ECE 220 Computer Systems & Programming

Lecture 24 – C to LC-3 with Linked Data Structure

ECE ILLINOIS

ILLINOIS

C to LC-3 – Assembly Translation with linked data structure

Recursive tree traversal

Problem statement: Convert the following function from C to LC-3. This function recursively traverses a binary tree.

```
void inorder(t_node *node)
{
    // Base case
    if(node ==NULL)
        return;
    // Recursive case
    else{
        inorder(node->left);
        printf("%d ", node->data);
        inorder(node->right);
    }
}
```

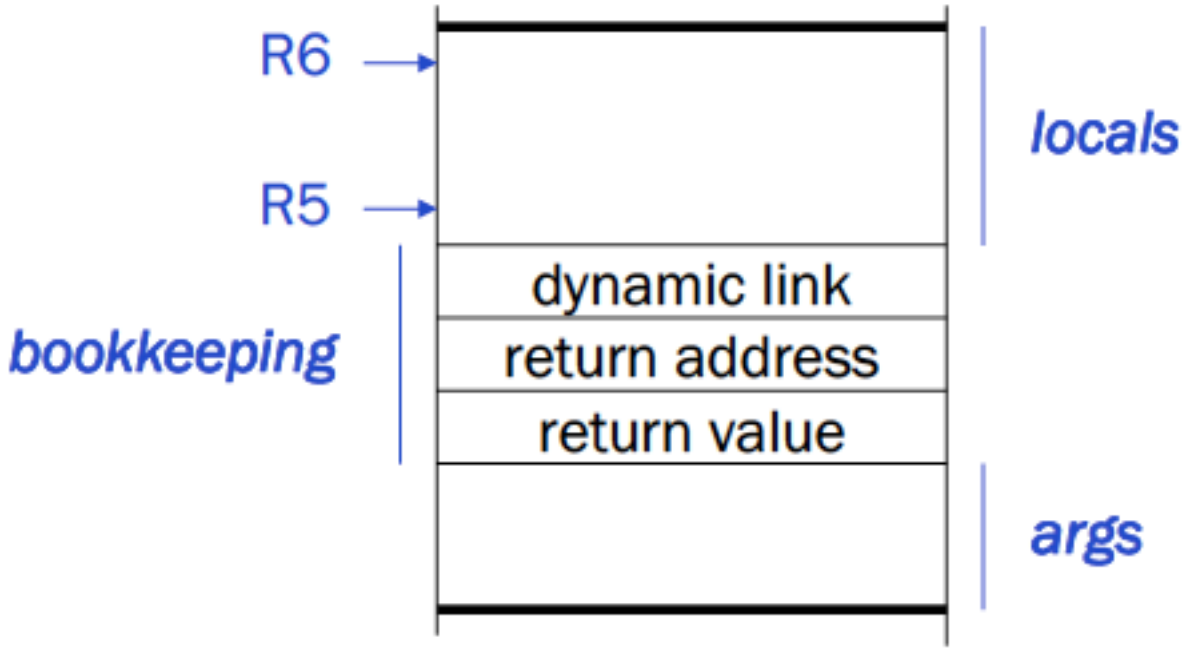
```
typedef struct nodeTag t_node;
struct nodeTag
{
    int data;
    t_node *left;
    t_node *right;
};
```

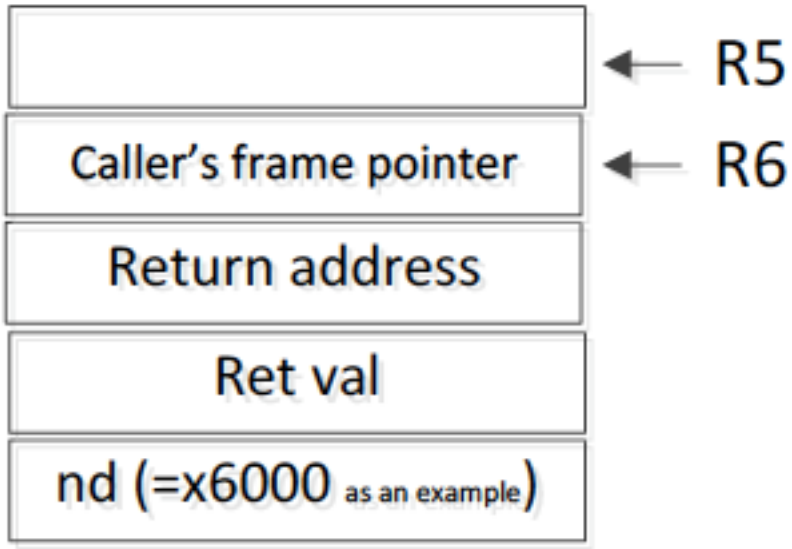
inOrder LC3 (t_node *node)

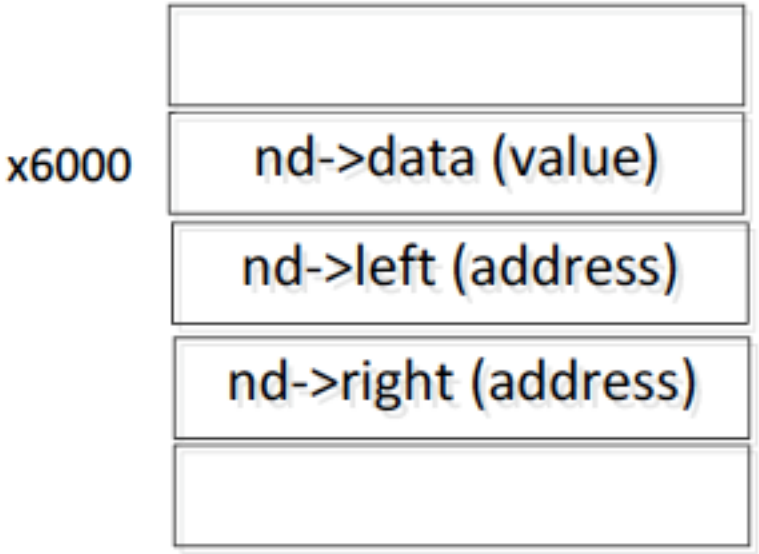
```
void inorder(t_node *node)
{
    // Base case
    if(node ==NULL)
        return;
    // Recursive case
    else{
        inorder(node->left);
        printf("%d ", node->data);
        inorder(node->right);
    }
}
```

```
typedef struct nodeTag t_node;
struct nodeTag
{
    int data;
    t_node *left;
    t_node *right;
};
```

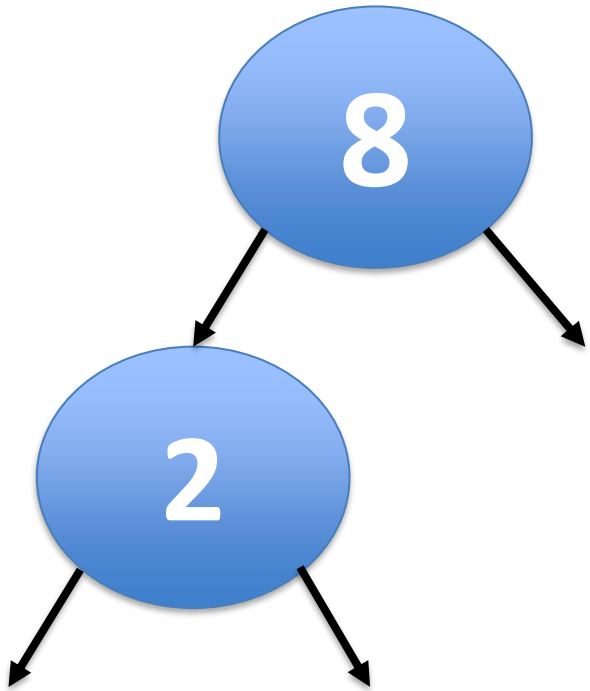
Activation Record







Data_inOrder.asm



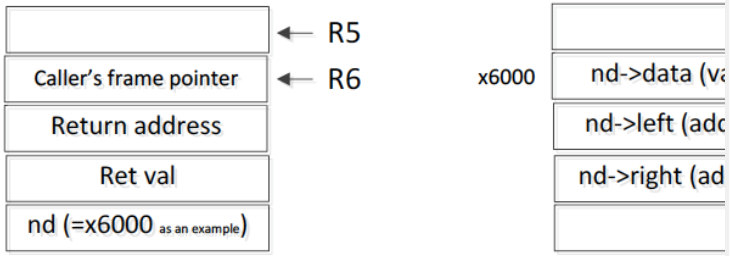

```
.ORIG x6000

.FILL x38 ; ascii 8
.FILL x6003
.FILL x0    ;NULL

.FILL x32 ; ascii 2
.FILL x0
.FILL x0   ;NULL

.END
```

```
typedef struct nodeTag t_node;
struct nodeTag
{
    int data;
    t_node *left;
    t_node *right;
};
```

inOrder.asm

```c
void inorder(t_node *node)
{
    // Base case
    if(node ==NULL)
        return;
    // Recursive case
    else{
        inorder(node->left);
        printf("%d ", node->data);
        inorder(node->right);
    }
}
```

```
.ORIG x6000

.FILL x38 ; ascii 8
.FILL x6003
.FILL x0    ;NULL

.FILL x32 ; ascii 2
.FILL x0
.FILL x0   ;NULL

.END
```



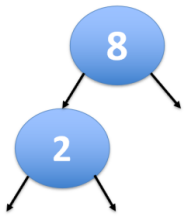

```
STACK .FILL x7000
ND_RT .FILL x6000
```

```
1  .ORIG x3000
2
3  ;;R5 - frame pointer
4  ;;R6 - stack pointer
5  ;;MAIN
6  LD R6, STACK
7  LD R5, STACK
8  LD R1, ND_RT
9  STR R1, R6, #0   ;push nd (x6000) to stack
10 JSR INORDER
11 HALT
12
13 ;;INORDER TRAVERSAL
14 INORDER
15 ;;Part 1 - push book keeping info
16 ;allocate space for return value
17 ADD R6, R6, #-1
18 ;Push return address to stack
19 ADD R6, R6, #-1
20 STR R7, R6, #0
21 ;Store old frame pointer
22 ADD R6, R6, #-1
23 STR R5, R6, #0
24 ;Set up new frame pointer
25 ADD R5, R6, #-1
26
27 ;;Part 2 - implement function logic
28 ;if (nd == NULL) skip to the end (Done)
29 LDR R1, R5, #4
30 BRz DONE
```

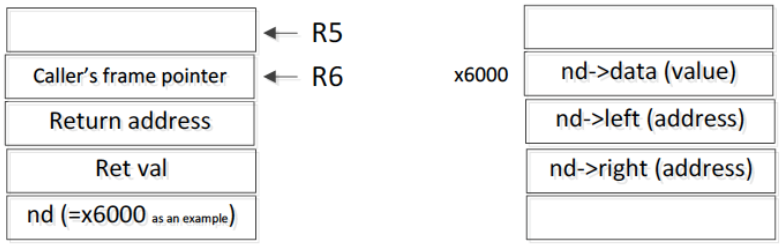
inOrder.asm(cont)

```c
void inorder(t_node *node)
{
    // Base case
    if(node ==NULL)
        return;
    // Recursive case
    else{
        inorder(node->left);
        printf("%d ", node->data);
        inorder(node->right);
    }
}
```

```
.ORIG x6000

.FILL x38 ; ascii 8
.FILL x6003
.FILL x0    ;NULL

.FILL x32 ; ascii 2
.FILL x0
.FILL x0   ;NULL

.END
```

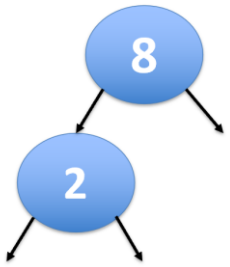



```
32 ;inorder(nd->left);
33 LDR R2, R1, #1          ;load nd->left to R2
34 ADD R6, R6, #-1
35 STR R2, R6, #0          ;push nd->left to stack
36 JSR INORDER            ;;;;;;;;;;;;;;;;;;;;;;;;
```

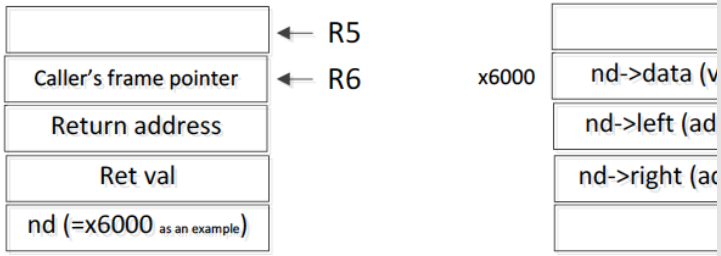


inOrder.asm

```c
void inorder(t_node *node)
{
    // Base case
    if(node ==NULL)
        return;
    // Recursive case
    else{
        inorder(node->left);
        printf("%d ", node->data);
        inorder(node->right);
    }
}
```




```
.ORIG x6000

.FILL x38 ; ascii 8
.FILL x6003
.FILL x0    ;NULL

.FILL x32 ; ascii 2
.FILL x0
.FILL x0   ;NULL

.END
```



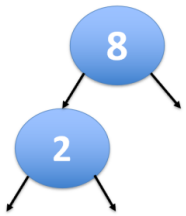

```
STACK .FILL x7000
ND_VAL .FILL x6000
```

```
12
13  ;;INORDER TRAVERSAL
14  INORDER
15  ;;Part 1 - push book keeping info
16  ;allocate space for return value
17  ADD R6, R6, #-1
18  ;Push return address to stack
19  ADD R6, R6, #-1
20  STR R7, R6, #0
21  ;Store old frame pointer
22  ADD R6, R6, #-1
23  STR R5, R6, #0
24  ;Set up new frame pointer
25  ADD R5, R6, #-1
26
27  ;;Part 2 - implement function logic
28  ;if (nd == NULL) skip to the end (Done)
29  LDR R1, R5, #4
30  BRz DONE
```

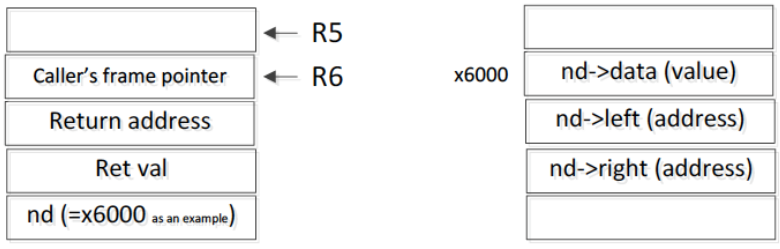
inOrder.asm(cont)

```c
void inorder(t_node *node)
{
    // Base case
    if(node ==NULL)
        return;
    // Recursive case
    else{
        inorder(node->left);
        printf("%d ", node->data);
        inorder(node->right);
    }
}
```

```
.ORIG x6000

.FILL x38 ; ascii 8
.FILL x6003
.FILL x0    ;NULL

.FILL x32 ; ascii 2
.FILL x0
.FILL x0   ;NULL

.END
```

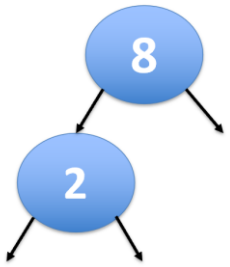



```
32 ;inorder(nd->left);
33 LDR R2, R1, #1          ;load nd->left to R2
34 ADD R6, R6, #-1
35 STR R2, R6, #0          ;push nd->left to stack
36 JSR INORDER            ;;;;;;;;;;;;;;;;;;;;;;;;
```

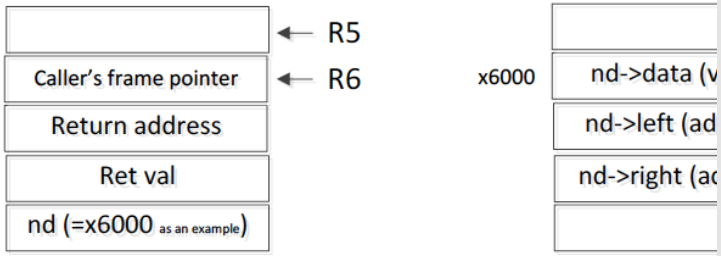


inOrder.asm

```c
void inorder(t_node *node)
{
    // Base case
    if(node ==NULL)
        return;
    // Recursive case
    else{
        inorder(node->left);
        printf("%d ", node->data);
        inorder(node->right);
    }
}
```




```
.ORIG x6000

.FILL x38 ; ascii 8
.FILL x6003
.FILL x0    ;NULL

.FILL x32 ; ascii 2
.FILL x0
.FILL x0   ;NULL

.END
```



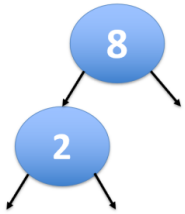

```
STACK .FILL x7000
ND_VAL .FILL x6000
```

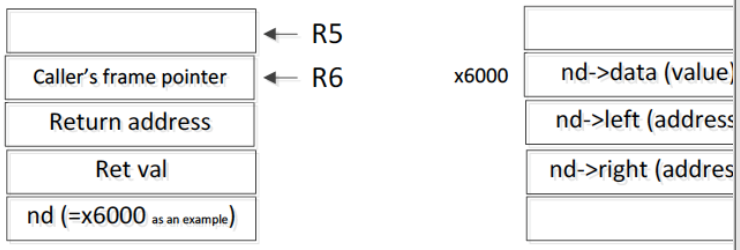
```
12
13  ;;INORDER TRAVERSAL
14  INORDER
15  ;;Part 1 - push book keeping info
16  ;allocate space for return value
17  ADD R6, R6, #-1
18  ;Push return address to stack
19  ADD R6, R6, #-1
20  STR R7, R6, #0
21  ;Store old frame pointer
22  ADD R6, R6, #-1
23  STR R5, R6, #0
24  ;Set up new frame pointer
25  ADD R5, R6, #-1
26
27  ;;Part 2 - implement function logic
28  ;if (nd == NULL) skip to the end (Done)
29  LDR R1, R5, #4
30  BRz DONE
```

inOrder.asm(cont)

```c
void inorder(t_node *node)
{
    // Base case
    if(node ==NULL)
        return;
    // Recursive case
    else{
        inorder(node->left);
        printf("%d ", node->data);
        inorder(node->right);
    }
}
```

```
.ORIG x6000

.FILL x38 ; ascii 8
.FILL x6003
.FILL x0    ;NULL

.FILL x32 ; ascii 2
.FILL x0
.FILL x0   ;NULL

.END
```

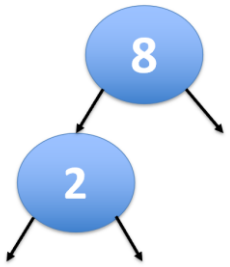




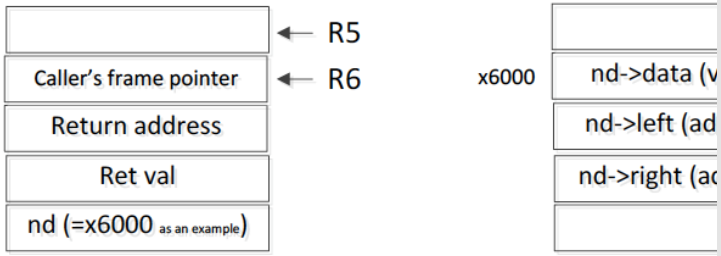

```
32 ;inorder(nd->left);
33 LDR R2, R1, #1          ;load nd->left to R2
34 ADD R6, R6, #-1
35 STR R2, R6, #0          ;push nd->left to stack
36 JSR INORDER            ;;;;;;;;;;;;;;;;;;;;;;;;
37 ADD R6, R6, #2   ;(left return);stack tear down
38 ;printf("%c", nd->data);
39 LDR R1, R5, #4
40 LDR R0, R1, #0
41 OUT
42 ;inorder(nd->right);
43 ;LDR R1, R5, #4          ;reload nd first
44 LDR R3, R1, #2          ;load nd->right to R3
45 ADD R6, R6, #-1
46 STR R3, R6, #0          ;push nd->right to stack
47 JSR INORDER             ;;;;;;;;;;;;;;;;;;;;;;;;;;
48 ADD R6, R6, #2 ;(right return);stack tear down
49
50 ;;Part 3 - tear down part of activation record
51 ;;(prepare to return)
52 DONE
53 LDR R5, R6, #0          ;restore old frame pointer
54 ADD R6, R6, #1
55
56 LDR R7, R6, #0          ;restore return address
57 ADD R6, R6, #1
58
59 RET
60
61 STACK .FILL x7000
62 ND_VAL .FILL x6000
63
64 .END
```

inOrder.asm

```c
void inorder(t_node *node)
{
    // Base case
    if(node ==NULL)
        return;
    // Recursive case
    else{
        inorder(node->left);
        printf("%d ", node->data);
        inorder(node->right);
    }
}
```




```
.ORIG x6000

.FILL x38 ; ascii 8
.FILL x6003
.FILL x0    ;NULL

.FILL x32 ; ascii 2
.FILL x0
.FILL x0   ;NULL

.END
```



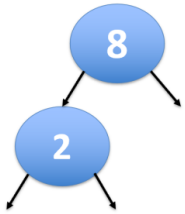

```
STACK .FILL x7000
ND_VAL .FILL x6000
```

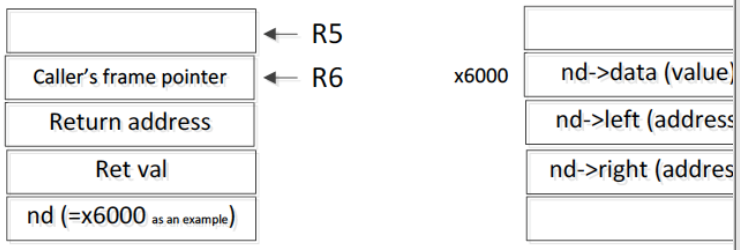
```
12
13  ;;INORDER TRAVERSAL
14  INORDER
15  ;;Part 1 - push book keeping info
16  ;allocate space for return value
17  ADD R6, R6, #-1
18  ;Push return address to stack
19  ADD R6, R6, #-1
20  STR R7, R6, #0
21  ;Store old frame pointer
22  ADD R6, R6, #-1
23  STR R5, R6, #0
24  ;Set up new frame pointer
25  ADD R5, R6, #-1
26
27  ;;Part 2 - implement function logic
28  ;if (nd == NULL) skip to the end (Done)
29  LDR R1, R5, #4
30  BRz DONE
```

inOrder.asm(cont)

```c
void inorder(t_node *node)
{
    // Base case
    if(node ==NULL)
        return;
    // Recursive case
    else{
        inorder(node->left);
        printf("%d ", node->data);
        inorder(node->right);
    }
}
```

```
.ORIG x6000

.FILL x38 ; ascii 8
.FILL x6003
.FILL x0    ;NULL

.FILL x32 ; ascii 2
.FILL x0
.FILL x0   ;NULL

.END
```






```
32 ;inorder(nd->left);
33 LDR R2, R1, #1          ;load nd->left to R2
34 ADD R6, R6, #-1
35 STR R2, R6, #0          ;push nd->left to stack
36 JSR INORDER            ;;;;;;;;;;;;;;;;;;;;;;;;
37 ADD R6, R6, #2   ;(left return);stack tear down
38 ;printf("%c", nd->data);
39 LDR R1, R5, #4
40 LDR R0, R1, #0
41 OUT
42 ;inorder(nd->right);
43 ;LDR R1, R5, #4          ;reload nd first
44 LDR R3, R1, #2          ;load nd->right to R3
45 ADD R6, R6, #-1
46 STR R3, R6, #0          ;push nd->right to stack
47 JSR INORDER             ;;;;;;;;;;;;;;;;;;;;;;;;;;
48 ADD R6, R6, #2 ;(right return);stack tear down
49
50 ;;Part 3 - tear down part of activation record
51 ;;(prepare to return)
52 DONE
53 LDR R5, R6, #0          ;restore old frame pointer
54 ADD R6, R6, #1
55
56 LDR R7, R6, #0          ;restore return address
57 ADD R6, R6, #1
58
59 RET
60
61 STACK .FILL x7000
62 ND_VAL .FILL x6000
63
64 .END
```

Left return: (inOrder.asm)		
x6FF4		R5(new)
	R5(old) = x6FF8	R6
	R.A (left Return)	
	R.V	<-R6 (when DONE is executed 1st)
x6FF8	x0	R5(new)
After 1st RET R5 is updated -	R5(old) = x6FFC	R6 <- R6 after 1st RET
with x6FF8	R.A (left return)	R2=[R1+1]=[6004]=x0
R1=[R5+4]=[6FFC]=x6003	R.V	R0 = [R1]=[6003]= 2 (printed)
x6FFC	x6003	R5(new)
	R5(old)	R6
	R.A=HALT (R7)	R2=[R1+1]=[6001]=x6003
	R.V	
x7000	x6000	R6
		main

Right Return (inOrder.asm)		
x6FF4		R5(new)
After 2nd RET R5 is updated -	R5 (old) =x6FF8	R6
with x6FF8	R.A (right return) R7	
(after 2nd DONE, RET)->R6	R.V	R3= [R1+2]=[6004]= 0 (NULL)
x6FF8	x0	R5(new) <-R6
After 2nd RET R5 is updated -	R5(old) = x6FFC	R6 <- R6 after 2nd RET
with x6FFC	R.A (left return)	R2=[R1+1]=[6004]=x0 (NULL)
After 2nd return R7 is left return	R.V	R0 = [R1]=[6003]= 2 (printed) | **<-R6**
x6FFC	x6003	R5(new)
	R5(old)	R6
	R.A=HALT (R7)	R2=[R1+1]=[6001]=x6003
	R.V	
x7000	x6000	R6
		main

Recursive linked list traversal

Problem statement: Convert the following function from C to LC-3. This function recursively traverses a linked list and prints its content.

/* typedef struct tag {char data; struct tag *next;} node; */

```
int print_list(node *head)
{
  if (!head) return 0;
  printf("%c", head->data);
  return print_list(head->next);
}
```

Main function: (print_list.asm)

```
    .ORIG x3000
MAIN
    LD R5, RSTACK
    LD R6, RSTACK

    LD R0, HEAD
    STR R0, R6, #0 ; push list head address to the stack

    JSR PRINT_LIST

    HALT
HEAD
    .FILL x2004
RSTACK
    .FILL x7000
```

```
PRINT_LIST
    ; Bookkeeping
    ADD R6, R6, #-3 ; Space for bookkeeping
    STR R7, R6, #1  ; Save return address
    STR R5, R6, #0  ; Save prev. frame pointer

    ADD R5, R6, #-1 ; Move frame pointer

    ; if (!head) return 0;
    LDR R1, R5, #4  ; R1 <- head
    BRz DONE        ; if head is NULL

    ; printf("%c", head->data);
    LDR R0, R5, #4
    LDR R0, R0, #0
    OUT
```

```
    ; print_list(head->next)
    LDR R1, R1, #1  ; R1 <- head->next
    ADD R6, R6, #-1 ; Push head->next as parameter
    STR R1, R6, #0

    JSR PRINT_LIST

    ; return
    LDR R0, R6, #0  ; Load return value to R0
    STR R0, R5, #3  ; Store return value from R0 to correct location

    ADD R6, R6, #2

    BR TEARDOWN

DONE
    AND R0, R0, #0
    STR R0, R5, #3

TEARDOWN
     LDR R7, R5, #2  ; Restore R7
     LDR R5, R5, #1  ; Restore R5
     ADD R6, R6, #2  ; Pop stack
     RET
     .END
```

Data file: data.asm

```
1  ; data.asm
2
3      .ORIG x2000
4
5      .FILL x43
6      .FILL x2006
7
8      .FILL x41
9      .FILL x2000
10
11     .FILL x46
12     .FILL x2002
13
14     .FILL x45
15     .FILL x0
16
17     .END
```

C to LC-3 – Assembly Translation with linked data structure

Recursive tree traversal

Problem statement: Convert the following function from C to LC-3. This function recursively traverses a binary tree.

```
void TraverseTree(t_node *nd)
{
    if (nd != NULL)
{
   TraverseTree(nd->left);
   TraverseTree(nd->right);
}
}
```

C to LC-3 – Assembly Translation with linked data structure

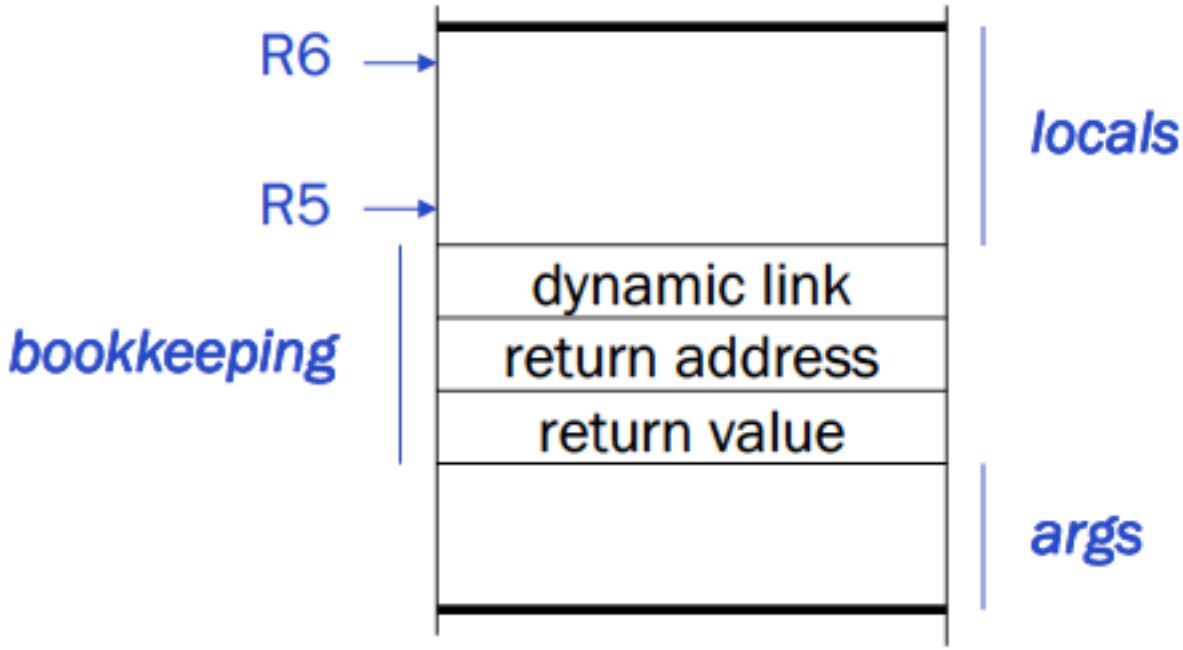
Recursive tree traversal

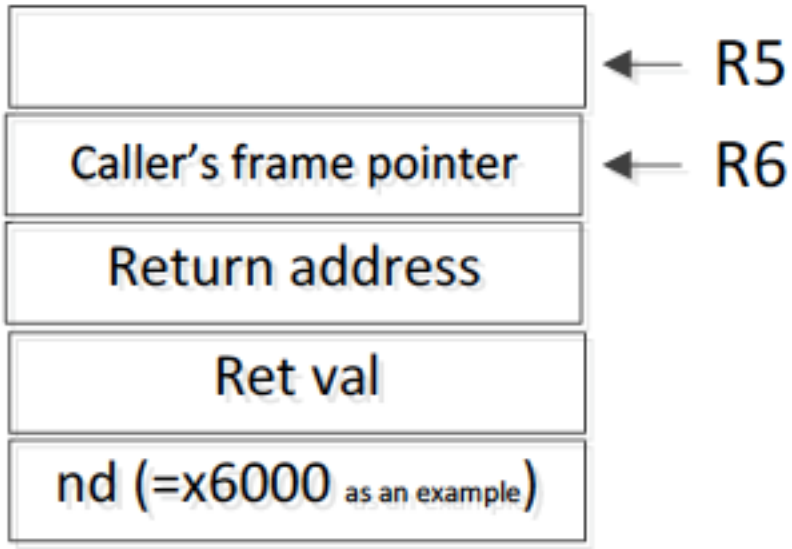
Problem statement: Convert the following function from C to LC-3. This function recursively traverses a binary tree.

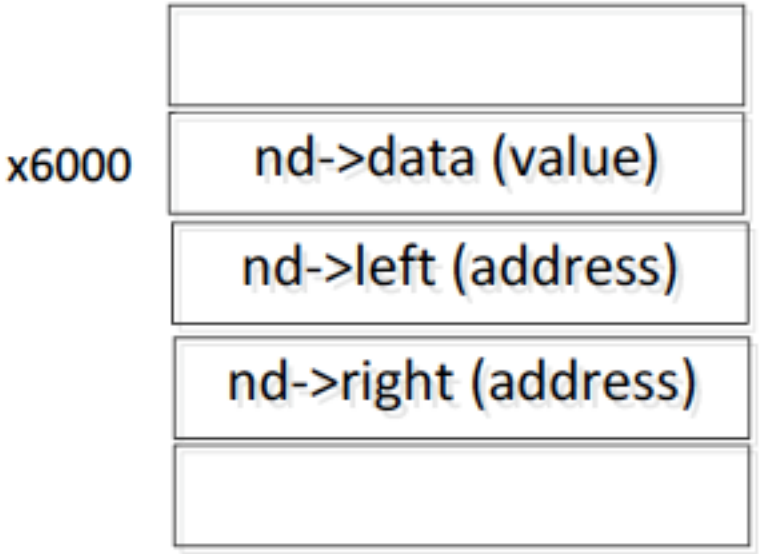
```
void TraverseTree(t_node *nd)
{
   if (nd != NULL)
{
   TraverseTree(nd->left);
   TraverseTree(nd->right);
}
}
```

```
typedef struct nodeTag t_node;
struct nodeTag
{
    int data;
    t_node *left;
    t_node *right;
};
```

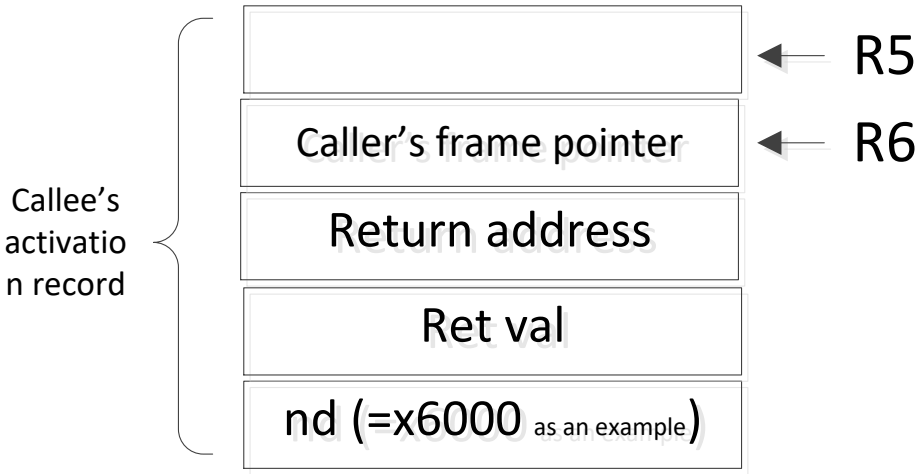
Activation Record

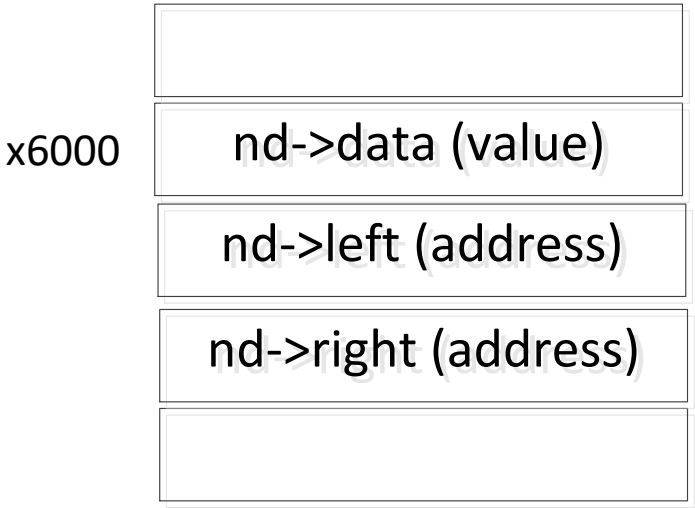






Step#1





TRAVERSE_TREE

; Allocate space for return value

ADD R6, R6, #-1

; Push return address to stack

ADD R6, R6, #-1

STR R7, R6, #0

; Store callee's frame pointer

ADD R6, R6, #-1

STR R5, R6, #0

; Set up new frame pointer

ADD R5, R6, #-1

Step#2: Implement Logic Function

```
; if (nd == NULL), skip to the end
    LDR R0, R5, #4;
    BRz DONE

    ; TraverseTree(nd->left);
LDR R1, R0, #1 ; load nd->left to R1

     ; push nd->left to stack
    ADD R6, R6, #-1
    STR R1, R6, #0

      ; call subroutine
    JSR TRAVERSE_TREE
```

```
DONE
        ; Restore frame pointer
        LDR R5, R6, #0
        ADD R6, R6, #1

        ; Restore return address
        LDR R7, R6, #0
        ADD R6, R6, #1
RET
```

Step#2: Implement Logic Function

```
; if (nd == NULL), skip to the end
    LDR R0, R5, #4;
    BRz DONE

    ; TraverseTree(nd->left);
LDR R1, R0, #1 ; load nd->left to R1

    ; push nd->left to stack
    ADD R6, R6, #-1
    STR R1, R6, #0

    ; call subroutine
    JSR TRAVERSE_TREE
```

```
; tear-down the rest of the stack
    ADD R6, R6, #2
    ; TraverseTree(nd->right);
LDR R0, R5,#4
LDR R2, R0, #2    ; load nd->right to R2
    ; push nd->right to stack
    ADD R6, R6, #-1;
    STR R2, R6, #0;

    ; call subroutine
    JSR TRAVERSE_TREE

    ; tear-down the rest of the stack
    ADD R6, R6, #2
```

Teardown the activation record, return:

```
DONE
        ; Restore frame pointer
        LDR R5, R6, #0
        ADD R6, R6, #1

        ; Restore return address
        LDR R7, R6, #0
        ADD R6, R6, #1
RET
```