```
5  struct node {
6    int ID;
7    struct node * left;
8    struct node * right;
9  };
```

Figure 1: Definition of `struct node`.

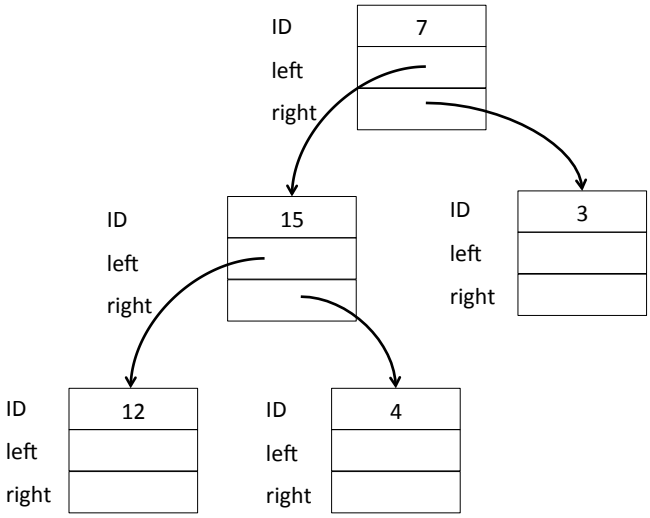

Figure 2: Illustration of a binary tree built with `struct node`.

In this part of the exam we will work with a recursive tree-search code. This program operates on a data structure that is defined as shown in Figure 1. This data structure is used to create a binary tree in memory as illustrated in Figure 2. We are interested in generating the MIPS code for a function called `PathSum` that takes two parameters: a pointer to a node in the binary tree and an integer value `ID`. `PathSum` is a recursive function that returns the sum of the IDs in the path from the node passed as a parameter to `PathSum` to the node identified by the value of the parameter `ID`. For instance, for the binary tree shown in Figure 2, if the first parameter passed to PathSum is a pointer to the node with `ID=7`, and the value of the second parameter is 4, then that invocation of `PathSum` will return 26, which is the sum of the values of the `ID`s in the path from the node 7 to the node 4. An implementation of `PathSum` in C is shown in Figure 3(b). Assume that this code is running in an architecture where all the integer values are represented in 32 bits and all pointers are also represented in 32 bits.

```c
int main(int argc, char *argv[]){
    int i;
    struct node *root;
    struct node NodeBank[NUM_NODES];
    struct node *current;

    for(i=0 ; i< NUM_NODES ; i++){
        current = &(NodeBank[i]);
        current->ID = i;
        if (2*i+1 < NUM_NODES)
            current->left = &(NodeBank[2*i+1]);
        else
            current->left = NULL;
        if (2*i+2 < NUM_NODES)
            current->right = &(NodeBank[2*i+2]);
        else
            current->right = NULL;
        }
  root = &(NodeBank[0]);
  for(i=0 ;i<NUM_NODES ; i++)
      printf("PathSum(%d) = %d\n",i,PathSum(root, i));
}
```

```c
int PathSum(struct node *N, int ID){
  int found;
  int N_ID;

  if(N == NULL)
    return -1;
  N_ID = N->ID;
  if(N_ID == ID)
    return ID;
  found = PathSum(N->left, ID);
  if(found != -1)
    return (N_ID + found);
  found = PathSum(N->right, ID);
  if(found != -1)
    return (N_ID + found);
  return -1;
}
```

Figure 3: (a) main (b) PathSum

Question 4 (20 points): Figure 3(a) shows the code for a main function that invokes `PathSum`. In this part of the question you will write code for one of the lines of the `main` procedure. All the assumptions below refer to the use of registers in `main`.

Assume that the first element of the array `NodeBank` is found at the memory location whose address is `$fp-800`. The value of the local variable `i` is in register `$s0`, the pointer `current` is in register `$s1`. Write assembly code, using a minimum number of MIPS assembly instructions, to execute the statement in line 29 of the `main` procedure in the C implementation.

First we need to compute the address of `&(NodeBank[2*i+2]`. This address is given by:
`NodeBank + (2*i+2)*12 = NodeBank + 24*i + 24 = NodeBank + 24*(i+1)`

Figure 4 shows two solutions for this question. Solution 1 uses the minimal number of instructions, while solution 2 does not.

```
8  ; Solution 1
9      addi    $t0, $fp, -800 ; $t0 <-- &NodeBank
10     li      $t1, 24        ; $t1 <-- 24
11     mul     $t2, $s0, $t1  ; $t2 <-- 24*i
12     addi    $t3, $t2, 24   ; $t3 <-- 24*i + 24
13     add     $t4, $t0, $t3  ; $t3 <-- &(NodeBank[2*i+2])
14     sw      $t4, 8($s1)    ; current->right <-- $t1
15
16 ; Solution 2
17     addi    $t0, $fp, -800 ; $t0 <-- &NodeBank
18     sll     $t1, $s0, 3    ; $t1 <-- 8*i
19     sll     $t2, $s0, 4    ; $t2 <-- 16*i
20     add     $t3, $t1, $t2  ; $t3 <-- 24*i
21     addi    $t4, $t3, 24   ; $t4 <-- 42*i + 24
22     add     $t5, $t0, $t4  ; $t5 <-- &(NodeBank[2*i+2])
23     sw      $t5, 8($s1)    ; current->right <-- $t1
```

Figure 4: Two solutions for the NodeBank question. Solution 1 uses the minimal number of instructions.

Question 5 (30 points): Write the MIPS assembly code for the function `PathSum` whose C code is shown in Figure 3(b). You must follow all the MIPS procedure call conventions. You must map the local variable `N_ID` to register `$s2`. While you are allowed to use SPIM pseudo instructions, you are not allowed to use any instruction that takes a constant that is larger than 16 bits.

```
; $s0: N
; $s1: ID
; $s2: N_ID

PathSum:
    add  $sp, $sp, -16
    sw   $s0, 0($sp)
    sw   $s1, 4($sp)
    sw   $s2, 8($sp)
    sw   $ra, 12($sp)
    bne  $a0, $zero, check_id ; if N != NULL
    li   $v0, -1              ; $v0 <-- -1
    j    done
check_id:
    lw   $s2, 0($a0)          ; N_ID <-- N->ID
    bne  $s2, $a1, go_left    ; if N_ID != ID
    move $v0, $a1             ; $v0 <-- ID
    j    done
go_left:
    move $s1, $a1             ; $s1 <-- ID
    move $s0, $a0             ; $s0 <-- N
    lw   $a0, 4($s0)          ; $a0 <-- N->left
    jal  PathSum              ; PathSum(N->left, ID)
    li   $t1, -1
    bne  $v0, $t1 return      ; if $v0 != -1
    lw   $a0, 8($s0)          ; $a0 <-- N->right
    move $a1, $s1             ; $a1 <-- ID
    jal  PathSum              ; PathSum(N->right, ID)
    li   $t1, -1
    beq  $v0, $t1, done       ; if $v0 == -1
return:
    add  $v0, $v0, $s2        ; $v0 <-- found + N_ID
done:
    lw   $s0, 0($sp)
    lw   $s1, 4($sp)
    lw   $s2, 8($sp)
    lw   $ra, 12($sp)
    add  $sp, 16
    jal  $ra
```