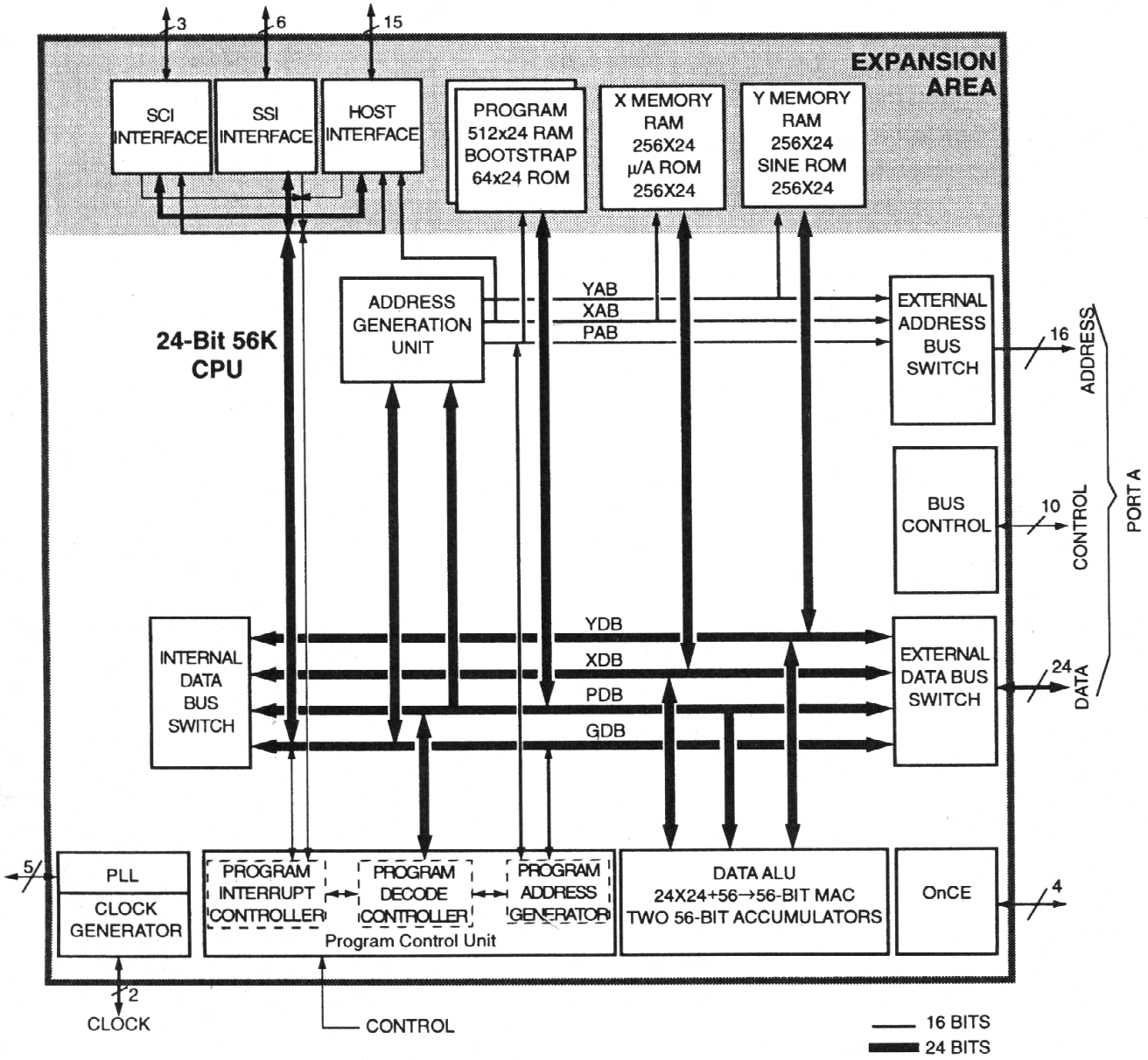
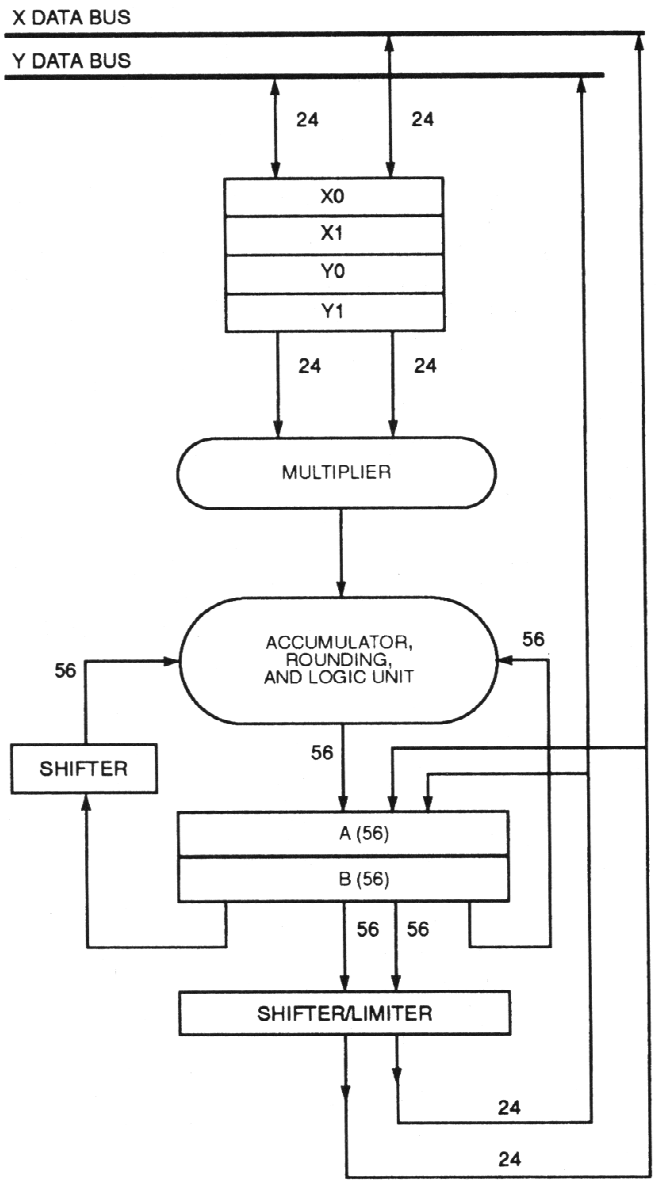
Motorola DSP 56000 Core

- Three independent Execution Units:
  - Data Arithmetic Logic Unit (data ALU)
  - Address Generation Unit (AGU)
  - Program Control Unit (PCU)
- Four independent 24-bit Data Buses:
  - X Data Bus (XDB)
  - Y Data Bus (YDB)
  - Global Data Bus (GDB)
- Three independent 16-bit Address Buses:
  - X Address Bus (XAB)
  - Y Address Bus (YAB)
  - Global Address Bus (GAB)


EXPANSION AREA
SCI INTERFACE
SSI INTERFACE
HOST INTERFACE
PROGRAM 512x24 RAM BOOTSTRAP 64x24 ROM
X MEMORY RAM 256X24 μ/A ROM 256X24
Y MEMORY RAM 256X24 SINE ROM 256X24
24-Bit 56K CPU
ADDRESS GENERATION UNIT
YAB
XAB
PAB
EXTERNAL ADDRESS BUS SWITCH
16
ADDRESS
BUS CONTROL
10
CONTROL
PORT A
INTERNAL DATA BUS SWITCH
YDB
XDB
PDB
GDB
EXTERNAL DATA BUS SWITCH
24
DATA
PLL
CLOCK GENERATOR
5
PROGRAM INTERRUPT CONTROLLER
PROGRAM DECODE CONTROLLER
PROGRAM ADDRESS GENERATOR
Program Control Unit
DATA ALU 24X24+56→56-BIT MAC TWO 56-BIT ACCUMULATORS
OnCE
4
2
CLOCK
CONTROL
16 BITS
24 BITS

Data Registers and MAC/ALU

- Data input registers: X1,X0; Y1,Y0
  - either 4 independent 24-bit registers: X0, X1, Y0, Y1
  - or 2 48-bit registers: X, Y
- Data accumulator registers: A2,A1,A0; B2,B1,B0
  - (A2,A1,A0) and (B2,B1,B0) form two 56-bit accumulator registers
  - A is A2:A1:A0 and B is B2:B1:B0
  - A2 & B2 are 8-bit “extensions,” rest are 24-bit
- MAC and Logic Unit
  - performs all calculations: $\times$, $\div$, $+$, $-$, AND, OR, NOT
  - up to 3 input operands,
  - result must be in A-Accumulator or B-Accumulator


X DATA BUS
Y DATA BUS
24
24
X0
X1
Y0
Y1
24
24
MULTIPLIER
ACCUMULATOR,
ROUNDING,
AND LOGIC UNIT
56
56
56
SHIFTER
A (56)
B (56)
56
56
SHIFTER/LIMITER
24
24

Two's Complement Fractional Data

- Bit weighting:
  Word: $|-2^0 \ldots 2^{-23}|$
  Long Word: $|-2^0 \ldots 2^{-23}|2^{-24} \ldots 2^{-47}|$
  Accumulator: $|-2^8 \ldots 2^1|2^0 \ldots 2^{-23}|2^{-24} \ldots 2^{-47}|$
- Example: 0.875 in 24-bit word
  Hexadecimal: $ 7000000,
  Binary: %011100000000000000000000
  $= 0 \times -2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$
- Example: -0.875 in 24-bit word
  Hexadecimal: $ 9000000,
  Binary: %100100000000000000000000
  $= 1 \times -2^0 + 1 \times 2^{-3}$
- To convert negative (MSB=1) 2's complement word to positive: negate bits and add 1
  - e.g. $100100\ldots00 \rightarrow 011011\ldots11 \rightarrow 011100\ldots00$

MAC and Logic Unit

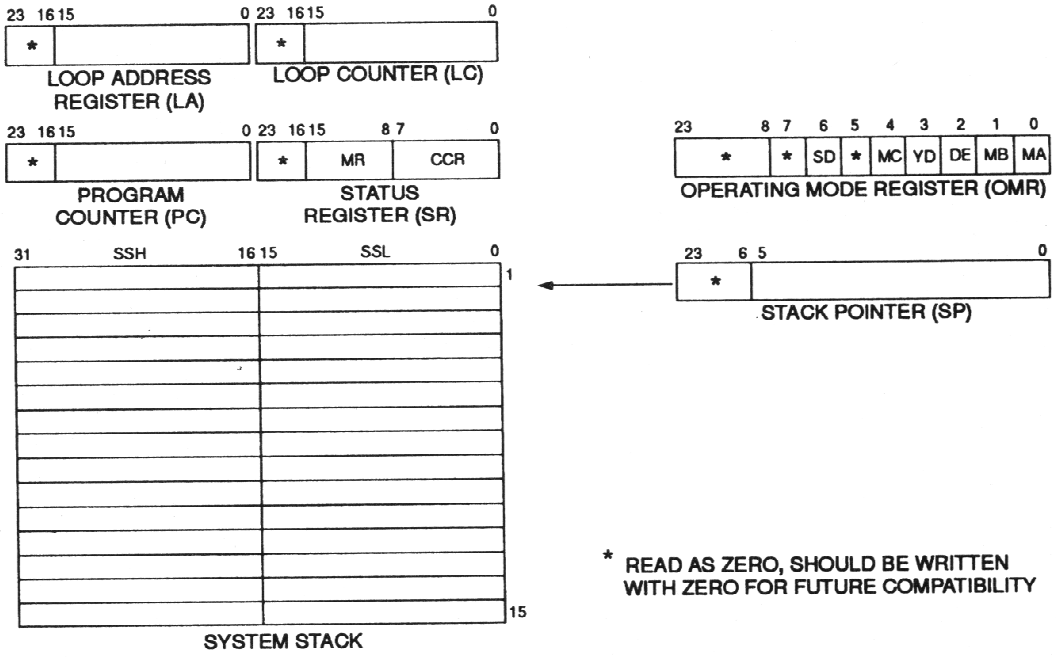
- addition, subtraction, multiplication
- logical AND, OR, NOT, XOR
- rounding
- left & right shift, up & down scaling
- limiting
- bit testing and clearing

Program Control

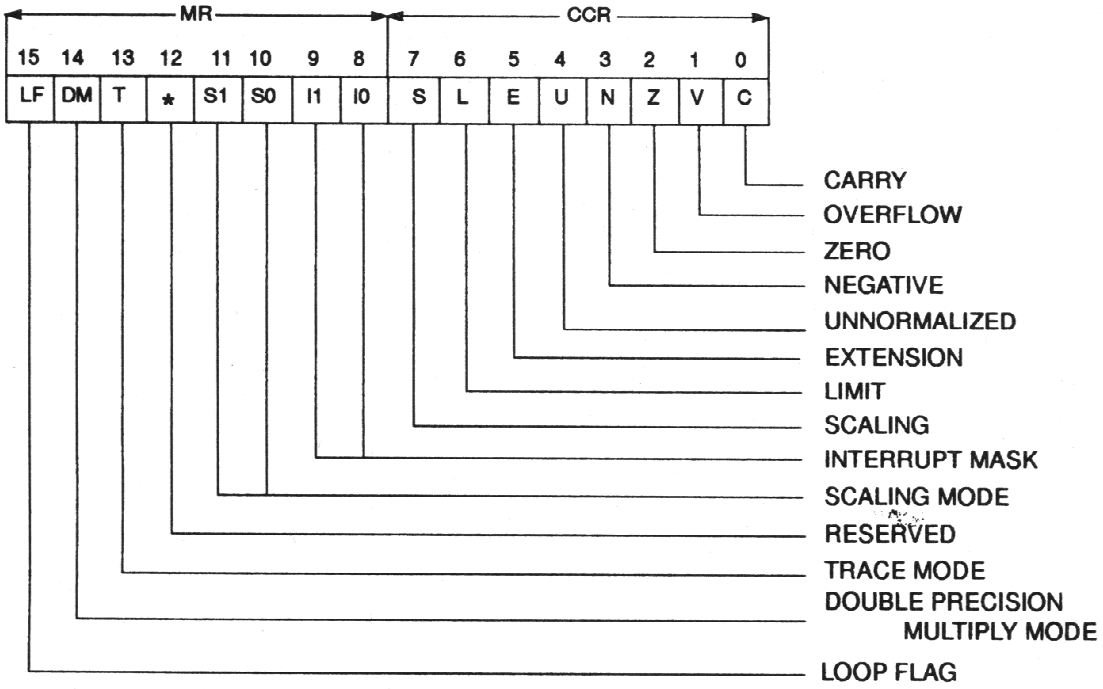
- Program Control Register (PC)
  - 16-bits (24 with upper 8 as 0)
  - points to Program Memory (P-Memory) of next instruction or immediate operand
- Status Register (SR)
  - 16-bits
  - MS 8 bits: Mode Register (MR), LS 8: Condition Code Register (CCR)
- System Stack (SS)
  - separate 32×15 internal memory
  - stores PC, SR for subroutines, interrupts, looping
  - can also store LC and LA
- Stack Pointer (SP)
  - 6 bits: LS 4 bits is pointer, MS 2 bits underflow and error flags

- Loop Counter (LC)
  - 16 bits
  - # times hardware DO or REPEAT is repeated
- Loop Address Register (LA)
  - points to last instruction in hardware DO
- Operating Mode Register (OMR)
  - various modes of DSP

PROGRAM CONTROL UNIT

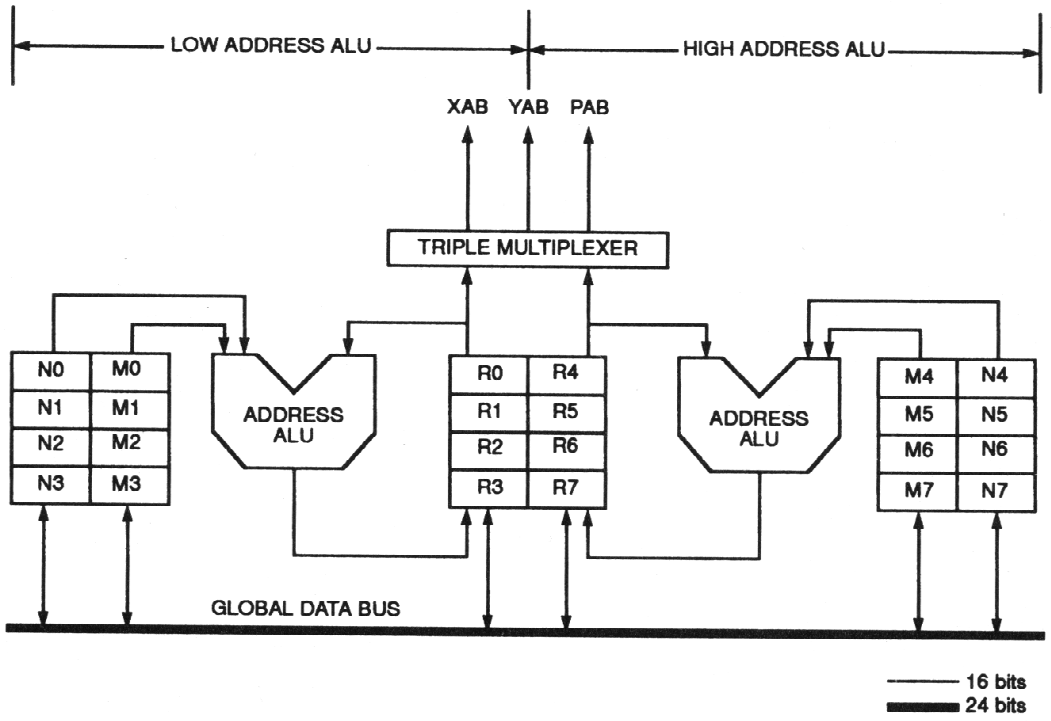


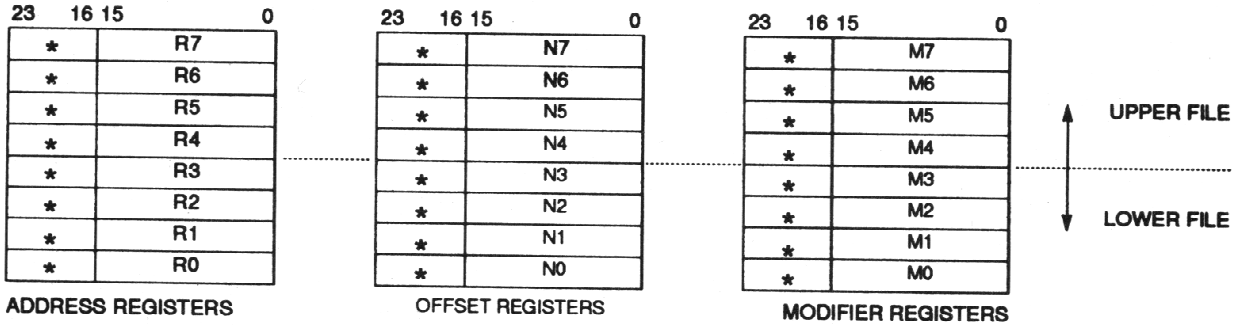
* READ AS ZERO, SHOULD BE WRITTEN WITH ZERO FOR FUTURE COMPATIBILITY


MR
CCR
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
LF DM T * S1 S0 I1 I0 S L E U N Z V C
CARRY
OVERFLOW
ZERO
NEGATIVE
UNNORMALIZED
EXTENSION
LIMIT
SCALING
INTERRUPT MASK
SCALING MODE
RESERVED
TRACE MODE
DOUBLE PRECISION MULTIPLY MODE
LOOP FLAG

Address Generation Unit

- 8 16-bit Address Registers R0. . . R7 that point to memory locations
- 8 16-bit Offset Registers N0. . . N7 that are added to the respective Rn to form the final address
- 8 16-bit Modifier Registers M0. . . M7 that specify type of arithmetic to update Address Register





* Written as don't care; read as zero

Modifier Encoding

Modifier	Address Arithmetic Type
0000	Reverse Carry (Bit Reverse)
0001	Modulo 2
0002	Modulo 3
⋮	⋮
7FFE	Modulo 32767
7FFF	Modulo 32768
8001	Multiple Wrap-around Modulo 2
8003	Multiple Wrap-around Modulo 2^2
8007	Multiple Wrap-around Modulo 2^3
800F	Multiple Wrap-around Modulo 2^4
801F	Multiple Wrap-around Modulo 2^5
⋮	⋮
9FFF	Multiple Wrap-around Modulo 2^{13}
BFFF	Multiple Wrap-around Modulo 2^{14}
FFFF	Linear

Addressing Modes

- Register Direct
  - refers to contents of a register
  - `move x1,a0`
  - x1 → a0
- Immediate
  - a value (constant) given in the instruction
  - `move #$818181,a0`
  - #$818181 → a0
- Absolute
  - address is specified in instruction
  - `move x:$fffe,a0`
  - x:$fffe → a0
  - can move to or from a register

- Register Indirect
  - contents of register specifies an address
  - many flavours to allow many types of indexing
- No Update RI
  - `move b1,y:(r0)`
  - y:(r0) $\rightarrow$ b1
- Post-increment by One RI
  - after address is used, add one to register Rn
  - `move b0,y:(r1)+`
  - y:(r1) $\rightarrow$ b0, r1+1 $\rightarrow$ r1
- Post-decrement by One RI
  - after address is used, subtract one from register Rn
  - `move b0,y:(r1)-`
  - y:(r1) $\rightarrow$ b0, r1-1 $\rightarrow$ r1
- Pre-decrement by One RI
  - before address is used, subtract one from register Rn

- `move b0,y:-(r1)`
- r1-1 → r1, y:(r1) → b0

- Post-increment by Offset Nn RI
  - after address is used, add register Nn to register Rn
  - `move b0,y:(r1)+n1`
  - y:(r1) → b0, r1+n1 → r1
  - also post-decrement
- Indexed with Offset Nn
  - address is sum of index register Rn and offset register Nn, but Rn remains unchanged
  - `move a1,x(r2+n2)`
  - x:(r2+n2) → a1

Instruction Set

- Instruction groups: Move, Arithmetic, Logical, Bit Manipulation, Loop, and Program Control
- Many instructions can specify one or two data transfers to be executed in parallel with opcode: “parallel-move” operations

Opcode	Operand Xfr	X-Bus Xfr	Y-Bus Xfr
MOVE	X1,A		
MOVE		X0,X:(R2)-	
MOVE			X0,Y:(R3)+N3
RND	A	X:(R0)+,R0	Y:(R4)+,Y0
MAC	X0,Y0,B	X:(R0)+,R0	Y:(R4)+,Y0
ADD	X,B	X:(R6)+,X1	Y0,Y:(R7)-

Parallel-Moves

Parallel-Move Operations	Opcode Operands	Parallel-Move Examples
Immediate	ADD B,A	#$81,B0
Reg. to Reg.	ADD B,A	X1,B
	ADD X1,A	B,R0
Addr. Reg. update	ADD B,A	(R1)+N1
X or Y Mem	ADD A,B	A,X:$1000
	ADD A,B	Y:-(R3),A
Y or Y Mem & reg	ADD X,A	A,X:(R3+N3) A,Y1
XY Mem	ADD X,B	X:(R0)+,X1 Y0,Y:(R7)-

U=Address Register Update
Y=Y-Memory Move
Y+R=Y-Memory & Register Move
X=X-Memory Move
X+R=X-Memory & Register Move
XY=XY-Memory Move

Addressing Modes	U	X Y	X+R Y+R	XY
Register Direct	no	no	no	no
Address Register Indirect				
No Update	no	yes	yes	yes
Postinc 1	yes	yes	yes	yes
Postdec 1	yes	yes	yes	yes
Postinc Nn	yes	yes	yes	yes
Postdec Nn	yes	yes	yes	no
Index by Nn	no	yes	yes	no
Predec 1	no	yes	yes	no
Special				
Immediate	no	yes	yes	no
Absolute	no	yes	yes	no
Immediate short	no	no	no	no

Move Instructions

LUA	Load updated address
MOVE	Move data
MOVEC or MOVE	Move control register
MOVEM or MOVE	Move program memory
MOVEP	Move peripheral data

Arithmetic Instructions

ABS	Absolute value
ADD	Add
ADDR	Shift right then add
ASR	Arithmetic shift right
CMP	Compare
DEC	Decrement by one
INC	Increment by one
MACR	Multiply-accumulate & round
MPYR	Multiply & round
NORM	Normalize iteration
SBC	Subtract with carry
SUBL	Shift left then subtract
Tcc	Transfer conditionally
TST	Test

ADC	Add with carry
ADDL	Shift left then add
ASL	Arithmetic shift left
CLR	Clear
CMPM	Compare Magnitude
DIV	Divide Iteration
MAC	Multiply-accumulate
MPY	Multiply
NEG	Negate
RND	Round
SUB	Subtract
SUBR	Shift right then subtract
TSF	Transfer

Addition and Subtraction Instructions

Mnemonic	Operation	Syntax	Src S	Dest D
ADC	S+D+C → D	ADC S,D [PM...]	X,Y	A,B
ADD	S+D → D	ADD S,D [PM...]	A	B
	B	A		
	X,X1,X0	A,B		
	Y,Y1,Y0	A,B		
ADDL	S+2D → D	ADDL S,D [PM...]	A	B
	B	A		
ADDR	S+D/2 → D	ADDR S,D [PM...]	A	B
	B	A		
SBC	D-S-C → D	SBC S,D [PM...]	X,Y	A,B
SUB	D-S → D	SUB S,D [PM...]	A	B
	B	A		
	X,X1,X0	A,B		
	Y,Y1,Y0	A,B		
SUBL	2D-S → D	SUBL S,D [PM...]	A	B
	B	A		
SUBR	D/2-S → D	SUBR S,D [PM...]	A	B
	B	A		

Multiply (and Accumulate) Instructions

Mnemonic	Operation	Syntax
MAC	D$\pm$(S1$\times$S2)$\rightarrow$D	MAC $\pm$S1,S2,D [PM...]
	D$\pm$(S1$\times 2^{-n}$)$\rightarrow$D	MAC $\pm$S1,#n,D [PM...]
MACR	round(D$\pm$(S1$\times$S2))$\rightarrow$D	MACR $\pm$S1,S2,D [PM...]
	round(D$\pm$(S1$\times 2^{-n}$))$\rightarrow$D	MACR $\pm$S1,#n,D [PM...]
MPY	$\pm$(S1$\times$S2)$\rightarrow$D	MPY $\pm$S1,S2,D [PM...]
	$\pm$(S1$\times 2^{-n}$)$\rightarrow$D	MPY $\pm$S1,#n,D [PM...]
MPYR	round($\pm$(S1$\times$S2))$\rightarrow$D	MPYR $\pm$S1,S2,D [PM...]
	round($\pm$(S1$\times 2^{-n}$))$\rightarrow$D	MPYR $\pm$S1,#n,D [PM...]

with S1 & S2 being any combination of X0,X1,Y0,Y1

Logical Instructions

AND	Logical AND	ANDI	AND immediate control register
EOR	Logical exclusive OR	LSL	Logical shift left
LSR	Logical shift right	NOT	Complement
OR	Logical inclusive OR	ORI	OR immediate control register
ROL	Rotate left	ROR	Rotate right

Bit Manipulation Instructions

BCLR	Bit test and clear	BSET	Bit set
BCHG	Bit test and change	BTST	Bit test on memory and

Program Control Instructions

DO	Hardware DO loop
DEBUG	Enter debug mode
Ill	Illegal instruction
JMP	Jump
JSET	Jump if bit set
JSR	Jump subroutine
JSSET	Jump subroutine if bit set
REP	Repeat next instruction
RTI	Return from interrupt
STOP	Stop processing
WAIT	Wait for interrupt

ENDO	Terminates hardware loop
DEBUGcc	Enter debug mode conditionall
Jcc	Jump conditionally
JCLR	Jump if bit clear
JScc	Jump to subroutine conditional
JSCLR	Jump subroutine if bit clear
NOP	No operation
RESET	Reset on-chip peripherals
RTS	Return from subroutine
SWI	Software interrupt

Hardware DO example

```
    for (i=0; i<3; ++i) {
        c[i]=a[i]*b[i];
    }
```

a is X:$1000, b is Y:$2000, and c is X:$1500

```
  org    p:$100
  move   #$1000,r0
  move   #$2000,r4
  move   #$1500,r1
  move                x:(r0),x0
  move                                y:(r4),y0
  do     #3,_endloop
  mpyr   x0,y0,a      x:(r0)+,x0      y:(r4)+,y0
  nop
  move                a,x(r1)+
  nop
_endloop
```

REP Instruction

Mnemonic	Operation	Syntax
REP	LC $\rightarrow$ Temp; #xxx $\rightarrow$ LC	REP #xxx
	Repeate next instruction until LC = 1	REP X:¡ea¿
	Temp $\rightarrow$ LC	REP Y:¡ea¿
		REP S

Example: filter $y[n] = \sum_{k=0}^{M-1} b_k x[n-k]$

```
clr     a           x0,x:(r1)+   y:(r4)+,y0
rep     #ntaps-1
mac     x0,y0,a     x:(r1)+,x0   y:(r4)+,y0
macr    x0,y0,a     (r1)-
move    a,x0
```

```
;*****************************************************************
;pass.asm : PASSES AUDIO STRAIGHT THROUGH DSP56307EVM
;*****************************************************************
;
      nolist
      include 'ioequ.asm'
      include 'intequ.asm'
      include 'ada_equ.asm'
      include 'vectors.asm'
      list

;*****************************************************************
;Buffers for talking to the CS4218
;*****************************************************************
;
       org    x:0
RX_BUFF_BASE    equ     *
RX_data_1_2     ds      1       ; data time slot 1/2 for RX ISR (left audio)
RX_data_3_4     ds      1       ; data time slot 3/4 for RX ISR (right audio)

TX_BUFF_BASE    equ     *
TX_data_1_2     ds      1       ; data time slot 1/2 for TX ISR (left audio)
TX_data_3_4     ds      1       ; data time slot 3/4 for TX ISR (right audio)

RX_PTR          ds      1       ; pointer for RX buffer
TX_PTR          ds      1       ; pointer for TX buffer

CTRL_WD_12      equ     MIN_LEFT_ATTN+MIN_RIGHT_ATTN+LIN2+RIN2
CTRL_WD_34      equ     MIN_LEFT_GAIN+MIN_RIGHT_GAIN
```

```
;*****************************************************************
;Main Program
;*****************************************************************
        org     p:$400
START
        movep   #$040006,x:M_PCTL  ; PLL 7 X 12.288 = 86.016MHz
        movep   #$012421,x:M_BCR   ; AARx - 1 wait state
        ori     #3,mr              ; mask interrupts
        movec   #0,sp              ; clear hardware stack pointer
        move    #0,omr             ; operating mode 0
        move    #$40,r6            ; initialise stack pointer
        move    #-1,m6             ; linear addressing
        jsr     ada_init           ; initialize codec
loop
        jset    #3,x:M_SSISR0,*    ; wait for RX frame sync
        jclr    #3,x:M_SSISR0,*    ; wait for RX frame sync

        move    x:RX_BUFF_BASE,a   ; receive left
        move    x:RX_BUFF_BASE+1,b ; receive right

        nop                        ; pass data straight through
        nop

        move    a,x:TX_BUFF_BASE   ; transmit left
        move    b,x:TX_BUFF_BASE+1 ; transmit right
        jmp     loop

        include 'ada_init.asm'
        end
```