

Key to Practical 7

Calculator (Part 4)

Step 1

```

Itoa      ; Save registers on the stack.
          movem.l d0/a0,-(a7)

          ; If D0.W is positive or null, branch to \positive.
          tst.w   d0
          bpl     \positive

\negative ; Otherwise write the '-' character into the string.
          ; (And make A0.L point to the next character.)
          move.b  #'-',(a0)+

          ; 0 - D0.W -> D0.W
          neg.w   d0

\positive ; Convert D0.W
          jsr     Uitoa

\quit     ; Restore registers from the stack and return from subroutine.
          movem.l (a7)+,d0/a0
          rts

```

Step 2

```

          ; =====
          ; Vector Initialization
          ; =====

          org      $0

vector_000 dc.l     $ffb500
vector_001 dc.l     Main

          ; =====
          ; Main Program
          ; =====

          org      $500

Main      ; Display the following message: "Enter an expression:"
          ; (The message is displayed in the top left-hand corner.)
          movea.l  #sInput,a0
          clr.b    d1
          clr.b    d2
          jsr      Print

          ; Get the user expression.
          ; (The string is stored in the memory location sBuffer.)
          ; (It is displayed two lines below the previous message.)
          movea.l  #sBuffer,a0
          addq.b   #2,d2
          move.l   #60000,d3
          move.l   #8000,d4

```

```

jsr      GetInput

; Remove spaces.
jsr      RemoveSpace

; Display the following message (two lines below): "Result:"
movea.l  #sResult,a0
addq.b   #2,d2
jsr      Print

; Increment the line number by 2.
addq.b   #2,d2

; Calculate the result (result -> D0.L).
; If error, branch to \error.
movea.l  #sBuffer,a0
jsr      GetExpr
bne      \error

\noError ; No error occurs.
; Convert the integer result into a string.
; The string is stored in the address sBuffer (A0 = sBuffer).
jsr      Itoa

; Display the result and exit.
jsr      Print
bra      \quit

\error   ; An error occurs.
; Display an error message.
movea.l  #sError,a0
jsr      Print

\quit    ; Breakpoint.
illegal

; =====
; Subroutines
; =====

; ...
; ...
; (All the subroutines)
; ...
; ...

GetInput incbin  "GetInput.bin"
PrintChar incbin "PrintChar.bin"

; =====
; Data
; =====

sInput   dc.b    "Enter an expression:",0
sResult  dc.b    "Result:",0
sError   dc.b    "Error",0
sBuffer  ds.b    60

```

Step 3Split **NextOp** into two subroutines: **NextOp1** and **NextOp2**.

```

NextOp1      ; If the character is null (end of string),
              ; the string does not contain any operators.
              ; A0 points to the null character. Branch to \quit.
tst.b    (a0)
beq      \quit

              ; Compare the character to the '+' and '-' operators.
              ; If the character is an operator, branch to \quit.
              ; (A0 holds the address of the operator.)
cmpi.b   #'+',(a0)
beq      \quit

cmpi.b   #'-',(a0)
beq      \quit

              ; Go on with the next character.
addq.l   #1,a0
bra      NextOp1

\quit        ; Return from subroutine.
rts

```

```

NextOp2      ; If the character is null (end of string),
              ; the string does not contain any operators.
              ; A0 points to the null character. Branch to \quit.
tst.b    (a0)
beq      \quit

              ; Compare the character to the '*' and '/' operators.
              ; If the character is an operator, branch to \quit.
              ; (A0 holds the address of the operator.)
cmpi.b   #'*',(a0)
beq      \quit

cmpi.b   #'/',(a0)
beq      \quit

              ; Go on with the next character.
addq.l   #1,a0
bra      NextOp2

\quit        ; Return from subroutine.
rts

```

Split **GetNum** into two subroutines: **GetNum1** and **GetNum2**.

```

GetNum1      ; Save registers on the stack.
              movem.l d1/a1-a2,-(a7)

              ; Store the address of the string in A1.
              movea.l a0,a1

              ; Find the next operator ('*' or '/') or the null character
              ; (meaning the character that follows the number),
              ; and store its address in A2.
              jsr    NextOp2
              movea.l a0,a2

              ; Store the operator or the null character in D1.
              move.b (a2),d1

              ; Replace the operator by the null character.
              clr.b (a2)

              ; Convert the number
              ; (A0 must hold the memory location of the number).
              movea.l a1,a0
              jsr    Convert

              ; If no error occurs,
              ; D0 holds the integer value of the number.
              ; We can return true (no error).
              beq    \true

\false       ; Return false (error).
              ; D0 has not been modified.
              ; A0 points to the string.
              ; We just have to restore the operator held in D1.
              move.b d1,(a2)

              ; And return Z = 0.
              andi.b #%11111011,ccr
              bra    \quit

\true        ; Return true (no error).
              ; First, restore the operator held in D1.
              move.b d1,(a2)

              ; Then, store the address that follows the number in A0.
              movea.l a2,a0

              ; Finally, return Z = 1.
              ori.b  #%00000100,ccr

\quit        ; Restore registers from the stack and return from subroutine.
              movem.l (a7)+,d1/a1-a2
              rts

```

```

GetNum2      ; Save registers on the stack.
             movem.l d1/a1-a2,-(a7)

             ; Store the address of the string in A1.
             movea.l a0,a1

             ; Find the next operator ('+' or '-') or the null character
             ; (meaning the character that follows the term),
             ; and store its address in A2.
             jsr     NextOp1
             movea.l a0,a2

             ; Store the operator or the null character in D1.
             move.b (a2),d1

             ; Replace the operator by the null character.
             clr.b (a2)

             ; Convert the term
             ; (A0 must hold the memory location of the term).
             movea.l a1,a0
             jsr     GetExpr1

             ; If no error occurs,
             ; D0 holds the integer value of the number.
             ; We can return true (no error).
             beq     \true

\false       ; Return false (error).
             ; D0 has not been modified.
             ; A0 points to the string.
             ; We just have to restore the operator held in D1.
             move.b d1,(a2)

             ; And return Z = 0.
             andi.b #%11111011,ccr
             bra     \quit

\true        ; Return true (no error).
             ; First, restore the operator held in D1.
             move.b d1,(a2)

             ; Then, store the address that follows the number in A0.
             movea.l a2,a0

             ; Finally, return Z = 1.
             ori.b  #%00000100,ccr

\quit        ; Restore registers from the stack and return from subroutine.
             movem.l (a7)+,d1/a1-a2
             rts

```

Split **GetExpr** into two subroutines: **GetExpr1** and **GetExpr2**.

```

GetExpr1      ; Save registers on the stack.
               movem.l d1-d2/a0,-(a7)

               ; Convert the first number of the expression (result -> D0).
               ; If error, return false.
               jsr      GetNum1
               bne      \false

               ; The first number is stored in D1.
               ; (D1 is used to contain the result of the successive operations.)
               move.l   d0,d1

\loop          ; The operator or the null character is stored in D2.
               ; If it is the null character, return true (no error).
               move.b   (a0)+,d2
               beq       \true

               ; Convert the next number (result -> D0).
               ; If error, return false.
               jsr      GetNum1
               bne      \false

               ; Determine the operation to perform (*, /).
               cmp.b    #'/',d2
               beq       \divide

\multiply      ; Perform the operation and branch to loop.
               muls.w   d0,d1
               bra       \loop

\divide        ; If the divisor is null (division by 0), return false (error).
               tst.w    d0
               beq       \false

               ; The quotient is 16 bits wide.
               ; Perform a sign extend operation to increase the length to 32 bits.
               divs.w   d0,d1
               ext.l    d1
               bra       \loop

\false         ; Return Z = 0 (error).
               andi.b   #%11111011,ccr
               bra       \quit

\true          ; Return Z = 1 (no error).
               ; (Copy the final result into D0.)
               move.l   d1,d0
               ori.b     #%00000100,ccr

\quit          ; Restore registers from the stack and return from subroutine.
               movem.l (a7)+,d1-d2/a0
               rts

```

```

GetExpr2      ; Save registers on the stack.
               movem.l d1-d2/a0,-(a7)

               ; Convert the first term of the expression (result -> D0).
               ; If error, return false.
               jsr      GetNum2
               bne      \false

               ; The first term is stored in D1.
               ; (D1 is used to contain the result of the successive operations.)
               move.l   d0,d1

\loop         ; The operator or the null character is stored in D2.
               ; If it is the null character, return true (no error).
               move.b   (a0)+,d2
               beq       \true

               ; Convert the next term (result -> D0).
               ; If error, return false.
               jsr      GetNum2
               bne      \false

               ; Determine the operation to perform (+, -).
               cmp.b    #'-',d2
               beq       \subtract

\add          ; Perform the operation and branch to loop.
               add.l    d0,d1
               bra       \loop

\subtract     sub.l     d0,d1
               bra       \loop

\false        ; Return Z = 0 (error).
               andi.b   #%11111011,ccr
               bra       \quit

\true         ; Return Z = 1 (no error).
               ; (Copy the final result into D0.)
               move.l   d1,d0
               ori.b    #%00000100,ccr

\quit         ; Restore registers from the stack and return from subroutine.
               movem.l (a7)+,d1-d2/a0
               rts

```

Finally, modify the main program.

```

Main      ; Display the following message: "Enter an expression:"
          ; (The message is displayed in the top left-hand corner.)
          movea.l #sInput,a0
          clr.b   d1
          clr.b   d2
          jsr     Print

          ; Get the user expression.
          ; (The string is stored in the memory location sBuffer.)
          ; (It is displayed two lines below the previous message.)
          movea.l #sBuffer,a0
          addq.b  #2,d2
          move.l  #60000,d3
          move.l  #8000,d4
          jsr     GetInput

          ; Remove spaces.
          jsr     RemoveSpace

          ; Display the following message (two lines below): "Result:"
          movea.l #sResult,a0
          addq.b  #2,d2
          jsr     Print

          ; Increment the line number by 2.
          addq.b  #2,d2

          ; Calculate the result (result -> D0.L).
          ; If error, branch to \error.
          movea.l #sBuffer,a0
          jsr     GetExpr2
          bne     \error

\noError  ; No error occurs.
          ; Convert the integer result into a string.
          ; The string is stored in the address sBuffer (A0 = sBuffer).
          jsr     Itoa

          ; Display the result and exit.
          jsr     Print
          bra     \quit

\error    ; An error occurs.
          ; Display an error message.
          movea.l #sError,a0
          jsr     Print

\quit     ; Breakpoint.
          illegal

```