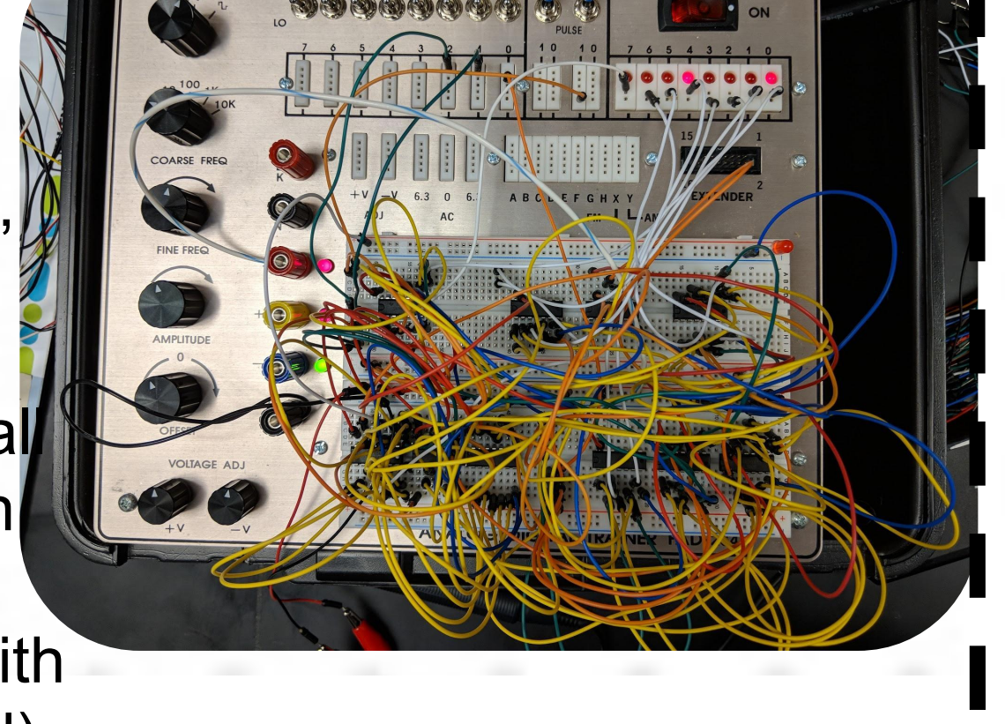
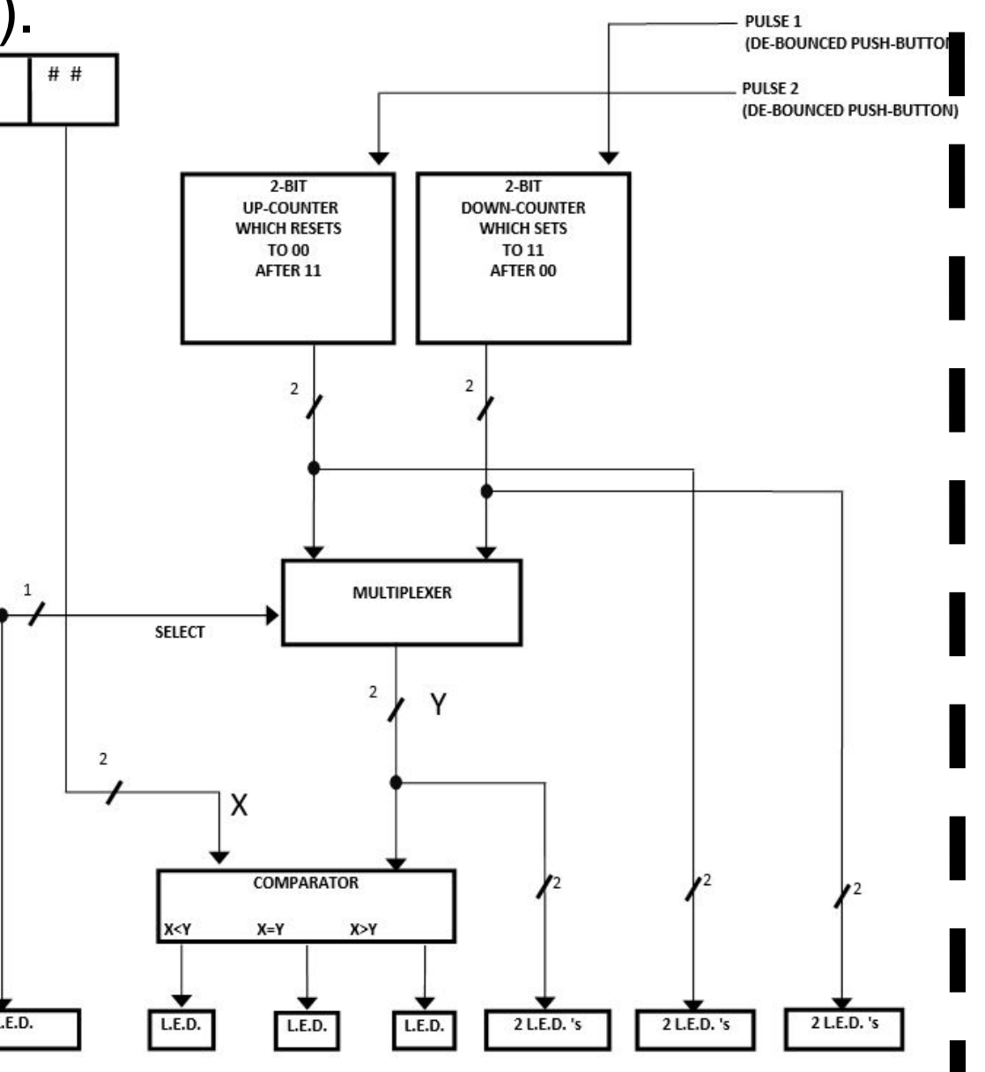
Lab 2: Beginning Computer Instruction-Set Design, and PLC Equivalent Processing

1. A Logisim circuit simulation using only FLIP-FLOP's, INVERTORS, AND's, OR's, and NAND's for all circuits
2. TTL SSI chips on circuit trainers using only FLIP-FLOP's, INVERTORS, AND's, OR's, NAND's for all circuits. Also implement all 10 LED's (extra LED's are in stock, and resistors too, if needed)
3. NanoLC Programmable Logic Controllers (PLC's) with improvised inputs and outputs (debouncing not required). You may encode your inputs and outputs and use the display screen on the base unit; You may also input and output an encoded serial bit stream if you run out of parallel ways to input and output everything.

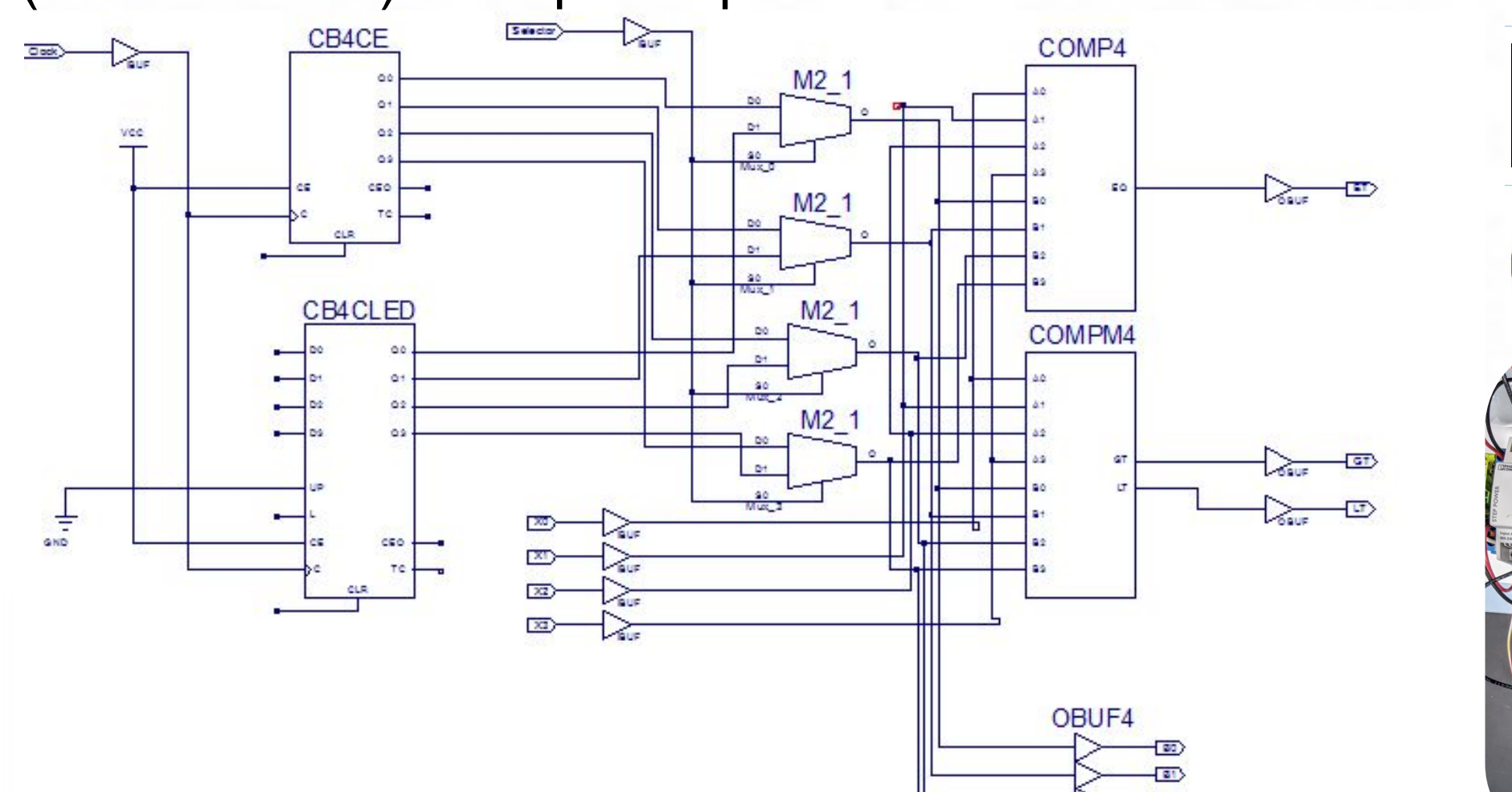


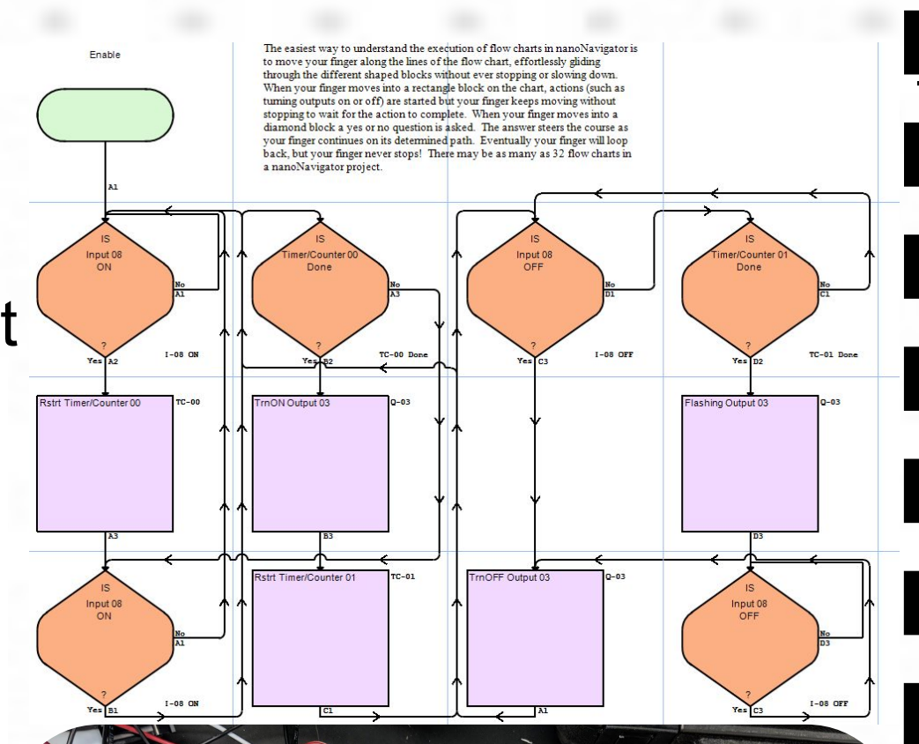


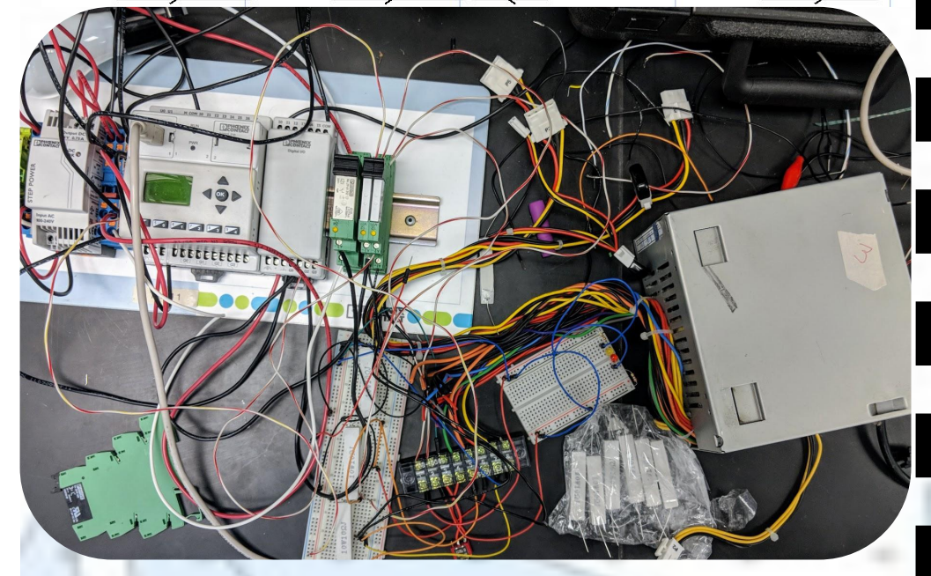
Lab 3: Computer Instruction-Set Design continued, including FPGA; Plus PLC Power-Control

1. Using a NanoLC PLC Base Unit, I/O unit, and Relay, turn on a 120 volt light bulb five seconds after an external input pushbutton is pressed.
2. Design and implement the circuit below in two ways:
• A Logisim circuit simulation using only FLIP-FLOP's, INVERTORS, AND's, OR's, and NAND's for all circuits
• Field Programmable Gate Array (FPGA) using the largest functional blocks possible (I,e, avoid using only FLIP-FLOP's, INVERTORS, AND's, OR's, and NAND's)

Lab3 Opcode:
Instruction Set:
(OP-CODE=1): Compare operand to up-counter count
(OP-CODE=0): Compare operand to down-counter count





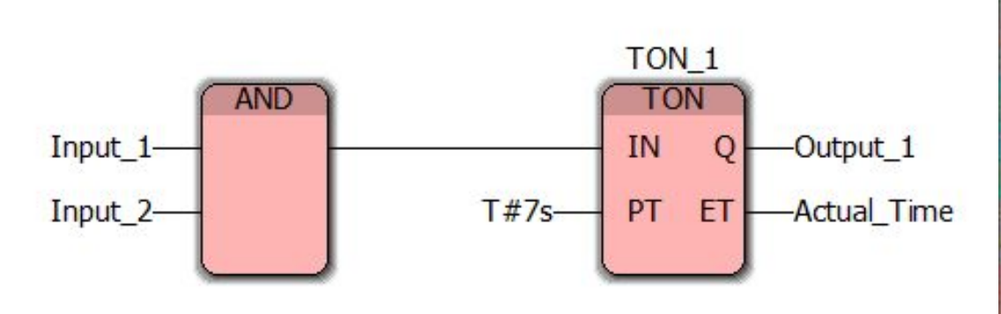


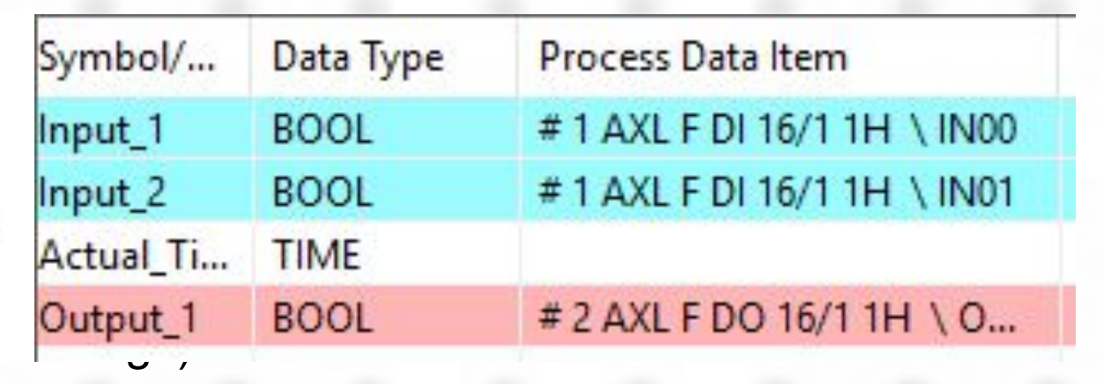
Lab 5 Computer Instruction-Set Design continued with 2-way superscalar pipelines, including FPGA; Plus Advanced $5000 Phoenix Contact PLC's

1. Using an Advanced AXC/AXL PLC, perform the lab experiment shown in our custom Lab Manual and include in report info you recommend be added to this manual
2. Duplicate everything done below in the last lab. You will be creating two parallel pipelines in a 2-way Superscalar architecture. Create a STACK of all 13 instructions of your instruction set, and create a FINITE STATE MACHINE that alternates FETCHING sequential instructions from your stack, alternating which pipe receives the instruction. Let hardware DECODE and EXECUTE as in last lab. NOTE: This architecture is pipelined and not yet fully superscalar since FETCHES are not yet done simultaneously.

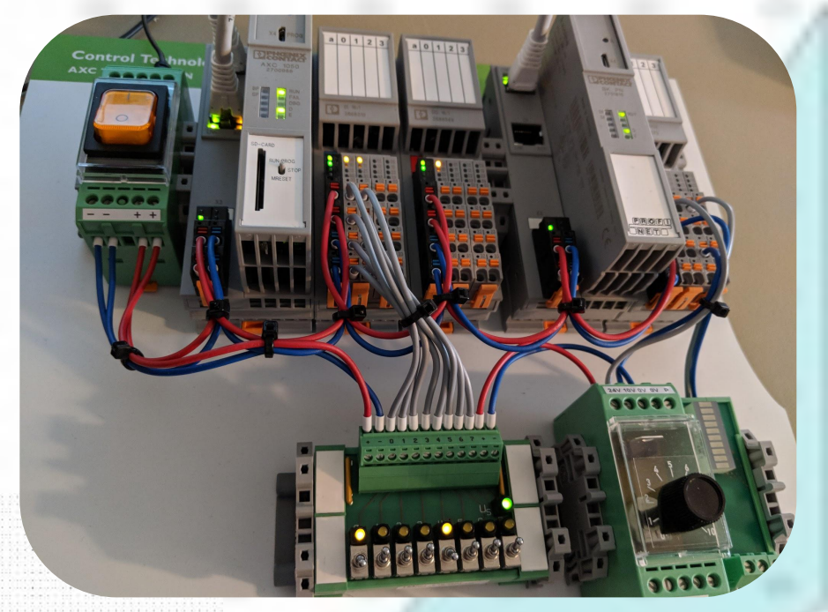
Instruction Set
DON'T CHANGE THE OP-CODES !

AND TURN OFF OPERATIONS THAT ARE NOT

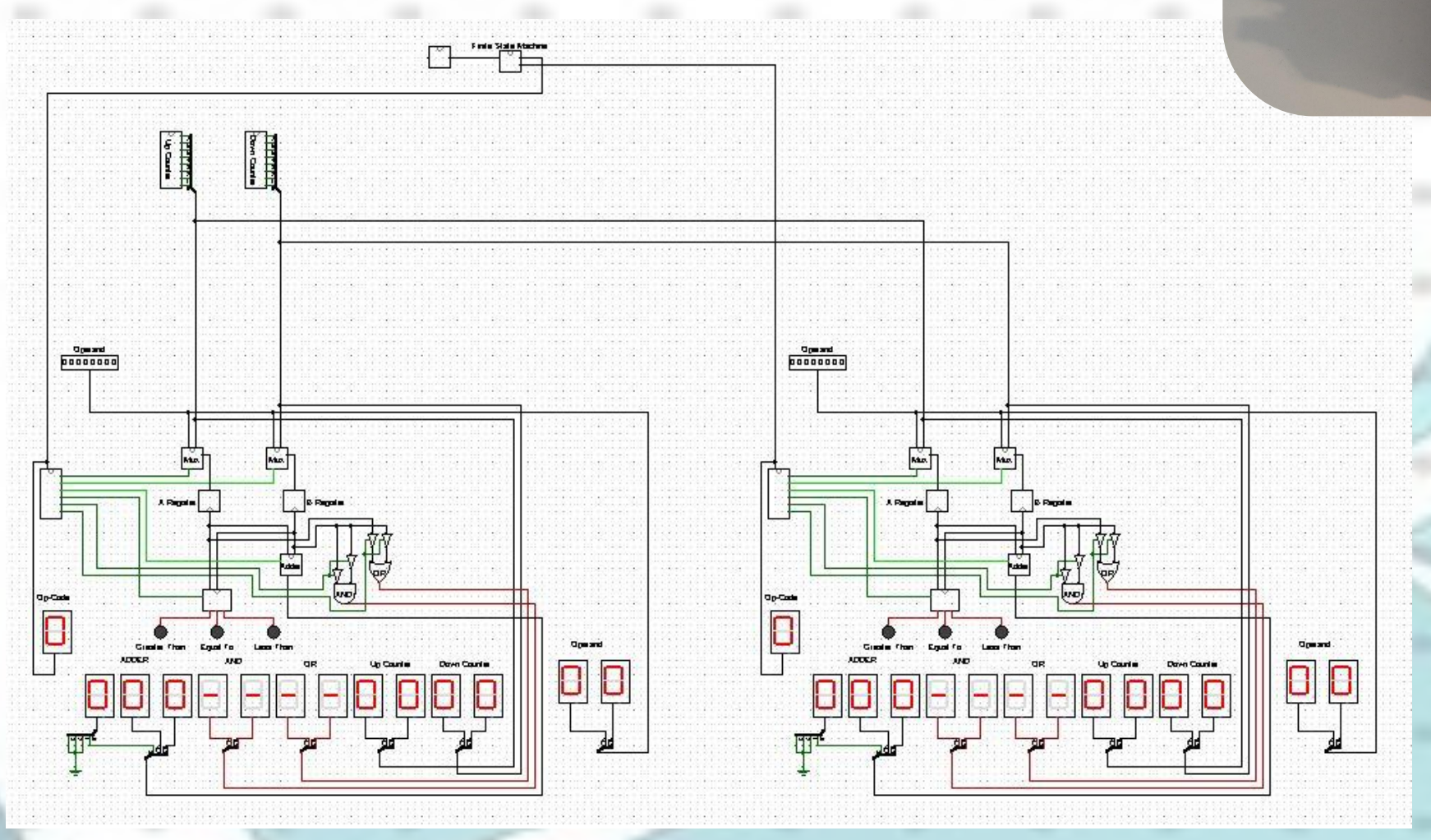




Symbol/...	Data Type	Process Data Item
Input_1	BOOL	# 1 AXL F DI 16/1 1H \ IN00
Input_2	BOOL	# 1 AXL F DI 16/1 1H \ IN01
Actual_Ti...	TIME	
Output_1	BOOL	# 2 AXL F DO 16/1 1H \ O...



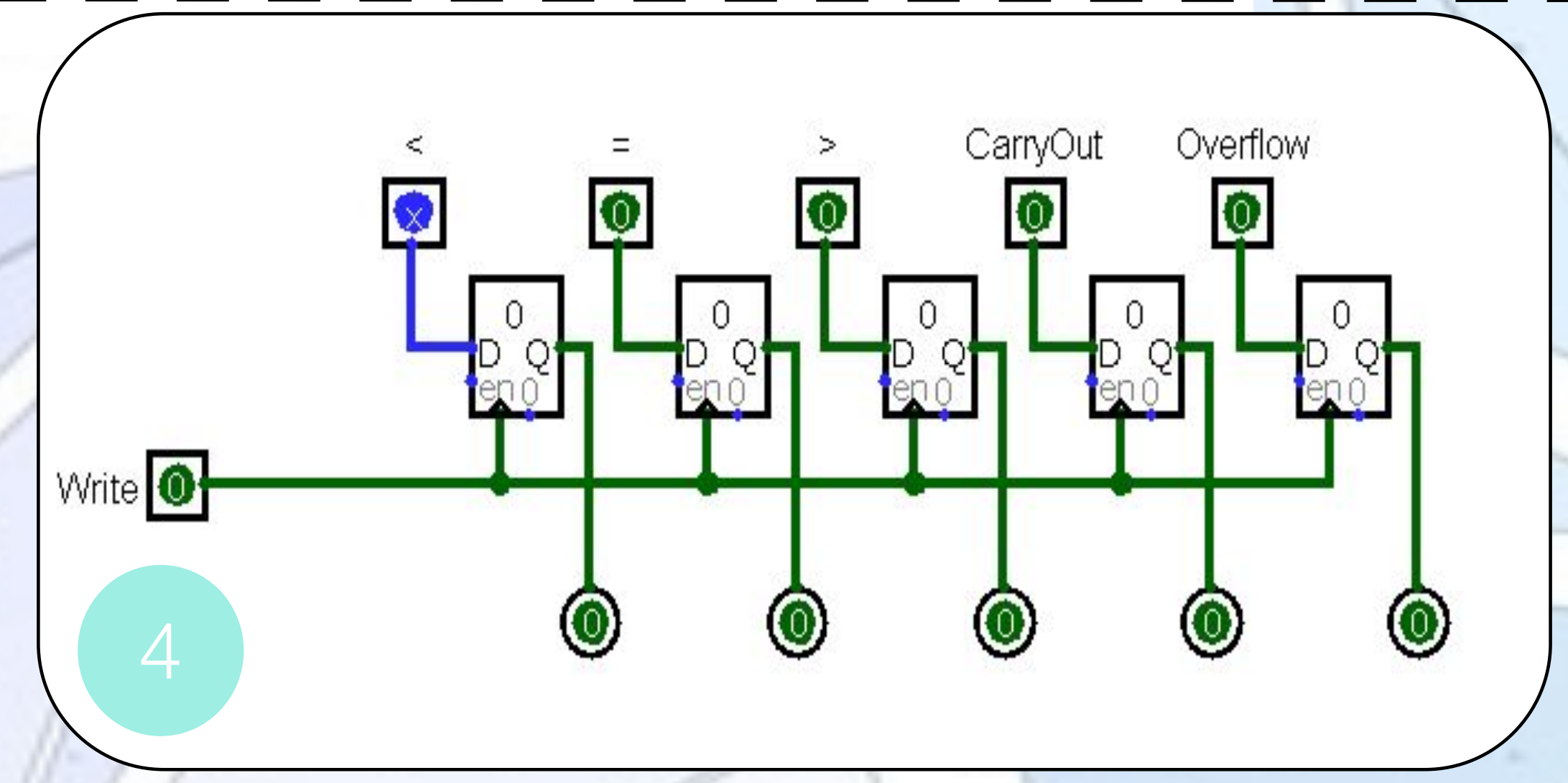
(OP-CODE = 0000) Add Immediate Data to counter #1 (UP-COUNTER)
(OP-CODE = 0001) Add Immediate Data to counter #2 (DOWN-COUNTER)
(OP-CODE = 0010) Compare Immediate Data with counter #1 (UP-COUNTER)
(OP-CODE = 0011) Compare Immediate Data with counter #2 (DOWN-COUNTER)
(OP-CODE = 0100) AND Immediate Data to counter #1 (UP-COUNTER)
(OP-CODE = 0101) AND Immediate Data to counter #2 (DOWN-COUNTER)
(OP-CODE = 0110) OR Immediate Data to counter #1 (UP-COUNTER)
(OP-CODE = 0111) OR Immediate Data to counter #2 (DOWN-COUNTER)
(OP-CODE = 1000) Add Counters
(OP-CODE = 1001) Compare Counters
(OP-CODE = 1010) AND Counters
(OP-CODE = 1011) OR Counters
(OP-CODE = 11XX) Turn off everything but display of Op-code and Operand

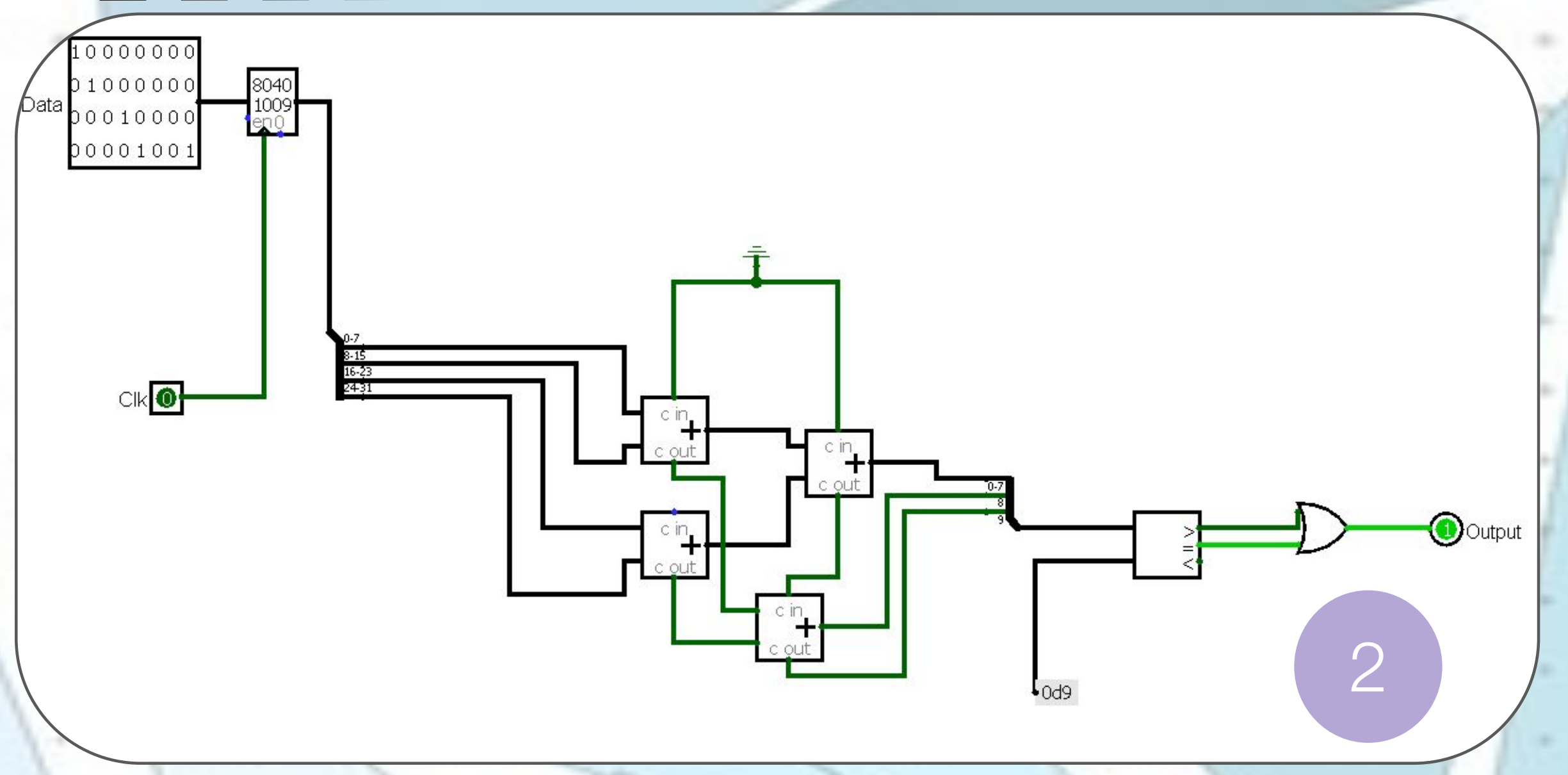


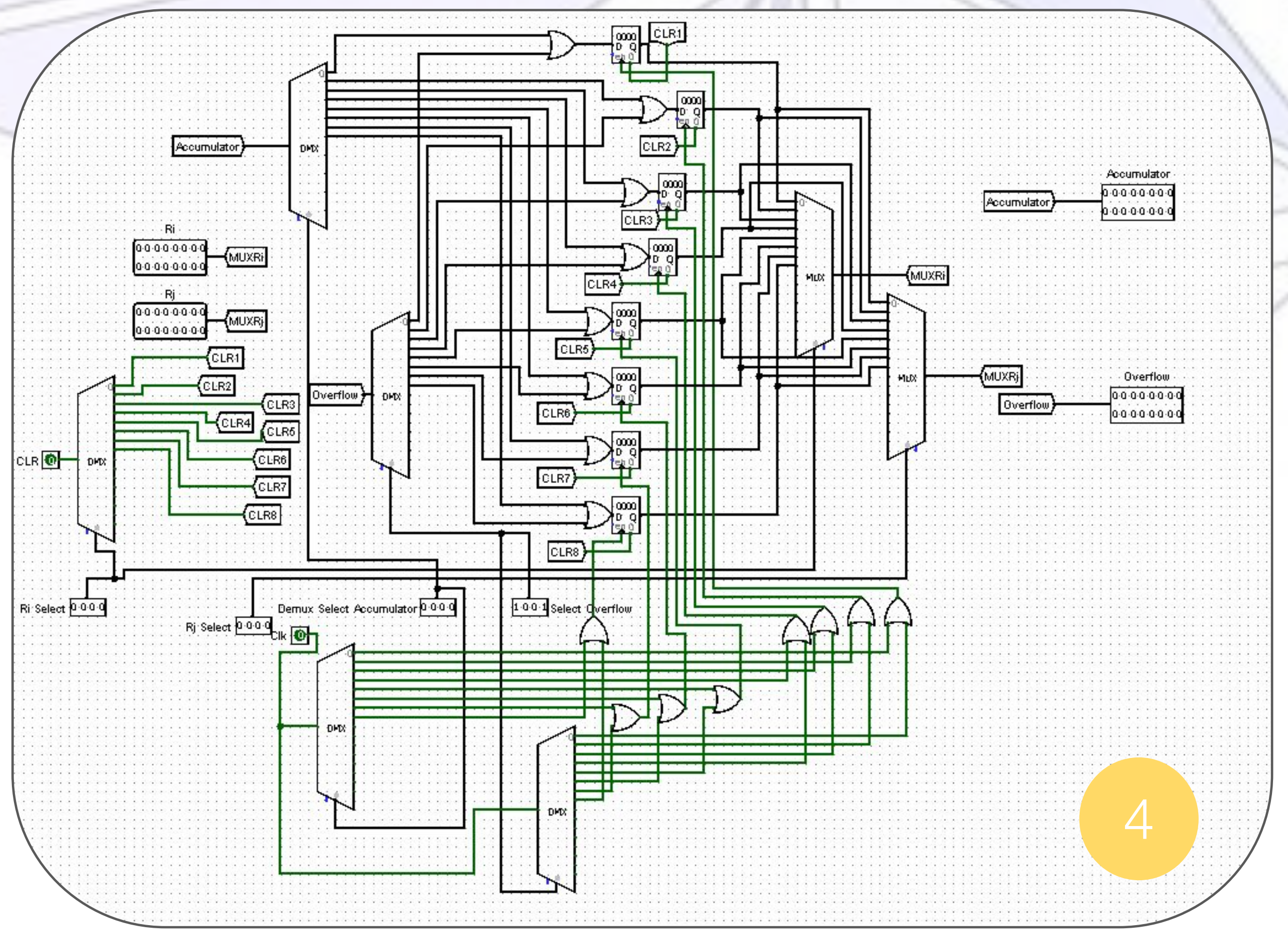
Lab 6/7/8: Vector-Array / Neuron Processor Design

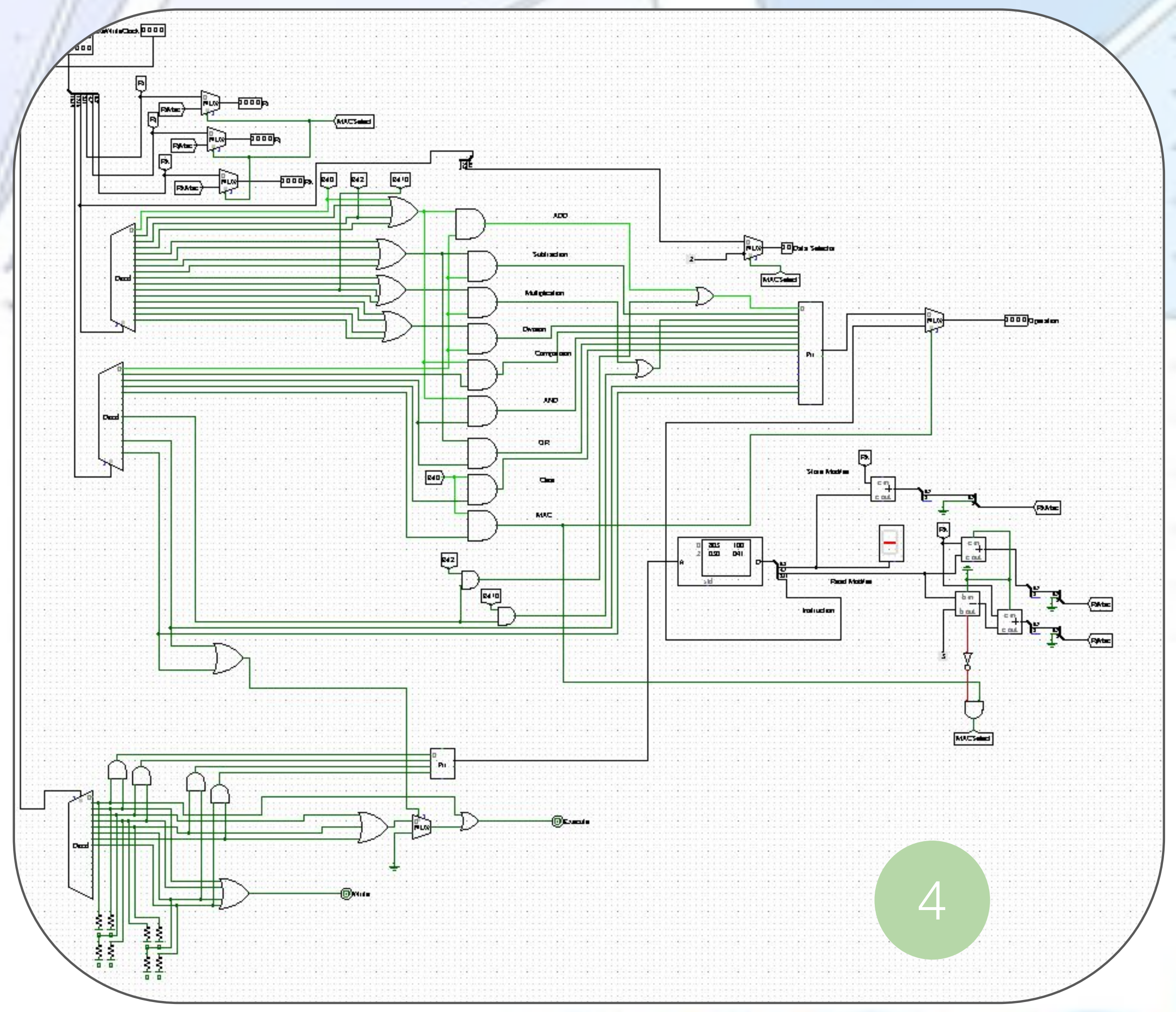
1. Create four parallel scalar functional units with vector registers V_i, V_j, V_k created from R_i, R_j, R_k of the units.
2. Add a 32-bit adder to add the two 16-bit product results from each unit after a vector multiply, as part of a matrix row x column instruction
3. Put results into a 32 bit scalar accumulator, then into a neuron transfer function.
4. Create an embedded code stack, controlled by a program counter via a master control unit with a final state machine that implements the simple pipeline of fetch, decode, execute, and write-back; plus any special states.
5. Then embed a carefully crafted assembly language code segment to demonstrate the functionality of your instruction set and circuitry in the minimal amount of time that you can defend as providing comprehensive testing of all scalar, vector, matrix, and neuron machine instructions and hardware.

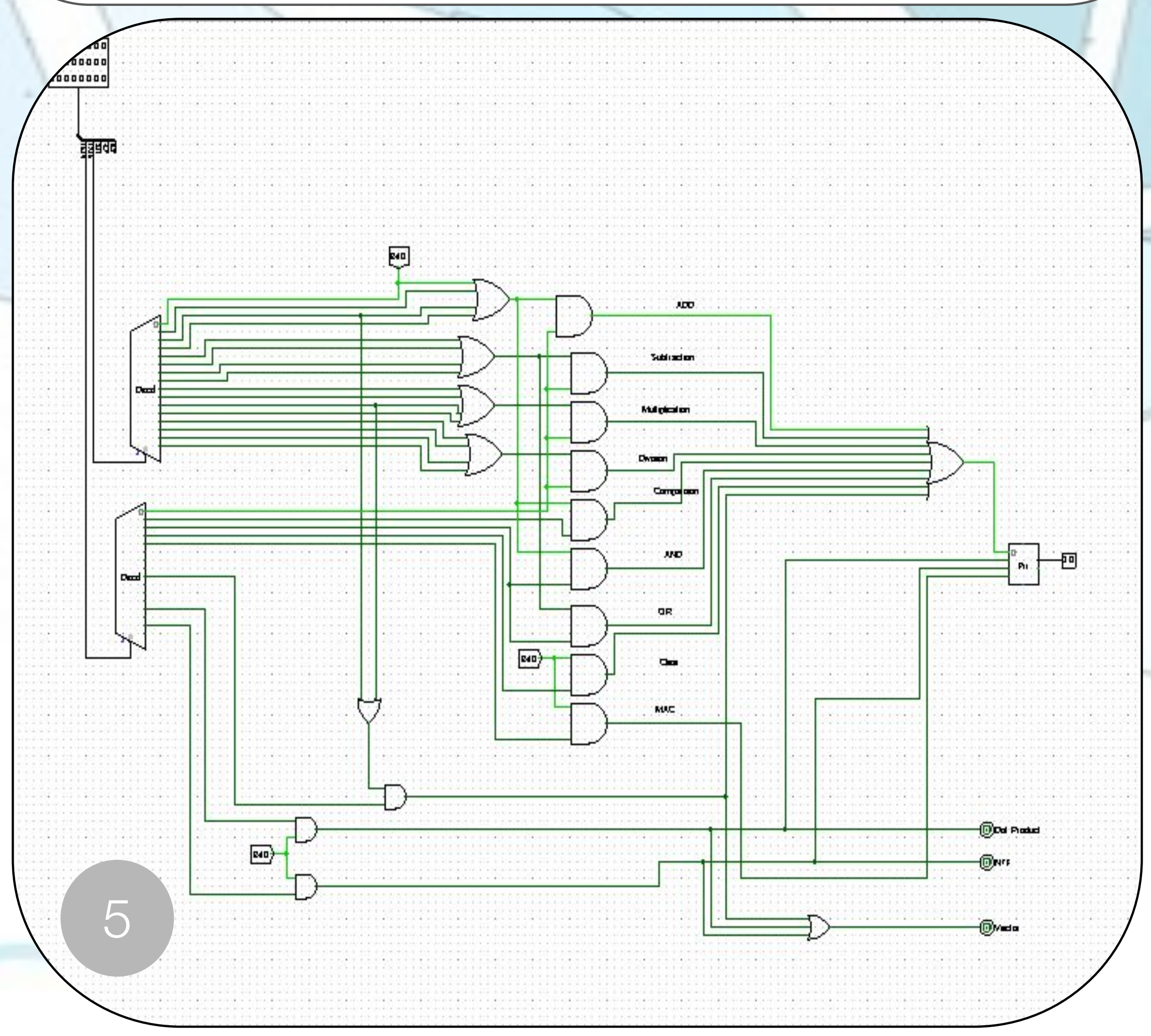
5 **Main**
3 **Master Control**
3 **Control Logic**
1 **Parallel Scalar**
2 **Neuron Transfer Function**
5 **Execute Step**
4 **Register Bank**
4 **Status Register**

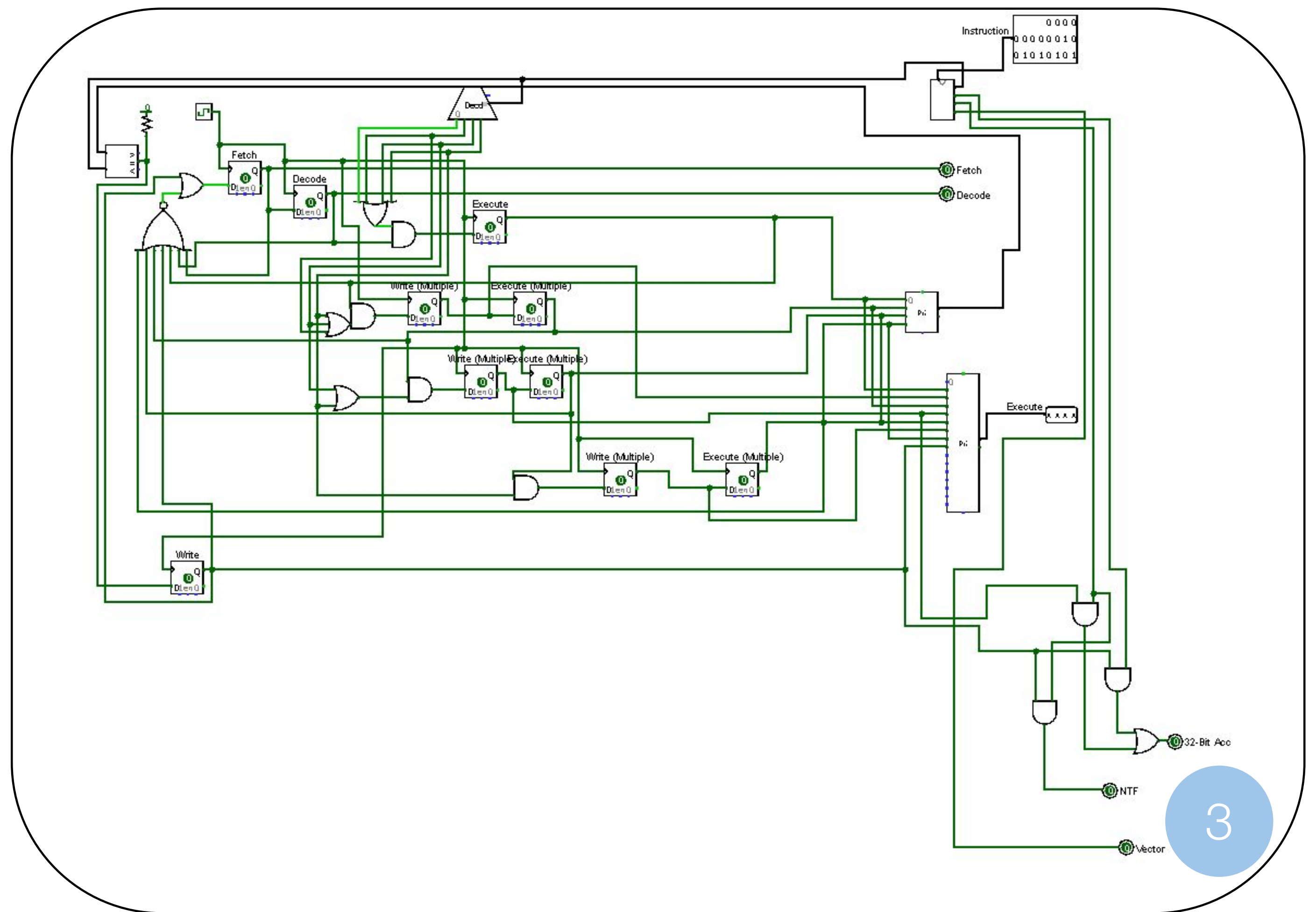


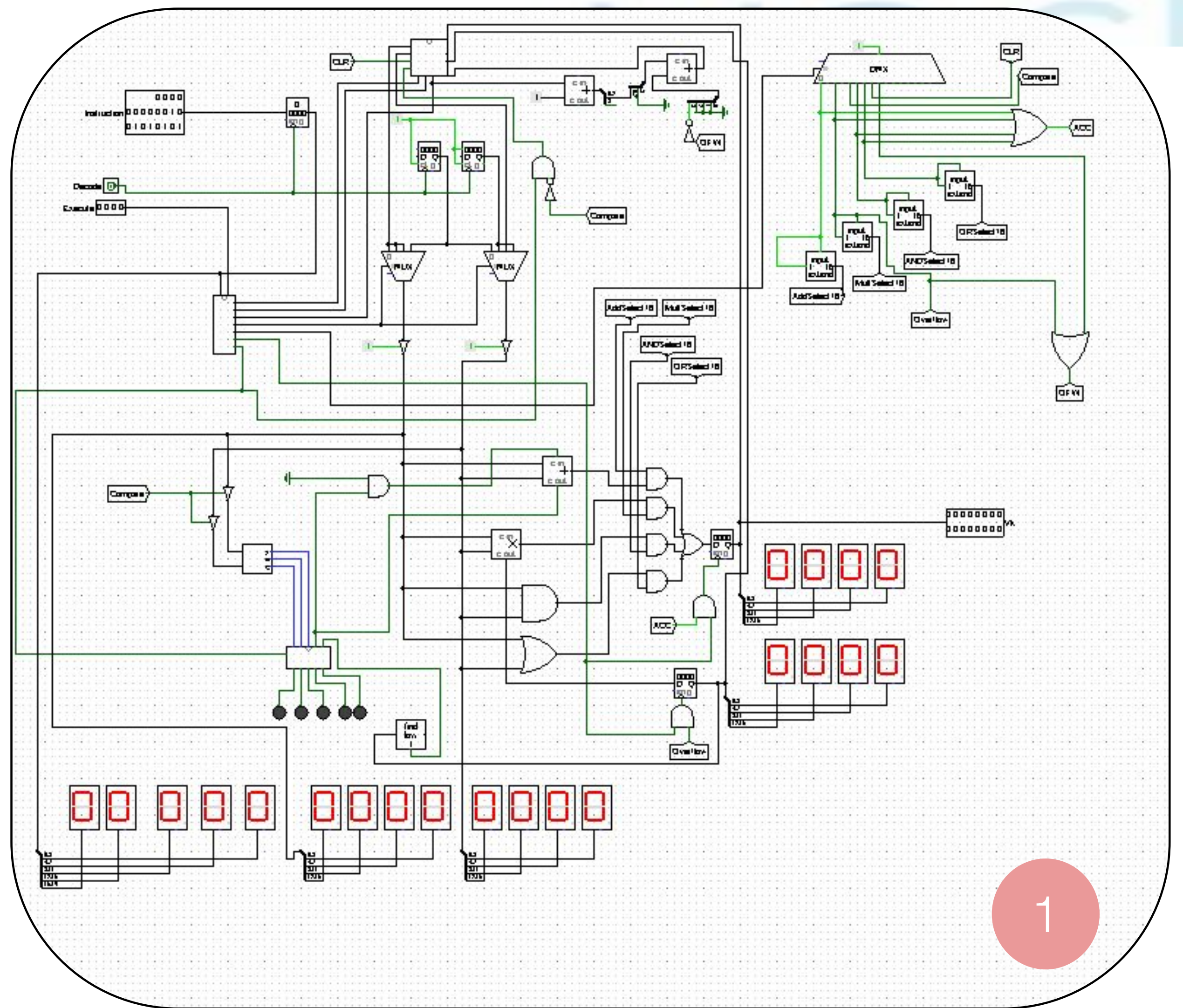


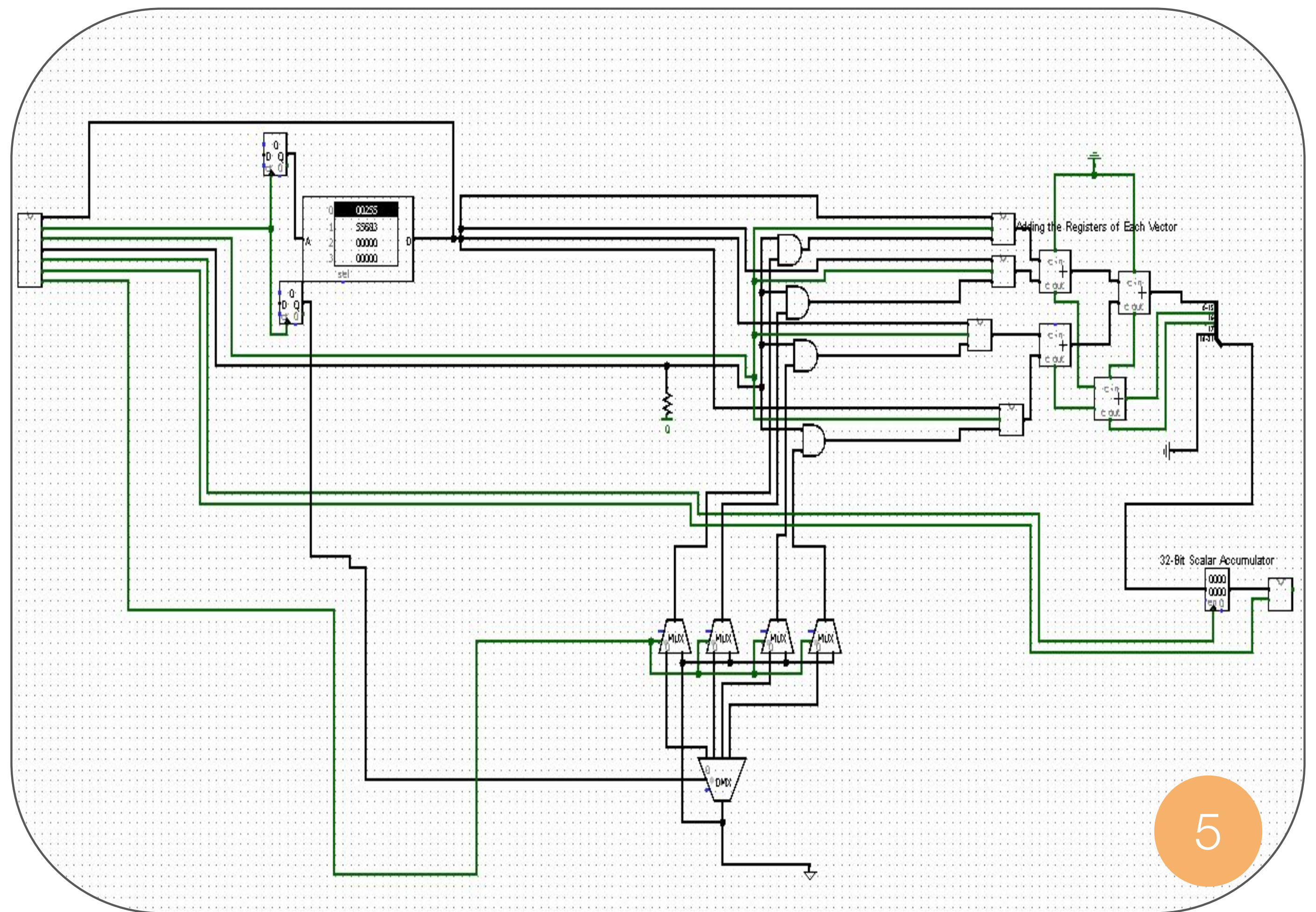












MACHINE INSTRUCTION SET

SCALAR ARITHMETIC ADDITION
```
00h (OP-CODE = 00000000) Ri + counter#1  →Rk
01h (OP-CODE = 00000001) Ri + counter#2  →Rk
02h (OP-CODE = 00000010) Ri + Rj  →Rk
03h (OP-CODE = 00000011) counter#1 + counter#2  →Rk
```
SCALAR ARITHMETIC SUBTRACTION
```
04h to 07h (OP-CODE = 000001XX)  Reserved for subtraction instructions
```
SCALAR ARITHMETIC MULTIPLICATION
```
08h (OP-CODE = 00001000) Ri x counter#1  →Rk, overflow →Rk+1 (wrap to R0)
09h (OP-CODE = 00001001) Ri x counter#2  →Rk, overflow →Rk+1 (wrap to R0)
0Ah (OP-CODE = 00001010) Ri x Rj  →Rk, overflow →Rk+1 (wrap to R0)
0Bh (OP-CODE = 00001011) counter#1 x counter#2  →Rk, overflow→Rk+1(wrap to R0)
```
SCALAR ARITHMETIC DIVISION
```
0Ch to 0Fh (OP-CODE = 000011XX)  Reserved for division instructions
```
SCALAR ARITHMETIC COMPARISON
```
10h (OP-CODE = 00010000) Compare Ri with counter#1  →Rk
11h (OP-CODE = 00010001) Compare Ri with counter#2  →Rk
12h (OP-CODE = 00010010) Compare Ri with Rj  →Rk
13h (OP-CODE = 00010011) Compare Counters
```
SCALAR LOGICAL AND
```
20h (OP-CODE = 00100000) Ri AND counter#1  →Rk
21h (OP-CODE = 00100001) Ri AND counter#2  →Rk
22h (OP-CODE = 00100010) Ri AND Rj  →Rk
23h (OP-CODE = 00100011) AND counters  →Rk
```
SCALAR LOGICAL OR
```
24h (OP-CODE = 00100100) Ri OR counter#1  →Rk
25h (OP-CODE = 00100101) Ri OR counter#2  →Rk
26h (OP-CODE = 00100110) Ri OR Rj  →Rk
27h (OP-CODE = 00100111) OR counters  →Rk
```
CLEAR
```
30h (OP-CODE = 00110000) Clear Ri
```
MAC
(Multiply, Accumulate). Accumulator = Accumulator + (Ri x Rj) *considering overflow also*
```
40h (OP-CODE = 01000000) Step 1: Accumulator  →Rk+3 (wrap to R0+)
                         Step 2: Overflow →Rk+4 (wrap to R0+)
                         Step 3: Ri x Rj →Rk, overflow →Rk+1 (wrap to R0)
                         Step 4: (Rk+3)+Rk →Rk
                         Step 5: (Rk+4)+(Rk+1)+Carry →Rk+1
```

VECTOR-ARRAY / MATRIX &

NEURON INSTRUCTIONS

Vi, Vj, and Vk are created from Ri's, Rj's, and Rk's of the four parallel functional units

VECTOR ARITHMETIC ADDITION
```
82h (OP-CODE = 10000010) Vi + Vj  →Vk
```
VECTOR ARITHMETIC SUBTRACTION
```
84h to 87h (OP-CODE = 100001XX)  Reserved for subtraction instructions
```
VECTOR ARITHMETIC MULTIPLICATION
```
0Ah (OP-CODE = 10001010) Vi x Vj  →Vk, overflow →Vk+1 (wrap to R0)
```
VECTOR ARITHMETIC DIVISION
```
8Ch to 8Fh (OP-CODE = 100011XX)  Reserved for division instructions
```
MATRIX ROW x COLUMN (i.e., Dot-Product)
```
C0h (OP-CODE = 11000000) Vi x Vj  →Vk, overflow →Vk+1 (wrap to R0)
     Vk(1)+Vk(2)+Vk(3)+Vk(4) v 32-Bit Scalar Accumulator
```
NEURON TRANSFER FUNCTION
```
E0h (OP-CODE = 11100000) Vi x Vj  →Vk, overflow →Vk+1 (wrap to R0)
     Vk(1)+Vk(2)+Vk(3)+Vk(4) → 32-Bit Scalar Accumulator
     32-Bit Scalar Accumulator → Neuron Transfer Function
```

EGCR433 Summary:Dillon Dotson (Senior Engineering), Patrick Durofchalk (Senior Computer Engineering), Miguel Gonzalez (Sophomore Computer Engineering)