

**LA TROBE UNIVERSITY**  
**DEPARTMENT OF ELECTRONIC ENGINEERING**  
**ELECTRONICS-2**

**Microprocessors: Lab 4 -**  
**REVERSE POLISH HEX CALCULATOR**

**Aims:**

Learn to design and write a application program in assembly language.

**Equipment:**

HC-COM, power supply, PC & Null Modem Cable, M68HC11 reference.

**Introduction:**

In many systems, a stack is used for the passing of parameters and return addresses. If the number of parameters does not match within a procedure, then a 'stack crash' occurs when the return address is invalid, and a return instruction is executed.

To get around this problem, a separate numeric stack can be used. This is how the Intel 80x87... floating point co-processor works. It has a 8-number deep floating point stack.

Floating point processor instructions all start with the letter F. For example **FILD** (**F**loating **P**oint **I**nteger **L**oad) loads an integer onto the floating point stack and converts to floating point. **FISTP** **F**loating point **I**nteger **S**tore and **P**op converts the top-of-stack to integer (performs rounding) and saves the number to an integer variable.

In this lab we will add functions to skeleton code of a calculator.

In our calculator we provide a stack, and subroutines:

**NLDD**        **N**umeric Stack **L**oad from **D** register  
**NSTPD**      **N**umeric Stack **S**Tore and **P**op to **D** register

**The Problem:**

1. You are required to write a 8/16 bit hexadecimal calculator program which you can use to test the 68HC11 native ALU instructions ADD, SUB, MUL and DIV.

You may build on the code from previous labs.

- (a) prompt the user for a command,
- (b) select the correct stack based operation,
- (c) print the stack after each operation,

Calculator commands are:

+ Add  
- Subtract  
/ Divide  
\* Multiply  
n Enter a number to the numeric stack  
p Pull (Pop) a number from the numeric stack

REVERSE POLISH CALCULATORS is a simple method where the numbers are pushed onto the stack, then the operations are performed on the stack, and the results are stored on the stack.

Example use of the calculator

n 1234

n 4321

+

Will push 1234 onto the stack, then 4321 onto the stack, finally the + command will pop 2 values from the stack, add them together and push them back onto the stack.

Hint: use atoi and outHexD from last weeks labs.

Procedures:

Answer all questions, perform all the exercises and prepare your report in your lab book as you proceed. You may refer to lecture notes, hand-outs and other references.

During this lab, use **Hyperterminal** for communications to HCCOM. Configure Hyperterminal for **9600** baud, **8** data bits, **no parity**, **1 stop bit** and **Flow Control : none**.

You may use notepad to write your program.  
Save the file as LAB4.ASM.

Use the AS11 assembler to assemble the code to a S19 file.  
**AS11 LAB4.ASM -L > LAB4.LST**

Then enter **LOAD 2000** on HCCOM, and send the file using hyperterminal's **Transfer->send text file (select files of type "All Files"** and find your program LAB4.S19)

Once the file is successfully transferred, you can run the program using **G 2000** (if it starts at 2000).

The problem is Divided into functional blocks:

- (1) Print a prompt message(s) to the terminal,
- (2) read the command as a single character
- (3) Gather responses from the user,
- (4) Select the appropriate subroutine,
- (5) execute the subroutine
- (6) On error, stop immediately and display a suitable error message.

Incorporate the following skeleton code and subroutines and into your programs:  
\* filename: skeleton.asm

```
        ORG    $2000
        JMP    START

invCmdStr fcc 'Invalid command'
        fcb    13, 10, 0
MAXCHAR EQU 16
OutChar equ $FFAF
OUTLHF  equ $FFB2
OUTRHF  equ $ffb5
CRLF    equ $FFC4
InChar  equ $FFCD

hello   FCC    'Reverse Polish Calculator'
        FCB    0
tempString rmb    16
        fcb    0
stkUnder FCC    'Numeric Stack underflow'
        FCB    0
stkOver  fcc    'Numeric stack overflow'
        Fcb    0
nPrompt FCC    'Enter Hex number to push:'
        FCB    0

* SPACE FOR NUMERIC STACK
numEnd   rmb    100
numStart rmb    2
stkPtr   rmb    2
param1   rmb    2

START:
        ldx    #hello
        jsr    outStringX
        ldy    #numStart
        sty    stkPtr

* Loop forever
cmdLoop:
        jsr    showStackTop

* Here we read a 1 character command
        jsr    InChar

* then we execute the selected command
* this is like a CASE statement:
        cmpa   #'n'
        beq   getNumber
        cmpa   #'N'
        beq   getNumber
        cmpa   #'+'
        beq   stkAdd
        cmpa   #'-'
        beq   stkSub
```

```

        cmpa    #'/'
        beq     stkDiv
        cmpa    #'*'
        beq     stkMul
        cmpa    #'p'
        beq     pop
        cmpa    #'P'
        beq     pop
OTHERWISE:
        ldx     #invCmdStr
        jsr     outStringX
* End CASE statement
        bra     cmdLoop
*
getNumber:
        jsr     getHex
        bra     cmdLoop
*
stkAdd:
        jsr     add
        bra     cmdLoop
*
stkSub:
        jsr     sub
        bra     cmdLoop
*
stkDiv:
        jsr     div
        bra     cmdLoop
*
stkMul:
        jsr     mul
        bra     cmdLoop
*
pop:
        jsr     drop
        bra     cmdLoop
*
add:
        jsr     NSTPD
        STD     param1
        jsr     NSTPD
        ADDD    param1
        jsr     NLDD
        rts

```

```

*****
* Add your code to the subroutines below
*****

```

```

sub:
    rts

```

```

div:
    rts

```

```

mul:
    rts

```

```

*****
***** Add your code above here *****
*****

```

```

drop:
        jsr     NSTPD
        jsr     outHexD
        rts
*

```

```
* -----  
* Load D to the Numeric stack  
* -----
```

```
NLDD:  
*      Numeric Stack Load from D register  
      pshy  
      ldy   stkPtr  
      cmpy  #numEnd  
      bls   stackOver  
      stab 0,y  
      dey  
      staa 0,y  
      dey  
      sty   stkPtr  
      puly  
      rts
```

```
stackOver:  
      ldx   #stkOver  
      jsr   outStringX  
      swi
```

```
showStackTop:  
      psha  
      pshb  
      pshy  
  
      jsr   NSTPD  
      jsr   NLDD  
      jsr   outHexD  
  
      puly  
      pulb  
      pula  
      rts
```

```
* -----  
* Numeric stack store and pop D  
* -----
```

```
NSTPD:  
*      Numeric Stack STore and Pop to D register  
      pshy  
      ldy   stkPtr  
      cmpy  #numStart  
      bhi   stackUnder  
      iny  
      ldaa 0,y  
      iny  
      ldab 0,y  
  
      sty   stkPtr  
      puly  
      rts
```

```
stackUnder  
      ldx   #stkUnder  
      jsr   outStringX  
      swi
```

```

* -----
* Read a hex number & push onto stack
* -----
getHex:
    ldx    #nPrompt
    jsr    outStringX

    ldx    #tempString
    ldab   #MAXCHAR
    jsr    inStringXB

    jsr    AToIXD
    psha
    pshb
    jsr    outHexD
    pulb
    pula
    jsr    NLDD

    rts

* -----
* Console Input inStringXB
* -----
* Reads an ASCIIZ string pointed to by Index register X.
* Maximum length to read is specified by Accumulator B.
* Pressing carriage return also terminates input, placing a 0 at the end of
the string.
* -----
* USAGE:
*   inputBuffer fcc "00010"
*       FCB 0
* SAMPLE CODE to read 5 characters into the string inputBuffer:
*   LDAB #5
*   LDX #inputBuffer
* Get a string of at most B characters, into buffer pointed to by X
*   jsr inStringXB
*   LDX #inputBuffer
*   jsr atoiXD          * convert as hex number
* -----
inStringXB:
    pshx          * Save pointer to string
    pshb          * B = maximum # chars to read
    psha          * Save Acc A
getAgain:
    pshb          * Preserve B
    jsr    InChar * get character into Acc. A
    pulb          * Recover B
*
    cmpa    #13   * is is Carriage Return (CR)?
    beq    foundEnter * CR character? - yes: break out of loop
    staa   0,x    * no: save read char in buffer
    inx    * point to next buffer position
    decb   * decrement the limit counter
    bne    getAgain * max count not = 0, loop again
foundEnter:
    clr    0,x    * ensure string is terminated with a <null> character
    pula   * recover Acc A
    pulb   * recover Acc B
    pulx   * recover pointer to string
    rts

```

```

* -----
* SUBROUTINE ATOI - ASCII to INT16
* Pass a null-terminated string to this routine in X, and it will
* return the converted 16 bit value in the D register.
*
* Upon entry
* Index Register X points to the beginning of the string to convert
*
* Upon return:
* Accumulator D holds the converted 16 bit Integer
*
* Side effects:
* Upon return X points to the string terminator
* Garbage in - Garbage Out
* -----
*
AToIXD:
    PSHY * SAVE Y REGISTER
    LDD #0 * Start with 0 as our interim result in accumulator D
MoreChar:
    XGDY * Swap D with Y; Y now contains the interim result
    LDAB 0,X * Get the character pointed to by the X register
    CMPB #0 * Compare with the String Terminator (Null)
    BEQ AllConverted * Finished conversion
*
* now convert the ASCII letters/numbers -> binary
    SUBB #$30 * '0' - $30 => 0, ... '9'-$30 -> 9, 'A'-$30->17
    CMPB #9 * is result in range 0..9?
    BLE IsNumeric * If it is 0..9 then Character is converted
    SUBB #7 * else it was letter & convert letters A..F to binary
IsNumeric:
    ANDB #$0F * Mask any bits in top nybble of byte
    XGDY * Swap the interim result back into D
    ASLD * Multiply our interim result by 16, by shifting 4 times
    ASLD
    ASLD
    ASLD
    XGDY * Swap D with Y; Y now contains the interim result
    ABY * IY = IY + B = append the binary value to our interim result

    XGDY * Swap the interim result back into D
*
    INX * X = X + 1 - point to the next character in the string
    BRA MoreChar * loop for more characters to convert
*
AllConverted:
    XGDY * Return result in D register
    PULY * RESTORE Y REGISTER
    RTS * Return from subroutine

*
* -----
* Console Output - an ASCIIZ string pointed to by Index register X.
* -----
outStringX:
    pshx
again:
    ldaa    0,x          * get the character pointed to by X
    cmpa    #0          * Compare with null - character 0
    beq     foundNull   * found null character so break out of loop
    jsr     OutChar     * output character in Acc. A
    inx
    bra     again

foundNull:
    pulx

```

```

        rts

* -----
* Output 2 hex digit version of Accumulator A
* -----
outHexA:
    psha
    pshb
    pshx
    pshy

    TAB
    JSR    OUTLHF ; output high nybble
    TBA    ; restore it to A
    JSR    OUTFHF ; output low nybble

    puly
    pulx
    pulb
    pula
    rts

* -----
* Output 4 hex digit version of Accumulator D
* -----
outHexD:
    psha
    pshb

    pshb
    jsr    outHexA
    pula
    jsr    outHexA

    JSR    CRLF ; output carriage return - line feed

    pulb
    pula
    rts

*
* -----
* This subroutine delays (a little over) 100 micro-seconds
* -----
* Busy-wait a number of clock cycles
* -----
delay100us:
    psha ; save Accumulator A
*      LDAA #40 ; Generate a "short" delay > 100 microsec
seconds/clock ; 40 loops for 200 clock cycles at 0.5 micro
DELAYLOOP:
    DECA ;
    BNE  DELAYLOOP
    pula ; restore Accumulator A
    RTS ;

```