CpE 390 Microprocessor Systems

Lecture 5

Advanced Assembly Programming (2)

Character and Word Counting

- find the character count and word count of a string.

-A string is terminated by the NULL character.

-A new word is identified by skipping over the white space characters.

-When a new word is identified, it must be scanned through before the next word can be identified.

- we will include space, carriage return, line feed, and tab characters in the character count

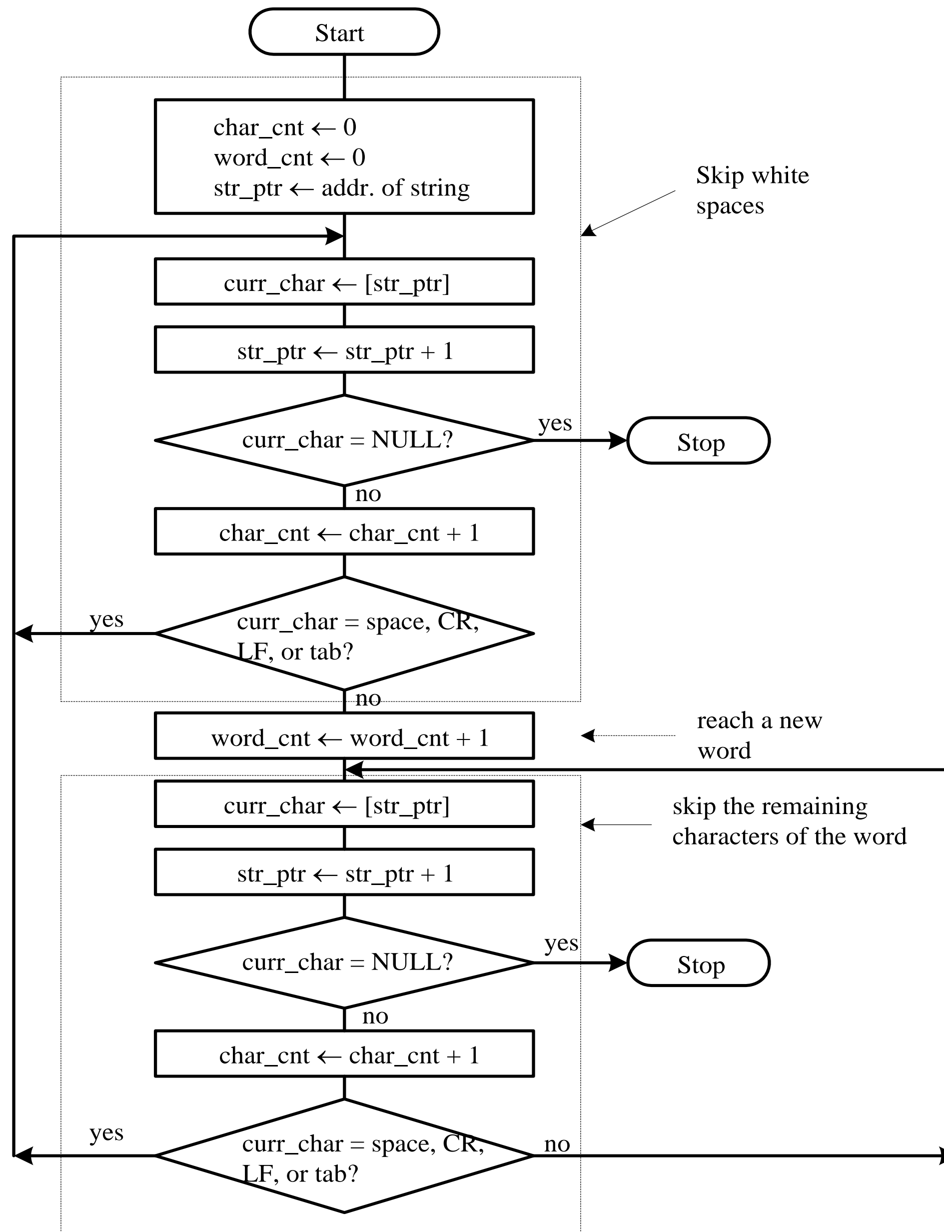


Figure 4.4 Logic flow for counting characters and words in a string

Example 4.7 Write a program to count the number of characters and words contained in a given string.

Solution:

```
tab         equ     $09
sp          equ     $20
cr          equ     $0D
lf          equ     $0A
            org     $800
char_cnt    rmb     1
word_cnt    rmb     1
string_x    fcc     "this is a strange test string to count chars and words."
            fcb     0
            org     $1000
            ldx     #string_x
            clr     char_cnt
            clr     word_cnt
string_lp   ldab    1,x+        ; get one character and move string pointer
            lbeq    done        ; is this the end of the string?
            inc     char_cnt
; the following 8 instructions skip white space characters between words
            cmpb    #sp
            beq     string_lp
            cmpb    #tab
```

```
                beq         string_lp
                cmpb        #cr
                beq         string_lp
                cmpb        #lf
                beq         string_lp
; a non-white character is the start of a new word
                inc         word_cnt
wd_loop         ldab        1,x+        ; get one character and move pointer
                beq         done
                inc         char_cnt
; the following 8 instructions check the end of a word
                cmpb        #sp
                lbeq        string_lp
                cmpb        #tab
                lbeq        string_lp
                cmpb        #cr
                lbeq        string_lp
                cmpb        #lf
                lbeq        string_lp
                bra         wd_loop
done            swi
                end
```

String Insertion

- The pointers to the string and the substring to be inserted are given.
  The insertion point is given.
  The procedure is given in Figure 4.6.

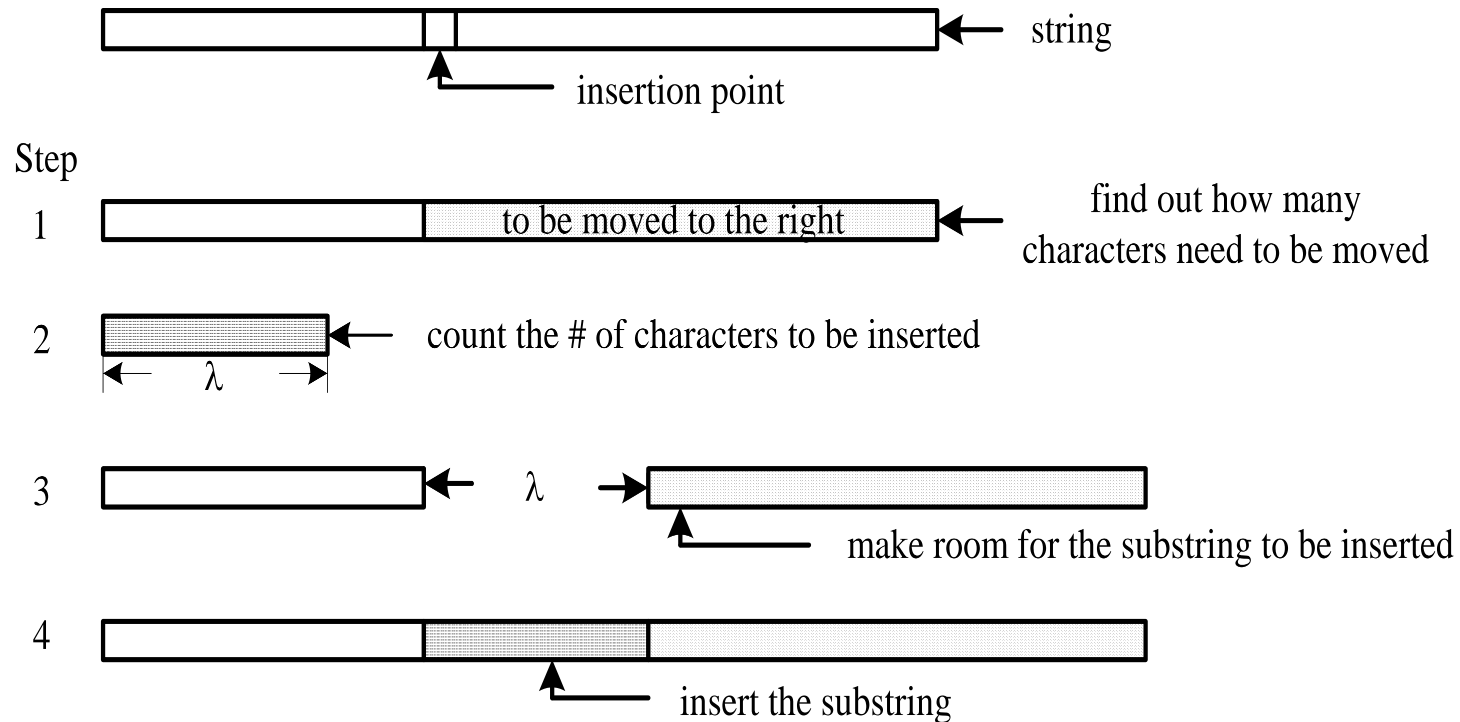


Figure 4.6 Major steps of substring insertion

Example 4.9 Write a program to implement the string insertion algorithm.

```
            org     $800
ch_moved    rmb     1
char_cnt    rmb     1
sub_strg    fcc     "the first and most famous "
            fcb     0
string_x    fcc     "Yellowstone is national park."
            fcb     0
offset      equ     15
ins_pos     equ     string_x+offset   ; insertion point

            org     $1000
; the next 7 instructions count the number of characters to be moved
            ldaa    #1
            staa    ch_moved
            ldx     #ins_pos          ; use x to point to the insertion point
cnt_moved   ldaa    1,x+
            beq     cnt_chars         ; check if it is null character, if yes, go to cnt_chars
            inc     ch_moved          ; increase the cnt of characters to be moved
            bra     cnt_moved         ; loop continue
cnt_chars   dex                       ; subtract 1 from x so it points to the NULL character
            ldy     #sub_strg         ; use y as a pointer to the substring
            clr     char_cnt
; the following 3 instructions count the move distance
char_loop   ldab    1,y+
```

```
            beq     mov_loop    ; check if it is null character
            inc     char_cnt    ; increase the cont of characters to be inserted
            bra     char_loop
mov_loop    tfr     x,y         ; make a copy of x in y,  y point to NULL of string
            ldab    char_cnt
            aby                 ; (b)+(y)->(y), y point to (NULL of string + char_cnt to be inserted)
            ldab    ch_moved    ; place the number of characters to be moved in B
again       movb    1,x-,1,y-   ; move characters of the string from last one till insertion point
            dbne    b,again     ; make room for insertion
            ldx     #ins_pos    ; set up pointers to prepare insertion
            ldy     #sub_strg   ;           "
            ldab    char_cnt
insert_lp   movb    1,y+,1,x+   ; insert substring to string from insertion point
            dbne    b,insert_lp
            swi
            end
```

Structured Programming

- One fundamental issue when developing a software program, especially in assembly language, is to maintain a consistent structure, so that your program will be
  - Easy to debug
  - Easy to verify
  - Easy to maintain
- Structured program brings quality of software through modular programming
  - Main program contains the logical structures of the algorithm when subroutine executes many of the details
- Flowcharts provides convenient mechanisms to visualize conditional branching and function calls

Program Structure

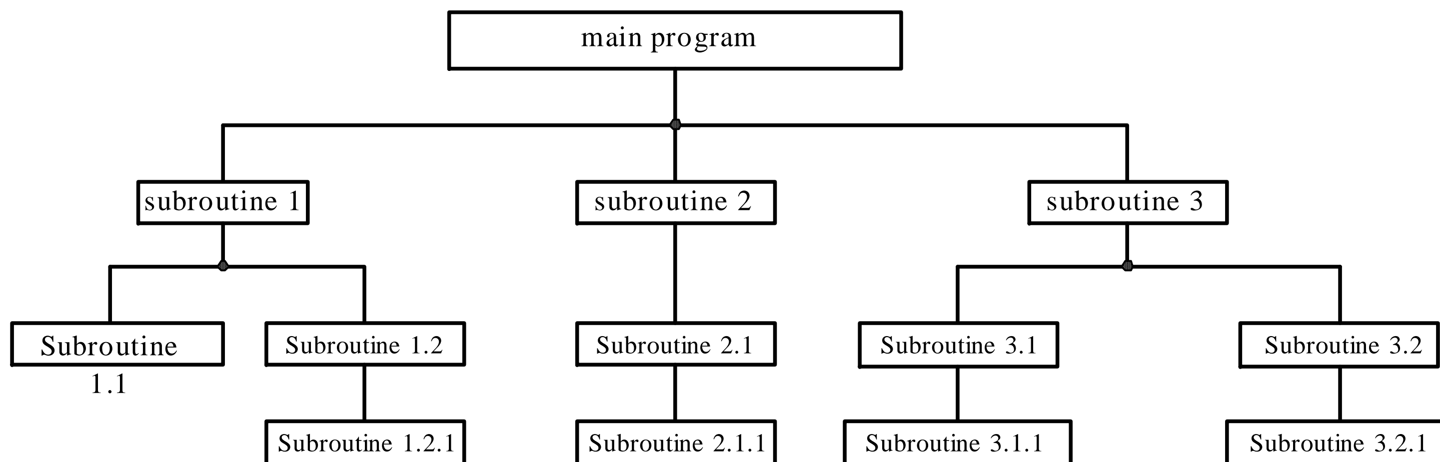


Figure 4.7 A structured program

Subroutine Processing

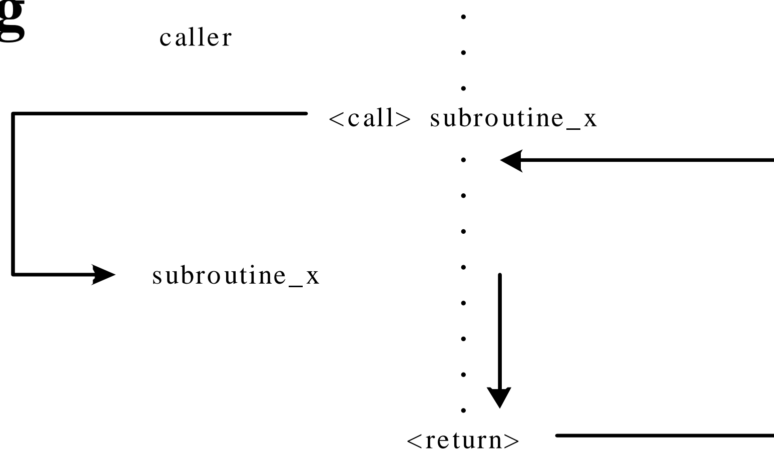


Figure4.8 Program flow during a subroutine call

Subroutines

- A sequence of instructions that can be called from various places in the program
- Allows the same operation to be performed with different parameters
- Simplifies the design of a complex program by using the divide-and-conquer approach

Instructions Related to Subroutine Calls

```
[<label>]   BSR   <rel>    [<comment>]  ; branch to subroutine
[<label>]   JSR   <opr>    [<comment>]  ; jump to subroutine
[<label>]   RTS            [<comment>]  ; return from subroutine
[<label>]   CALL  <opr>    [<comment>]  ; to be used in expanded memory (16-bit addr mode)
            RTC                         ; return from CALL
```

where

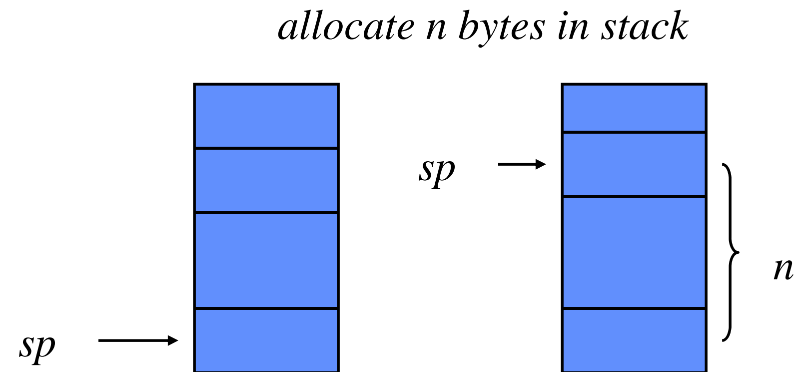
<rel> is the offset to the subroutine
<opr> is the address of the subroutine and is specified in the DIR, EXT, or INDEXED addressing mode.

Subroutines

- CALL <opr> ; call a subroutine
  - Expanded memory (larger than 64KB) supported by some 68HC12 members
  - Treat 16KB memory space from $8000-$BFFF as a program memory window
  - An 8-bit program page register (PPAGE) is added to select one of the 256 16KB program memory pages to be accessed
  - The call function pushes the current value of PPAGE register along with the return address onto the stack and then transfer the program control to the subroutine.
  - The <opr> filed in the call instruction specifies the page number and the starting address of the subroutine within that page

- RTC ; return from call
  - PPAGE and the return address are restored from the stack
  - Program execution continues at the restored address.

Issues in Subroutine Calls



1. ***Parameter passing***
   - Use registers
   - Use the stack
   - Use global memory
2. ***Returning results***
   - Use registers
   - Use the stack (caller created a hole in which the result will be placed)
   - Use global memory
3. ***Local variable allocation***
   - Allocated by the callee
   - The following instruction is the most efficient method of local variable allocation.

```
leas  -n,sp        ; allocate n bytes in the stack for local variables
```

4. ***Local variable deallocation***
   - performed by the subroutine
   - The following instruction is the most efficient method of local variable deallocation.

```
leas  n,sp         ; deallocate n bytes from the stack
```

Stack Frame

- The region in the stack that holds incoming parameters, the subroutine return address, local variables, and saved registers is referred to as stack frame.
- The stack frame is also called **activation record**.

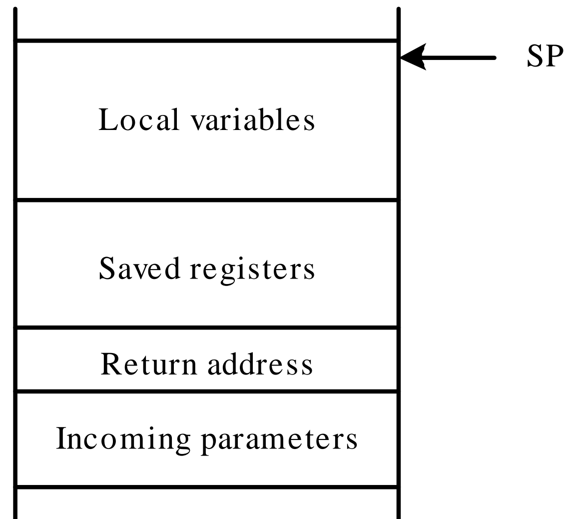


Figure 4.9 Structure of the 68HC12 stack frame

- *When BSR, ISR, or CALL is executed, the return address is automatically pushed on stack*
- *When RTS or RTC is executed, the return address is pulled from stack*
- *Local variables and saved registers have to be released before RTS or RTC*

Example 4.10 Draw the stack frame for the following program segment after the leas –10,sp instruction is executed:

```
        ldd     #$1234
        pshd
        ldx     #$4000
        pshx
        jsr     sub_xyz
        …
sub_xyz pshd
        pshx
        pshy
        leas    -10,sp
        …
```

Solution:

The stack frame is shown in Figure 4.10.

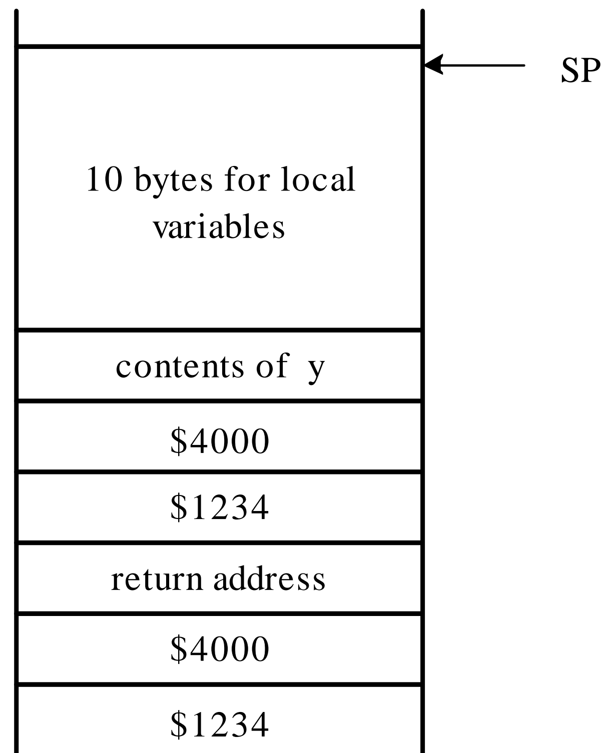


Figure 4.10 Stack frame of example 4.10

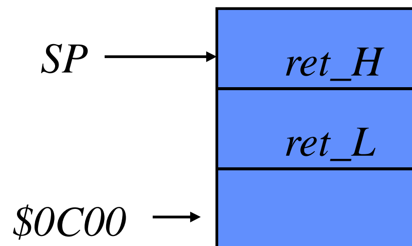
Examples of Subroutines

- Example: From Lab2, move data from memory buffer DATA1 to BUF1 and DATA2 to BUF2. Use subroutine to implement data transfer.

  - *Use register to pass parameters*

```
; main
ORG    $1000
LDS    #$0C00
LDX    #DATA1
LDY    #BUF1
LDAA   #8
JSR    Transfer
LDX    #DATA2
LDY    #BUF2
LDAA   #8
JSR    Transfer
```

```
Transfer   CMPA  #0
           BEQ   exit
next       MOVW  2, X+, 2, Y+
           DBNE  A, next
exit       RTS
```



Incoming parameter: none
Saved register: none
Local parameters: none

Using stack to pass parameter of counter value

```
        ORG $1000
        LDS  #$0C00
        LDX  #DATA1
        LDY  #BUF1
        LDAA  #8
        PSHA          ; push counter into stack
        JSR  Transfer
        PULA

Transfer    TST  2, SP      ;test counter
            BEQ  exit       ; if zero, exit
next        MOVW   2, X+,2,Y+
            DEC  2, SP      ; decrease counter
            BNE  next
exit        RTS
```

Use memory to pass parameter of counter value

```
mem    EQU  $8000

       LDX   #DATA1
       LDY   #BUF1
       MOVB  #8, mem ; save counter in mem
       JSR   Transfer

Transfer   TST   mem  ;   test counter
           BEQ  exit
next       MOVW  2, X+, 2, Y+
           DEC  mem;   decrease counter
           BNE next
exit       RTS
```

Examples of Subroutines

4.9.1. Finding the Greatest Common Divisor (GCD) of integers m and n

Algorithm

Step 1

```
If m = n then
    gcd ← m;
    return;
```

Step 2

```
if n < m then swap m and n.
```

Step 3

```
gcd ← 1
if m = 1 then return.
```

Step 4

```
For i = 2 to m do
    if (m % i = 0 and n % i = 0)
            gcd ← i;
```

Example 4.11 Write a subroutine to compute the greatest common divisor of two 16-bit unsigned integers.

Solution:

The two 16-bit incoming parameters are passed in index register X and double accumulator D. The stack frame of this subroutine is shown in Figure 4.11.

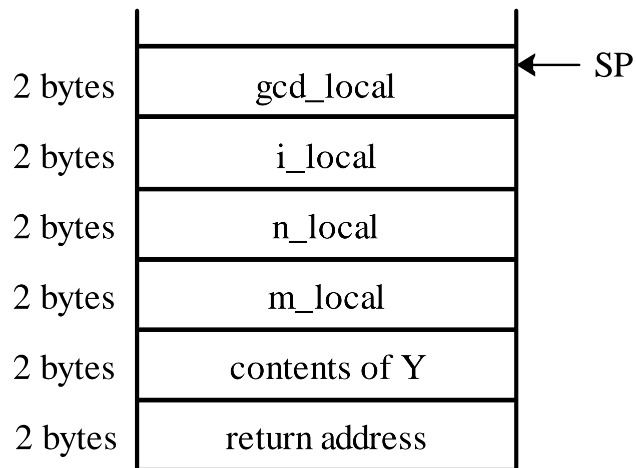


Figure 4.11 Stack frame of example 4.11

```
gcd_local  equ   0              ; the distance of variable gcd_local from the stack top
i_local    equ   2              ; the distance of variable i_local from the stack top
n_local    equ   4              ; the distance of variable m_local from the stack top
m_local    equ   6              ; the distance of variable n_local from the stack top
local_var  equ   8              ; number of bytes to be allocated to local variables
           org   $800
m          dw    375            ; the first operand
n          dw    1250           ; the second operand
gcd        rmw   1

           org   $1000
           ldx   m
           ldd   n
           jsr   find_gcd       ; jump to subroutine find_gcd
           std   gcd            ; save the gcd in memory
           swi
find_gcd   pshy
           leas  -local_var,sp  ; allocate space for local variables (load effective addr to sp)
           stx   n_local,sp     ; n -> local n
           std   m_local,sp     ; m -> local m
           ldy   #1
           sty   gcd_local,sp   ; initialize gcd to 1
           cpd   n_local,sp     ; compare m with n
```

```
         beq     m_equ_n       ; gcd = m if m = n
         blo     m_less_n      ; it is fine if m < n
         exg     d,x           ; swap m and n
         std     n_local,sp    ; also make sure the stack frame copy of
         stx     m_local,sp    ; m and n are swapped
m_less_n cpd     #1
         beq     done          ; if m = 1, then gcd = 1
         ldx     #2
         stx     i_local,sp    ; initialize i to 2
loop     ldx     i_local,sp
         cpx     m_local,sp
         bhi     done
         ldd     m_local,sp
         idiv                  ; divide m by i (d)/(x)=(x), remainder ->d
         cpd     #0
         bne     next_i
         ldd     n_local,sp
         ldx     i_local,sp
         idiv                  ; divide n by i
         cpd     #0
         bne     next_i
         ldd     i_local,sp
```

```
            std     gcd_local,sp  ; set i as the current gcd
next_i      ldx     i_local,sp
            inx
            stx     i_local,sp    ; increment i
            jmp     loop
m_equ_n     ldd     m_local,sp    ;  m -> d
            bra     exit
done        ldd     gcd_local,sp  ;  1 -> d
exit        leas    local_var,sp  ; deallocate local variables
            puly
            rts
            end
```

Don’t get up

- Lecture 6 now

Homework #4

- See course website: http://390.revan.us
  - click homework tab
- Please submit a hard copy