COLDFIRE®

CPU

COLDFIRE ® FEATURES

- **SIMPLE INSTRUCTION SET ARCHITECTURE**

  - VARIABLE LENGTH RISC
  - OPTIMIZED FOR HIGH LEVEL LANGUAGE CONSTRUCTS
  - 16 USER-VISIBLE 32-BIT WIDE REGISTERS
  - SUPERVISOR/USER MODES FOR SYSTEM PROTECTION
  - VECTOR BASE REGISTER TO RELOCATE EXCEPTION VECTOR TABLE

- **MAC MODULE SUPPORTS 16/32 MULTIPLY-ACCUMULATE**

- **DYNAMIC BUS SIZING**

  - SUPPORTS 32-, 16-, AND 8-BIT PORTS
  - SINGLE BUS CLOCK INPUT
  - 256MB OF OFF-CHIP LINEAR ADDRESS SPACE
  - MEMORY MAPPED-I/O
  -
  - RISC ARCHITECTURE THAT ALLOWS OPERAND MANIPULATION IN MEMORY

- **ON-BOARD DEBUG MODULE**
  - ALLOWS FOR BACKGROUND DEBUG
  - ALLOWS FOR REAL TIME DEBUG

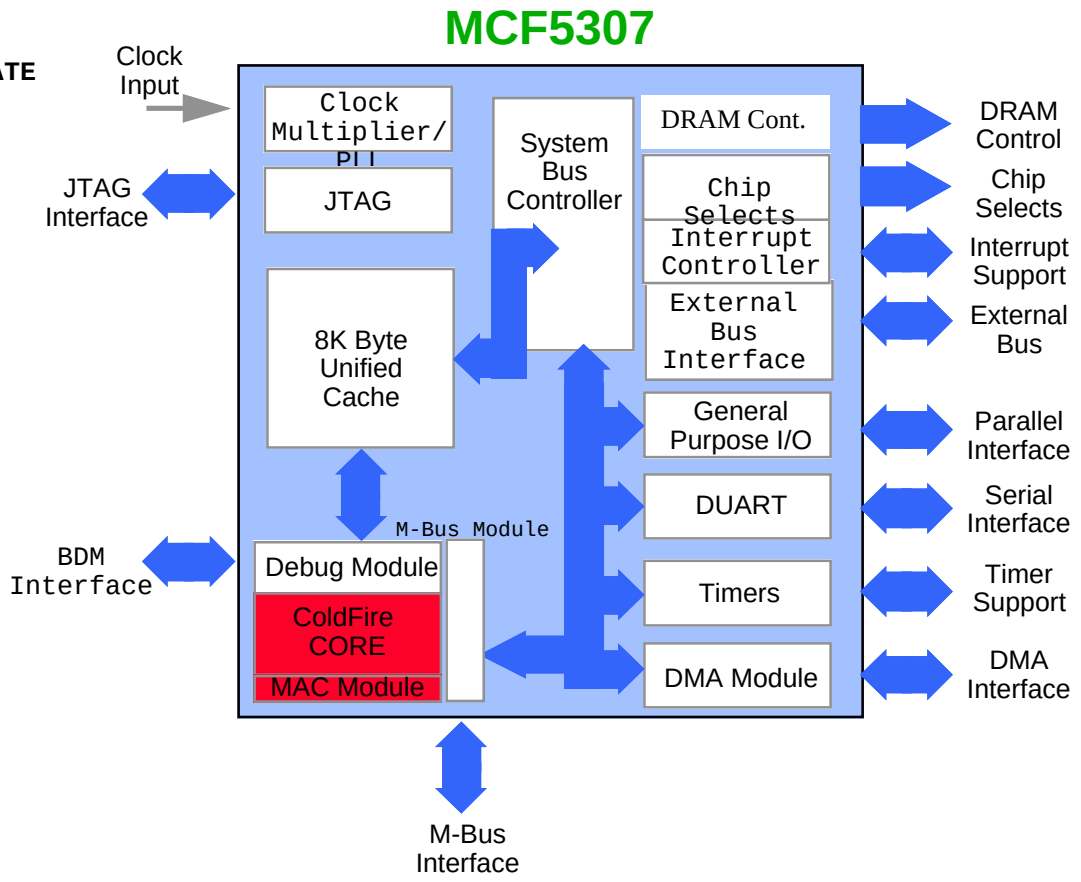
- **ON BOARD JTAG INTERFACE**

- **LOW INTERRUPT LATENCY**

- **FULLY STATIC OPERATION**

- **UP TO 100MHZ CORE CLOCK FREQUENCY**

EXTERNAL BUS FREQUENCY IS PROGRAMMABLE FOR:
- 1/2, 1/3, & 1/4 OF CORE CLOCK FREQUENCY



INSTRUCTION PIPELINE

- **UPWARD COMPATIBLE WITH VERSION 2 ColdFire**

- **ENHANCED THREE STAGE INSTRUCTION PIPLINE FOR BRANCH PREDICTION**

- **SIMPLE INSTRUCTION SET ARCHITECTURE**

  - VARIABLE LENGTH RISC
  - OPTIMIZED FOR HIGH LEVEL LANGUAGE CONSTRUCTS
  - 16 USER-VISIBLE 32-BIT WIDE REGISTERS
  - SUPERVISOR/USER MODES FOR SYSTEM PROTECTION
  - VECTOR BASE REGISTER TO RELOCATE EXCEPTION VECTOR TABLE

- **EXTERNAL BUS INTERFACE**

  - SUPPORTS 32-, 16-, AND 8-BIT PORTS
  - SINGLE BUS CLOCK INPUT
  - 256MB OF OFF-CHIP LINEAR ADDRESS SPACE
  - MEMORY MAPPED-I/O
  -
  - RISC ARCHITECTURE THAT ALLOWS OPERAND MANIPULATION IN MEMORY

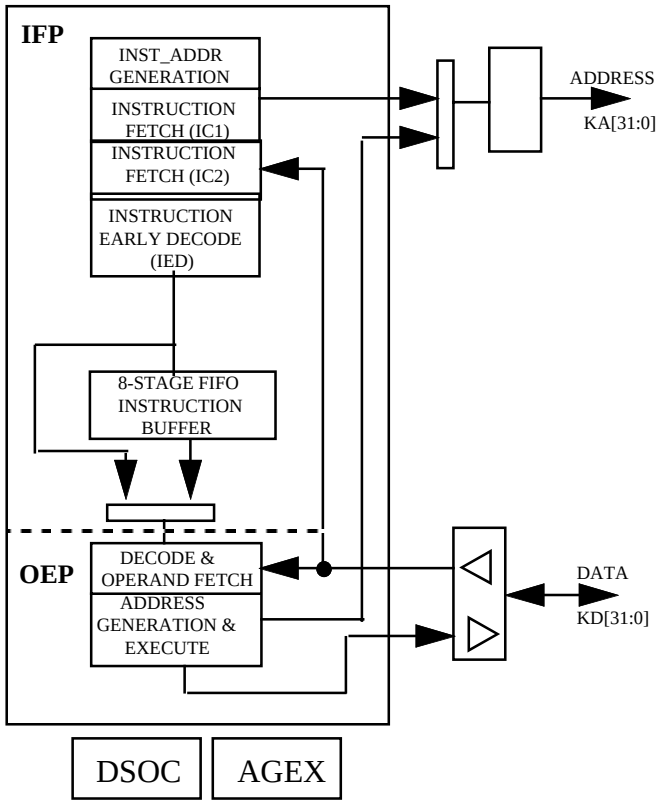
- **ON-BOARD DEBUG MODULE**
  - ALLOWS FOR BACKGROUND DEBUG
  - ALLOWS FOR REAL TIME DEBUG
  - ALLOWS FOR REAL TIME TRACE

- **ON BOARD JTAG INTERFACE**

- **LOW INTERRUPT LATENCY**

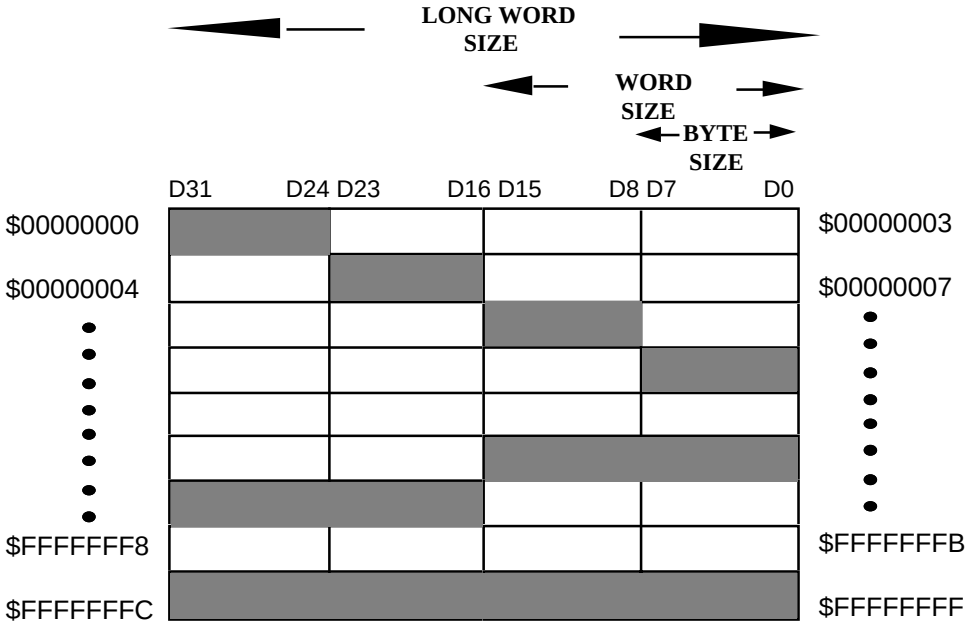
- **FULLY STATIC OPERATION**

IAG -Instruction Address Generation
IC1 - Instruction Cycle 1(Address Driven on KBus)
IC2 - Instruction Cycle 2 (Instruction is loaded in IC2)
IED Instruction Early Decode (Partial Decode is started)
IB- Instruction Buffer (Opcode is loaded in the next entry in 8-stage Buff)



DSOC - Decode & Operand Fetch
AGEX - Operand Address Generation/Execute Stage

MEMORY ORGANIZATION

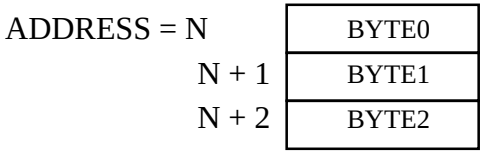


Instruction, Data Misalignment not supported.

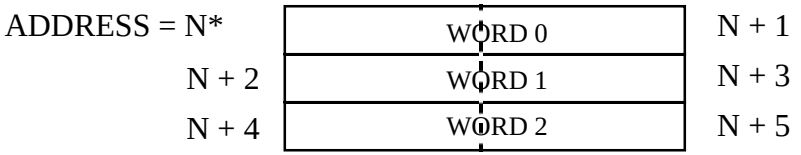
Note: Due to dynamic bus sizing, 8 bit devices will appear as sequential

MEMORY DATA FORMAT

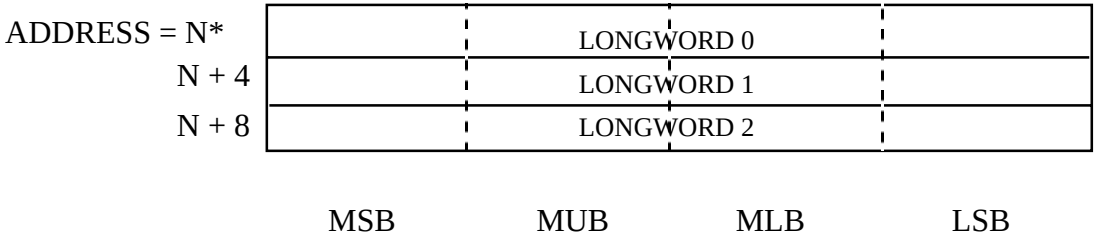
1 BYTE = 8 BITS



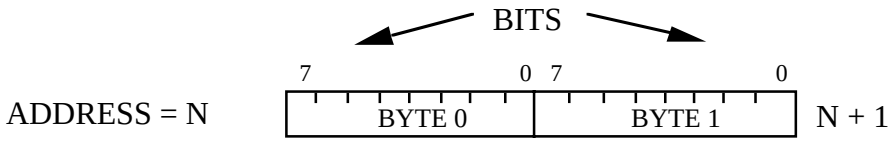
1 WORD = 16 BITS



1 LONG WORD = 32 BITS

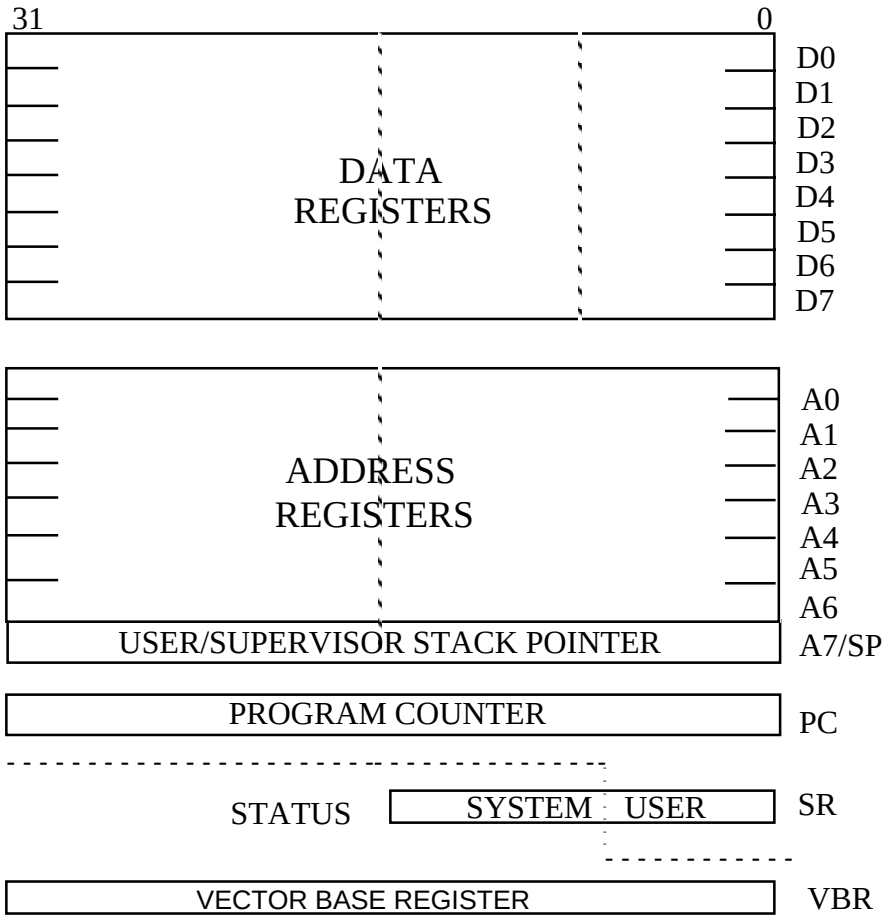


1 BIT

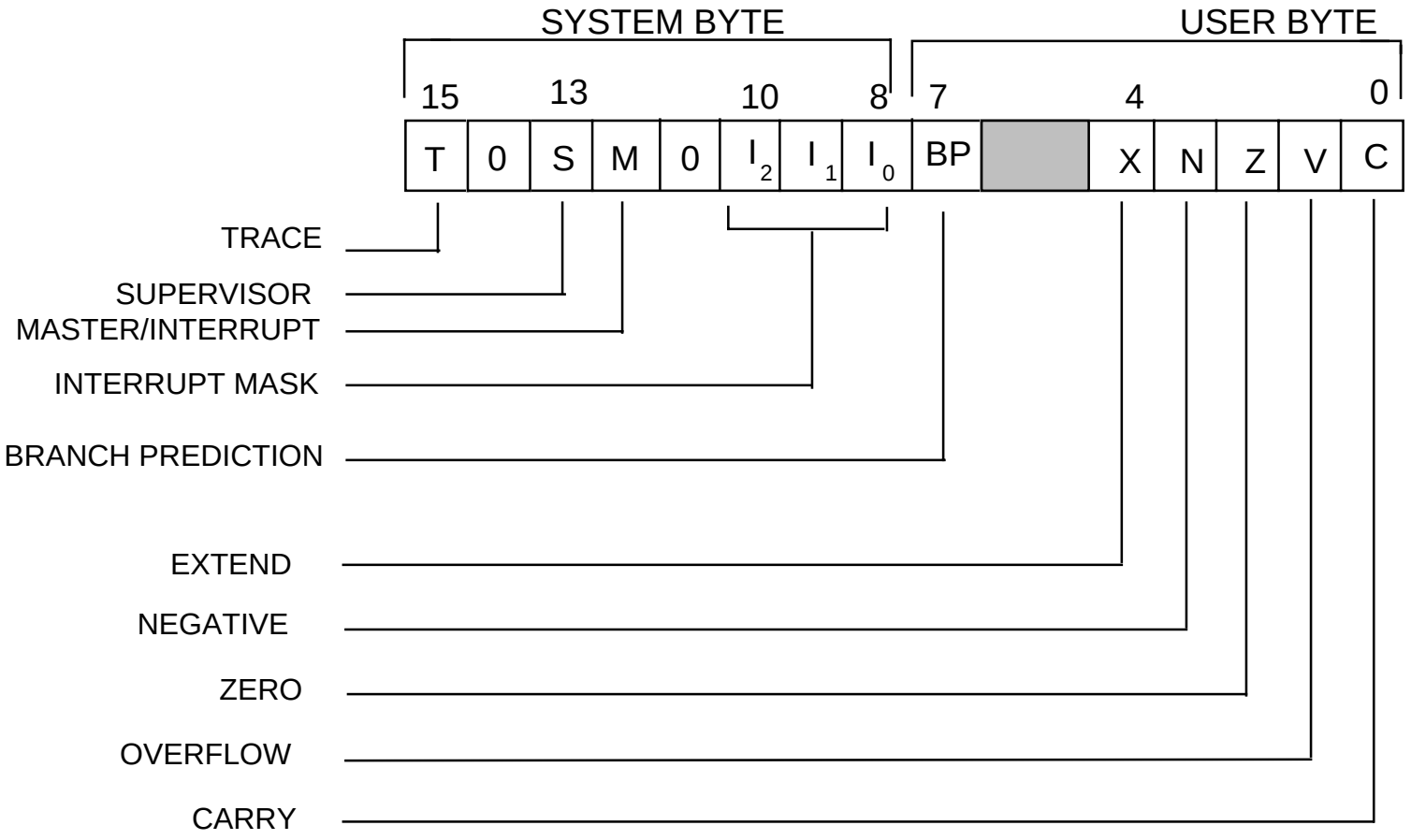


* N IS AN EVEN NUMBER (LONGWORD accesses must be LONGWORD aligned)

PROGRAMMING MODEL



STATUS REGISTER

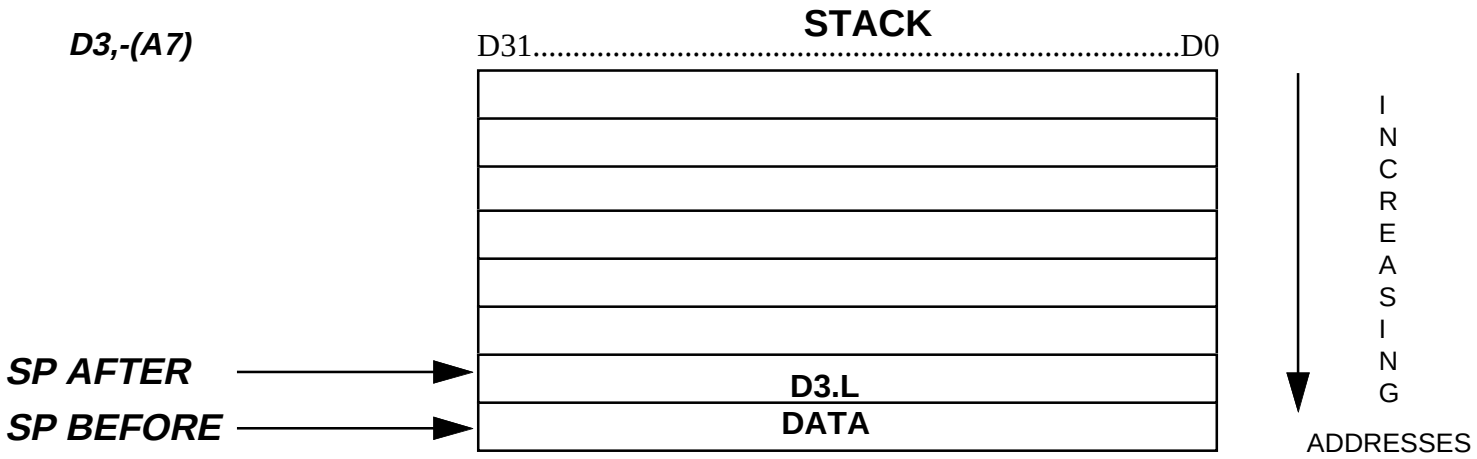


UNUSED BITS ARE ZERO

STACK OPERATION

EXAMPLE:

MOVE.L D3,-(A7)

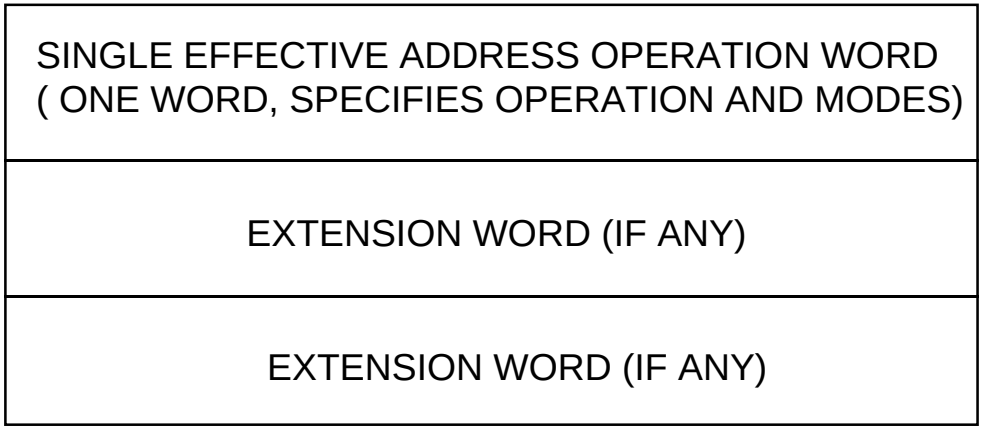


ADDRESSING MODES

COLDFIRE® ADDRESSING MODES

FEATURES:

- SIMPLE ADDRESSING MODES
- VARIABLE LENGTH REDUCED INSTRUCTION SET (RISC)
- INSTRUCTION LENGTH 1, 2, OR 3 WORDS
- OPERAND MANIPULATION IN MEMORY INCREASES CODE DENSITY
- EFFICIENT NUMBER OF REGISTER FOR OPERAND MANIPULATION
- FAST 1-CLOCK EXECUTION FOR MANY INSTRUCTIONS



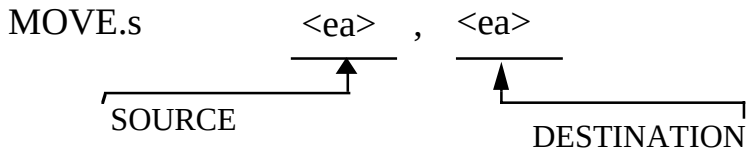
INSTRUCTION WORD GENERAL FORMAT

ASSEMBLY LANGUAGE FORMAT

- **Instructions contain 3 kinds of information:**

  1. Type of function to be performed.
  2. The size of the operand(s).
  3. The location of operand(s).

EXAMPLE:



- WHERE 's' SPECIFIES LENGTH OF DATA BEING TRANSFERRED

  B – BYTE

  W – WORD

  L – LONG WORD

ADDRESSING MODES

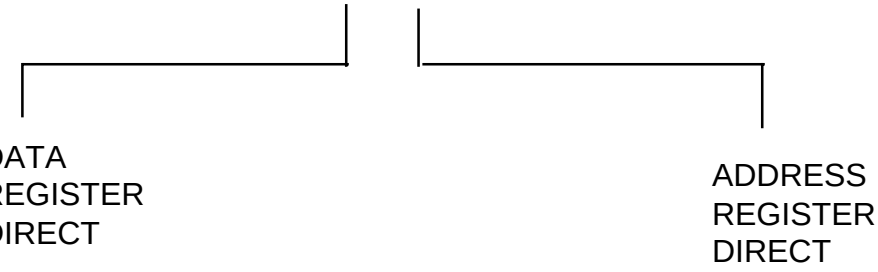
Assembly Language Notation	Addressing Mode Name
Dn An	Data Register Direct Address Register Direct
(An) (An) + -(An) d16(An) (d8,An,Xn.SIZE*SCALE)	Address Register Indirect (ARI) ARI with Postincrement ARI with Predecrement ARI with Displacement ARI with Index (8 bit displacement)
xxx.W xxx.L	Absolute Short Absolute Long
d16 (PC) (d8,PC,Xn.SIZE*SCALE) Immediate	PC Relative with Displacement PC Relative with Index (8 bit disp)

REGISTER DIRECT ADDRESSING MODE

USED TO ACCESS INTERNAL REGISTERS

EA = Dn OR An

EXAMPLE: MOVE.S D1,A2 INSTRUCTION LENGTH = 1 WORD



WHERE .S = WORD OR LONG WORD

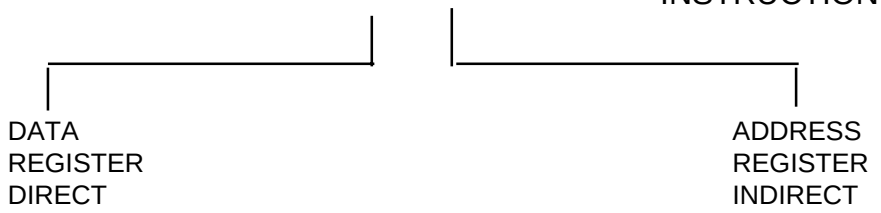
REGISTER INDIRECT ADDRESSING MODE

USED AS VARIABLE REFERENCE POINTER TO MEMORY

EA = (An)

EXAMPLE: MOVE.S D1,(A2)

INSTRUCTION LENGTH = 1 WORD




```
        MOVE.L   #COUNT,D0
LOOP    MOVE.L   (A1),(A2)
        ADDA.L   #4,A1
        ADDA.L   #4,A2
        SUB.L    #1,D0
        BNE      LOOP
        –
        –
```

INIT LOOP COUNTER
MOVE DATA
ADVANCE POINTERS TO
TO NEXT ENTRIES
DECREMENT LOOP COUNTER
LOOP IF NOT DONE

WHERE .S = BYTE, WORD OR LONG WORD

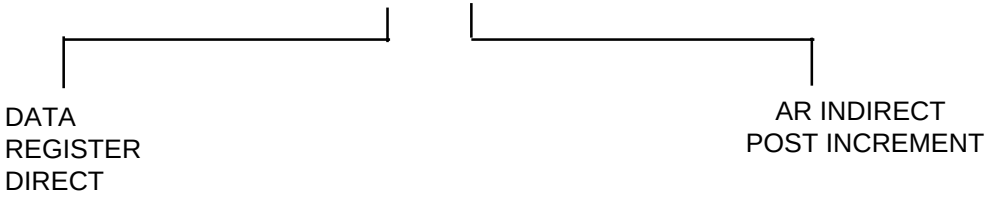
REGISTER INDIRECT POST INCREMENT

USED AS VARIABLE REFERENCE POINTER TO ACCESS SEQUENTIAL DATA IN MEMORY

EA = (An)+ An IS INCREMENTED AFTER THE OPERAND IS ACCESSED

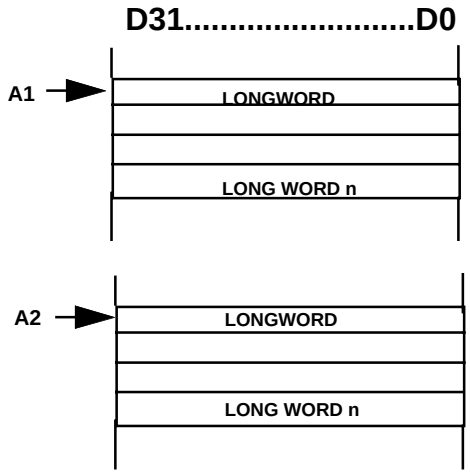
THIS MODE MAY BE USED FOR REGISTER UNSTACKING

EXAMPLE: MOVE.S D1,(A2)+ INSTRUCTION LENGTH = 1 WORD



PROGRAM EXAMPLE:

```
        MOVE.L    #COUNT,D0   /INIT LOOP COUNTER
LOOP    MOVE.L    (A1)+,(A2)+ /MOVE DATA
        SUB.L     #1,D0       /DECREMENT COUNTER
        BNE       LOOP        /IF NOT DONE LOOP
        _
        _
```



WHERE .S = BYTE, WORD OR LONG WORD

REGISTER INDIRECT PREDECREMENT

USED AS VARIABLE REFERENCE POINTER TO ACCESS SEQUENTIAL DATA IN MEMORY

THIS MODE MAY BE USED FOR REGISTER STACKING

EA = -(An)

An IS DECREMENTED BEFORE THE OPERAND IS ACCESSED

EXAMPLE: MOVE.S D1,-(A2) INSTRUCTION LENGTH = 1 WORD

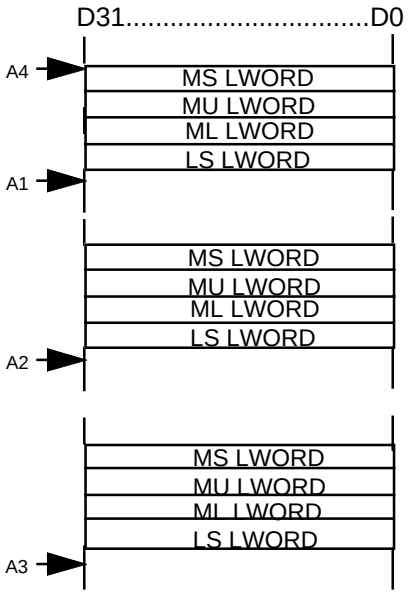
DATA REGISTER DIRECT

AR INDIRECT PREDECREMENT

PROGRAM EXAMPLE:

```
        MOVE.W    #0,CCR
LOOP    MOVE.L    -(A1),D1     /GET 1st. OPERAND
        MOVE.L    -(A2),D2     /GET 2nd. OPERAND
        ADDX.L    D1,D2        /ADD EXTENDED
        MOVE.L    D2,-(A3)     /SAVE RESULT
        CMP.L     A1,A4        /ARE WE DONE
        BLS       LOOP         /IF NOT DONE LOOP
        _
        _
```

WHERE .S = BYTE, WORD OR LONG WORD



REGISTER INDIRECT WITH 16-BIT DISPLACEMENT

USED TO REFERENCE ELEMENTS WITHIN AN ARRAY

ACCESS INDIVIDUAL I/O LOCATIONS WITHIN A BLOCK

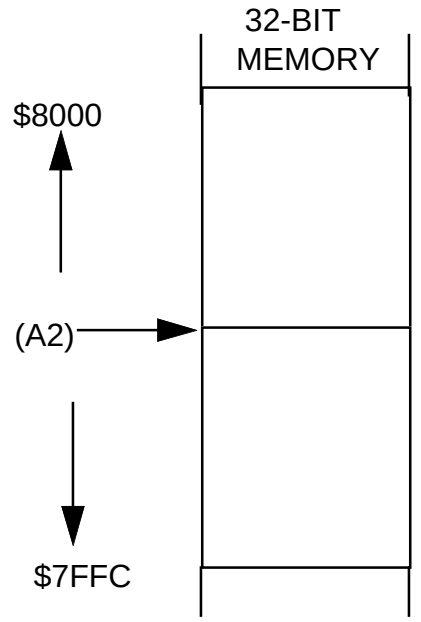
EA = (An) + 16-BIT SIGNED DISPLACEMENT

EXAMPLE

MOVE.S D1,10(A2) INSTRUCTION LENGTH = 2 WORDS

DATA REGISTER DIRECT

AR INDIRECT WITH 16d.



WHERE .S = BYTE, WORD OR LONG WORD

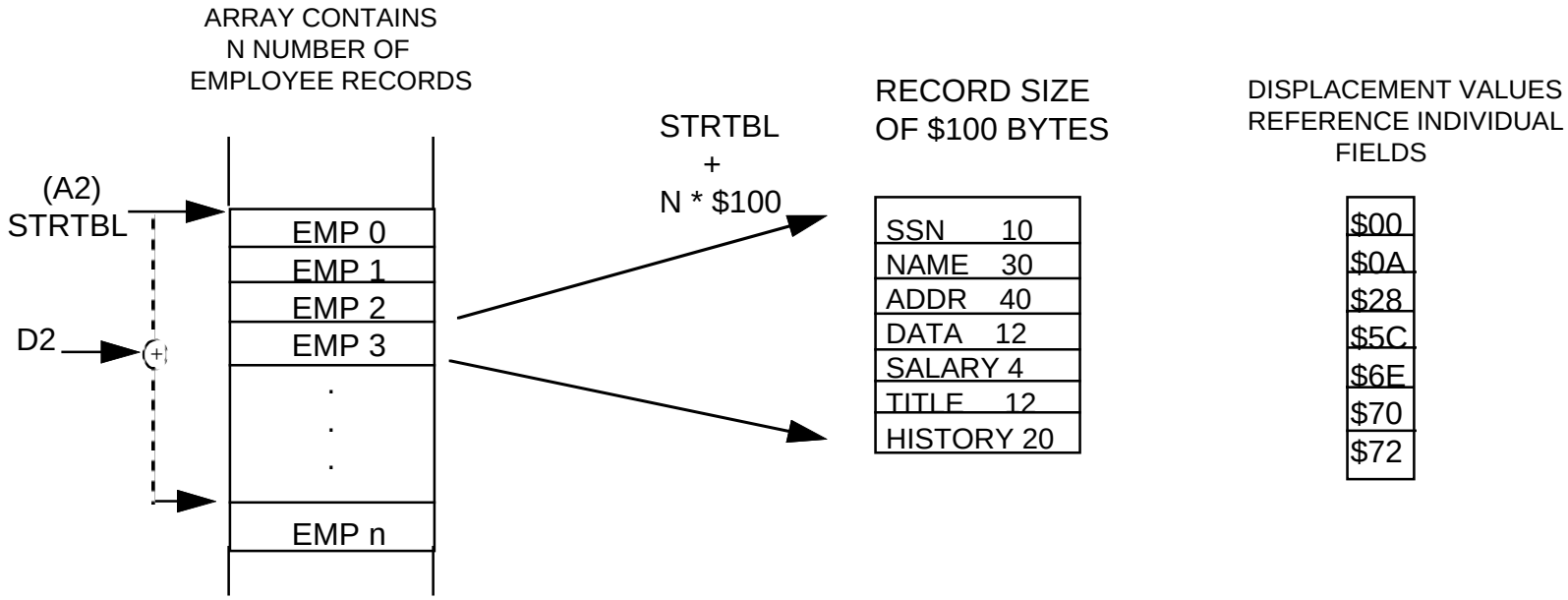
ADDRESS REGISTER INDIRECT WITH INDEXED AND 8-BIT DISPLACEMENT

USED TO REFERENCE DATA WITHIN COMPLEX STRUCTURE:
Eg: ACCESSING SINGLE OR MULTIPLE FIELDS WITHIN MULTIPLE RECORD ARRAY

EA = (An) + (Rx* SCALE) + 8-BIT DISPLACEMENT

EXAMPLE: MOVE.S (SALARY,A2,D2),D5

A2 = TBL START ADDRESS
D2 = POINT TO N EMP RECORD
SALARY = POINTS TO A FIELD WITHIN A RECORD



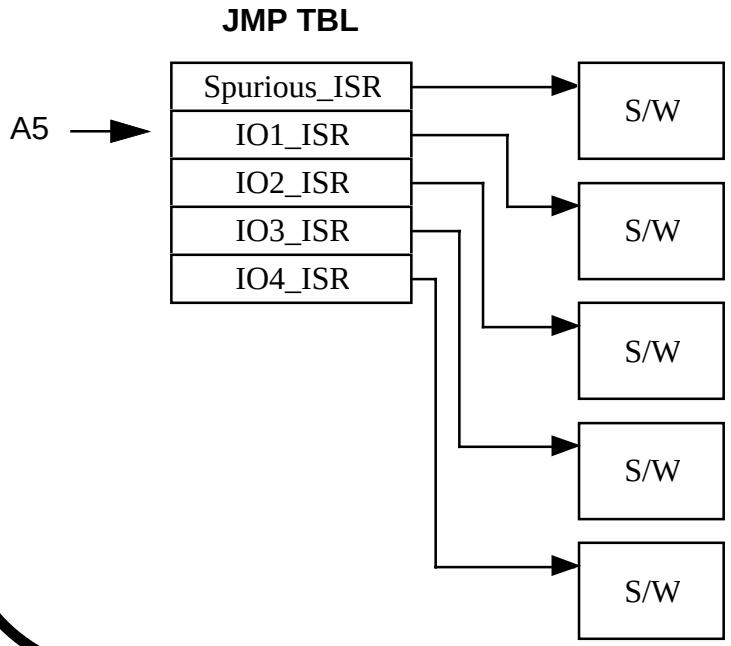
SCALED INDEX

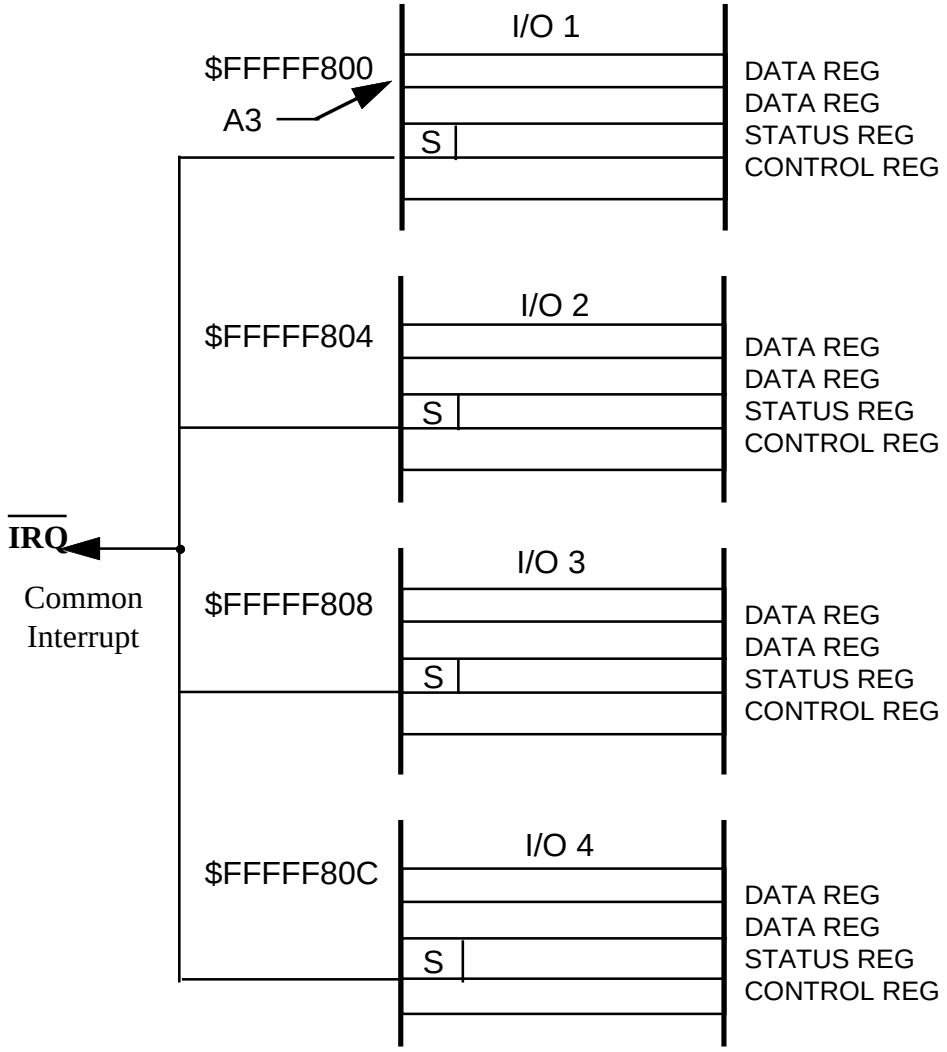
MAY BE USED TO POLL A NUMBER OF I/O DEVICES AND
JUMP TO APPROPRIATE ISR ROUTINE

EA = (An) + (Rx* SCALE) + 8-BIT DISPLACEMENT

EXAMPLE:

```
        MOVE.L    #3,D1
NEXT    BTST.B    #7,($2,A3,D1*4)
        BNE       DONE
        SUBQ.L    #1,D1
        BPL       NEXT
DONE    LEA       (A5,D1*4),A3
        JMP       (A3)
```





WHERE 'S' IS I/O INTERRUPT STATUS FLAG

ABSOLUTE SHORT ADDRESSING MODE

USED TO DEFINE A PERMANENT ADDRESS WITHIN A 64K BYTE RANGE
THESE LOCATIONS CAN BE ACCESSED IN 1 WORD ADDRESS

ALLOWS ACCESSES TO 1st. AND LAST PAGE IN 1 WORD ADDRESS

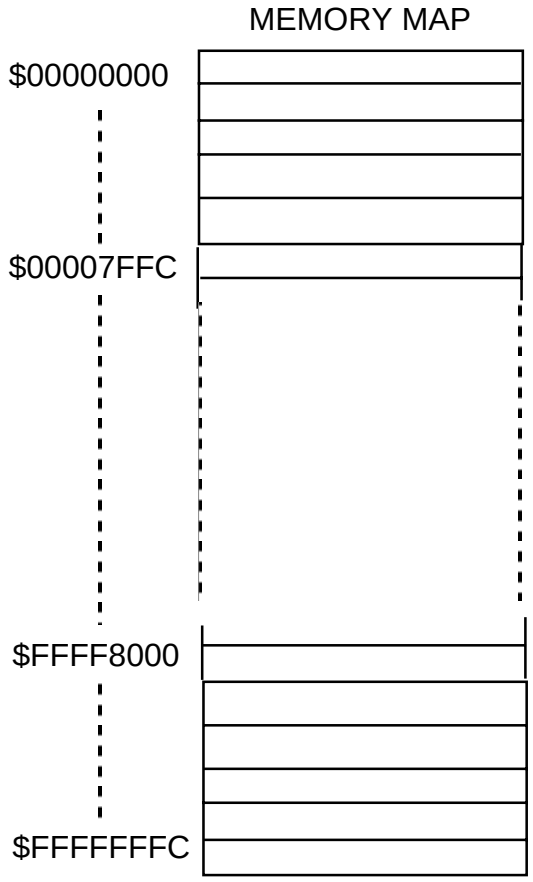
EA = XXXX WHERE XXXX IS A RANGE FROM $0000 - $7FFF
OR $8000 - $FFFF

EXAMPLES:

```
CLR.L   $3000

JMP     $9000
```

INSTRUCTION LENGTH = 2 WORDS



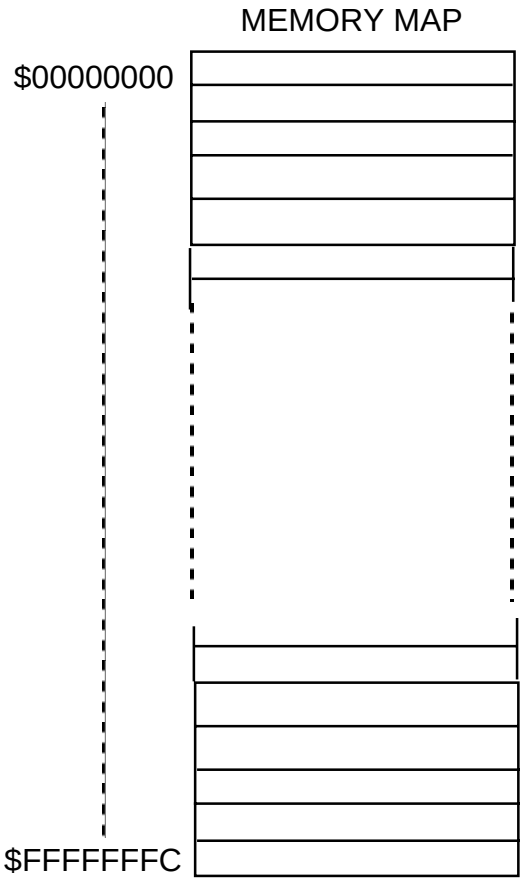
ABSOLUTE LONG ADDRESSING MODE

USED TO DEFINE A PERMANENT ADDRESS ANYWHERE IN THE MEMORY MAP
2 WORDS ARE REQUIRED FOR THE ADDRESS

EA = XXXXXX WHERE XXXX IS A RANGE FROM $00000000 - $FFFFFFFF

EXAMPLES: CLR.L $300000 INSTRUCTION LENGTH 3 WORDS

MOVE.B D1,$50000



IMMEDIATE ADDRESSING

USED TO SPECIFY CONSTANT VALUES TO INITIALIZE MPU REGISTERS,
EXTERNAL MEMORY LOCATIONS AND I/O PERIPHERAL DEVICES

EA = #XXXX WHERE #XXXX IS THE DATA CONSTANT

EXAMPLES:	MOVE.W	#$1234,IO_CNTR_REG(A5)	INSTRUCTION LENGTH = 2 WORDS
	MOVE.L	#$12345678, D5	INSTRUCTION LENGTH =3 WORDS

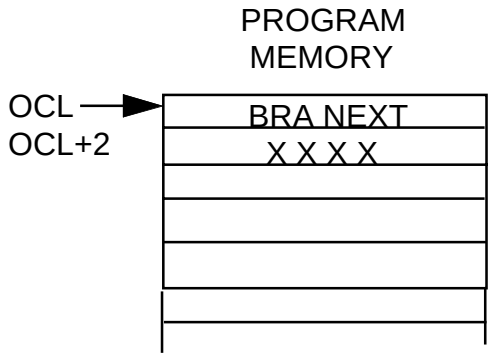
PROGRAM COUNTER RELATIVE

MAINLY USED FOR CONDITIONAL AND UNCONDITIONAL BRANCHES

EA = PC + 16-BIT SIGNED DISPLACEMENT

```
EXAMPLES:          BRA       OUT
                   _
                   _
                   _
OUT                _
                   _
                   _
                   _
                   _
                   _
LOOP               _
                   BNE       LOOP
```

INSTRUCTION LENGTH 1 WORD FOR SHORT BRANCHES
AND 2 WORDS FOR LONG BRANCHES



OPCODE LOCATION + 2 OR + 4 IS RELATIVE ADDRESS
TO WHICH THE DISPLACEMENT IS ADDED

COLDFIRE®

INSTRUCTION SET

INSTRUCTION CATEGORIES

- DATA MOVEMENT
- INTEGER ARITHMETIC
- LOGICAL & ARITHMETICAL SHIFTS
- BIT MANIPULATION
- PROGRAM CONTROL
- SPECIAL

COLDFIRE ® INSTRUCTION SET

DATA

CLR	MOVE
EXT	MOVEQ
EXTB	SWAP

SHIFT

ASL	LSR
ASR	LSL

CONTROL

BRA	*RTE
BSR	RTS
JSR	TRAP
JMP	TRAPF
NOP	

ARITHMETIC

ADD	MUL
CMP	NEG
TST	SUB

MULTIPLE PRECISION

ADDX	SUBX
NEGX	

CONDITIONAL

Bcc	Scc

LOGICAL

AND	NOT
EOR	OR

BIT MANIPULATION

BCHG	BSET
BCLR	BTST

SPECIAL INSTRUCTIONS

ILLEGAL	LINK	MOVEM
LEA	UNLK	*MOVEC
PEA	*STOP	MAC

DEBUG

*PULSE
WDDATA
*WDEBUG
! HALT

* PRIVILEGED INSTRUCTIONS
! PRIVILEGED BY DEFAULT

DATA MOVEMENT INSTRUCTIONS

INSTRUCTION	SIZE	OPERATION
MOVE	B, W, L	MOVE DATA FROM SRC. TO DEST.
MOVEA	W, L	MOVE DATA TO DEST An REG.
MOVE TO CCR	W	Dn OR #DATA → CCR
MOVE FROM CCR	W	CCR → Dn
*MOVE TO SR	W	#DATA → SR
*MOVE FROM SR	W	SR → Dn
MOVEQ	L	MOVE 8-BIT SIGN EXTENDED TO Dn REG.
SWAP	W	SWAP LOWER AND UPPER WORD OF Dn
EXT	B, W	EXTEND BYTE TO WORD & WORD TO LONG
EXTB	L	EXTEND BYTE TO LONG WORD
CLR	B, W, L	CLEAR Dn OR MEMORY

* PRIVILEGED INSTRUCTIONS

ARITHMETIC INSTRUCTIONS

INSTRUCTION	SIZE	OPERATION
ADD	L	SRC. + DEST. → DEST.
ADDI	L	#DATA + Dn → Dn
ADDA	L	SRC + An → An
ADDQ	L	ADDQ DATA (RANGE 1 - 8) TO <EA>
SUB	L	SRC. + DEST. → DEST.
SUBI	L	#DATA - Dn → Dn
SUBQ	L	SUBQ DATA (RANGE 1 - 8) TO <EA>
SUBA	L	SRC + An → An
MULU	W, L	MULU <EA>,Dn 16/32 x 16/32 → 32Dn
MULS	W, L	MULS <EA>,Dn 16/32 x 16/32 → 32Dn
DIVS	W/L	DIVIDES 32-BIT OF Dn/16-bit <ea>
DIVU	W/L	DIVIDES 32-BITS OF Dn/16-bits <ea>
NEG	L	0 - Dn → Dn
CMP	L	DEST. - SRC. → CC
TST	B, W, L	TESTED DEST. → CC

MULTIPLY INSTRUCTIONS SIGNED & UNSIGNED

- FORMAT

  MULS.W/L <size> <ea>,Dn

  MULU.W/L <size> <ea>,Dn

- RESULTS
  - 2 POSSIBILITIES

    16 X 16 BIT MULTIPLY YIELDING A 32-BIT RESULT

    32 X 32 BIT MULTIPLY YIELDING A 32-BIT RESULT

  - CONDITION CODE BITS

    N SET IF MOST SIGNIFICANT BIT OF RESULT IS SET

    Z SET IF RESULT IS 0

    V CLEARED

    C CLEARED

    X NOT AFFECTED

DIVIDE INSTRUCTIONS SIGNED & UNSIGNED

SignedDivide:

DIVS.W <ea>,Dn 32-bit Dn/16 --> {16r:16q} in Dn
DIVS.L <ea>,Dn 32-bit Dn/32 --> 32q in Dn
REMS.L <ea>,Dn:Dm 32-bit Dm/32 --> 32r in Dn

UnsignedDivide:

DIVS.W <ea>,Dn 32-bit Dn/16 --> {16r:16q} in Dn
DIVS.L <ea>,Dn 32-bit Dn/32 --> 32q in Dn
REMU.L <ea>,Dn:Dm 32-bit Dm/32 --> 32r in Dn

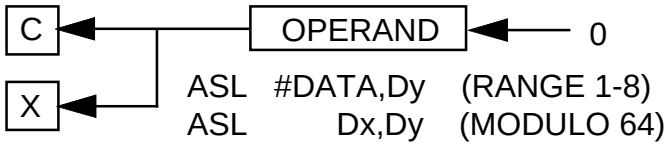
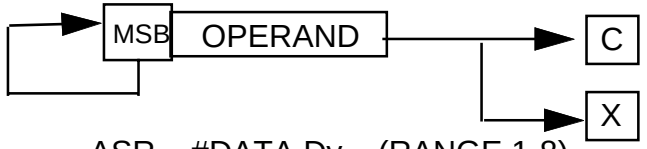
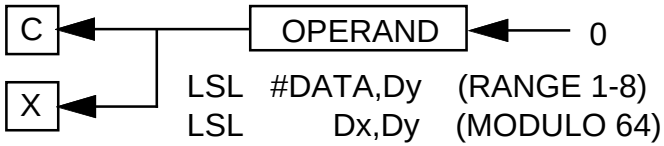
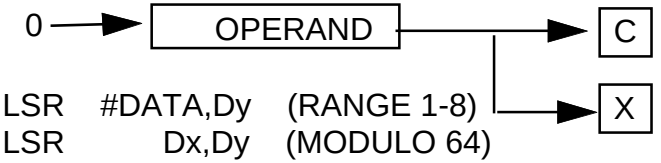
If a Divide by 0 is attempted, the Processor responds with a Divide-by-zero exception.
Stacked PC points to offending instruction.

On Overflow: CCR[v-bit] = 1
[z-bit] = 0
[n-bit] = 0
[c-bit] = 0

MULTIPLE PRECISION INSTRUCTIONS

INSTRUCTION	SIZE	OPERATION
ADDX	L	Dx + Dy + X → Dx
SUBX	L	Dx - Dy - X → Dx
NEGX	L	0 - Dn - X → Dx

SHIFT INSTRUCTIONS

INSTRUCTION	SIZE	OPERATION
*ASL	L	 ASL #DATA,Dy (RANGE 1-8) ASL Dx,Dy (MODULO 64)
ASR	L	 ASR #DATA,Dy (RANGE 1-8) ASR Dx,Dy (MODULO 64)
LSL	L	 LSL #DATA,Dy (RANGE 1-8) LSL Dx,Dy (MODULO 64)
LSR	L	 LSR #DATA,Dy (RANGE 1-8) LSR Dx,Dy (MODULO 64)

*OVERFLOW IS CLEARED

LOGICAL INSTRUCTIONS

INSTRUCTION	SIZE	OPERATION
AND	L	<EA> ^ Dn → Dn
OR	L	<EA> v Dy → Dn
EOR	L	<EA> ⊕ Dn → Dn
NOT	L	$\overline{Dn}$

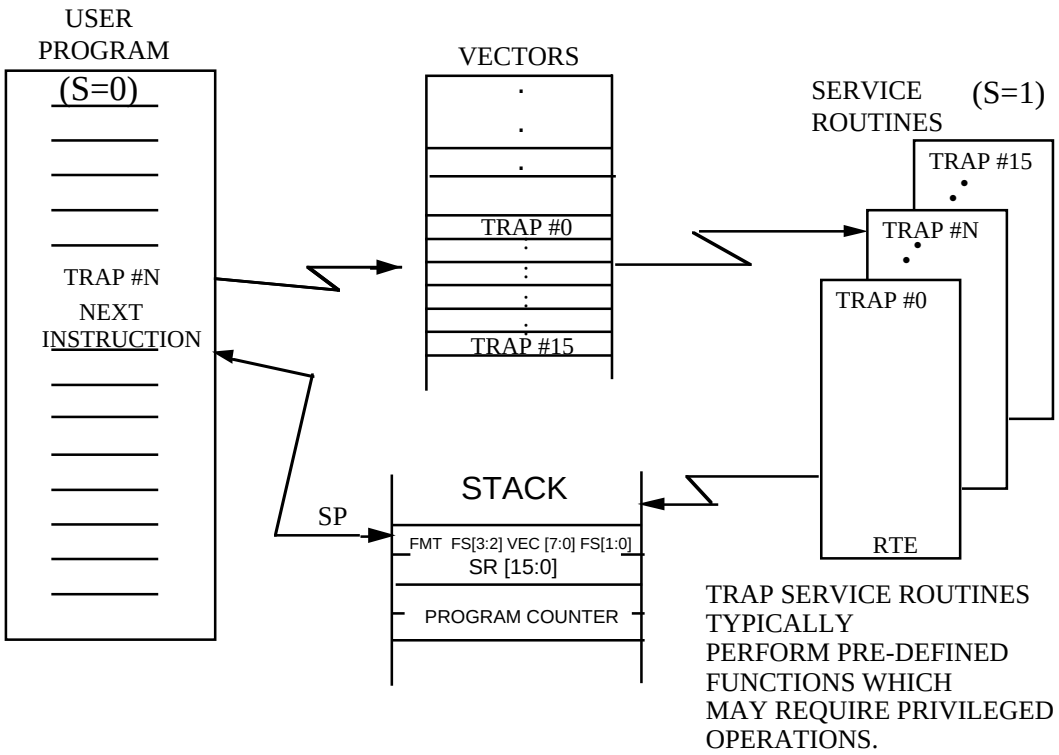
CONTROL INSTRUCTIONS

INSTRUCTION	SIZE	OPERATION
BRA	B, W	OCL + 2/4 + Disp. → PC
BSR	B, W	SP - 4 → SP ; PC → (SP) PC + dn → PC
JMP	UNSIZED	DEST. ADDR → PC
JSR	UNSIZED	SP - 4 → SP ; PC → (SP) PC + dn → PC
RTS	UNSIZED	(SP) → PC; SP + 4 → SP
*RTE	NONE	2(SP) → SR; 4(SP) → PC SP + 8 → SP
NOP	UNSIZED	PC + 2 → PC

* PRIVILEGED INSTRUCTION

TRAP INSTRUCTIONS

INSTRUCTION	SIZE	OPERATION
TRAP	UNSIZED	1 → S-BIT of SR; SP - 4 → SP PC → (SP); SP - 2 → SP SR → (SP); SP - 2 → SP FMT/OFFSET → (SP); VECTOR → PC
TRAPF	UNSIZED W, L	NO OPERATION (VARIABLE LENGTH NOP 1, 2 OR 3 WORDS)



CONDITIONAL BRANCH INSTRUCTIONS

CMP s,d Destination - Source → CCR
(d) - (s) → (r)

Bcc <label> if cc in CCR true, then branch

— **Bcc instructions can then be selected to determine:**

MNEMONIC	CONDITION	CCR TEST	INDICATION
(**L**) BMI	MINUS	N=1	r=NEGATIVE
(**L**) BPL	PLUS	N=0	r=POSITIVE
*(**L**) BVS	OVERFLOW	V=1	r=SIGN ERROR
*(**L**) BVC	NO OVERFLOW	V=0	r=SIGN OK
*(**L**)BLT	LESS	[N⊕V]=1	D < M
*(**L**)BGE	GREATER OR EQUAL	[N⊕V]=0	D >= M
* (**L**)BLE	LESS OR EQUAL	[Z+(N⊕V)]=1	D <= M
*(**L**) BGT	GREATER	[Z+(N⊕V)]=0	D > M
(**L**)BEQ	EQUAL	Z=1	D = M
(**L**) BNE	NOT EQUAL	Z=0	D <> M
(**L**)BHI	HIGHER	[C+Z]=0	D > M
(**L**) BLS	LOWER OR SAME	[C+Z]=1	D <= M
(**L**)BCC (BHS)	CARRY CLEAR	C=0	D >= M
(**L**) BCS (BLO)	CARRY SET	C=1	D < M

* Use for signed arithmetic only

FIND MAXIMUM VALUE

THIS ROUTINE FINDS THE MAXIMUM VALUE IN A LIST OF SIGNED DATA. MODIFY THE ROUTINE
TO FIND MAXIMUM VALUE IN A LIST OF UNSIGNED DATA.
ALSO CALCULATE NUMBER OF INSTRUCTIONS AND TIMING.

```
                ORG       $30000              ORIGINATE  SIGNED DATA AT ADDRESS $30000

SIGNED          FCL       -$60000, $50540, -$20500, $907500
                FCL       -$30000, $19800, $50050, $602560
                FCL       -$30565, -$58760, $22290, -$74325
                FCL       -$23410, $456055, -$291760, -$165670

                ORG       $30100              ORIGINATE  SIGNED DATA AT ADDRESS $30100

UNSIGNED        FCL       $67008, $59950, $25675, $712345
                FCL       $30980, $11210, $ABCD5, $63395
                FCL       $35000, $50060, $212540, $BC325
                FCL       $C0000, $90705, $DDCC0, $34517

                ORG       $30200              BEGIN PROGRAM AT $31000

BEGIN           MOVE.L    #SIGNED,A0           INIT POINTER TO DATA TABLE
                MOVEQ     #$10,D0              INIT LOOP COUNTER
GET_DATA        MOVE.L    (A0),D1              GET NEXT LONG WORD
DEC_CNT         SUBQ.L    #1,D0               DECREMENT COUNTER
                BEQ       DONE                IF COUNT  = 0, GO TO DONE
                ADDA.L    #$4,A0              SET POINTER TO NEXT OPERAND
                CMP.L     (A0),D1             COMPARE NEXT DATA
                BGE       DEC_CNT             IF D1 >; THEN GO DECREMENT CNT
                BRA       GET_DATA            GO UPDATE D1 WITH THE HIGHER DATA
DONE            BRA       DONE                LOOP FOR EVER
```

SET ON CONDITION

CMP s,d Destination - Source → CCR

Scc Dn if cc in CCR true, then 1s → DEST
Else 0s → DEST

MNEMONIC	CONDITION	CCR TEST	INDICATION
SMI	MINUS	N=1	r=NEGATIVE
SPL	PLUS	N=0	r=POSITIVE
SVS	OVERFLOW	V=1	r=SIGN ERROR
* SVC	NO OVERFLOW	V=0	r=SIGN OK
* SLT	LESS	[N⊕ V]=1	D < M
* SGE	GREATER OR EQUAL	[N⊕ V]=0	D >= M
* SLE	LESS OR EQUAL	[Z+(N⊕ V)]=1	D <= M
* SGT	GREATER	[Z+(N⊕ V)]=0	D > M
SEQ	EQUAL	Z=1	D = M
SNE	NOT EQUAL	Z=0	D <> M
SHI	HIGHER	[C+Z]=0	D > M
SLS	LOWER OR SAME	[C+Z]=1	D <= M
SCC (BHS)	CARRY CLEAR	C=0	D >= M
SCS (BLO)	CARRY SET	C=1	A < M
ST	TRUE	-	-
SF	FALSE	-	-

SET ON CONDITION

USED TO EVALUATE MULTIPLE CONDITIONS

EXAMPLE: HIGH LEVEL LANGUAGE CONDITION EVALUATIONS

IF A = B ^ C > D THEN GOTO CONDITIONS

ELSE

–

–

–

ASSEMBLY LANGUAGE EQUIVALENT

```
CLR.L       D0
CLR.L       D1
CMP         A,B
SEQ.B        D1
CMP         C,D
SLT.B        D2
AND.L       D1,D2
BNE         CONDITIONS
 –
 –
```

BIT MANIPULATION INSTRUCTIONS

INSTRUCTION	SIZE	OPERATION
BTST	B, L	TEST (BIT n) OF DEST. → Z
BSET	B, L	TEST (BIT n) OF DEST. → Z 1 → (BIT n) OF DEST.
BCLR	B, L	TEST (BIT n) OF DEST. → Z 0 → (BIT n) OF DEST.
BCHG	B, L	TEST (BIT n) OF DEST. → Z ~ → (BIT n) OF DEST.

EXAMPLES:

```
BSET.B    #7,(A5)    BIT n RANGE 0 - 7
BCHG.L    D1,D7      BIT n RANGE 0 - 31
```

SPECIAL INSTRUCTIONS CONT'd

INSTRUCTION	SIZE	OPERATION
LEA	L	<EA> → An
PEA	L	<EA> → - (SP)

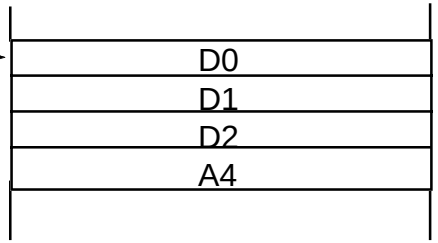
EXAMPLES:

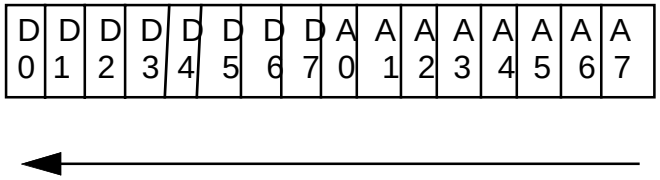
```
LEA     (LABEL,A5,D6*4), A5
LEA     -16(SP),SP
PEA     ($10,A2,D3)
PEA     (A0,D3*2)
```

SPECIAL INSTRUCTIONS

INSTRUCTION	SIZE	OPERATION
MOVEM	L	REGISTER LIST → DEST. SOURCE → REGS.

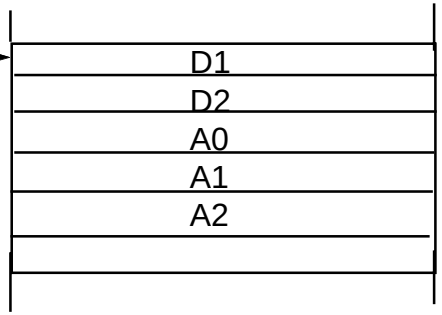
MOVEM.L D0-D2/A4, (A2)

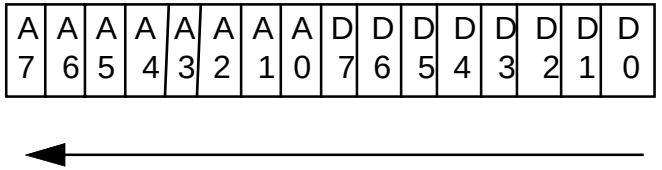




MOVEM.L (SP), A0-A2/D1-D2

SP →





ADDRESSING MODES ALLOWED:

(An)
(d16,An)

MOVEM EXAMPLES

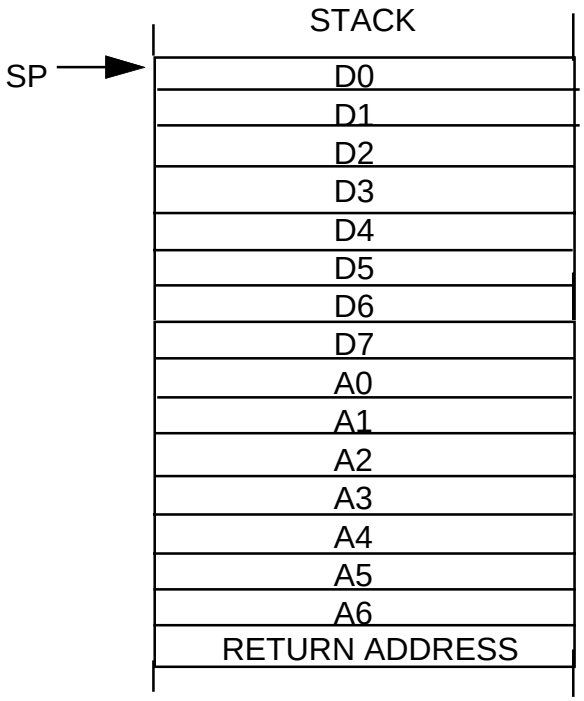
SUBROUTINE

```
LEA    -12(A7),A7
MOVEM.L  A1/D4-D5,(SP)
 -
 -
MOVEM.L  (SP),D4/A1/D5
 LEA    12(A7),A7
 RTS
```

MUST SAVE A RESTORE MAIN REGS.

SERVICE ROUTINE

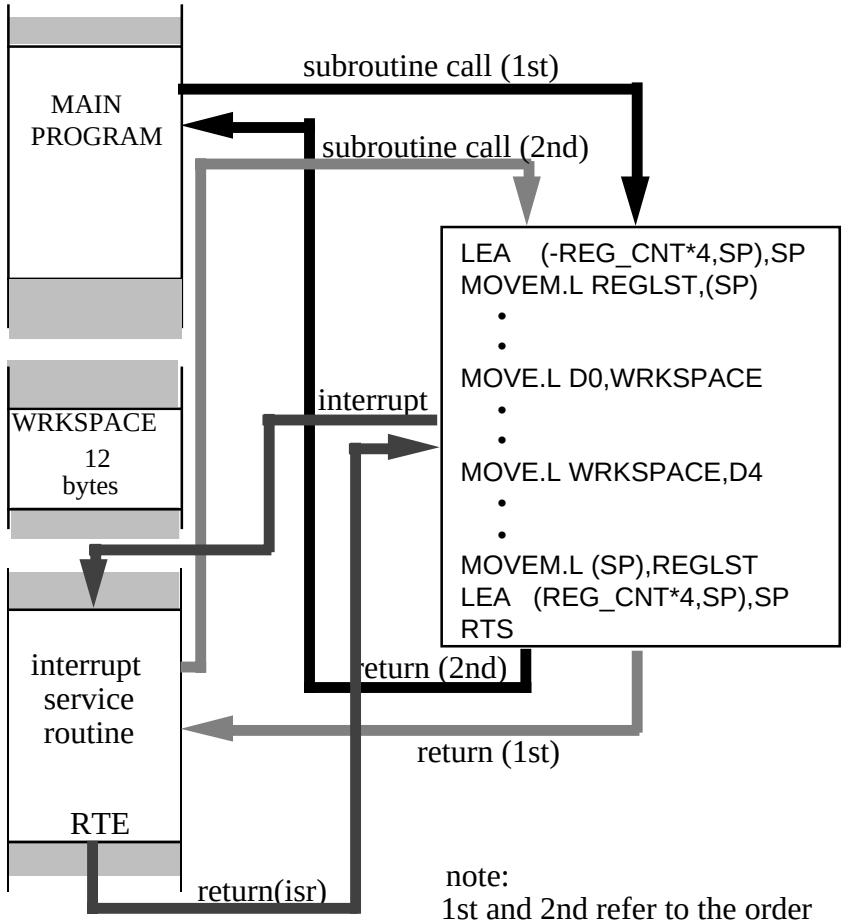
```
LEA       -60(A7),A7
MOVEM.L  A0-A6/D0-D7,(SP)
  -
  -
 MOVEM.L  (A7),D0-D7/A0-A6
LEA       60(A7),A7
 RTE
```



THE PROBLEM OF REENTRANCY

REENTRANT -A PROGRAM WHICH MAY BE EXITED BY MEANS OF AN INTERRUPT OR SUBROUTINE CALL AND BE RE-ENTERED AND OPERATE PROPERLY.

EXAMPLE: NOT REENTRANT (USING FIXED ADDRESS)

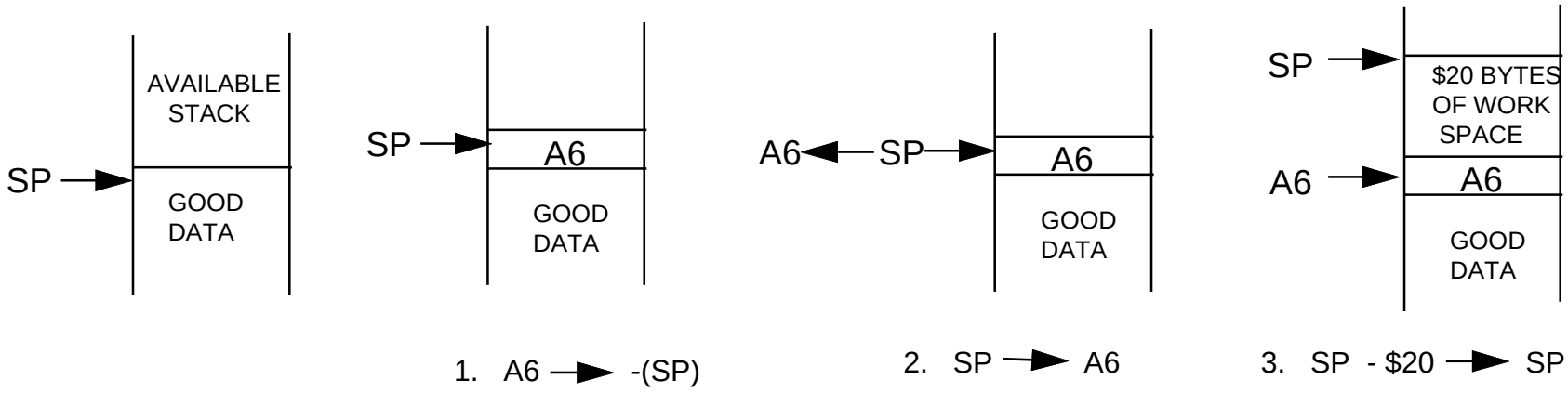


note:
1st and 2nd refer to the order in which events occur

SPECIAL INSTRUCTIONS CONT'd

INSTRUCTION	SIZE	OPERATION
LINK	W	SP - 4 → SP ; An → (SP); SP → An; SP + dn → SP

EXAMPLE: LINK A6, #-$20



1. A6 → -(SP)

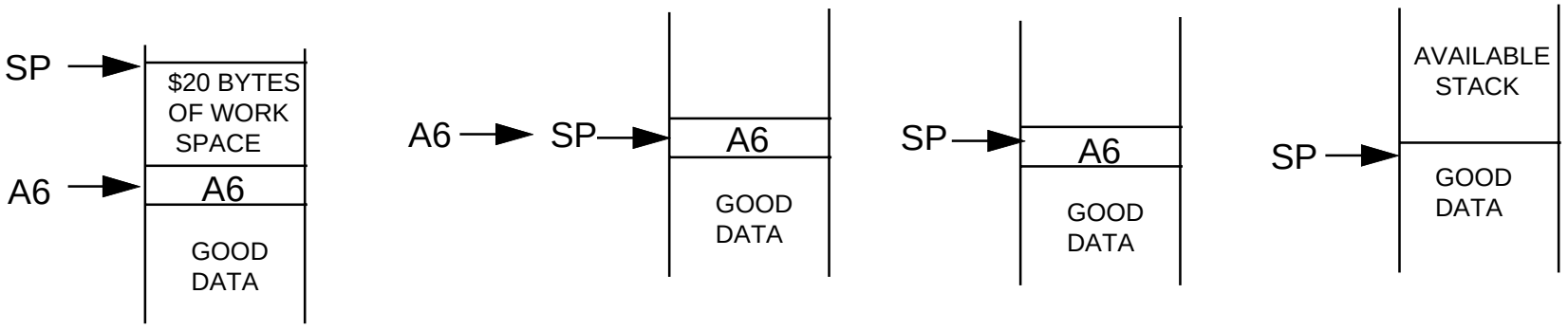
2. SP → A6

3. SP - $20 → SP

SPECIAL INSTRUCTION CONT'd

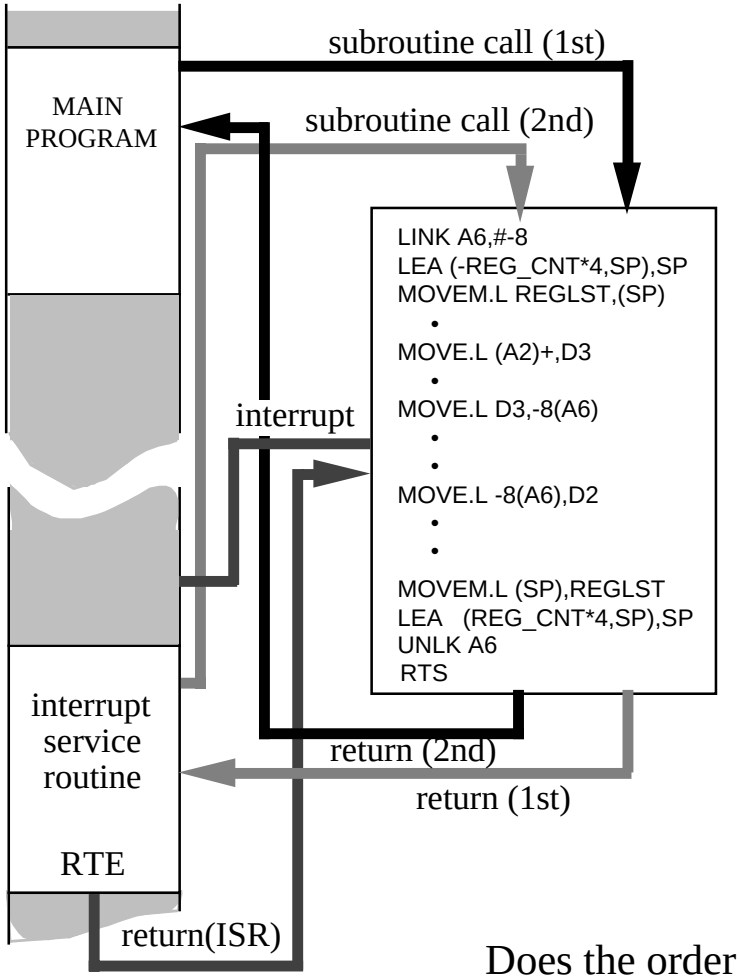
INSTRUCTION	SIZE	OPERATION
UNLK	W	An → SP ; (SP)+ → An

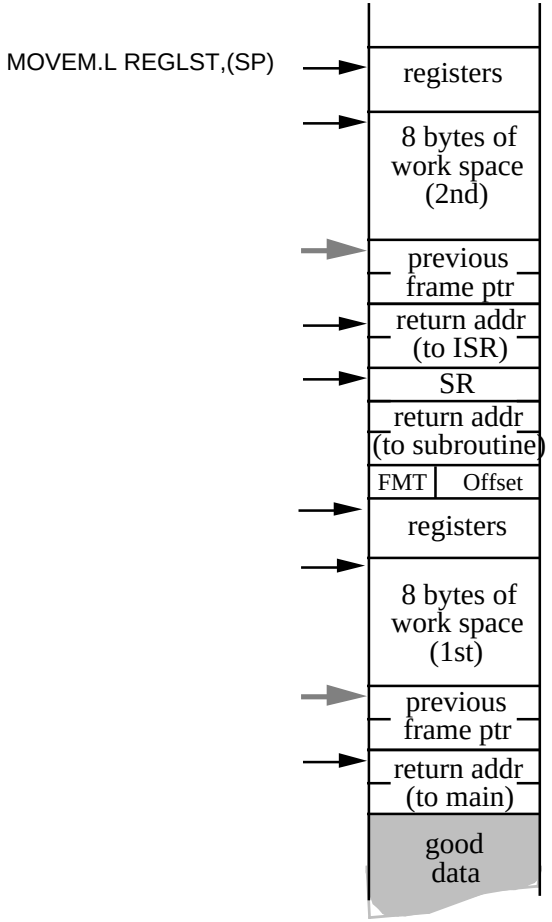
EXAMPLE: UNLK A6



1. A6 → SP 2. (SP)+ → A6

REENTRANT SOLUTION





note: 1st and 2nd refer to the order in which events occur.

Does the order of the MOVEM and LINK instructions in the subroutine make any difference?

SPECIAL INSTRUCTIONS CONT'd

INSTRUCTION	SIZE	OPERATION
*STOP	W	#DATA → SR ; STOP INSTRUCTION EXECUTION & WAIT FOR INTERRUPT

• STOP: LOADS #DATA INTO SR & STOPS BUS CYCLE ACTIVITY. THEN WAITS FOR INTERRUPT REQUEST HIGHER THAN THE MASK SET BY THE #DATA

EXIT STOP INSTRUCTION IF:	INSTRUCTION EXECUTION RESUMES IN:
INTERRUPT REQUEST > MASK LEVEL.	INTERRUPT SERVICE ROUTINE.
T = 1 WHEN STOP BEGINS EXECUTION.	TRACE SERVICE ROUTINE.
S = 0 AFTER #DATA IS IN SR.	PRIVILEGE VIOLATION SERVICE ROUTINE FOR
EXTERNAL RESET OCCURS.	RESET EXCEPTION SERVICE ROUTINE.
BKPT	ENTERS HALTED STATE

* PRIVILEGED INSTRUCTION

MOVEC INSTRUCTION

INSTRUCTION	SIZE	OPERATION
*MOVEC	L	Rn → Rc

* PRIVILEGED INSTRUCTION

CPU SPACE MAP

RC [11:0]	REGISTER DEFINITION
$002	CACHE CONTROL REGISTER (CACR)
$003	-
$004	ACCESS CONTROL REGISTER 0
$005	ACCESS CONTROL REGISTER 1
$006	ACCESS CONTROL REGISTER 2
$007	ACCESS CONTROL REGISTER 3
$08x	-
$18x	-
$801	VECTOR BASE REGISTER (VBR)
$80E	-
$C00	ROM BASE ADDRESS REGISTER (ROMBAR)
$C04	RAM BASE ADDRESS REGISTER (RAMBAR0)
$C05	RAM BASE ADDRESS REGISTER (RAMBAR1)
$C0F	MODULE BASE ADDRESS REGISTER (MBAR)

REGISTERS ARE IMPLEMENTATION SPECIFIC

MULTIPLY-ACCUMULATE

MODULE

MULTIPLY-ACCUMULATE UNIT (MAC)

Features:

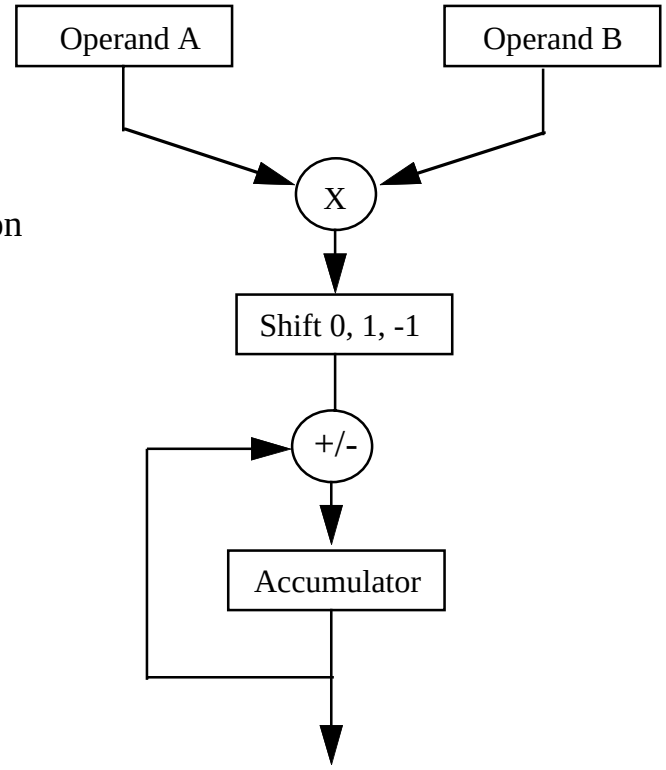
- Signed and Unsigned Integer Multiplies

- Multiply-accumulate operation supporting:
  - 16x16 [un]signed Multiply-accumulate in 1 Clk Cycle
  - 32x32 [un]signedMultiply-accumulate in 3 Clk Cycles
  - Product may be Shifted once right or left prior to Accumulation

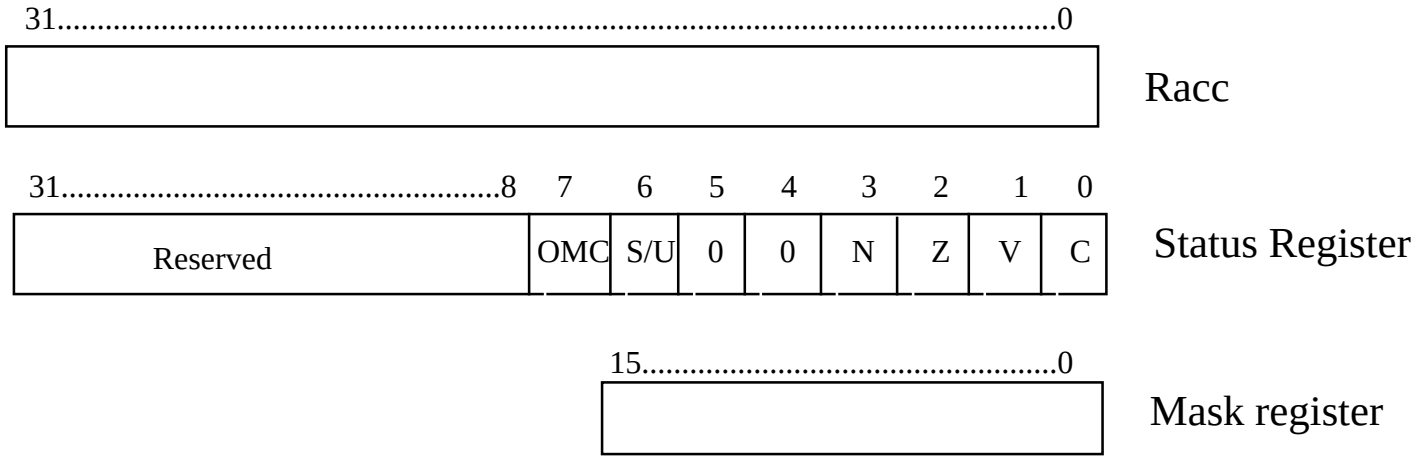
- Register-based Arithmetic operations

- Circular Buffer Mode Support

- MAC Module Instructions to support operand manipulation



MAC PROGRAMMING MODEL



Status Register

OMC - Overflow/Saturation Mode Control
0 = Disable Saturation Mode
1 = Enable Saturation Mode

S/U - Signed/Unsigned Operations
0 = Signed (If OMC=1), on overflow Racc
saturates to the most positive value
of $7FFF_FFFF and most negative value of $8000_0000

1 = Unsigned (If OMC =1), on overflow, Racc saturates
to the smallest value of $0000_0000
or to largest value of $FFFF_FFFF
(On overflow, the Racc value remains unchanged until is loaded
with new value or until V flag is cleared)

N - Negative
Sets when the MSB of result =1

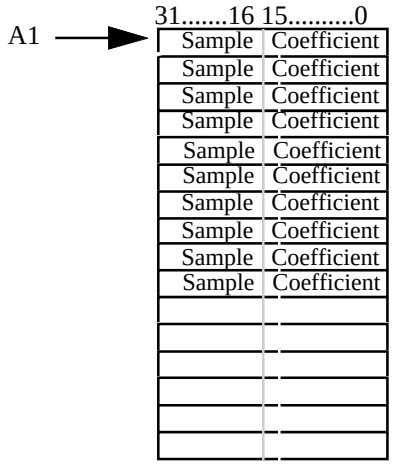
Z - Zero
Sets when the result of a Mac
operation = $0000_0000

V - Overflow
sets when a MAC operation result
in overflow. V-bit is cleared when Racc
or MACSR is loaded

C - Carry
This bit is always cleared

MAC OPERATION

The MAC Instruction is optimized for 16x16 Multiply_accumulate.
The MAC Module is capable of loading an operand into a register while multipy&accumulate.
The operand may be a 32-bit value comprised of a sample and coefficient.



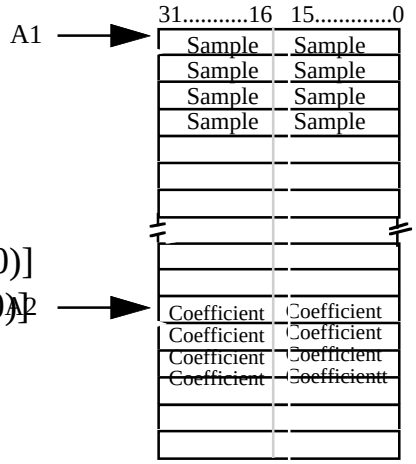
Three option of MAC operations are possible:

- MAC Module allow multiply_accumulate of a register upper and lower words, i.e.(Sample x cofficient)

Example_1: MAC.W D1.U,D1.L,(A1)+,D1 [Execution Time in Clk ~ = 2(1/0)]

Example_2:MAC.L Ry,Rx [Execution Time in Clk ~ = 3(0/0)]

- Loading multiple registers using MOVEM instruction



Example_3: MOVEM.L (A1),D0-D3
MOVEM.L (A2),D4-D7
MAC.W D0.U,D4.U [Execution Time in Clk ~ =1 (0/0)]
MAC.W D0.L,D4.L [Execution Time in Clk ~ = 1(0/0)]
-
-
MAC.W D3.U,D7.U
MAC.W D3.L,D7.L

MAC INSTRUCTION DETAILS 1 OF 2

MAC | MSAC Rx,RySF Racc+(Rx*Ry){<<|>>}SF --> Racc

Opcode

15.....12	11	10	9	8	7	6	5	4	3..... 0
1 0 1 0	Rx	Rx	Rx	0	0	Rx	0	0	Ry Ry Ry Ry
- - - -	SZ	Scale Factor		0	U/Lx	U/Yx	-	-	- - - -

Rx Src Register
0000 = D0
o
o
o
1111 = A7

Ry Src Register
0000 = D0
o
o
o
1111 = A7

SZ - SIZE
0 = Word-sized
1 = Long-sized

U/Lx, U/Ly - Word Select
0 = Lower word
1 = Upper word

Scale Factor
00 = None
01 = Product <<1
10 = Reserved
11 = Product >>1

MAC INSTRUCTIONS DETAILS 2 OF 2

MAC | MSAC **Rx,RySF,<EA>,RW** **Racc+(Rx*Ry){<<|>>}SF --> Racc (<EA>) --> Rw**

Opcode

15.....12	11	10	9	8	7	6	5..... 0
1 0 1 0	Rw	Rw	Rw	0	1	Rw	<ea>
Rx Rx Rx Rx	SZ	Scale Factor		0	U/Lx	U/Yx	Mask 0 Ry Ry Ry Ry

Rw Dest Register

0000 = D0
o
o
o
1111 = A7

Rx Src Input Register

0000 = D0
o
o
o
1111 = A7

Ry Src Input Register

0000 = D0
o
o
o
1111 = A7

SZ - SIZE

0 = Word-sized
1 = Long-sized

U/Lx, U/Ly - Word Select

0 = Lower word
1 = Upper word

Scale Factor

00 = None
01 = Product <<1
10 = Reserved
11 = Product >>1

Mask

0 = Rmask Register not used in <ea>
1 = Rmask Register is used in <ea>

<ea> Addressing Mode

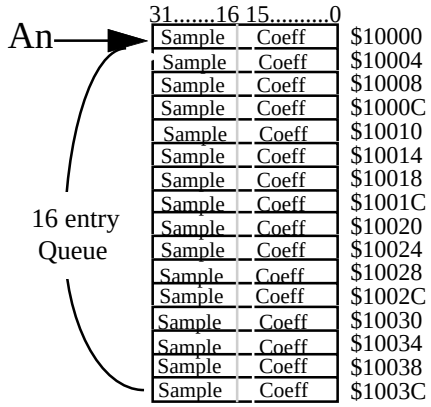
(An), (An)+, -(An), (d16,An)

MAC OPERATION WITH MASK

The Multiply-accumulate Rmask register provides a mechanism to establish a circular queue up to 65k entries.

Rmask Register

1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1

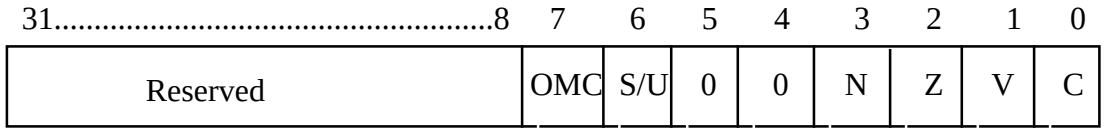


MOV.L <ea>,Rmask

<ea> Addressing Modes:
Dn, An, & #(data)

[Execution Time in Clk ~ = 1(0/0)]

MAC CONDITIONAL BRANCHING



MAC
Status Register

A copy of MACSR may be loaded into Core CCR by the instruction:

MOV.L MACSR,CCR Operation: MACSR[4:0] --> CCR[4:0] [Execution Time in Clk ~ = 1(0/0)]

The Core Condition Code Register will be affected as follow:

CCR.X = 0
CCR.N = Set to value of MACSR.N
CCR.Z = Set to value of MACSR.Z
CCR.V = Set to value of MACSR.V
CCR.C = Cleared

A Core conditional branch instruction may be executed to evaluate a MAC result:

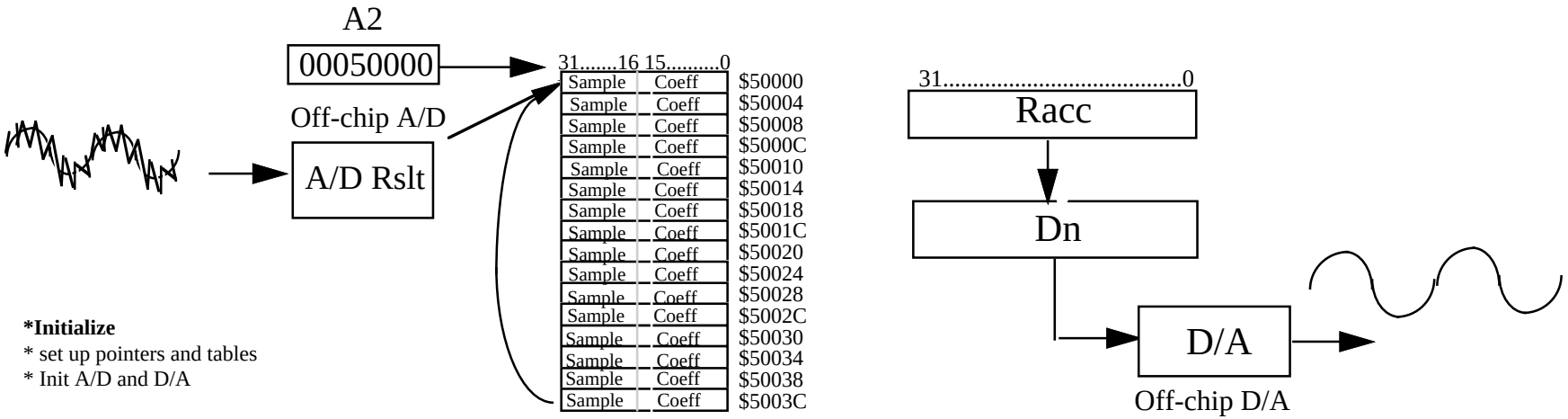
BMI/PL (Branch on Minus/Plus)
BEQ/NE (Branch on Equal/Not equal)
BVC/VS (Branch on Overflow Clear/set)

NOTE: The MACSR Condition Code Bits map to same location of Core CCR.

MAC INSTRUCTIONS SUMMARY

INSTRUCTION	MNEMONIC	OPERATION
Multiply Accumulate	MAC Rx,Ry SF MSAC Rx,Ry SF	Multiply two [un]signed operands, then add/sub the product to/from the Accumulator
Multiply Accumulate with Load	MAC Rx,Ry SF MSAC Rx,Ry SF	Multiply two [un]signed operands, then addthe product to the Accumulator while loading a register from memory
Load Accumulator	MOV.L (Ry,#imm),Racc	Load the Accumulator with 32-bit operand
Store Accumulator	MOV.L Racc,Rx	Copy the contents of Accumulator to a register
Load MAC Status Register	MOV.L (Ry,#imm)MACSR	Write a value to MACSR
Store MAC Status Register	MOV.L MACSR,Rx	Copy the value of MACSR to a register
Load MAC Mask Register	MOV.L (Ry,imm),RMASK	Write a value to the MAC Mask Register
Store MAC Mask Register	MOV.L RMASK, Rx	Copy the value of RMASK to a register
Move MACSR to CCR	MOV.L MACSR,CCR	Copy MAC Status register to the CORE Condition Code register

FIR FILTER EXAMPLE



***Initialize**
* set up pointers and tables
* Init A/D and D/A

```
            MOVEQ.L     #0,D0               /clear D0
            MOVE.B      D0,ADCTL            /initiate A/D conversion
WAIT        BTST        #COF,ADCSR          /wait here until
            BEQ         WAIT                /conversion is complete
```

***Get Sample**

```
            MOVE.W      ADRSLT,(A2)         /put new sample in sample table
            MOVE.L      (A2)+,D1            /load next Sample and Coeff.
            MOVE.B      D0,ADCTL            /initiate next conversion
            MOVE.L      D0,Racc             /clear Racc
            MOVEQ.L     #16,D3              /set number of Taps to 16
            MOVE.L      #$FFF7,Rmask        /set circular buffer to 16 entries
LOOP        MAC.W       D1.U,D1.L,(A2)+,D1  /multiply-accumulate/load next entry into D1 (Mask is used in Addr generation)
            SUBQ.L      #1,D3               /loop until all 16 entries are
            BNE         LOOP                /multiplied-accumulated
            MOVE.L      Racc,D4             /get accumulated result in D4
            MOVE.L      D4,DAREG            /output result to D/A converter
```

***Shift Sample Table**

```
            MOVE.L      #$0038,D6           /point index to last entry-1
            MOVE.W      (A2,D6),D5          /get last entry - 1 and
Shift       MOVE.W      D5,(2,A2,D6)        /discard oldest sample
            SUB.L       #$04,D6             /move samples down until
            BNE         SHIFT               /all entries have been shifted
            BRA         WAIT                /go get next conversion result (Allow enough time for conversion)
```

DSP APPLICATIONS

- **Mass Storage**
  - Disc/Zip Drives
  - Tape Drives
- **Voice Coding and Compression**
  - Digital Speech Transmission
  - Cellular Phones
  - Speech Recognition
- **Automotive**
  - Active Suspension
  - Digital Radio
  - Adaptive Noise Cancellers
- **Telecommunication**
  - Modems
  - PBX
  - Echo Cancelling
  - Hands-Free Telephones
- **Digital Audio**
  - Car/Home Stereo Surround Processors,
  - Recording
  - Audio-Music Synthesis