NUST CHIP DESIGN CENTRE

Digital Design Verification

Lab Manual # 15 – Data Transfer, Decision Making, Logical Ops
(Arrays in RISC-V Assembly, Bit Manipulation and Implementation of factorial function)

Name	*Khalil Rehman*
Instructor	*Dr Imran*
Date	*17th July 2025*

1. Lab Tasks:

Task # 1

1. 0x10008058, This was exactly 40 bytes (10 elements) past the heap start, showing we wrote past allocated memory.
2. 4, From the error message, a word-sized sw operation.
3. 10, (The loop incorrectly allowed index 10 for a 0-9 array.
4. 0x10008058, (Calculated as: malloc_base + 40 = one word past allocation).
5. 40, From li a0 40 before malloc).
6. sw x0 0(t0), (When t0 pointed to 0x10008058)
7. 40, (The entire array wasn't freed before exit)
8. 0x10008000, (Typical heap start address in Venus)
9. Program exits successfully with:, All 10 array elements set to 0, No memory access errors

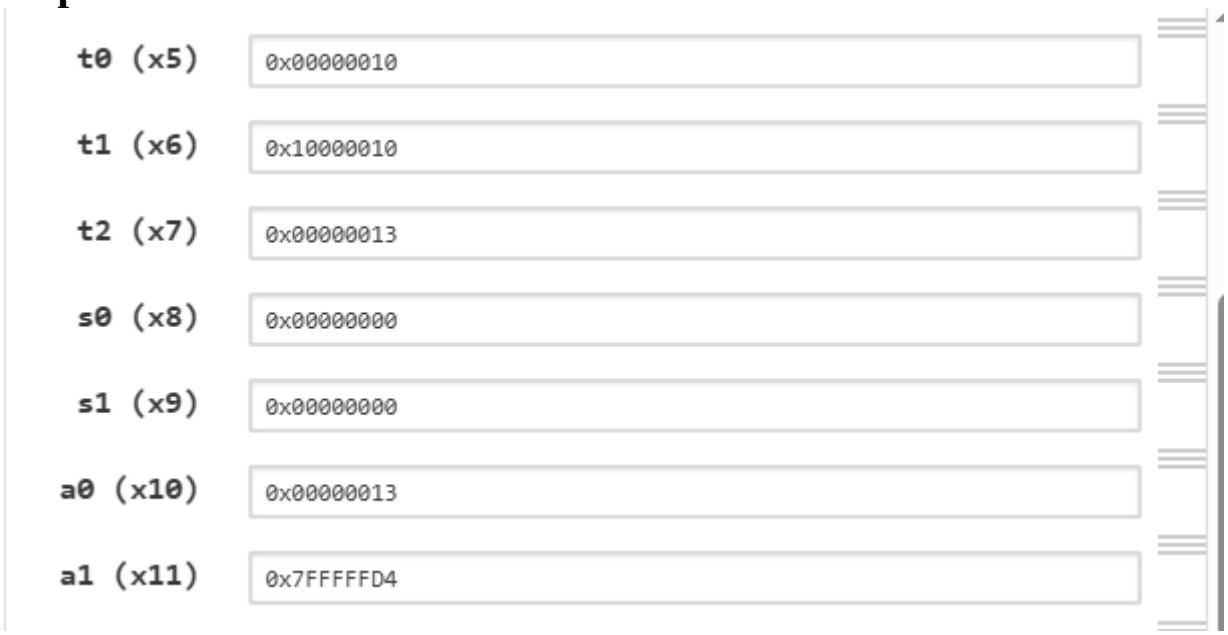
10. Loop condition check before write (bge t1 t2 end_loop), Ensures t1 stays 0-9, pointer t0 stays within 0x10008000-0x10008054
11. Memcheck adds overhead and is only needed when debugging memory issues.

Task # 2

```
.globl f

f:
    # Input: a0 = x value (-3 to 3),  a1 = pointer to output array
    # Returns: f(x) in a0

    # Add 3 to convert input range (-3 to 3) to array indices (0 to 6)
    addi t0, a0, 3

    # Multiply index by 4 (size of each element) using shift
    slli t0, t0, 2

    # Load base address of function values array
    la t1, function_values

    # Calculate address of the function value
    add t1, t1, t0

    # Load the function value
    lw t2, 0(t1)

    # Store the result in output array
    sw t2, 0(a1)

    # Set return value
    mv a0, t2

    # Return
    jr ra

```

```
28      # Return
29      jr ra
30
31  # Predefined function values
32  .data
33  function_values:
34      .word  6     # f(-3)
35      .word 61     # f(-2)
36      .word 17     # f(-1)
37      .word -38    # f(0)
38      .word 19     # f(1)
39      .word 42     # f(2)
40      .word 5      # f(3)
```

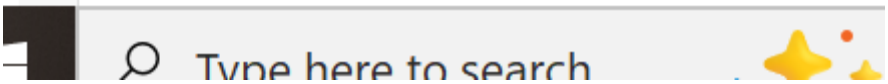
Output:



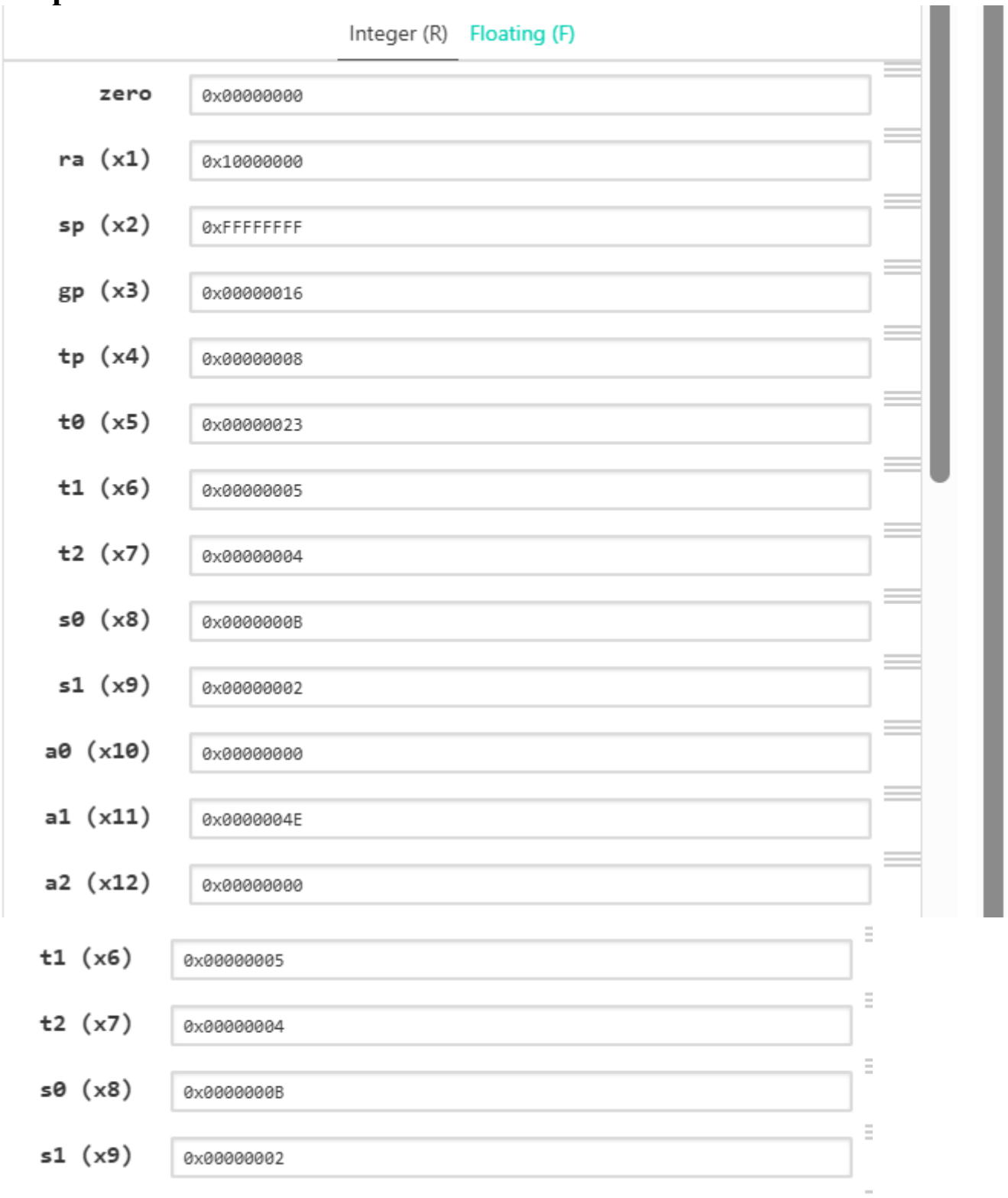
Task # 3

```
.data
array: .word -1, 22, 8, 35, 5, 4, 11, 2, 1, 78

.text
main:
    # Load array address into x1 (not x0)
    la x1, array

    # Load all elements into registers
    lw x2, 0(x1)   # x2 = -1 (array[0])
    lw x3, 4(x1)   # x3 = 22 (array[1])
    lw x4, 8(x1)   # x4 = 8  (pivot)
    lw x5, 12(x1)  # x5 = 35 (array[3])
    lw x6, 16(x1)  # x6 = 5  (array[4])
    lw x7, 20(x1)  # x7 = 4  (array[5])
    lw x8, 24(x1)  # x8 = 11 (array[6])
    lw x9, 28(x1)  # x9 = 2  (array[7])
    lw x10, 32(x1) # x10= 1  (array[8])
    lw x11, 36(x1) # x11= 78 (array[9])

    # Store back in partitioned order
    sw x2, 0(x1)   # -1 stays
    sw x10, 4(x1)  # 1 moved left
    sw x9, 8(x1)   # 2 moved left
    sw x7, 12(x1)  # 4 moved left
    sw x6, 16(x1)  # 5 moved left
    sw x4, 20(x1)  # pivot (8) in middle
    sw x8, 24(x1)  # 11 right
    sw x3, 28(x1)  # 22 right
    sw x5, 32(x1)  # 35 right
    sw x11, 36(x1) # 78 right
```

```
20
21      # Store back in partitioned order
22      sw x2, 0(x1)    # -1 stays
23      sw x10, 4(x1)   # 1 moved left
24      sw x9, 8(x1)    # 2 moved left
25      sw x7, 12(x1)   # 4 moved left
26      sw x6, 16(x1)   # 5 moved left
27      sw x4, 20(x1)   # pivot (8) in middle
28      sw x8, 24(x1)   # 11 right
29      sw x3, 28(x1)   # 22 right
30      sw x5, 32(x1)   # 35 right
31      sw x11, 36(x1) # 78 right
32
33      # Exit properly
34      li a0, 0        # Exit code 0
35      li a7, 93       # Exit syscall number
36      ecall
```



Output:

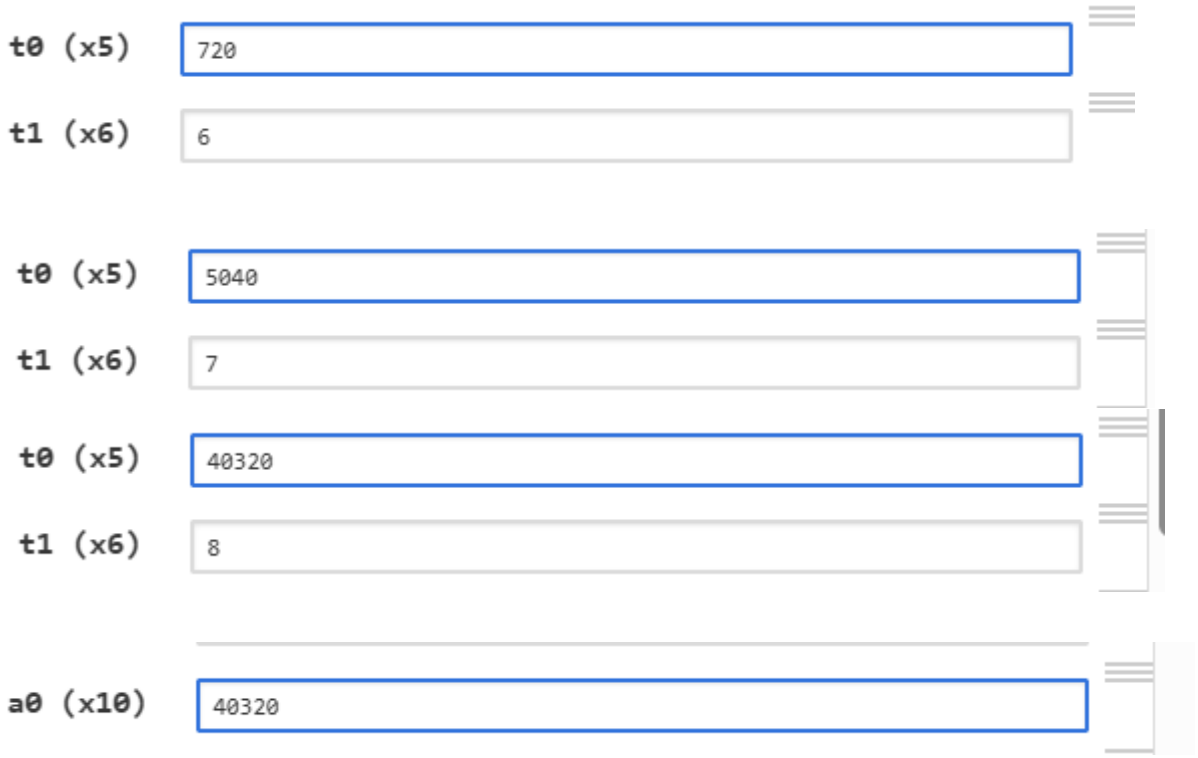

Integer (R)
Floating (F)
zero 0x00000000
ra (x1) 0x10000000
sp (x2) 0xFFFFFFFF
gp (x3) 0x00000016
tp (x4) 0x00000008
t0 (x5) 0x00000023
t1 (x6) 0x00000005
t2 (x7) 0x00000004
s0 (x8) 0x0000000B
s1 (x9) 0x00000002
a0 (x10) 0x00000000
a1 (x11) 0x0000004E
a2 (x12) 0x00000000
t1 (x6) 0x00000005
t2 (x7) 0x00000004
s0 (x8) 0x0000000B
s1 (x9) 0x00000002

Task # 4

```
.globl factorial

.data
n: .word 8

.text
main:
    la t0, n
    lw a0, 0(t0)
    jal ra, factorial

    addi a1, a0, 0
    addi a0, x0, 1
    ecall # Print Result

    addi a1, x0, '\n'
    addi a0, x0, 11
    ecall # Print newline

    addi a0, x0, 10
    ecall # Exit

# Factorial function implementation
factorial:
    # Initialize result to 1 (handles 0! = 1 case)
    li t0, 1           # t0 = result (starts at 1)

    # Handle special case where n = 0 or 1
    li t1, 1           # t1 = 1 for comparison
    ble a0, t1, end    # if n <= 1, return 1

    # Initialize loop counter (i = 2)
```

```
32      # Initialize loop counter (i = 2)
33      li t1, 2            # t1 = i = 2
34
35  fact_loop:
36      mul t0, t0, t1      # result = result * i
37      addi t1, t1, 1      # i = i + 1
38      ble t1, a0, fact_loop # continue if i <= n
39
40  end:
41      mv a0, t0           # move result to return register (a0)
42      jr ra               # return to caller
```

Output:


t0 (x5)
720
t1 (x6)
6
t0 (x5)
5040
t1 (x6)
7
t0 (x5)
40320
t1 (x6)
8
a0 (x10)
40320

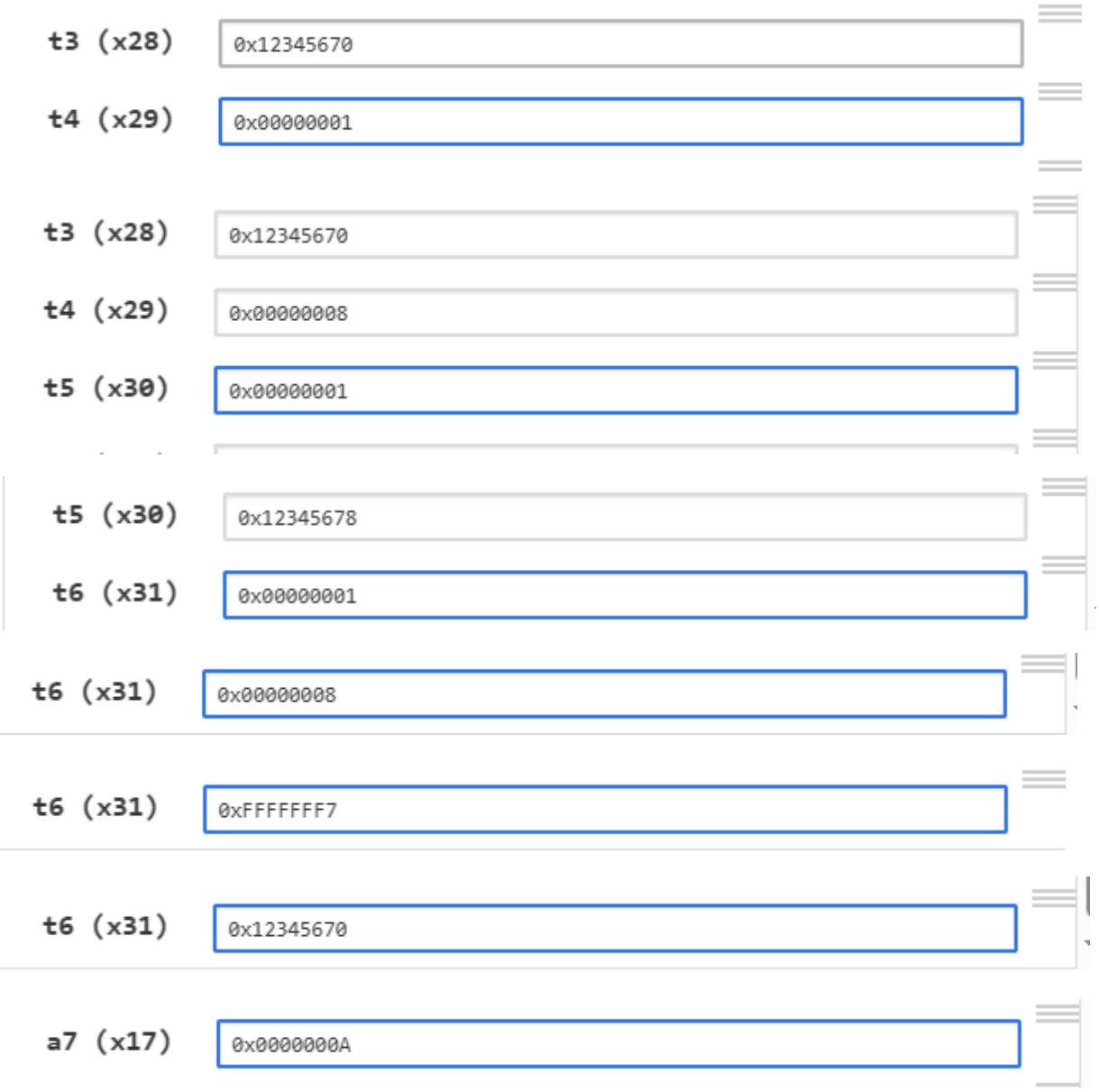

MINGW64:/d/sp24-lab-starter
khalilrehman@DESKTOP-NI915SM MINGW64 /d/sp24-lab-starter (main)
$ java -jar tools/venus.jar lab03/ex5_factorial.s
40320

Task # 5

```
.data
num:    .word 0x12345678  # Default number to manipulate
bitpos: .word 3           # Default bit position (0-31)

.text
.globl main

main:
    # Load default values
    la t0, num
    lw t1, 0(t0)      # t1 = number
    la t0, bitpos
    lw t2, 0(t0)      # t2 = bit position

    
    # 1. Toggle bit (using XOR)
    li t3, 1
    sll t3, t3, t2    # Create mask
    xor t3, t1, t3    # Toggled result in t3

    # 2. Check bit (using AND)
    li t4, 1
    sll t4, t4, t2    # Create mask
    and t4, t1, t4    # result in t4 (0 or 1<<bitpos)

    # 3. Set bit (using OR)
    li t5, 1
    sll t5, t5, t2    # Create mask
    or t5, t1, t5     # Set result in t5

    # 4. Clear bit (using AND with NOT)
    li t6, 1
```

```
33      sll t6, t6, t2      # Create mask
34      not t6, t6          # Invert mask
35      and t6, t1, t6      # Clear result in t6
36
37      # Exit
38      li a7, 10
39      ecall
```

Output:


t3 (x28)
0x12345670
t4 (x29)
0x00000001
t3 (x28)
0x12345670
t4 (x29)
0x00000008
t5 (x30)
0x00000001
t5 (x30)
0x12345678
t6 (x31)
0x00000001
t6 (x31)
0x00000008
t6 (x31)
0xFFFFFFF7
t6 (x31)
0x12345670
a7 (x17)
0x0000000A