Week 6 - Assembly Assignment

. Write an assembly program to sort an array using bubble sort to sort N-elements

```
.data
a:  .word 15, 8, 12  # Array of 3 elements
n:  .word 3          # Number of elements in the array (N)

.text
.globl _start
_start:
    la x6, n              # Load address of N
    lw x21, 0(x6)         # Load the value of N into x21
    addi x21, x21, -1      # Set the outer loop limit to (N - 1)

    li x1, 0              # Outer loop counter (i)

loop1:
    li x2, 0              # Inner loop counter (j)
    la x10, a              # Load the base address of the array into x10

loop2:
    lw x11, 0(x10)        # Load word from address x10 into x11
    lw x12, 4(x10)        # Load word from address x10 + 4 (next element) into x12
    ble x11, x12, no_swap  # If x11 <= x12, no need to swap

    # Swap elements
    sw x12, 0(x10)        # Store x12 (larger) in the current position
    sw x11, 4(x10)        # Store x11 (smaller) in the next position

no_swap:
    addi x10, x10, 4       # Increment the pointer to the next element (4 bytes)
    addi x2, x2, 1        # Increment inner loop counter (j)
    sub x6, x21, x1        # Calculate the effective limit for inner loop (N - 1 - i)
    bge x2, x6, end_inner_loop # If j >= (N - 1 - i), end inner loop

    j loop2               # Continue inner loop

end_inner_loop:
    addi x1, x1, 1        # Increment outer loop counter (i)
    bge x1, x21, end_program  # Stop outer loop if i >= (N - 1)

    j loop1               # Continue outer loop

end_program:
    # Exit program without infinite loop
    li a7, 93             # Exit syscall number for RISC-V
    ecall
```

. Write an assembly program to calculate the factorial of a number with & without recursion

Without recursion:

```
.data
   num: .word 5          # The number to calculate the factorial of
   result: .word 1       # Store the result of the factorial

.text
   la x5, num            # Load address of 'num' into x5
   lw x6, 0(x5)          # Load 'num' into x6 (x6 = 5)

   li x7, 1              # Initialize the result to 1 (x7 = 1)

factorial_loop:
   beq x6, x0, done      # If x6 is 0 (end of the loop), done
   mul x7, x7, x6        # result = result * x6
   addi x6, x6, -1       # Decrement x6 (x6 = x6 - 1)
   j factorial_loop      # Repeat the loop

done:
   la x8, result         # Load address of result into x8
   sw x7, 0(x8)          # Store the result in memory
   j exit                # Jump to exit (end of program)

exit:
   j exit                # Infinite loop to end the program
```

WITH RECURSION:

```
.data
   num: .word 5          # The number to calculate the factorial of
   result: .word 0       # Store the result of the factorial

.text
   la x5, num            # Load address of 'num' into x5
   lw x6, 0(x5)          # Load 'num' into x6 (x6 = 5)

   # Start the recursive factorial function
   jal ra, factorial     # Jump to factorial function

   la x8, result         # Load address of result into x8
   sw x6, 0(x8)          # Store the result in memory

exit:
   j exit                # Infinite loop to end the program

# Recursive factorial function
factorial:
   addi sp, sp, -4       # Make space on stack for return address
   sw ra, 0(sp)          # Save return address
```

```
    bge x6, x0, factorial_continue  # If x6 >= 1, continue the recursion
    li x6, 1             # If x6 == 0, return 1 (base case)

    j factorial_done      # Jump to finish the recursion

factorial_continue:
    addi x6, x6, -1       # Decrement x6 by 1
    jal ra, factorial     # Recursive call to factorial

    mul x6, x6, a0        # Multiply the result (return value) by the current x6

factorial_done:
    lw ra, 0(sp)         # Restore return address
    addi sp, sp, 4        # Clean up stack
    ret                  # Return from the function
```

. Write an assembly program to do matrix multiplication

```
.data
A:  .word 1, 2, 3, 4      # Matrix A: 1 2 / 3 4
B:  .word 5, 6, 7, 8      # Matrix B: 5 6 / 7 8
C:  .space 16             # Result matrix C (2x2), 4 words (each 4 bytes)

.text
    # Load matrix A elements
    la x5, A            # Load address of A into x5
    lw x6, 0(x5)        # Load A[0][0] into x6 (1)
    lw x7, 4(x5)        # Load A[0][1] into x7 (2)
    lw x8, 8(x5)        # Load A[1][0] into x8 (3)
    lw x9, 12(x5)       # Load A[1][1] into x9 (4)

    # Load matrix B elements
    la x10, B           # Load address of B into x10
    lw x11, 0(x10)      # Load B[0][0] into x11 (5)
    lw x12, 4(x10)      # Load B[0][1] into x12 (6)
    lw x13, 8(x10)      # Load B[1][0] into x13 (7)
    lw x14, 12(x10)     # Load B[1][1] into x14 (8)

    # Matrix multiplication C[0][0] = A[0][0]*B[0][0] + A[0][1]*B[1][0]
    mul x15, x6, x11    # x15 = A[0][0] * B[0][0] (1 * 5 = 5)
    mul x16, x7, x13    # x16 = A[0][1] * B[1][0] (2 * 7 = 14)
    add x17, x15, x16   # C[0][0] = 5 + 14 = 19
    la x18, C           # Load address of C into x18
    sw x17, 0(x18)      # Store C[0][0] = 19 in C

    # Matrix multiplication C[0][1] = A[0][0]*B[0][1] + A[0][1]*B[1][1]
    mul x15, x6, x12    # x15 = A[0][0] * B[0][1] (1 * 6 = 6)
    mul x16, x7, x14    # x16 = A[0][1] * B[1][1] (2 * 8 = 16)
    add x17, x15, x16   # C[0][1] = 6 + 16 = 22
    sw x17, 4(x18)      # Store C[0][1] = 22 in C
```

```
    # Matrix multiplication C[1][0] = A[1][0]*B[0][0] + A[1][1]*B[1][0]
    mul x15, x8, x11    # x15 = A[1][0] * B[0][0] (3 * 5 = 15)
    mul x16, x9, x13    # x16 = A[1][1] * B[1][0] (4 * 7 = 28)
    add x17, x15, x16   # C[1][0] = 15 + 28 = 43
    sw x17, 8(x18)      # Store C[1][0] = 43 in C

    # Matrix multiplication C[1][1] = A[1][0]*B[0][1] + A[1][1]*B[1][1]
    mul x15, x8, x12    # x15 = A[1][0] * B[0][1] (3 * 6 = 18)
    mul x16, x9, x14    # x16 = A[1][1] * B[1][1] (4 * 8 = 32)
    add x17, x15, x16   # C[1][1] = 18 + 32 = 50
    sw x17, 12(x18)     # Store C[1][1] = 50 in C

exit:
    j exit              # Infinite loop to end the program
```