THE UNIVERSITY OF ARIZONA
UA South

CYBV 471 Assembly Programming for Security Professionals
Week 15

Data Structures-2
Floating Point Arithmetic

Agenda

- **Data Structure-2**
  - Understand Linked lists
  - Linked lists in C
  - Create Linked lists in assembly in .data section
  - Create Linked lists in assembly in .text section
  - Doubly-linked lists
  - Binary trees

- **Floating Point Arithmetic**
  - Floating point registers
  - Moving floating point data
  - Addition and Subtraction
  - Multiplication and division
  - Conversion
  - Floating point comparison
  - Mathematical functions

Week 15 Assignments

- **Learning Materials**

  1- Week 15 Presentation

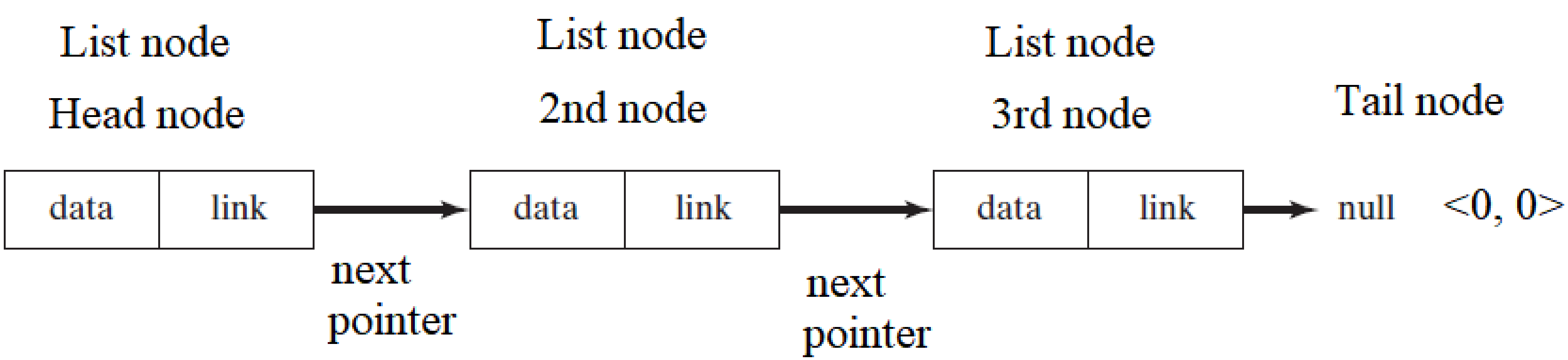
  2- Read Pages 68-71 (NASM manual)

- **Assignment**

  1- Complete “Lab 15 ” by coming Sunday 11:59 PM.

Linked List

- Linked List is a linear data structure. Each element of the linked list is a structure
- Linked list looks like an array of structures
- However, unlike arrays, linked list elements are not stored at contiguous location
- Each element could be stored in different memory location
- To construct the linked list, the elements are linked using pointers
- Each node in a linked list contains a data area and a link (next) area
- First node in the linked list is called “head”
- Last node in the linked list is called “tail” that has “NULL” pointer



Linked List in C

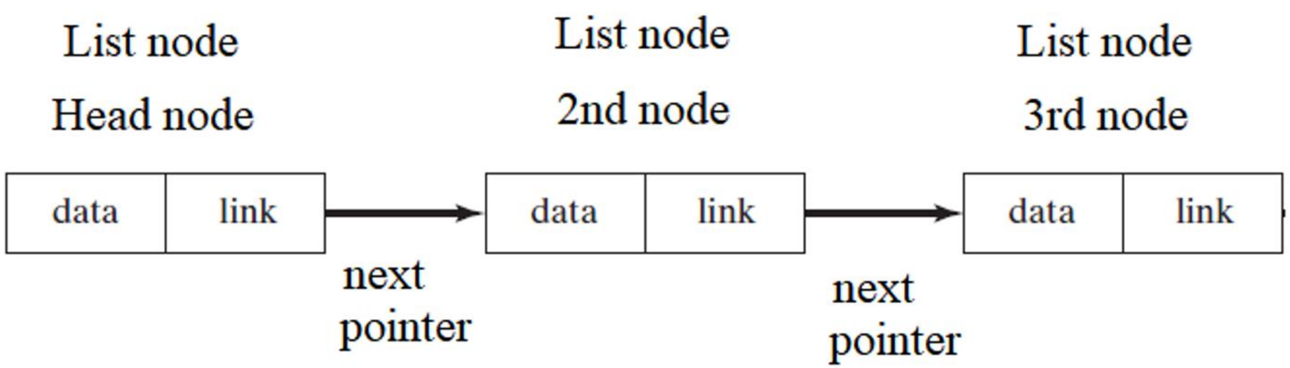
```
// Program to create a simple linked  list with 3 nodes
#include<stdio.h>
#include<stdlib.h>
struct Node                        // define ListNode structure
   {
  int data;                              // data section has one field only. We could have several fields
  struct Node *next;              // a pointer to the next structure in the linked list
   };
int main()
 {
  struct Node* head = NULL;          // declare first node, head pointer points to NULL
  struct Node* second = NULL;      // declare second node, second pointer points to NULL
  struct Node* third = NULL;         // declare third node, third pointer points to NULL

   // allocate memory for each node and return a pointer for each node created in the memory
  head = (struct Node*)malloc(sizeof(struct Node));     // head pointer points to the head node
  second = (struct Node*)malloc(sizeof(struct Node));  // second pointer points to the second node
  third = (struct Node*)malloc(sizeof(struct Node));     // third pointer points to the third node
```

Linked List in C

```
/* Three blocks have been allocated  dynamically.
   We have pointers to these three blocks as first,
   second and third
     head           second            third
       |              |                 |
       |              |                 |
   +---+-----+     +----+----+     +----+----+
   | #   | #   |     | #   | #   |     |  # |   # |
   +---+-----+     +----+----+     +----+----+
```

// We didn't assign yet values for data field and next pointer field

How could create linked list?



Linked List in C

```
// create the linked list
  head->data = 1;                //assign data in first node
  head->next = second;       // Link first node with the second node
```

```
/* data has been assigned to data part of first
   block (block pointed by head).  And next
   pointer of first block points to second.
   So they both are linked.

       head           second          third
        |                |              |
        |                |              |
    +---+---+     +----+----+     +-----+-----+
    | 1  | o----->| #  | #  |     |  #  | #  |
    +---+---+     +----+----+     +-----+-----+
*/
```

Linked List in C

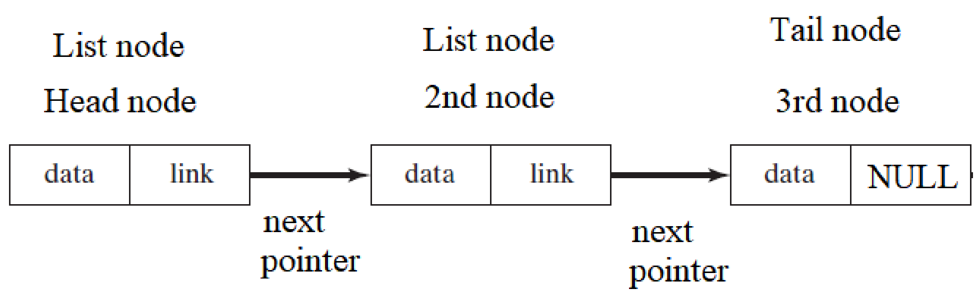
```
// create the linked list
  second->data = 2;            //assign data in second node
  second->next = third;        // Link second node with the third node
```

```
/* data has been assigned to data part of second
   block (block pointed by second). And next
   pointer of the second block points to third
   block. So all three blocks are linked.

       head          second          third
        |              |               |
        |              |               |
    +---+---+      +---+---+      +----+----+
    | 1  | o----->| 2 | o-----> |  # |  # |
    +---+---+      +---+---+      +----+----+      */
```

Linked List in C

```
// create the linked list
  third->data = 3;            //assign data in third node
  third->next = NULL;         // Terminate the linked list
 return 0;    end the program
}
```

```
/* data has been assigned to data part of third
  block (block pointed by third). And next pointer
  of the third block is made NULL to indicate
  that the linked list is terminated here.

   We have the linked list ready.

          head
            |
            |
     +---+---+     +---+---+       +----+------+
     | 1  | o----->| 2  | o-----> |  3 | NULL |
     +---+---+     +---+---+       +----+------+
```



Linked List in Assembly

- 1- Create “Node” above the .data section (global data structure)
- 2- Create the “Linked list” in the .data section
- Or create the “Linked lists” in the .text section

Create the Linked List in .Data Section

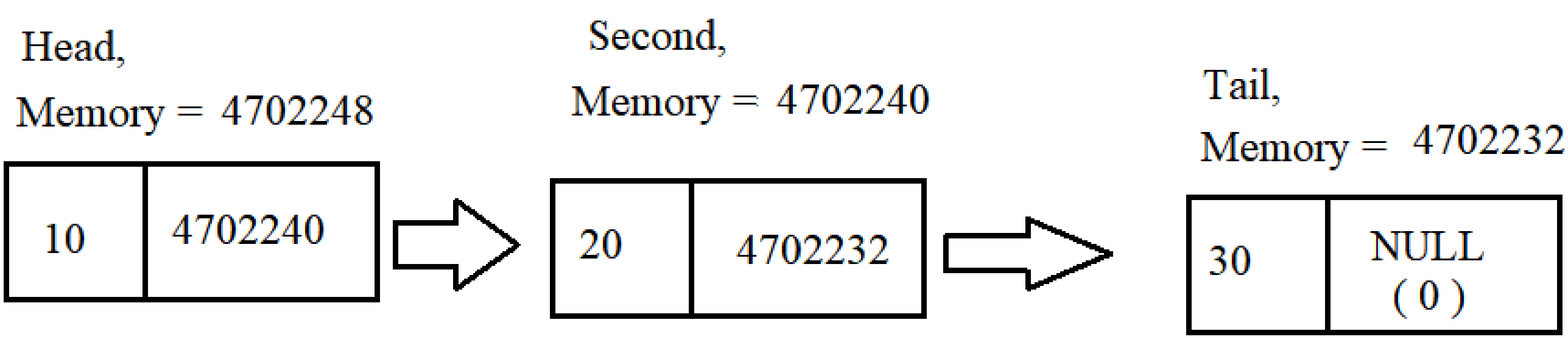
```
; link1.asm     create a linked list from three nodes

 STRUC Node                          ;  define a node structure
       .Value:       resd 1          ;  data fie
       .NextPtr:     resd 1          ;  pointer field
        .size:
 ENDSTRUC

section .data
 ;declare three nodes and create the linked list

Head: ISTRUC Node
    AT  Node.Value,    dd 0                  ; initilaize the fields
    AT  Node.NextPtr,  dd Second             ; point to Second node
IEND

Second: ISTRUC Node
     AT  Node.Value,    dd 0                 ; initilaize the fields
     AT  Node.NextPtr,  dd Tail              ; point to Tail node
IEND

Tail: ISTRUC Node
     AT  Node.Value,    dd 0                 ; initilaize the fields
     AT  Node.NextPtr,  dd 0                 ; NULL
IEND

   msg1:    db "Print nodes information at the start of program ",10, 0
   msgL1:    equ $-msg1

   msg2:    db "Printing the linked list information ",10, 0
   msgL2:    equ $-msg2

   msg3:    db "Print pointer values at the end of program ",10, 0
   msgL3:    equ $-msg3
```

```
SECTION .text
 global main
 main:
      push ebp
      mov ebp, esp

      mov ecx,msg1              ; print start values
      mov edx,msgL1
      call PString

     ; print start values of each node
      mov eax, Head             ; Memory location of head node
      call printDec
      call  println

      mov eax, [Head]           ; content of Memory location of head node
      call printDec
      call  println

      mov eax, Second           ; Memory location of tail node
      call printDec
      call  println

      mov eax, [Second]         ; Memory location of tail node
      call printDec
      call  println

      mov eax, Tail             ; Memory location of tail node
      call printDec
      call  println

      mov eax, [Tail]           ; Memory location of tail node
      call printDec
      call  println
```

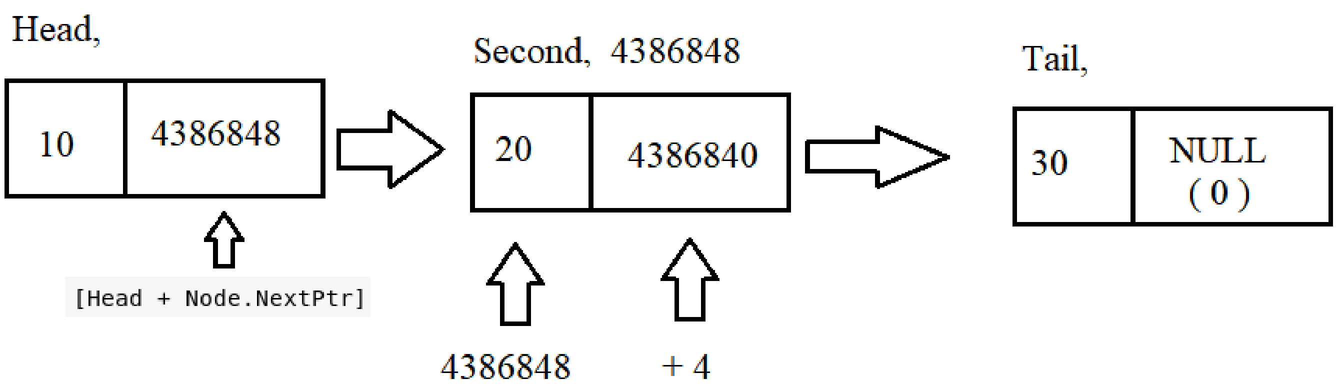
```
; Set the head node value
mov   WORD [Head + Node.Value],10          ; set Node.value for Head structure

;Set the second node value
mov  WORD [Second + Node.Value],20        ; set Node.value for Second structure

; Set the tail node value
mov  WORD [Tail + Node.Value],30          ; set Node.Value for Tail structure

mov ecx,msg2            ; print linked list information
mov edx,msgL2
call PString

; print the data field of head node
mov eax, [Head + Node.Value]      ; Date value
call printDec
call  println

mov eax, [Head + Node.NextPtr]    ; pointer value
call printDec
call  println

; print the data field of second node
mov eax, [Second + Node.Value]      ; Date value
call printDec
call  println

mov eax, [Second + Node.NextPtr]    ; pointer value
call printDec
call  println

; print the data field of tail node
mov eax, [Tail + Node.Value]      ; Date value
call printDec
call  println
```

```
mov ecx,msg3              ; print end values
mov edx,msgL3
call PString

; print end values of each node
mov eax,  Head              ; Memory location of tail node
call printDec
call  println

mov eax,  [Head]             ; content of Memory location of head node
call printDec
call  println

mov eax,  Second             ; Memory location of tail node
call printDec
call  println

mov eax,  [Second]           ; Memory location of tail node
call printDec
call  println

mov eax,  Tail              ; Memory location of tail node
call printDec
call  println

mov eax,  [Tail]             ; Memory location of tail node
call printDec
call  println


; exit the program and cleaning
mov esp, ebp
pop ebp
ret
```

```
root@kali-Test: ~/Desktop/Week-14
File  Edit  View  Search  Terminal  Help
root@kali-Test:~/Desktop/Week-14# nasm -g -f elf link1.asm -o link1.o
root@kali-Test:~/Desktop/Week-14# gcc -m32 -lc link1.o -o link1
root@kali-Test:~/Desktop/Week-14# ./link1
Print nodes information at the start of program
4702248
0
4702240
0
4702232
0
Printing the linked list information
10
4702240
20
4702232
30
0
Print pointer values at the end of program
4702248
10
4702240
20
4702232
30
root@kali-Test:~/Desktop/Week-14#
```



Linked List in Assembly Language




```
mov esi, [Head + Node.NextPtr]
mov eax, [esi]        ; should be data value of second node
call printDec
call  println


mov eax, [esi+ 4] ; should be the nexPtr value of second node = Tail memory
call printDec
call  println
```

```
root@kali-Test:~/Desktop/Week-14# nasm -g -f elf link3.asm -o link3.o
root@kali-Test:~/Desktop/Week-14#  gcc -m32 -lc link3.o -o link3
root@kali-Test:~/Desktop/Week-14# ./link3
10
4386848
20
4386840
30
0
Access values inside second node from memory information of Head node
20
4386840
```

Create Linked List in .Text Section

```
;link2.asm     create a linked list from three nodes inside the .text section

STRUC Node                          ;  define a node structure
      .Value:      resd 1           ;  data fie
      .NextPtr:     resd 1          ;  pointer field
      .size:
ENDSTRUC
section .data
 ;declare three nodes with intialize values = 0

Head: ISTRUC Node
    AT  Node.Value,    dd 0          ; initilaize the fields
    AT  Node.NextPtr,  dd 0          ; NULL
IEND

Second: ISTRUC Node
    AT  Node.Value,    dd 0          ; initilaize the fields
    AT  Node.NextPtr,  dd 0          ; NULL
IEND

Tail: ISTRUC Node
    AT  Node.Value,    dd 0          ; initilaize the fields
    AT  Node.NextPtr,  dd 0          ; NULL
IEND
msg1:     db "Print pointer values at the start of program ",10, 0
msgL1:     equ $-msg1

msg2:     db "Create a listed links ",10, 0
msgL2:     equ $-msg2

msg3:     db "Printing the linked list information ",10, 0
msgL3:     equ $-msg3

msg4:     db "Print pointer values at the end of program = ",10, 0
msgL4:     equ $-msg4
```

```
SECTION .text
 global main
 main:
      push ebp
      mov ebp, esp

      mov ecx,msg1                ; print start values
      mov edx,msgL1
      call PString

     ; print start values of each node
      mov eax, Head               ; Memory location of head node
      call printDec
      call  println

      mov eax, [Head]             ; content of Memory location of head node
      call printDec
      call  println

      mov eax, Second             ; Memory location of tail node
      call printDec
      call  println

      mov eax, [Second]           ; Memory location of tail node
      call printDec
      call  println

      mov eax, Tail               ; Memory location of tail node
      call printDec
      call  println

      mov eax, [Tail]             ; Memory location of tail node
      call printDec
      call  println
```

Create Linked List in .Text Section

```
; Set the head node
 mov   DWORD [Head + Node.Value],10        ; set x value for first point
 mov   DWORD [Head + Node.NextPtr], Second      ; set the head pointer to second node

 ;Set the second node
 mov  DWORD [Second + Node.Value],20         ; set data value for first node to 10
 mov  DWORD [Second + Node.NextPtr], Tail      ; set the pointer value for second node to third node

  ; Set the tail node
 mov   DWORD [Tail + Node.Value],30            ; set data value for first node to 10
 mov  DWORD [Tail + Node.NextPtr], 0           ; set the pointer value for the tail node to NULL
                                               ; end of the linked list

 mov ecx,msg2             ; print linked list information
 mov edx,msgL2
 call PString

 mov ecx,msg3            ; print linked list information
 mov edx,msgL3
 call PString

 ; print the data field of head node

  mov eax, [Head + Node.Value]      ; Date value
  call printDec
  call  println

  mov eax, [Head + Node.NextPtr]    ; pointer value
  call printDec
  call  println

; print the data field of second node
 mov eax, [Second + Node.Value]      ; Date value
 call printDec
 call  println

 mov eax, [Second + Node.NextPtr]    ; pointer value
 call printDec
 call  println

; print the data field of Tail node
 mov eax, [Tail + Node.Value]      ; Date value
 call printDec
 call  println

 mov eax, [Tail + Node.NextPtr]    ; pointer value
 call printDec
 call  println
```

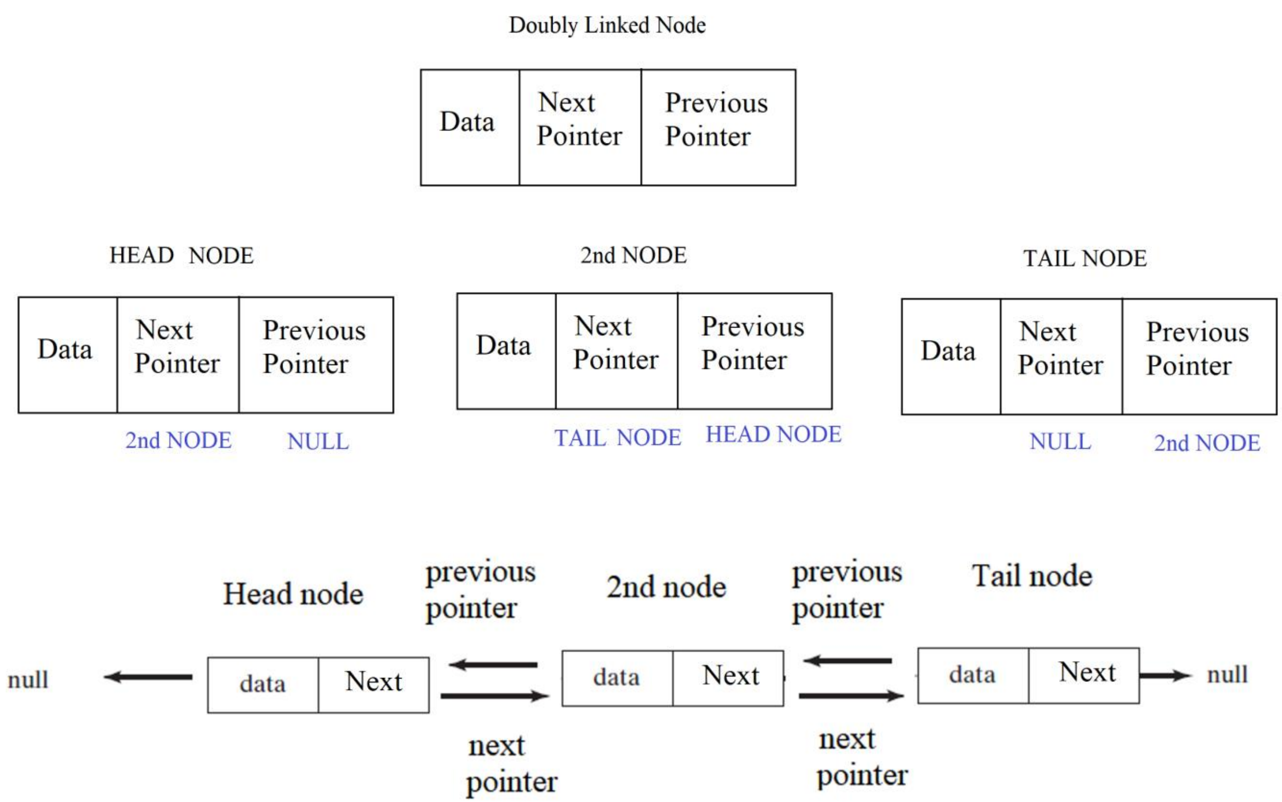
Then print the end values as before

Create Linked List in .Text Section

```
root@kali-Test: ~/Desktop/Week-14
File  Edit  View  Search  Terminal  Help
root@kali-Test:~/Desktop/Week-14# gcc -m32 -lc link2.o -o link2
root@kali-Test:~/Desktop/Week-14#  ./link2
Print pointer values at the start of program
5115928
0
5115936
0
5115944
0
Create a listed links
Printing the linked list information
10
5115936
20
5115944
30
0
Print pointer values at the end of program =
5115928
10
5115936
20
5115944
30
root@kali-Test:~/Desktop/Week-14#
```

Doubly Linked List

- Doubly Linked List is a linear data structure.
- A **D**oubly **L**inked **L**ist (DLL) contains an extra pointer, *previous pointer*, together with next pointer and data field which included in single linked list.
- First node in the linked list is called “head” points to second node. Its previous pointer points to “NULL”
- Last node in the linked list is called “tail” . Its next pointer link points to “NULL”



Doubly Linked List in C

```
// Program to create a simple doubly linked  list with 3 nodes
#include<stdio.h>
#include<stdlib.h>
struct Node                          // define ListNode structure
{
  int data;                              // data section has one field only. We could have several fields
  struct Node *next;               // a pointer to the next structure in the linked list
  struct Node *previous;        // a pointer to the next structure in the linked list
};
int main()
 {
  struct Node* head = NULL;          // declare first node, head pointer points to NULL
  struct Node* second = NULL;      // declare second node, second pointer points to NULL
 struct Node* third = NULL;         // declare third node, third pointer points to NULL

   // allocate memory for each node and return a pointer for each node created in the memory
  head = (struct Node*)malloc(sizeof(struct Node));     // head pointer points to the head node
  second = (struct Node*)malloc(sizeof(struct Node));  // second pointer points to the second node
  third = (struct Node*)malloc(sizeof(struct Node));     // third pointer points to the third node
```

Doubly Linked List in Assembly

1- Declare a node

```
STRUC NODE
    .Value: resd 1        ; data field
    .NextPtr   resd 1     ; next pointer field
    .PrevPtr   resd  1    ; previous pointer field
```

2- Initialize a node

```
section . data
    Tail: ISTRUC Node
        AT   Node.Value, dd 10
        AT   Node.NextPtr, dd 0
        AT  Node.PrevPtr, dd Second
     IEND
```

3- Access node field

```
section .text
mov eax, [Second + Node.Value]
mov eax, [Second + Node.NextPtr]
mov eax, [Second + Node.PrevtPtr]
```

Binary Tree

A binary tree is made of nodes, where each node contains a "left" reference, a "right" reference, and a data element.

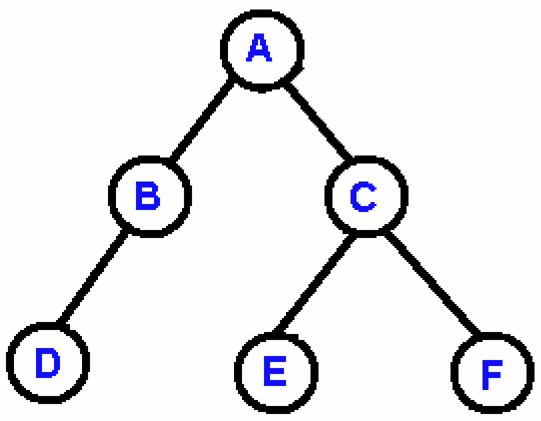
The topmost node in the tree is called the root.

Every node (excluding a root) in a tree is connected by a directed edge from exactly one other node. This node is called a parent.

On the other hand, each node can be connected to arbitrary number of nodes, called children.

Nodes with no children are called leaves, or external nodes.

Nodes which are not leaves are called internal nodes. Nodes with the same parent are called siblings



Binary Tree

1- Declare a node

```
STRUC NODE
    .Value: resd 1        ; data field
    .LeftPtr   resd 1     ; left pointer field
    .RightPtr   resd  1   ; right pointer field
```

2- Initialize a node

```
section . data
    Tail: ISTRUC Node
        AT   Node.Value, dd 10
        AT   Node.LeftPtr, dd 0
        AT  Node.RightPtr, dd Second
     IEND
```

3- Access node field

```
section .text
mov eax, [Second + Node.Value]
mov eax, [Second + Node.LeftPtr]
mov eax, [Second + Node.RightPtr]
```

Floating Point Arithmetic

Floating-Point Representation

- Representation and dealing with floating point numbers are different than those of integer numbers
- Floating point numbers stored in specific format and registers and have special OP codes

- Floating point numbers are stored based on binary numbers

- Sometimes fraction part of floating point number can't easily be represented in binary

  29.875 > 1110.111

  29.85 > 1110.11011011001100110……

  (continuous digits)

  (need to have approximate value)

- How could represent floating point number in computer?

Floating-Point Representation
Single Precision-32 bits representation

- One bit for the sign, 8 bits for the exponent, and 23 bits for the fraction number (1.xxxxxx)
- Sliding the actual number (in the 23 bits field) back and forth will change the the exponent value by adding and subtracting to it
- Limited for scientific and financial calculations
- Biased exponent value = Actual exponent value + 127

1 bit	8 bits	23 bits

Sign
0: +
1: -

Biased Exponent Value
(0 to 255)

Actual Exponent Value
(-127 to 128)

The fraction part
Assume: 1.Fraction part

Normailzed Numebr Represenation

Positive binary number

Normalizing Binary Floating-Point Numbers Single Precision-32 bits representation

- Normalized number occurs when a single "1" appears to the left of the binary point
- Un-normalized: shift binary point until exponent is zero
- Examples

Unnormalized	Normalized
1110.1	1.1101×2^{3}
.000101	1.01×2^{-4}
1010001.	1.010001×2^{6}

Floating-Point Representation
Single Precision-32 bits representation

- Example: 13.625 in decimal. How could we present 13.625 as floating point in 32-bits ?
- The number 13.625 is 1101.101 in binary
- Normalize the number: Shift the decimal point to left three times
- 1101.101 = 1.101101 E3
- Sign bit = 0
- Actual exponent = 3.
- Biased exponent = 3 + 127 = 130 (8 bits, 130 = 10000010)
- Fraction part = 10110100000000000000000 (23 bits)

Decimal Number	Binary Number	Normalize Number	Sign , Exponent, Fraction
13.625	1101.101	1.101101 E3	0, 10000010, 10110100000000000000000
		$1.101101 * 2^3$	

$$\{1. (1/2 + 0/4 + 1/8 + 1/16 + 0/32 + 1/64) \} * 8$$

$$= (1 + 0.5 + 0 + 0.125 + 0.0625 + 0.015625) * 8$$

$$= 1.703125 * 8 = 13.625$$

Floating-Point Representation
Single Precision-32 bits representation

- How can we calculate the actual number from the 32 bit presentation?
- The 32 presentation is 0, 10000010, 101101000000000 (32 bits)
- Sign bit is 0: positive number
- Biased exponent = 130. The actual exponent is 130 – 127 = 3
- Fraction part = 10110100000000000000000 (23 bits)
- The actual number is (1. Fraction) E (actual value)
- The actual number is 1.101101 E 3 = 1.101101 * 8 = 1.703125 * 8 = 13.625

Decimal Number	Binary Number	Normalize Number	Sign , Exponent, Fraction
13.625	1101.101	1.101101 E3	0, 10000010, 10110100000000000000000
		$1.101101 * 2^{3}$	

$$\{1.\,(1/2 + 0/4 + 1/8 + 1/16 + 0/32 + 1/64)\} * 8$$

$$= (1 + 0.5 + 0 + 0.125 + 0.0625 + 0.015625) * 8$$

$$= 1.703125 * 8 = 13.625$$

Floating-Point Representation
Single Precision-32 bits representation

1 bit	8 bits	23 bits

Sign
0: +
1: -

Biased Exponent Value
(0 to 255)

Actual Exponent Value
(-127 to 128)

The fraction part
Assume: 1.Fraction part

Normailzed Numebr Represenation

Positive binary number

Binary Value	Biased Exponent	Sign, Exponent, Fraction
-1.11	127	1 01111111 11000000000000000000000
+1101.101	130	0 10000010 10110100000000000000000
-.00101	124	1 01111100 01000000000000000000000
+100111.0	132	0 10000100 00111000000000000000000
+.0000001101011	120	0 01111000 10101100000000000000000

Floating-Point Representation
Double Precision-64 bits representation

- One bit for the sign, 11 bits for the exponent, and 53 bits for the number
- Provide more accuracy to represent the floating point number
- It is the default format of floating-point number in C
- About 14 digits of precision in decimal.
- More appropriate for scientific and financial calculations
- Biased exponent value = Actual exponent value + 1023

1 bit	11 bits	52 bits
Sign 0: + 1: -	Biased Exponent Value (0 to 2047) Actual Exponent Value (-1023 to 1024)	The fraction part Assume: 1.Fraction part Normailzed Numebr Represenation Positive binary number More accuracy represenattaion

Floating-Point Representation
Double Extended Precision- 80 bits representation

- One bit for the sign, 16 bits for the exponent, and 63 bits for the number
- Used by computer hardware for internal floating point calculations
- This internal format you won’t need to deal with unless you are doing some very special things with floating point numbers
- Internal conversions between other format to the 80 bits format is automatic
- Biased exponent value = Actual exponent value + 16383

1 bit	15 bits	63 bits

Sign
0: +
1: -

Biased Exponent Value
(0 to 32767)

Actual Exponent Value
(-16383 to 16384)

The fraction part
Assume: 1.Fraction part

Normailzed Numebr Represenation

Positive binary number

More accuracy represenattaion

Floating Point Arithmetic

- Special operations have to be done before arithmetic can be performed to floating points numbers

- Adding or subtracting two floating numbers
  - They have to have same exponent values
  - The numbers have to be shifted to adjust the exponent values of the two numbers

- Multiplication: Add the two exponents
  - Some precision could be lost with the multiplication (rounding)

- Division: Subtract the two exponents
  - Some precision could be lost with the division (rounding)

Addition: Floating Point Arithmetic

- Adding or subtracting two floating numbers

  - They have to have same exponent values
- If they are not already equal, then they must be made equal by shifting the significand of the number with the smaller exponent.
- For example, consider 10.375 + 6.34375 = 16.71875
- In binary, the result is 10.00011 * 8 = 10000.110 in binary = 16.75.

$$\begin{array}{r} 10.375 \\ +\,6.34375 \\ \hline 16.71875 \end{array}$$

$$\begin{array}{lr} & 1.0100110 \times 2^3 \\ + & 1.1001011 \times 2^2 \\ \hline \end{array}$$

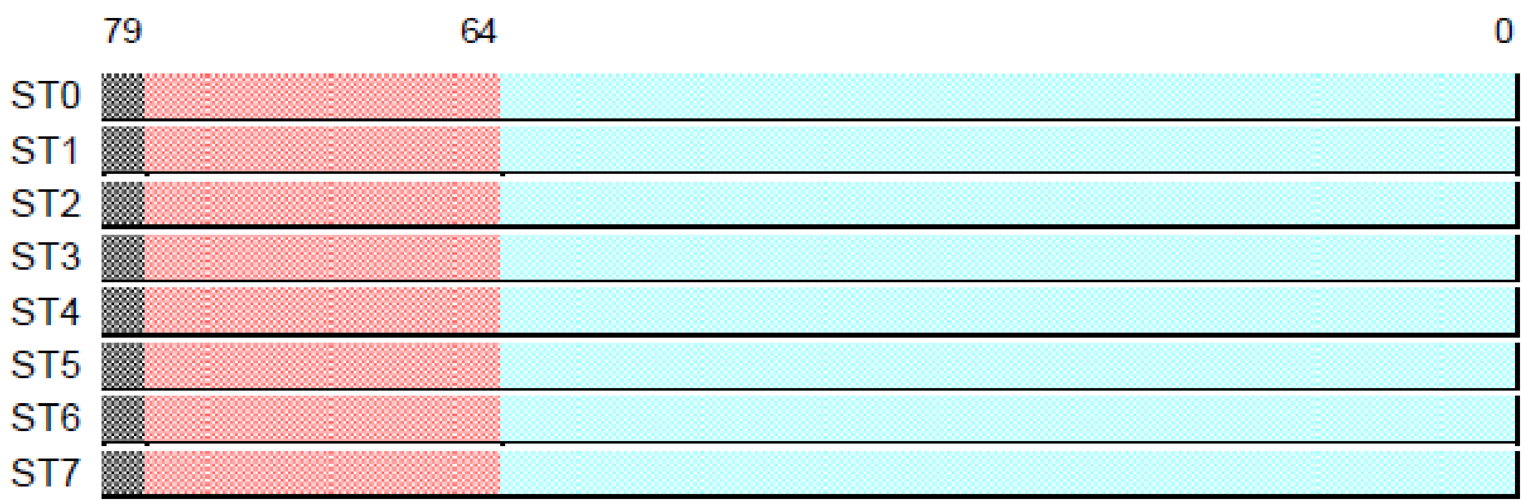
$$\begin{array}{lr} & 1.0100110 \times 2^3 \\ + & 0.1100110 \times 2^3 \\ \hline & 10.0001100 \times 2^3 \end{array}$$

Floating Point Unit
The Numeric Coprocessor

- Intel provides the Floating Point Unit (FPU) (as part of the CPU, but before it used to be a separate unit.
- The coprocessor for the 8086/8088 was called the 8087.
- The 80x86 FPUs add 13 registers to the 80x86 and later processors: ***eight floating point data registers, a control register, a status register***, a tag register, an instruction pointer, and a data pointer
- The registers are named ST0, ST1, ST2, . . . ST7. The floating point registers are used differently than the integer registers of the main CPU.
- The floating point registers are organized as a stack.

FPU Data Registers

- The FPUs provide eight 80- bit data registers organized as a stack
- ST0 refers to the item on the top of the stack, ST1 refers to the next item on the stack, and so on.
- Floating point instructions push and pop items on the stack



FPU Instructions

- The FPU adds over 80 new instructions to the 80x86 instruction set. We can classify these instructions as:
  - Data movement instructions
  - Conversions
  - Arithmetic instructions
  - Comparisons
  - Constant instructions
  - Transcendental instructions
  - Miscellaneous instructions

Floating Point Instructions

- To make it easy to distinguish the normal CPU instructions (such as mov, sub, add), from coprocessor ones, all floating point instructions start with an F.

FLD Instruction

- There are several instructions that load data onto the top of the coprocessor register stack:
- The FLD instruction loads a 32 bit, 64 bit, or 80 bit floating point value onto the stack

FLD source

- Loads a floating point number from memory onto the top of the stack.
- The source may be a single, double or extended precision number or a coprocessor register.
- Examples:
  - fld(st1);
  - fld(real32_variable);
  - fld(real64_variable);
  - fld(real80_variable);

- FLD1 stores a “1” on the top of the stack.
- FLDZ stores a “zero” on the top of the stack

FST Instruction

- FST Instructions store data from the stack into Memory

FST *dest*

- Stores the top of the stack (ST0) into memory.
- The destination could be a single or double precision number or a coprocessor register

FSTP *dest*

- Stores the top of the stack into memory just as FST; however, after the number is stored, its value is popped from the stack.

FIST *memory dest*

- Converts the number in ST0 into integer and stores the value in memory *dest*

Addition Instruction

- Each of the addition instructions compute the sum of ST0 and another operand.
- The result is always stored in a coprocessor register ST0.

  FADD src ; ST0 = STO + src .

- The src may be any coprocessor register or a single or double precision number in memory.

  FADD dest, STO ; dest = dest + ST0.

- The dest may be any coprocessor register.

Addition Instruction

➢ Example: Calculate the sum of array of floating elements

```
segment .bss
array        resq SIZE
sum          resq 1

segment .text
      mov    ecx, SIZE
      mov    esi, array
      fldz                  ; ST0 = 0
lp:
      fadd   qword [esi]    ; ST0 += *(esi)
      add    esi, 8         ; move to next double
      loop   lp
      fstp   qword sum      ; store result into sum
```

Subtraction Instruction

- Each of the subtraction instructions compute the subtract of ST0 and another operand.
- The result is always stored in a coprocessor register ST0.

  FSUB src ; ST0 = ST0 - src .
- The src may be any coprocessor register or a single or double precision number in memory.

  FSUB dest, STO ; dest = dest - ST0.

- The dest may be any coprocessor register.

Multiplication Instruction

- Each of the multiplication instructions compute the subtract of ST0 and another operand.
- The result is always stored in a coprocessor register ST0.

  FMUL src ; ST0 = ST0 * src .

- The src may be any coprocessor register or a single or double precision number in memory.

  FMUL dest, STO ; dest = dest * ST0.

- The dest may be any coprocessor register.

Division Instruction

- Each of the division instructions compute the subtract of ST0 and another operand.
- The result is always stored in a coprocessor register ST0.

  FDIV src ; ST0 = ST0 / src .

- The src may be any coprocessor register or a single or double precision number in memory.

  FDIV dest, STO ; dest = dest / ST0.

- The dest may be any coprocessor register.

Other Instructions

- FPREM and FPREM1 – Computes the remainder
- FRNDINT – Rounds the number on top of stack
- FABS – Computes absolute value of ST0
- FCHS – Changes the sign of ST0
- FCOM, FCOMP, and FCOMPP – for comparison
- FSIN, FCOS, and FSINCOS – Computes the Sine and Cosine
- FPTAN – Computes the Tangent
- FYL2X – Computes $\log_2(x)$
- **See chapter 6 for more instructions and NASM manual**
- **See the following link for a summary of point instructions**

https://www.csee.umbc.edu/courses/undergraduate/CMSC313/fall04/burt_katz/lectures/Lect12/floatingpoint.html

Putting It All Together

You should know:

- **Data Structure-2**
  - Understand Linked lists
  - Linked lists in C
  - Create Linked lists in assembly in .data section
  - Create Linked lists in assembly in .text section
  - Doubly-linked lists
  - Binary trees

- **Floating Point Arithmetic**
  - Floating point registers
  - Moving floating point data
  - Addition and Subtraction
  - Multiplication and division
  - Conversion
  - Floating point comparison
  - Mathematical functions

Questions?

Thank you for your effort during the semester