ITESO

Universidad Jesuita
de Guadalajara

1st Practice:

Hanoi Towers recursively

Raúl Méndez Álvarez

Is685434

Ingeniería en Sistemas Computacionales

6/17/209

Arquitectura computacional

Rodrigo Aldana López

Assembly solution file:


Hanoi bueno - Raúl Méndez.asm

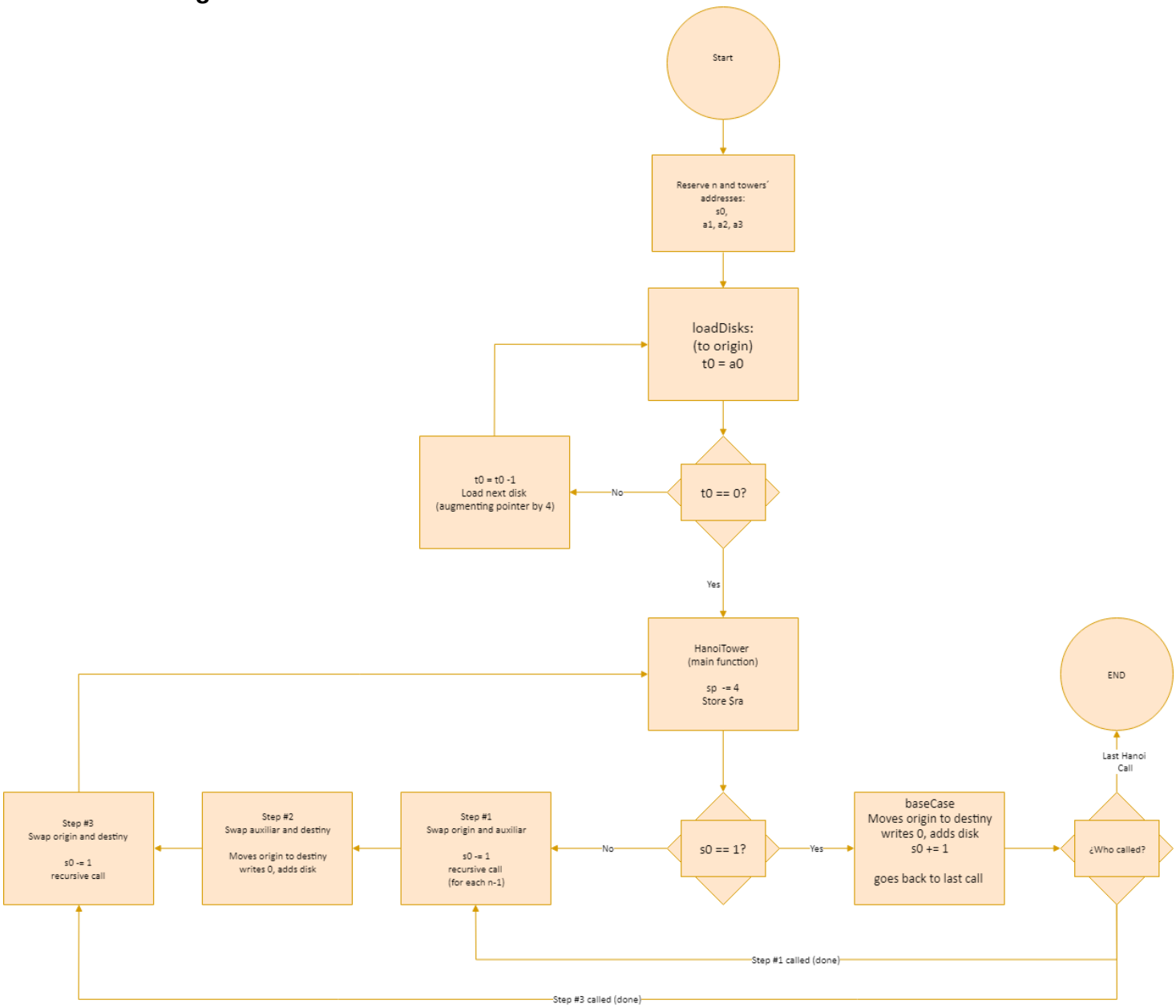

hanoi C.c

```
#Raúl Méndez
#Hanoi Towers recursively

#Final IC: 5414 (5409 shortest)
#R: 1790
#I: 3368
#J: 256
#$sp bytes used: 32


.data
    #Prompt message for user input and finish message
    promptM: .asciiz "Enter the number of disks (n): " #asciiz implies a
string; adds \n at the end
    doneM: .asciiz "\nDone!"

.text

    #Addresses shwon on the Data Segment visualizer in MARS
    addi $a1, $zero, 0x10010000 #Origin or A tower
    addi $a2, $zero, 0x10010020 #Auxiliar or B tower
    addi $a3, $zero, 0x10010040 #Destiny or C tower

    #Read n value from user
    li $v0, 4 #4 signals printing
    la $a0, promptM #Prints "prompt" var in data (message)
    syscall #Console call

    li $v0, 5 #5 signals reading keyboard input
    syscall

    add $s0, $v0, $zero    #Stores input in s0: s0 = n
    add $t0, $s0, $zero     #Stores s0 in t0 in order for the functs to work
with it (as temp var 0)
    add $t1, $zero, $zero   #Temporal value 1
    add $t2, $zero, $zero   #Temporal value 2
    #(side note/comment: addi with 0 vs add with $zero seemed to have no
impact on the final IC)

loadDisks: #Loads the n-1 disks onto aux tower
```

```
    sw $t0, 0($a1)          #Adds curent disk onto origin tower
    addi $t0, $t0, -1       #Disks are now n-1

    addi $a1, $a1, 4        #Increments a1´s "pointer" to the next address
(next space in origin tower)
    bne $t0, 0, loadDisks   #Loop: loads remaining disks until t0 = 0

    jal HanoiTower          #Copies current address to $ra and jumps to
HanoiTower
    j done                  #End of progam funct

HanoiTower: #excecute
    addi $sp, $sp, -4       #Takes 32 bits from sp = reserves 32 bits for $ra
(for recursion)
    sw $ra, 0($sp)          #Store $ra

    #If equivalent: case case requirement if equivalent:
    beq $s0, 1, baseCase    #n = 1: goes to base case, else: continues with
the code
    #-----------------------------------------------------------------

#Step 1: origin-aux swap
    #Preparing data before next HanoiTower call
    addi $s0, $s0, -1       #n-1 (for s0 as a loop var)
    add $t1, $a2, $zero     #Saves aux in a temp var
    add $a2, $a3, $zero     #Swaps origin and aux disks
    add $a3, $t1, $zero
    #Swaps values in order to take n-1 disks from origin to aux tower;
    #a hanoiTower call for each n-1 disk

    jal HanoiTower          #Recursive call

#Step 2: auxiliar-destiny swap
    #moving disks
    add $t0, $a3, $zero     #Saves destiny in temp var
    add $a3, $a2, $zero     #Swaps auxiliar and destiny
    add $a2, $t0, $zero

    #Moving origin to destiny
    addi $a1, $a1, -4   #Takes the disk from destiny
    lw $t3, 0($a1)      #Loads origin to temp var
        sw $zero, 0($a1)    #Writes a 0 in disk´s past place, before moving it
to destiny
        sw $t3, 0($a3)      #Saves address from origin to destiny
        addi $a3, $a3, 4    #Adds disk to next place in destiny tower

#Step 3: origin-destiny swap
```

```
    add $t1, $a1, $zero     #Save origin to a temporary variable
    add $a1, $a2, $zero     #Swap origin and destiny
    add $a2, $t1, $zero

    addi $s0, $s0, -1       #n-1
    jal HanoiTower      #Recursive call

    #Recovering swaps to initial tower values
    add $t1, $a1, $zero
    add $a1, $a2, $zero
    add $a2, $t1, $zero

    lw $ra, 0($sp)          #Loads $ra back
    addi $sp, $sp, 4        #Gets the 32 bits back to sp

    addi $s0, $s0, 1        #We add the disk to n that was substracted before:
n+1
    jr $ra              #Goes back to last HanoiTower call: "rewinding"
process begins

baseCase:#BaseCase: Moves origin to destiny
    #Moving origin to destiny
    addi $a1, $a1, -4       #Takes disk from destiny
    lw $t3, 0($a1)          #Loads origin to temp var
        sw $zero, 0($a1)        #Writes a 0 in disk´s past place, before
moving it to destiny
        sw $t3, 0($a3)          #Saves address from origin to destiny
        addi $a3, $a3, 4        #Adds disk to next place in destiny tower

    addi $s0, $s0, 1        #Adds the disk back to n

    lw $ra, 0($sp)          #Loads $ra back
    addi $sp, $sp, 4        #Gets the 32 bits back to sp
    jr $ra                  #Go back to last HanoiTower Call

done:
    #Displays finish message
    li $v0, 4 #Will print
    la $a0, doneM #Finish message
    syscall

    #End of Program equivalent
    li $v0, 10 #Signals success return
    syscall
```

Flow diagram:



Note: feel free to look at the .ZIP for the full sized image

Flow description:

My algorithm would begin by associating each of the addresses to each of its respective tower as well as register. We´d also always have n stored in s0, even though such variable is inputed by the user beforehand.

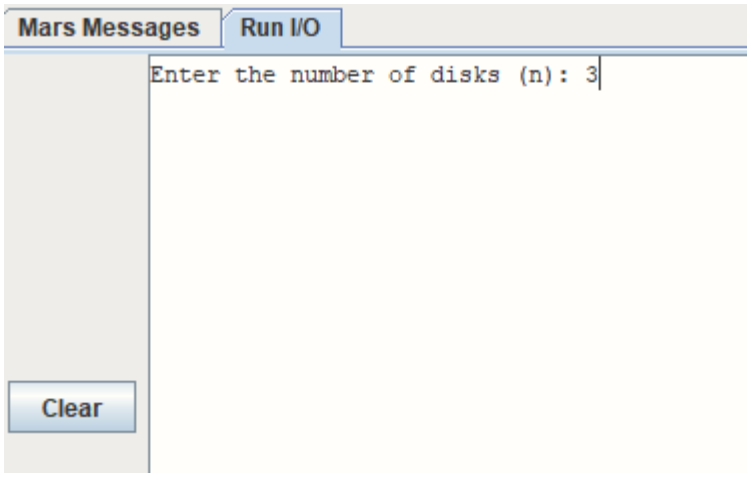
The loadDisks function or subroutine would then begin to load each and every disk onto its respective space or place on tower A/origin tower until the amount of disks are complete. And then it would proceed to call the HanoiTower function for the first time, which would initially allocate the memory necessary to store the ra registry onto the top stack position.

The function would first try to validate whether the amount of disks is or isn´t 1 in order to jump to the base case (which would move the respective tower to its according position) or to the internal function steps (which would swap the pointers or addresses on each of the towers accordingly in order to follow the algorithm described on the assignment).

The process would loop in and out of the steps, unwinding and then rewinding when its ready to swap the disks to their respective next tower, step by step, all by switching the auxiliary tower accordingly. Once the algorithms takes us to the last step or to the top of the rewinding process, it would put the last disk onto its final place and finally it would restore the size of the sp as well as s0 to finally be done with the process.

MARS simulation for n=3:

First, the user would be prompted to input the amount of disks using the keyboard:



We can see the initial values allocated in each address, which are associated with registers a1, a3 and a3:

	Value (+0)	Value (+4)	Value (+8)
0x10010000	1702129221	1752440946	1970151525
0x10010020	1852785674	8549	0
0x10010040	0	0	0
0x10010060	0	0	0

The program begins to load the 3 disks into tower A or origin tower (in a1):

	Value (+0)	Value (+4)	Value (+8)
0x10010000	3	1752440946	1970151525
0x10010020	1852785674	8549	0
0x10010040	0	0	0
0x10010060	0	0	0
0x10010080	0	0	0

	Value (+0)	Value (+4)	Value (+8)
0x10010000	3	2	1970151525
0x10010020	1852785674	8549	0
0x10010040	0	0	0
0x10010060	0	0	0
0x10010080	0	0	0

	Value (+0)	Value (+4)	Value (+8)
0x10010000	3	2	1
0x10010020	1852785674	8549	0
0x10010040	0	0	0

The program begins to swap each of the disks accordingly:

	Value (+0)	Value (+4)	Value (+8)
0x10010000	3	2	0
0x10010020	1852785674	8549	0
0x10010040	1	0	0

	Value (+0)	Value (+4)	Value (+8)
0x10010000	3	0	0
0x10010020	2	8549	0
0x10010040	1	0	0

	Value (+0)	Value (+4)	Value (+8)
0x10010000	3	0	0
0x10010020	2	1	0
0x10010040	0	0	0

	Value (+0)	Value (+4)	Value (+8)
0x10010000	0	0	0
0x10010020	2	1	0
0x10010040	3	0	0

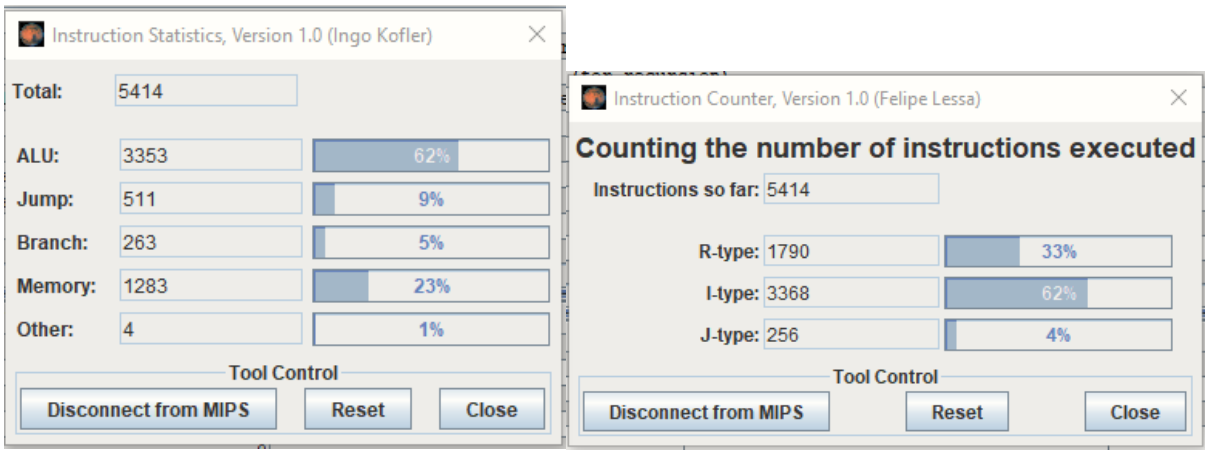
	Value (+0)	Value (+4)	Value (+8)
0x10010000	1	0	0
0x10010020	2	0	0
0x10010040	3	0	0

	Value (+0)	Value (+4)	Value (+8)
0x10010000	1	0	0
0x10010020	0	0	0
0x10010040	3	2	0

Finally, we´ve got the end result:

	Value (+0)	Value (+4)	Value (+8)
0x10010000	0	0	0
0x10010020	0	0	0
0x10010040	3	2	1

IC & statistics



Worst case stack use: 32 bytes

Final program size: 4,235 bytes

Conclusions:

Just as I initially expected, the sole idea of just coming to terms with the fact that I was about to build an assembly app on my own -without a team/partner- would already be stressing the crap out of me.

And yeah, I know; it´s just an assignment, and just the first one for that matter, but the matter of fact here is that I actually don´t consider or see myself as a good programmer: when it comes to the idea of creating solutions to a certain interesting problem, I get pretty excited – but when it comes to getting those ideas back from the skies of my imagination, yeah, it gets quite the opposite way around. Let´s just say it´s not a comfortable feeling to just watch the clock fly away when you´ve still done crap.

Recursion as a whole has always been confusing to me: from my first approaches when taking those intro to Computer Science courses to my last ones with those Algorithm ones; it gets pretty confusing to me when it comes to imagining how the hell is that some algorithm streamlines or branches onto multiple complicated processes. Divide and Conquer as a solving approach, for example, sometimes tinkers with my sanity…

At first, I couldn't find a logical way of visualizing the original provided algorithm as a whole, or basically as any piece of code by itself, so I had a bit of trouble with writing it first and then *translating* it into assembly code.

When it came to optimize the final IC, I actually had a pretty fun time, trying to figure out the *instruction count cost* or repercussions for each of the instructions as well as pseudo-instructions whilst keeping in mind the fact that all of them would be in a certain amount of internal recursive loops, and therefore would grow on a certain exponential manner.

It actually was impressing to see how modifying or changing instructions so simple as addi vs add or sub vs addi with a negative would impact so much on my final IC; all of this took my final count to a quite low number when compared to the initial 7000´s.

Git repo:

https://github.com/RaulMendezA/HanoiTowersRecursively.git