates the end of loop address LA (PC relative address extension word xxxx) by evaluating the end of loop expression and subtracting one. Thus the end of loop expression in the source code represents the “next address” after the end of the loop. If a simple end of loop address label is used, it should be placed after the last instruction in the loop.

Since the end of loop comparison is at fetch time and ahead of the end of loop execution, instructions which change program flow or change the system stack may not be used near the end of the loop without some restrictions. Proper hardware loop operation is guaranteed if no instruction starting at address LA-2, LA-1 or LA specifies the program controller registers SR, SP, SSL, LA, LC or (implicitly) PC as a destination register; or specifies SSH as a source or destination register. Also, SSH cannot be specified as a source register in the DOR instruction itself. The assembler will generate a warning if the restricted instructions are found within their restricted boundaries.

Implementation Notes:

DOR SP,xxxx The actual value that will be loaded in the LC is the value of the SP before the DOR instruction incremented by one.

DOR SSL,xxxx The LC will be loaded with its previous value that was saved in the stack by the DOR instruction itself.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
●	●	×	×	×	×	×	×
CCR							

● S Set if the instruction sends A/B accumulator contents to XDB or YDB.

● L Set if data limiting occurred

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

DOR [X or Y]:ea,label

23							16	15							8	7							0
0	0	0	0	0	1	1	0	0	1	M	M	M	R	R	R	0	S	0	1	0	0	0	0
PC RELATIVE DISPLACEMENT																							

DOR [X or Y]:aa,label

23							16	15							8	7							0
0	0	0	0	0	1	1	0	0	0	a	a	a	a	a	a	0	S	0	1	0	0	0	0
PC RELATIVE DISPLACEMENT																							

DOR #xxx, label

23							16	15							8	7							0
0	0	0	0	0	1	1	0	i	i	i	i	i	i	i	i	1	0	0	1	h	h	h	h
PC RELATIVE DISPLACEMENT																							

DOR S, label

23							16	15							8	7							0
0	0	0	0	0	1	1	0	1	1	D	D	D	D	D	D	0	0	0	1	0	0	0	0
PC RELATIVE DISPLACEMENT																							

Instruction Fields:

{ea}	**MMMRRR**	Effective Address (see Table A-19 on page A-242)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{label}		24-bit Address Displacement in 24-bit extension word
{aa}	**aaaaaa**	Absolute Address [0-63]
{#xxx}	**hhhhiiiiiiii**	Immediate Short Data [0-4095]
{S}	**DDDDDD**	Source register [all on-chip registers **except SSH**] (see Table A-22 on page A-243)

A-6.36 Start PC Relative Infinite Loop (DOR FOREVER)

DOR FOREVER DOR FOREVER

Start PC Relative Infinite Loop

Operation:	**Assembler Syntax:**
SP+1 → SP;LA → SSH;LC → SSL SP+1 → SP;PC → SSH;SR → SSL;PC+xxxx → LA 1 → LF; 1 →FV	DOR FOREVER,label

Description: Begin a hardware DO loop that is to be repeated for ever and whose range of execution is terminated by the destination operand (shown above as label). No overhead other than the execution of this DOR FOREVER instruction is required to set up this loop. DOR FOREVER loops can be nested. During the first instruction cycle, the current contents of the Loop Address (LA) and the Loop Counter (LC) registers are pushed onto the system stack. The loop counter (LC) register is pushed onto the stack but is not updated by this instruction.

During the second instruction cycle, the current contents of the Program Counter (PC) register and the Status Register (SR) are pushed onto the system stack. Stacking the LA, LC, PC, and SR registers permits nesting DOR FOREVER loops. The DOR FOREVER instruction's destination operand (shown as label) is then loaded into the Loop Address (LA) register after having been added to the PC. This 24-bit operand is located in the instruction's 24-bit relative address extension word as shown in the opcode section. The value in the Program Counter (PC) register pushed onto the system stack is the address of the first instruction following the DOR FOREVER instruction (i.e., the first actual instruction in the DOR FOREVER loop). This value is read (i.e., copied but not pulled) from the top of the system stack to return to the top of the loop for another pass through the loop.

During the third instruction cycle, the Loop Flag (LF) and the ForeVer flag are set. This results in the PC being repeatedly compared with LA to determine if the last instruction in the loop has been fetched. If LA equals PC, the last instruction in the loop has been fetched and SSH is read (i.e copied but not pulled) into the PC to fetch the first instruction in the loop again. The loop counter (LC) register is then decremented by one without being tested. This register can be used by the programer to count the number of loops already executed.

When executing a DOR FOREVER loop, the instructions are actually fetched each time through the loop. Therefore, a DOR FOREVER loop can be interrupted. DOR FOREVER loops can also be nested. When DOR FOREVER loops are nested, the end of loop addresses must also be nested and are not allowed to be equal. The assembler generates an error message when DOR FOREVER loops are improperly nested.

Note: The assembler calculates the end of loop address LA (PC relative address extension word xxxx) by evaluating the end of loop expression and subtracting one. Thus the end of loop expression in the source code represents the “next address” after the end of the loop. If a simple end of loop address label is used, it should be placed after the last instruction in the loop.

The loop counter (LC) register is never tested by the DOR FOREVER instruction and the only way of terminating the loop process is to use either the ENDDO or BRKcc instructions. LC is decremented every time PC=LA so that it can be used by the programmer to keep track of the number of times the DOR FOREVER loop has been executed. If the programer wants to initialize LC to a particular value before the DOR FOREVER, care should be taken to save it before if the DO loop is nested. If so, LC should also be restored immediately after exiting the nested DOR FOREVER loop.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23							16	15							8	7							0
DOR FOREVER	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
	PC RELATIVE DISPLACEMENT																							

Instruction Fields: None.

A-6.37 End Current DO Loop (ENDDO)

ENDDO ENDDO

End Current DO Loop

Operation:	**Assembler Syntax:**
SSL(LF) → SR;SP – 1→ SP SSH → LA; SSL → LC;SP –1 → SP	ENDDO

Description: Terminate the current hardware DO loop before the current loop counter (LC) equals one. If the value of the current DO loop counter (LC) is needed, it must be read before the execution of the ENDDO instruction. Initially, the loop flag (LF) is restored from the system stack and the remaining portion of the status register (SR) and the program counter (PC) are purged from the system stack. The loop address (LA) and the loop counter (LC) registers are then restored from the system stack.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23							16	15							8	7							0
ENDDO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0

Instruction Fields: None

A-6.38 Logical Exclusive OR (EOR)

EOR EOR

Logical Exclusive OR

Operation:	**Assembler Syntax:**
S ⊕ D[47:24]→D[47:24] (parallel move)	EOR S,D (parallel move)
#xx ⊕ D[47:24]→D[47:24]	EOR #xx,D
#xxxxxx ⊕ D[47:24]→D[47:24]	EOR #xxxxxx,D

where ⊕ denotes the logical XOR operator

Description: Logically exclusive OR the source operand S with bits 47–24 of the destination operand D and store the result in bits 47–24 of the destination accumulator. The source can be a 24-bit register, 6-bit short immediate or 24-bit long immediate. This instruction is a 24-bit operation. The remaining bits of the destination operand D are not affected.

When using 6-bit immediate data, the data is interpreted as an unsigned integer. That is, the 6 bits will be right aligned and the remaining bits will be zeroed to form a 24-bit source operand.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	●	●	●	×
CCR							

● N Set if bit 47 of the result is set
● Z Set if bits 47-24 of the result are zero

● V Always cleared

✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Instruction Formats and opcodes:

EOR S,D

23							16	15							8	7							0
DATA BUS MOVE FIELD																0	1	J	J	d	0	1	1
OPTIONAL EFFECTIVE ADDRESS EXTENSION																							

EOR #xx,D

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	1	i	i	i	i	i	i	1	0	0	0	d	0	1	1

EOR #xxxxxx,D

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	1	0	0	d	0	1	1
IMMEDIATE DATA EXTENSION																							

Instruction Fields:

{S}	**JJ**	Source register [X0,X1,Y0,Y1] (see Table A-12 on page A-239)
{D}	**d**	Destination accumulator [A/B] (see Table A-10 on page A-239)
{#xx}	**iiiiii**	6-bit Immediate Short Data
{#xxxxxx}		24-bit Immediate Long Data extension word

A-6.39 Bit Field (EXTRACT)

EXTRACT EXTRACT

Extract Bit Field

Operation:	**Assembler Syntax:**

Offset = S1[5:0]
Width = S1[17:12]
	EXTRACT S1,S2,D

S2[(offset+width-1):offset] → D[(width-1):0] S2[offset+width-1] → D[55:width] (sign extension)	

Offset = #CO[5:0]
Width = #CO[17:12]
	EXTRACT #CO,S2,D

S2[(offset+width-1):offset] → D[(width-1):0] S2[offset+width-1] → D[55:width] (sign extension)	

Description: Extract a bit-field from source accumulator S2. The bit-field width is specified by bits 17-12 in S1 register or in immediate control word #CO. The offset from the least significant bit is specified by bits 5-0 in S1 register or in immediate control word #CO. The extracted field is placed in the destination accumulator D, aligned to the right. The construction of the control register can be done by using the MERGE instruction.

This is a 56 bit operation. Bits outside the field are filled with sign extension according to the most significant bit of the extracted bit field.

Notes:

1) In 16 bit arithmetic mode, the offset field is located in bits 13-8 of the control register and the width field is located in bits 21-16 of the control register. These fields corresponds to the definition of the fields in the MERGE instruction.

2) In 16 bit arithmetic mode, when the width value is zero, then the result will be undefined.

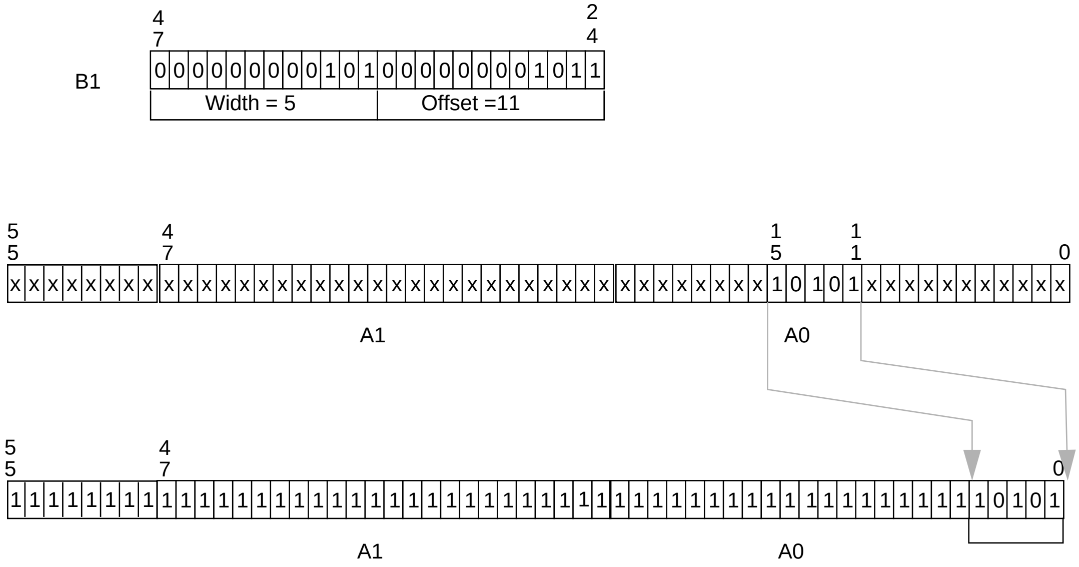
3) If offset + width exceeds the value of 56, the result will be undefined.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	✔	✔	✔	✔	●	●
CCR							

- ● V Always cleared
- ● C Always cleared
- × This bit is unchanged by the instruction
- ✔ This bit is changed according to the standard definition

Example: EXTRACT B1,A,A



Instruction Formats and opcodes:

	23							16	15							8	7							0
EXTRACT S1,S2,D	0	0	0	0	1	1	0	0	0	0	0	1	1	0	1	0	0	0	0	s	S	S	S	D

	23							16	15							8	7							0
EXTRACT #CO,S2,D	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	s	0	0	0	D
	CONTROL WORD EXTENSION																							

Instruction Fields:

{S2}	**s**	Source accumulator [A,B] (see Table A-10 on page A-239)
{D}	**D**	Destination accumulator [A,B] (see Table A-10 on page A-239)
{S1}	**SSS**	Control register [X0,X1,Y0,Y1,A1,B1] (see Table A-15 on page A-240)
{#CO}		Control word extension.

A-6.40 Extract Unsigned Bit Field (EXTRACTU)

EXTRACTU EXTRACTU

Extract Unsigned Bit Field

Operation:	**Assembler Syntax:**

Offset = S1[5:0]
Width = S1[17:12]
	EXTRACTU S1,S2,D

S2[(offset+width-1):offset] → D[(width-1):0] zero → D[55:width]	

Offset = #CO[5:0]
Width = #CO[17:12]
	EXTRACTU #CO,S2,D

S2[(offset+width-1):offset] → D[(width-1):0] zero → D[55:width]	

Description: Extract an unsigned bit-field from source accumulator S2. The bit-field width is specified by bits 17-12 in S1 register or in immediate control word #CO. The offset from the least significant bit is specified by bits 5-0 in S1 register or in immediate control word #CO. The extracted field is placed in the destination accumulator D, aligned to the right. The construction of the control register can be done by using the MERGE instruction.

This is a 56 bits operation. Bits outside the field are filled with zeros.

Notes:

1) in 16 bit arithmetic mode, the offset field is located in bits 13-8 of the control register and the width field is located in bits 21-16 of the control register. These fields corresponds to the definition of the fields in the MERGE instruction.

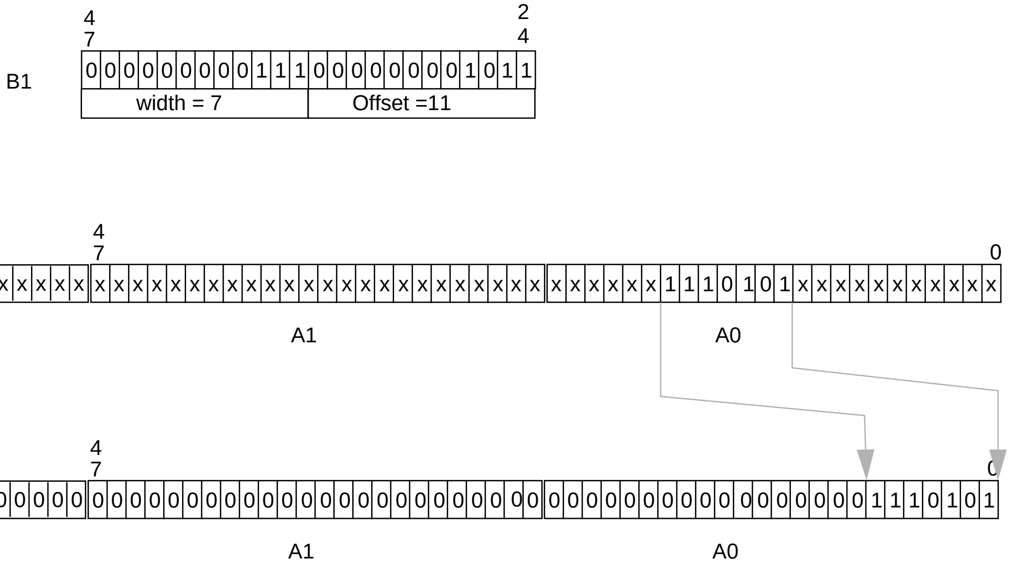
2) If offset + width exceeds the value of 56, the result will be undefined.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	✔	✔	✔	✔	●	●
CCR							

- ● V Always cleared
- ● C Always cleared
- × This bit is unchanged by the instruction
- ✔ This bit is changed according to the standard definition

Example :EXTRACTU B1,A,A



Instruction Formats and opcodes:

	23							16	15							8	7							0
EXTRACTU S1,S2,D	0	0	0	0	1	1	0	0	0	0	0	1	1	0	1	0	1	0	0	s	S	S	S	D

	23							16	15							8	7							0
EXTRACTU #CO,S2,D	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	1	0	0	s	0	0	0	D
	CONTROL WORD EXTENSION																							

Instruction Fields:

{S2}	**s**	Source accumulator [A,B] (see Table A-10 on page A-239)
{D}	**D**	Destination accumulator [A,B] (see Table A-10 on page A-239)
{S1}	**SSS**	Control register [X0,X1,Y0,Y1,A1,B1] (see Table A-15 on page A-240)
{#CO}		Control word extension.

A-6.41 Execute Conditionally without CCR Update (IFcc)

IFcc IFcc

Execute Conditionally without CCR Update

Operation:
If cc, then opcode operation

Assembler Syntax:
Opcode-Operands IFcc

Description: If the specified condition is true, execute and store result of the specified Data ALU operation. If the specified condition is false, no destination is altered. The CCR is never updated with the condition codes generated by the Data ALU operation.

The instructions that can conditionally be executed by using IFcc are the arithmetic and logical instructions that are considered as “parallel” instructions. See Table A-3 and Table A-4 for a list of those instructions.

The conditions that the term “**cc**” may specify are listed on Table A-42 on page A-250

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23 16	15 8	7 0
IFcc	0 0 1 0 0 0 0 0	0 0 1 0 C C C C	INSTRUCTION OPCODE

Instruction Fields:

{cc} **CCCC** Condition code (see Table A-43 on page A-251)

A-6.42 Execute Conditionally with CCR Update (IFcc.U)

IFcc.U IFcc.U

Execute Conditionally with CCR Update

Operation:
If cc, then opcode operation

Assembler Syntax:
Opcode-Operands IFcc

If the specified condition is true, execute and store result of the specified Data ALU operation and update the CCR with the status information generated by the Data ALU operation. If the specified condition is false, no destination is altered and the CCR is not affected.

The instructions that can conditionally be executed by using IFcc.U are the arithmetic and logical instructions that are considered as “parallel” instructions. See Table A-3 and Table A-4 for a list of those instructions.

The conditions that the term “**cc**” may specify are listed on Table A-42 on page A-250

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
●	●	●	●	●	●	●	●
CCR							

● If the specified condition is true changed according to the instruction. Not changed otherwise.

Instruction Formats and opcodes:

	23							16	15							8	7 0
IFcc.U	0	0	1	0	0	0	0	0	0	0	1	1	C	C	C	C	INSTRUCTION OPCODE

Instruction Fields:

{cc}	**CCCC**	Condition code (see Table A-43 on page A-251)

A-6.43 Illegal Instruction Interrupt (ILLEGAL)

ILLEGAL ILLEGAL

Illegal Instruction Interrupt

Operation:
Begin Illegal Instruction exception processing

Assembler Syntax:
Opcode-Operands IFcc

Description: The ILLEGAL instruction is executed as if it were a NOP instruction. Normal instruction execution is suspended and illegal instruction exception processing is initiated. The interrupt vector address is located at address P:$3E. The interrupt priority level (I1, I0) is set to 3 in the status register if a long interrupt service routine is used. The purpose of the ILLEGAL instruction is to force the DSP into an illegal instruction exception for test purposes. Exiting an illegal instruction is a fatal error. A long exception routine should be used to indicate this condition and cause the system to be restarted.

If the ILLEGAL instruction is in a DO loop at LA and the instruction at LA–1 is being interrupted, then LC will be decremented twice due to the same mechanism that causes LC to be decremented twice if JSR, REP, etc. are located at LA. This is why JSR, REP, etc. at LA are restricted. Clearly restrictions cannot be imposed on illegal instructions.

Since REP is uninterruptable, repeating an ILLEGAL instruction results in the interrupt not being initiated until after completion of the REP. After servicing the interrupt, program control will return to the address of the second word following the ILLEGAL instruction. Of course, the ILLEGAL interrupt service routine should abort further processing, and the processor should be reinitialized.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23							16	15							8	7							0
ILLEGAL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Instruction Fields: None

A-6.44 Increment by One (INC)

INC INC

Increment by One

Operation:

D +1➞ D

Assembler Syntax:

INC D

Description: Increment by one the specified operand and store the result in the destination accumulator. One is added from the LSB of D.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	✔	✔	✔	✔	✔	✔	✔
CCR							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23 16	15 8	7 0
INC D	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 1 0 0 d

Instruction Fields:

{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

A-6.45 Insert Bit field (INSERT)

INSERT INSERT

Insert Bit Field

Operation:	**Assembler Syntax:**

Offset =S1[5:0]
Width =S1[17:12]
	INSERT S1,S2,D

S2[(width-1):0] → D[(offset+width-1):offset]	

Offset = #CO[5:0]
Width = #CO[17:12]
	INSERT #CO,S2,D

S2[(width-1):0] → D[(offset+width-1):offset]	

Description: Insert a bit-field into the destination accumulator D. The bit-field whose width is specified by bits 17-12 in S1 register, begins at the least significant bit of the S2 register. This bit-field is inserted in the destination accumulator D, with an offset according to bits 5-0 in S1 register. S1 operand can be an immediate control word #CO. Width specified by S1 should not exceed value of 24. The construction of the control register can be done by using the MERGE instruction.
This is a 56 bit operation. Any bits outside the field remain unchanged.

Notes:

1) In 16 bit arithmetic mode, the offset field is located in bits 13-8 of the control register and the width field is located in bits 21-16 of the control register. These fields corresponds to the definition of the fields in the MERGE instruction. Width specified by S1 should not exceed value of 16.

2) In 16 bit arithmetic mode, the offset value, located in the offset field, should be the needed offset pre-incremented by the user by a bias of 16.

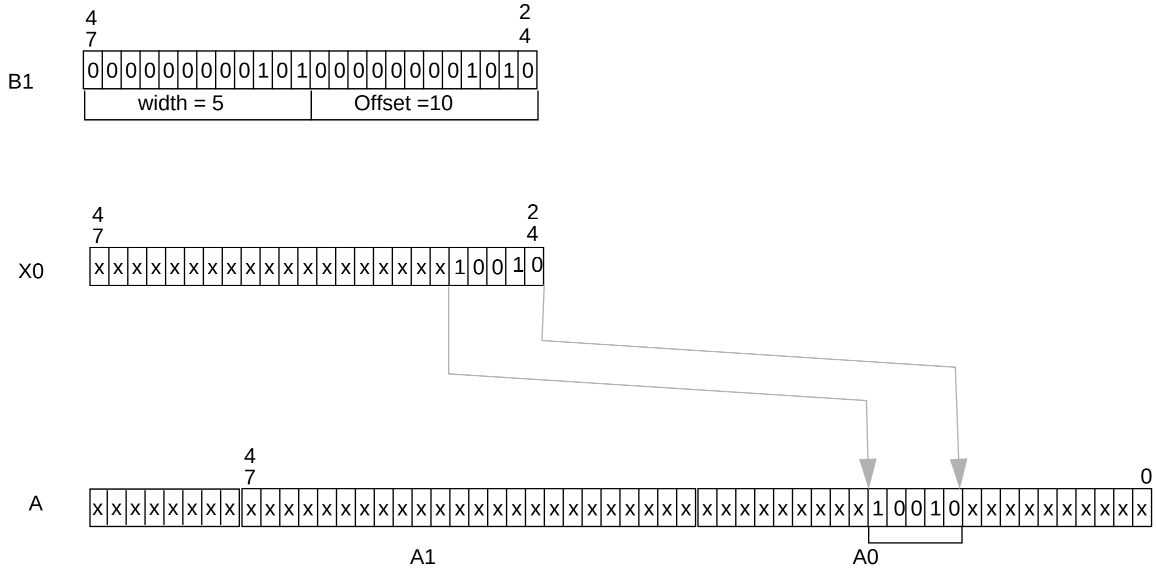
2) If offset + width exceeds the value of 56, the result will be undefined.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	✓	✓	✓	✓	●	●
CCR							

●	V	Always cleared
●	C	Always cleared
×		This bit is unchanged by the instruction
✓		This bit is changed according to the standard definition

Example: INSERT B1,X0,A



Instruction Formats and opcodes:

INSERT S1,S2,D

23							16	15							8	7							0
0	0	0	0	1	1	0	0	0	0	0	1	1	0	1	1	0	q	q	q	S	S	S	D

INSERT #CO,S2,D

23							16	15							8	7							0
0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	1	0	q	q	q	0	0	0	D
CONTROL WORD EXTENSION																							

Instruction Fields:

{D}	**D**	Destination accumulator [A,B] (see Table A-10 on page A-239)
{S1}	**SSS**	Control register [X0,X1,Y0,Y1,A1,B1] (see Table A-15 on page A-240)
{S2}	**qqq**	Source register [X0,X1,Y0,Y1,A0,B0] (see Table A-15 on page A-240)
{#CO}		Control word extension.

A-6.46 Jump Conditionally (JCC)

Jcc Jcc

Jump Conditionally

Operation:	Assembler Syntax:
If cc, then 0xxxx →PC else PC+1 →PC	Jcc xxx
If cc, then ea →PC else PC+1 →PC	Jcc ea

Description: Jump to the location in program memory given by the instruction's effective address if the specified condition is true. If the specified condition is false, the program counter (PC) is incremented and the effective address is ignored. However, the address register specified in the effective address field is always updated independently of the specified condition. All memory alterable addressing modes may be used for the effective address. A Fast Short Jump addressing mode may also be used. The 12-bit data is zero extended to form the effective address.

The conditions that the term "**cc**" can specify are listed on Table A-42 on page A-250.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

Jcc xxx

23 16	15 8	7 0
0 0 0 0 1 1 1 0	C C C C a a a a	a a a a a a a a

Jcc ea

23 16	15 8	7 0
0 0 0 0 1 0 1 0	1 1 M M M R R R	1 0 1 0 C C C C
OPTIONAL EFFECTIVE ADDRESS EXTENSION		

Instruction Fields:

{cc}	**CCCC**	Condition code (see Table A-43 on page A-251)
{xxx}	**aaaaaaaaaaaa**	Short Jump Address
{ea}	**MMMRRR**	Effective Address (see Table A-18 on page A-241)

A-6.47 Jump if Bit Clear (JCLR)

JCLR JCLR

Jump if Bit Clear

Operation:	Assembler Syntax:
If S{n}=0 then xxxx → PC else PC+ 1 → PC	JCLR #n,[X or Y]:ea,xxxx
If S{n}=0 then xxxx → PC else PC+ 1 → PC	JCLR #n,[X or Y],aa,xxxx
If S{n}=0 then xxxx → PC else PC+ 1 → PC	JCLR #n,[X or Y]:pp,xxxx
If S{n}=0 then xxxx → PC else PC+ 1 → PC	JCLR #n,[X or Y]:qq,xxxx
If S{n}=0 then xxxx → PC else PC+ 1 → PC	JCLR #n,S,xxxx

Description: Jump to the 24-bit absolute address in program memory specified in the instruction’s 24-bit extension word if the nth bit of the source operand S is clear. The bit to be tested is selected by an immediate bit number from 0–23. If the specified memory bit is not clear, the program counter (PC) is incremented and the absolute address in the extension word is ignored. However, the address register specified in the effective address field is always updated independently of the state of the nth bit. All address register indirect addressing modes may be used to reference the source operand S. Absolute Short and I/O Short addressing modes may also be used.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	×	×	×	×
CCR							

✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Instruction Formats and opcodes:

JCLR #n,[X or Y]:ea,xxxx

23							16	15							8	7							0
0	0	0	0	1	0	1	0	0	1	M	M	M	R	R	R	1	S	0	b	b	b	b	b
ABSOLUTE ADDRESS EXTENSION																							

JCLR #n,[X or Y]:aa,xxxx

23							16	15							8	7							0
0	0	0	0	1	0	1	0	0	0	a	a	a	a	a	a	1	S	0	b	b	b	b	b
ABSOLUTE ADDRESS EXTENSION																							

JCLR #n,[X or Y]:pp,xxxx

23							16	15							8	7							0
0	0	0	0	1	0	1	0	1	0	p	p	p	p	p	p	1	S	0	b	b	b	b	b
ABSOLUTE ADDRESS EXTENSION																							

JCLR #n,[X or Y]:qq,xxxx

23							16	15							8	7							0
0	0	0	0	0	0	0	1	1	0	q	q	q	q	q	q	1	S	0	b	b	b	b	b
ABSOLUTE ADDRESS EXTENSION																							

JCLR #n,S,xxxx

23							16	15							8	7							0
0	0	0	0	1	0	1	0	1	1	D	D	D	D	D	D	0	0	0	b	b	b	b	b
ABSOLUTE ADDRESS EXTENSION																							

Instruction Fields:

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address (see Table A-19 on page A-242)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{xxxx}		24-bit absolute Address extension word
{aa}	**aaaaaa**	Absolute Address [0-63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0-$FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80-$FFFFBF]
{S}	**DDDDDD**	Source register [all on-chip registers] (see Table A-22 on page A-243)

A-6.48 Jump (JMP)

JMP JMP

Jump

Operation:	Assembler Syntax:
0xxx ➞ Pc	JMP xxx
ea➞ Pc	JMP ea

Description: Jump to the location in program memory given by the instruction’s effective address. All memory alterable addressing modes may be used for the effective address. A Fast Short Jump addressing mode may also be used. The 12-bit data is zero extended to form the effective address.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

JMP ea

23 16	15 8	7 0
0 0 0 0 1 0 1 0	1 1 M M M R R R	1 0 0 0 0 0 0 0
OPTIONAL EFFECTIVE ADDRESS EXTENSION		

JMP xxx

23 16	15 8	7 0
0 0 0 0 1 1 0 0	0 0 0 0 a a a a	a a a a a a a a

Instruction Fields:

{xxx}	**aaaaaaaaaaaa**	Short Jump Address
{ea}	**MMMRRR**	Effective Address (see Table A-18 on page A-241)

A-6.49 Jump to Subroutine Conditionally (JScc)

JScc JScc

Jump to Subroutine Conditionally

Operation:				**Assembler Syntax:**	
If	cc,	then	SP+1→SP; PC →SSH;SR →SSL;0xxx →PC	JScc	xxx
		else	PC+1→PC		
If	cc,	then	SP+1→SP; PC →SSH;SR →SSL;ea →PC	JScc	ea
		else	PC+1→PC		

Description: Jump to the subroutine whose location in program memory is given by the instruction's effective address if the specified condition is true. If the specified condition is true, the address of the instruction immediately following the JScc instruction (PC) and the system status register (SR) are pushed onto the system stack. Program execution then continues at the specified effective address in program memory. If the specified condition is false, the program counter (PC) is incremented, and any extension word is ignored. However, the address register specified in the effective address field is always updated independently of the specified condition. All memory alterable addressing modes may be used for the effective address. A fast short jump addressing mode may also be used. The 12-bit data is zero extended to form the effective address.

The conditions that the term "**cc**" can specify are listed on Table A-42 on page A-250.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

		23							16	15							8	7							0
JScc	xxx	0	0	0	0	1	1	1	1	C	C	C	C	a	a	a	a	a	a	a	a	a	a	a	a

		23							16	15							8	7							0
JScc	ea	0	0	0	0	1	0	1	1	1	1	M	M	M	R	R	R	1	0	1	0	C	C	C	C
		OPTIONAL EFFECTIVE ADDRESS EXTENSION																							

Instruction Fields:

{cc}	**CCCC**	Condition code (see Table A-43 on page A-251)
{xxx}	**aaaaaaaaaaaa**	Short Jump Address
{ea}	**MMMRRR**	Effective Address (see Table A-18 on page A-241)

A-6.50 Jump to Subroutine if Bit Clear (JSCLR)

JSCLR JSCLR

Jump to Subroutine if Bit Clear

Operation:	Assembler Syntax:

If S{n}=0 then SP+1→SP;PC →SSH;SR →SSL;
;xxxx →PC
else PC+1→PC
	JSCLR #n,[X or Y]:ea,xxxx

If S{n}=0 then SP+1→SP;PC →SSH;SR →SSL;
;xxxx →PC
else PC+1→PC
	JSCLR #n,[X or Y],aa,xxxx

If S{n}=0 then SP+1→SP;PC →SSH;SR →SSL;
;xxxx →PC
else PC+1→PC
	JSCLR #n,[X or Y]:pp,xxxx

If S{n}=0 then SP+1→SP;PC →SSH;SR →SSL;
;xxxx →PC
else PC+1→PC
	JSCLR #n,[X or Y]:qq,xxxx

If S{n}=0 then SP+1→SP;PC →SSH;SR →SSL;
;xxxx →PC
else PC+1→PC
	JSCLR #n,S,xxxx

Description: Jump to the subroutine at the 24-bit absolute address in program memory specified in the instruction’s 24-bit extension word if the n^{th} bit of the source operand S is clear. The bit to be tested is selected by an immediate bit number from 0–23. If the n^{th} bit of the source operand S is clear, the address of the instruction immediately following the JSCLR instruction (PC) and the system status register (SR) are pushed onto the system stack. Program execution then continues at the specified absolute address in the instruction’s 24-bit extension word. If the specified memory bit is not clear, the program counter (PC) is incremented and the extension word is ignored. However, the address register specified in the effective address field is always updated independently of the state of the n^{th} bit. All address register indirect addressing modes may be used to reference the source operand S. Absolute short and I/O short addressing modes may also be used·

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	×	×	×	×
CCR							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

JSCLR #n,[X or Y]:ea,xxxx

23							16	15							8	7							0
0	0	0	0	1	0	1	1	0	1	M	M	M	R	R	R	1	S	0	b	b	b	b	b
ABSOLUTE ADDRESS EXTENSION																							

JSCLR #n,[X or Y]:aa,xxxx

23							16	15							8	7							0
0	0	0	0	1	0	1	1	0	0	a	a	a	a	a	a	1	S	0	b	b	b	b	b
ABSOLUTE ADDRESS EXTENSION																							

JSCLR #n,[X or Y]:pp,xxxx

23							16	15							8	7							0
0	0	0	0	1	0	1	1	1	0	p	p	p	p	p	p	1	S	0	b	b	b	b	b
ABSOLUTE ADDRESS EXTENSION																							

JSCLR #n,[X or Y]:qq,xxxx

23							16	15							8	7							0
0	0	0	0	0	0	0	1	1	1	q	q	q	q	q	q	1	S	0	b	b	b	b	b
ABSOLUTE ADDRESS EXTENSION																							

JSCLR #n,S,xxxx

23							16	15							8	7							0
0	0	0	0	1	0	1	1	1	1	D	D	D	D	D	D	0	0	0	b	b	b	b	b
ABSOLUTE ADDRESS EXTENSION																							

Instruction Fields:

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address (see Table A-19 on page A-242)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{xxxx}		24-bit absolute Address extension word
{aa}	**aaaaaa**	Absolute Address [0-63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0-$FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80-$FFFFBF]
{S}	**DDDDDD**	Source register [all on-chip registers] (see Table A-22 on page A-243)

A-6.51 Jump if Bit Set (JSET)

JSET JSET

Jump if Bit Set

Operation:						**Assembler Syntax:**	
If	S{n}=1	then	xxxx	→	PC	JSET	#n,[X or Y]:ea,xxxx
		else	PC+ 1	→	PC		
If	S{n}=1	then	xxxx	→	PC	JSET	#n,[X or Y],aa,xxxx
		else	PC+ 1	→	PC		
If	S{n}=1	then	xxxx	→	PC	JSET	#n,[X or Y]:pp,xxxx
		else	PC+ 1	→	PC		
If	S{n}=1	then	xxxx	→	PC	JSET	#n,[X or Y]:qq,xxxx
		else	PC+ 1	→	PC		
If	S{n}=1	then	xxxx	→	PC	JSET	#n,S,xxxx
		else	PC+ 1	→	PC		

Description: Jump to the 24-bit absolute address in program memory specified in the instruction's 24-bit extension word if the nth bit of the source operand S is set. The bit to be tested is selected by an immediate bit number from 0–23. If the specified memory bit is not set, the program counter (PC) is incremented, and the absolute address in the extension word is ignored. However, the address register specified in the effective address field is always updated independently of the state of the nth bit. All address register indirect addressing modes may be used to reference the source operand S. Absolute short and I/O short addressing modes may also be used.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	×	×	×	×
CCR							

✔	This bit is changed according to the standard definition
×	This bit is unchanged by the instruction

Instruction Formats and opcodes:

JSET #n,[X or Y]:ea,xxxx

23 16	15 8	7 0
0 0 0 0 1 0 1 0	0 1 M M M R R R	1 S 1 b b b b b
ABSOLUTE ADDRESS EXTENSION		

JSET #n,[X or Y]:aa,xxxx

23 16	15 8	7 0
0 0 0 0 1 0 1 0	0 0 a a a a a a	1 S 1 b b b b b
ABSOLUTE ADDRESS EXTENSION		

JSET #n,[X or Y]:pp,xxxx

23 16	15 8	7 0
0 0 0 0 1 0 1 0	1 0 p p p p p p	1 S 1 b b b b b
ABSOLUTE ADDRESS EXTENSION		

JSET #n,[X or Y]:qq,xxxx

23 16	15 8	7 0
0 0 0 0 0 0 0 1	1 0 q q q q q q	1 S 1 b b b b b
ABSOLUTE ADDRESS EXTENSION		

JSET #n,S,xxxx

23 16	15 8	7 0
0 0 0 0 1 0 1 0	1 1 D D D D D D	0 0 1 b b b b b
ABSOLUTE ADDRESS EXTENSION		

Instruction Fields:

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address (see Table A-19 on page A-242)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{xxxx}		24-bit Absolute Address in extension word
{aa}	**aaaaaa**	Absolute Address [0-63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0-$FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80-$FFFFBF]
{S}	**DDDDDD**	Source register [all on-chip registers] (see Table A-22 on page A-243)

A-6.52 Jump to Subroutine (JSR)

JSR JSR

Jump to Subroutine

Operation:	**Assembler Syntax:**
SP+1→SP; PC→SSH; SR→SSL; 0xxx→PC	JSR xxx
SP+1→SP; PC→SSH; SR→SSL; ea→PC	JSR ea

Description: Jump to the subroutine whose location in program memory is given by the instruction’s effective address. The address of the instruction immediately following the JSR instruction (PC) and the system status register (SR) is pushed onto the system stack. Program execution then continues at the specified effective address in program memory. All memory alterable addressing modes may be used for the effective address. A fast short jump addressing mode may also be used. The 12-bit data is zero extended to form the effective address.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

JSR ea

23 16	15 8	7 0
0 0 0 0 1 0 1 1	1 1 M M M R R R	1 0 0 0 0 0 0 0
OPTIONAL EFFECTIVE ADDRESS EXTENSION		

JSR xxx

23 16	15 8	7 0
0 0 0 0 1 1 0 1	0 0 0 0 a a a a	a a a a a a a a

Instruction Fields:

{xxx}	**aaaaaaaaaaaa**	Short Jump Address
{ea}	**MMMRRR**	Effective Address (see Table A-18 on page A-241)

A-6.53 Jump to Subroutine if Bit Set (JSSET)

JSSET JSSET

Jump to Subroutine if Bit Set

Operation:	Assembler Syntax:

If S{n}=1 then SP+1→SP;PC →SSH;SR →SSL;
;xxxx →PC
else PC+1→PC
	JSSET #n,[X or Y]:ea,xxxx

If S{n}=1 then SP+1→SP;PC →SSH;SR →SSL;
;xxxx →PC
else PC+1→PC
	JSSET #n,[X or Y],aa,xxxx

If S{n}=1 then SP+1→SP;PC →SSH;SR →SSL;
;xxxx →PC
else PC+1→PC
	JSSET #n,[X or Y]:pp,xxxx

If S{n}=1 then SP+1→SP;PC →SSH;SR →SSL;
;xxxx →PC
else PC+1→PC
	JSSET #n,[X or Y]:qq,xxxx

If S{n}=1 then SP+1→SP;PC →SSH;SR →SSL;
;xxxx →PC
else PC+1→PC
	JSSET #n,S,xxxx

Description: Jump to the subroutine at the 24-bit absolute address in program memory specified in the instruction's 24-bit extension word if the nth bit of the source operand S is set. The bit to be tested is selected by an immediate bit number from 0–23. If the nth bit of the source operand S is set, the address of the instruction immediately following the JSSET instruction (PC) and the system status register (SR) are pushed onto the system stack. Program execution then continues at the specified absolute address in the instruction's 24-bit extension word. If the specified memory bit is not set, the program counter (PC) is incremented, and the extension word is ignored. However, the address register specified in the effective address field is always updated independently of the state of the nth bit. All address register indirect addressing modes may be used to reference the source operand S. Absolute short and I/O short addressing modes may also be used.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	×	×	×	×
CCR							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

JSSET #n,[X or Y]:ea,xxxx

23							16	15							8	7							0
0	0	0	0	1	0	1	1	0	1	M	M	M	R	R	R	1	S	1	b	b	b	b	b
ABSOLUTE ADDRESS EXTENSION																							

JSSET #n,[X or Y]:aa,xxxx

23							16	15							8	7							0
0	0	0	0	1	0	1	1	0	0	a	a	a	a	a	a	1	S	1	b	b	b	b	b
ABSOLUTE ADDRESS EXTENSION																							

JSSET #n,[X or Y]:pp,xxxx

23							16	15							8	7							0
0	0	0	0	1	0	1	1	1	0	p	p	p	p	p	p	1	S	1	b	b	b	b	b
ABSOLUTE ADDRESS EXTENSION																							

JSSET #n,[X or Y]:qq,xxxx

23							16	15							8	7							0
0	0	0	0	0	0	0	1	1	1	q	q	q	q	q	q	1	S	1	b	b	b	b	b
ABSOLUTE ADDRESS EXTENSION																							

JSSET #n,S,xxxx

23							16	15							8	7							0
0	0	0	0	1	0	1	1	1	1	D	D	D	D	D	D	0	0	1	b	b	b	b	b
ABSOLUTE ADDRESS EXTENSION																							

Instruction Fields:

{#n}	**bbbbb**	Bit number [0-23]
{ea}	**MMMRRR**	Effective Address (see Table A-19 on page A-242)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{xxxx}		24-bit PC absolute Address extension word
{aa}	**aaaaaa**	Absolute Address [0-63]
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0-$FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80-$FFFFBF]
{S}	**DDDDDD**	Source register [all on-chip registers] (see Table A-22 on page A-243)

A-6.54 Load PC Relative Address (LRA)

LRA LRA

Load PC Relative Address

Operation:	Assembler Syntax:
PC+Rn➞D	LRA Rn,D
PC+xxxx➞D	LRA xxxx,D

Description: The PC is added to the specified displacement and the result is stored in destination D. The displacement is a 2’s complement 24-bit integer that represents the relative distance from the current PC to the destination PC. Long Displacement and Address Register PC Relative addressing modes may be used. Note that if D is SSH, the SP will be preincremented by one.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcode:

LRA Rn,D

23							16	15							8	7							0
0	0	0	0	0	1	0	0	1	1	0	0	0	R	R	R	0	0	0	d	d	d	d	d

LRA xxxx,D

23							16	15							8	7							0
0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	d	d	d	d	d
LONG DISPLACEMENT																							

Instruction Fields:

{Rn} **RRR** Address register [R0-R7]

{D} **ddddd** Destination address register [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A,B,R0-R7,N0-N7] (see Table A-31 on page A-245)

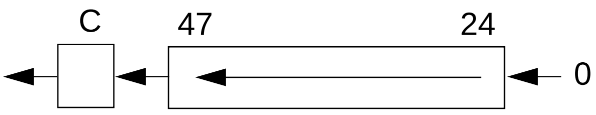
{xxxx} 24-bit PC Long Displacement

A-6.55 Logical Shift Left (LSL)

LSL LSL

Logical Shift Left

Operation:



Assembler Syntax:

LSL D (parallel move)

LSL #ii,D

LSL S,D

Description:

Single-bit shift:

Logically shift bits 47–24 of the destination operand D one bit to the left and store the result in the destination accumulator. Prior to instruction execution, bit 47 of D is shifted into the carry bit C, and a zero is shifted into bit 24 of the destination accumulator D.

Multi-bit shift:

The contents of bits 47-24 of the destination accumulator D are shifted left #ii bits. Bits shifted out of position 47 are lost, but for the last bit which is latched in the carry bit. Zeros are supplied to the vacated positions on the right. The result is placed into bits 47-24 of the destination accumulator D. The number of bits to shift is determined by the 5-bit immediate field in the instruction, or by the unsigned integer located in the control register S. If a zero shift count is specified, the carry bit is cleared.

This is a 24 bit operation. The remaining bits of the destination accumulator are not affected.

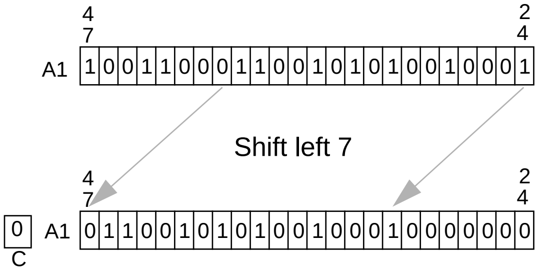
Note: The number of shifts should not exceed the value of 24.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	●	●	●	●
CCR							

- ● N Set if bit 47 of the result is set
- ● Z Set if bits 47-24 of the result are zero
- ● V Always cleared
- ● C Set if the last bit shifted out of the operand is set. Cleared otherwise.Cleared for a shift count of zero.
- × This bit is unchanged by the instruction

Example: LSL #7, A



47
24
A1 1 0 0 1 1 0 0 0 1 1 0 0 1 0 1 0 1 0 0 1 0 0 0 1
Shift left 7
47
24
0
C
A1 0 1 1 0 0 1 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0

Instruction Formats and opcodes:

LSL D

23 … 8	7							0
DATA BUS MOVE FIELD	0	0	1	1	D	0	1	1
OPTIONAL EFFECTIVE ADDRESS EXTENSION								

LSL #ii,D

23							16	15							8	7							0
0	0	0	0	1	1	0	0	0	0	0	1	1	1	1	0	1	0	i	i	i	i	i	D

LSL S,D

23							16	15							8	7							0
0	0	0	0	1	1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	s	s	s	D

Instruction Fields:

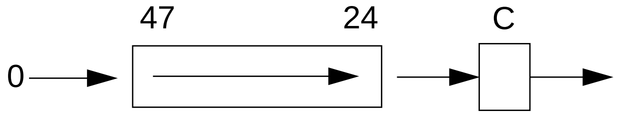
{D}	**D**	Destination accumulator [A,B] (see Table A-10 on page A-239)
{S}	**sss**	Control register [X0,X1,Y0,Y1,A1,B1] (see Table A-15 on page A-240)
{#ii}	**iiiii**	5bit unsigned integer [0-23] denoting the shift amount

A-6.56 Logical Shift Right (LSR)

LSR LSR

Logical Shift Right

Operation:



Assembler Syntax:

LSR D (parallel move)

LSR #ii,D

LSR S,D

Description:

Single-bit shift:

Logically shift bits 47–24 of the destination operand D one bit to the right and store the result in the destination accumulator. Prior to instruction execution, bit 24 of D is shifted into the carry bit C, and a zero is shifted into bit 47 of the destination accumulator D.

Multi-bit shift:

The contents of bits 47-24 of the destination accumulator D are shifted right #ii bits. Bits shifted out of position 24 are lost, but for the last bit which is latched in the carry bit. Zeros are supplied to the vacated positions on the left. The result is placed into bits 47-24 of the destination accumulator D. The number of bits to shift is determined by the 5-bit immediate field in the instruction, or by the unsigned integer located in the control register S. If a zero shift count is specified, the carry bit is cleared.

This is a 24 bit operation. The remaining bits of the destination register are not affected.

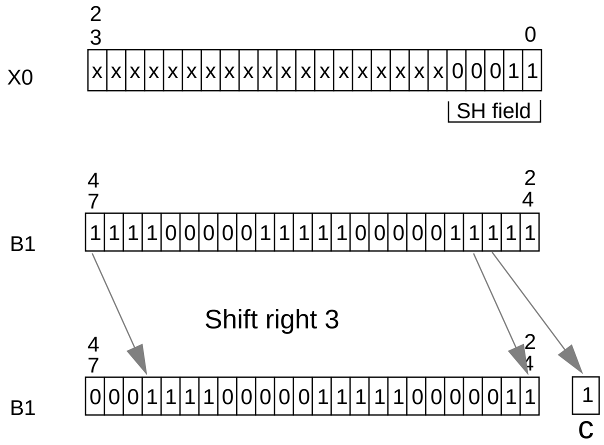
Note: The number of shifts should not exceed the value of 24.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	●	●	●	●
CCR							

- ● N Set if bit 47 of the result is set
- ● Z Set if bits 47-24 of the result are zero
- ● V Always cleared
- ● C Set if the last bit shifted out of the operand is set. Cleared otherwise. Cleared for a shift count of zero
- × This bit is unchanged by the instruction

Example: LSR X0,B



Instruction Formats and opcodes:

LSR D

23 8	7							0
DATA BUS MOVE FIELD	0	0	1	0	D	0	1	1
OPTIONAL EFFECTIVE ADDRESS EXTENSION								

LSR #ii,D

23							16	15							8	7							0
0	0	0	0	1	1	0	0	0	0	0	1	1	1	1	0	1	1	i	i	i	i	i	D

LSR S,D

23							16	15							8	7							0
0	0	0	0	1	1	0	0	0	0	0	1	1	1	1	0	0	0	1	1	s	s	s	D

Instruction Fields:

{D}	**D**	Destination accumulator [A,B] (see Table A-10 on page A-239)
{S}	**sss**	Control register [X0,X1,Y0,Y1,A1,B1] (see Table A-15 on page A-240)
{#ii}	**iiiii**	5 bit unsigned integer [0-23] denoting the shift amount

A-6.57 Load Updated Address (LUA)

LUA LUA

Load Updated address

Operation:	**Assembler Syntax:**
ea→D (No update performed)	LUA ea,D
Rn+aa→D	LUA (Rn+aa),D
ea→D (No update performed)	LEA ea,D
Rn+aa→D	LEA (Rn+aa),D

Description: Load the updated address into the destination address register D. The source address register and the update mode used to compute the updated address are specified by the effective address (ea). **Note that the source address register specified in the effective address is not updated. This is the only case where an address register is not updated although stated otherwise in the effective address mode bits.** Only the following addressing modes may be used: Post+N, Post-N, Post+1, Post-1.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcode:

		23							16	15							8	7							0
LUA/ LEA	ea,D	0	0	0	0	0	1	0	0	0	1	0	M	M	R	R	R	0	0	0	d	d	d	d	d

		23							16	15							8	7							0
LUA/ LEA	(Rn+aa),D	0	0	0	0	0	1	0	0	0	0	a	a	a	R	R	R	a	a	a	a	d	d	d	d

Note: LEA is a synonym for LUA. The simulator on-line disassembly will translate the opcodes into LUA.

Instruction Fields:

{ea}	**MMRRR**	Effective address (see Table A-20 on page A-242)
{D}	**ddddd**	Destination address register [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A,B,R0-R7,N0-N7] (see Table A-31 on page A-245)
{D}	**dddd**	Destination address register [R0-R7,N0-N7] (see Table A-25 on page A-244)
{aa}	**aaaaaaa**	7-bit sign extended short displacement address
{Rn}	**RRR**	Source address register [R0-R7]

Note: **RRR** refers to a **source** address register (R0-R7), while **dddd/ddddd** refer to a **destination** address register R0-R7 or N0-N7.

A-6.58 Signed Multiply-Accumulate (MAC)

MAC MAC

Signed Multiply Accumulate

Operation:	**Assembler Syntax:**
D±S1∗S2→D (parallel move)	MAC (±)S1,S2,D (parallel move)
D±S1∗S2→D (parallel move)	MAC (±)S2,S1,D (parallel move)
D±(S1∗2^{-n})→D (**no** parallel move)	MAC (±)S,#n,D (**no** parallel move)

Description: Multiply the two signed 24-bit source operands S1 and S2 (**or** the signed 24-bit source operand S by the positive 24-bit immediate operand 2^{-n}) and add/subtract the product to/from the specified 56-bit destination accumulator D. The "–" sign option is used to negate the specified product prior to accumulation. The default sign option is "+".

Note: When the processor is in the Double Precision Multiply Mode, the following instructions do not execute in the normal way and should only be used as part of the double precision multiply algorithm:

MAC X1, Y0, A	MAC X1, Y0, B
MAC X0, Y1, A	MAC X0, Y1, B
MAC Y1, X1, A	MAC Y1, X1, B

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	✔	✔	✔	✔	✔	X
CCR							

✔	This bit is changed according to the standard definition
×	This bit is unchanged by the instruction

Instruction Formats and opcodes 1:

	23 16	15 8	7 0
MAC (±)S1,S2,D	DATA BUS MOVE FIELD		1 Q Q Q d k 1 0
MAC (±)S2,S1,D	OPTIONAL EFFECTIVE ADDRESS EXTENSION		

Instruction Fields:

{S1,S2} **QQQ** Source registers S1,S2
[X0*X0,Y0*Y0,X1*X0,Y1*Y0,X0*Y1,Y0*X0,X1*Y0,Y1*X1]
(see Table A-26 on page A-244)

{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

{±} **k** Sign [+,-] (see Table A-29 on page A-244)

Instruction Formats and opcode 2:

	23 16	15 8	7 0
MAC (±)S,#n,D	0 0 0 0 0 0 0 1	0 0 0 s s s s s	1 1 Q Q d k 1 0

Instruction Fields:

{S} **QQ** Source register [Y1,X0,Y0,X1]] (see Table A-27 on page A-244)

{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

{±} **k** Sign [+,-] (see Table A-29 on page A-244)

{#n} **sssss** Immediate operand (see Table A-32 on page A-246)

A-6.59 Signed MAC with Immediate Operand (MACI)

MACI MACI

Signed Multiply-Accumulate with Immediate Operand

Operation:	**Assembler Syntax:**
D±#xxxxxx∗S→D	MACI (±)#xxxxxx,S,D

Description: Multiply the two signed 24-bit source operands #xxxxxx and S and add/subtract the product to/from the specified 56-bit destination accumulator D. The "–" sign option is used to negate the specified product prior to accumulation. The default sign option is "+".

condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	✔	✔	✔	✔	✔	✔	×
CCR							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcode:

	23							16	15							8	7							0
MACI (±)#xxxxxx,S,D	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	1	q	q	d	k	1	0
	IMMEDIATE DATA EXTENSION																							

Instruction Fields:

{S}	**qq**	Source register [X0,Y0,X1,Y1] (see Table A-28 on page A-244)
{D}	**d**	Destination accumulator [A,B] (see Table A-10 on page A-239)
{±}	**k**	Sign [+,-] (see Table A-29 on page A-244)
#xxxxxx		24-bit Immediate Long Data extension word

A-6.60 Mixed Multiply-Accumulate (MAC su/uu)

MAC(su,uu) MAC(su,uu)

Mixed Multiply Accumulate

Operation:	**Assembler Syntax:**	
D±S1∗S2→D (S1 unsigned, S2 unsigned)	MACuu	(±)S1,S2,D (no parallel move)
D±S1∗S2→D (S1 signed, S2 unsigned)	MACsu	(±)S2,S1,D (no parallel move)

Description: Multiply the two 24-bit source operands S1 and S2 and add/subtract the product to/from the specified 56-bit destination accumulator D. One or two of the source operands can be unsigned. The "–" sign option is used to negate the specified product prior to accumulation. The default sign option is "+".

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	✔	✔	✔	✔	✔	✔	×
CCR							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

MACsu (±)S1,S2,D
MACuu (±)S1,S2,D

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	0	1	0	0	1	1	0	1	s	d	k	Q	Q	Q	Q

Instruction Fields:

{S1,S2} **QQQQ** Source registers S1,S2 [all combinations of X0,X1,Y0 and Y1] (see Table A-30 on page A-245)

{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

{±} **k** Sign [+,-] (see Table A-29 on page A-244)

{s} [ss,us] (see Table A-40 on page A-249)

A-6.61 Signed MAC and Round (MACR)

MACR MACR

Signed Multiply Accumulate and Round

Operation:	**Assembler Syntax:**
$D\pm S1*S2+r\rightarrow D$ (parallel move)	MACR (±)S1,S2,D (parallel move)
$D\pm S1*S2+r\rightarrow D$ (parallel move)	MACR (±)S2,S1,D (parallel move)
$D\pm(S1*2^{-n})+r\rightarrow D$ (**no** parallel move)	MACR (±)S,#n,D (**no** parallel move)

Description: Multiply the two signed 24-bit source operands S1 and S2 (**or** the signed 24-bit source operand S by the positive 24-bit immediate operand 2^{-n}), add/subtract the product to/from the specified 56-bit destination accumulator D, and then round the result using either convergent or two's complement rounding. The rounded result is stored in the destination accumulator D.

The "–" sign option negates the specified product prior to accumulation. The default sign option is "+".

The contribution of the LS bits of the result is rounded into the upper portion of the destination accumulator. Once rounding has been completed, the LS bits of the destination accumulator D are loaded with zeros to maintain an unbiased accumulator value which may be reused by the next instruction. The upper portion of the accumulator contains the rounded result which may be read out to the data buses. Refer to the RND instruction for more complete information on the rounding process.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	✔	✔	✔	✔	✔	×
CCR							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes 1:

MACR (±)S1,S2,D
MACR (±)S2,S1,D

23 16 15 8	7 0
DATA BUS MOVE FIELD	1 Q Q Q d k 1 1
OPTIONAL EFFECTIVE ADDRESS EXTENSION	

Instruction Fields:

{S1,S2}	**QQQ**	Source registers S1,S2 [X0*X0,Y0*Y0,X1*X0,Y1*Y0,X0*Y1,Y0*X0,X1*Y0,Y1*X1] (see Table A-26 on page A-244)
{D}	**d**	Destination accumulator [A,B] (see Table A-10 on page A-239)
{±}	**k**	Sign [+,-] (see Table A-29 on page A-244)

Instruction Formats and opcode 2:

MACR (±)S,#n,D

23 16	15 8	7 0
0 0 0 0 0 0 0 1	0 0 0 s s s s s	1 1 Q Q d k 1 1

Instruction Fields:

{S}	**QQ**	Source register [Y1,X0,Y0,X1]] (see Table A-27 on page A-244)
{D}	**d**	Destination accumulator [A,B] (see Table A-10 on page A-239)
{±}	**k**	Sign [+,-] (see Table A-29 on page A-244)
{#n}	**sssss**	Immediate operand (see Table A-32 on page A-246)

A-6.62 Signed MAC and Round with Immediate Operand (MACRI)

MACRI MACRI

Signed Multiply-Accumulate and Round with Immediate Operand

Operation:

D±#xxxxxx∗S→D

Assembler Syntax:

MACRI (±)#xxxxxx,S,D

Description: Multiply the two signed 24-bit source operands #xxxxxx and S, add/subtract the product to/from the specified 56-bit destination accumulator D, and then round the result using either convergent or two's complement rounding. The rounded result is stored in the destination accumulator D.

The "–" sign option negates the specified product prior to accumulation. The default sign option is "+".

The contribution of the LS bits of the result is rounded into the upper portion of the destination accumulator. Once rounding has been completed, the LS bits of the destination accumulator D are loaded with zeros to maintain an unbiased accumulator value which may be reused by the next instruction. The upper portion of the accumulator contains the rounded result which may be read out to the data buses. Refer to the RND instruction for more complete information on the rounding process.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	✔	✔	✔	✔	✔	✔	×
CCR							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcode:

	23							16	15							8	7							0
MACRI (±)#xxxxxx,S,D	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	1	q	q	d	k	1	1
	IMMEDIATE DATA EXTENSION																							

Instruction Fields:

{S}	**qq**	Source register [X0,Y0,X1,Y1] (see Table A-28 on page A-244)
{D}	**d**	Destination accumulator [A,B] (see Table A-10 on page A-239)
{±}	**k**	Sign [+,-] (see Table A-29 on page A-244)
#xxxxxx		24-bit Immediate Long Data extension word

A-6.63 Transfer by Signed Value (MAX)

MAX MAX

Transfer by Signed Value

Operation:

If B – A ≤ 0 then A ➞ B

Assembler Syntax:

MAX A,B (parallel move)

Description: Subtract the signed value of the source accumulator from the signed value of the destination accumulator. If the difference is negative or zero (i.e. A ≥ B) then transfer the source accumulator to destination accumulator, otherwise do not change destination accumulator.

This is a 56 bit operation.

Note: The Carry condition code signifies that a transfer has been performed.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	×	×	×	●
CCR							

● C Cleared if the conditional transfer was performed. Set otherwise.
✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23 16 15 8	7	0
MAX A, B	DATA BUS MOVE FIELD	0 0 0 1	1 1 0 1
	OPTIONAL EFFECTIVE ADDRESS EXTENSION		

A-6.64 Transfer by Magnitude (MAXM)

MAXM MAXM

Transfer by Magnitude

Operation:

If |B| – |A| ≤ 0 then A ➞ B

Assembler Syntax:

MAXM A,B (parallel move)

Description: Subtract the absolute value (magnitude) of the source accumulator from the absolute value of the destination accumulator. If the difference is negative or zero (i.e. |A| ≥ |B|) then transfer the source accumulator to destination accumulator, otherwise do not change destination accumulator.

This is a 56 bit operation.

Note: The Carry condition code signifies that a transfer has been performed.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	×	×	×	●
CCR							

● C Cleared if the conditional transfer was performed. Set otherwise.
✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Instruction Formats and opcodes:

MAXM A, B

23 16 15 8	7	0
DATA BUS MOVE FIELD	0 0 0 1	0 1 0 1
OPTIONAL EFFECTIVE ADDRESS EXTENSION		

A-6.65 Merge Two Half Words (MERGE)

MERGE MERGE

Merge Two Half Words

Operation:

{S[11:0],D[35:24]} → D[47:24]

Assembler Syntax:

MERGE S,D

Description: The contents of bits 11-0 of the source register are concatenated to the contents of bits 35-24 of the destination accumulator. The result is stored in the destination accumulator. This instruction is a 24-bit operation. The remaining bits of the destination accumulator D are not affected.

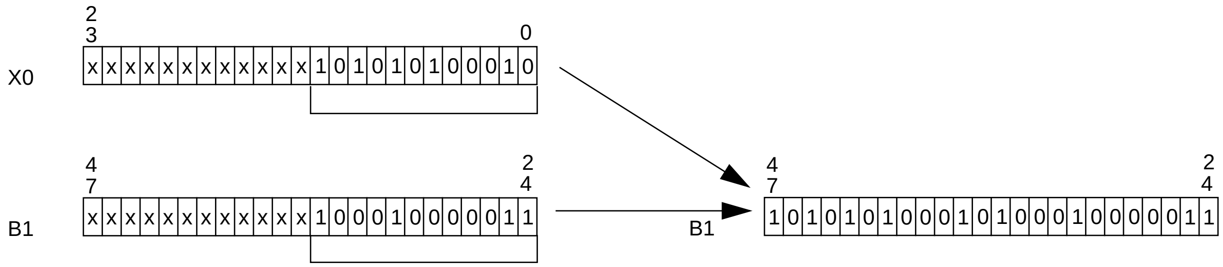
Notes:
1) This instruction may be used in conjunction with EXTRACT or INSERT instructions to concatenate width and offset fields into a control word.
2) In 16 bit arithmetic mode the contents of bits 15-8 of the source register are concatenated to the contents of bits 39-32 of the destination accumulator. The result is placed in bits 47-32 of the destination accumulator.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	●	●	●	×
CCR							

- N Set if bit 47 of the result is set
- Z Set if bits 47-24 of the result are zero
- V Always cleared

Example: MERGE X0,B


2
3
0
X0
x x x x x x x x x x x x 1 0 1 0 1 0 1 0 0 0 1 0
4
7
2
4
B1
x x x x x x x x x x x x 1 0 0 0 1 0 0 0 0 0 1 1
4
7
2
4
B1
1 0 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1 1

Instruction Formats and opcodes:

		23							16	15							8	7							0
MERGE	S,D	0	0	0	0	1	1	0	0	0	0	0	1	1	0	1	1	1	0	0	0	S	S	S	D

Instruction Fields:

{D} **D** Destination accumulator [A,B] (see Table A-10 on page A-239)
{S} **SSS** Source register [X0,X1,Y0,Y1,A1,B1] (see Table A-15 on page A-240)

A-6.66 Move Data (MOVE)

MOVE MOVE

Move Data

Operation: S→D

Assembler Syntax: MOVE S,D

Description: Move the contents of the specified data source S to the specified destination D. This instruction is equivalent to a data ALU NOP with a parallel data move.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	×	×	×	×
CCR							

✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23 16 15 8	7							0
MOVE S,D	DATA BUS MOVE FIELD	0	0	0	0	0	0	0	0
	OPTIONAL EFFECTIVE ADDRESS EXTENSION								

Instruction Fields:

See **Parallel Move Descriptions** for data bus move field encoding.

Parallel Move Descriptions: Thirty of the sixty-two 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 the instruction cycle. Ten types of parallel moves are permitted, including register to register moves, register to memory moves, and memory to register moves. However, not all addressing modes are allowed for each type of memory reference. The following section contains detailed descriptions about each type of parallel move operation.

A-6.67 NO Parallel Data Move

No Parallel Data Move

Operation:	**Assembler Syntax:**
(. . . . . .)	(. . . . . .)

where (. . . . .) refers to any arithmetic or logical instruction which allows parallel moves.

Description: Many instructions in the instruction set allow parallel moves. The parallel moves have been divided into 10 opcode categories. This category is a parallel move NOP and does not involve data bus move activity.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23							16	15							8	7 0
(. . . . . .)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	INSTRUCTION OPCODE

Instruction Format:

(defined by instruction)

A-6.68 Immediate Short Data Move (I)

Immediate Short Data Move

Operation: (. . . . .), #xx→D

Assembler Syntax: (. . . . .) #xx,D

where (. . . . .) refers to any arithmetic or logical instruction which allows parallel moves.

Description: Move the 8-bit immediate data value (#xx) into the destination operand D.

If the destination register D is A0, A1, A2, B0, B1, B2, R0–R7, or N0–N7, the 8-bit immediate short operand is interpreted as an **unsigned integer** and is stored in the specified destination register. That is, the 8-bit data is stored in the eight LS bits of the destination operand, and the remaining bits of the destination operand D are zeroed.

If the destination register D is X0, X1, Y0, Y1, A, or B, the 8-bit immediate short operand is interpreted as a **signed fraction** and is stored in the specified destination register. That is, the 8-bit data is stored in the eight MS bits of the destination operand, and the remaining bits of the destination operand D are zeroed.

If the arithmetic or logical opcode-operand portion of the instruction specifies a given destination accumulator, that same accumulator or portion of that accumulator may not be specified as a destination D in the parallel data bus move operation. Thus, if the opcode-operand portion of the instruction specifies the 56-bit A accumulator as its destination, the parallel data bus move portion of the instruction may not specify A0, A1, A2, or A as its destination D. Similarly, if the opcode-operand portion of the instruction specifies the 56-bit B accumulator as its destination, the parallel data bus move portion of the instruction may not specify B0, B1, B2, or B as its destination D. That is, **duplicate destinations are NOT allowed within the same instruction.**

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23							16	15							8	7 0
(. . . . .) #xx,D	0	0	1	d	d	d	d	d	i	i	i	i	i	i	i	i	INSTRUCTION OPCODE

Instruction Fields:

{#xx} **iiiiiiii** 8-bit Immediate Short Data

{D} **ddddd** Destination register [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A,B,R0-R7,N0-N7] (see Table A-31 on page A-245)

A-6.69 Register to Register Data Move (R)

R **R**

Register to Register Data Move

Operation:	**Assembler Syntax:**
(. . . . .); S→D	(. . . . .) S,D

where (. . . . .) refers to any arithmetic or logical instruction which allows parallel moves.

Description: Move the source register S to the destination register D.

If the arithmetic or logical opcode-operand portion of the instruction specifies a given destination accumulator, that same accumulator or portion of that accumulator may not be specified as a destination D in the parallel data bus move operation. Thus, if the opcode-operand portion of the instruction specifies the 56-bit A accumulator as its destination, the parallel data bus move portion of the instruction may not specify A0, A1, A2, or A as its destination D. Similarly, if the opcode-operand portion of the instruction specifies the 56-bit B accumulator as its destination, the parallel data bus move portion of the instruction may not specify B0, B1, B2, or B as its destination D. That is, **duplicate destinations are NOT allowed within the same instruction**.

If the opcode-operand portion of the instruction specifies a given source or destination register, that same register or portion of that register may be used as a source S in the parallel data bus move operation. This allows data to be moved in the same instruction in which it is being used as a source operand by a data ALU operation. That is, **duplicate sources are allowed within the same instruction**.

Note: The MOVE A,B operation will result in a 24-bit positive or negative saturation constant being stored in the B1 portion of the B accumulator if the signed integer portion of the A accumulator is in use.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	×	×	×	×
CCR							

✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23							16	15							8	7 0
(. . . . .) S,D	0	0	1	0	0	0	e	e	e	e	e	d	d	d	d	d	INSTRUCTION OPCODE

Instruction Fields:

{S} **eeeee** Source register [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A,B,R0-R7,N0-N7] (see Table A-31 on page A-245)

{D} **ddddd** Destination register [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A,B,R0-R7,N0-N7] (see Table A-31 on page A-245)

A-6.70 Address Register Update (U)

U U

Address Register Update

Operation:	**Assembler Syntax:**
(. . . . .); ea→Rn	(. . . . .) ea

where (. . . . .) refers to any arithmetic or logical instruction which allows parallel moves.

Description: Update the specified address register according to the specified effective addressing mode. All update addressing modes may be used.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23							16	15							8	7	0
(. . . . .) ea	0	0	1	0	0	0	0	0	0	1	0	M	M	R	R	R	INSTRUCTION OPCODE	

Instruction Fields:

{ea} **MMRRR** Effective Address (see Table A-20 on page A-242)

A-6.71 X Memory Data Move (X:)

X: X:

X Memory Data Move

Operation:	Assembler Syntax:
(. . . . .); X:ea→D	(. . . . .) X:ea,D
(. . . . .); X:aa→D	(. . . . .) X:aa,D
(. . . . .); S→X:ea	(. . . . .) S,X:ea
(. . . . .); S→X:aa	(. . . . .) S,X:aa
X:(Rn+xxx)→D	MOVE X:(Rn+xxx),D
X:(Rn+xxxx)→D	MOVE X:(Rn+xxxx),D
D→X:(Rn+xxx)	MOVE D,X:(Rn+xxx)
D→X:(Rn+xxxx)	MOVE D,X:(Rn+xxxx)

where (. . . . .) refers to any arithmetic or logical instruction which allows parallel moves.

Description: Move the specified word operand from/to X memory. All memory addressing modes, including absolute addressing and 24-bit immediate data, may be used. Absolute short addressing may also be used.

If the arithmetic or logical opcode-operand portion of the instruction specifies a given destination accumulator, that same accumulator or portion of that accumulator may not be specified as a destination D in the parallel data bus move operation. Thus, if the opcode-operand portion of the instruction specifies the 56-bit A accumulator as its destination, the parallel data bus move portion of the instruction may not specify A0, A1, A2, or A as its destination D. Similarly, if the opcode-operand portion of the instruction specifies the 56-bit B accumulator as its destination, the parallel data bus move portion of the instruction may not specify B0, B1, B2, or B as its destination D. That is, **duplicate destinations are NOT allowed within the same instruction**.

If the opcode-operand portion of the instruction specifies a given source or destination register, that same register or portion of that register may be used as a source S in the parallel data bus move operation. This allows data to be moved in the same instruction in which it is being used as a source operand by a data ALU operation. That is, **duplicate sources are allowed within the same instruction**.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	×	×	×	×
CCR							

✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Note: The MOVE A,X:ea operation will result in a 24-bit positive or negative saturation constant being stored in the specified 24-bit X memory location if the signed integer portion of the A accumulator is in use.

Instruction Formats and opcodes 1:

(. . . . .)X:ea,D
(. . . . .)S,X:ea
(. . . . .)#xxxxxx,D

23 16	15 8	7 0
0 1 d d 0 d d d	W 1 M M M R R R	INSTRUCTION OPCODE
OPTIONAL EFFECTIVE ADDRESS EXTENSION		

(. . . . .)X:aa,D
(. . . . .)S,X:aa

23 16	15 8	7 0
0 1 d d 0 d d d	W 0 a a a a a a	INSTRUCTION OPCODE

Instruction Fields:

{ea}	**MMMRRR**	Effective Address (see Table A-16 on page A-241)
	W	Read S / Write D bit (see Table A-33 on page A-246)
{S,D}	**ddddd**	Source/Destination registers [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A,B,R0-R7,N0-N7] (see Table A-31 on page A-245)
{aa}	**aaaaaa**	6-bit Absolute Short Address

Instruction Formats and opcodes 2:

MOVE X:(Rn+xxxx),D
MOVE S,X:(Rn+xxxx)

23							16	15							8	7							0
0	0	0	0	1	0	1	0	0	1	1	1	0	R	R	R	1	W	D	D	D	D	D	D
Rn RELATIVE DISPLACEMENT																							

MOVE X:(Rn+xxx),D
MOVE S,X:(Rn+xxx)

23							16	15							8	7							0
0	0	0	0	0	0	1	a	a	a	a	a	a	R	R	R	1	a	0	W	D	D	D	D

Instruction Fields:

	W	Read S / Write D bit (see Table A-33 on page A-246)
{xxx}	**aaaaaaa**	7-bit sign extended Short Displacement Address
{Rn}	**RRR**	Address register (R0-R7)
{D}	**DDDD**	Source/Destination registers [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A,B] (see Table A-34 on page A-246)
{S,D}	**DDDDDD**	Source/Destination registers [all on-chip registers] (see Table A-22 on page A-243)

A-6.72 X Memory and Register Data Move (X:R)

X:R X:R

X Memory and Register Data Move

Operation:	**Assembler Syntax:**
Class I	
(. . . . .); X:ea→D1; S2→D2	(. . . . .) X:ea,D1 S2,D2
(. . . . .); S1→X:ea; S2→D2	(. . . . .) S1,X:ea S2,D2
(. . . . .); #xxxxxx→D1; S2→D2	(. . . . .) #xxxxxx,D1 S2,D2
Class II	
(. . . . .); A→X:ea; X0→A	(. . . . .) A,X:ea X0,A
(. . . . .); B→X:ea; X0→B	(. . . . .) B,X:ea X0,B

where (. . . . .) refers to any arithmetic or logical instruction which allows parallel moves.

Description: Class I: Move a one-word operand from/to X memory and move another word operand from an accumulator (S2) to an input register (D2). All memory addressing modes, including absolute addressing and 24-bit immediate data, may be used. The register to register move (S2,D2) allows a data ALU accumulator to be moved to a data ALU input register for use as a data ALU operand in the following instruction.

Class II: Move one-word operand from a data ALU accumulator to X memory and one-word operand from data ALU register X0 to a data ALU accumulator. One effective address is specified. All memory addressing modes, excluding long absolute addressing and long immediate data, may be used.

For both Class I and Class II X:R parallel data moves, if the arithmetic or logical opcode-operand portion of the instruction specifies a given destination accumulator, that same accumulator or portion of that accumulator may not be specified as a destination D1 in the parallel data bus move operation. Thus, if the opcode-operand portion of the instruction specifies the 56-bit A accumulator as its destination, the parallel data bus move portion of the instruction may not specify A0, A1, A2, or A as its destination D1. Similarly, if the opcode-operand portion of the instruction specifies the 56-bit B accumulator as its destination, the parallel data bus move portion of the instruction may not specify B0, B1, B2, or B as its destination D1. That is, **duplicate destinations are NOT allowed within the same instruction**.

If the opcode-operand portion of the instruction specifies a given source or destination register, that same register or portion of that register may be used as a source S1 and/or S2 in the parallel data bus move operation. This allows data to be moved in the same instruction in which it is being used as a source operand by a data ALU operation. That is, **duplicate sources are allowed within the same instruction**. Note that S1 and S2 may specify the same register.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	×	×	×	×
CCR							

✔	This bit is changed according to the standard definition
×	This bit is unchanged by the instruction

<u>Class I</u> Instruction Formats and opcodes:

(. . . . .) X:ea,D1 S2,D2
(. . . . .) S1,X:ea S2, D2
(. . . . .) #xxxxxx,D1 S2,D2

23							16	15							8	7 0
0	0	0	1	f	f	d	F	W	0	M	M	M	R	R	R	INSTRUCTION OPCODE
OPTIONAL EFFECTIVE ADDRESS EXTENSION																

Instruction Fields:

{ea}	**MMMRRR**	Effective Address (see Table A-15 on page A-240)
	W	Read S1 / Write D1 bit (see Table A-33 on page A-246)
{S1,D1}	**ff**	S1/D1 register [X0,X1,A,B] (see Table A-35 on page A-247)
{S2}	**d**	S2 accumulator [A,B] (see Table A-10 on page A-239)
{D2}	**F**	D2 input register [Y0,Y1] (see Table A-35 on page A-247)

<u>Class II</u> Instruction Formats and opcodes:

(. . . . .)A→X:ea X0→A
(. . . . .)B→X:ea X0→B

23							16	15							8	7 0
0	0	0	0	1	0	0	d	0	0	M	M	M	R	R	R	INSTRUCTION OPCODE
OPTIONAL EFFECTIVE ADDRESS EXTENSION																

Instruction Fields:

{ea}	**MMMRRR**	Effective Address (see Table A-19 on page A-242)
	d	Move opcode (see Table A-37 on page A-247)

A-6.73 Y Memory Data Move (Y:)

Y: Y:

Y Memory Data Move

Operation:	Assembler Syntax:
(. . . . .); Y:ea→D	(. . . . .) Y:ea,D
(. . . . .); Y:aa→D	(. . . . .) Y:aa,D
(. . . . .); S→Y:ea	(. . . . .) S,Y:ea
(. . . . .); S→Y:aa	(. . . . .) S,Y:aa
Y:(Rn+xxx)→D	MOVE Y:(Rn+xxx),D
Y:(Rn+xxxx)→D	MOVE Y:(Rn+xxxx),D
D→Y:(Rn+xxx)	MOVE D,Y:(Rn+xxx)
D→Y:(Rn+xxxx)	MOVE D,Y:(Rn+xxxx)

where (. . . . .) refers to any arithmetic or logical instruction which allows parallel moves.

Description: Move the specified word operand from/to Y memory. All memory addressing modes, including absolute addressing and 24-bit immediate data, may be used. Absolute short addressing may also be used.

If the arithmetic or logical opcode-operand portion of the instruction specifies a given destination accumulator, that same accumulator or portion of that accumulator may not be specified as a destination D in the parallel data bus move operation. Thus, if the opcode-operand portion of the instruction specifies the 56-bit A accumulator as its destination, the parallel data bus move portion of the instruction may not specify A0, A1, A2, or A as its destination D. Similarly, if the opcode-operand portion of the instruction specifies the 56-bit B accumulator as its destination, the parallel data bus move portion of the instruction may not specify B0, B1, B2, or B as its destination D. That is, **duplicate destinations are NOT allowed within the same instruction**.

If the opcode-operand portion of the instruction specifies a given source or destination register, that same register or portion of that register may be used as a source S in the parallel data bus move operation. This allows data to be moved in the same instruction in which it is being used as a source operand by a data ALU operation. That is, **duplicate sources are allowed within the same instruction**.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	×	×	×	×
CCR							

✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Note: The MOVE A,Y:ea operation will result in a 24-bit positive or negative saturation constant being stored in the specified 24-bit Y memory location if the signed integer portion of the A accumulator is in use.

Instruction Formats and opcodes 1:

(. . . . .)Y:ea,D
(. . . . .)S,Y:ea
(. . . . .)#xxxxxx,D

23 16	15 8	7 0
0 1 d d 1 d d d	W 1 M M M R R R	INSTRUCTION OPCODE
OPTIONAL EFFECTIVE ADDRESS EXTENSION		

(. . . . .)Y:aa,D
(. . . . .)S,Y:aa

23 16	15 8	7 0
0 1 d d 1 d d d	W 0 a a a a a a	INSTRUCTION OPCODE

Instruction Fields:

{ea}	**MMMRRR**	Effective Address (see Table A-15 on page A-240)
	W	Read S / Write D bit (see Table A-33 on page A-246)
{S,D}	**ddddd**	Source/Destination registers [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A,B,R0-R7,N0-N7] (see Table A-31 on page A-245)
{aa}	**aaaaaa**	Absolute Short Address

Instruction Formats and opcodes 2:

MOVE Y:(Rn+xxxx),D
MOVE D,Y:(Rn+xxxx)

23							16	15							8	7							0
0	0	0	0	1	0	1	1	0	1	1	1	0	R	R	R	1	W	D	D	D	D	D	D
Rn RELATIVE DISPLACEMENT																							

MOVE Y:(Rn+xxx),D
MOVE D,Y:(Rn+xxx)

23							16	15							8	7							0
0	0	0	0	0	0	1	a	a	a	a	a	a	R	R	R	1	a	1	W	D	D	D	D

Instruction Fields:

	W	Read S / Write D bit (see Table A-33 on page A-246)
{xxx}	**aaaaaaa**	7-bit sign extended Short Displacement Address
{Rn}	**RRR**	Address register (R0-R7)
{D}	**DDDD**	Source/Destination registers [X0,X1,Y0,Y1,A0,B0,A2,B2,A1,B1,A,B] (see Table A-34 on page A-246)
{S,D}	**DDDDDD**	Source/Destination registers [all on-chip registers] (see Table A-22 on page A-243)

A-6.74 Register and Y Memory Data Move (R:Y)

R:Y R:Y

Register and Y Memory Data Move

Operation:	**Assembler Syntax:**
Class I	
(. . . . .); S1→D1; Y:ea→D2	(. . . . .) S1,D1 Y:ea,D2
(. . . . .); S1→D1; S2→Y:ea	(. . . . .) S1,D1 S2,Y:ea
(. . . . .); S1→D1; #xxxxxx→D2	(. . . . .) S1,D1 #xxxxxx,D2
Class II	
(. . . . .); Y0 →A; A→Y:ea	(. . . . .) Y0,A A,Y:ea
(. . . . .); Y0→B; B→Y:ea	(. . . . .) Y0,B B,Y:ea

where (. . . . .) refers to any arithmetic or logical instruction which allows parallel moves.

Description: Class I: Move a one-word operand from an accumulator (S1) to an input register (D1) and move another word operand from/to Y memory. All memory addressing modes, including absolute addressing and 24-bit immediate data, may be used. The register to register move (S1,D1) allows a data ALU accumulator to be moved to a data ALU input register for use as a data ALU operand in the following instruction.

Class II: Move one-word operand from a data ALU accumulator to Y memory and one-word operand from data ALU register Y0 to a data ALU accumulator. One effective address is specified. All memory addressing modes, excluding long absolute addressing and long immediate data, may be used. Class II move operations have been added to the R:Y parallel move (and a similar feature has been added to the X:R parallel move) as an added feature available in the first quarter of 1989.

For both Class I and Class II R:Y parallel data moves, if the arithmetic or logical opcode-operand portion of the instruction specifies a given destination accumulator, that same accumulator or portion of that accumulator may not be specified as a destination D2 in the parallel data bus move operation. Thus, if the opcode-operand portion of the instruction specifies the 56-bit A accumulator as its destination, the parallel data bus move portion of the instruction may not specify A0, A1, A2, or A as its destination D2. Similarly, if the opcode-operand portion of the instruction specifies the 56-bit B accumulator as its destination, the parallel data bus move portion of the instruction may not specify B0, B1, B2, or B as its destination D2. That is, duplicate destinations are NOT allowed within the same instruction.

If the opcode-operand portion of the instruction specifies a given source or destination register, that same register or portion of that register may be used as a source S1 and/or S2 in the parallel data bus move operation. This allows data to be moved in the same instruction in which it is being used as a source operand by a data ALU operation. That is, **duplicate sources are allowed within the same instruction**. Note that S1 and S2 may specify the same register.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	×	×	×	×
CCR							

✔	This bit is changed according to the standard definition
×	This bit is unchanged by the instruction

<u>Class I</u> Instruction Formats and opcodes:

(. . . . .)S1,D1 Y:ea,D2
(. . . . .)S1,D1 S2,Y:ea
(. . . . .)S1,D1 #xxxxxx,D2

23 … 16	15 … 8	7 … 0
0 0 0 1 d e f f	W 1 M M M R R R	INSTRUCTION OPCODE
OPTIONAL EFFECTIVE ADDRESS EXTENSION		

Instruction Fields :

{ea}	**MMMRRR**	Effective Address (see Table A-15 on page A-240)
	W	Read S2 / Write D2 bit (see Table A-33 on page A-246)
{S1}	**d**	S1 accumulator [A,B] (see Table A-10 on page A-239)
{D1}	**e**	D1 input register [X0,X1] (see Table A-36 on page A-247)
{S2,D2}	**ff**	S2/D2 register [Y0,Y1,A,B] (see Table A-36 on page A-247)

Class II Instruction Formats and opcodes:

(. . . . .)Y0 → A A → Y:ea
(. . . . .)Y0 → B B → Y:ea

23							16	15							8	7 0
0	0	0	0	1	0	0	d	1	0	M	M	M	R	R	R	INSTRUCTION OPCODE
OPTIONAL EFFECTIVE ADDRESS EXTENSION																

Instruction Fields:

MMMRRR ea=6-bit Effective Address (see Table A-19 on page A-242)
d Move opcode (see Table A-37 on page A-247)

A-6.75 Long Memory Data Move (L:)

L: L:

Long Memory Data Move

Operation:	Assembler Syntax:
(. . . . .); X:ea → D1; Y:ea → D2	(. . . . .) L:ea,D
(. . . . .); X:aa → D1; Y:aa → D2	(. . . . .) L:aa,D
(. . . . .); S1 → X:ea; S2 → Y:ea	(. . . . .) S,L:ea
(. . . . .); S1 → X:aa; S2 → Y:aa	(. . . . .) S,L:aa

where (. . . . .) refers to any arithmetic or logical instruction which allows parallel moves.

Description: Move one 48-bit long-word operand from/to X and Y memory. Two data ALU registers are concatenated to form the 48-bit long-word operand. This allows efficient moving of both double-precision (high:low) and complex (real:imaginary) data from/to one effective address in L (X:Y) memory. The same effective address is used for both the X and Y memory spaces; thus, only one effective address is required. Note that the A, B, A10, and B10 operands reference a single 48-bit signed (double-precision) quantity while the X, Y, AB, and BA operands reference two separate (i.e., real and imaginary) 24-bit signed quantities. All memory alterable addressing modes may be used. Absolute short addressing may also be used.

If the arithmetic or logical opcode-operand portion of the instruction specifies a given destination accumulator, that same accumulator or portion of that accumulator may not be specified as a destination D in the parallel data bus move operation. Thus, if the opcode-operand portion of the instruction specifies the 56-bit A accumulator as its destination, the parallel data bus move portion of the instruction may not specify A, A10, AB, or BA as destination D. Similarly, if the opcode-operand portion of the instruction specifies the 56-bit B accumulator as its destination, the parallel data bus move portion of the instruction may not specify B, B10, AB, or BA as its destination D. That is, duplicate destinations are NOT allowed within the same instruction.

If the opcode-operand portion of the instruction specifies a given source or destination register, that same register or portion of that register may be used as a source S in the parallel data bus move operation. This allows data to be moved in the same instruction in which it is being used as a source operand by a data ALU operation. That is, duplicate sources are allowed within the same instruction.

Note: The operands A10, B10, X, Y, AB, and BA may be used only for a 48-bit long memory move as previously described. These operands may not be used in any other type of instruction or parallel move.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	×	×	×	×
CCR							

✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Note: The MOVE A,L:ea operation will result in a 48-bit positive or negative saturation constant being stored in the specified 24-bit X and Y memory locations if the signed integer portion of the A accumulator is in use. The MOVE AB,L:ea operation will result in either one or two 24-bit positive and/or negative saturation constant(s) being stored in the specified 24-bit X and/or Y memory location(s) if the signed integer portion of the A and/or B accumulator(s) is in use.

Instruction Formats and opcodes:

(. . . . .)L:ea,D
(. . . . .)S,L:ea

23 16	15 8	7 0
0 1 0 0 L 0 L L	W 1 M M M R R R	INSTRUCTION OPCODE
OPTIONAL EFFECTIVE ADDRESS EXTENSION		

(. . . . .)L:aa,D
(. . . . .)S,L:aa

23 16	15 8	7 0
0 1 0 0 L 0 L L	W 0 a a a a a a	INSTRUCTION OPCODE

Instruction Fields:

{ea}	**MMMRRR**	Effective Address (see Table A-18 on page A-241)
	W	Read S / Write D bit (see Table A-33 on page A-246)
{L}	**LLL**	Two data ALU registers (see Table A-23 on page A-243)
{aa}	**aaaaaa**	Absolute Short Address

A-6.76 XY Memory Data Move (X: Y:)

X: Y: X: Y:

XY Memory Data Move

Operation:	**Assembler Syntax:**
(. . . . .); X:<eax> → D1; Y:<eay> → D2	(. . . . .) X:<eax>,D1 Y:<eay>,D2
(. . . . .); X:<eax> → D1; S2 → Y:<eay>	(. . . . .) X:<eax>,D1 S2,Y:<eay>
(. . . . .); S1 → X:<eax>; Y:<eay> → D2	(. . . . .) S1,X:<eax> Y:<eay>,D2
(. . . . .); S1 → X:<eax>; S2 → Y:<eay>	(. . . . .) S1,X:<eax> S2,Y:<eay>

where (. . . . .) refers to any arithmetic or logical instruction which allows parallel moves.

Description: Move a one-word operand from/to X memory and move another word operand from/to Y memory. Note that two independent effective addresses are specified (<eax> and <eay>) where one of the effective addresses uses the lower bank of address registers (R0–R3) while the other effective address uses the upper bank of address registers (R4–R7). All parallel addressing modes may be used.

If the arithmetic or logical opcode-operand portion of the instruction specifies a given destination accumulator, that same accumulator or portion of that accumulator may not be specified as a destination D1 or D2 in the parallel data bus move operation. Thus, if the opcode-operand portion of the instruction specifies the 56-bit A accumulator as its destination, the parallel data bus move portion of the instruction may not specify A as its destination D1 or D2. Similarly, if the opcode-operand portion of the instruction specifies the 56-bit B accumulator as its destination, the parallel data bus move portion of the instruction may not specify B as its destination D1 or D2. That is, **duplicate destinations are NOT allowed within the same instruction**. D1 and D2 may not specify the same register.

If the instruction specifies an access to an internal X-I/O and internal Y-I/O modules (reflected by the address of the X memory space and of the Y memory space), than only the access to the internal X-I/O module will be executed. The access to the Y-I/O module will be discarded.

If the opcode-operand portion of the instruction specifies a given source or destination register, that same register or portion of that register may be used as a source S1 and/or

S2 in the parallel data bus move operation. This allows data to be moved in the same instruction in which it is being used as a source operand by a data ALU operation. That is, **duplicate sources are allowed within the same instruction**. Note that S1 and S2 may specify the same register.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	×	×	×	×
CCR							

✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Instruction Formats and opcodes:

(. . . . .)X:<eax>,D1 Y:<eay>,D2
(. . . . .)X:<eax>,D1 S2,Y:<eay>
(. . . . .)S1,X:<eax> Y:<eay>,D2
(. . . . .)S1,X:<eax> S2,Y:<eay>

23 16	15 8	7 0
1 w m m e e f f	W r r M M R R R	INSTRUCTION OPCODE

Instruction Fields :

{<eax>} **MMRRR** 5-bit X Effective Address (R0–R3 or R4–R7)
{<eay>} **mmrr** 4-bit Y Effective Address (R4–R7 or R0–R3)
{S1,D1} **ee** S1/D1 register [X0,X1,A,B]
{S2,D2} **ff** S2/D2 register [Y0,Y1,A,B]
MMRRR,mmrr,ee,ff: see Table A-38 on page A-248
W X move Operation Control (See Table A-33 on page A-246)
w Y move Operation Control (See Table A-33 on page A-246)

A-6.77 Move Control Register (MOVEC)

MOVEC MOVEC

Move Control Register

Operation:	Assembler Syntax:	
[X or Y]:ea→D1	MOVE(C)	[Xor Y]:ea,D1
[X or Y]:aa→D1	MOVE(C)	[Xor Y]:aa,D1
S1→[X or Y]:ea	MOVE(C)	S1,[X or Y]:ea
S1→[X or Y]:aa	MOVE(C)	S1,[X or Y]:aa
S1→D2	MOVE(C)	S1,D2
S2→D1	MOVE(C)	S2,D1
#xxxx→D1	MOVE(C)	#xxxx,D1
#xx→D1	MOVE(C)	#xx,D1

Description: Move the contents of the specified source **control register** S1 or S2 to the specified destination or move the specified source to the specified destination **control register** D1 or D2. The control registers S1 and D1 are a subset of the S2 and D2 register set and consist of the address ALU modifier registers and the program controller registers. These registers may be moved to or from any other register or memory space. All memory addressing modes, as well as an immediate short addressing mode, may be used.

If the system stack register SSH is specified as a source operand, the system stack pointer (SP) is postdecremented by 1 after SSH has been read. If the system stack register SSH is specified as a destination operand, the system stack pointer (SP) is preincremented by 1 before SSH is written. This allows the system stack to be efficiently extended using software stack pointer operations.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
●	●	●	●	●	●	●	●
CCR							

For D1 or D2=SR operand :

- S Set according to bit 7 of the source operand
- L Set according to bit 6 of the source operand
- E Set according to bit 5 of the source operand
- U Set according to bit 4 of the source operand
- N Set according to bit 3 of the source operand
- Z Set according to bit 2 of the source operand
- V Set according to bit 1 of the source operand
- C Set according to bit 0 of the source operand

For D1 and D2≠SR operand :

- S Set if data growth been detected
- L Set if data limiting has occurred during the move

Instruction Formats and opcodes:

MOVE(C) [X or Y]:ea,D1
MOVE(C) S1,[X or Y]:ea
MOVE(C) #xxxx,D1

23							16	15							8	7							0
0	0	0	0	0	1	0	1	W	1	M	M	M	R	R	R	O	S	1	d	d	d	d	d
OPTIONAL EFFECTIVE ADDRESS EXTENSION																							

MOVE(C) [X or Y]:aa,D1
MOVE(C) S1,[X or Y]:aa

23							16	15							8	7							0
0	0	0	0	0	1	0	1	W	0	a	a	a	a	a	a	0	S	1	d	d	d	d	d

MOVE(C) S1,D2
MOVE(C) S2,D1

23							16	15							8	7							0
0	0	0	0	0	1	0	0	W	1	e	e	e	e	e	e	1	0	1	d	d	d	d	d

MOVE(C) #xx,D1

23							16	15							8	7							0
0	0	0	0	0	1	0	1	i	i	i	i	i	i	i	i	1	0	1	d	d	d	d	d

Instruction Fields:

{ea}	**MMMRRR**	Effective Address (see Table A-15 on page A-240)
	W	Read S / Write D bit (see Table A-33 on page A-246)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{S1,D1}	**ddddd**	Program Controller register [M0-M7,EP,VBA,SZ,SR,OMR,SP,SSH,SSL,LA,LC] (see Table A-41 on page A-249)
{aa}	**aaaaaa**	aa=6-bit Absolute Short Address
{S2,D2}	**eeeeee**	S2/D2 register [all on-chip registers] (see Table A-22 on page A-243)
{#xx}	**iiiiiiii**	#xx=8-bit Immediate Short Data

A-6.78 Move Program Memory (MOVEM)

MOVEM MOVEM

Move Program Memory

Operation:	Assembler Syntax:	
S→P:ea	MOVE(M)	S,P:ea
S→P:aa	MOVE(M)	S,P:aa
P:ea→D	MOVE(M)	P:ea,D
P:aa→D	MOVE(M)	P:aa,D

Description: Move the specified operand from/to the specified **program** (P) **memory location**. This is a powerful move instruction in that the source and destination registers S and D may be **any** register. All memory alterable addressing modes may be used as well as the absolute short addressing mode.

If the system stack register SSH is specified as a source operand, the system stack pointer (SP) is postdecremented by 1 after SSH has been read. If the system stack register SSH is specified as a destination operand, the system stack pointer (SP) is preincremented by 1 before SSH is written. This allows the system stack to be efficiently extended using software stack pointer operations.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
●	●	●	●	●	●	●	●
CCR							

For D=SR operand :

- S Set according to bit 7 of the source operand
- L Set according to bit 6 of the source operand
- E Set according to bit 5 of the source operand
- U Set according to bit 4 of the source operand
- N Set according to bit 3 of the source operand
- Z Set according to bit 2 of the source operand
- V Set according to bit 1 of the source operand
- C Set according to bit 0 of the source operand

For D≠SR operand :

- S Set if data growth been selected
- L Set if data limiting has occurred during the move

Instruction Formats and opcodes:

MOVE(M) S,P:ea
MOVE(M) P:ea,D

23							16	15							8	7							0
0	0	0	0	0	1	1	1	W	1	M	M	M	R	R	R	1	0	d	d	d	d	d	d
OPTIONAL EFFECTIVE ADDRESS EXTENSION																							

MOVE(M) S,P:aa
MOVE(M) P:aa,D

23							16	15							8	7							0
0	0	0	0	0	1	1	1	W	0	a	a	a	a	a	a	0	0	d	d	d	d	d	d

Instruction Fields:

{ea}	**MMMRRR**	Effective Address (see Table A-18 on page A-241)
	W	Read S / Write D bit (see Table A-33 on page A-246)
{ S,D}	**dddddd**	Source/Destination register [all on-chip registers] (see Table A-22 on page A-243)
{aa}	**aaaaaa**	Absolute Short Address

A-6.79 Move Peripheral Data (MOVEP)

MOVEP MOVEP

Move Peripheral Data

Operation:	Assembler Syntax:	
[X or Y]:pp → D	MOVEP	[X or Y]:pp,D
[X or Y]:qq → D	MOVEP	[X or Y]:qq,D
[X or Y]:pp → [X or Y]:ea	MOVEP	[X or Y]:pp,[X or Y]:ea
[X or Y]:qq → [X or Y]:ea	MOVEP	[X or Y]:qq,[X or Y]:ea
[X or Y]:pp → P:ea	MOVEP	[X or Y]:pp,P:ea
[X or Y]:qq → P:ea	MOVEP	[X or Y]:qq,P:ea
S → [X or Y]:pp	MOVEP	S,[X or Y]:pp
S → [X or Y]:qq	MOVEP	S,[X or Y]:qq
[X or Y]:ea → [X or Y]:pp	MOVEP	[X or Y]:ea,[X or Y]:pp
[X or Y]:ea → [X or Y]:qq	MOVEP	[X or Y]:ea,[X or Y]:qq
P:ea → [X or Y]:pp	MOVEP	P:ea,[X or Y]:pp
P:ea → [X or Y]:qq	MOVEP	P:ea,[X or Y]:qq

Description: Move the specified operand from/to the specified **X or Y I/O peripheral**. The I/O short addressing mode is used for the I/O peripheral address. All memory addressing modes may be used for the X or Y memory effective address; all memory alterable addressing modes may be used for the P memory effective address. ALL the I/O space ($FFFF80-$FFFFFF) can be accessed, except for the P: reference opcode.

If the system stack register SSH is specified as a source operand, the system stack pointer (SP) is postdecremented by 1 after SSH has been read. If the system stack register SSH is specified as a destination operand, the system stack pointer (SP) is preincremented by 1 before SSH is written. This allows the system stack to be efficiently extended using software stack pointer operations.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
●	●	●	●	●	●	●	●
CCR							

For D=SR operand :

- ● S Set according to bit 7 of the source operand
- ● L Set according to bit 6 of the source operand
- ● E Set according to bit 5 of the source operand
- ● U Set according to bit 4 of the source operand
- ● N Set according to bit 3 of the source operand
- ● Z Set according to bit 2 of the source operand
- ● V Set according to bit 1 of the source operand
- ● C Set according to bit 0 of the source operand

For D≠SR operand :

- ● S Set if data growth been selected
- ● L Set if data limiting has occurred during the move

Instruction Formats and opcodes:

X: or Y: Reference (high I/O address)

MOVEP [X or Y]:pp,[X or Y]:ea
MOVEP [X or Y]:ea,[X or Y]:pp

23 16	15 8	7 0
0 0 0 0 1 0 0 s	W 1 M M M R R R	1 S p p p p p p
OPTIONAL EFFECTIVE ADDRESS EXTENSION		

X: or Y: Reference (low I/O address)

MOVEP X:qq,[X or Y]:ea
MOVEP [X or Y]:ea,X:qq

23 16	15 8	7 0
0 0 0 0 0 1 1 1	W 1 M M M R R R	0 S q q q q q q
OPTIONAL EFFECTIVE ADDRESS EXTENSION		

X: or Y: Reference (low I/O address)

MOVEP Y:qq,[X or Y]:ea
MOVEP [X or Y]:ea,Y:qq

23 16	15 8	7 0
0 0 0 0 0 1 1 1	W 0 M M M R R R	1 S q q q q q q
OPTIONAL EFFECTIVE ADDRESS EXTENSION		

P: Reference (high I/O address)

MOVEP P:ea,[X or Y]:pp
MOVEP [X or Y]:pp,P:ea

							16	15							8	7							0
0	0	0	0	1	0	0	s	W	1	M	M	M	R	R	R	0	1	p	p	p	p	p	p

P: Reference (low I/O address)

MOVEP P:ea,[X or Y]:qq
MOVEP [X or Y]:qq,P:ea

							16	15							8	7							0
0	0	0	0	0	0	0	0	1	W	M	M	M	R	R	R	0	S	q	q	q	q	q	q

Register Reference (high I/O address)

MOVEP S,[X or Y]:pp
MOVEP [X or Y]:pp,D

23							16	15							8	7							0
0	0	0	0	1	0	0	s	W	1	d	d	d	d	d	d	0	0	p	p	p	p	p	p

Register Reference: (low I/O address)

MOVEP S,X:qq
MOVEP X:qq,D

23							16	15							8	7							0
0	0	0	0	0	1	0	0	W	1	d	d	d	d	d	d	1	q	0	q	q	q	q	q

Register Reference: (low I/O address)

MOVEP S,Y:qq
MOVEP Y:qq,D

23							16	15							8	7							0
0	0	0	0	0	1	0	0	W	1	d	d	d	d	d	d	0	q	1	q	q	q	q	q

Instruction Fields:

{ea}	**MMMRRR**	Effective Address (see Table A-16 on page A-241)
{pp}	**pppppp**	I/O Short Address [64 addresses: $FFFFC0-$FFFFFF]
{qq}	**qqqqqq**	I/O Short Address [64 addresses: $FFFF80-$FFFFBF]
{X/Y}	**S**	Memory space [X,Y] (see Table A-17 on page A-241)
{X/Y}	**s**	Peripheral space [X,Y] (see Table A-17 on page A-241)
	W	Read/write-peripheral (see Table A-33 on page A-246)
{S,D}	**dddddd**	Source/Destination register [all on-chip registers] (see Table A-22 on page A-243)

A-6.80 Signed Multiply (MPY)

MPY MPY

Signed Multiply

Operation:	**Assembler Syntax:**	
$\pm S1*S2 \rightarrow D$ (parallel move)	MPY	(±)S1,S2,D (parallel move)
$\pm S1*S2 \rightarrow D$ (parallel move)	MPY	(±)S2,S1,D (parallel move)
$\pm(S1*2^{-n}) \rightarrow D$ (**no** parallel move)	MPY	(±)S,#n,D (**no** parallel move)

Description: Multiply the two signed 24-bit source operands S1 and S2 and store the resulting product in the specified 56-bit destination accumulator D. Or, multiply the signed 24-bit source operand S by the positive 24-bit immediate operand 2^{-n} and store the resulting product in the specified 56-bit destination accumulator D. The “–” sign option is used to negate the specified product prior to accumulation. The default sign option is “+”.

Note: When the processor is in the Double Precision Multiply Mode, the following instructions do not execute in the normal way and should only be used as part of the double precision multiply algorithm:

MPY Y0, X0, A MPY Y0, X0, B

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	✔	✔	✔	✔	✔	×
CCR							

✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Instruction Formats and opcodes 1:

MPY (±)S1,S2,D
MPY (±)S2,S1,D

23 16 15 8	7 4	3 0
DATA BUS MOVE FIELD	1 Q Q Q	d k 0 0
OPTIONAL EFFECTIVE ADDRESS EXTENSION		

Instruction Fields:

{S1,S2} **QQQ** Source registers S1,S2
[X0*X0,Y0*Y0,X1*X0,Y1*Y0,X0*Y1,Y0*X0,X1*Y0,Y1*X1]
(see Table A-26 on page A-244)

{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

{±+/-} **k** Sign [+,-] (see Table A-29 on page A-244)

Instruction Formats and opcode 2:

MPY (±)S,#n,D

23 16	15 8	7 0
0 0 0 0 0 0 0 1	0 0 0 s s s s s	1 1 Q Q d k 0 0

Instruction Fields:

{S} **QQ** Source register [Y1,X0,Y0,X1]] (see Table A-27 on page A-244)

{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

{±} **k** Sign [+,-] (see Table A-29 on page A-244)

{#n} **sssss** Immediate operand (see Table A-32 on page A-246)

A-6.81 Mixed Multiply (MPY su/uu)

MPY(su,uu) MPY(su,uu)

Mixed Multiply

Operation:	**Assembler Syntax:**
±S1∗S2→D (S1 unsigned, S2 unsigned)	MPYuu (±)S1,S2,D (no parallel move)
±S1∗S2→D (S1 signed, S2 unsigned)	MPYsu (±)S2,S1,D (no parallel move)

Description: Multiply the two 24-bit source operands S1 and S2 and store the resulting product in the specified 56-bit destination accumulator D. One or two of the source operands can be unsigned. The “–” sign option is used to negate the specified product prior to accumulation. The default sign option is “+”.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	✓	✓	✓	✓	✓	✓	×
CCR							

✓ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes :

	23							16	15							8	7							0
MPYsu (±)S1,S2,D MPYuu (±)S1,S2,D	0	0	0	0	0	0	0	1	0	0	1	0	0	1	1	1	1	s	d	k	Q	Q	Q	Q

Instruction Fields:

{S1,S2}	**QQQQ**	Source registers S1,S2 [all combinations of X0,X1,Y0 and Y1] (see Table A-30 on page A-245)
{D}	**d**	Destination accumulator [A,B] (see Table A-10 on page A-239)
{±}	**k**	Sign [+,-] (see Table A-29 on page A-244)
{s}		[ss,us] (see Table A-40 on page A-249)

A-6.82 Signed Multiply with Immediate Operand (MPYI)

MPYI MPYI

Signed Multiply with Immediate Operand

Operation:	**Assembler Syntax:**
±#xxxxxx∗S→D	MPYI (±)#xxxxxx,S,D

Description: Multiply the immediate 24-bit source operand #xxxxxx with the 24-bit register source operand S and store the resulting product in the specified 56-bit destination accumulator D. The “–” sign option is used to negate the specified product prior to accumulation. The default sign option is “+”.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	✔	✔	✔	✔	✔	✔	×
CCR							

✔	This bit is changed according to the standard definition
×	This bit is unchanged by the instruction

Instruction Formats and opcode:

	23							16	15							8	7							0
MPYI (±)#xxxxxx,S,D	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	1	q	q	d	k	0	0
	IMMEDIATE DATA EXTENSION																							

Instruction Fields:

{S}	**qq**	Source register [X0,Y0,X1,Y1] (see Table A-28 on page A-244)
{D}	**d**	Destination accumulator [A,B] (see Table A-10 on page A-239)
{±}	**k**	Sign [+,-] (see Table A-29 on page A-244)
#xxxxxx		24-bit Immediate Long Data extension word

A-6.83 Signed Multiply and Round (MPYR)

MPYR MPYR

Signed Multiply and Round

Operation:	Assembler Syntax:	
$\pm S1 * S2 + r \rightarrow D$ (parallel move)	MPYR	(±)S1,S2,D (parallel move)
$\pm S1 * S2 + r \rightarrow D$ (parallel move)	MPYR	(±)S2,S1,D (parallel move)
$\pm (S1 * 2^{-n}) + r \rightarrow D$ (**no** parallel move)	MPYR	(±)S,#n,D (**no** parallel move)

Description: Multiply the two signed 24-bit source operands S1 and S2 (**or** the signed 24-bit source operand S by the positive 24-bit immediate operand 2^{-n}), round the result using either convergent or two’s complement rounding, and store it in the specified 56-bit destination accumulator D.

The “–” sign option is used to negate the product prior to rounding. The default sign option is “+”.

The contribution of the LS bits of the result is rounded into the upper portion of the destination accumulator. Once the rounding has been completed, the LS bits of the destination accumulator D are loaded with zeros to maintain an unbiased accumulator value which may be reused by the next instruction. The upper portion of the accumulator contains the rounded result which may be read out to the data buses. Refer to the RND instruction for more complete information on the rounding process.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	✔	✔	✔	✔	✔	×
CCR							

✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Instruction Formats and opcodes 1:

MPYR (±)S1,S2,D
MPYR (±)S2,S1,D

23 16 15 8	7							0
DATA BUS MOVE FIELD	1	Q	Q	Q	d	k	0	1
OPTIONAL EFFECTIVE ADDRESS EXTENSION								

Instruction Fields 1:

{S1,S2} **QQQ** Source registers S1,S2
[X0*X0,Y0*Y0,X1*X0,Y1*Y0,X0*Y1,Y0*X0,X1*Y0,Y1*X1]
(see Table A-26 on page A-244)
{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)
{±} **k** Sign [+,-] (see Table A-29 on page A-244)

Instruction Formats and opcode 2:

MPYR (±)S,#n,D

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	0	0	s	s	s	s	s	1	1	Q	Q	d	k	0	1

Instruction Fields 2:

{S} **QQ** Source register [Y1,X0,Y0,X1]] (see Table A-27 on page A-244)
{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)
{±} **k** Sign [+,-] (see Table A-29 on page A-244)
{#n} **sssss** Immediate operand (see Table A-32 on page A-246)

A-6.84 Signed Multiply and Round with Immediate Operand (MPYRI)

MPYRI MPYRI

Signed Multiply and Round with Immediate Operand

Operation:	**Assembler Syntax:**
±#xxxxxx∗S+r →D	MPYRI (±)#xxxxxx,S,D

Description: Multiply the two signed 24-bit source operands #xxxxxx and S, round the result using either convergent or two’s complement rounding, and store it in the specified 56-bit destination accumulator D.

The “–” sign option is used to negate the product prior to rounding. The default sign option is “+”.

The contribution of the LS bits of the result is rounded into the upper portion of the destination accumulator. Once the rounding has been completed, the LS bits of the destination accumulator D are loaded with zeros to maintain an unbiased accumulator value which may be reused by the next instruction. The upper portion of the accumulator contains the rounded result which may be read out to the data buses. Refer to the RND instruction for more complete information on the rounding process.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	✔	✔	✔	✔	✔	✔	×
CCR							

✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Instruction Formats and opcode:

	23							16	15							8	7							0
MPYRI (±)#xxxxxx,S,D	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	1	q	q	d	k	0	1
	IMMEDIATE DATA EXTENSION																							

Instruction Fields:

{S}	**qq**	Source register [X0,Y0,X1,Y1] (see Table A-28 on page A-244)
{D}	**d**	Destination accumulator [A,B] (see Table A-10 on page A-239)
{±}	**k**	Sign [+,-] (see Table A-29 on page A-244)
#xxxxxx		24-bit Immediate Long Data extension word

A-6.85 Negate Accumulator (NEG)

NEG NEG

Negate Accumulator

Operation:

0–D ➞ D (parallel move)

Assembler Syntax:

NEG D (parallel move)

Description: Negate the destination operand D and store the result in the destination accumulator. This is a 56-bit, twos-complement operation.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	✔	✔	✔	✔	✔	×
CCR							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

		23 16 15 8	7							0
NEG	D	DATA BUS MOVE FIELD	0	0	1	1	d	1	1	0
		OPTIONAL EFFECTIVE ADDRESS EXTENSION								

Instruction Fields:

{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

A-6.86 No Operation (NOP)

NOP NOP

No Operation

Operation:

PC+1➞PC

Assembler Syntax:

NOP

Description: Increment the program counter (PC). Pending pipeline actions, if any, are completed. Execution continues with the instruction following the NOP.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcode:

	23							16	15							8	7							0
NOP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Instruction Fields : None

A-6.87 Norm Accumulator Iteration (NORM)

NORM NORM

Norm Accumulator Iteration

Operation:

If $\overline{E}$ • U • $\overline{Z}$=1, then ASL D and Rn–1→Rn
else if E=1, then ASR D and Rn+1→R
else NOP

where $\overline{E}$ denotes the logical complement of E, and
where • denotes the logical AND operator

Assembler Syntax:

NORM Rn,D

Description: Perform one normalization iteration on the specified destination operand D, update the specified address register Rn based upon the results of that iteration, and store the result back in the destination accumulator. This is a 56-bit operation. If the accumulator extension is not in use, the accumulator is unnormalized, and the accumulator is not zero, the destination operand is arithmetically shifted one bit to the left, and the specified address register is decremented by 1. If the accumulator extension register is in use, the destination operand is arithmetically shifted one bit to the right, and the specified address register is incremented by 1. If the accumulator is normalized or zero, a NOP is executed and the specified address register is not affected. Since the operation of the NORM instruction depends on the E, U, and Z condition code register bits, these bits must correctly reflect the current state of the destination accumulator prior to executing the NORM instruction.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	✓	✓	✓	✓	✓	●	×
CCR							

●	Set if bit 55 is changed as a result of a left shift
✓	This bit is changed according to the standard definition
×	This bit is unchanged by the instruction

Instruction Formats and opcode:

	23							16	15							8	7							0
NORM Rn,D	0	0	0	0	0	0	0	1	1	1	0	1	1	R	R	R	0	0	0	1	d	1	0	1

Instruction Fields:

{D}	**d**	Destination accumulator [A,B] (see Table A-10 on page A-239)
{Rn}	**RRR**	Address register [R0-R7]

A-6.88 Fast Accumulator Normalization (NORMF)

NORMF NORMF

Fast Accumulator Normalization

Operation:

If S[23]=0 then ASR S,D
else ASL -S,D

Assembler Syntax:

NORMF S,D

Description: Arithmetically shift the destination accumulator either left or right as specified by the source operand sign and value. If the source operand is negative then the accumulator is left shifted, and if the source operand is positive then it is right shifted. The source accumulator value should be between +56 to -55 (or +40 to -39 in sixteen bit mode). This instruction can be used to normalize the specified accumulator D, by arithmetically shifting it either left or right so as to bring the leading one or zero to bit location 46. The number of needed shifts is specified by the source operand. This number could be calculated by a previous CLB instruction. For normalization the source accumulator value should be between +8 to -47 (or +8 to -31 in sixteen bit mode).

This is a 56 bit operation.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	✔	✔	✔	✔	✔	●	×
CCR							

● V Set if bit 55 is changed any time during the shift operation. Cleared otherwise.
✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction.

Example:

```
CLB         A,B         ;Count leading bits.

NORMF       B1,A        ;Normalize A.
```

If the base exponent is stored in R1 it can be updated by the following commands.

```
MOVE        B1,N1       ;Update N1 with shift amount

MOVE        (R1)+N1     ;Increment or decrement exponent
```

	Before execution	After execution
CLB A,B	A: $20:000000:000000	B: $00:000007:000000
NORMF B1,A	A: $20:000000:000000	A: $00:400000:000000

Explanation of example: Prior to execution, the 56-bit A accumulator contains the value $20:000000:000000. The CLB instruction updates the B accumulator to the number of needed shifts, 7 in this example. The NORMF instruction performs 7 shifts to the right on A accumulator, and normalization of A is achieved. The exponent register is updated according to the number of shifts.

Instruction Formats and opcode

	23							16	15							8	7							0
NORMF S,D	0	0	0	0	1	1	0	0	0	0	0	1	1	1	1	0	0	0	1	0	s	s	s	D

Instruction Fields:

{S}	**sss**	Source register [X0,X1,Y0,Y1,A1,B1] (see Table A-15 on page A-240)
{D}	**D**	Destination accumulator [A,B] (see Table A-10 on page A-239)

A-6.89 Logical Complement (NOT)

NOT NOT

Logical Compliment

Operation:

$\overline{D[47:24]} \rightarrow D[47:24]$ (parallel move)

where “—” denotes the logical NOT operator

Assembler Syntax:

NOT D (parallel move)

Description: Take the ones complement of bits 47–24 of the destination operand D and store the result back in bits 47–24 of the destination accumulator. This is a 24-bit operation. The remaining bits of D are not affected.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	●	●	●	×
CCR							

●	N	Set if bit 47 of the result is set
●	Z	Set if bits 47–24 of the result are zero
●	V	Always cleared
✔		This bit is changed according to the standard definition
×		This bit is unchanged by the instruction

Instruction Formats and opcodes:

NOT D

23 16 15 8	7							0
DATA BUS MOVE FIELD	0	0	0	1	d	1	1	1
OPTIONAL EFFECTIVE ADDRESS EXTENSION								

Instruction Fields:

{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

A-6.90 Logical Inclusive OR (OR)

OR OR

Logical Inclusive OR

Operation:	**Assembler Syntax:**
S+D[47:24] → D[47:24] (parallel move)	OR S,D (parallel move)
#xx+D[47:24] → D[47:24]	OR #xx,D
#xxxxxx+D[47:24] → D[47:24]	OR #xxxxxx,D

where + denotes the logical inclusive OR operator

Description: Logically inclusive OR the source operand S with bits 47–24 of the destination operand D and store the result in bits 47–24 of the destination accumulator. The source can be a 24-bit register, 6-bit short immediate or 24-bit long immediate. This instruction is a 24-bit operation. The remaining bits of the destination operand D are not affected.

When using 6-bit immediate data, the data is interpreted as an unsigned integer. That is, the 6 bits will be right aligned and the remaining bits will be zeroed to form a 24-bit source operand.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	●	●	●	×
CCR							

- ● N Set if bit 47 of the result is set
- ● Z Set if bits 47–24 of the result are zero
- ● V Always cleared
- ✔ This bit is changed according to the standard definition
- × This bit is unchanged by the instruction

Instruction Formats and opcodes:

OR S,D

23							16	15							8	7							0
DATA BUS MOVE FIELD																0	1	J	J	d	0	1	0
OPTIONAL EFFECTIVE ADDRESS EXTENSION																							

OR #xx,D

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	1	i	i	i	i	i	i	1	0	0	0	d	0	1	0

OR #xxxxxx,D

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	1	0	0	d	0	1	0
IMMEDIATE DATA EXTENSION																							

Instruction Fields:

{S}	**JJ**	Source input register [X0,X1,Y0,Y1] (see Table A-12 on page A-239)
{D}	**d**	Destination accumulator [A/B] (see Table A-10 on page A-239)
{#xx}	**iiiiii**	6-bit Immediate Short Data
{#xxxxxx}		24-bit Immediate Long Data extension word

A-6.91 OR Immediate with Control Register (ORI)

ORI ORI

OR Immediate with Control register

Operation:

#xx+D ➞ D

where + denotes the logical inclusive OR operator

Assembler Syntax:

OR(I) #xx,D

Description: Logically OR the 8-bit immediate operand (#xx) with the contents of the destination control register D and store the result in the destination control register. The condition codes are affected only when the condition code register is specified as the destination operand.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
●	●	●	●	●	●	●	●
CCR							

For CCR Operand:

- S Set if bit 7 of the immediate operand is set
- L Set if bit 6 of the immediate operand is set
- E Set if bit 5 of the immediate operand is set
- U Set if bit 4 of the immediate operand is set
- N Set if bit 3 of the immediate operand is set
- Z Set if bit 2 of the immediate operand is set
- V Set if bit 1 of the immediate operand is set
- C Set if bit 0 of the immediate operand is set

For MR and OMR Operands: The condition codes are not affected using these operands.

Instruction Formats and opcodes:

	23							16	15							8	7							0
OR(I) #xx,D	0	0	0	0	0	0	0	0	i	i	i	i	i	i	i	i	1	1	1	1	1	0	E	E

Instruction fields:

{D}	**EE**	Program Controller register [MR,CCR,COM,EOM] (see Table A-13 on page A-239)
{#xx}	**iiiiiiii**	Immediate Short Data

A-6.92 Program-Cache Flush (PFLUSH)

PFLUSH PFLUSH

Program Cache Flush

Operation:

Flush instruction cache

Assembler Syntax:

PFLUSH

Description: Flush the whole instruction cache, unlock all cache sectors, set the LRU stack and tag registers to their default values.

The PFLUSH instruction is enabled only in Cache Mode. In PRAM Mode it will cause an illegal instruction trap.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcode:

	23							16	15							8	7							0
PFLUSH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Instruction Fields: None

A-6.93 Program-Cache Flush Unlock Sectors(PFLUSHUN)

PFLUSHUN PFLUSHUN

Program Cache Flush Unlocked Sectors

Operation:	**Assembler Syntax:**
Flush Unlocked instruction cache sectors	PFLUSHUN

Description: Flush the instruction cache sectors which are unlocked, set the LRU stack to its default value and set the unlocked tag registers to their default values.

The PFLUSHUN instruction is enabled only in Cache Mode. In PRAM Mode it will cause an illegal instruction trap.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcode:

	23							16	15							8	7							0
PFLUSHUN	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Instruction Fields: None

A-6.94 Program-Cache Global Unlock (PFREE)

PFREE PFREE

Program Cache Global Unlock

Operation:	**Assembler Syntax:**
Unlock all locked sectors	PFREE

Description: Unlock all the locked cache sectors in the instruction cache.

The PFREE instruction is enabled only in Cache Mode. In PRAM Mode it will cause an illegal instruction trap.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcode:

	23							16	15							8	7							0
PFREE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Instruction Fields: None

A-6.95 Lock Instruction Cache Relative Sector (PLOCKR)

PLOCKR PLOCKR

Lock Instruction Cache Relative Sector

Operation:	**Assembler Syntax:**
Lock sector by PC+xxxx	PLOCKR xxxx

Description: Lock the cache sector to which the sum PC + specified displacement belongs. If the sum does not belong to any cache sector, then load the 17 most significant bits of the sum into the least recently used cache sector tag, and then lock that cache sector. Update the LRU stack accordingly.

The displacement is a 2’s complement 24-bit integer that represents the relative distance from the current PC to the address to be locked.

The PLOCKR instruction is enabled only in Cache Mode. In PRAM Mode it will cause an illegal instruction trap.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

PLOCKR xxxx

23							16	15							8	7							0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
ADDRESS EXTENSION WORD																							

Instruction Fields: None

A-6.96 Unlock instruction Cache Sector (PUNLOCK)

PUNLOCK PUNLOCK

Unlock Instruction Cache Sector

Operation:	**Assembler Syntax:**
Unlock sector by effective address	PUNLOCK ea

Description: Unlock the cache sector to which the specified effective address belongs. If the specified effective address does not belong to any cache sector, and is therefore definitely unlocked, nevertheless, load the least recently used cache sector tag with the17 most significant bits of the specified address. Update the LRU stack accordingly. All memory alterable addressing modes may be used for the effective address, but not a short absolute address.

The PUNLOCK instruction is enabled only in Cache Mode. In PRAM Mode it will cause an illegal instruction trap.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23 16	15 8	7 0
PUNLOCK ea	0 0 0 0 1 0 1 0	1 1 M M M R R R	1 0 0 0 0 0 0 1
	ADDRESS EXTENSION WORD		

Instruction Fields:

{ea} **MMMRRR** Effective Address (see Table A-18 on page A-241)

A-6.97 Unlock instruction Cache Relative Sector (PUNLOCKR)

PUNLOCKR PUNLOCKR

Unlock Instruction Cache Relative Sector

Operation:

Unlock sector by PC+xxxx

Assembler Syntax:

PUNLOCKR xxxx

Description: Unlock the cache sector to which the sum PC + specified displacement belongs. If the sum does not belong to any cache sector, and is therefore definitely unlocked, nevertheless, load the least recently used cache sector tag with the 17 most significant bits of the sum. Update the LRU stack accordingly.

The displacement is a 2’s complement 24-bit integer that represents the relative distance from the current PC to the address to be locked.

The PUNLOCKR instruction is enabled only in Cache Mode. In PRAM Mode it will cause an illegal instruction trap.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

PUNLOCKR xxxx

23							16	15							8	7							0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
ADDRESS EXTENSION WORD																							

Instruction Fields: None

A-6.98 Repeat Next Instruction (REP)

REP REP

Repeat Next Instruction

Operation:	Assembler Syntax:
LC → TEMP; [X or y]:ea → LC Repeat next instruction until LC=1 TEMP → LC	REP [X or Y]:ea
LC → TEMP; [X or Y]:aa → LC Repeat next instruction until LC=1 TEMP → LC	REP [X or Y]:aa
LC → TEMP;S → LC Repeat next instruction until LC=1 TEMP → LC	REP S
LC → TEMP;#xxx → LC Repeat next instruction until LC=1 TEMP → LC	REP #xxx

Description: Repeat the **single-word instruction** immediately following the REP instruction the specified number of times. The value specifying the number of times the given instruction is to be repeated is loaded into the 24-bit loop counter (LC) register. The single-word instruction is then executed the specified number of times, decrementing the loop counter (LC) after each execution until LC=1. When the REP instruction is in effect, the repeated instruction is fetched only one time, and it remains in the instruction register for the duration of the loop count. Thus, **the REP instruction is not interruptible** (sequential repeats are also not interruptible). The current loop counter (LC) value is stored in an internal temporary register. If LC is set equal to zero, the instruction is repeated 65,536 times. The instruction's effective address specifies the address of the value which is to be loaded into the loop counter (LC). All address register indirect addressing modes may be used. The absolute short and the immediate short addressing modes may also be used. The four MS bits of the 12-bit immediate value are zeroed to form the 24-bit value that is to be loaded into the loop counter (LC).

If the system stack register SSH is specified as a source operand, the system stack pointer (SP) is postdecremented by 1 after SSH has been read.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	×	×	×	×
CCR							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23							16	15							8	7							0
REP [X or Y]:ea	0	0	0	0	0	1	1	0	0	1	M	M	M	R	R	R	0	S	1	0	0	0	0	0

	23							16	15							8	7							0
REP [X or Y]:aa	0	0	0	0	0	1	1	0	0	0	a	a	a	a	a	a	0	S	1	0	0	0	0	0

	23							16	15							8	7							0
REP #xxx	0	0	0	0	0	1	1	0	i	i	i	i	i	i	i	i	1	0	1	0	h	h	h	h

	23							16	15							8	7							0
REP S	0	0	0	0	0	1	1	0	1	1	d	d	d	d	d	d	0	0	1	0	0	0	0	0

Instruction Fields:

{ea}	**MMMRRR**	Effective Address (see Table A-19 on page A-242)
{X/Y}	**S**	Memory Space [X,Y] (see Table A-17 on page A-241)
{aa}	**aaaaaa**	Absolute Short Address
{#xxx}	**hhhhiiiiiiii**	Immediate Short Data
{S}	**dddddd**	Source register [all on-chip registers] (see Table A-22 on page A-243)

A-6.99 Reset On-Chip Peripheral Devices (RESET)

RESET RESET

Reset On-Chip Peripherals Devices

Operation:	**Assembler Syntax:**
Reset the interrupt priority register and all on-chip peripherals	RESET

Description: Reset the interrupt priority register and all on-chip peripherals. This is a **software reset** which is **NOT** equivalent to a hardware reset since only on-chip peripherals and the interrupt structure are affected. The processor state is not affected, and execution continues with the next instruction. All interrupt sources are disabled except for the stack error, NMI, illegal instruction, Trap, Debug request and hardware reset interrupts.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcode:

	23 16	15 8	7 0
RESET	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 0 0 0 0 1 0 0

Instruction Fields: None

A-6.100 Round Accumulator (RND)

RND RND

Round Accumulator

Operation:	**Assembler Syntax:**
D+r ➞ D (parallel move)	RND D (parallel move)

Description: Round the 56-bit value in the specified destination operand D and store the result in the destination accumulator (A or B). The contribution of the LS bits of the operand is rounded into the upper portion of the operand by adding a rounding constant to the LS bits of the operand. The upper portion of the destination accumulator contains the rounded result. The boundary between the lower portion and the upper portion is determined by the scaling mode bits S0 and S1 in the status register (SR).

Two types of rounding can be used: convergent rounding (also called round to nearest (even)) or two's complement rounding. The type of rounding is selected by the rounding mode bit (RM) in the MR portion of the status register.

In both these rounding modes a rounding constant is first added to the unrounded result. The value of the rounding constant added is determined by the scaling mode bits S0 and S1 in the status register (SR). A "1" is positioned in the rounding constant aligned with the most significant bit of the current LS portion, i.e. the rounding constant weight is actually equal to half the weight of the upper's portion least significant bit.

The following table shows the rounding position and rounding constant as determined by the scaling mode bits:

				Rounding Constant				
S1	**S0**	**Scaling Mode**	**Rounding Position**	**55 - 25**	**24**	**23**	**22**	**21 - 0**
0	0	No Scaling	23	0. . . .0	0	1	0	0. . . .0
0	1	Scale Down	24	0. . . .0	1	0	0	0. . . .0
1	0	Scale Up	22	0. . . .0	0	0	1	0. . . .0

Secondly, if convergent rounding is used, the result of this addition is tested and if all the bits of the result to the right of, and including, the rounding position are cleared, then the bit to the left of the rounding position is cleared in the result. This ensures that the result will not be biased.

Thirdly, in both rounding modes, the least significant bits of the result are cleared. The number of least significant bits cleared is determined by the scaling mode bits in the status register. All bits to the right of, and including, the rounding position are cleared in the result.

In Sixteen Bit Arithmetic mode the 40-bit value (in the 56-bit destination operand D) is rounded and stored in the destination accumulator (A or B). This implies that the boundary between the lower portion and upper portion is in a different position then in 24 bit mode. The following table shows the rounding position and rounding constant in sixteen bit arithmetic mode, as determined by the scaling mode bits:

			Rounding	Rounding Constant				
S1	**S0**	**Scaling Mode**	**Position**	**55 - 33**	**32**	**23**	**22**	**21 - 8**
0	0	No Scaling	31	0. . . .0	0	1	0	0. . . .0
0	1	Scale Down	32	0. . . .0	1	0	0	0. . . .0
1	0	Scale Up	30	0. . . .0	0	0	1	0. . . .0

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	✔	✔	✔	✔	✔	✕
CCR							

✔ This bit is changed according to the standard definition

✕ This bit is unchanged by the instruction

Instruction Formats and opcodes:

RND D

23 16 15 8	7							0
DATA BUS MOVE FIELD	0	0	0	1	d	0	0	1
OPTIONAL EFFECTIVE ADDRESS EXTENSION								

Instruction Fields:

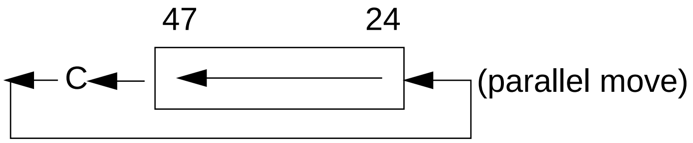
{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

A-6.101 Rotate Left (ROL)

ROL ROL

Rotate Left

Operation:



Assembler Syntax: ROL D (parallel move)

Description: Rotate bits 47–24 of the destination operand D one bit to the left and store the result in the destination accumulator.The carry bit receives the previous value of bit 47 of the operand.The previous value of the carry bit is shifted into bit 24 of the operand.This instruction is a 24-bit operation. The remaining bits of the destination operand D are not affected.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	●	●	●	●
CCR							

- ● N Set if bit 47 of the result is set
- ● Z Set if bits 47–24 of the result are zero
- ● V Always cleared
- ● C Set if bit 47 of the destination operand is set, cleared otherwise.
- ✔ This bit is changed according to the standard definition
- × This bit is unchanged by the instruction

Instruction Formats and opcodes:

ROL D

23 16 15 8	7	0
DATA BUS MOVE FIELD	0 0 1 1	d 1 1 1
OPTIONAL EFFECTIVE ADDRESS EXTENSION		

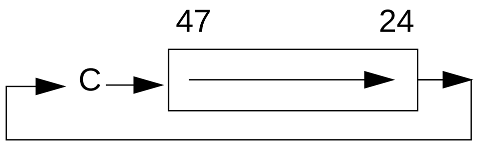
Instruction Fields:

{D}	**d**	Destination accumulator [A,B] (see Table A-10 on page A-239)

A-6.102 Rotate Right (ROR)

ROR ROR

Rotate Right

Operation: 

(parallel move)

Assembler Syntax: ROR D (parallel move)

Description: Rotate bits 47–24 of the destination operand D one bit to the right and store the result in the destination accumulator.The carry bit receives the previous value of bit 24 of the operand.The previous value of the carry bit is shifted into bit 47 of the operand. This instruction is a 24-bit operation. The remaining bits of the destination operand D are not affected.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	●	●	●	●
CCR							

● N Set if bit 47 of the result is set
● Z Set if bits 47–24 of the result are zero
● V Always cleared
● C Set if bit 24 of the destination operand is set, cleared otherwise.
✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Instruction Formats and opcodes:

		23 16 15 8	7	0
ROR	D	DATA BUS MOVE FIELD	0 0 1 0	d 1 1 1
		OPTIONAL EFFECTIVE ADDRESS EXTENSION		

Instruction Fields:

{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

A-6.103 Return from Interrupt (RTI)

RTI RTI

Return from Interrupt

Operation: **Assembler Syntax:**

SSH → PC; SSL → SR; SP–1 → SP RTI

Description: Pull the program counter (PC) and the status register (SR) from the system stack. The previous program counter and status register are lost.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
●	●	●	●	●	●	●	●
CCR							

- ● All All the Status Register bits are set according to the value pulled from the stack

Instruction Formats and opcode:

	23							16	15							8	7							0
RTI	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Instruction Fields: None

A-6.104 Return from Subroutine (RTS)

RTS RTS

Return from Subroutine

Operation:

SSH → PC; SP–1 → SP

Assembler Syntax:

RTS

Description: Pull the program counter (PC) from the system stack. The previous program counter is lost. The status register (SR) is not affected.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcode:

	23							16	15							8	7							0
RTS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0

Instruction Fields: None

A-6.105 Subtract Long with Carry (SBC)

SBC SBC

Subtract Long with Carry

Operation:	**Assembler Syntax:**
D–S–C → D (parallel move)	SBC S,D (parallel move)

Description: Subtract the source operand S and the carry bit C of the condition code register from the destination operand D and store the result in the destination accumulator. Long words (48 bits) are subtracted from the (56-bit) destination accumulator.

Note: The carry bit is set correctly for multiple-precision arithmetic using long-word operands if the extension register of the destination accumulator (A2 or B2) is the sign extension of bit 47 of the destination accumulator (A or B).

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	✔	✔	✔	✔	✔	✔
CCR							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

SBC S,D

23 16 15 8	7	0
DATA BUS MOVE FIELD	0 0 1 J	d 1 0 1
OPTIONAL EFFECTIVE ADDRESS EXTENSION		

Instruction Fields:

{S} **J** Source register [X,Y] (see Table A-11 on page A-239)

{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

A-6.106 Stop Instruction Processing (STOP)

STOP STOP

Stop Instruction Processing

Operation:	**Assembler Syntax:**
Enter the stop processing state and stop the clock oscillator	STOP

Description: Enter the STOP processing state. All activity in the processor is suspended until the $\overline{\text{RESET}}$,$\overline{\text{DE}}$ or $\overline{\text{IRQA}}$ pin is asserted or the Debug Request JTAG command is detected. The clock oscillator is gated off internally. The STOP processing state is a low-power standby state.

During the STOP state, the destination port is in an idle state with the control signals held inactive, the data pins are high impedance, and the address pins are unchanged from the previous instruction.

If the exit from the STOP state was caused by a low level on the $\overline{\text{RESET}}$ pin, then the processor will enter the reset processing state.

If the exit from the STOP state was caused by a low level on the $\overline{\text{IRQA}}$ pin, then the processor will service the highest priority pending interrupt and will not service the $\overline{\text{IRQA}}$ interrupt unless it is highest priority. If no interrupt is pending, the processor will resume program execution at the instruction following the STOP instruction that caused the entry into the STOP state. Program execution (interrupt or normal flow) will resume after an internal delay counter counts:

- If the Stop Delay (SD, OMR[6]) bit is cleared - 131,070 clock cycles
- If the Stop Delay (SD, OMR[6]) bit is set - 24 clock cycles
- If the STOP Processing State (PSTP, PCTL[17]) is set - 8.5 clock cycles

During the clock stabilization count delay, all peripherals and external interrupts are cleared and re-enabled/arbitrated at the end of the count interval. If the $\overline{\text{IRQA}}$ pin is asserted when the STOP instruction is executed, the clock will not be gated off, and only the internal delay counter will be started.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcode:

	23							16	15							8	7							0
STOP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1

Instruction Fields: None

A-6.107 Subtract (SUB)

SUB SUB

Subtract

Operation:	**Assembler Syntax:**
D–S → D (parallel move)	SUB S, D (parallel move)
D–#xx → D	SUB #xx, D
D–#xxxxxx → D	SUB #xxxxxx ,D

Description: Subtract the source operand from the destination operand D and store the result in the destination operand D. The source can be a register (word - 24 bits, long word - 48 bits or accumulator - 56 bits), short immediate (6 bits) or long immediate (24 bits).

When using 6-bit immediate data, the data is interpreted as an unsigned integer. That is, the 6 bits will be right aligned and the remaining bits will be zeroed to form a 24-bit source operand.

Note: The carry bit is set correctly using word or long-word source operands if the extension register of the destination accumulator (A2 or B2) is the sign extension of bit 47 of the destination accumulator (A or B). The carry bit is always set correctly using accumulator source operands.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✓	✓	✓	✓	✓	✓	✓	✓
CCR							

✓ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Instruction Formats and opcodes:

SUB S,D

23							16	15							8	7							0
DATA BUS MOVE FIELD																0	J	J	J	d	1	0	0
OPTIONAL EFFECTIVE ADDRESS EXTENSION																							

SUB #xx,D

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	1	i	i	i	i	i	i	1	0	0	0	d	1	0	0

SUB #xxxxxx,D

23							16	15							8	7							0
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	1	0	0	d	1	0	0
IMMEDIATE DATA EXTENSION																							

Instruction Fields:

{S}	**JJJ**	Source register [B/A,X,Y,X0,Y0,X1,Y1] (see Table A-14 on page A-240)
{D}	**d**	Destination accumulator [A/B] (see Table A-10 on page A-239)
{#xx}	**iiiiii**	6-bit Immediate Short Data
{#xxxxxx}		24-bit Immediate Long Data extension word

A-6.108 Shift Left and Subtract Accumulators (SUBL)

SUBL SUBL

Shift Left and Subtract Accumulators

Operation:

2∗D–S ➞ D (parallel move)

Assembler Syntax:

SUBL S,D (parallel move)

Description: Subtract the source operand S from two times the destination operand D and store the result in the destination accumulator. The destination operand D is arithmetically shifted one bit to the left, and a zero is shifted into the LS bit of D prior to the subtraction operation. The carry bit is set correctly if the source operand does not overflow as a result of the left shift operation. The overflow bit may be set as a result of either the shifting or subtraction operation (or both). This instruction is useful for efficient divide and decimation in time (DIT) FFT algorithms.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	✔	✔	✔	✔	●	✔
CCR							

● V Set if overflow has occurred in the result or if the MS bit of the destination operand is changed as a result of the instruction’s left shift

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23 16 15 8	7	0
SUBL S,D	DATA BUS MOVE FIELD	0 0 0 1	d 1 1 0
	OPTIONAL EFFECTIVE ADDRESS EXTENSION		

Instruction Fields:

{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

{S} 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

A-6.109 Shift Right and Subtract Accumulators (SUBR)

SUBR SUBR

Shift Right and Subtract Accumulators

Operation:

D/2 –S ➞ D (parallel move)

Assembler Syntax:

SUBR S,D (parallel move)

Description: Subtract the source operand S from one-half the destination operand D and store the result in the destination accumulator. The destination operand D is arithmetically shifted one bit to the right while the MS bit of D is held constant prior to the subtraction operation. In contrast to the SUBL instruction, the carry bit is always set correctly, and the overflow bit can only be set by the subtraction operation, and not by an overflow due to the initial shifting operation. This instruction is useful for efficient divide and decimation in time (DIT) FFT algorithms.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	✔	✔	✔	✔	✔	✔
CCR							

✔ This bit is changed according to the standard definition
× This bit is unchanged by the instruction

Instruction Formats and opcodes:

	23 16 15 8	7	0
SUBR S,D	DATA BUS MOVE FIELD	0 0 0 0	d 1 1 0
	OPTIONAL EFFECTIVE ADDRESS EXTENSION		

Instruction Fields:

{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)
{S} 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

A-6.110 Transfer Conditionally (Tcc)

Tcc Tcc

Transfer Conditionally

Operation:	**Assembler Syntax:**
If cc, then S1 → D1	Tcc S1,D1
If cc, then S1 → D1 and S2 → D2	Tcc S1,D1 S2,D2
If cc, then S2 → D2	Tcc S2,D2

Description: Transfer data from the specified source register S1 to the specified destination accumulator D1 if the specified condition is true. If a second source register S2 and a second destination register D2 are also specified, transfer data from address register S2 to address register D2 if the specified condition is true. If the specified condition is false, a NOP is executed.

The conditions that the term “**cc**” can specify are listed on Table A-42 on page A-250.

When used after the CMP or CMPM instructions, the Tcc instruction can perform many useful functions such as a “maximum value,” “minimum value,” “maximum absolute value,” or “minimum absolute value” function. The desired value is stored in the destination accumulator D1. If address register S2 is used as an address pointer into an array of data, the address of the desired value is stored in the address register D2. The Tcc instruction may be used after any instruction and allows efficient searching and sorting algorithms.

The Tcc instruction uses the internal data ALU paths and internal address ALU paths. The Tcc instruction does not affect the condition code bits.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcode:

Tcc S1,D1

23							16	15							8	7							0
0	0	0	0	0	0	1	0	C	C	C	C	0	0	0	0	0	J	J	J	d	0	0	0

Tcc S1,D1 S2,D2

23							16	15							8	7							0
0	0	0	0	0	0	1	1	C	C	C	C	0	t	t	t	0	J	J	J	d	T	T	T

Tcc S2,D2

23							16	15							8	7							0
0	0	0	0	0	0	1	0	C	C	C	C	1	t	t	t	0	0	0	0	0	T	T	T

Instruction Fields:

{cc}	**CCCC**	Condition code (see Table A-43 on page A-251)
{S1}	**JJJ**	Source register [B/A,X0,Y0,X1,Y1] (see Table A-24 on page A-243)
{D1}	**d**	Destination accumulator [A/B] (see Table A-10 on page A-239)
{S2}	**ttt**	Source address register [R0-R7]
{D2}	**TTT**	Destination Address register [R0-R7]

A-6.111 Transfer Data ALU Register (TFR)

TFR TFR

Transfer Data ALU Register

Operation:

S→D (parallel move)

Assembler Syntax:

TFR S,D (parallel move)

Description: Transfer data from the specified source data ALU register S to the specified destination data ALU accumulator D. TFR uses the internal data ALU data paths; thus, data does not pass through the data shifter/limiters. This allows the full 56-bit contents of one of the accumulators to be transferred into the other accumulator **without** data shifting and/or limiting. Moreover, since TFR uses the internal data ALU data paths, parallel moves are possible.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✔	✔	×	×	×	×	×	×
CCR							

✔ This bit is changed according to the standard definition

× This bit is unchanged by the instruction

Instruction Formats and opcodes:

TFR S,D

23 16 15 8	7							0
DATA BUS MOVE FIELD	0	J	J	J	d	0	0	1
OPTIONAL EFFECTIVE ADDRESS EXTENSION								

Instruction Fields:

{S} **JJJ** Source register [B/A,X0,Y0,X1,Y1] (see Table A-24 on page A-243)

{D} **d** Destination accumulator [A/B] (see Table A-10 on page A-239)

A-6.112 Software Interrupt (TRAP)

TRAP TRAP

Software Interrupt

Operation:

Begin trap exception process

Assembler Syntax:

TRAP

Description: Suspend normal instruction execution and begin TRAP exception processing. The interrupt priority level (I1,I0) is set to 3 in the status register (SR) if a long interrupt service routine is used.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcode:

	23							16	15							8	7							0
TRAP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Instruction Fields: None

A-6.113 Conditional Software Interrupt (TRAPcc)

TRAPcc TRAPcc

Conditional Software Interrupt

Operation: **Assembler Syntax:**

If cc then Begin software exception processing TRAPcc

Description:

If the specified condition is true, normal instruction execution is suspended and software exception processing is initiated. The interrupt priority level (I1,I0) is set to 3 in the status register if a long interrupt service routine is used. If the specified condition is false, instruction execution continues with the next instruction.

The conditions that the term “**cc**” may specify are listed on Table A-42 on page A-250.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcode:

	23							16	15							8	7							0
TRAPcc	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	C	C	C	C

Instruction Fields:

{cc} **CCCC** Condition code (see Table A-43 on page A-251)

A-6.114 Test Accumulator (TST)

TST TST

Test Accumulator

Operation:	**Assembler Syntax:**
S–0 (parallel move)	TST S (parallel move)

Description: Compare the specified source accumulator S with zero and set the condition codes accordingly. No result is stored although the condition codes are updated.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
✓	✓	✓	✓	✓	✓	●	×
CCR							

●	V	Always cleared
✓		This bit is changed according to the standard definition
×		This bit is unchanged by the instruction

Instruction Formats and opcodes:

TST S

23	16	15	8	7							0
DATA BUS MOVE FIELD				0	0	0	0	d	0	1	1
OPTIONAL EFFECTIVE ADDRESS EXTENSION											

Instruction Fields:

{S} **d** Source accumulator [A,B] (see Table A-10 on page A-239)

A-6.115 Wait for interrupt (WAIT)

WAIT WAIT

Wait for Interrupt or DMA request

Operation:	**Assembler Syntax:**
Disable clocks to the processor core and enter the WAIT processing state	WAIT

Description: Enter the low-power standby WAIT processing state. The internal clocks to the processor core and memories are gated off, and all activity in the processor is suspended until an unmasked interrupt occurs or an enabled DMA channel receives a request. The clock oscillator and the internal I/O peripheral clocks remain active. If WAIT is executed when an interrupt is pending, the interrupt will be processed; the effect will be the same as if the processor never entered the WAIT state. If WAIT is executed when the DMA is active, the effect will be the same as if the processor never entered the WAIT state. When an unmasked interrupt or external (hardware) processor RESET occurs, the processor leaves the WAIT state and begins exception processing of the unmasked interrupt or RESET condition. The processor will exit from the WAIT state also when a Debug Request ($\overline{DE}$) pin is asserted or when a Debug Request JTAG command is detected.

Condition Codes:

7	6	5	4	3	2	1	0
S	L	E	U	N	Z	V	C
×	×	×	×	×	×	×	×
CCR							

× This bit is unchanged by the instruction

Instruction Formats and opcode:

	23							16	15							8	7							0
WAIT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0

Instruction Fields: None

A-7 INSTRUCTION PARTIAL ENCODING

This section gives the encodings for (1) various groupings of registers used in the instruction encodings, (2) condition code combinations, (3) addressing, and (4) 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.

A-7.1 Partial Encodings for Use in Instruction Encoding

Table A-10. Destination Accumulator Encoding

D/	d/S/D
A	0
B	1

Table A-11. Data ALU Operands Encoding

S	J
X	0
Y	1

Table A-12. Data ALU Source Operands Encoding

S	JJ
X0	00
Y0	01
X1	10
Y1	11

Table A-13. Program Control Unit Register Encoding

Register	EE
MR	00
CCR	01
COM	10
EOM	11

Table A-14. Data ALU Operands Encoding

S	J J J
B/A*	001
X	010
Y	011
X0	100
Y0	101
X1	110
Y1	111

* 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.

Table A-15. Data ALU operands encoding

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.

Table A-16. Effective Addressing Mode Encoding #1

Effective Addressing Mode	MMMRRR
(Rn)-Nn	0 0 0 r r r
(Rn)+Nn	0 0 1 r r r
(Rn)-	0 1 0 r r r
(Rn)+	0 1 1 r r r
(Rn)	1 0 0 r r r
(Rn+Nn)	1 0 1 r r r
-(Rn)	1 1 1 r r r
Absolute address	1 1 0 0 0 0
Immediate data	1 1 0 1 0 0

“rrr” refers to an address register R0-R7

Table A-17. Memory/Peripheral Space

Space	S
X Memory	0
Y Memory	1

Table A-18. Effective Addressing Mode Encoding #2

Effective Addressing Mode	MMMRRR
(Rn)-Nn	0 0 0 r r r
(Rn)+Nn	0 0 1 r r r
(Rn)-	0 1 0 r r r
(Rn)+	0 1 1 r r r
(Rn)	1 0 0 r r r
(Rn+Nn)	1 0 1 r r r
-(Rn)	1 1 1 r r r
Absolute address	1 10 0 0 0

“rrr” refers to an address register R0-R7

Table A-19. Effective Addressing Mode Encoding #3

Effective Addressing Mode	MMMRRR
(Rn)-Nn	0 0 0 r r r
(Rn)+Nn	0 0 1 r r r
(Rn)-	0 1 0 r r r
(Rn)+	0 1 1 r r r
(Rn)	1 0 0 r r r
(Rn+Nn)	1 0 1 r r r
-(Rn)	1 1 1 r r r

“rrr” refers to an address register R0-R7

Table A-20. Effective Addressing Mode Encoding #4

Effective Addressing Mode	MMRRR
(Rn)-Nn	0 0 r r r
(Rn)+Nn	0 1 r r r
(Rn)-	1 0 r r r
(Rn)+	1 1 r r r

“rrr” refers to an address register R0-R7

Table A-21. 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 A-22. Six-Bit Encoding For all On-Chip Registers

Destination Register	D D D D D D / d d d d d d
4 registers in Data ALU	0001DD
8 accumulators in Data ALU	001DDD
8 address registers in AGU	010TTT
8 address offset registers in AGU	011NNN
8 address modifier registers in AGU	100FFF
1address register in AGU	101EEE
2 program controller register	110VVV
8 program controller registers	111GGG

See Table A-21 for the specific encodings.

Table A-23. 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 A-24. Data ALU Source Registers Encoding

S	J J J
B/A*	000
X0	100
Y0	101
X1	110
Y1	111

* 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.

Table A-25. AGU Address and Offset Registers Encoding

Dest. Addr. Reg. D	dddd
R0-R7	onnn
N0-N7	1nnn

Table A-26. Data ALU Multiply Operands Encoding #1

S1*S2	Q Q Q	S1*S2	Q Q Q
X0,X0	0 0 0	X0,Y1	1 0 0
Y0,Y0	0 0 1	Y0,X0	1 0 1
X1,X0	0 1 0	X1,Y0	1 1 0
Y1,Y0	0 1 1	Y1,X1	1 1 1

Note: Only the indicated S1*S2 combinations are valid.X1*X1 and Y1*Y1 are not valid.

Table A-27. Data ALU Multiply Operands Encoding #2

S	Q Q
Y1	00
X0	01
Y0	10
X1	11

Table A-28. Data ALU Multiply Operands Encoding #3

S	qq
X0	00
Y0	01
X1	10
Y1	11

Table A-29. Data ALU Multiply Sign Encoding

Sig n	k
+	0
-	1

Table A-30. Data ALU Multiply Operands Encoding #3

S1*S2	Q Q Q Q	S1*S2	Q Q Q Q
X0,X0	0 0 0 0	X0,Y1	01 0 0
Y0,Y0	0 0 0 1	Y0,X0	01 0 1
X1,X0	0 0 1 0	X1,Y0	01 1 0
Y1,Y0	0 0 1 1	Y1,X1	01 1 1
X1,X1	1 0 0 0	Y1,X0	1 1 0 0
Y1,Y1	1 0 0 1	X0,Y0	1 1 0 1
X0,X1	1 0 1 0	Y0,X1	1 1 1 0
Y0,Y1	1 0 1 1	X1,Y1	1 1 1 1

Table A-31. 5-Bit Register Encoding #1

D/S	ddddd / eeeee	D/S	ddddd / eeeee
X0	00100	B2	01011
X1	00101	A1	01100
Y0	00110	B1	01101
Y1	00111	A	01110
A0	01000	B	01111
B0	01001	R0-R7	10 r r r
A2	01010	N0-N7	11 n n n

“rrr”=Rn number, “nnn”=Nn number

Table A-32. Immediate Data ALU Operand Encoding

n	sssss	constant
1	00001	010000000000000000000000
2	00010	001000000000000000000000
3	00011	000100000000000000000000
4	00100	000010000000000000000000
5	00101	000001000000000000000000
6	00110	000000100000000000000000
7	00111	000000010000000000000000
8	01000	000000001000000000000000
9	01001	000000000100000000000000
10	01010	000000000010000000000000
11	01011	000000000001000000000000
12	01100	000000000000100000000000
13	01101	000000000000010000000000
14	01110	000000000000001000000000
15	01111	000000000000000100000000
16	10000	000000000000000010000000
17	10001	000000000000000001000000
18	10010	000000000000000000100000
19	10011	000000000000000000010000
20	10100	000000000000000000001000
21	10101	000000000000000000000100
22	10110	000000000000000000000010
23	10111	000000000000000000000001

Table A-33. Write Control Encoding

Operation	W
Read Register or Peripheral	0
Write Register or Peripheral	1

Table A-34. ALU Registers Encoding

Destination Register	D D D D
4 registers in Data ALU	01DD
8 accumulators in Data ALU	1DDD

See Table A-21 for the specific encodings.

Table A-35. X:R Operand Registers Encoding

S1, D1	f f	D2	F
X0	00	Y0	0
X1	01	Y1	1
A	10		
B	11		

Table A-36. R:Y Operand Registers Encoding

D1	e	S2, D2	f f
X0	0	Y0	00
X1	1	Y1	01
		A	10
		B	11

Table A-37. Single-Bit Special Register Encoding Tables

d	X:R Class II Opcode	R:Y Class II Opcode
0	A → X:<ea> , X0 → A	Y0 → A , A → Y:<ea>
1	B → X:<ea> , X0 → B	Y0 → B , B → Y:<ea>

Table A-38. X:Y: Move Operands Encoding Tables

X Effective Addressing Mode	MMRRR
(Rn)+Nn	01sss
(Rn)-	10sss
(Rn)+	11sss
(Rn)	00sss

where “sss” refers to an address register R0-R7

Y Effective Addressing Mode	mmrr
(Rn)+Nn	01tt
(Rn)-	10tt
(Rn)+	11tt
(Rn)	00tt

where “tt” refers to an address register R4-R7 or R0-R3 which is in the opposite address register bank from the one used in the X effective address

S1,D1	e e	S2,D2	f f
X0	00	Y0	00
X1	01	Y1	01
A	10	A	10
B	11	B	11

Table A-39. Signed/Unsigned partial encoding #1

ss/su/uu	ss
ss	00
su	10
uu	11
reserved	01

Table A-40. Signed/Unsigned partial encoding #2

su/uu	s
su	0
uu	1

Table A-41. 5-Bit Register Encoding

S1,D1	ddddd
M0-M7	00nnn
EP	01010
VBA	10000
SC	10001
SZ	11000
SR	11001
OMR	11010
SP	11011
SSH	11100
SSL	11101
LA	11110
LC	11111

where “nnn”=Mn number (M0-M7)

Table A-42. Condition Codes Computation Equations

	"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	$Z+(\overline{U}\bullet\overline{E})=0$

where

$\overline{U}$ denotes the logical complement of U,

$+$ denotes the logical OR operator,

$\bullet$ denotes the logical AND operator, and

$\oplus$ denotes the logical Exclusive OR operator

Table A-43. Condition Codes Encoding

Mnemonic	CCCC	Mnemonic	CCCC
CC(HS)	0000	CS(LO)	1000
GE	0001	LT	1001
NE	0010	EQ	1010
PL	0011	MI	1011
NN	0100	NR	1100
EC	0101	ES	1101
LC	0110	LS	1110
GT	0111	LE	1111

The condition code computation equations are listed on Table A-42

A-7.2 Parallel Instruction Encoding of the Operation Code

The operation code encoding for the instructions which allow parallel moves is divided into the multiply and nonmultiply instruction encodings shown in the following subsection.

A-7.2.1 Multiply Instruction Encoding

The 8-bit operation code for multiply instructions allowing parallel moves has different fields than the nonmultiply instruction's operation code.

The 8-bit operation code=**1QQQ dkkk** where

QQQ=selects the inputs to the multiplier (see Table A-26)
kkk = three unencoded bits k2, k1, k0
d = destination accumulator
d = 0 → A
d = 1 → B

Table A-44. Operation Code K0-2 Decode

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

A-7.2.2 NonMultiply 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 = **0JJJ Dkkk** where JJJ=1/2 instruction number
kkk=1/2 instruction number
D=0 → A
D=1 → B

Table A-45. Nonmultiply Instruction Encoding

JJJ	D = 0 Src Oper	D = 1 Src Oper	kkk							
			000	001	010	011	100	101	110	111
000	B	A	MOVE[1]	TFR	ADDR	TST	*	CMP	SUBR	CMPM
001	B	A	ADD	RND	ADDL	CLR	SUB	*	SUBL	NOT
010	B	A	—	—	ASR	LSR	—	—	ABS	ROR
011	B	A	—	—	ASL	LSL	—	—	NEG	ROL
010	X1 X0	X1 X0	ADD	ADC	—	—	SUB	SBC	—	—
011	Y1 Y0	Y1 Y0	ADD	ADC	—	—	SUB	SBC	—	—
100	X0_0	X0_0	ADD	TFR	OR	EOR	SUB	CMP	AND	CMPM
101	Y0_0	Y0_0	ADD	TFR	OR	EOR	SUB	CMP	AND	CMPM
110	X1_0	X1_0	ADD	TFR	OR	EOR	SUB	CMP	AND	CMPM
111	Y1_0	Y1_0	ADD	TFR	OR	EOR	SUB	CMP	AND	CMPM

Note: * = Reserved

1 = Special Case #1

Table A-46. Special Case #1

OPERCODE	Operation
00000000	MOVE
00001000	reserved