Week 3 - Assembly Assignment

. Write an assembly program to check if a number is a 2 out of 5 number

```
    .data
a:  .byte 0x12  # Example number (binary: 00010010)

    .text
    la x10, a           # Load address of a
    lbu x11, 0(x10)     # Load the byte into x11

    # Mask the lower 5 bits and check only the lower 5 bits (0x1F = 00011111)
    andi x11, x11, 0x1F  # Mask the lower 5 bits (0x1F = 00011111)

    li x22, 5            # Set the bit count limit to 5 (for 5 bits)
    li x23, 0            # Initialize bit count to 0

check_bits:
    andi x12, x11, 0x01  # Mask the least significant bit (0x01)
    beq x12, x0, next_bit  # If the bit is 0, skip to the next bit
    addi x23, x23, 1      # Increment bit count (set bit found)

next_bit:
    srli x11, x11, 1      # Shift the number right by 1 to check the next bit
    addi x22, x22, -1     # Decrease the remaining bit count
    bnez x22, check_bits  # Loop until 5 bits are checked

    # After checking 5 bits, compare the count with 2
    li x24, 2             # Load the expected count (2 bits)
    beq x23, x24, valid_number  # If the count is 2, it's a valid 2-out-of-5 number

    j exit                # If not, exit the program

valid_number:
    # Code for valid 2-out-of-5 number (you can store result or proceed as needed)
    # For now, let's exit the program.
    j exit

exit:
    addi x15, x0, 0x00    # Exit code
```

. Write an assembly program to encode a number using Hamming code.

```
.data
    data: .byte 0xA         # Input data (4-bit), 0xA = 1010
    result: .space 1        # Space for 7-bit Hamming code

.text
    la x10, data            # Load address of 'data'
```

```
    lb x11, 0(x10)          # Load the 4-bit data value into x11 (byte size)

    # Prepare for Hamming encoding
    # p1 = parity for positions 1 and 3 -> xor(data bits at positions 3)
    li x12, 0               # Clear x12 (used for parity calculation)
    andi x13, x11, 0x1      # Extract bit at position 3 (bit 0 of x11)
    xor x12, x12, x13       # XOR to calculate parity for p1 (position 1)
    andi x13, x11, 0x4      # Extract bit at position 3 (bit 2 of x11)
    xor x12, x12, x13       # XOR to calculate parity for p1 (position 3)

    andi x12, x12, 0x1      # Take the result mod 2 (either 0 or 1)

    # Store p1 in the first position (bit 1)
    slli x12, x12, 6        # Shift p1 to position 1 (shift left by 6)
    or x11, x11, x12        # Set p1 in the 7-bit result

    # p2 = parity for positions 2 and 3 -> xor(data bits at positions 3)
    li x12, 0               # Clear x12 for parity calculation
    andi x13, x11, 0x1      # Extract bit at position 2
    xor x12, x12, x13       # XOR for p2
    andi x13, x11, 0x4      # Extract bit at position 3
    xor x12, x12, x13       # XOR for p2

    andi x12, x12, 0x1      # Take the result mod 2 (either 0 or 1)

    # Store p2 in the second position (bit 2)
    slli x12, x12, 5        # Shift p2 to position 2 (shift left by 5)
    or x11, x11, x12        # Set p2 in the 7-bit result

    # p3 = parity for positions 3 and 4 -> xor(data bits at positions 3 and 4)
    li x12, 0               # Clear x12 for parity calculation
    andi x13, x11, 0x1      # Extract bit at position 3
    xor x12, x12, x13       # XOR for p3
    andi x13, x11, 0x2      # Extract bit at position 4
    xor x12, x12, x13       # XOR for p3

    andi x12, x12, 0x1      # Take the result mod 2 (either 0 or 1)

    # Store p3 in the third position (bit 3)
    slli x12, x12, 4        # Shift p3 to position 3 (shift left by 4)
    or x11, x11, x12        # Set p3 in the 7-bit result

    # Final encoded result is in x11 (7 bits)
    la x10, result          # Load address of result
    sb x11, 0(x10)          # Store the encoded 7-bit result

exit:
    addi x15, x0, 0x00      # Exit program
```