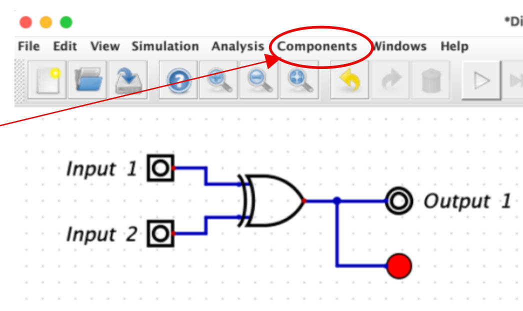
Lab 10 : Digital CPU Foundations (Gates → Subcircuits → PC & Memory)

Objectives:

- Create and simulate the minimal 4-bit ISA
- **CPU goal:**
  - Instruction: 8 bits
  - Data width: 4 bits
  - Registers: 8 register - R0–R7 (four 4-bit regs)
  - Data RAM: 8 locations (3-bit address), each 4-bit
  - Program ROM: 16 locations (4 bits), each 9-bit instruction
  - Instruction: 9-bit
    - **LDR** – Load from register (Opcode 000): Rd = Rs Format: 000 | Rd[2:0] | Rs[2:0]
    - **LDM** – Load from memory (Opcode 001): Rd = [Addr] Format: 001 | Rd[2:0] | Src[2:0]
    - **LDI** – Load Immedate (Opcode 010): Rd = #Immedate Format: 010 | Rd[2:0] | #Immedate
    - **STR** – Store to memory (Opcode 011): [Addr] = Rs Format: 011 | Rs[2:0] | Dest[2:0]
    - **ADD** – Add register (opcode 100): Rd = Rd + Rs Format: 100 | Rd[2:0] | Rs[2:0]
    - **BZ** – Branch if zero (Opcode 101): If Zero, branch Format: 101 | 00 | Addr[3:0]
    - **Legend:**
      - **Rd:** Destination register (3 bits).
      - **Rs:** Source register (3 bits).
      - **Src:** data at source address (3 bits).
      - **Dest:** data at destination address (3 bits).
      - **Immediate** : Constant (3 bits).
      - **Addr:** address in ROM (4 bits)

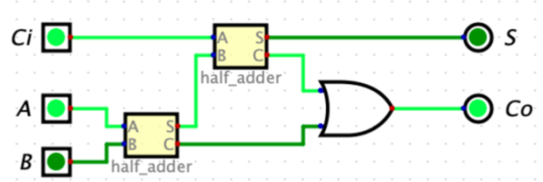
Lab 10.1 : Install and launch



1. Download the latest **Digital.jar** file from the official GitHub repository: https://github.com/hneemann/Digital/releases
2. Ensure you have Java installed on your system (Mac, Windows, or Linux).
3. Run program by launch Digital.jar. If it doesn't start, run: `java -jar Digital.jar`
4. Create a XOR Gate with two inputs and one output by
   4.1. Add 2 inputs : Components -> IO -> **Input**.
   4.2. Add output: Components -> IO -> **Output**.
   4.3. Label both input and putput by right click on component and label it.
   4.4. Add XOR-Gate : Components -> Logic -> **XOR**.
   4.5. Wire inputs , output to the gate.
   4.6. Simulate by click on the **play button** located in the toolbar.
   4.7. Try to change inputs and notices output.
5. To further process the circuit, the simulation must first be stopped with the **stop button** in the tool bar
6. At menu bar "Analysis", try Analysis and Synthesis.

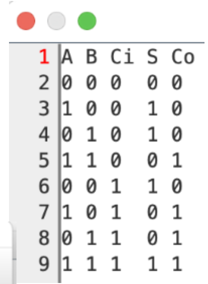
TA Cecking: Show the TA simulated XOR circuit and its truth table. ____

Lab 10.2 : Hierarchical Design: Adder / ALU

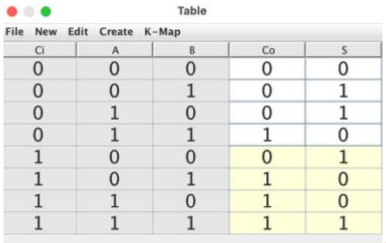


1. Make new circuit for half adder as: sum = A ⊕ B (A XOR B) , Carry = A · B (A AND B).
2. Save as "**halfAdder.dig**".
3. Create new empty file and save as "**fullAdder.dig**".

4. Make full adder from two half adder:
    4.1. Added to the new circuit via the *Components→Custom* menu→half_adder.
    4.2. Check the resut by "play"
5. Test case:
    5.1. Add Componets -> Misc. -> Test
    5.2. Right click **Test** on and select **Edit Detached**.
    5.3. Add test data as below teabke and click on Run Tests to see the result



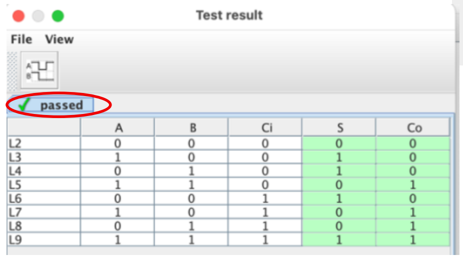
```
A B Ci S Co
0 0 0  0 0
1 0 0  1 0
0 1 0  1 0
1 1 0  0 1
0 0 1  1 0
1 0 1  0 1
0 1 1  0 1
1 1 1  1 1
```



Table

File New Edit Create K-Map

Ci	A	B	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



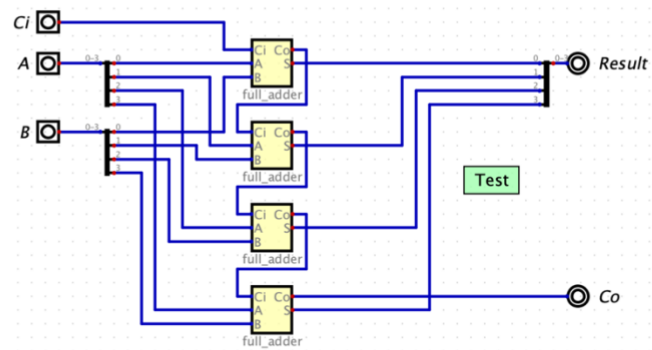
Test result

File View

passed

	A	B	Ci	S	Co
L2	0	0	0	0	0
L3	1	0	0	1	0
L4	0	1	0	1	0
L5	1	1	0	0	1
L6	0	0	1	1	0
L7	1	0	1	0	1
L8	0	1	1	0	1
L9	1	1	1	1	1

    5.4. The result should be pass
    5.5. Save the citcuit.
6. **4-bit adder.** Create ripple-carry adder by make "**rcAdder.dig**" from full adder
    6.1. Add four full_adder subcircuits (Compoments -> Custom -> full_adder).
    6.2. Add Input , label **A** and **B** , set **Data Bits** to **4.**
    6.3. Add two Splitter/Merger from Components -> Wires
    6.4. Set Splitter/Merger as Input Splitting:4 and Output splitting 1, 1, 1, 1
    6.5. Connect as Diagram
    6.6. Add Output label **Result,** set **Data Bits** to **4.**
    6.7. Add Splitter/Merger for output and set to Input 1, 1, 1, 1 Output 4.
    6.8. And Output for **Co**
    6.9. Load "**test_adder.txt**" and add in Test script, circuit must pass all test.
    6.10. Save the circuit.



TA Checking: Inputting various 4-bit numbers and showing the correct sum and final carry-out. ____________________

Lab 10.3 : Building the Final 4-bit ALU

1. Our ALU is simple: it adds two numbers and checks if the result is zero.
2. Create a new circuit save file as "**miniALU.dig".** Place your 4-bit adder sub-circuit (**rcAdder.dig)**.
3. Connect two 4-bit inputs, A and B, to the adder. The output is 4-bit label **Result**.
4. Add Compoments -> Wires -> Constantant Value. Set to **0** and connect to **Ci.**
5. **Zero Flag Logic**: We need to check if all 4 bits of the ALU_Result are zero.
    5.1. Use a splitter to separate the 4-bit ALU_Result (Input: 4, Output: 1, 1, 1, 1).
    5.2. Feed all 4 bits into a 4-input **NOR** gate (change Inumbers of inputs in properties to 4).
    5.3. The output of this **NOR** gate is your Zero_Flag. It will be 1 only when all input bits are 0.
6. **Flag Latching:**
    6.1. Add a Register component from Components -> Memory -> Register.
    6.2. In the Register's properties, set Data Bits to **1**.
    6.3. Connect the output of the 4-input **NOR** gate to the **D** input of this register.
    6.4. Add a new 1-bit Input to your ALU circuit and label it **FlagWrite**. Connect it to the en (enable) pin of the register. The flag will only be updated when this is high.
    6.5. Add another new 1-bit Input and label it **Clock**. Connect it to the C (clock) pin of the register.
    6.6. Add a 1-bit Output component and label it **Zero_Flag**.
    6.7. Connect the Q output of the register to this **Zero_Flag** output.
7. Save the circuit.

TA Checking: Demonstrate the complete latching mechanism of your ALU's Zero Flag. ___________________________

1. Set for Zero: Input values that result in zero (e.g., A=5, B=11 for 5 + (-5) in 4-bit two's complement). Show that the combinational logic (the NOR gate output) is 1.
2. Latch the '1': With FlagWrite set to 1, advance the Clock by one tick. Show that the final Zero_Flag output is now latched and holding the value 1.
3. Hold the '1': Change the inputs to a non-zero result (e.g., A=2, B=3). Show that the NOR gate output is now 0.
4. Demonstrate Unlatch Condition: With FlagWrite set to 0, advance the Clock. Show that the Zero_Flag output *remains* 1, proving that the register is correctly holding the previous value and ignoring the new input.
5. Latch the '0': Set FlagWrite back to 1 and advance the Clock. Show that the Zero_Flag output now updates to 0.

Lab 10.4 : Register File

1. Create the Register File Circuit:
   - 1.1. Create a new file and save it as "**registerFile.dig**".
   - 1.2. Add a Register File component from Components -> Memory - RAM -> Register File.
   - 1.3. Configure Properties: Right-click the component and set:
     - 1.3.1. Data Bits: 4
     - 1.3.2. Address Bits: 3 (3 bits can address 8 unique registers, R0 to R7).
   - 1.4. Add and connect input: : 4-bit to **Din** (data in), 1-bit to **we** (write enable), 3-bit **Rw** (register to be written), 1-bit to **C** (clock), 3-bit to **Ra** and **Rb**.
   - 1.5. Add and connect output: 4-bit to Da and 4-bit to Db
   - 1.6. Click "Play" for simulation. Right click to see the data.

TA Checking:

1. **Write to R2:**_________________________________
   - 1.1. Set we (Write Enable) to **1**.
   - 1.2. Set Rw (Write Address) to **10** (binary for 2).
   - 1.3. Set Din (Data In) to **0111** (binary for 7).
   - 1.4. Advance the Clock one tick. The value 7 is now stored in R2.
   - 1.5. Right click on register file to see data.
   - 1.6. **Disable Write:** Set we back to **0**.
2. **Read from R2 on Port A:** Set Ra (Read Address A) to **10**. Show that the Da output is now **0111**. ________________
3. **Read from R2 on Port B:** Set Rb (Read Address B) to **10**. Show that the Db output is also **0111**. ________________
4. **Verify Independence:** Write a different value (e.g., 5) to a different register (e.g., R1, address 01) and show you can read it back without affecting the value in R2. ____

Lab 10.5: Program Counter and Memory (ROM and RAM)

Part A: Program Counter (PC)

1. Create the PC Circuit:
   1. Create a new file, "**programCounter.dig".**
   2. Add a Counter with preset from Components -> Memory -> Counter with preset.
   3. Configure Properties: Set Data Bits to 4.
   4. Add Output: **out**: 4-bit Output. This is the current address sent to the Program ROM.
   5. Connect the counter's **en** (enable) pin to a **Supply voltage** (Components -> Wires -> Supply voltage) to ensure it's always enabled to count.
   6. Connect the **dir** pin to a **Ground component** (Components -> Wires -> Ground).
   7. Add input : 1-bit to **C**, 4-bit to **in**, 1-bit to **ld**, 1-bit to **clr**.
   8. Ass output: 3-bit to **out**.
   9. Save the circuit.

TA Checking:

1. Output count form 0 to 7 and back to 0 on every **clock** tick. ____
2. Set data **in** and **ld** = 1 and clock tick, output the data in.____
3. Clear to 0 when clr is 1 and clock tick.____

Part B: Program ROM

1. Create the ROM Circuit:
    1. Create a new file, "**programROM.dig**".
    2. Add a ROM component from Components -> Memory -> ROM.
    3. Configure Properties:
        - Address Bits: 3
        - Data Bits: 8
    4. Connect the 3-bit address Input to the **A** pin of the ROM.
    5. Connect the 8-bit instruction Output to the **D** pin of the ROM.
    6. Connect the **sel** (select) pin to a Supply voltage component (Components -> Wires -> Supply voltage).
    7. Load Program: Right-click the ROM component and select "Edit Content". Enter the 8-bit instructions.
    8. Save the circuit.

TA Checking: Load instruction into ROM, set address to see the correct output. ________________________________

Part C: Data RAM

1. Create the RAM Circuit:
    1. Create a new file, "**dataRAM.dig**".
    2. Add a RAM, separated Ports component from Components -> Memory - RAM -> RAM, separated Ports.
    3. Configure Properties:
        - Address Bits: 3
        - Data Bits: 4
    4. Inputs: Addr (Address): 3-bit to **A**, **Din** (Data In): 4-bit, **str** (Store): 1-bit, **ld** (Load): 1-bit Input. **C** (Clock): 1-bit.
       Output: Dout (Data Out): 4-bit Output.
    5. Click "Play" to simulate the RAM. Right click to see the data

TA checking: Store and read data from RAM : ____________________________

1. To store data: set address to Addr, set data **Din**, **MemWrite** = High, **clock** tick
2. To load data: set address to Addr, **MemRead** = High, **clock** tick

---

Lab 10.6: Multiplexer

A Multiplexer selects exactly one of N data inputs and forwards it to the output based on a selector (sel).

1. Create new file. Add multiplexer on canvas (Compoments -> Plexers -> Multiplexer).
2. Set data bits = 4, Selector bits = 2. This will create multiplexer with 4 4-bit input (in_0, in_1, in_2, in_3) and 2-bit selector 00 output from **in_0**, 01 output from **in_1**, 10 output from **n_2** and 11 output from n_**4**.
3. Add 4-bit input to **in_0**, **in_1**, **in_2**, **in_3**.
4. Add 4-bit output to **out**.
5. Set difference data for each input, set selector and observ the result.

TA checking: ______________________________________

LAB Submission

https://docs.google.com/forms/d/e/1FAIpQLSdFhcWXlzpc5v1fWzFxQ9mqXijq-1aHaBRmQkgNlX_Dh2eLJQ/viewform?usp=header