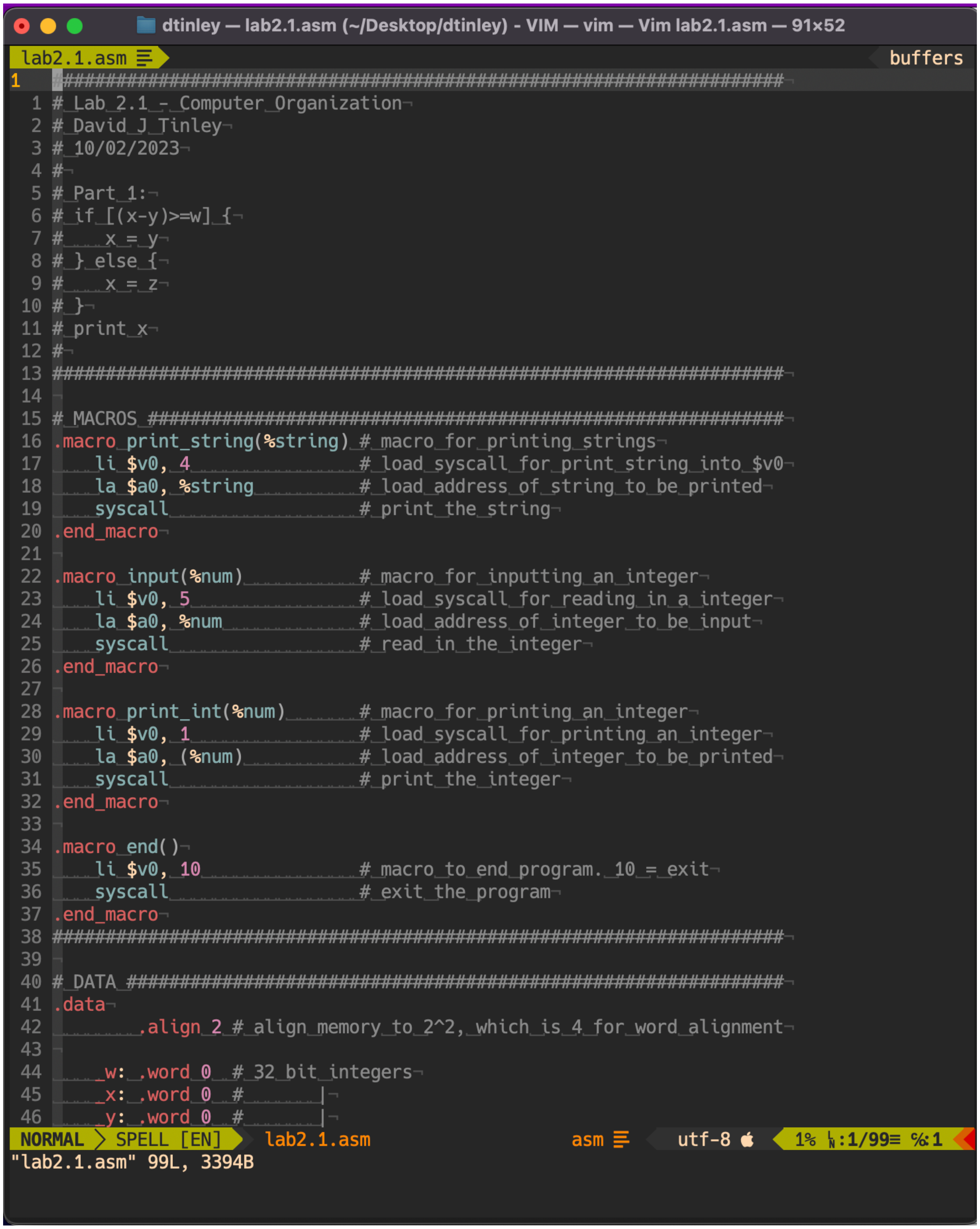
David J Tinley
10/03/2023
Lab 2.1 Report
Computer Organization

For Lab 2.1 I was able to reuse many of the macro's I had written for the previous labs.



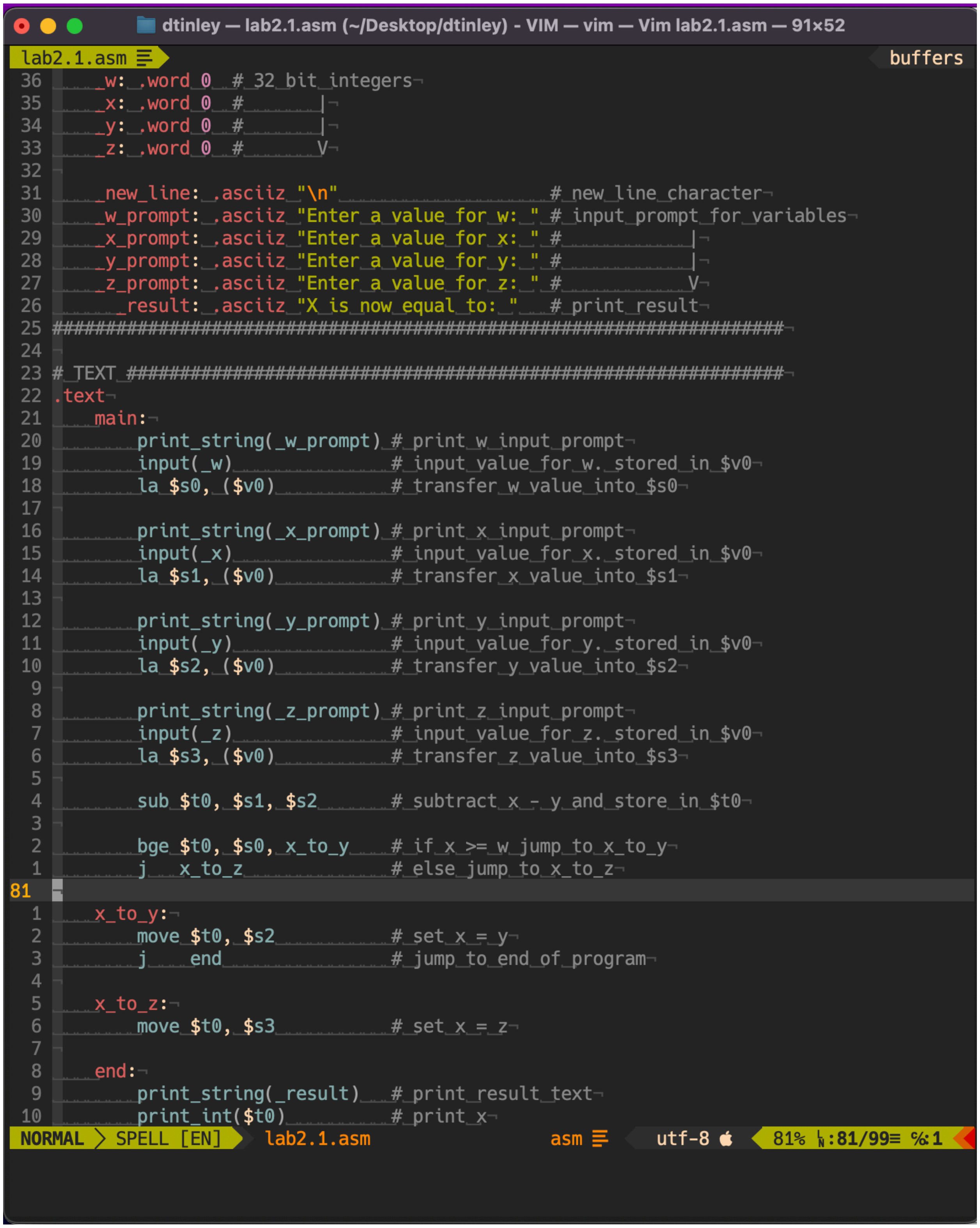
dtinley — lab2.1.asm (~/Desktop/dtinley) - VIM — vim — Vim lab2.1.asm — 91×52

lab2.1.asm   buffers

```
########################################################################
# Lab 2.1 - Computer Organization
# David J Tinley
# 10/02/2023
#
# Part 1:
# if [(x-y)>=w] {
#     x = y
# } else {
#     x = z
# }
# print x
#
########################################################################

# MACROS ###############################################################
.macro print_string(%string) # macro for printing strings
    li $v0, 4                # load syscall for print string into $v0
    la $a0, %string          # load address of string to be printed
    syscall                  # print the string
.end_macro

.macro input(%num)           # macro for inputting an integer
    li $v0, 5                # load syscall for reading in a integer
    la $a0, %num             # load address of integer to be input
    syscall                  # read in the integer
.end_macro

.macro print_int(%num)       # macro for printing an integer
    li $v0, 1                # load syscall for printing an integer
    la $a0, (%num)           # load address of integer to be printed
    syscall                  # print the integer
.end_macro

.macro end()
    li $v0, 10               # macro to end program. 10 = exit
    syscall                  # exit the program
.end_macro
########################################################################

# DATA #################################################################
.data
        .align 2 # align memory to 2^2, which is 4 for word alignment

    _w: .word 0  # 32 bit integers
    _x: .word 0  #       |
    _y: .word 0  #       |
```

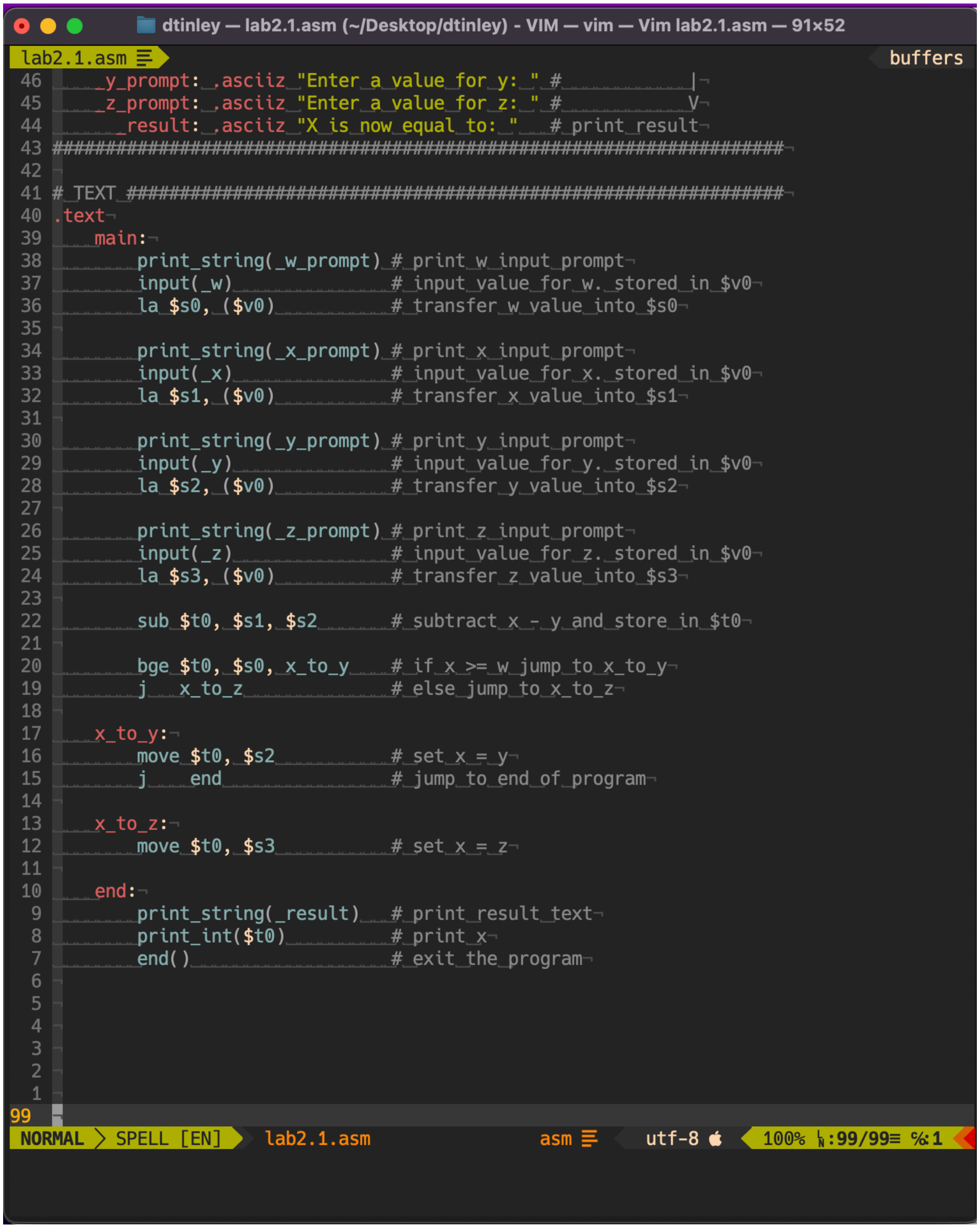
NORMAL SPELL [EN] lab2.1.asm   asm utf-8   1% :1/99≡ %:1
"lab2.1.asm" 99L, 3394B

Also, in the picture above, I started declaring the data for the program and aligning it to the size of word memory.


dtinley — lab2.1.asm (~/Desktop/dtinley) - VIM — vim — Vim lab2.1.asm — 91×52

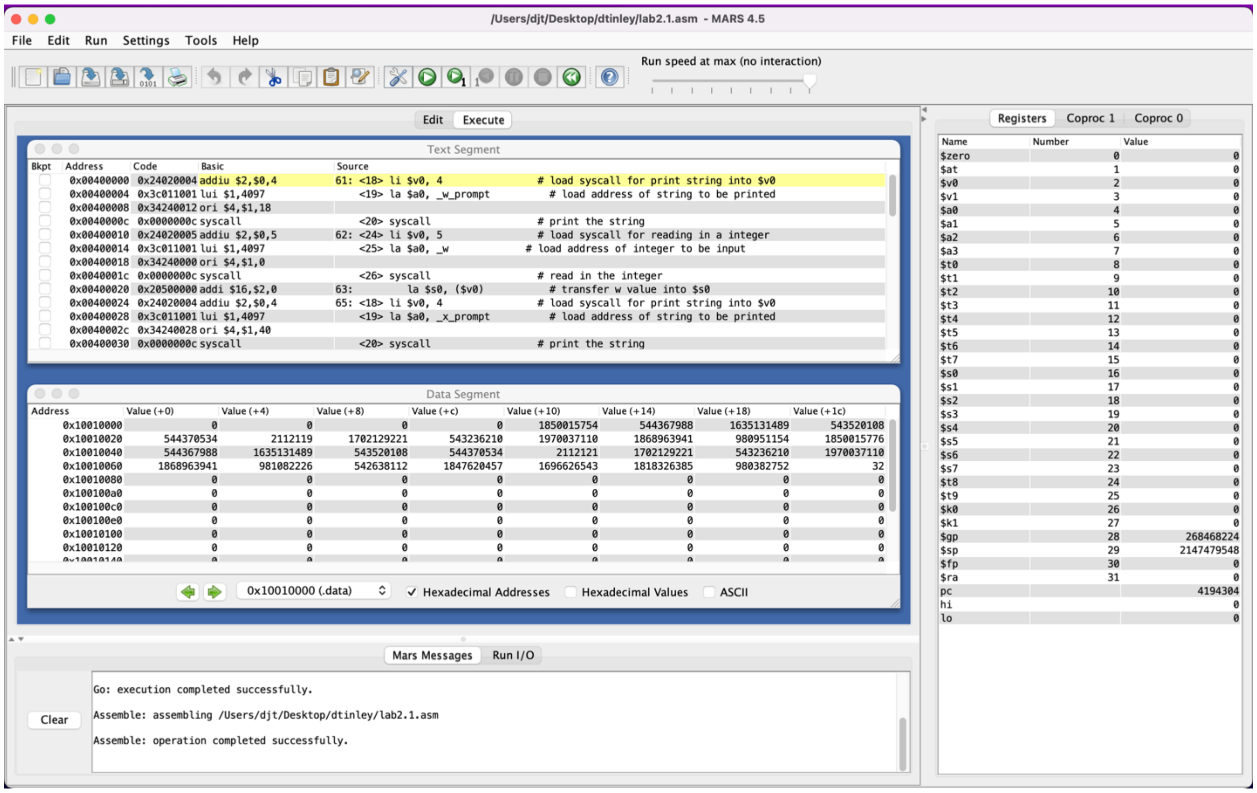
```
lab2.1.asm                                                              buffers
    _w: .word 0  # 32 bit integers
    _x: .word 0  #        |
    _y: .word 0  #        |
    _z: .word 0  #        V

    _new_line: .asciiz "\n"                    # new line character
    _w_prompt: .asciiz "Enter a value for w: " # input prompt for variables
    _x_prompt: .asciiz "Enter a value for x: " #            |
    _y_prompt: .asciiz "Enter a value for y: " #            |
    _z_prompt: .asciiz "Enter a value for z: " #            V
      _result: .asciiz "X is now equal to: "   # print result
######################################################################

# TEXT ###############################################################
.text
    main:
        print_string(_w_prompt) # print w input prompt
        input(_w)               # input value for w. stored in $v0
        la $s0, ($v0)           # transfer w value into $s0

        print_string(_x_prompt) # print x input prompt
        input(_x)               # input value for x. stored in $v0
        la $s1, ($v0)           # transfer x value into $s1

        print_string(_y_prompt) # print y input prompt
        input(_y)               # input value for y. stored in $v0
        la $s2, ($v0)           # transfer y value into $s2

        print_string(_z_prompt) # print z input prompt
        input(_z)               # input value for z. stored in $v0
        la $s3, ($v0)           # transfer z value into $s3

        sub $t0, $s1, $s2       # subtract x - y and store in $t0

        bge $t0, $s0, x_to_y    # if x >= w jump to x_to_y
        j   x_to_z              # else jump to x_to_z


    x_to_y:
        move $t0, $s2           # set x = y
        j    end                # jump to end of program

    x_to_z:
        move $t0, $s3           # set x = z

    end:
        print_string(_result)   # print result text
        print_int($t0)          # print x
NORMAL > SPELL [EN] > lab2.1.asm          asm    utf-8    81% :81/99≡ %:1
```

The screenshot above is the main body of my assembly code. In the data section I also had made the text prompts and text output in this section. Next in the text section I display the prompts one at a time for the user to enter a value for each variable. After each variable is entered and stored in $v0, I transfer it to one of the s registers since $v0 will be overwritten in the next call. After the prompt, input, and transfer calls I subtract the x value from the y value and store the result in $t0.


dtinley — lab2.1.asm (~/Desktop/dtinley) - VIM — vim — Vim lab2.1.asm — 91×52
lab2.1.asm
buffers
NORMAL
SPELL [EN]
lab2.1.asm
asm
utf-8
100% :99/99≡ %:1

```
    _y_prompt:  .asciiz "Enter a value for y: " #            |
    _z_prompt:  .asciiz "Enter a value for z: " #            V
      _result:  .asciiz "X is now equal to: "   # print result
####################################################################

# TEXT #############################################################
.text
    main:
        print_string(_w_prompt) # print w input prompt
        input(_w)               # input value for w. stored in $v0
        la $s0, ($v0)           # transfer w value into $s0

        print_string(_x_prompt) # print x input prompt
        input(_x)               # input value for x. stored in $v0
        la $s1, ($v0)           # transfer x value into $s1

        print_string(_y_prompt) # print y input prompt
        input(_y)               # input value for y. stored in $v0
        la $s2, ($v0)           # transfer y value into $s2

        print_string(_z_prompt) # print z input prompt
        input(_z)               # input value for z. stored in $v0
        la $s3, ($v0)           # transfer z value into $s3

        sub $t0, $s1, $s2       # subtract x - y and store in $t0

        bge $t0, $s0, x_to_y    # if x >= w jump to x_to_y
        j   x_to_z              # else jump to x_to_z

    x_to_y:
        move $t0, $s2           # set x = y
        j   end                 # jump to end of program

    x_to_z:
        move $t0, $s3           # set x = z

    end:
        print_string(_result)   # print result text
        print_int($t0)          # print x
        end()                   # exit the program
```

Continuing in the screenshot above, I use the branch if greater than or equal to operator to compare the new value of x to the value in w. If x is greater than or equal to w, the program jumps to the section x_to_y, where x is set to the value of y. It then jumps to section called end, where the new value of x is printed to the terminal and the program executes the end macro that I made, which just calls the syscall 10 for exit. If x is less than w the program jumps to the section x_to_z where it is assigned the value of z. It then jumps to the end section and executes the same processes described above.

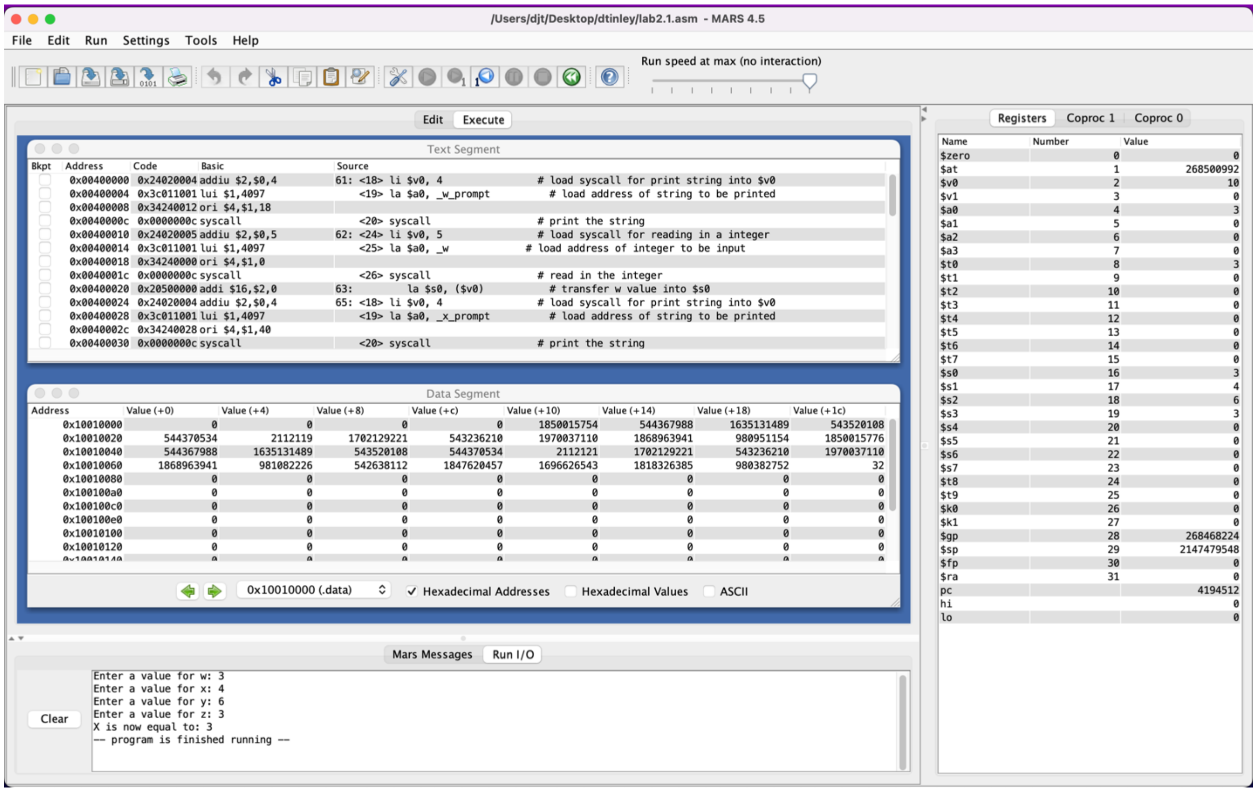

/Users/djt/Desktop/dtinley/lab2.1.asm - MARS 4.5
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
Text Segment

```
Bkpt Address    Code       Basic           Source
     0x00400000 0x24020004 addiu $2,$0,4   61: <18> li $v0, 4                # load syscall for print string into $v0
     0x00400004 0x3c011001 lui $1,4097         <19> la $a0, _w_prompt          # load address of string to be printed
     0x00400008 0x34240012 ori $4,$1,18
     0x0040000c 0x0000000c syscall             <20> syscall                  # print the string
     0x00400010 0x24020005 addiu $2,$0,5   62: <24> li $v0, 5                # load syscall for reading in a integer
     0x00400014 0x3c011001 lui $1,4097         <25> la $a0, _w             # load address of integer to be input
     0x00400018 0x34240000 ori $4,$1,0
     0x0040001c 0x0000000c syscall             <26> syscall                  # read in the integer
     0x00400020 0x20500000 addi $16,$2,0   63:          la $s0, ($v0)          # transfer w value into $s0
     0x00400024 0x24020004 addiu $2,$0,4   65: <18> li $v0, 4                # load syscall for print string into $v0
     0x00400028 0x3c011001 lui $1,4097         <19> la $a0, _x_prompt          # load address of string to be printed
     0x0040002c 0x34240028 ori $4,$1,40
     0x00400030 0x0000000c syscall             <20> syscall                  # print the string
```

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0	0	0	0	1850015754	544367988	1635131489	543520108
0x10010020	544370534	2112119	1702129221	543236210	1970037110	1868963941	980951154	1850015776
0x10010040	544367988	1635131489	543520108	544370534	2112121	1702129221	543236210	1970037110
0x10010060	1868963941	981082226	542638112	1847620457	1696626543	1818326385	980382752	32
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0
0x10010100	0	0	0	0	0	0	0	0
0x10010120	0	0	0	0	0	0	0	0

```
Go: execution completed successfully.

Assemble: assembling /Users/djt/Desktop/dtinley/lab2.1.asm

Assemble: operation completed successfully.
```

Name	Number	Value
$zero	0	0
$at	1	0
$v0	2	0
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	0
$t1	9	0
$t2	10	0
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	0
$s1	17	0
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	0
pc		4194304
hi		0
lo		0


/Users/djt/Desktop/dtinley/lab2.1.asm - MARS 4.5
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
Text Segment

```
Bkpt Address    Code       Basic           Source
     0x00400000 0x24020004 addiu $2,$0,4   61: <18> li $v0, 4                # load syscall for print string into $v0
     0x00400004 0x3c011001 lui $1,4097         <19> la $a0, _w_prompt          # load address of string to be printed
     0x00400008 0x34240012 ori $4,$1,18
     0x0040000c 0x0000000c syscall             <20> syscall                  # print the string
     0x00400010 0x24020005 addiu $2,$0,5   62: <24> li $v0, 5                # load syscall for reading in a integer
     0x00400014 0x3c011001 lui $1,4097         <25> la $a0, _w             # load address of integer to be input
     0x00400018 0x34240000 ori $4,$1,0
     0x0040001c 0x0000000c syscall             <26> syscall                  # read in the integer
     0x00400020 0x20500000 addi $16,$2,0   63:          la $s0, ($v0)          # transfer w value into $s0
     0x00400024 0x24020004 addiu $2,$0,4   65: <18> li $v0, 4                # load syscall for print string into $v0
     0x00400028 0x3c011001 lui $1,4097         <19> la $a0, _x_prompt          # load address of string to be printed
     0x0040002c 0x34240028 ori $4,$1,40
     0x00400030 0x0000000c syscall             <20> syscall                  # print the string
```

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0	0	0	0	1850015754	544367988	1635131489	543520108
0x10010020	544370534	2112119	1702129221	543236210	1970037110	1868963941	980951154	1850015776
0x10010040	544367988	1635131489	543520108	544370534	2112121	1702129221	543236210	1970037110
0x10010060	1868963941	981082226	542638112	1847620457	1696626543	1818326385	980382752	32
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0
0x10010100	0	0	0	0	0	0	0	0
0x10010120	0	0	0	0	0	0	0	0

```
Enter a value for w: 3
Enter a value for x: 4
Enter a value for y: 6
Enter a value for z: 3
X is now equal to: 3
-- program is finished running --
```

Name	Number	Value
$zero	0	0
$at	1	268500992
$v0	2	10
$v1	3	0
$a0	4	3
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	3
$t1	9	0
$t2	10	0
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	3
$s1	17	4
$s2	18	6
$s3	19	3
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	0
pc		4194512
hi		0
lo		0

In conclusion I learned a lot about how values and addresses are stored in the registers and memory. I had a hard time getting the memory alignment to work when using my swap macro for the program, but I eventually figured it out.