Algorithm for Multiplication

First, lets look at decimal multiplication:

```
      256
  ×   387
  -------
     1792
    2048
    768
  -------
    99072
```

7 × 256 is first intermediate result

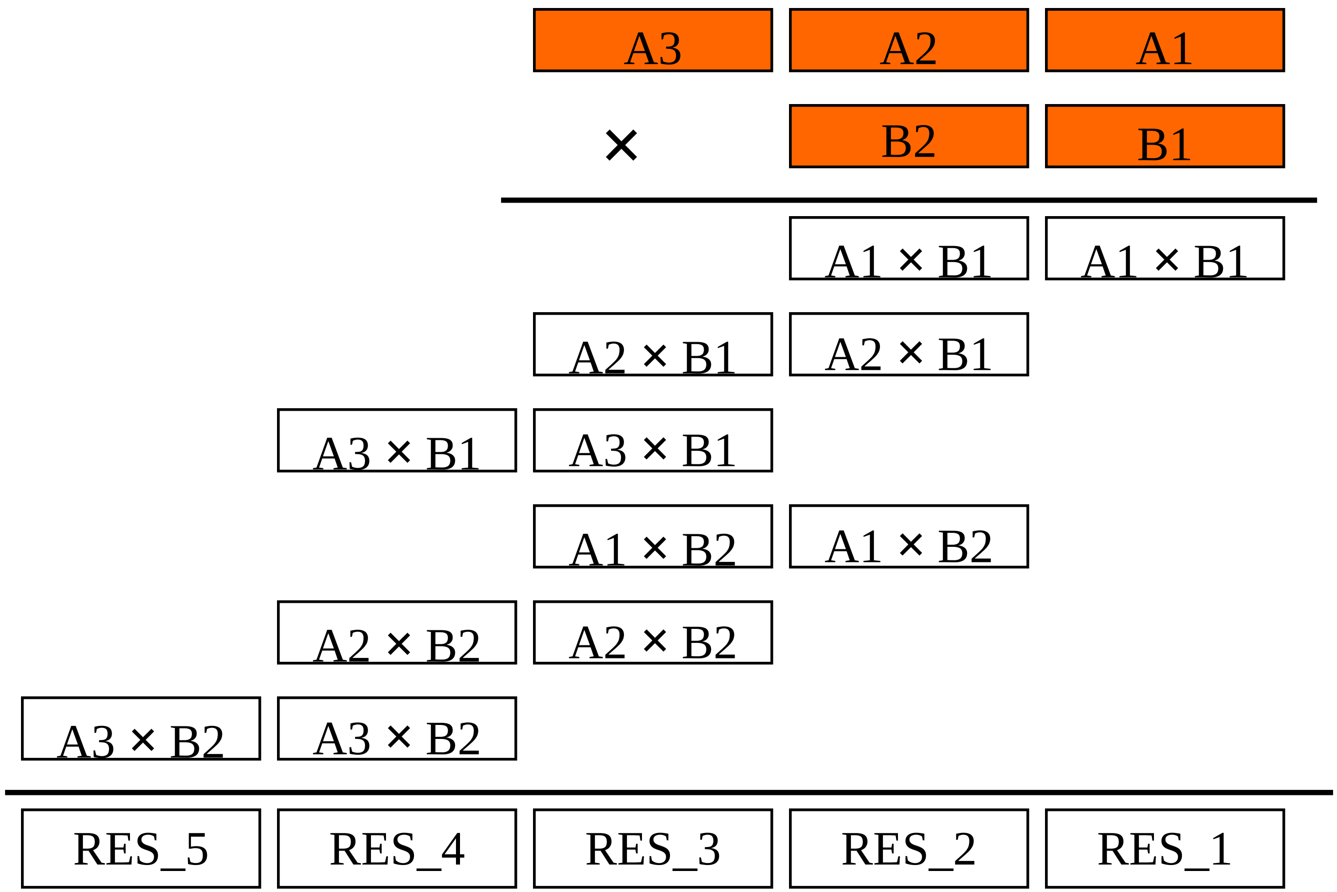
8 × 256 shifted and added to intermediate result

3 × 256 shifted and added to intermediate result

Final result

Now, lets look at an example for binary multiplication:

Instead of each bit, it is possible to multiply bytes:



Example: Write a program to multiply \$5678 by \$1234 and store the result at memory locations \$1100 to \$1103. Use the method that we just illustrated in previous slide.

```
LDAA   #$78        ;load ML into A
LDAB   #$34        ;load NL into B
MUL                ;multiply ML by NL
STD    $1102       ;store the partial product MLNL at 02 & 03

LDAA   #$56        ;load MH into B
LDAB   #$12        ;load NH into B
MUL                ;multiply MH by NH
STD    $1100       ;store the partial product MHNH at 00 & 01

LDAA   #$56        ; load MH into A
LDAB   #$34        ; load NL into B
MUL                ;generate partial product MHNL
```

* The following two instructions add M_HN_L to memory locations at \$1101 and \$1102

```
ADDD   $1101
STD    $1101
```

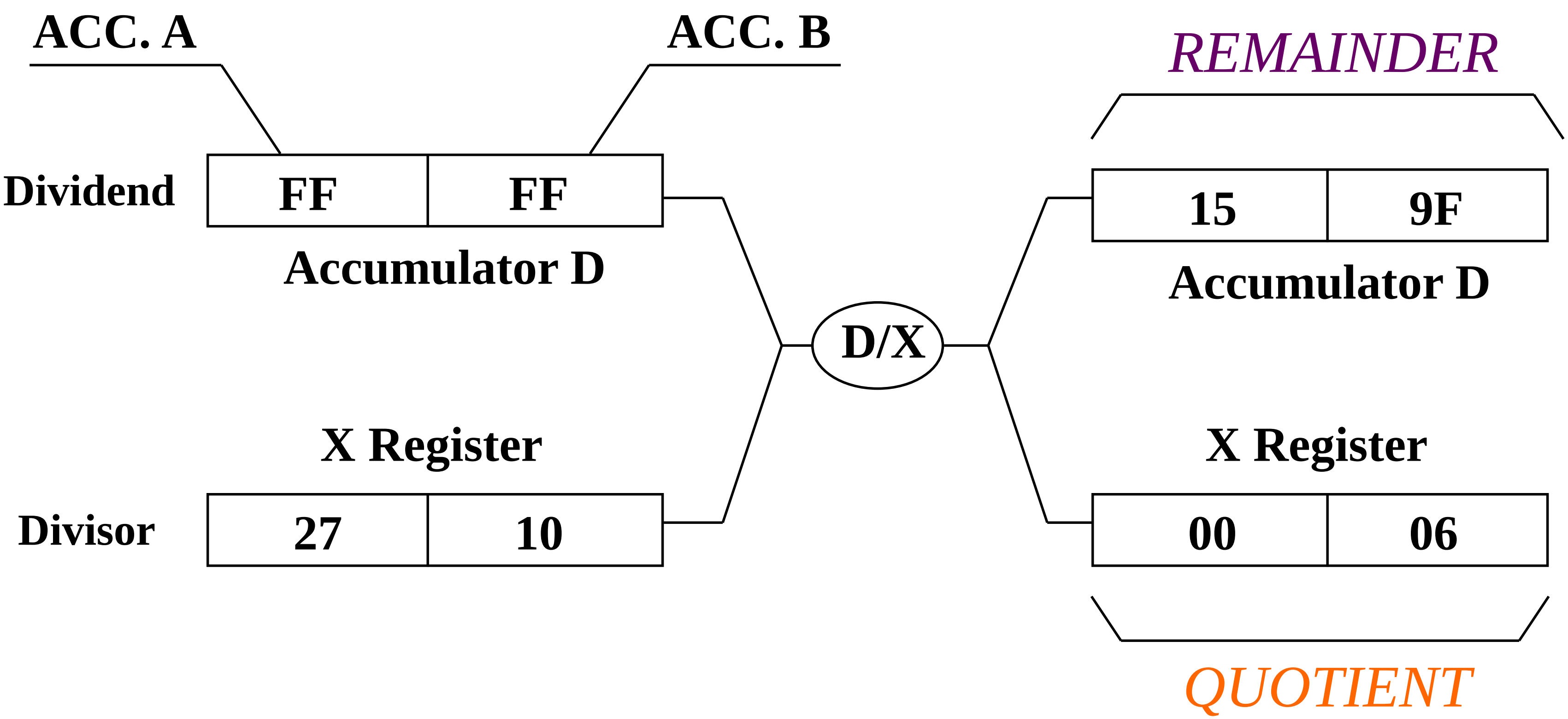
Program continued…

```
* The following three instructions add the C flag to memory location at $1100
        LDAA    $1100
        ADCA    #0
        STAA    $1100

        LDAA    #$78        ;load ML into A
        LDAB    #$12        ;load NH into B
        MUL                 ;generate the partial product MLNH
* The following two instructions add MLNH to memory locations at $1101 and $1102
        ADDD    $1101
        STD     $1101
* The following three instructions add the C flag to memory location at $1100
        LDAA    $1100
        ADCA    #0
        STAA    $1100

        END
```

Example For IDIV:

Program instructions:

```
LDD   #$FFFF
LDX   #$2710
IDIV
```



Example:

Write a program to convert the 16-bit binary number stored at location called ‘Bin’ to decimal format and store the result in 5 consecutive memory locations starting at address ‘Dec’. Each BCD digit is stored in one byte.

```
          ORG     $1000
          LDY     #Dec       ; set pointer at most significant digit
          LDD     Bin        ; get the 16-bit binary number
          LDX     #10        ;
          IDIV               ; compute the least significant digit
          STAB    4,Y        ; save the least significant digit in place
          XGDX               ; place the quotient in D to get the next digit
          LDX     #10
          IDIV                ; compute the next significant digit
          STAB    3,Y        ; save the next significant digit in place
          XGDX               ; place the quotient in D to get the next digit
          LDX     #10
          IDIV                ; compute the next significant digit
          STAB    2,Y        ; save the next significant digit in place
          XGDX               ; place the quotient in D to get the next digit
          LDX     #10
          IDIV                ; compute the next significant digit
          STAB    1,Y        ; save the next significant digit in place
          XGDX               ; swap the most significant digit to B
          STAB    0,Y        ; save the most significant BCD digit
          SWI
Bin       FDB     $F5AC      ; $F5AC = 62892
Dec       RMB     5
          END
```

Try to implement previous code segment using loop:

```
        ORG     $1000
        LDY     #Dec+4      ; set pointer at least significant digit
        LDX     Bin         ; get the 16-bit binary number

Loop    XGDX                ; put the binary number or quotient in D
        LDX     #10         ;
        IDIV                ; compute the next significant digit start from least
        STAB    1,Y-        ; save next significant digit in place & adjust pointer
        CPY     #Dec        ; as long as within the range, continue Loop
        BHI     Loop
        XGDX                ; get the MSD into acc. B
        STAB    0,Y         ; save it in place
        SWI

Bin     FDB     $F5AC
Dec     RMB     5

        END
```

Branch Instructions:

Unary Branches		
Mnemonic	Function	Operation
BRA	Branch Always	1 = 1
BRN	Branch Never	1 = 0

Simple Branches		
Mnemonic	Function	Operation
BCC	Branch if Carry Clear	C = 0
BCS	Branch if Carry Set	C = 1
BEQ	Branch if Equal	Z = 1
BMI	Branch if Minus	N = 1
BNE	Branch if not Equal	Z = 0
BPL	Branch if Plus	N = 0
BVC	Branch if Overflow Clear	V = 0
BVS	Branch if Overflow Set	V = 1

Branch Instructions *continued …*

Unsigned Branches		
Mnemonic	**Function**	**Operation**
BHI	Branch if Higher	$C + Z = 0$
BHS	Branch if Higher or Same	$C = 0$
BLO	Branch if Lower	$C = 1$
BLS	Branch if Lower or Same	$C + Z = 1$

Signed Branches		
Mnemonic	**Function**	**Operation**
BGE	Branch if Greater than or Equal	$N \oplus V = 0$
BGT	Branch if Greater than	$Z + (N \oplus V) = 0$
BLE	Branch if Less than or Equal	$Z + (N \oplus V) = 1$
BLT	Branch if Less than	$N \oplus V = 1$

- These are all short branch instructions. One-byte relative offset.
- There is another set of long branch instructions. Mnemonic-wise the only difference is that they start with letter L. They all have two-byte relative offset. E.g. LBEQ → Long Branch if Equal

Compare and Test Instructions:

Compare Instructions		
Mnemonic	**Function**	**Operation**
CBA	Compare A to B	(A) – (B)
CMPA	Compare A to Memory	(A) – (M)
CMPB	Compare B to Memory	(B) – (M)
CPD	Compare D to Memory	(D) – (M:M+1)
CPS	Compare SP to Memory	(SP) – (M:M+1)
CPX	Compare X to Memory	(X) – (M:M+1)
CPY	Compare Y to Memory	(Y) – (M:M+1)

Test Instructions		
Mnemonic	**Function**	**Operation**
TST	Test Memory for zero or minus	(M) - $00
TSTA	Test A for zero or minus	(A) - $00
TSTB	Test B for zero or minus	(B) - $00

Loop Primitive Instructions

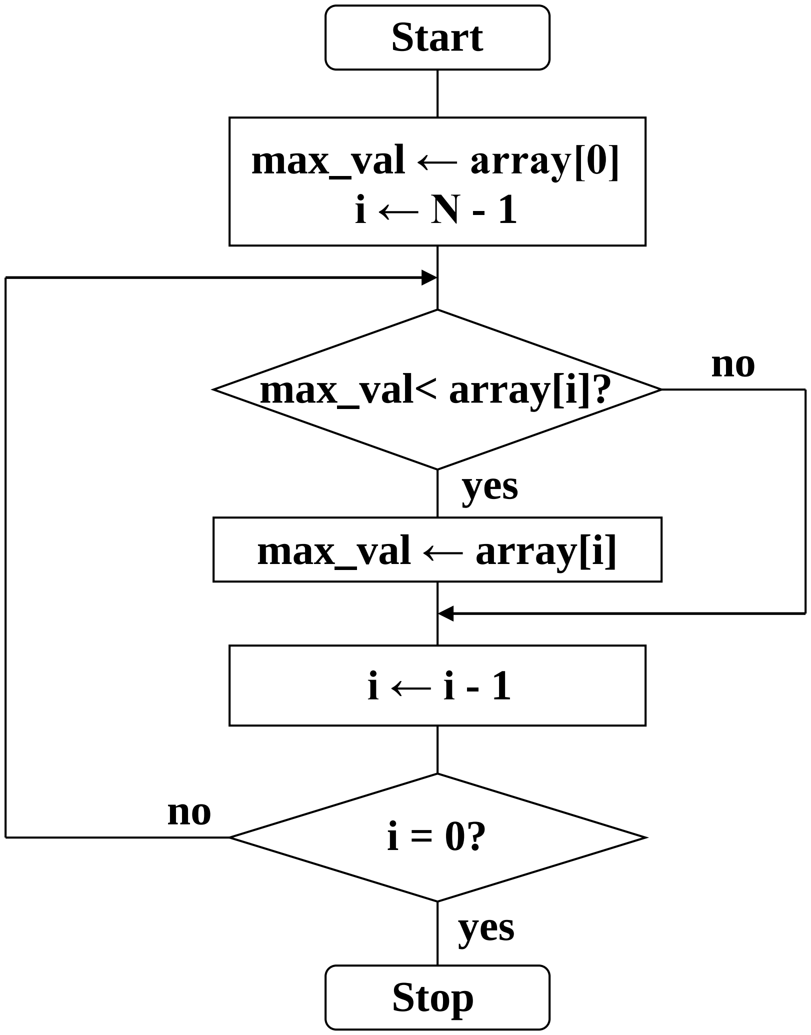
- A lot of the program loops are implemented by incrementing or decrementing a loop count.
- The branch is taken when either the loop count is equal to zero or not equal to zero, depending on the application.
- The HCS12 provides a set of loop primitive instructions for implementing this type of looping mechanism.
- These instructions test a counter value in a register or accumulator (A, B, D, X, Y, or SP) for zero or nonzero value as a branch condition.
- There are predecrement, preincrement, and test only versions of these instructions.
- The branch range is one byte relative from the instruction immediately following the loop primitive instruction.
- The syntax for these instructions is as follow:

```
DBEQ        X,loop
```

Loop Primitive Instructions		
Mnemonic	Function	Operation
DBEQ	Decrement counter and branch if = 0 (counter = A, B, D, X, Y, or SP)	counter ← (counter) – 1 If (counter) = 0' then branch else continue to next instruction
DBNE	Decrement counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP)	counter ← (counter) – 1 If (counter) ≠ 0' then branch else continue to next instruction
IBEQ	Increment counter and branch if = 0 (counter = A, B, D, X, Y, or SP)	counter ← (counter) + 1 If (counter) = 0' then branch else continue to next instruction
IBNE	Increment counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP)	counter ← (counter) + 1 If (counter) ≠ 0' then branch else continue to next instruction
TBEQ	Test counter and branch if = 0 (counter = A, B, D, X, Y, or SP)	If (counter) = 0' then branch else continue to next instruction
TBNE	Test counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP)	If (counter) ≠ 0' then branch else continue to next instruction

Example:

Write a program to find the maximum element from an array of N 8-bit elements using the primitive loop instruction.



```
        ORG     $1000
        LDAA    array           ; set array[0] as temporary array max
        STAA    max_val         ;
        LDX     #array+N-1      ; start from the end of array
        LDAB    #N-1            ; compute the least significant digit
loop    LDAA    max_val         ;
        CMPA    0,X             ; compare max_val with array[i]
        BHS     chk_end         ; no update if max_val is larger
        MOVB    0,X,max_val     ; update max_val
chk_end DEX                     ; move the array pointer
        DBNE    B,loop          ; decrement loop count, branch if not zero yet
        LDD     max             ; push the print parameter into stack
        PSHD                    ;
        LDD     #msg            ; store starting address of msg in acc. D
        LDX     printf          ; call printf subroutine
        JSR     0,X             ;
        LEAS    2,SP            ; balance the stack
        SWI                     ; return to D-Bug12 monitor

max     DB      0
max_val RMB     1
array   DB      1,3,5,6,19,41,53,28,13,42,76,14,20,54,64,74,29,33,41,45
msg     FCC     'The maximum element of the vector is %u'
        DB      $0D,$0A,0
printf  EQU     $EE88
N       EQU     20
        END
```

Decrementing and Incrementing Instructions:

Decrement Instructions

Mnemonic	Function	Operation
DEC	Decrement memory by 1	$M \leftarrow [M]$ - \$01
DECA	Decrement A by 1	$A \leftarrow [A]$ - \$01
DECB	Decrement B by 1	$B \leftarrow [B]$ - \$01
DES	Decrement SP by 1	$SP \leftarrow [SP]$ - \$01
DEX	Decrement X by 1	$X \leftarrow [X]$ - \$01
DEY	Decrement Y by 1	$Y \leftarrow [Y]$ - \$01

Increment Instructions

Mnemonic	Function	Operation
INC	Increment memory by 1	$M \leftarrow [M]$ + \$01
INCA	Increment A by 1	$A \leftarrow [A]$ + \$01
INCB	Increment B by 1	$B \leftarrow [B]$ + \$01
INS	Increment SP by 1	$SP \leftarrow [SP]$ + \$01
INX	Increment X by 1	$X \leftarrow [X]$ + \$01
INY	Increment Y by 1	$Y \leftarrow [Y]$ + \$01

Branch Instructions Based on Bit Condition

- In certain applications, one needs to make branch decisions based on the value of a few bits.
- The HCS12 provides two special conditional branch instructions for this purpose.
- The syntax of these two branch instructions are

```
BRCLR        opr,msk,rel
BRSET        opr,msk,rel
```

where

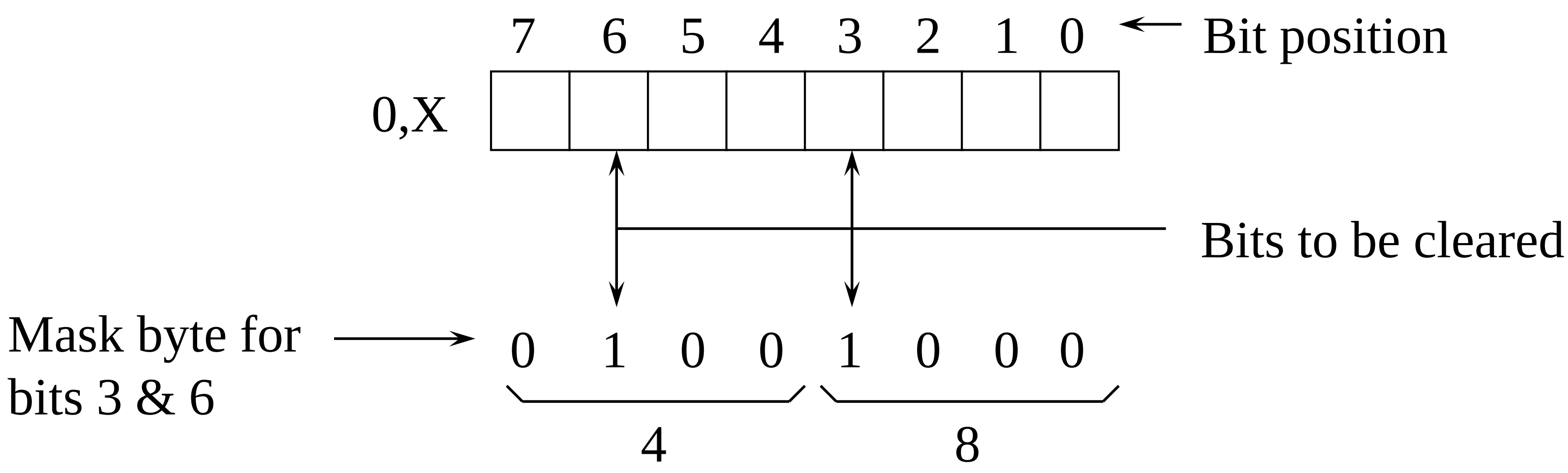
opr – specifies the memory location to be checked and can be specified using direct, extended, and all index addressing modes

msk – is an 8-bit mask that specifies the bits of the memory location to be checked. The bits to be checked correspond to those bit positions that are 1s in the mask.

rel – is the branch offset and it is specified in 8-bit relative mode.

Example:

```
Loop    LDAA    3,Y
          ⋮
        BRCLR   0,X $48 Loop
```

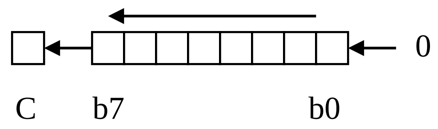
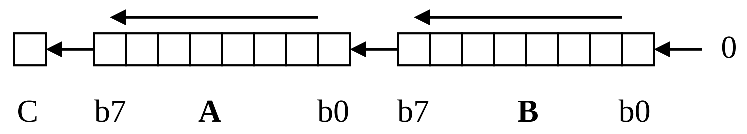
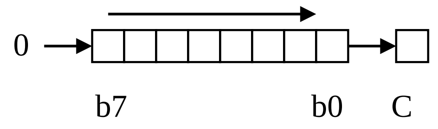
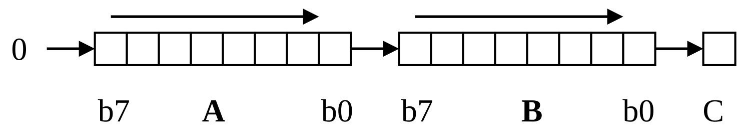


Example: Write a program to count the number of elements that are divisible by four in an array of N 8-bit numbers.

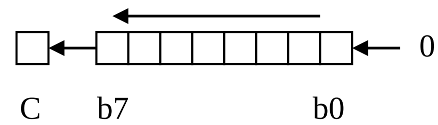
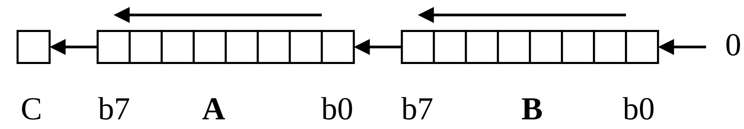
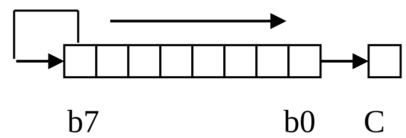
```
            ORG     $1000
            CLR     total                   ; initialize total to zero
            LDX     #array                  ; use X reg as array pointer
            LDAB    #N                      ; use acc. B as a loop counter
loop        BRCLR   0,X,$03,yes             ; if number divisible by 4
            BRA     chkend                  ; continue
yes         INC     total                   ; add one to the total
chkend      INX                             ; move the array pointer
            DBNE    B,loop                  ; decrement loop count, branch if not zero yet
            LDAB    total                   ; push the print parameter into stack
            CLRA                            ;
            PSHD                            ;
            LDD     #msg                    ; store starting address of msg in acc. D
            LDX     printf                  ; call printf subroutine
            JSR     0,X                     ;
            LEAS    2,SP                    ; balance the stack
            SWI                             ; return to D-Bug12 monitor

total       RMB     1
array       DB      2,3,4,8,12,13,19,24,33,32,20,18,53,52,80,82,90,94,100,102
msg         FCC     ‘The number of elements that are divisible by 4 are %u’
            DB      $0D,$0A,0
printf      EQU     $EE88
N           EQU     20
            END
```

Shift Instructions:

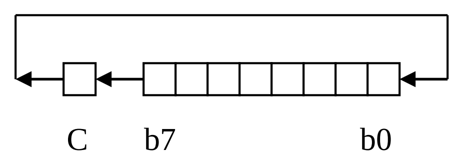
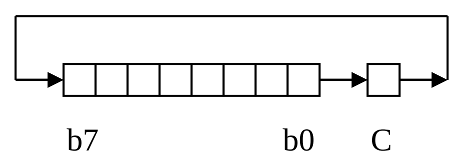
Logical Shift Instructions

Mnemonic	Function	Operation
LSL <opr> LSLA LSLB	Logical shift left memory Logical shift left A Logical shift left B	
LSLD	Logical shift left D	
LSR <opr> LSRA LSRB	Logical shift right memory Logical shift right A Logical shift right B	
LSRD	Logical shift right D	

Arithmetic Shift Instructions

Mnemonic	Function	Operation
ASL <opr> ASLA ASLB	Arithmetic shift left memory Arithmetic shift left A Arithmetic shift left B	
ASLD	Arithmetic shift left D	
ASR <opr> ASRA ASRB	Arithmetic shift right memory Logical shift right A Logical shift right B	

Rotate Instructions

Mnemonic	Function	Operation
ROL <opr> ROLA ROLB	Rotate left memory thru carry Rotate left A through carry Rotate left B through carry	
ROR <opr> RORA RORB	Rotate right memory thru carry Rotate right A through carry Rotate right B through carry	

Example: Write a program to count the number of 0s contained in memory locations $1000 ~ $1001 and save the result at memory location $1005.

```
            ORG     $1000
            DW      $2355
zero_cnt    RMB     1
lp_cnt      RMB     1


            ORG     $1010
            CLR     zero_cnt            ; initialize the zero count to 0
            MOVB    #16,lp_cnt          ; initialize lp_cnt to 16
            LDD     $1000               ; place the 16-bit number in acc. D
loop        LSRD                        ;
            BCS     chkend              ; branch if the least sig. bit is 1
yes         INC     zero_cnt            ; add one to the total
chkend      DEC     lp_cnt              ;
            BNE     loop                ; have we checked all 16 bit yet?
            MOVB    zero_cnt,$1005      ; store the result in location $1005
            SWI                         ; return to D-Bug12 monitor

            END
```

Example: Write a subroutine called mult10 such that every time the subroutine is called, it will multiply the content of accumulator D by 10 using shift command.

```
        ORG     $1000
         :
         :
        LDD     #20
        JSR     mult10
         :
         :
Mult10  LSLD                    ; multiply D by 2
        STD     temp            ; save DX2 in temp
        LSLD                    ; multiply D by 2 again
        LSLD                    ; create D X 8
        ADDD    temp            ; sum DX2 and DX8 to get DX10
        RTS                     ;


Temp    RMB     2

        END
```

Boolean Logic Instructions

Mnemonic	Function	Operation
ANDA <opr>	AND A with memory	$A \leftarrow (A) \bullet (M)$
ANDB <opr>	AND B with memory	$B \leftarrow (B) \bullet (M)$
ANDCC<opr>	AND CCR with memory	$CCR \leftarrow (CCR) \bullet (M)$
EORA <opr>	Exclusive OR A with memory	$A \leftarrow (A) \oplus (M)$
EORB <opr>	Exclusive OR B with memory	$B \leftarrow (B) \oplus (M)$
ORA <opr>	OR A with memory	$A \leftarrow (A) + (M)$
ORB <opr>	OR B with memory	$B \leftarrow (B) + (M)$
ORCC <opr>	OR CCR with memory	$CCR \leftarrow (CCR) + (M)$
CLC	Clear C bit in CCR	$C \leftarrow 0$
CLI	Clear I bit in CCR	$I \leftarrow 0$
CLV	Clear V bit in CCR	$V \leftarrow 0$
COM <opr>	One's complement memory	$M \leftarrow \$FF - (M)$
COMA	One's complement A	$A \leftarrow \$FF - (A)$
COMB	One's complement B	$B \leftarrow \$FF - (B)$
NEG <opr>	Two's complement memory	$M \leftarrow \$00 - (M)$
NEGA	Two's complement A	$A \leftarrow \$00 - (A)$
NEGB	Two's complement B	$B \leftarrow \$00 - (B)$

Bit Test and Bit Manipulate Instructions

Mnemonic	Function	Operation
BCLR <opr>[1],msk8	Clear bits in memory	$M \leftarrow (M) \bullet (\overline{mm})$
BITA <opr>[2]	Bit test A	$(A) \bullet (M)$
BITB <opr>[2]	Bit test B	$(B) \bullet (M)$
BSET <opr>[1],msk8	Set bits in memory	$M \leftarrow (M) + (mm)$

Note: 1. <opr> can be specified using direct, extended, and indexed (exclude indirect) addressing modes.
2. <opr> can be specified using all except relative addressing modes for BITA and BITB.
3. msk8 is an 8-bit value.

Example: Write a set of instructions to set bits 2, 5, and 7 in the memory location \$2500 and clear bits 1, 3, 6, and 7 in memory location \$2510.

```
BSET    $2500,$A4
BCLR    $2510,$CA
```

Example:

Write a program to determine the largest of the twenty 8-bit numbers stored in consecutive memory locations starting at address ‘Nums’. Save the largest number at memory location called ‘Max’. All the numbers are positive.

```
            NAM     EXAMPLE
            ORG     $1000
Nums        FCB     250,28,38,168,251,222,38,55,183,232
            FCB     12,88,198,209,246,77,253,9,133,200
Max         RMB     1

            ORG     $1050









            SWI
            END
```

Example:

Write a program to determine the largest of the twenty 8-bit numbers stored in consecutive memory locations starting at address ‘Nums’. Save the largest number at memory location called ‘Max’. All the numbers are positive.

```
            NAM     EXAMPLE
            ORG     $C000
Nums        FCB     250,28,38,168,251,222,38,55,183,232
            FCB     12,88,198,209,246,77,253,9,133,200
Max         RMB     1

            ORG     $C050
            LDAB    #19         ; set the counter
            LDX     #Nums       ; set the pointer to first number
            LDAA    0,X         ; put the first number in Acc. A
LOOP        CMPA    1,X         ; check the next number with A, if A is bigger or
            BHS     Next          ; equal go to next number, otherwise replace the
            LDAA    1,X             ; number with the content of A.
Next        INX
            DECB
            BNE     LOOP            ; repeat the process till all numbers are checked
            STAA    Max         ; put the largest number in memory location Max
            SWI
            END
```

Finding the Square Root

Consider the following series:

$$\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$$

e.g. if $\mathrm{n} = 5$,

$$1+2+3+4 = \frac{5(4)}{2} = 10$$

for $\mathrm{n} = 8$,

$$1+2+3+4+5+6+7 = \frac{8(7)}{2} = 28$$

Finding the Square Root *continued …*

Lets rewrite the series as follow :

$$\sum_{i=0}^{n-1} 2i = n^2 - n \quad \Rightarrow \quad n^2 = \sum_{i=0}^{n-1} (2i+1)$$

Suppose, we want to compute the square root of q, and n is the integer value that is closest to true square root.

One of the three following relationships is satisfied :

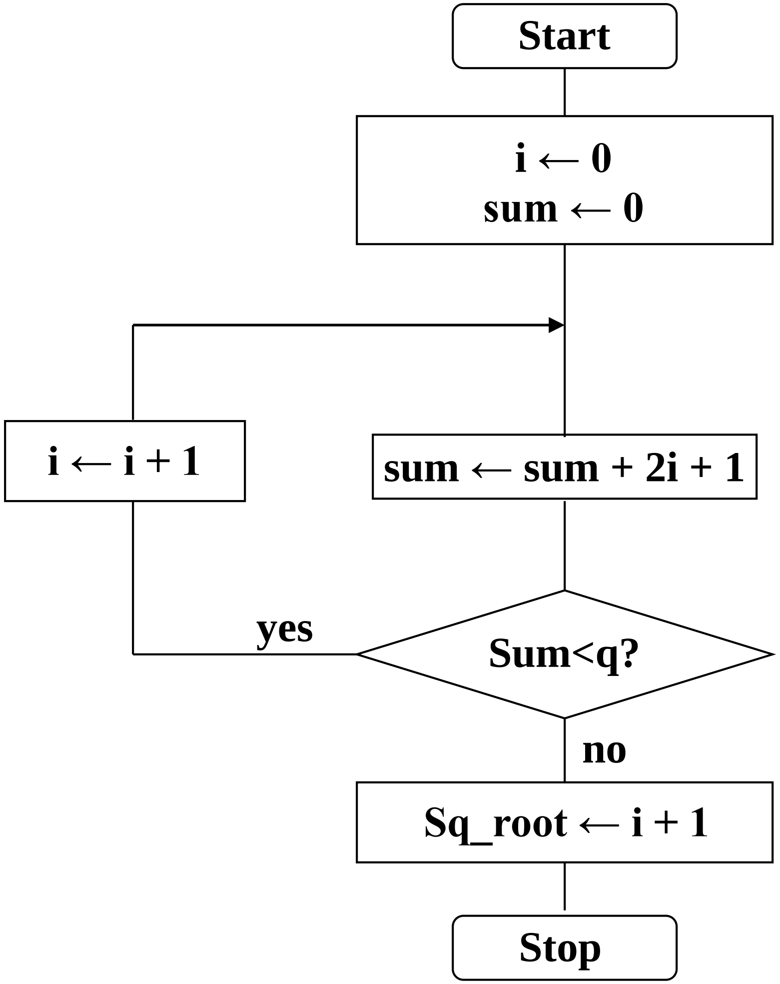
$$n^2 < q$$

$$n^2 = q$$

$$n^2 > q$$

Finding the Square Root *continued …*

Algorithm for finding the square root of integer *q*



Finding the Square Root *continued …*

Accuracy: lets assume q = 820, this procedure creates n = 28. which gives n^2=784 and is not very accurate.

Remedy: If q is less than two-byte, then multiply it with 10000 and find the square root of the new number. Then divide the answer by 100 and quotient is integer part of the square root and the remainder is the first two decimal digits.

$$q = 60{,}000 \;\Rightarrow\; q_1 = 60{,}000 \times 10{,}000 = 600{,}000{,}000$$

$$n_1 = 24496 \;\Rightarrow\; n = \frac{n_1}{100} = 244 \quad and \quad \frac{96}{100} = 0.96$$

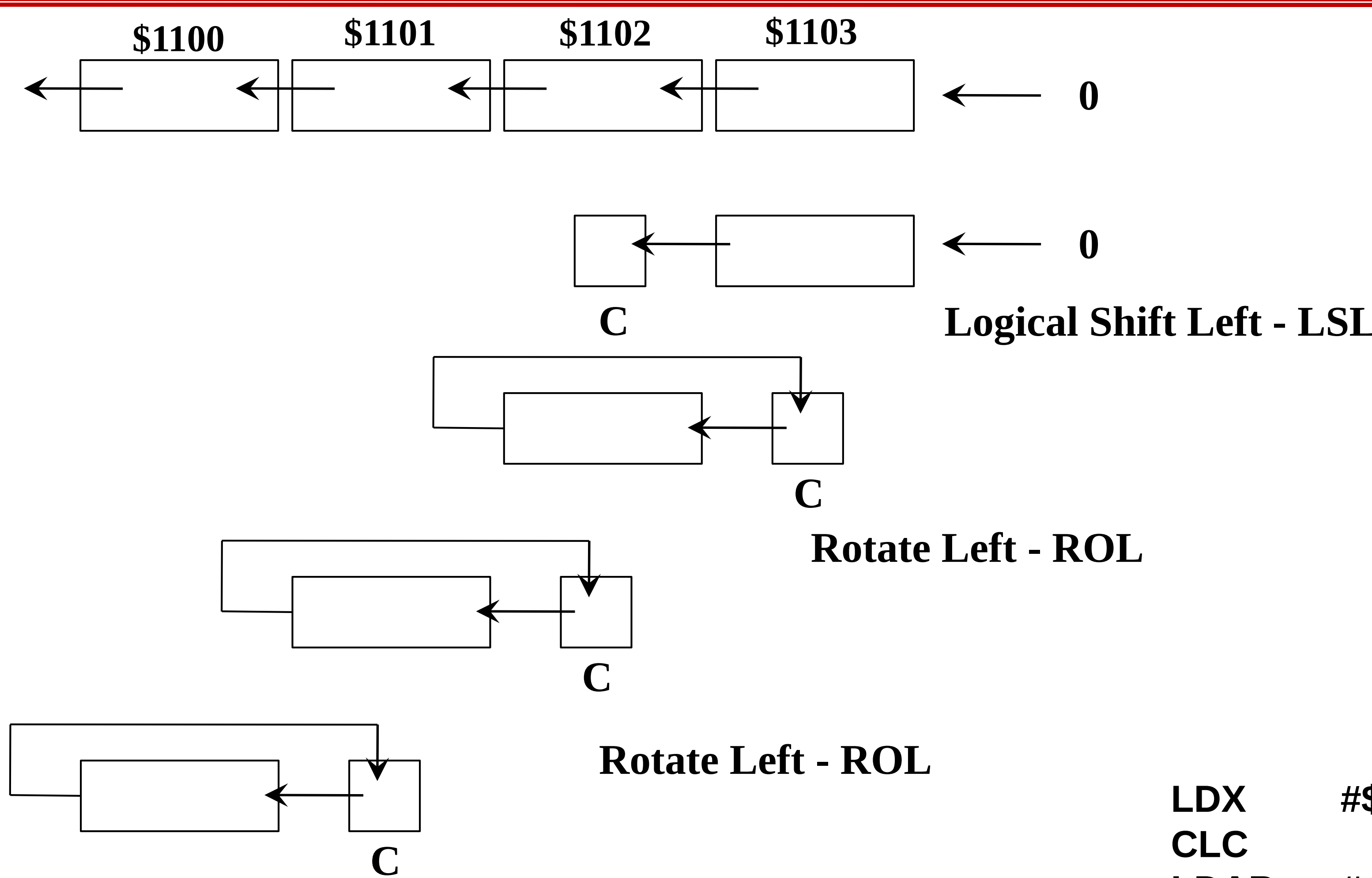
$$Therefore, \quad n = 244 \;\Rightarrow\; n^2 = 59536$$

$$but\ if \quad n = 244.96 \;\Rightarrow\; n^2 = 60005.4$$

Project 2 – Finding the Square Root of a number

- Ask user to enter a number up to 4,294,967,295 (2^{32} - 1).
- Make sure number is correct and within the range.
- Pack the number into binary (up to 4-bytes).
- If number is 2-byte or less, multiply it by 10000.
- If number is 3-bytes, multiply it by 100.
- Find square root of the number and adjust for decimal point.
- Output the number and its square root to terminal in appropriate way.
- Ask if user wants to try again.

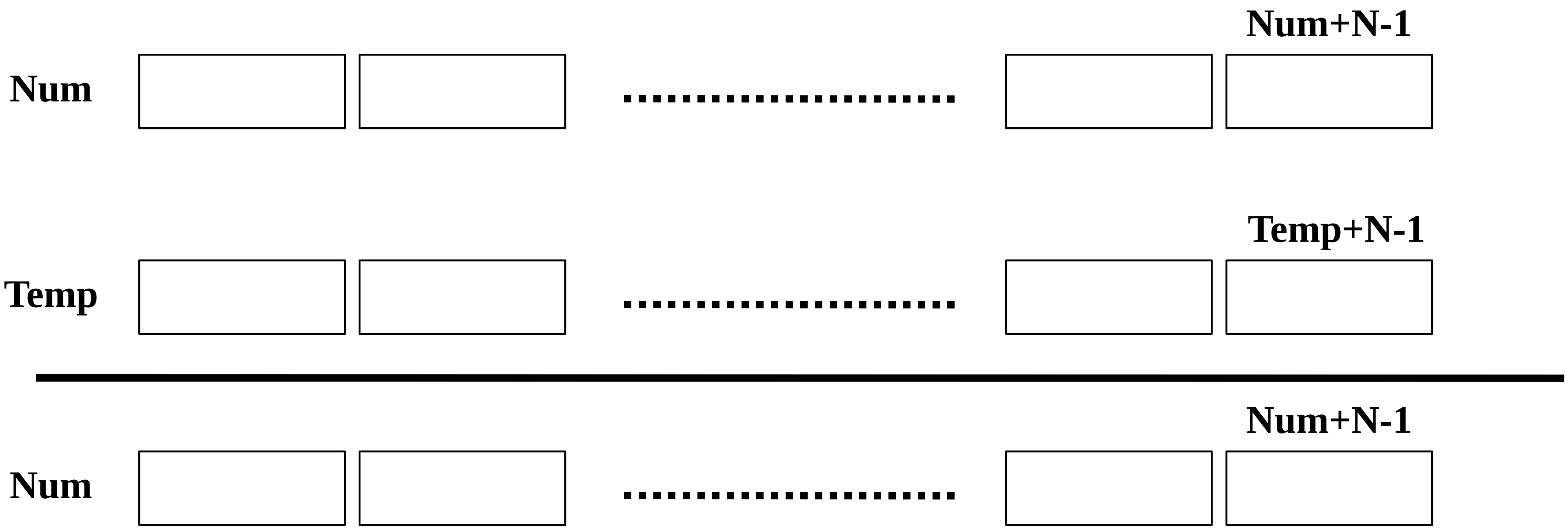
Shift 4-byte Number to the Left by 1-bit



Rotate Left - ROL

```
          LDX     #$1103
          CLC
          LDAB    #4
L1        ROL     1,X-
          DBNE    B,L1
```

Add two N-byte Numbers in Memory




```
        LDX     #Num+N-1        ; set pointer to LS Byte
        LDY     #Temp+N-1       ; set pointer to LS Byte
        CLC
        LDAB    #N              ; set counter
L1      LDAA    0,X             ; add same order bytes
        ADCA    1,Y-             ; together w/carry and
        STAA    1,X-              ; store it back
        DBNE    B,L1            ; do it N-times
```

How to adjust 2- & 3-byte number to 4-byte

Bnum	Bnum+1	Bnum+2	Bnum+3

```
        LDD     Bnum        ; get MS 2-bytes
        BEQ     two         ; if 0, then mult with 10000
        CMPA    #0          ; get MS-byte
        BEQ     one         ; if 0, then mult by 100
        BRA     find        ; otherwise, calculate square root
two     LDD     Bnum+2      ; get two-byte number
        LDY     #10000      ; multiply it by 10000
        EMUL                ;
        STD     Bnum+2      ; store the result in Bnum
        STY     Bnum        ;
        BRA     find        ; calculate square root
one     LDD     Bnum+2      ;get 1st 2-bytes
        LDY     #100        ; multiply by 100
        EMUL                ;
        STD     Bnum+2      ; store 1st part of the multiplication
        LDD     Bnum        ; pick up last two byte of the number
        STY     Bnum        ; store 2nd part of mult.
        LDY     #100        ; multiply by 100 2nd part
        EMUL                ;
        ADDD    Bnum        ; add to result last 2-byte of 1st product
        STD     Bnum        ; store the last 2-byte of the result
find                        ; calculate square root
```

How to Remember and Adjust for Decimal

Use a flag to differentiate between 3 cases:

1. Flag = 0 $\rightarrow$ no decimal
2. Flag = 1 $\rightarrow$ one decimal digit
3. Flag = 2 $\rightarrow$ two decimal digit

How would you take care of the following result?

378.09

Be careful on the use of %u conversion on '***printf***' function!

Quiz 1
Converting degree Fahrenheit to Celsius

- Ask user to enter a temperature in Fahrenheit from 32 up to 255.
- Read in the temperature.
- Pack the numbers into binary (1 byte).
- Convert it to degree Celsius.
- Output the temperature to the terminal as follow:

  XXX ^{o}F YYY ^{o}C

- Ask if user wants to try again.

$$^{o}C = \frac{^{o}F - 32}{1.8} \Rightarrow {}^{o}C = \frac{\left(^{o}F - 32\right)\times 10}{18}$$

$$^{o}C = \frac{^{o}F \times 5 - 160}{9}$$

```
LDAB  F          ; get Fahrenheit value
LDAA  #5         ; multiply By 5
MUL              ;
SUBD  #160       ; subtract 160
LDX   #9         ; divide by 9
IDIV             ;
PSHX             ; push Celsius value into stack
CLRA             ; push Fahrenheit value into stack
LDAB  F          ;
PSHD             ;
LDD   #outmsg    ; output the result
LDX   printf     ;
JSR   0,X        ;
LEAS  4,SP       ; balance the stack
LDD   #repmsg    ; ask for repeat
LDX   printf     ;
JSR   0,X        ;
LDX   getchar    ; check answer
JSR   0,X        ;
CMPB  #'y'       ; act accordingly
BEQ   start      ;
SWI
```

How to Take Care of Negative Numbers

```
        LDAB    F               ; get Fahrenheit value
        LDAA    #5              ; multiply By 5
        MUL                      ;
        SUBD    #160            ; subtract 160
        BCC     cont            ; if number isn't negative continue
        COMA                    ; complement the result
        COMB                     ;
        ADDD    #1                ;
        MOVB    #$2D,sign       ; incorporate the negative sign
cont    LDX     #9              ; divide by 9
        IDIV                     ;
        PSHX                    ; push Celsius value into stack
        CLRA                    ; push Fahrenheit value into stack
        LDAB    F                ;
        PSHD                      ;
        LDD     #outmsg                     ; output the result
        LDX     printf           ;
        JSR     0,X               ;
        LEAS    4,SP            ; balance the stack
```

How to Take Care of Negative Numbers *cont'd…*

```
        LDD     #repmsg                 ; ask for repeat
        LDX     printf            ;
        JSR     0,X               ;
        LDX     getchar           ; check answer
        JSR     0,X               ;
        CMPB    #'y'              ; act accordingly
        BEQ     start             ;
        SWI

outmsg  DB      CR,LF
        FCC     '       %u F is equal to '
sign    DB      '+'
        FCC     '%u C'
        DB      CR,LF,0
repmsg  FCC     'Would you like to enter another value (y/n)?'
        DB      CR,LF,0
CR      EQU     $0D
LF      EQU     $0A
Printf  EQU     $EE88
```

How to Add Decimal Point

$$^{o}C=\frac{^{o}F-32}{1.8}\ \Rightarrow\ ^{o}C=\frac{\left(^{o}F-32\right)\times 50}{9}=\frac{^{o}F\times 50-1600}{9}$$

$$\frac{^{o}C}{10}={}^{o}C(\text{integer})\ \text{ and remainder is decimal number}$$

```
        LDAB    F               ; get Fahrenheit value
        LDAA    #50             ; multiply By 5
        MUL                     ;
        SUBD    #1600           ; subtract 160
        BCC     cont            ; if number isn't negative continue
        COMA                    ; complement the result
        COMB                    ;
        ADDD    #1              ;
cont    MOVB    #'-',sign       ; incorporate the negative sign
        LDX     #9              ; divide by 9
        IDIV                    ;
        XGDX                    ; separate integer and decimal
        LDX     #10             ;
        IDIV                    ;
        PSHD                    ; push decimal point of Celsius
        PSHX                    ; push Celsius value into stack
```

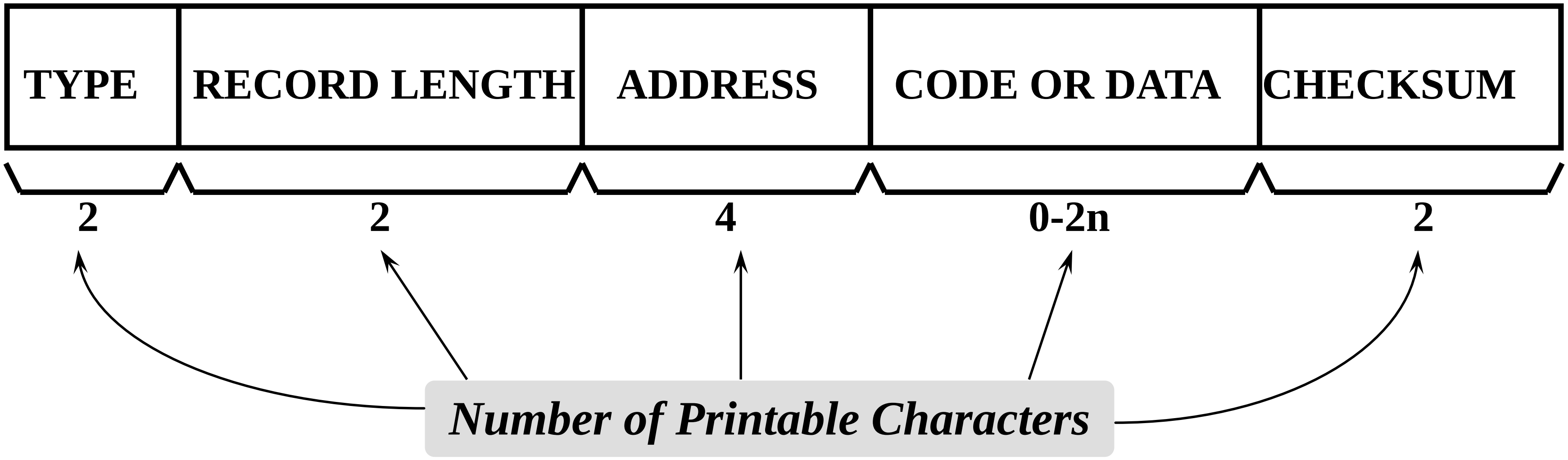
How to Add Decimal Point *cont'd…*

```
        CLRA                        ; push Fahrenheit value into stack
        LDAB    F                   ;
        PSHD                        ;
        LDD     #outmsg             ; output the result
        LDX     printf              ;
        JSR     0,X                 ;
        LEAS    6,SP                ; balance the stack
        LDD     #repmsg             ; ask for repeat
        LDX     printf              ;
        JSR     0,X                 ;
        LDX     getchar             ; check answer
        JSR     0,X                 ;
        CMPB    #'y'                ; act accordingly
        BEQ     start               ;
        SWI

outmsg  DB      CR,LF
        FCC     '       %u.0 F is equal to '
sign    DB      '+'
        FCC     '%u.%u C'
        DB      CR,LF,0
```

S-Record

- Motorola uses character strings called S-records to encode programs and data for serial transmission between computers.
- Each string begins with upper case S and is divided into five fields.
- There are ten possible types of s-records (S0-S9).
- Only three s-record types are used with the HCS12 microcontroller (S0, S1, and S9).
- An s-record encodes the object code into a printable ASCII format.
- The last byte of each s-record is a *checksum*.
- This value is used for error checking to ensure that the instruction has been transmitted error free.

Five Fields of S-record



S0-record: An S0-record is always the first character string and is referred to as the *header record*. The address field may be all zeros in S0 record.

Example:

S00A00006368342E4F5554D0

S0	Indicates the record is a header record.
0A	The number of bytes (in Hex) to follow in this string.
0000	Address field is all zeros.
63	
68	
34	
2E	ASCII representation for ch4.OUT
4F	
55	
54	
D0	checksum.

Checksum: The checksum is determined first by adding all the bytes in the record length, address, and code/data fields, then obtaining the one's complement of the least significant byte of the sum.

0A + 00 + 00 + 63 + 68 + 34 + 2E + 4F + 55 + 54 = 22F

Least Significant Byte of Sum = $2F

1's Complement of $2F = $D0 which is the *Checksum* value.

S1-record: S1-records contain the instructions and data to be operated on by the microcontroller.

Example:

S119C400BDFFCD8131270E8132270A813327068134270226EB3FBF

S1	Indicates the record contains code/data.
19	There are 25 bytes of binary data to follow.
C400	Starting address where the data is stored.
BD FF CD	Machine code for first instruction, JSR INCHAR.
81 31	Machine code for CMPA #$31.
. . .	Machine code for other instructions in the program.
3F	Machine code for SWI.
BF	Checksum.

19 + C4 + 00 + ... + 26 + EB + 3F = 840   1’ Comp. of $40 = $BF

S9-record: The S9-record is the termination string. The address field may also be all zeroes and there is no code/data field.

Example:

S9030000FC

S9	Indicates a termination record.
03	There are three bytes to follow.
0000	Address field is all zeroes.
FC	Checksum.

Calculating Checksum:

03 + 00 + 00 = 03

1’Complement of $03 = $FC

Sample of S-records

```
S015000046696C653A2074656D706C63642E61736D0AAE
S11310001610BD16102216104B868016108FCE11A6
S11310104A16111586C016108FCE115916111506D1
S11310201003CC1121FEEE881500FEEE8415007B22
S11310301159C030860A127B1120FEEE8415007B04
S1131040115AC030FB11207B11203DF61120865A25
S113105012CE00051810B7C5C30140CE000A1810FF
S1131060CB307B1165B7C5CE000A1810CB307B118D
S113107063B7C5CE000A1810CB307B1162B7C5C167
S1131080002706CB307B11613D180B2011613D36E2
S11310904D32014C320284F044448A025A32A7A7EA
S11310A0A74D320232840F48484C32028A025A3227
S11310B0A7A7A74D3202CD00F00326FD3D180BFF74
S11310C000331610E0862816108F860C16108F86B3
S11310D00616108F860116108FCD27100326FD3DAE
S11310E0CDEA600326FD3D364C32014C320284F0D9
S11310F044448A035A32A7A7A74D320232840F48C8
S1131100484C32028A035A32A7A7A74D3202CD00B7
S1131110F00326FD3DA63027061610E70611153DFF
S11311210D20456E74657220646567726565652043A0
S1131131656C636975732062657477656656E2030CB
S113114130202D2035303A0D005468652054656DEA
S10B1151702E206973203A009E
S109115B20DF43202D20DB
S10411642E58
S107116620DF46003C
S9030000FC
```

How to Divide Two 32-bit Numbers

$$EDIV \rightarrow \frac{Y:D}{X} \Rightarrow \begin{cases} Quotient \rightarrow X \\ \mathrm{Re}\,mainder \rightarrow D \end{cases}$$

$$2^{32} - 1 = 4{,}294{,}967{,}295$$

	3,589,204,795	$\boxed{0}$	counter
Carry Clear	- 1,000,000,000		
	2,589,204,795	$\boxed{1}$	
Carry Clear	- 1,000,000,000		
	1,589,204,795	$\boxed{2}$	
Carry Clear	- 1,000,000,000		
	589,204,795	$\boxed{3}$	Most significant digit
Carry Set	- 1,000,000,000		

dnum

dnum+9

How to Divide Two 32-bit Numbers *cont'd…*

589,204,795	0		1,000,000,000
- 100,000,000			100,000,000
489,204,795	1		10,000,000
- 100,000,000			1,000,000
389,204,795	2		100,000
- 100,000,000			10,000
289,204,795	3		1,000
- 100,000,000			100
189,204,795	4		10
- 100,000,000			
89,204,795	5	2nd most significant digit	
- 100,000,000			
CARRY SET			

Continue the process till you get to the unit digit – this is called

Successive Subtraction Method

<u>Project 3 – Multiplying two 10-digit numbers</u>

- Ask user to enter two numbers up to 4,294,967,295.
- Make sure numbers are correct and within the range.
- Pack the numbers into binary (up to 4-bytes).
- Multiply the numbers to get a binary number up to 8-bytes.
- Generate 8-byte long power of tens and store them into memory.
- Use Successive Subtraction method to convert 8-byte binary result into 20-digit decimal number.
- Output the result as follow:

$$85000 \times 125000 = 10625000000$$

- Ask if user wants to try again.

<u>Bonus:</u> Try to output the result as follow:

$$85{,}000 \times 125{,}000 = 10{,}625{,}000{,}000$$

no extra spaces are displayed.