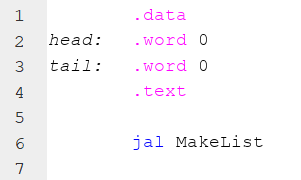
# Lab 8

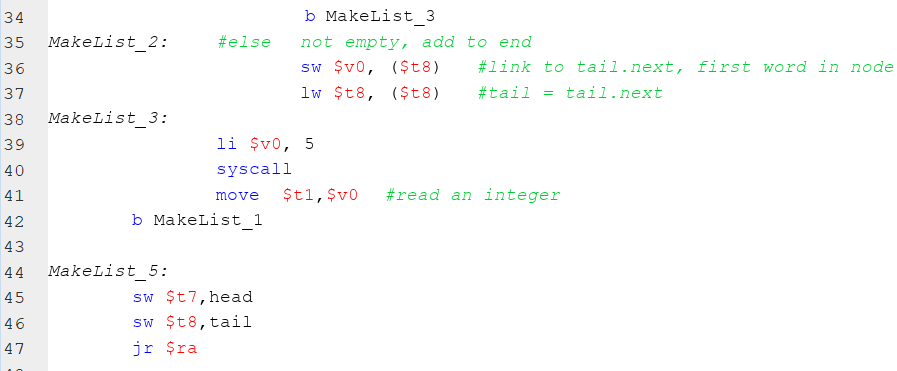
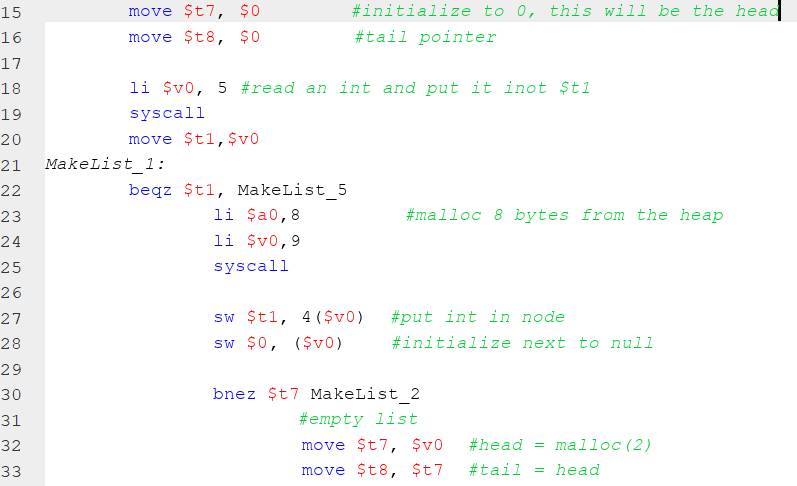
# Dynamic Memory and Recursion

In this lab you will be introducing dynamic memory allocation and integrating this into a recursive procedure call.

Part 1

The basic concept of dynamic memory allocation is the linked list. In general we want to allocate a chunk of memory from the system heap space. In java this was done by the ***new*** directive. In assembler this is done with a system call which amounts to the same thing. Our linked list will have a head and tail pointer

To speed things up, the start of your lab can be downloaded from the course website. The only variables that are declared are a head and tail pointer to the linked list. The MakeList procedure will repeatedly prompt the user for an integer until a ZERO is entered, these integers are then placed into their own node of the linked list. Each node is configured as ***next*** followed by the ***integer***. Let us look at the important parts of MakeList.



Next

Int

T7 & T8 are used as the head and tail pointers while we make the list, these are assigned at the end to head and tail.

Line 18 and 39 prompts the user to enter an integer, the input integer is tested at line 22.

Line 23 is a syscall which will allocate 2 words from the heap space, 2 words is 8 bytes as on line 23. V0 holds the returned address of the allocated memory.

We put the integer into the node (2nd word) Line 27. The ***next*** field is set to null.

30-37 takes care of an empty list and not empty case. These operations are very similar to the Java equivalent.

Part 2

Create a procedure which will traverse the linked list pointed to by head. Think about the Java process:

Set a pointer to what head is pointing to

While not null

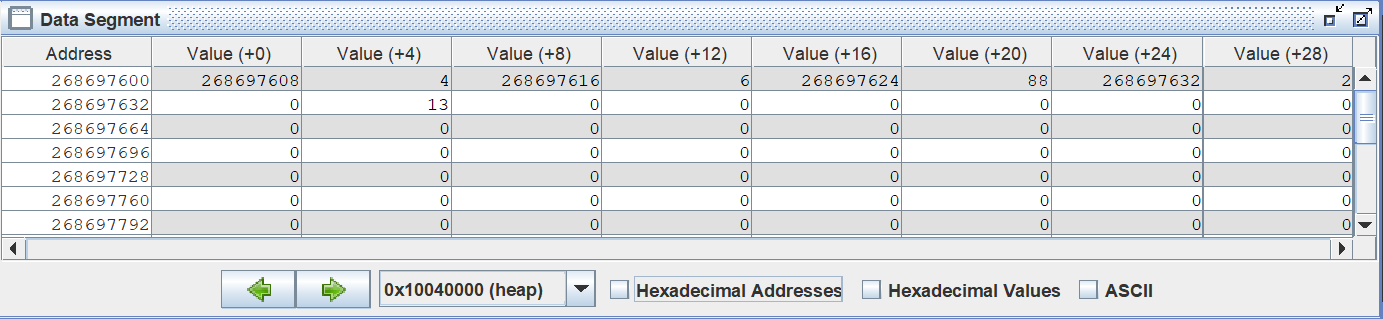
Print the contents

Advance the pointer

Things to consider, head is a variable which holds an address and this address is loaded to a register for the traversal.

The MakeList procedure advances the tail pointer, hint – hint. There is no null in assembler, just a 0 address which serves the same purpose. Null == 0.

When you are finished, take a look at the Heap. You can see each node where the first element is an address and the 2nd the integer payload. Be sure to select heap. You will notice in the .data the ***head*** and ***tail*** pointer addresses matching the first and last node in the heap.



Part 3

Let’s create a procedure to print the list out backward using recursion, here is the pseudo code:

void PrintRev(Node N)

If (N!=null)

PrintRev(N.next)

Write out N

}

}

You will need to set up the AR much like that used in the Fibonacci example. When you are complete, your program should create the list, print it out in forward order, then print it out in reverse order. Much like the pseudo code above, you will need to pass a pointer as a parameter.

Part 4

Try and code up a recursive factorial function. It will follow a similar pattern as the PrintRev procedure.

public static long factorial(int n) {

if (n == 1) return 1;

return n \* factorial(n-1);

}