Week 3

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

.data

input: .word 0b1011

output: . word 0

.text

start:

la x10, input

lw x11, 0(x10)

andi x12, x11, 0x1

srli x11, x11, 1

andi x13, x11, 0x1

srli x11, x11, 1

andi x14, x11, 0x1

srli x11, x11, 1

andi x15, x11, 0x1

xor x16, x15, x14

xor x16, x16, x12

xor x17, x15, x13

xor x17, x17, x12

xor x18, x15, x14

slli x19, x16, 6

slli x20, x17, 5

or x19, x19, x20

slli x20, x15, 4

or x19, x19, x20

slli x20, x18, 3

or x19, x19, x20

slli x20, x14, 2

or x19, x19, x20

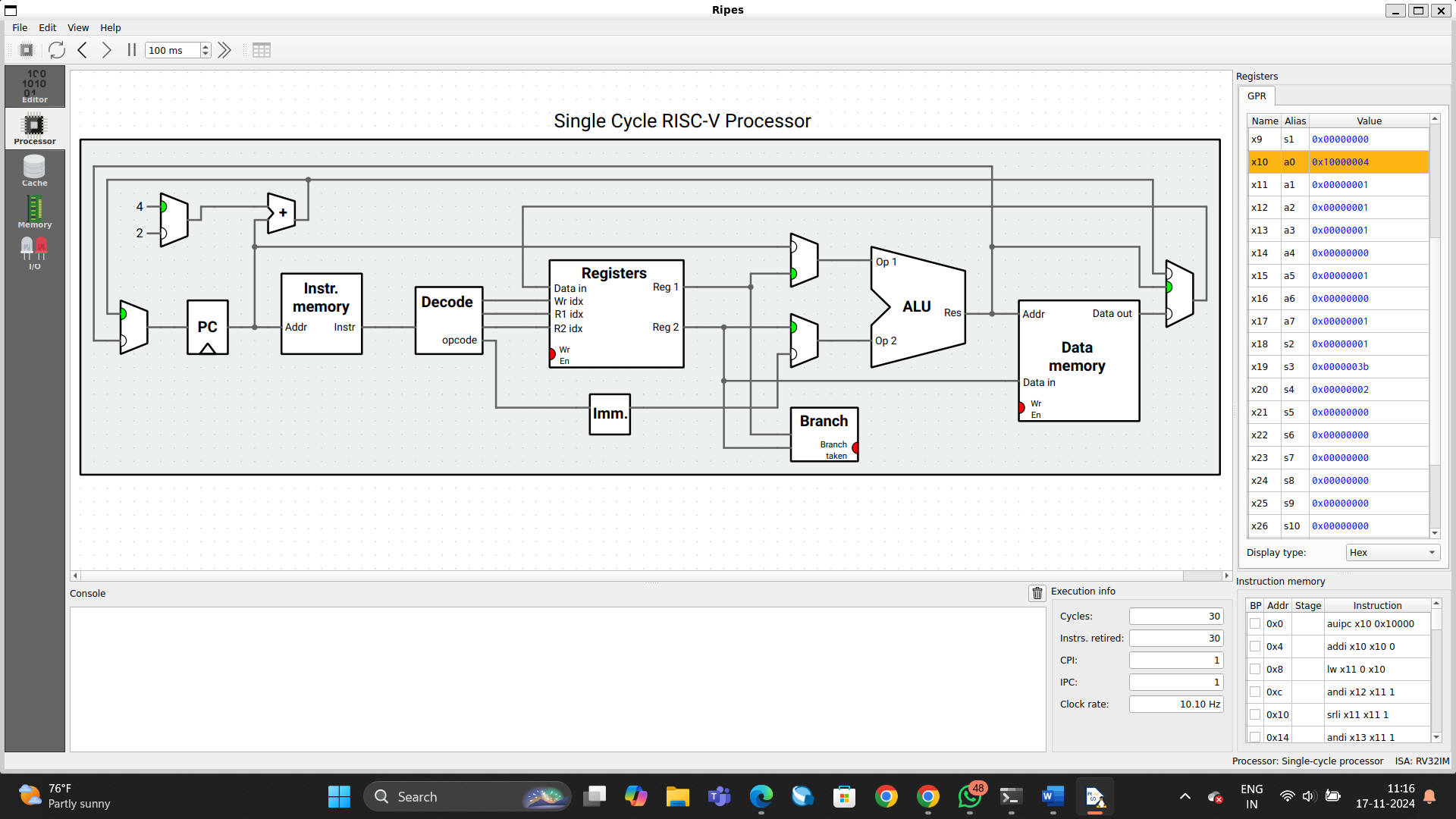
slli x20, x13, 1

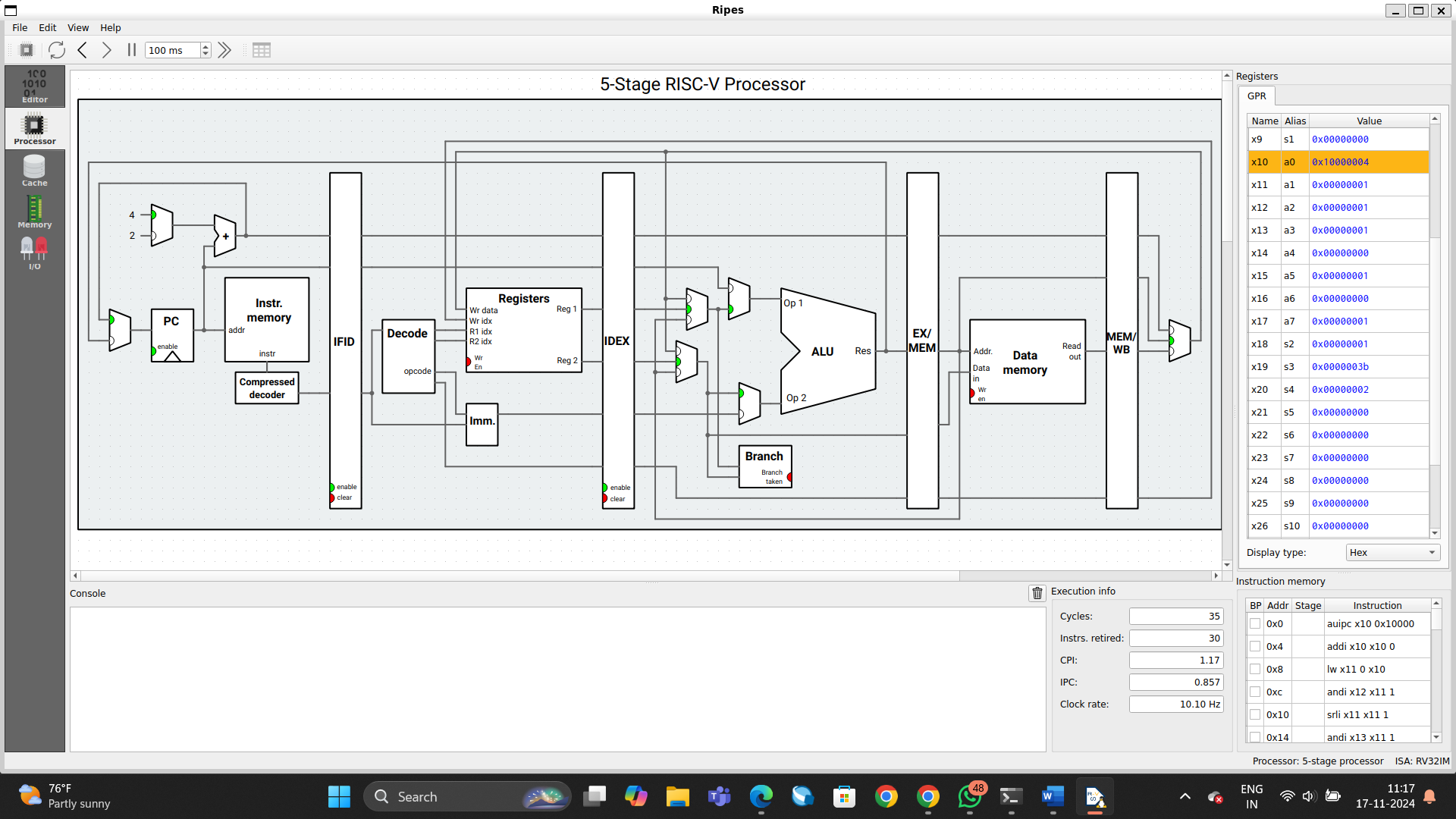
or x19, x19, x20

or x19, x19, x12

la x10, output

sw x19, 0(x10)





1. Write an assembly program to encode a umber using a hamming code

.data

data: .byte 0xA # Input data (4-bit), 0xA = 1010

result: .word 0 # 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

