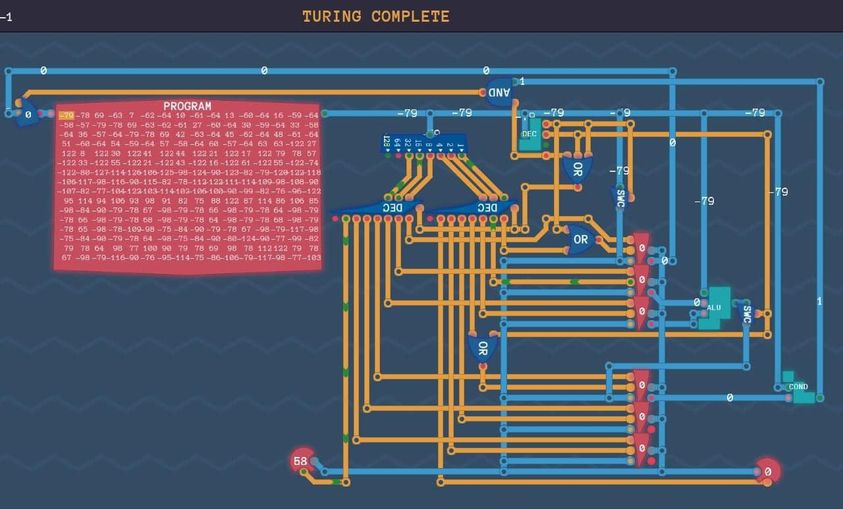
Một chiếc 8-bit CPU đơn giản có thể thực hiện các phép tính(8-bit ALU), điều kiện (COND) và ghi nhớ kết quả vào 6 thanh ghi REG(0-5).

Chương trình (Program) đc nạp vào là 1 dãy các dữ liệu đc giải mã ( qua Instruction Decoder ) và gửi yêu cầu tới các module/component, trong mỗi cycle clock.

Việc thực thi tuần tự chương trình đc Program-Counter đảm nhiệm qua việc lưu trữ vị trí của instruction đang đc xử lý & tự động +1 để nạp mã lệnh tiếp theo vào Decoder (DEC).



32-bit Registers lưu đc nhiều cỡ nào ?

Tương đương 1 ~ 4 chữ cái mã hoá dạng UTF-8 (vd: 00001011), hoặc 1 số nguyên 32-bit (4,294,967,296) viết xong là hết bộ nhớ 🤣

Cpu Arm các b hay dùng có khoảng 37 thanh ghi 32-bit như thế để truy xuất trước khi phải xờ tới RAM. Lý do cần tối ưu Heap vs. Stack.

\*\* Ảnh : Logic Design để truy xuất 4 thanh ghi Register theo dạng flag A0-B0-A1-B1 & tín hiệu đọc/ghi.

Giả lập trên game mình đã share 🧑‍🍳nhưng cũng có thể vẽ lại sang mọi sim khác.

\*\* Verilog code \*\*

module Register32(

input wire clk,

input wire reset,

input wire load,

input wire [1:0] sel,

// 2-bit selector to choose among A0, B0, A1, B1

input wire [7:0] data\_in,

// 8-bit data input

// 4x 4-bit registers

output reg [7:0] A0,

output reg [7:0] B0,

output reg [7:0] A1,

output reg [7:0] B1

);

always @(posedge clk or posedge reset) begin

if (reset) begin

// Reset all registers to 0

A0 <= 8'b0;

B0 <= 8'b0;

A1 <= 8'b0;

B1 <= 8'b0;

end else if (load) begin

// Load data into the selected register

case (sel)

2'b00: A0 <= data\_in;

2'b01: B0 <= data\_in;

2'b10: A1 <= data\_in;

2'b11: B1 <= data\_in;

endcase

end

end

endmodule

\*\* Giải thích \*\*

Inputs:

- clk: chân Clock

- reset: Reset tín hiệu để khởi tạo thanh ghi.

- load: tín hiệu cho phép nạp dữ liệu vào thanh ghi đã chọn.

- sel: 2-bit selector để chọn thanh ghi 8-bit nào trong số (A0, B0, A1, B1) để nạp dữ liệu vào.

- data\_in: đầu vào dữ liệu 8-bit

2. Outputs:

A0, B0, A1, B1: 4 thanh ghi 8-bit .

3. Flow chạy :

Khi có tín hiệu khởi tạo từ clk hoặc reset pin :

- Nếu reset pin = high => mọi thanh ghi (A0, B0, A1, B1) đc trả về 0.

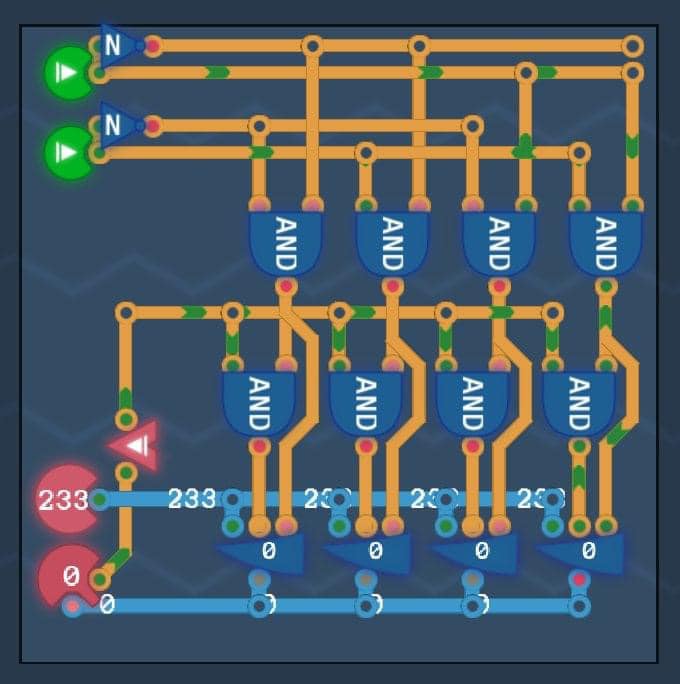
- Nếu load pin = high, dữ liệu từ data\_in đc nạp theo register đc sel selector chọn :

sel = 2'b00: nạp vào A0.

sel = 2'b01: nạp vào B0.

sel = 2'b10: nạp vào A1.

sel = 2'b11: nạo vào B1.



Dạo này mình mới biết 1 game trên Steam : Turing Complete ([https://store.steampowered.com/app/1444480/Turing\_Complete/](https://store.steampowered.com/app/1444480/Turing_Complete/?fbclid=IwZXh0bgNhZW0CMTAAAR3zMs8HqY_kM4oSK1rogcp5Olq4o50N58w_SfdXTFfqzqvXszKmecloWoM_aem_ZmFrZWR1bW15MTZieXRlcw)), có thể giả lập lại các Digital Logic Design khá tốt. Có thể vừa chơi - vừa học cách thiết kế từ các linh kiện cơ bản ( như NAND ) đến phức tạp ( CPU, RAM, Register, ALU, Bus... ). Đây là trò chơi đã khiến mình đi sâu vào tìm hiểu về FPGA cũng như ngành thiết kế vi mạch.

Game khá mượt, chỉ có điểm yếu là cách giải thích yêu cầu mỗi Level hơi khó hiểu và tối nghĩa, nhưng bù lại luôn có hướng dẫn cách giải =))

Theo mình thì đây là 1 công cụ hữu ích khi học Verilog/VHDL mà thấy khó tiếp cận vì bản thân các sim/tool chưa trực quan. Tương lai sẽ sớm có chức năng export -> verilog để test đc trên FPGA =))

\*\* Ảnh #1: Level thiết kế mạch copy data qua lại 6 Register.

\*\* Ảnh #2: Thiết kế CPU Intel 8008 ([https://www.youtube.com/watch?v=21bQBCfMDb8&t=1159s](https://www.youtube.com/watch?v=21bQBCfMDb8&t=1159s&fbclid=IwZXh0bgNhZW0CMTAAAR2bE6ly2977hv_AKyMBXeR-xv536gY8qoCu7DJI9a2O9K6a88xQ2fCam88_aem_ZmFrZWR1bW15MTZieXRlcw))

\*\* Bonus : ngoài ra thì open-source logic-sim mà hiện mình đang thấy làm tốt nhất có thể kể đến logisim-evolution ( [https://github.com/logisim-evolution/logisim-evolution](https://github.com/logisim-evolution/logisim-evolution?fbclid=IwZXh0bgNhZW0CMTAAAR1M0HSOWPMl32Zx8yVae89i-vhg-9AUH6Hj8VXLcNKpOO-RPCVMPTFfzIs_aem_ZmFrZWR1bW15MTZieXRlcw)), export thẳng design -> verilog cho FPGA phổ biến như Basys3.

