CpE 390 Microprocessor Systems

Lecture 4

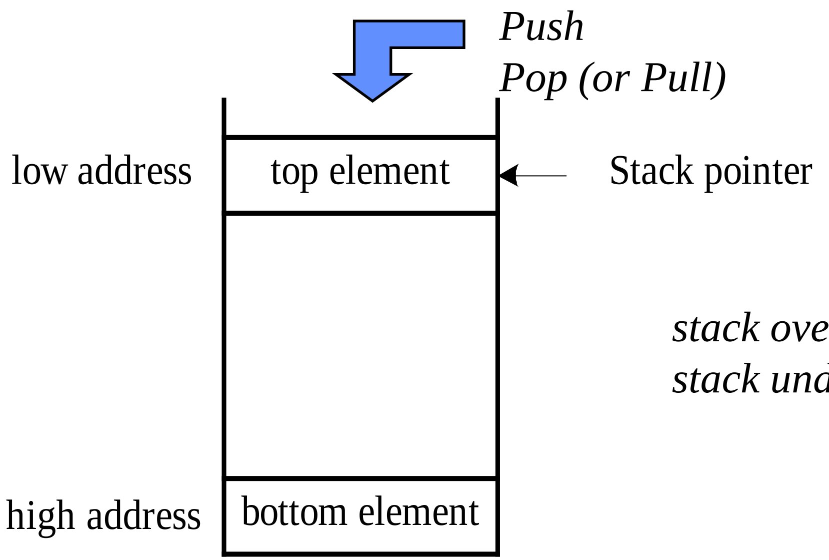
Advanced Assembly Programming (1)

Introduction

- ***Program = data structures + algorithm***
- Data structures to be discussed
  1. ***stacks***: a first-in-last-out data structure
  2. ***arrays***: a set of elements of the same type
  3. ***strings***: a sequence of characters terminated by a special character

Stack

A reserved RAM area in main memory where program perform push or pull operations



stack overflow: push too much
stack underflow: pull too much

Figure 4.1 Diagram of the 68HC12 stack

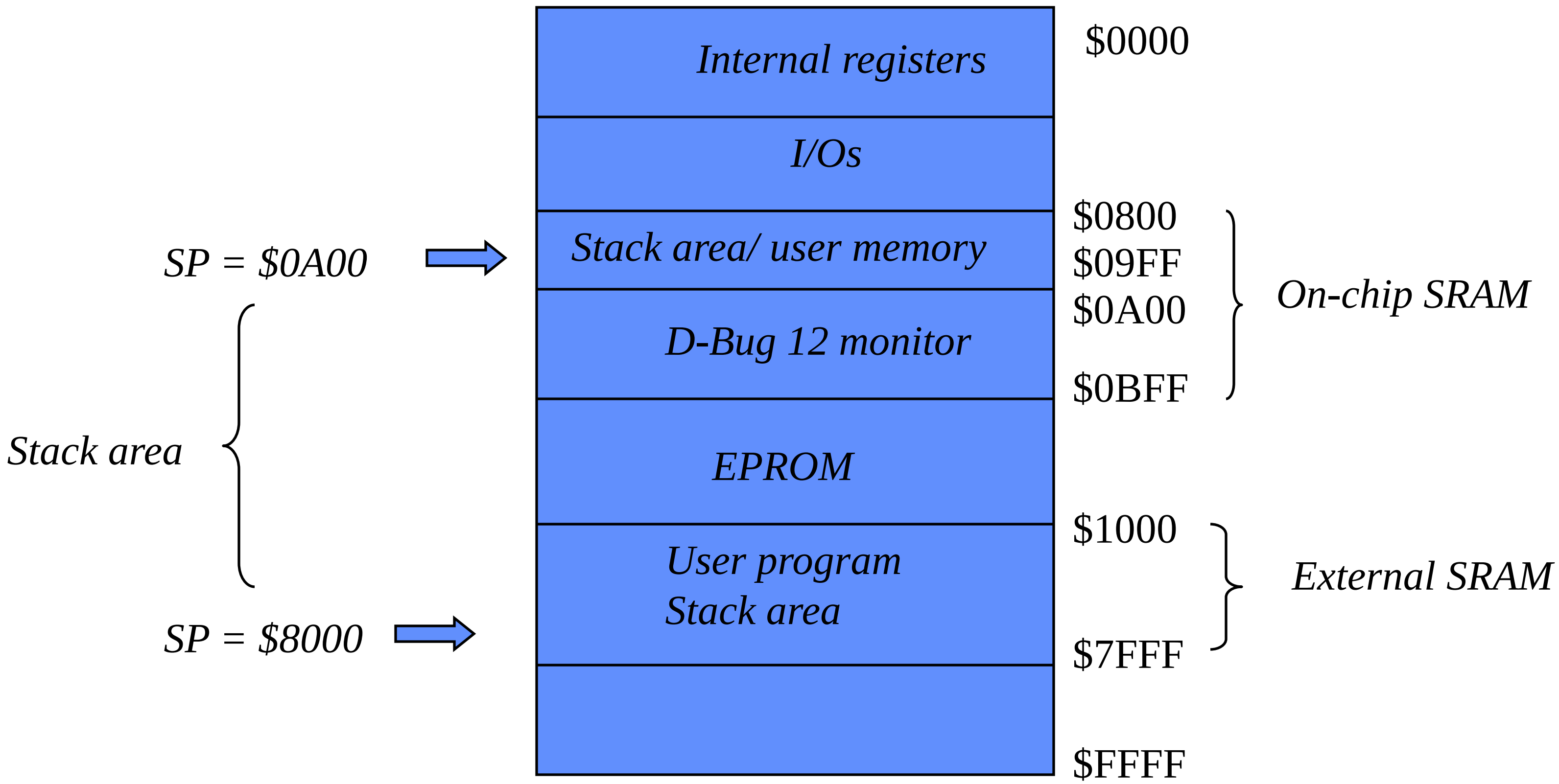
68HC12 Support for the Stack Data Structure

- A 16-bit stack pointer (SP)
- Instructions and addressing mode

Table 4.1 68HC12 push and pull instructions and their equivalent load and store instructions

Mnemonic	Function	Equivalent instruction
psha	push A into the stack	staa 1, -SP
pshb	push B into the stack	stab 1, -SP
pshc	push CCR into the stack	none
pshd	push D into stack	std 2, -SP
pshx	push X into the stack	stx 2, -SP
pshy	push Y into the stack	sty 2, -SP
pula	pull A from the stack	ldaa 1, SP+
pulb	pull B from the stack	ldab 1, SP+
pulc	pull CCR from the stack	none
puld	pull D from the stack	ldd 2, SP+
pulx	pull X from the stack	ldx 2, SP+
puly	pull Y from the stack	ldy 2, SP+

68HC12 Memory Map



68HC12 Support for the Stack Data Structure

- Usage
  - Saving the return address for a subroutine call
  - A good place to hold local variables
  - Context switch
    - When multiple tasks exit, the operating system would switch from one task to another based on different schedule algorithm
- Reasons
  - Dynamic allocation/release allows for reuse of memory
  - Limited scope of access provide data protection
  - Only the program that created the local variables can access it
  - The # of variables is limited only be the size of the stack allocation, more than registers

Indexable Data Structures

- Vectors and matrices are indexable data structures.
- The first element of a vector is associated with the index 0 in order to facilitate the address calculation.
- Assemblers directives db, dc.b, fdb are used to define arrays of 8-bit elements.
- Assemblers directives dw, dc.w, and fdb are used to define arrays of 16-bit elements.

Arrays

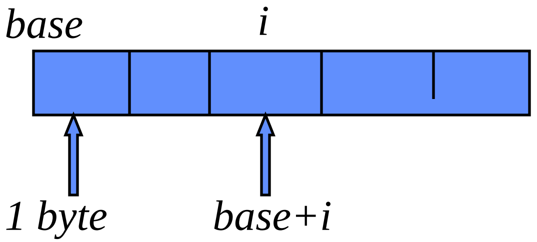
- Are indexable data structure, usually zero-origin indexing
- Are made up of elements of same types and precision
- One-dimentional=>vector, Two-dimentional => matrics
- Vectors
  - Define a vector

    In C: construct char data[4] = {0x05, 0x06,0x0A, 0x09};

    In Assembly:

    ```
    data    fcb(/db/dc.b)  $05, $06, $0A, $09   ; byte array
    data1   fdb(/fw/dc.w)  $0506, $0A09         ; word array
    ```
  - Access the ith element of a byte array

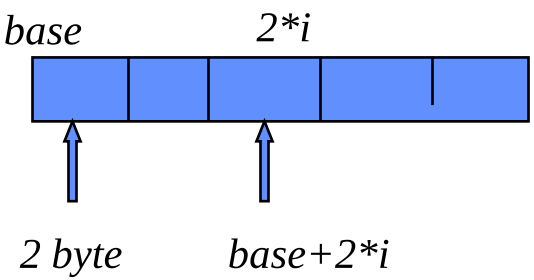
    ```
    LDAB #i     ; load index i to B
    LDX #data   ; load base address
    LDAA B,X    ; read the ith element
    ```



Arrays

- Vectors
  - Access the ith element of a word array

 LDAB #i   ; load index i to B
 LSLB   ; 2*i
 LDX #data   ; load base to X
 LDD B, X   ; read the ith element



  - Length of array

static: saves the length of the array as the 1st element

e.g.   data   fcb 4, $05, $06, $0A, $09

data   dw 5, 1, 10, 100, 1000, 10000

dynamic: using a termination code

NUL $00   ; ASCII code 0

EOT $04   ; ASCII code 4, end of transmission

EXT $03   ; ASCII code 3, end of text

e.g. data   fcb $05, $06, $0A, $09, 0, 0, 0, 0

Arrays

Example: Read one byte at a time start at memory location labeled with data, and move data bytes to location labeled with port B. Start over when reach a null character.

```
PORTB      EQU $0001
           ORG $1000
startover  LDX #DATA      ; load data address to X
next       LDAA 0, X      ; load data[i] to A
           BEQ startover  ; if a null character, start over
           STAA PORTB     ; otherwise, store data[i] to PORTB
           INX            ; increment index
           BRA next       ; go to next data address
           END
```

Example 4.2 Write a program to find out if the array **vec_x** contains a value **key**. The array has 16-bit elements and is not sorted.

Solution:

- Use the double accumulator to hold the key
- Use the index register X as a pointer to the array
- Use the index register Y to hold the loop count
- Need to compare **key** with every array element because it is not sorted.

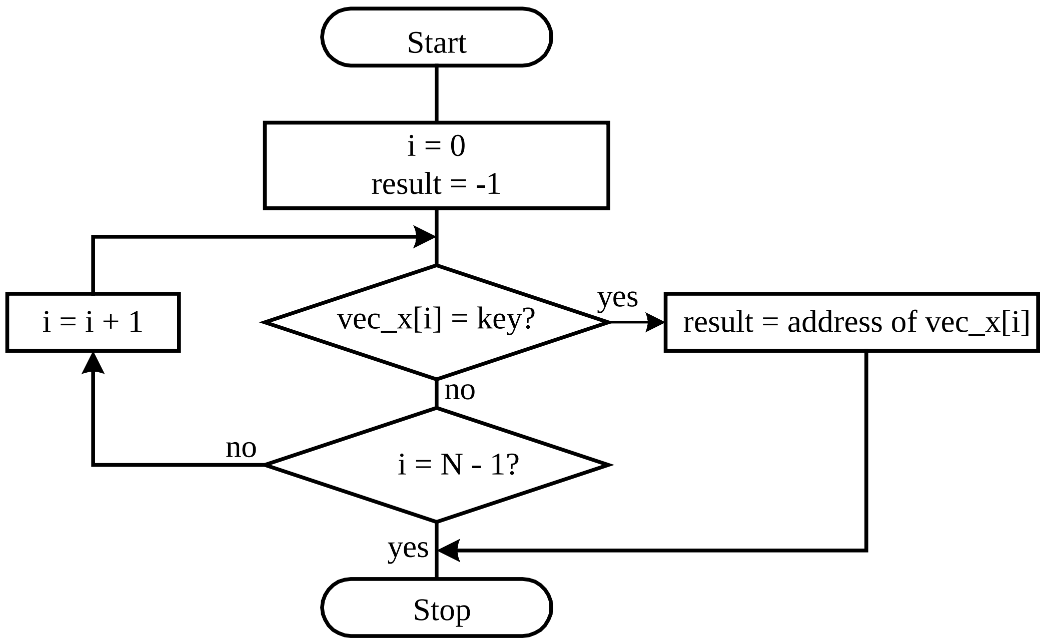


Figure 4.3 Flowchart for sequential search

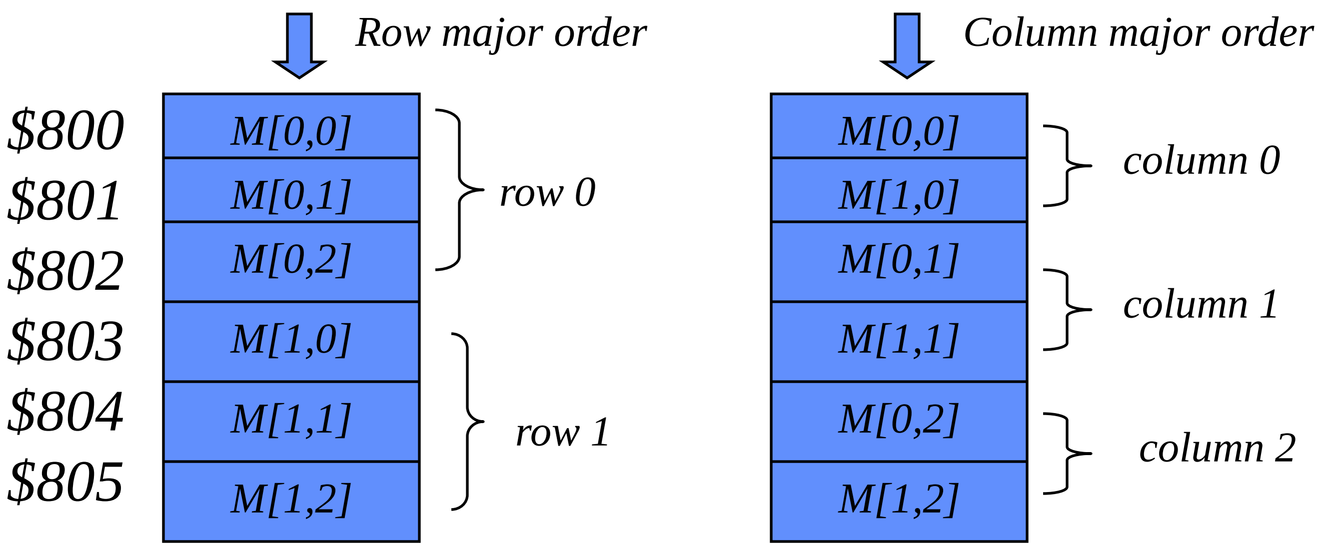
```
N         equ    30          ; array count
notfound  equ    -1
key       equ    190         ; define the searching key
          org    $800
result    rmw    1           ; reserve a word for result

          org    $1000       ; starting point of the program
          ldy    #N          ; set up loop count
          ldd    #notfound
          std    result      ; initialize the search result
          ldd    #key
          ldx    #vec_x      ; place the starting address of vec_x in X
loop      cpd    2,X+        ; compare the key with array element & move pointer
          beq    found
          dbne   Y,loop      ; have we gone through the whole array?
          bra    done
found     dex                ; need to restore the value of X to point to the
          dex                ; matched element
          stx    result
done      swi
vec_x     dw     13,15,320,980,42,86,130,319,430,4, 90,20,18,55,30,51,37
          dw     89,550,39, 78,1023,897,930,880,810,650,710,300,190
          end
```

Matrices

- Define a matrix: map the two-dimensional data structure into the linear address space of memory

Example: a byte matrix with two rows and three columns: unsigned char M[2][3]:

	Column 0	*1*	*2*
Row i = 0	*M[0,0]*	*M[0,1]*	*M[0,2]*
1	*M[1,0]*	*M[1,1]*	*M[1,2]*



Matrices

- To define in row major order

  MAT1 DB M[0,0],M[0,1],M[0,2],M[1,0]M[1,1],M[1,2]
- To define in column major order

  MAT2 DB M[0,0],M[1,0],M[0,1],M[1,1],M[0,2],M[1,2]
- Access the (*i*, *j*) element in row/column order matrix MAT1/MAT2

$$\text{address of } MAT1[i, j] = \text{base of } MAT1[0, 0] + i * \underbrace{n}_{\#\text{ of columns}} + j,$$

$$\text{address of } MAT2[i, j] = \text{base of } MAT2[0,0] + j * \underbrace{m}_{\#\text{ of rows}} + i$$

*Example: The memory address of MAT1 [1,2] = $800 + 1 * 3 + 2 = $805*

*The memory address of MAT2 [0,1] = $800 + 1*2 + 0 = $802*

Matrices

Example: Access a 2 x 3 row major order matrix M

m = 2, n=3,   address of M[i, j]=based address of *M[0,0]+3*i+j*

```
ORG    $1000
LDX     #base
LDAA   #i
LAAB   #3
MUL                      ; 3 * i
ADDB  #j                 ; 3 * i + j
LDAA B, X                ; read M[i, j]
```

Binary Search

Step 1

Initialize variables max and min to n -1 and 0, respectively.

Step 2

If max < min, then stop. No element matches the key.

Step 3

Let mean = (max + min)/2

Step 4

If key = arr[mean], then key is found in the array, exit.

Step 5

If key < arr[mean], then set max to mean - 1 and go to step 2.

Step 6

If key > arr[mean], then set min to mean + 1 and go to step 2.

Example 4.3 Write a program to implement the binary search algorithm and also a sequence of instructions to test it.

Solution:

```
n       equ     30          ; array count
key     equ     69          ; key to be searched
        org     $800
max     rmb     1           ; maximum index value for comparison
min     rmb     1           ; minimum index value for comparison
mean    rmb     1           ; the average of max and min
result  rmb     1           ; search result
        org     $1000
        clra
        staa    min         ; initialize min to 0
        staa    result      ; initialize result to 0
        ldaa    #n-1
        staa    max         ; initialize max to n-1
        ldx     #arr        ; use X as the pointer to the array
loop    ldab    min
        cmpb    max
        lbhi    notfound    ; if min>max, then not found (unsigned comparison)
        addb    max         ; compute mean
        lsrb                ; logic shift B to right to compute (min+max)/2
```

```
          stab    mean            ; save mean
          ldaa    b,x             ; get a copy of the element arr[mean]
          cmpa    #key
          beq     found
          bhi     search_lo
          ldaa    mean
          inca
          staa    min             ; place mean+1 in min to continue
          bra     loop
search_lo ldaa    mean
          deca
          staa    max
          bra     loop
found     ldaa    #1
          staa    result
notfound  swi
arr       db      1,3,6,9,11,20,30,45,48,60
          db      61,63,64,65,67,69,72,74,76,79
          db      80,83,85,88,90,110,113,114,120,123
          end
```

Strings

- Is a data structure with equal size elements that allows only sequential access (in contrast, an array allows random access to any element in order)
- The bytes of the string are always read in order from the first to last
- Contains a sequential characters terminated by a NULL (ASCII code 0) or EOT (ASCII code 4).
- In C, ASCII strings are stored with NULL. When a string is defined, the C compiler automatically adds NULL (zero) at the end.

Example: const char str[ ] = “Hello, world”

H	*E*	*L*	*L*	*O*	*,*	*W*	*O*	*R*	*L*	*D*	*\0*

→ *NULL*

- In assembly, the zero must be explicitly defined

```
example:   str1   fcc   “Hello, world”
                  fcb   0
```

Strings

- A number in the computer is represented as a binary number.
- A number must be converted to ASCII code before output so that it can be understood.
- To convert a binary into an ASCII string, we divide the binary number by 10 repeatedly until the quotient is zero. For each remainder, the number $30 is added.

Example 4.4 Write a program to convert the unsigned 8-bit binary number in accumulator A into BCD digits terminated by a NULL character. Each digit is represented in ASCII code.

Solution:

1. 4 bytes are needed to hold the converted BCD digits including the NULL.
2. Repeated division by 10 method is used.

```
test_dat   equ    220
           org    $800
out_buf    rmb    4            ; reserve 4 bytes for results
temp       rmb    2            ; reserve 2 bytes for temp values
           org    $1000        ; starting address of the program
           ldaa   #test_dat    ; load the test data to A
           ldy    #out_buf     ; load the address of result to Y
           tab                 ; transfer the 8-bit value in  B
; check to see if the number has only one digit
           cmpb   #9
           bhi    chk_99       ; if greater than 9, go chk_99
           addb   #$30         ; otherwise, one digit, convert the digit into ASCII code
           stab   0,y          ; save the code and increment the pointer
           clr    1,y          ; terminated the string with NULL
           jmp    done
chk_99     clra
; check to see if the number has two digits
           cmpb   #99          ; is the number greater than 99
           bhi    three_dig    ; if yes, the string has three digits
           ldx    #10
           idiv                ; (D) / (X) -> X, remainder ->D
           addb   #$30         ; convert the lower digit
           stab   1,y          ; store the lowest digit
```

```
            xgdx                    ; swap the quotient to D, (D) <-> X
            addb        #$30
            stab        0,y          ; save the upper digit
            clr         2,y          ; terminated the string with NULL
            bra         done
three_dig   ldx         #10
            idiv
            addb        #$30
            stab        2,y          ; save the least significant digit
            xgdx                     ; swap the quotient to D
            ldx         #10
            idiv
            addb        #$30
            stab        1,y          ; save the middle digit
            xgdx                     ; swap the hundred's digit to B
            addb        #$30
            stab        0,y          ; save the ASCII code of the highest digit
            clr         3,y          ; terminate the string with NULL
done        swi
            end
```

y *y+1* *y+2* *y+3*

Example 4.5 Write a program to convert the 16-bit signed integer in D into a string of BCD digits.

Solution:

1. A signed 16-bit integer is in the range of -32768 to +32767.
2. A NULL character is needed to terminate the converted BCD string.
3. A minus character is needed for a negative number.
4. Up to 7 bytes are needed to hold the converted result.

```
          org     $800
out_buf   rmb     7             ; reserve 7 bytes for BCD number
temp      rmb     2
test_dat  equ     -4300

          org     $1000
          ldd     #test_dat     ; load a test data
          ldy     #out_buf      ; use Y as the pointer to the output buffer
          cpd     #0
          lbpl    chk_9         ; long branch if plus (N=0). if plus, no complement
          lbeq    zero          ; is the given number a 0?
; the following three instructions compute the magnitude of the given number in D
          coma                  ; complement A
          comb                  ; complement B
          addd    #1            ; 2's complement number
          std     temp
          ldaa    #$2D          ; place a negative sum
          staa    0,y           ; store the negative sign
          iny
          ldd     temp          ; load the magnitude of data in D
chk_9     cpd     #9            ; does the number have only one digit?
          lbhi    chk_99
          addb    #$30
          stab    0,y
```

```
        clr     1,y         ; terminate the string with NULL
        lbra    done
chk_99  cpd     #99         ; does the number have only two digits?
        lbhi    chk_999     ; branch if the number has more than two digits
        ldx     #10
        idiv
        addb    #$30        ; convert the ones digit to BCD ASCII
        stab    1,y         ; save the ones digit
        xgdx                ; swap the quotient to D, (D) <-> X
        addb    #$30
        stab    0,y         ; save the tens digit
        clr     2,y         ; add a NULL character
        lbra    done
chk_999 cpd     #999        ; does the number have only three digits?
        lbhi    chk_9999    ; branch if the number has more than three digits
        ldx     #10
        idiv
        addb    #$30        ; convert the ones digit
        stab    2,y         ; save the ones digit
        xgdx                ; swap the quotient to D, (D) <-> X
        ldx     #10
        idiv
        addb    #$30
```

```
        stab    1,y             ; save the tens digit
        xgdx
        addb    #$30            ; convert the hundreds digit
        stab    0,y
        clr     3,y             ; add a NULL character
        lbra    done
chk_9999 cpd    #9999           ; does the number have only four digits?
        lbhi    five_digit      ; branch if the number has five digits
        ldx     #10
        idiv
        addb    #$30
        stab    3,y             ; save the ones digit
        xgdx
        ldx     #10
        idiv
        addb    #$30
        stab    2,y
        xgdx
        ldx     #10
        idiv
        addb    #$30
        stab    1,y
        xgdx
```

```
            addb    #$30
            stab    0,y
            clr     4,y          ; add a NULL character
            lbra    done
five_digit  ldx     #10
            idiv
            addb    #$30
            stab    4,y          ; save the ones digit
            xgdx                 ; swap the quotient to D, (D) <-> X
            ldx     #10
            idiv
            addb    #$30
            stab    3,y          ; save the tens digit
            xgdx                 ; swap the quotient to D, (D) <-> X
            ldx     #10
            idiv
            addb    #$30
            stab    2,y          ; save the hundreds digit
            xgdx                 ; swap the quotient to D, (D) <-> X
            ldx     #10
            idiv
            addb    #$30
```

```
            stab    2,y           ; save the hundreds digit
            xgdx
            ldx     #10
            idiv
            addb    #$30
            stab    1,y           ; save the thousands digit
            xgdx
            addb    #$30
            stab    0,y           ; save the ten-thousands digit
            clr     5,y           ; add a NULL character
done        swi
zero        ldaa    #$30
            staa    0,y
            clr     1,y
            swi
            end
```

Homework #3

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