Linked lists in MIPS

A linked list of integers is a sequence of nodes. Each node is an 8-byte chunk of memory. The first 4 bytes represents the int-valued data associated with this node and the next 4 bytes represents the pointer to the next node in the list. The next field is zero (NULL) if there is no next node in the list.

Each node encodes an 8-bit message as a Hamming code in the 12 least significant bits of its data field. We will use such linked lists to “hide” two 8-bit values x and y using the following scheme: All except two nodes in the linked list will have unique values in the 20 most significant bits of their data field. The two nodes with identical values in their 20 most significant bits will contain x and y as Hamming codes in their 12 least significant bits. The node containing x will appear before the node containing y. The 12-bit codes may have up to 1 error each, which will need to be corrected before extracting the 8-bit x and y values.

Given the head node to such a linked list, your task is to determine the values of x and y. The following recursive code does this (inefficiently) in C++. Using mp3.s as a starting point, translate getXY and helper to solve this problem in MIPS.

Note: You should start thinking of a more efficient implementation of getXY, but for this MP translate the given code.

```c++
// The return value of this function is two 8-bit values. When you
// translate into MIPS, be sure to return x in $v0 and y in $v1

getXY(node *head)
{
    node *p = helper(head->next, head->data);
    if(p != NULL)
    {
        int x = extractData(correctError(head->data, 12), 12);
        int y = extractData(correctError(p->data, 12), 12);
        return (x, y); // return x in $v0, y in $v1
    }
    return getXY(head->next);
}

node *helper(node *head, int data)
{
    if(head == NULL)
        return NULL;
    if((head->data & 0xFFFFF000) == (data & 0xFFFFF000))
        return head;
    return helper(head->next, data);
}
```

Getting started

On the EWS machines, give the command `/homesta/cs232/Linux/bin/spimbot` to run the SPIMbot simulator. For debugging, recall from section that you can easily print the value of a register $reg with the command: `sw $reg, 0xFFFF0080($0)`.

IMPORTANT: Be sure to remove any such print statements from your final version before submitting!