Creating Power of 10 in 8-Byte Word

- If you are creating up to 10^{19}, you need to reserve $19 \times 8 = 152$ bytes of memory as a buffer and call it POW_10.
- Create a subroutine to multiply 8-byte number by 10 and call it (MUL_10).
- Start with putting 1 in an 8-byte number say ‘Res’.
- Call MUL_10 to multiply ‘Res’ by 10 and store the result at the *bottom* of POW_10 buffer.
- Adjust your pointer on the buffer for next result.
- Repeat the process till you generate all the necessary power of 10s.

Creating Power of 10 in 8-Byte Word …..

```
Pow-10  MOVW   #0,Res          ; set 8-byte number to 1.
        MOVW   #0,Res+2        ;
        MOVW   #0,Res+4        ;
        MOVW   #1,Res+6        ;
        LDY    #Pow10+150      ; set pointer at the bottomn
        MOVB   #19,count       ; set counter
Lpow    PSHY                   ; save pointer and multiply 8-byte
        LDX    #Res            ; number by 10.
        JSR    mul10           ;
        PULY                   ; recover the pointer
        MOVW   Res+6,2,Y-      ; store the result in Pow10
        MOVW   Res+4,2,Y-      ;
        MOVW   Res+2,2,Y-      ;
        MOVW   Res,2,Y-        ;
        DEC    count           ; if not done repeat
        BNE    Lpow            ;
        RTS

Res     RMB    8
temp    RMB    8
Pow10   RMB    152
count   RMB    1
```

Successive Subtraction Algorithm

1. Lets assume your 8-byte number is sitting in ‘Res’.
2. Create a temporary 8-byte variable called ‘temp’.
3. Create a 20-byte variable called ‘decnum’ as well as 1-byte counter.
4. Set pointers at the top of ‘POW_10’ and ‘decnum’.
5. Clear counter and copy ‘Res’ in ‘temp’.
6. Subtract ‘POW_10’ from ‘Res’, if result positive, increment counter, copy ‘Res’ in ‘temp’ and subtract again.
7. If result not positive, store counter in ‘decnum’, clear counter, copy ‘temp’ to ‘Res’, adjust both pointers and go back to step 6.
8. Continue till all digits are established.

How to Output 10-digit number

```
Out     LDY     #dnum
rep     LDAB    1,Y+
        BNE     prt
        CPY     #dnum+9
        BLO     rep
prt     CLRA
        ADDB    #$30
        PSHY
        LDX     putchar
        JSR     0,X
        PULY
        CPY     #dnum+9
        BHI     done
        LDAB    1,Y+
        JMP     prt
done    RTS
```

dnum	**0 0**
	0 0
	0 0
	0 8
	0 9
	0 0
	0 5
	0 6
	0 0
	0 0

8905600

Deque data Structure

A ***deque*** is a generalized data structure that includes two special cases: the ***stack*** and the ***queue***.

- A deque is a sequence of elements with two ends that we call them the top and the bottom.
- Either top or bottom element in the deque can be accessed.
- Ideally, deques are of infinite length, but practical deques have a maximum capacity.
- If this capacity is exceeded, we have an *overflow error.*
- A deque is implemented in assembly language by allocating an area of N bytes of *memory* as a *buffer* to it.

How to implement a Deque

There are various ways to configure a deque:

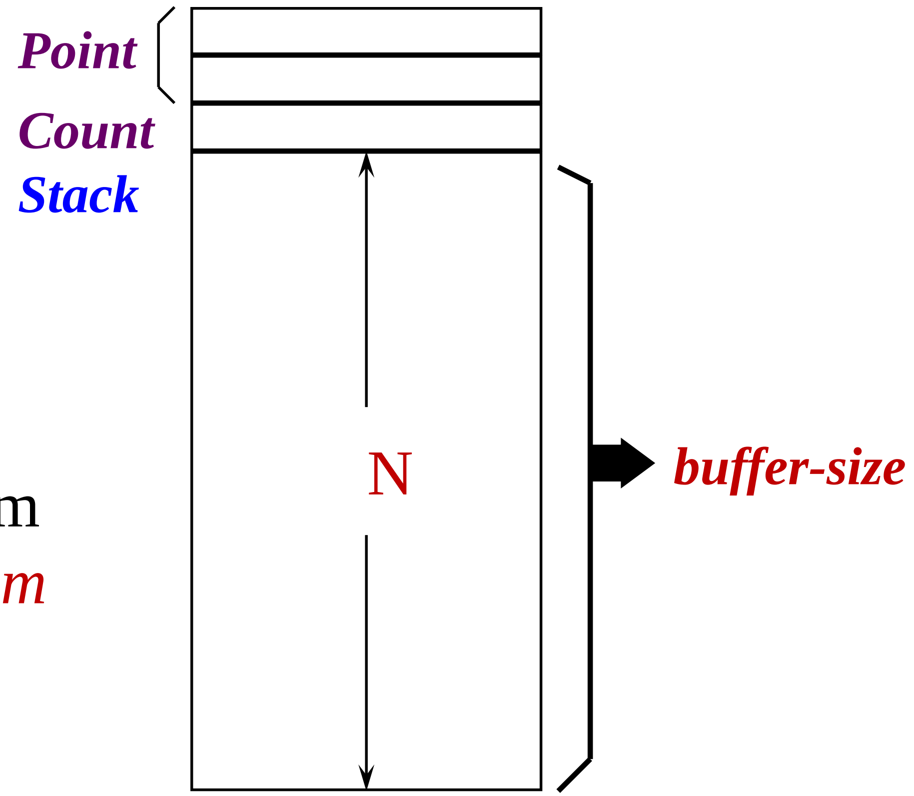
- First-In-First-Out system called (***FIFO***)
- Last-In-First-Out system called (***LIFO***)
- Use one pointer or Two pointers
- Start from bottom of the deque and go to top
- Start from top of the deque and go to bottom

Also each deque needs a *counter* to detect ***Overflow*** (exceeding the capacity) and ***underflow*** (pull more than you push).

Example: Write the necessary source code to create a LIFO (Last-In-First-Out) deque with one pointer.

Solution:

Address of first location in deque



Note! This deque starts from the *top* and fills to the *bottom*

This will be part of declaration

```
POINT   RMB   2
COUNT   DB    0
STACK   RMB   N
```

A segment of initialization at the beginning of main program

```
        LDX   #STACK
        STX   POINT
```

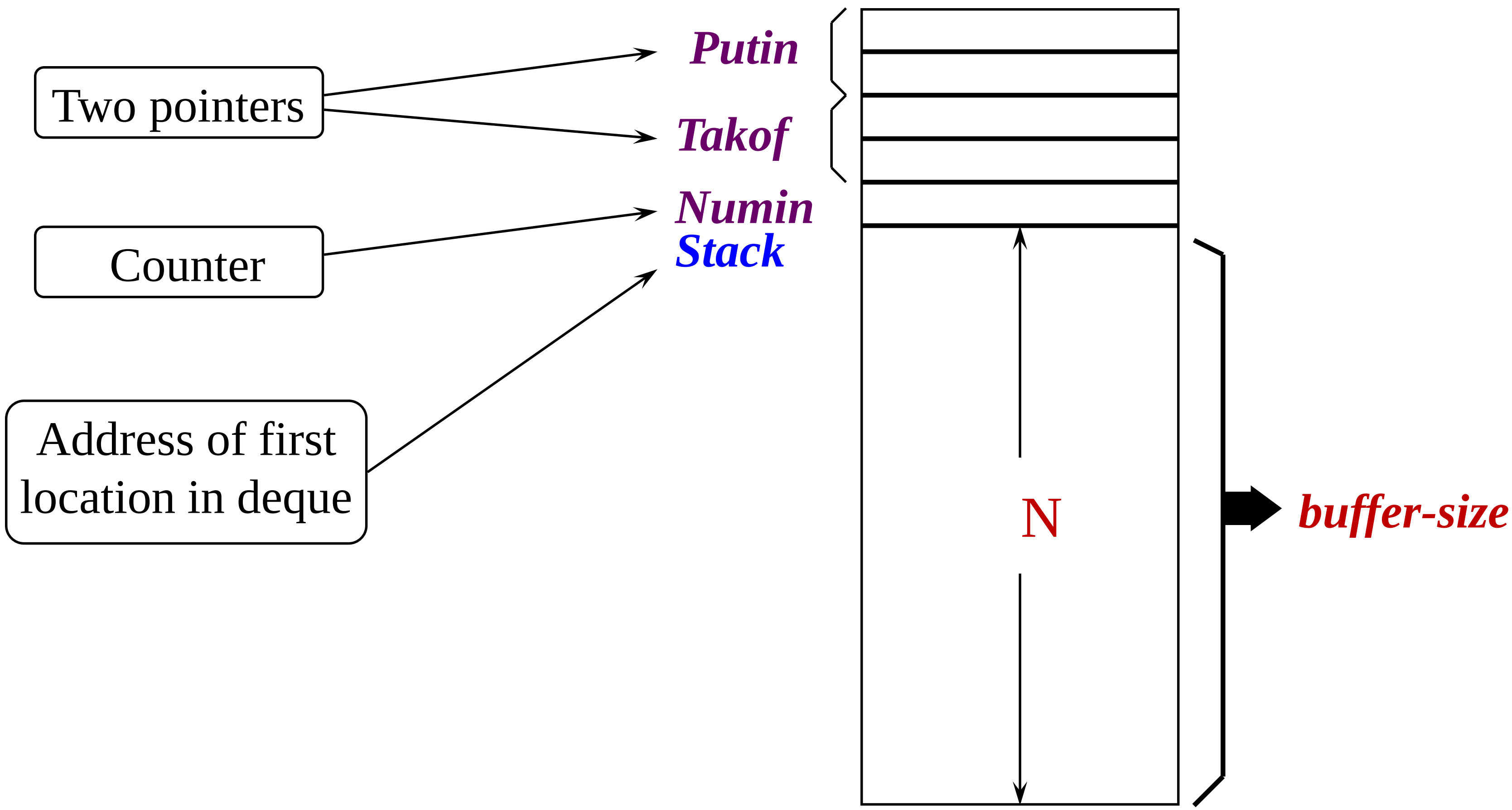
Two subroutines for Pushing and Popping

```
PUSH    LDAB  COUNT
        CMPB  #N
        BEQ   Err
        LDX   POINT
        STAA  1,X+
        STX   POINT
        INC   COUNT
        RTS
```

```
PULL    LDAB  COUNT
        CMPB  #0
        BEQ   Err
        LDX   POINT
        LDAA  1,-X
        STX   POINT
        DEC   COUNT
        RTS
```

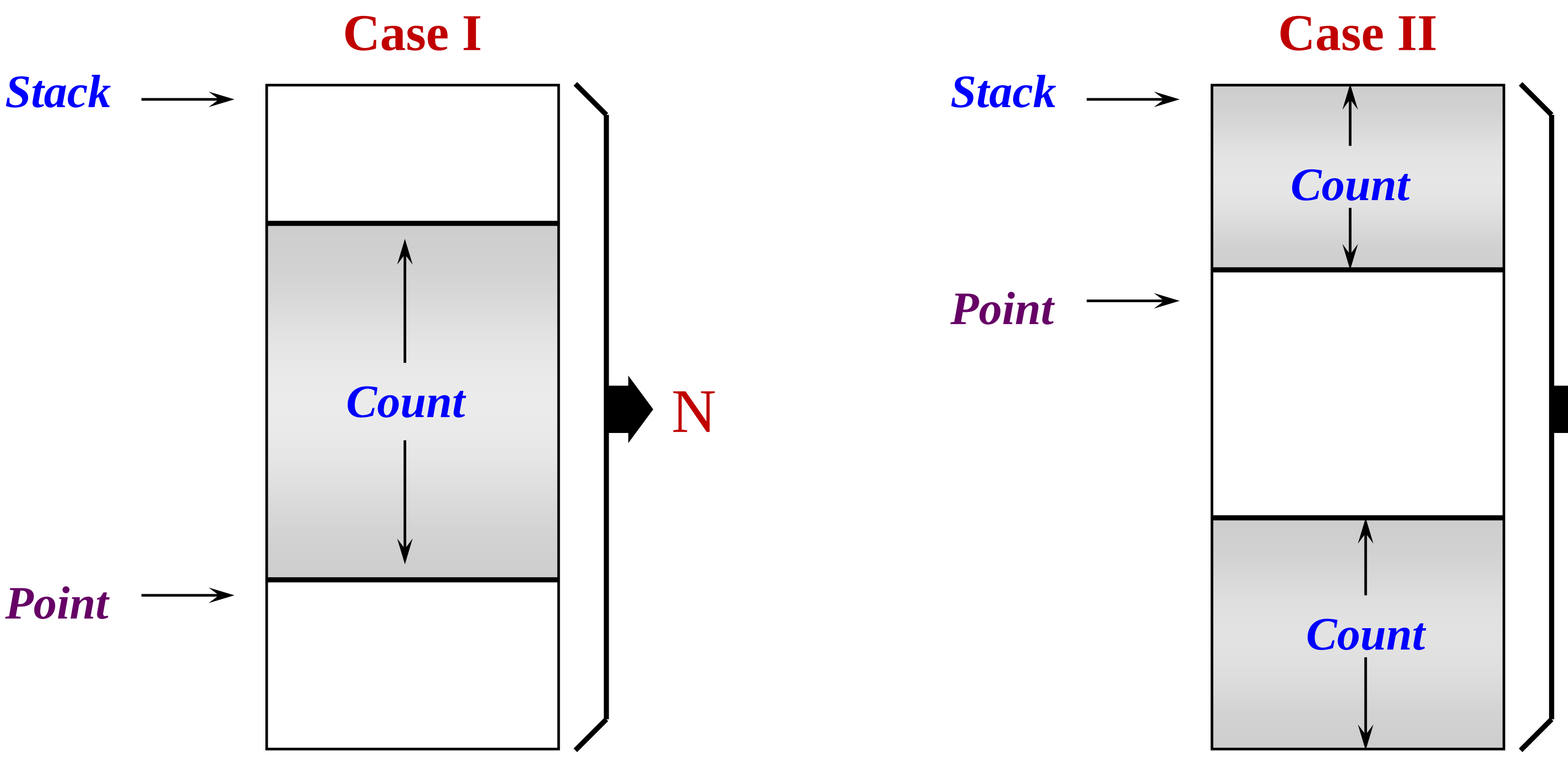
Example: Write the necessary source code to create a FIFO deque with two pointers.

Solution:



Example: Write the algorithm to create a FIFO deque using ONE pointer.

Solution: There are two cases to be considered.



Algorithm for Popping (Pulling)

Compare ***Count*** with zero
If *equal*, branch to ERROR

Compare [***point*** **-** ***Stack***] with ***Count***
If *greater* or *equal* branch to CASE I

CASE II   E.A. = ***Point*** + [$N - ***Count***]
Branch to CONT
CASE I   E.A. = ***Point*** **-** ***Count***
CONT   Load X with E.A.
Load Acc. A from 0,X
Decrement ***Count***
Return from subroutine

```
*************************************************************************************
*    Subroutine to pop from stack which uses one pointer and FIFO concept. The assumption is *
*    that buffer size is less than 256 and data byte will be popped into Accumulator A.      *
*************************************************************************************
POP     LDAA    COUNT       ;check if buffer is empty, then
        CMPA    #0          ;jump to error message.
        BEQ     ERR         ;

        LDD     POINT       ;find out which case are you dealing
        SUBD    #STACK      ;with and find effective address
        LDAA    COUNT       ;accordingly.
        CBA                 ;
        BHI     CASEII      ;

        LDD     POINT       ;case I, E.A. = point - count
        SUBB    COUNT       ;
        SBCA    #0          ;
        XGDX                ;
        BRA     CONT        ;

CASEII  LDAB    #N          ;case II, E.A. = point + [$N - count]
        SUBB    COUNT       ;
        LDX     POINT       ;
        ABX                 ;

CONT    LDAA    0,X         ;pop the data into acc. A
        DEC     COUNT       ;update count.
        RTS

ERR     LDD     #ERRMSG     ;let the user know that buffer is

        LDX     printf
        JSR     0,X         ;empty and return.
        RTS
```

Indexable Data Structures

Indexable structures include:

- Vectors
- Lists
- Arrays
- Tables

A ***Vector*** is a sequence of elements, where each element is associated with an index ***i*** that can be used to access it.

Example:

```
V    FCB    36,17,58,4,29
U    FDB    28,1028,876,425
```

The elements in these vectors can be accessed using indexed-addressing mode.

LDX #V LDAB #nth-element ABX LDAA 0,X	LDX #V LDAB #nth-element LDAA B,X
LDX #U LDAB #number_i ASLB ; multiply i by 2 ABX LDD 0,X	LDX #U LDAB #number_i ASLB ; multiply i by 2 LDAA B,X

A ***List*** is like a vector, being accessed by means of an index, but the elements of the list can be any combination of different precision words, characters, code words, and so on.

```
L       FDB     $F2A
        FCC     'Null'
        FCB     109
```

L	0F
	2A
	4E
	75
	6C
	6C
	6D

To get 109 in accumulator B:

```
LDAB        L+6
```

No space between L, +, and 6.

An ***Array*** is a vector whose elements are vectors of the same length. Normally, array is perceived as two-dimensional pattern.

24	18	36	99
56	42	75	10
83	17	66	13
92	87	5	58

If rows of the array are assumed as the elements of vector, data structure is called ***row major order array***

```
A1      FCB     11,12,13
        FCB     14,15,16
        FCB     17,18,19
        FCB     20,21,22
```

If the array is considered a vector of column vectors, the structure is a ***Column major order array***.

```
A2      FCB   11,14,17,20
        FCB   12,15,18,21
        FCB   13,16,19,22
```

Based on which order is used, an element from the i^{th} row and j^{th} column can be extracted from an array by a polynomial evaluation.

Example: address of $(i,j)^{th}$ element of a row major order array with N rows and M columns will be:

$$addressof(i, j) = (i \times M) + j + addressof(0,0)$$

Example: In a *major row order*, assume number of rows are 8 and number of columns are 10, i and j are in accumulators A and B respectively, and address of (0,0) element be A1. Write the instruction sequence to find the address of (i,j)th element.

```
PSHB            ; save it for after multiply
LDAB   #10      ; put M into acc. B
MUL             ; get i times M
ADDB   1,SP     ; add j
ADCA   #0       ; propagate carry
ADDD   #A1      ; add offset to initial address
XGDX            ; point X to A1(i,j)
LDAA   0,X      ; get A1(i,j) into acc. A
INS             ; balance the stack
```

i = 4, j = 6

N = 8 and M = 10

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$	$a_{0,6}$	$a_{0,7}$	$a_{0,8}$	$a_{0,9}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$	$a_{1,7}$	$a_{1,8}$	$a_{1,9}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,6}$	$a_{2,7}$	$a_{2,8}$	$a_{2,9}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,6}$	$a_{3,7}$	$a_{3,8}$	$a_{3,9}$
$a_{4,0}$	$a_{4,1}$	$a_{4,2}$	$a_{4,3}$	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	$a_{4,7}$	$a_{4,8}$	$a_{4,9}$
$a_{5,0}$	$a_{5,1}$	$a_{5,2}$	$a_{5,3}$	$a_{5,4}$	$a_{5,5}$	$a_{5,6}$	$a_{5,7}$	$a_{5,8}$	$a_{5,9}$
$a_{6,0}$	$a_{6,1}$	$a_{6,2}$	$a_{6,3}$	$a_{6,4}$	$a_{6,5}$	$a_{6,6}$	$a_{6,7}$	$a_{6,8}$	$a_{6,9}$
$a_{7,0}$	$a_{7,1}$	$a_{7,2}$	$a_{7,3}$	$a_{7,4}$	$a_{7,5}$	$a_{7,6}$	$a_{7,7}$	$a_{7,8}$	$a_{7,9}$

$$\begin{array}{c} 4 \times 10 \\ + \\ 6 \\ \hline i \times M + j \end{array}$$

Offset

$$\textit{Address of } a_{i,j} = \textit{Address of } a_{0,0} + (i \times M) + j$$

```
*****************************************************************
*                          JSR    INSERT                        *
*****************************************************************
* The subroutine puts A(i,j) which is 3-bytes long (triple precision) into   *
* locations N, N+1, and N+2. The size of array is 8×10 and stored in a row   *
* major order. Subroutine jumps to location ERROR if the indices are outside *
* the array's range.                                                          *
*                                                                             *
* Initial conditions on entering the subroutine:                             *
* A contains index i, 0 ≤ i < 8                                              *
* B contains index j, 0 ≤ j < 10                                             *
* X points to array A(i,j), specifically address of A(0,0)                   *
* Y points to location N                                                     *
*                                                                             *
* A(i,j) = A(0,0) + 3(10i+j)                                                 *
*        = A(0,0) + 30i + 3j  is used                                        *
*                                                                             *
* Registers A, B, and X are destroyed.                                       *
*****************************************************************
```

```
INSERT  CMPA  #8        ;check the range of index i
        BHS   err       ;use unsigned comparison to check for
        CMPB  #10       ;check for the range 0 <= j < 10
        BHS   err




Back    RTS
err     LDD   #Ermsg    ;print the error message on the screen
        LDX   printf
        JSR   0,X       ;and return from subroutine.
        BRA   Back      ;
Ermsg   FCC   'Indices out of range!'
        DB    $0D,$0A,0
```

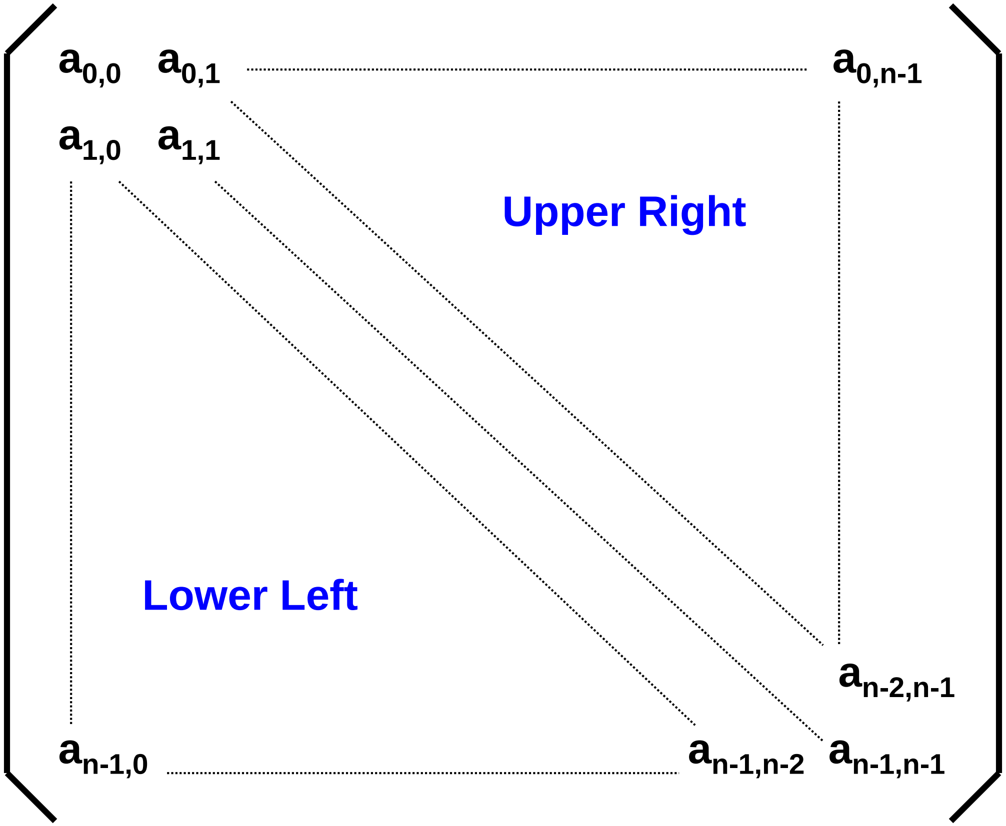
```
INSERT  CMPA   #8          ;check the range of index i
        BHS    err         ;use unsigned comparison to check for
        CMPB   #10         ;check for the range 0 <= j < 10
        BHS    err
        PSHB               ;save j
        LDAB   #30         ;load 30 to multiply it with i in acc. A
        MUL                ;acc. B contains 30i, since i < 8
        TBA                ;move 30i from B to A
        PULB               ;restore j in acc. B
        ABA                ;now acc. A has 30i+j
        ASLB               ;acc. B has 2j
        ABA                ;now acc. A has 30i+3j
        TAB                ;acc. B has 30i+3j
        ABX                ;point X to the location A(i,j)
        LDD    0,X         ;get first two bytes of A(i,j)
        STD    0,Y         ;store two bytes at N and N+1 locations
        LDAA   2,X         ;get third byte of A(i,j)
        STAA   2,Y         ;store the third byte at N+2 location
Back    RTS
err     LDD    #Ermsg      ;print the error message on the screen
        LDX    printf
        JSR    0,X         ;and return from subroutine.
        BRA    Back        ;
Ermsg   FCC    'Indices out of range!'
        DB     $0D,$0A,0
```

Example:

Write a program to transpose an $N \times N$ matrix.

Solution:

Note that matrix can be divided into ***upper right***, ***lower left***, and ***diagonal***.



```
            NAM     EXAMPLE
            ORG     $1000
N           EQU     8           ; dimension of the matrix
ILIMIT      EQU     N-2         ; upper limit of i
JLIMIT      EQU     N-1         ; upper limit of j
i           RMB     1           ; row index
j           RMB     1           ; column index
MAT         FCB     01,02,03,04,05,06,07,08
            FCB     09,10,11,12,13,14,15,16
            FCB     17,18,19,20,21,22,23,24
            FCB     25,26,27,28,29,30,31,32
            FCB     33,34,35,36,37,38,39,40
            FCB     41,42,43,44,45,46,47,48
            FCB     49,50,51,52,53,54,55,56
            FCB     57,58,59,60,61,62,63,64
            ORG     $1100
            CLR     i           ; initialize i to 0
LOOP1       LDAA    i           ;
            INCA                ;
            STAA    j           ; initialize j to i+1
* The following 7 instructions compute the address of element MAT(i,j) and leave it in X
LOOP2       LDAA    i           ; place the row index in A
            LDAB    #N          ; put N into B
            MUL                 ; multiply i x N
            ADDB    j           ; compute i x N + j
            ADCA    #0          ;
            ADDD    #MAT        ; compute MAT(0,0) + i x N + j
            XGDX                ; place the result in X
```

Continued in next slide …….

Continuation of the program :

```
* The following 7 instructions compute the address of element MAT(j,i) and leave it in Y

          LDAA     j          ; place the row index in A
          LDAB     #N         ; put N into B
          MUL                 ; multiply j x N
          ADDB     i          ; compute j x N + i
          ADCA     #0         ;
          ADDD     #MAT       ; compute MAT(0,0) + j x N + i
          XGDY                ; place the result in Y

* The following 4 instructions swap MAT(i,j) with MAT(j,i)

          LDAA     0,X
          LDAB     0,Y
          STAA     0,Y
          STAB     0,X

          LDAB     j
          INC      j
          CMPB     #JLIMIT    ; is j = N-1?
          BNE      LOOP2
          LDAA     i
          INC      i
          CMPA     #ILIMIT    ; is i = N-2?
          BNE      LOOP1
          SWI
          END
```

Sequential Data Structure

Sequential data is accessed by relative position. That is from pointer location the data before or after can be retrieved.

A ***string*** is an example of sequential data. After $\boldsymbol{i^{th}}$ element of sequence either $\boldsymbol{(i+1)^{th}}$ or $\boldsymbol{(i-1)^{th}}$ can be reached.

How to take care of end of data In sequential structure?

1- Keep track of length of data with counter.

2- Know the end address and compare it every time.

3- Put special character at the end of string.

4- For ASCII code, put 1 in the 7-bit of last character and check for it.

How to check for certain message to do certain task? For instance, check for user's input. If input is 'Happy' then go to routine which starts at address START.

```
            NAM     SEARCH
            ORG     $1100
user        RMB     2         ;holds initial address of string to
pass        FCC     'Happy'   ;be inserted by user

            ORG     $1200
SRCH        LDX     user      ;get starting address of user string
            LDY     #pass
            LDAA    #5
LOOP        LDAB    1,X+
            CMPB    1,Y+
            BNE     nogood
            DBNE    A,LOOP
            JMP     START
nogood      RMB     0
            END
```

Example: Write a program that asks user for password. Store the password and echo '*' for every character that user enters. Once carriage return is encountered, examine the password and allow or deny access accordingly.

```
        ORG     $1000
Again   LDD     #askpass        ;ask user to enter password
        LDX     printf          ;
        JSR     0,X             ;
        LDY     #usepass        ;set pointer to receive user's password
        PSHY                    ; save pointer
wait    LDX     getchar         ;receive password character by character
        JSR     0,X             ;output asterisk for each character and
        CMPB    #$0D            ;check for carriage return. Once you
        BEQ     check           ;encounter CR, go ahead and check the
        PULY                    ;password.
        STAB    1,Y+            ;
        PSHY                    ;
        CLRA                    ;
        LDAB    #'*'            ;
        LDX     putchar         ;
        JSR     0,X             ;
        BRA     wait            ;
check   LDX     #pass           ;start checking password. If there is an
        LDY     #usepass        ;error, display an error message and
```

```
loop    LDAA   1,X+         ;ask again. If password is correct, send
        CMPA   1,Y+         ;a greeting message and start the
        BNE    err          ; program.
        CPX    #endpass     ;
        BNE    loop         ;
        LDD    #greet       ;
        LDX    printf       ;
        JSR    0,X          ;
        JMP    start        ;
err     LDD    #errmsg      ;send error message and ask again
        LDX    printf       ;
        JSR    0,X          ;
        JMP    again        ;
start
           :
           :

pass     FCC   'What ever!'
endpass  RMB   1
usepass  RMB   50
askpass  FCC   'Enter Password:'
         DB    $0D,$0A,0
errmsg   FCC   'Wrong Password, Try again.'
         DB    $0D,$0A,0
greet    FCC   'Welcome to our humble game!'
         DB    $0D,$0A,0
```

Binary Search

- If an array (e.g. look up table) needs to be searched frequently, then *sequential search* is very inefficient.
- A better approach is to sort the array and use the *binary search* algorithm.
- Suppose that sorting array (in ascending order) has **n** elements and stored in memory locations starting at the label **arr**.
- Let *max* and *min* represent the highest and the lowest range of array indices to be searched and the variable *mean* represent the average of *max* and *min*.
- The idea of the binary search algorithm is to divide the stored array into three part:
  - The portion of the array with indices ranging from *mean*+1 to *max*.
  - The element with index equal to *mean* (middle element).
  - The portion of the array with indices ranging from *min* to *mean*-1.
- The binary search algorithm compares the key with the middle element and takes one of the following actions based on the comparison result:

Binary Search *continued …*

- If the key equals the middle element, then stop.

- If the key is larger than the middle element, then the key only can be found in the portion of the array with larger indices. The search will be continued on the upper half.

- If the key is smaller than the middle element, then the key only can be found in the portion of the array with smaller indices. The search will be continued on the lower half.

➢ the *binary search* algorithm can be formulated as follow:

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

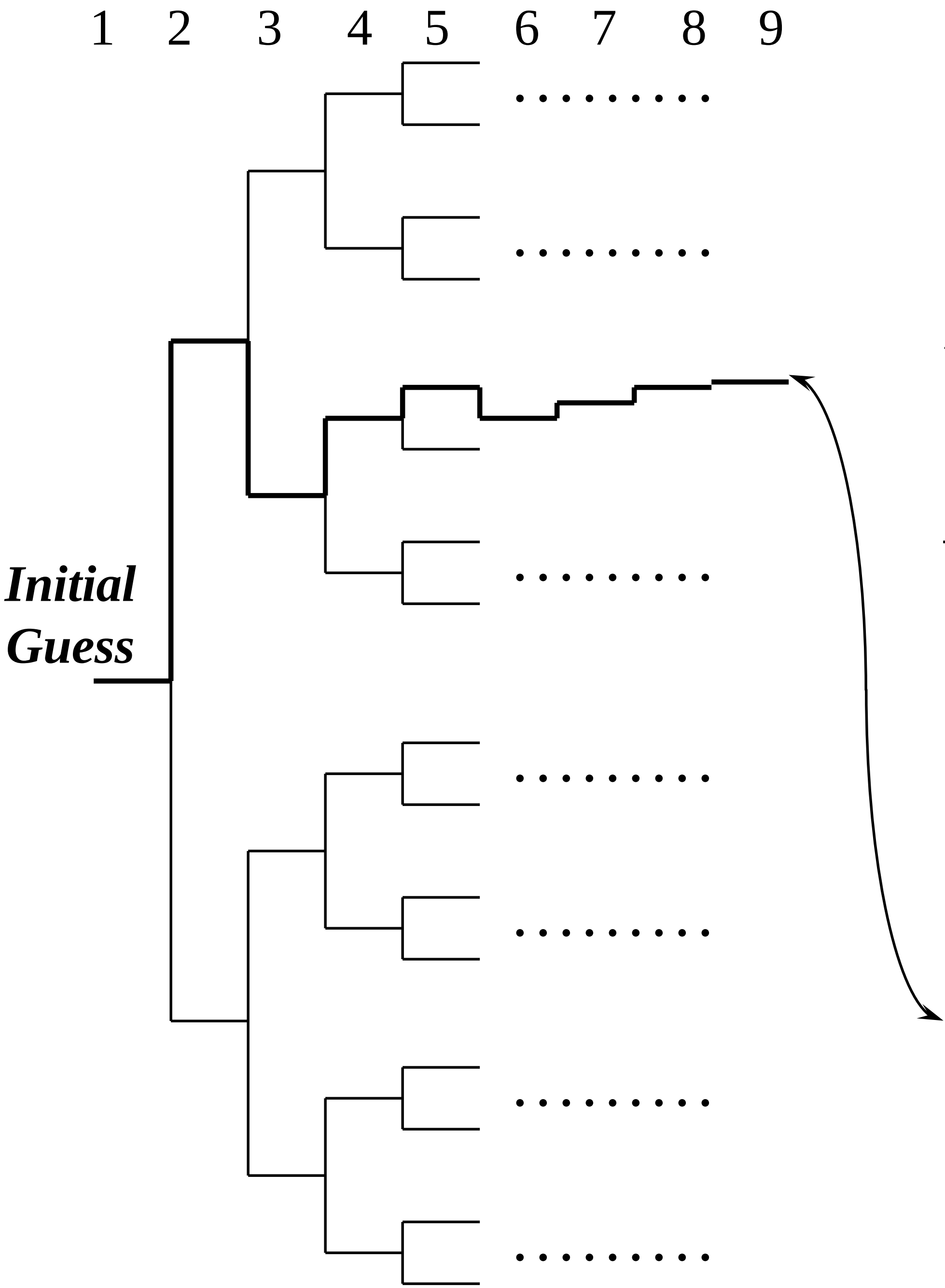
step 2 – If *max* < *min*, then stop. No element matches the key.

step 3 – Let *mean* = (*max* + *min*)/2.

step 4 – If *key* equals **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 step2.



A particular search:

Guess #	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Value
1	0	1	1	1	1	1	1	1	(127)
2	1	0	1	1	1	1	1	1	(191)
3	1	0	0	1	1	1	1	1	(159)
4	1	0	1	0	1	1	1	1	(175)
5	1	0	1	1	0	1	1	1	(183)
6	1	0	1	1	0	0	1	1	(179)
7	1	0	1	1	0	1	0	1	(181)
8	1	0	1	1	0	1	1	0	(182)
9	1	0	1	1	0	1	1	1	(183)

Best Estimate
LSB corrected on 9th try

Example: Write a program to implement the binary search algorithm and a sequence of instructions to test it. Use an array of n 8-bit elements for implementation.

```
n       EQU     45
key     EQU     69

        ORG     $1000
        CLRA
        STAA    min     ; initialize
        STAA    result  ; min $ result
        LDAB    #n-1
        STAB    max     ; initialize max
        LDX     #arr    ; set pointer
loop    CLRA
        LDAB    min
        CMPB    max
        LBHI    nfound
        ADDB    max
        ADCA    #0
        LSRD
        STAB    mean
        LDAA    B,X
        CMPA    #key
        BEQ     found
        BHI     srch_lo
        LDAA    mean
        INCA
        STAA    min
        BRA     loop
srch_lo LDAA    mean
        DECA
        STAA    max
        BRA     loop
found   STAA    result
nfound  SWI

max     RMB     1
min     RMB     1
mean    RMB     1
result  RMB     1

arr     DB      1,3,6,9,11,20,30,45
        DB      48,60,61,63,64,65
        DB      67,69,72,74,76,79,80
        DB      83,85,88,90,110,113
        DB      114,120,123,142,151
        DB      175,183,186,190,199
        DB      200,215,218,222,234
        DB      237,241,252

        END
```

Passing Parameters in Subroutine

The passing of arguments is implemented by means of:

- Registers
- Global variables
- Stack
- An argument list
- A table

An example for some cases will follow.

Passing Parameters By Register(s)

Example:

Write a subroutine to compute the average of an array with N 8-bit elements, and write the instruction sequence to call the subroutine. The array starts at ARRAY. Use registers to pass parameters and return the average in the B accumulator.

```
* Calling sequence …….
                .
                .
                .
                LDX        #ARRAY     ;
                LDAA       #N         ;
                JSR        average    ; call the subroutine "average"
                .
                .
                .
```

```
average   CMPA   #1        ; check if N < 1
          BLO    err       ; if array empty, exit with message
          BEQ    ans       ; single element array





ans       LDAB   0,X       ; get the single element and return
          RTS              ;
err       LDD    #errmsg   ; let user know that array is empty
          LDX    printf    ; display the message
          JSR    0,X       ;
          RTS              ;
errmsg    FCC    ‘The array is empty!! Can not find the average.’
          DB     $0D,$0A,0
```

```
average   CMPA   #1          ; check if N < 1
          BLO    err         ; if array empty, exit with message
          BEQ    ans         ; single element array
          TAB                ; create a 2-byte number out of N
          CLRA               ;
          XGDY               ; place N in Y
          PSHY               ; secure N in stack
          CLRA               ; initialize the array sum to zero
          CLRB               ;
loop      ADDB   1,X+        ; add an element to the sum
          ADCA   #0          ; add carry to the upper 8-bit of the sum
          DBNE   Y,loop      ; is this end of the loop?
          PULX               ; load N into X
          IDIV               ; compute the array average
          XGDX               ; B accumulator has the average
          RTS                ;
ans       LDAB   0,X         ; get the single element and return
          RTS                ;
err       LDD    #errmsg     ; let user know that array is empty
          LDX    printf      ; display the message
          JSR    OUTSTRG     ;
          RTS                ;
errmsg    FCC    'The array is empty!! Can not find the average.'
          DB     $0D,$0A,0
```

Passing Parameters Using Stack

Example:

Write a subroutine to find the maximum element of an array, and write an instruction sequence to call this subroutine. The following parameters are passed in the stack: ***array*** - the starting address of the given array; ***arcnt*** - the array count; and ***armax*** - address to hold the maximum element of the array.

```
* Calling sequence …….
              .
              .
              LDX      #array
              PSHX
              LDAA     #arcnt
              PSHA
              LDX      #armax
              PSHX
              JSR      MAX
              LEAS     5,SP        ; clean up the stack
              .
              .
              .
```

```
MAX     LEAY    1,SP        ; point Y to top element of stack
        LDX     3,Y         ; place address 'array' in X
        LDAA    0,X         ; get the first element

        LDAB    2,Y         ; load the array count into B
        CMPB    #1          ; check array count
        BLO     err         ; let user know array is empty
        BEQ     ans         ; single element array










ans
        STAA    0,X         ; save the array MAX
        RTS                 ;

err     LDD     #errmsg     ; let user know that array is empty

        LDX     printf      ; display the message
        JSR     0,X         ;
        RTS                 ;
errmsg  FCC     'The array is empty!! Can not find the maximum.'
        DB      $0D,$0A,0
```

```
MAX     LEAY    3,SP        ; point Y to top element of stack
        LDX     3,Y         ; place address 'array' in X
        LDAA    0,X         ; get the first element

        LDAB    2,Y         ; load the array count into B
        CMPB    #1          ; check array count
        BLO     err         ; let user know array is empty
        BEQ     ans         ; single element array
loop    CMPA    1,+X        ; compare the next element with the MAX
        BHS     noswap      ; should MAX be updated
        LDAA    0,X         ; update MAX
noswap  DBNE    B,loop      ; decrement loop count, is this end of the loop?
ans     LDX     0,Y         ; load 'armax' into X
        STAA    0,X         ; save the array MAX
        RTS                 ;

err     LDD     #errmsg     ; let user know that array is empty
        LDX     printf      ; display the message
        JSR     0,X         ;
        RTS                 ;
errmsg  FCC     'The array is empty!! Can not find the maximum.'
        DB      $0D,$0A,0
```

Passing Parameters By Argument List

Example:

How to implement equivalent of CALL SUB(A,B,C) ?

Main Program

```
* Calling sequence …….
                .
                .
                .
                JSR        SUB        ;
                FDB        A,B,C      ;
                .
                .
                .
```

Subroutine

```
SUB     LEAX    1,SP
        LDX     0,X      ; pointing at 1st.
*                         ;argument
        .
        LDD     0,X      ;
        LDY     4,X      ;
        .
        .
        .
        .
        PULX
        JMP     6,X
```

Instead of RTS

How to Create Hole in the Stack

Main Program

```
LEAS  -8,SP
BSR   SIN
LEAS  8,SP
```

Subroutine SIN

```
LEAX   1,SP
STAA   2,X

RTS
```

This example created an 8-byte hole in the stack. The hole can be as many byte as desired.