Lecture 11:
Advanced Arithmetic Instructions

Today's Goals

- Use basic **multiplication** and **division** instructions.
- Use **shift** and **rotate** instructions

Multiplication

- Three different multiplication instructions.
  - MUL
    - <u>Unsigned</u> 8 by 8 multiplication
    - D (A:B) ← A * B
  - EMUL
    - <u>Unsigned</u> 16 by 16 multiplication
    - Y:D ← D * Y
  - EMULS
    - <u>Signed</u> 16 by 16 multiplication
    - Y:D ← D * Y
- Note:
  - <u>Register Y</u> is being used for multiplication.

Example:

```
LDAA  #$12
LDAB  #$34
MUL
```

D(A:B) = 03A8

Division

- Five different division instructions
  - IDIV
    - **Unsigned** 16 by 16 integer division
    - X ← (quotient) (D) / (X)
    - D ← (remainder) (D) / (X)
  - IDIVS
    - **Signed** 16 by 16 integer division
    - X ← (quotient) (D) / (X)
    - D ← (remainder) (D) / (X)
  - FDIV
    - Like **IDIV**, but 16 by 16 **fractional** division.
    - Expect dividend to be smaller than divisor.
    - X ← (quotient) (D) / (X)
    - D ← (remainder) (D) / (X)

$$\frac{D}{X}$$

Example:

33÷2
= 16 and 1

16 ← quotient
1 ← remainder

33 ← dividend
--
2 ← divisor

5÷9
LDD #5
LDX #9
IDIV
(X) = 0, (D) = 5

Division – continued

- EDIV
  - Unsigned 32 by 16 integer division
  - Y ← (quotient) (Y:D) / (X)
  - D ← (remainder) (Y:D) / (X)
- EDIVS
  - Signed 32 by 16 integer division
  - Y ← (quotient) (Y:D) / (X)
  - D ← (remainder) (Y:D) / (X)

- Note: Register X is being used for division while Y for multiplication.

Example

Exponential Filter

- An **exponential filter** is often used to condition an incoming input signal.

$$X_{ave}(t) = \alpha \times X(t) + (1-\alpha) \times X_{ave}(t-1)$$

- *X(t)* is an input signal at time *t*.
- The weight factor α must be between 0 and 1.
- α determines how quickly the filtered value will respond to a change in input.
- Let's write a program
  - $\alpha = 5/9$
  - 8 bit input is supplied in address $0000
  - 8 bit output is written to $0001

Example – source code

$$X_{ave}(t) = 5/9 \times X(t) + (1-5/9) \times X_{ave}(t-1)$$

$$X_{ave}(t) = (5 \times X(t))/9 + ((9-5) \times X_{ave}(t-1))/9$$

Exponential Filter

$$X_{ave}(t) = \boxed{\alpha \times X(t)} + \boxed{(1-\alpha) \times X_{ave}(t-1)}$$

```
alpn  EQU   5
alpd  EQU   9

      ORG   $0000
Xin   DS.B  1
Xave  DS.B  1

      ORG   $2000
Loop:
      ; alpha * X (t)
      LDAA  Xin   ; A <-- (Xin)
      LDAB  #alpn ; B <-- 5
      MUL         ; D <-- (A) * (B) = (Xin) * 5
      LDX   #alpd ; X <-- 9
      IDIV        ; X <-- Q((D)/(X)),    D <-- R((D)/(X))
      TFR   X,Y   ; Y <-- (Xin * 5) / 9
      ; (1 - alpha) * Xave (t)
      ; (1 - 5 / 9) = (9 - 5) / 9
      LDAA  Xave
      LDAB  #(alpd-alpn)    ; (9-5) = 4
      MUL         ; D <-- Xave * 4
      LDX   #alpd ; X <-- 9
      IDIV        ; X <-- (Xave * 4) / 9
      TFR   X,B   ; The result of the second term in the equation
      TFR   Y,A   ; The result of the first term in the equation
      ABA         ; A <-- (A) + (B)
      STAA  Xave  ; Save the average value. Xave <-- (A)
      BRA   Loop
      SWI
```

- The program uses register Y to save the first term.
- The second term is saved to register X and transferred to B
- Transfer register Y to A to add those two terms. (ABA)

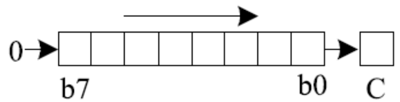
Example – listing file

```
          =00000005         alpn    EQU     5
          =00000009         alpd    EQU     9

          =00000000                 ORG     $0000
     0000 +0001             Xin     DS.B    1
     0001 +0001             Xave    DS.B    1

          =00002000                 ORG     $2000
     2000                   Loop:
                                    ; alpha * X (t)
     2000 96 00                     LDAA    Xin       ; A <-- (Xin)
     2002 C6 05                     LDAB    #alpn     ; B <-- 5
     2004 12                        MUL               ; D <-- (A) * (B) = (Xin) * 5
     2005 CE 0009                   LDX     #alpd     ; X <-- 9
     2008 1810                      IDIV              ; X <-- Q( (D)/(X) ),    D <-- R( (D)/(X) )
     200A B7 56                     TFR     X,Y       ; Y <-- (Xin * 5) / 9
                                    ; (1 - alpha) * Xave (t)
                                    ; (1 - 5 / 9) = (9 - 5) / 9
     200C 96 01                     LDAA    Xave
     200E C6 04                     LDAB    #(alpd-alpn)    ; (9-5) = 4
     2010 12                        MUL               ; D <-- Xave * 4
     2011 CE 0009                   LDX     #alpd     ; X <-- 9
     2014 1810                      IDIV              ; X <-- (Xave * 4) / 9
     2016 B7 51                     TFR     X,B       ; The result of the second term in the equation
     2018 B7 60                     TFR     Y,A       ; The result of the first term in the equation
     201A 1806                      ABA               ; A <-- (A) + (B)
     201C 5A 01                     STAA    Xave      ; Save the average value. Xave <-- (A)
     201E 20 E0                     BRA     Loop
     2020 3F                        SWI
```

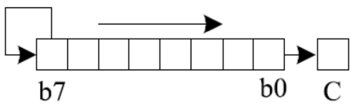
Shifts

Logical and Arithmetic <u>Right</u> Shift

- LSRx
  - **Logical** Shift **Right** for memory, A, B, or D
  - Bit shifted out into C CCR bit and **0 shifted in.**
  - Affects all four CCR bits – N always set to 0
  - Unsigned divide by 2



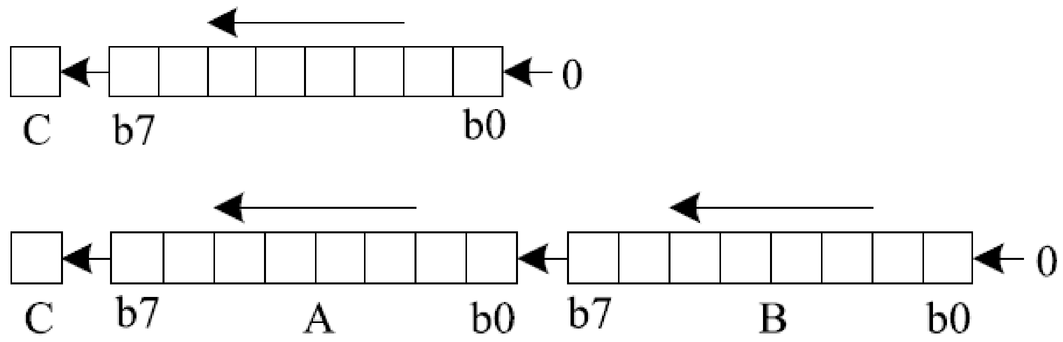
- ASRx
  - **Arithmetic** Shift **Right** for memory, A, B, or D
  - Bit shifted out into C CCR bit and **sign bit replicated.**
  - Affects all four CCR bits
  - Signed divide by 2



Shifts

Logical and Arithmetic <u>Right</u> Shift

- AS<u>L</u>x, LS<u>L</u>x
  - **Arithmetic and Logical** Shift **Left** for memory, A, B, or D
  - Bit shifted out into C CCR bit and **0 shifted in.**
  - Affects all four CCR bits
  - Unsigned/signed multiply by 2
  - ASRx is identical with LSLx (same opcodes)



Examples

C1 = 1100 0001

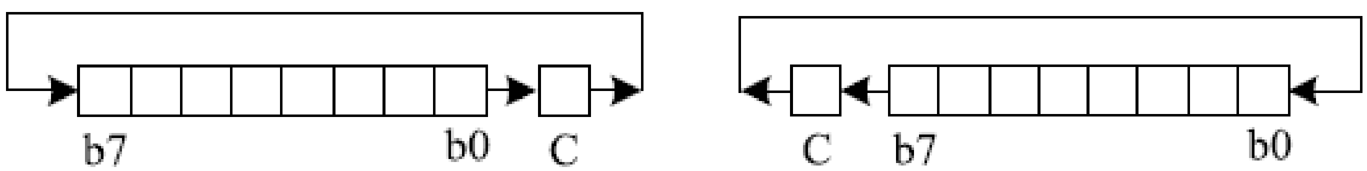
Code	A	N	Z	V	C	A (unsigned)	A (signed)
LDAA #$C1	C1	1	0	0	–	193	–63
ASLA	82	1	0	0	1	130	–126
ASLA	04	0	0	1	1	4	4

Code	A	N	Z	V	C	A (unsigned)	A (signed)
LDAA #$C1	C1	1	0	0	–	193	–63
ASRA	E0	1	0	0	1	224	–32
ASRA	F0	1	0	0	0	240	–16

Code	A	N	Z	V	C	A (unsigned)	A (signed)
LDAA #$C1	C1	1	0	0	–	193	–63
LSRA	60	0	0	1	1	96	96
LSRA	30	0	0	0	0	48	48

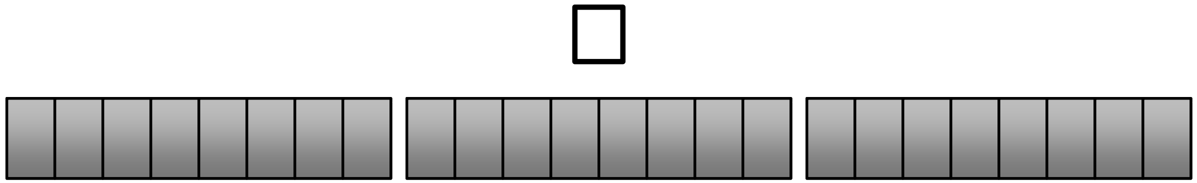
Rotates

- ROR<u>R</u>x, RO<u>L</u>x
  - **Rotate Right/Left** for memory, A, or B.
  - C CCR bit shifted in, bit shifted out goes to C CCR Bit.
  - Affects all four CCR bits.
- Rotates are used to extend shift operations to multi-precision numbers.



Example

Shift and Rotate



- Shift the signed three-byte number in addresses \$1000 – \$1002 right two bits

```
bits    EQU     2
num     EQU     $1000

        ORG     $2000
        LDAB    #bits
        LDX     num
again:
        ASR     0, X
        ROR     1, X
        ROR     2, X
        ; DBNE abdxys, rel9
        ;; Decrement Counter and Branch if not 0
        ; (cntr) – 1 → (cntr)
        ; if cntr not 0 then Branch
        DBNE    B, again
        SWI
```

BCD (Binary Coded Decimal)

- BCD is a very convenient way of storing numbers that will be sent to external 7-segment displays.
- In BCD, the hexadecimal number stored in memory looks like the decimal number.
  - i.e. if $27 appears in a memory location, it will be interpreted as 27_{10} in BCD.
- DAA (Decimal Adjust A)
  - DAA instruction corrects the answer.
    - The answer for the addition of 99h + 22h is BB
    - But DAA makes the addition be decimal addition (99 + 22 = 121).

Code	C	A	A (BCD)
LDAA #$99	–	99	99
ADDA #$22	0	BB	Invalid
DAA	1	21	21 w/ carry of 1

Questions?

Wrap-up

What we've learned

- **Multiplication** and **division** instructions.
  - Exponential filter
- Use **shift** and **rotate** instructions
  - Know difference between arithmetic and logical shift and rotate.

What to Come

- Boolean logic instructions