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:

	23							16	15							8	7							0
PLOCKR xxxx	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:	**Assembler Syntax:**
Unlock sector by PC+xxxx	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:

	23							16	15							8	7							0
PUNLOCKR xxxx	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:

Reset the interrupt priority register and all on-chip peripherals

Assembler Syntax:

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	**Rounding Constant**				
S1	**S0**	**Scaling Mode**	**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 Constant				
S1	**S0**	**Scaling Mode**	**Rounding 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:

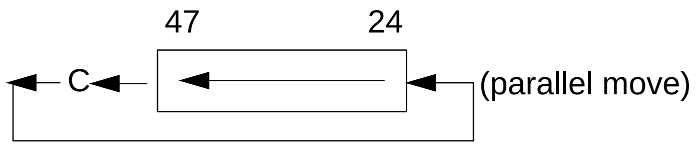
{D} **d** Destination accumulator [A,B] (see Table A-10 on page A-239)

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:

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

Instruction Fields:

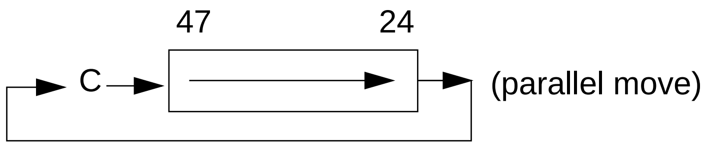
{D}	**d**	Destination accumulator [A,B] (see Table A-10 on page A-239)

A-6.102 Rotate Right (ROR)

ROR **ROR**

Rotate Right

Operation:



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:

SSH → PC; SSL → SR; SP–1 → SP

Assembler Syntax:

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:

	23 … 16 15 … 8	7 6 5 4	3 2 1 0
SBC S,D	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:

$2*D-S \rightarrow D$ (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:	**Assembler Syntax:**
D/2 –S → D (parallel move)	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:

		23 16	15 8	7 0
Tcc	S1,D1	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	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	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:	**Assembler Syntax:**
S→D (parallel move)	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:	**Assembler Syntax:**
Begin trap exception process	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:

If cc then Begin software exception processing

Assembler Syntax:

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:

S–0 (parallel move)

Assembler Syntax:

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:

	23 … 16	15 … 8	7 … 0
TST S	DATA BUS MOVE FIELD	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 ($\overline{\text{DE}}$) 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 $\rightarrow$ A
d = 1 $\rightarrow$ 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