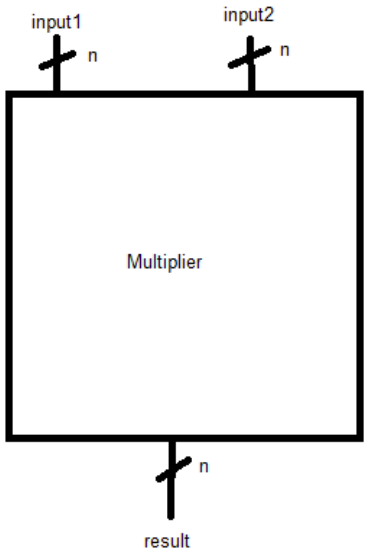
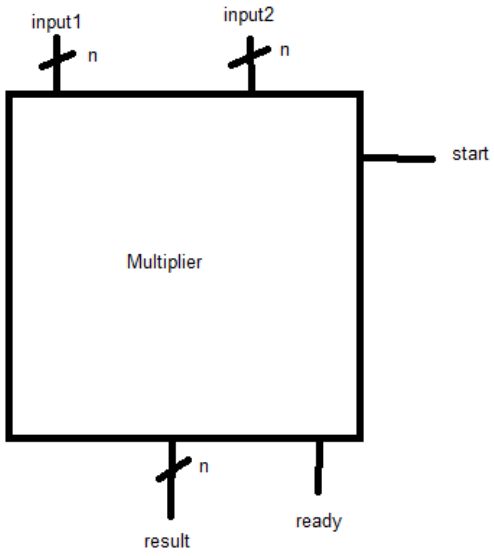
علی نظری ۹۹۱۰۲۴۰۱ تمرین سری پنجم درس DSD

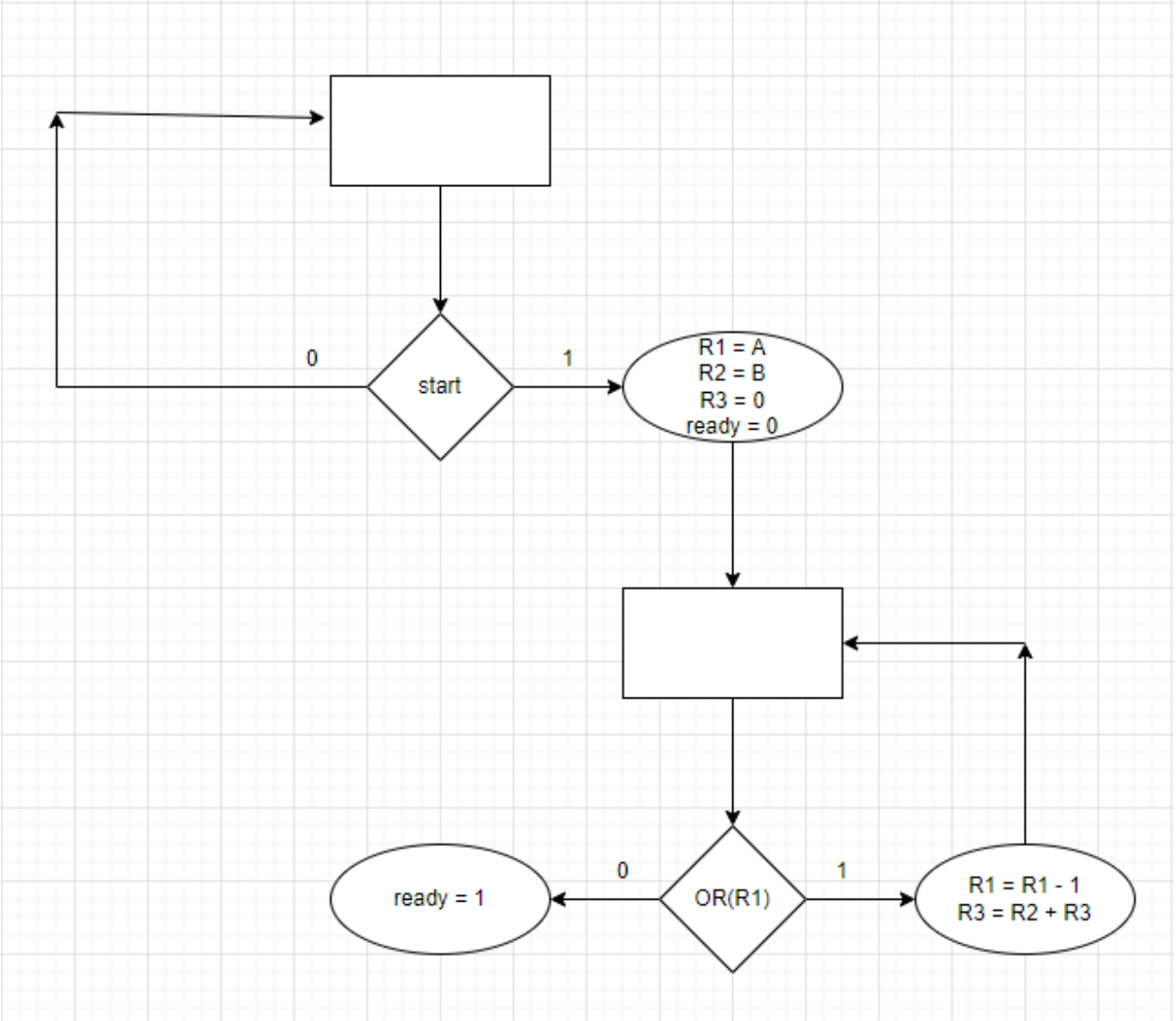
نخست باید ورودی ها و خروجی های مدار را مشخص کنیم که به شکل زیر است:



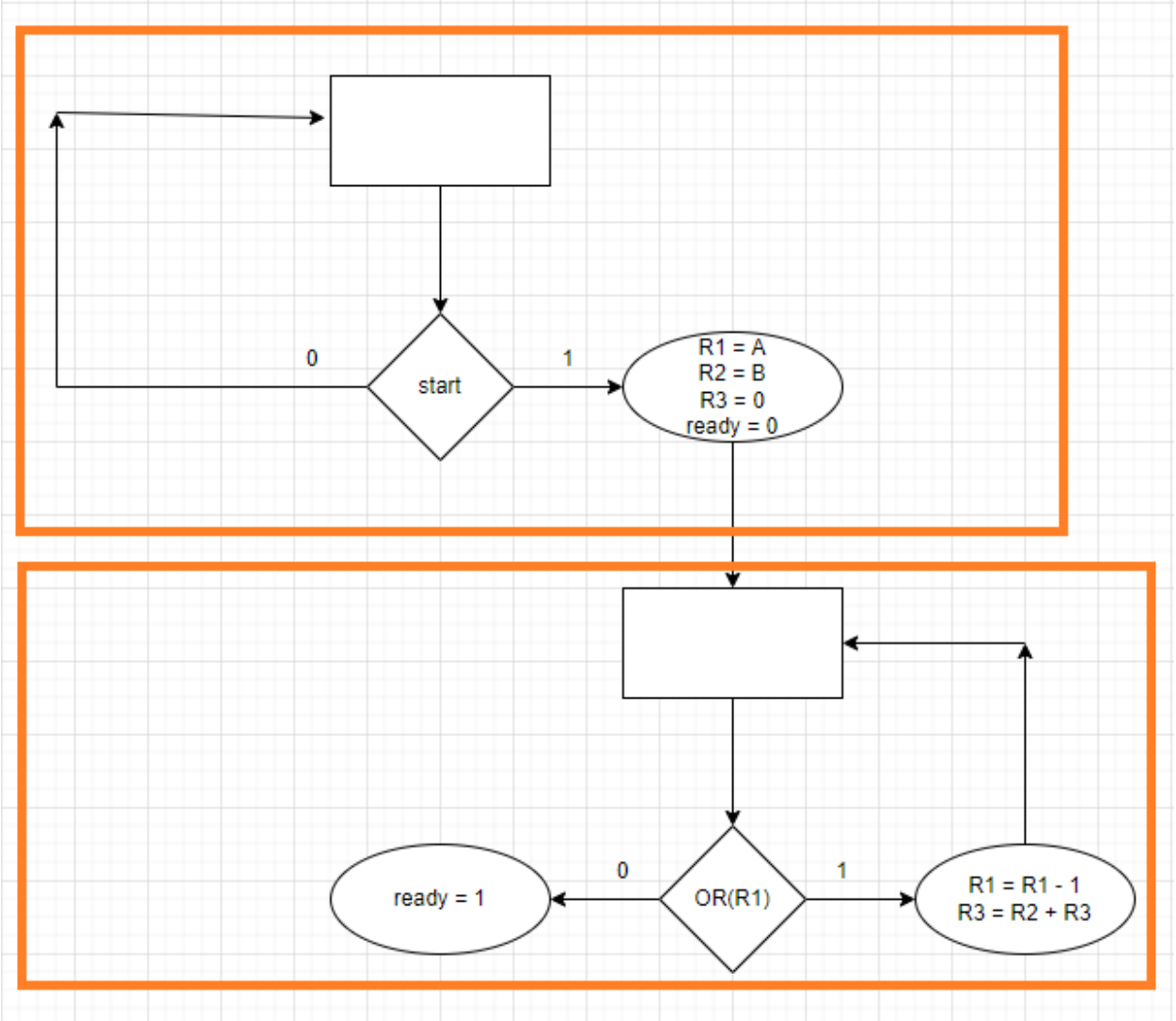
این مدار ترتیبی است چون کار در یک کلاک انجام نمی شود و نیاز است که کلاکی وجود داشته باشد تا مراحل مختلف را نشان دهد. از آنجا که خروجی هم درجا آماده نمی شود، پس یک بیت flag مانند هم نیاز داریم تا مشخص کند نتیجه آماده شده است یا نه که آن را با ready مشخص می کنیم. نیاز به یک دکمه Start هم هست تا مدار بداند چه زمانی کارش را شروع کند.



حال باید طراحی ASM Chart را آغاز کنیم. در این ASM ما ۲ رجیستر برای ورودی ها می خواهیم و یکی هم برای خروجی. Ready را هم با رجیستر مشخص می کنیم ولی start نیازی نیست register باشد و با همان wire آن را مشخص می کنیم. همانطور که سر کلاس گفته شد هم از عملگر تک اپرندی OR(R1) استفاده میکنیم که در واقع کل بیت های R1 را با هم OR می کند و صفر یا یک به ما می دهد به عنوان نتیجه. البته هم OR(R1) می توان استفاده کرد و هم R1 == 0. چون اعداد بدون علامت هستند، پس می توان ساده سازی هایی را در مدار اعمال کرد و در ASM که جلوتر گذاشته می شود هم توضیح داده می شود.



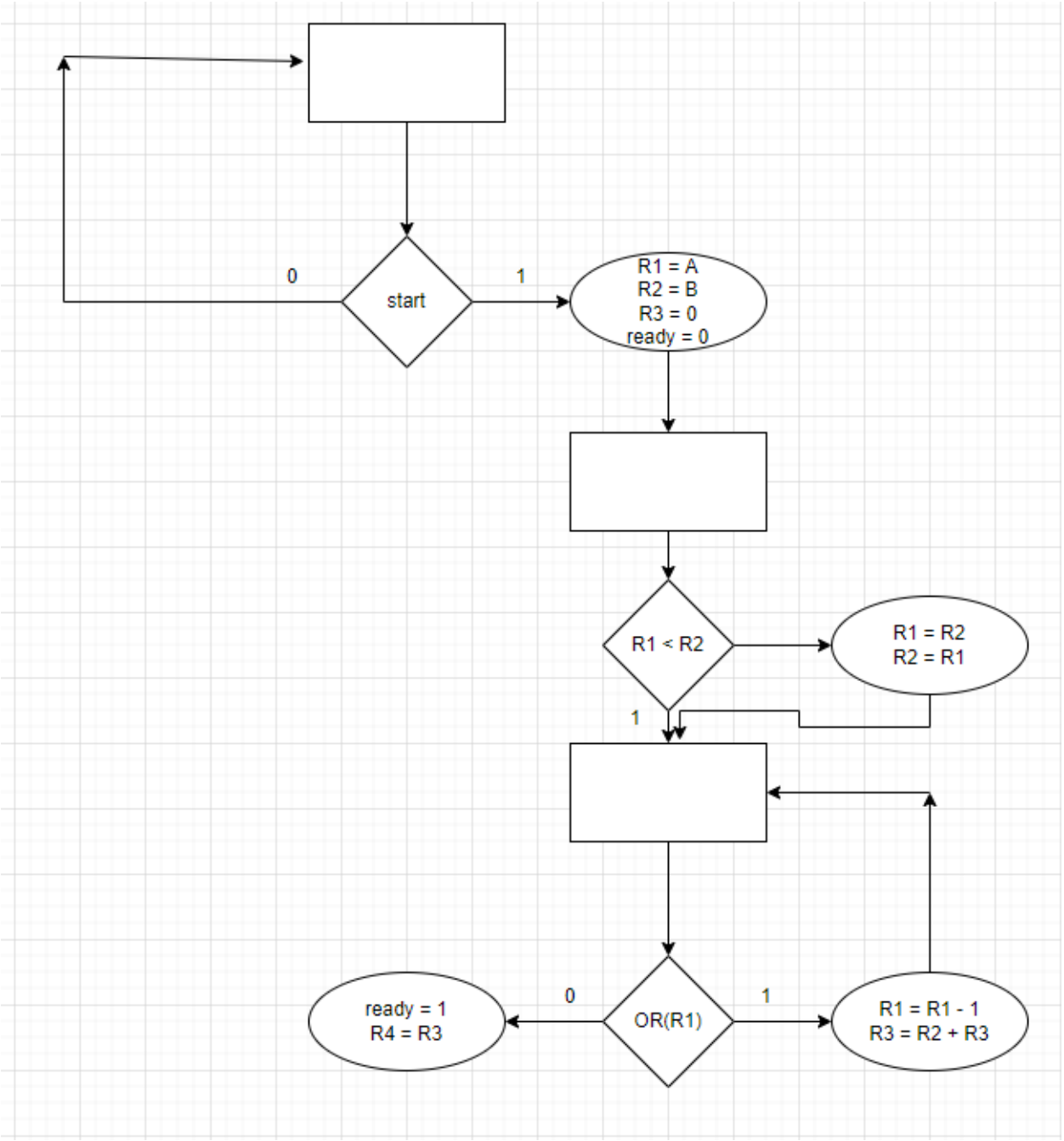
پس حالت اولیه ASM مانند شکل بالا می شود. بعد از یک شدن ready هم می توانیم به state اولیه برگردیم و منتظر بمانیم تا ورودی های جدید به ما داده شود. این موضوع هم در نظر گرفته شده است که کاربر نباید به رجیسترهای میانی دسترسی داشته باشد هم به خاطر امنیت و هم به خاطر اینکه پین های خروجی توان مصرفی بسیار بالایی دارند.
برای شبیه سازی، می توان از حالت cycle استفاده کرد که مانند زیر می شود و مثلا می خواهیم ضرب ۳ ضرب در ۲ را انجام دهیم. در این مدار خب ۴ تا reg وجود دارد و ready را هم reg گرفتیم. قبل از این، باید ASM Block ها را مشخص کنیم که به شکل زیر است:



اسم بالایی را init می گذاریم و اسم دومی را mult.

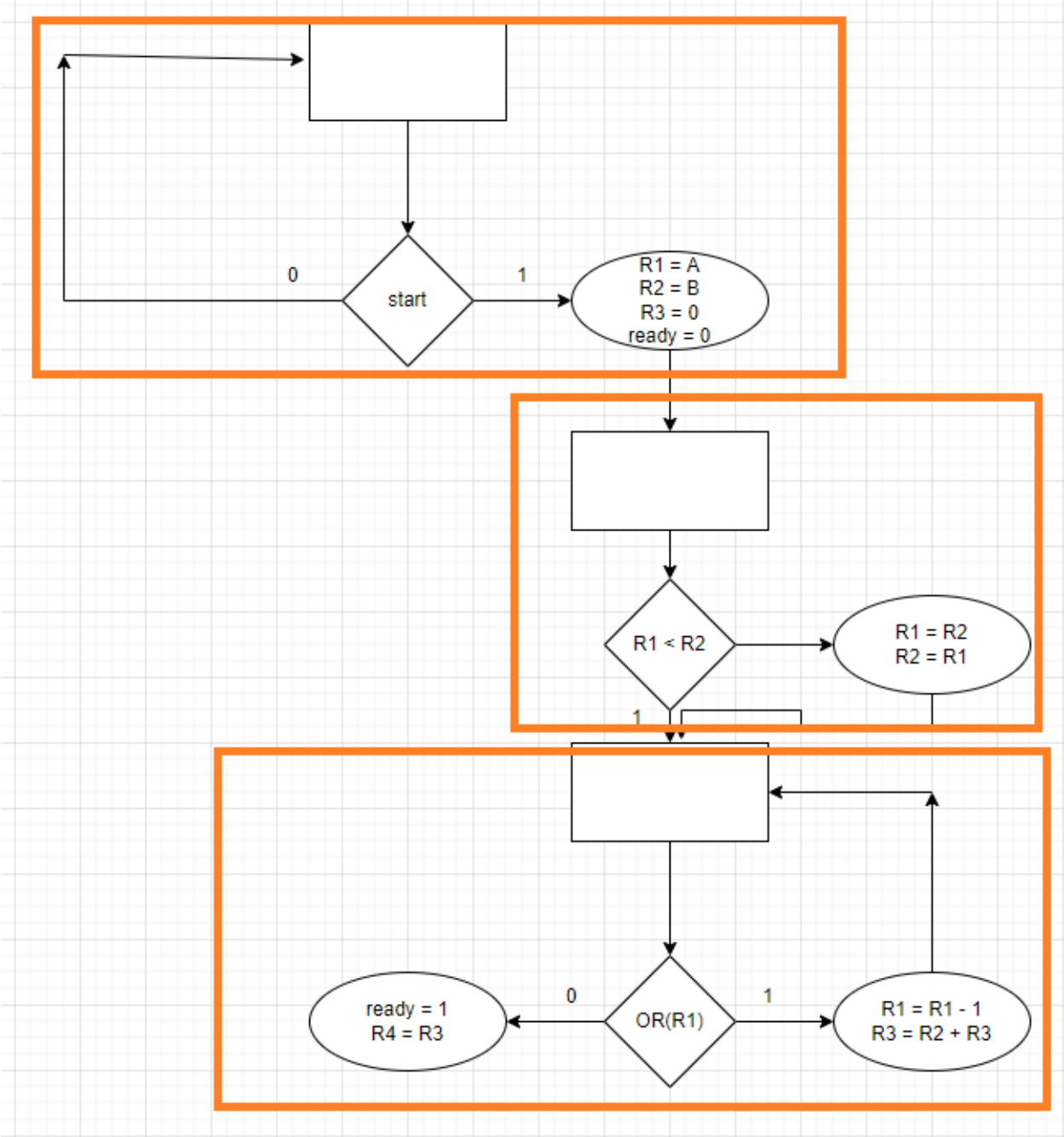
لبه کلاک	R1	R2	R3	R4	ready	ASM block
1	-	-	-	-	1	init
2	3	2	0	-	0	mult
3	2	2	2	-	0	mult
4	1	2	4	-	0	mult
5	0	2	6	-	0	mult
6	0	2	6	6	1	init

حال می فهمیم که اگر مقدار کمتر در R1 باشد بهتر است. چون تعداد بار کمتری نیاز است که ASM Chart را بپیماییم. پس ASM مانند شکل زیر می شود:

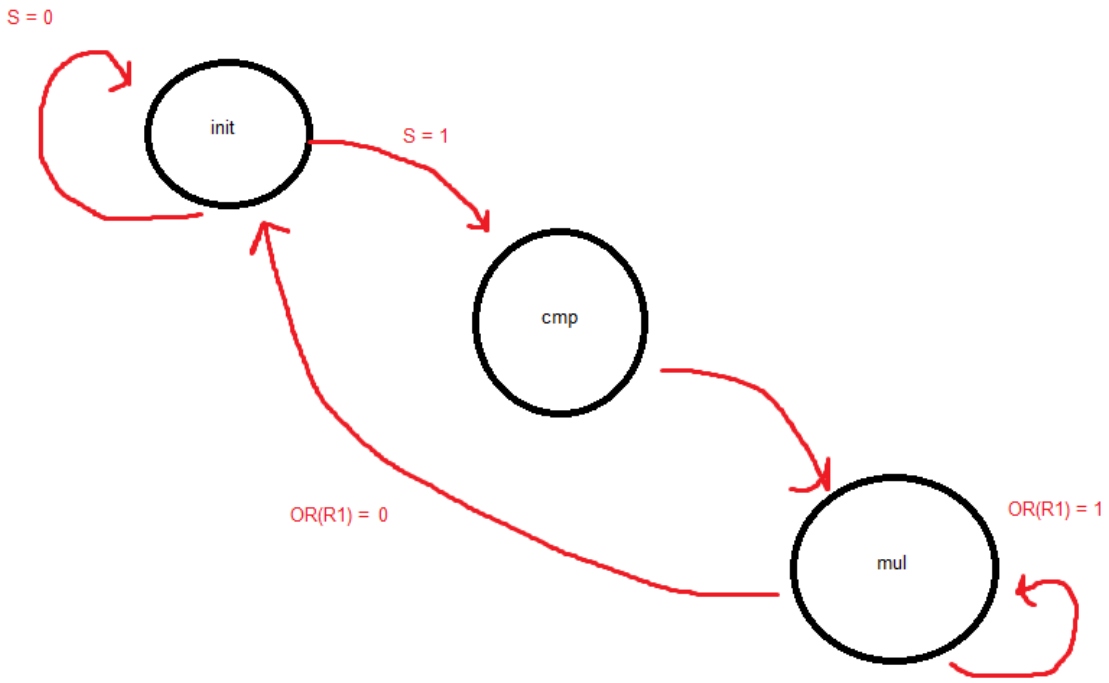


و این نکته را هم می دانیم که همزمان می توانیم هم در reg بنویسیم و هم از reg بخوانیم. حال طراحی کامل شده است و می توان بخش data path و control unit را طراحی کرد. می دانیم که data path مسئول دریافت ورودی و اعمال عملیات های مختلف است و cu کارش این است که عملیات و ترتیب آن ها را مشخص کند. تمام text های داخلی مربوط به data path است و شکل کلی asm chart هم مشخص کننده control unit است. حال مرحله بعدی این است که سخت افزار مورد نیاز برای این بخش Data path را به دست آوریم. می دانیم که هر reg قابلیت هایی را به شکل پیش فرض دارد که عبارت اند از increment کردن و decrement کردن و load و reset کردن. ما ۴ تا

رجیستر R1 تا R4 داریم و ready را هم reg نمیگیریم و FF می گیریم تا سخت افزار کمتری مصرف شود و Start هم که کلا wire است. مورد بعدی، مقایسه کننده است که نیاز ما می شود. یک adder هم می خواهیم. یک or کننده تک اپرندی هم نیاز است. حال اگر دقت کنیم، برخی از reg ها از دو ورودی در طول ASM Chart می آیند مثل R2 که یکبار از ورودی دوم می آید و یکبار هم از R1 پس باید بتوان بین این ها انتخاب کرد. پس به mux هم نیاز داریم. پس اینگونه resource های مشترک را هندل کرده ایم. در این حالت هم کلا ۲ تا mux می کنیم چون ۲ تا از reg های ما، common resource دارند. البته چون سنتز دستی در سوال خواسته نشده، از وارد شدن به جزئیات خودداری میکنم.
حال به بخش control unit می رسیم که برای طراحی آن، باید اول از همه ASM Block های موجود در شکل نهایی را مشخص کنیم.



نامگذاری هم به ترتیب همان init و cmp و mul می گذاریم. حال نمودار حالت مربوطه را باید بکشیم. هر بلاک را با دایره مشخص می کنین و مهم است که بگوییم با عوض شدن چه چیزی می توانیم از هر حالت به دیگری برویم.



سپس از طراحی هافمن استفاده می کنیم و کد بخش cu را می زنیم.

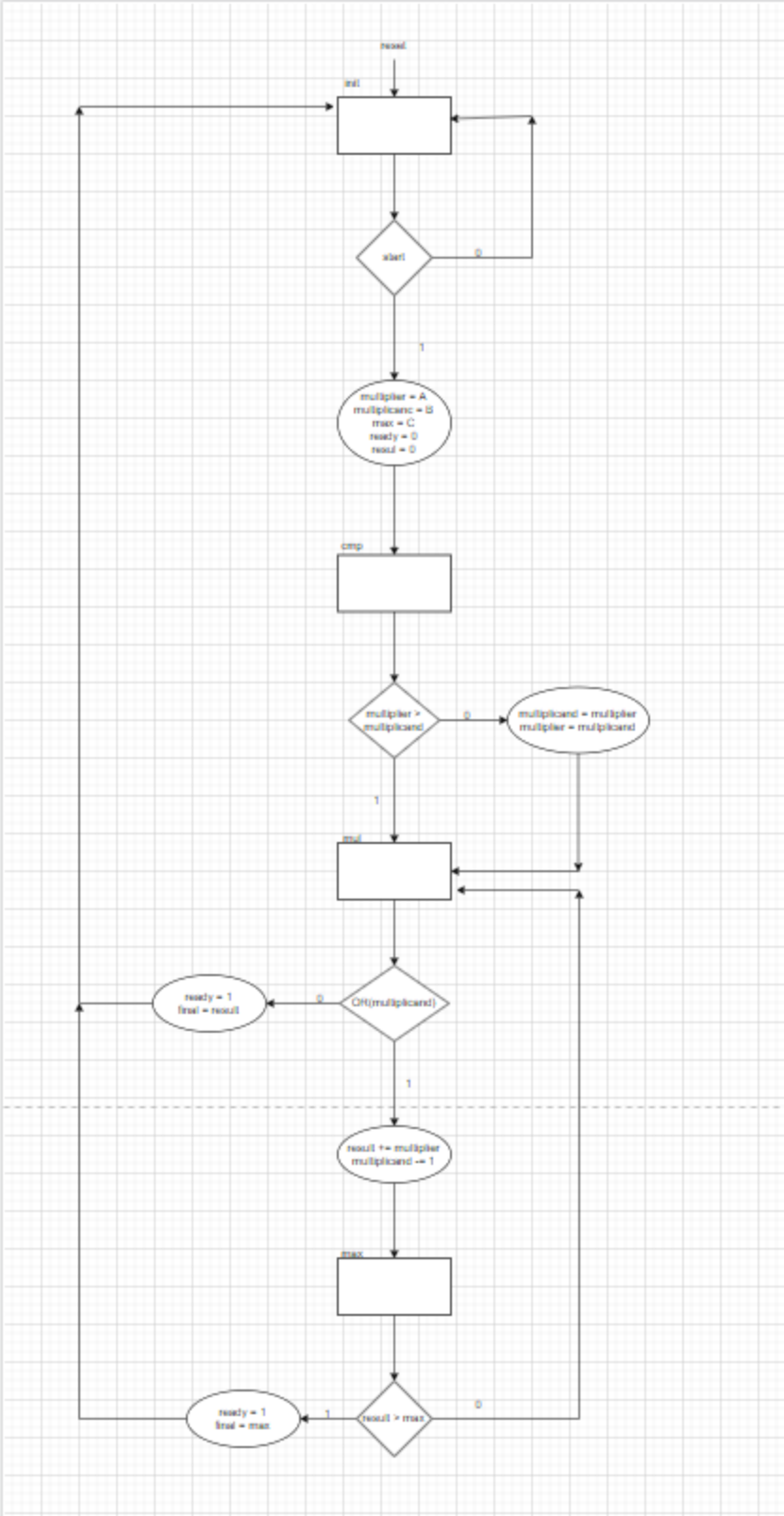
```
module CU(input S, OR_R1, CMP_L_R1, output reg L_R1, L_R2, L_R3, L_R4, R_R3,
Dec_R1, Sel_R1, Sel_R2, S_R, R_R);
        reg[1:0] p_state, n_state;
        localparam[1:0] init=1'b00, cmp=2'b01, mul=2'b10;
        always @(p_state or S or OR_R1 or CMP_L_R1)
        begin
                n_state = init;
                case (p_state):
                        init:
                                begin
                                end
                        cmp:
                                begin
                                end
                endcase
        end

        always @(posedge clk)
```

```
    begin
        if (rst) p_state = init;
        else p_state = n_state;
    end
endmodule
```

حال تمام این ها نسخه اولیه از برنامه بود و حال مقایسه با C را هم باید وارد کار کنیم که از این به بعد، به این مورد می پردازیم و Datapath را هم طراحی می کنیم که تا به حال این کار را انجام نداده ایم.

اول ASM Chart را کامل می کنیم که به علت بزرگ شدن آن، در یک صفحه نتوانستم عکس بندازم ولی فایل آن موجود است با استفاده از سایت Draw.io که نیازی به ثبتنام ندارد می توانید کامل آن را مشاهده کنید ولی کلیت آن به این شکل است:



حال دوباره باید سخت افزار های ممکن را مشخص کنیم:

مقایسه کننده

جمع کننده
و mux هم که به خاطر همان بحث common resource می خواهیم
همان OR کننده تک اپرندی
رجیستر با امکانات increment و decrement و reset و load
حال این سخت افزار ها را به شکل رفتاری باید پیاده سازی کنیم:
کد بخش or کننده تک اپرنده به شکل زیر است:

```
module DO_OR (input wire [31:0] in,output wire result);
	assign result = |in;
endmodule
```

برای رجیستر ها هم به ۳ نوع رجیستر نیاز داریم که یکی معمولی است و یکی هم با قابلیت inc و dec و یکی هم با قابلیت reset که فرض می کنیم این رجیستر ها همگی رجیسترهایی با تمام قابلیت های inc و dec و load و reset هستند:

```
module register (input wire clk,input wire load,input wire [31:0] data,output reg [31:0]
out,input wire dec, input wire reset);

	always @(posedge clk)
	begin
		if (load)
			out <= data;
		else if (dec)
			out <= out - 1;
		else if (reset)
			out <= 0;
	end

endmodule
```

حال مقایسه کننده را طراحی می کنیم:

```
module Comparator (input wire [31:0] a,input wire [31:0] b,output wire gt);
	assign gt = a > b ? 1 : 0;
endmodule
```

حال جمع کننده:

```
module Adder (input wire [31:0] a,input wire [31:0] b,output wire [31:0] out);
	assign out = a + b;
endmodule
```

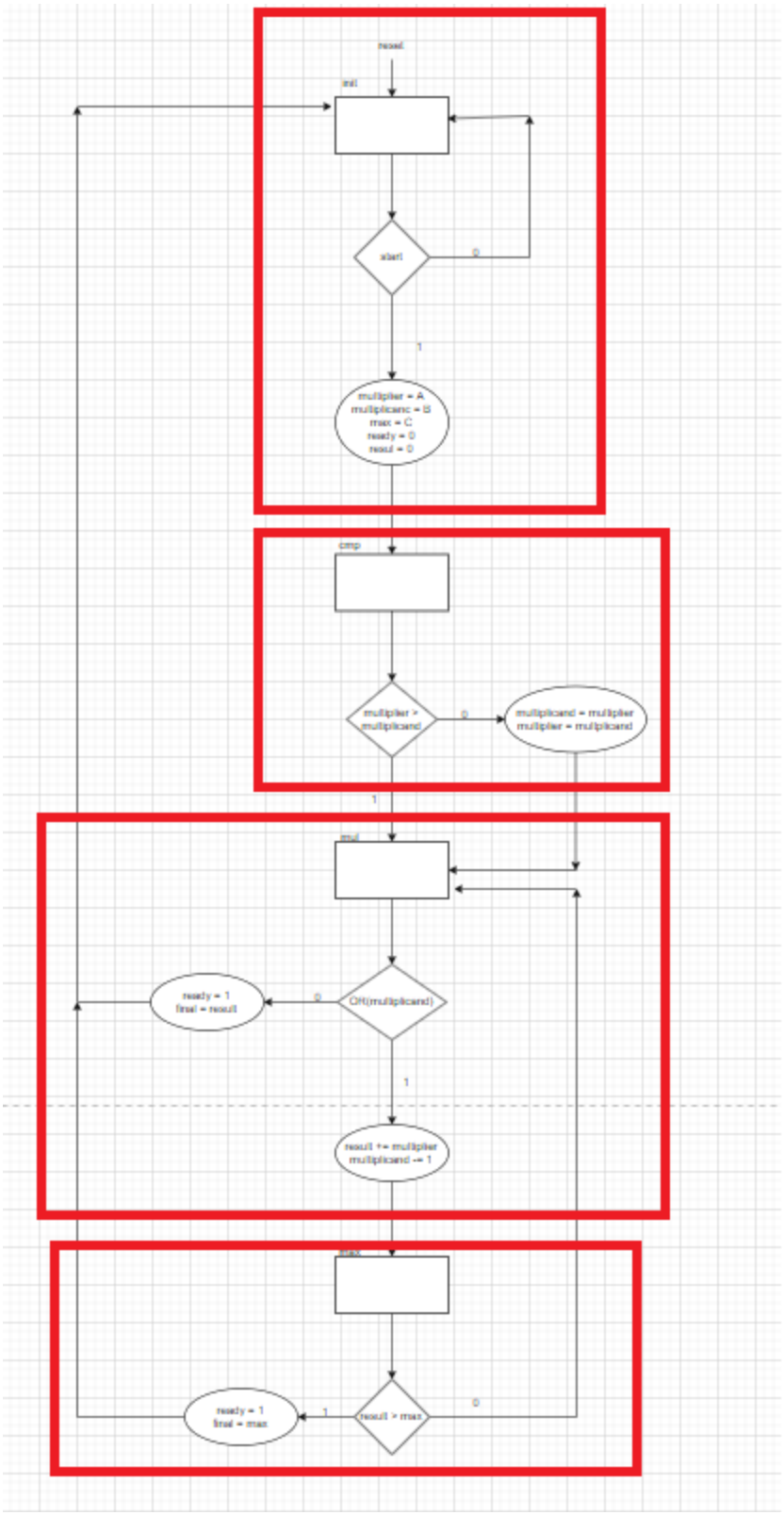
در نهایت هم mux را طراحی می کنیم:

```
module Mux (input wire [31:0] a,input wire [31:0] b,input wire select ,output wire [31:0] out);
	assign out = select ? b : a;
endmodule
```

حال Datapath را باید طراحی کنیم که در این مورد تنها باید از ماژول های بالا instance بگیریم و سیگنال های کنترلی را به آن بدهیم و همچنین سیگنال های status را هم خروجی آن ها بگذاریم تا بتوانیم به cu بدهیم تا بتواند تصمیم بگیرد. کد این بخش مانند زیر است:

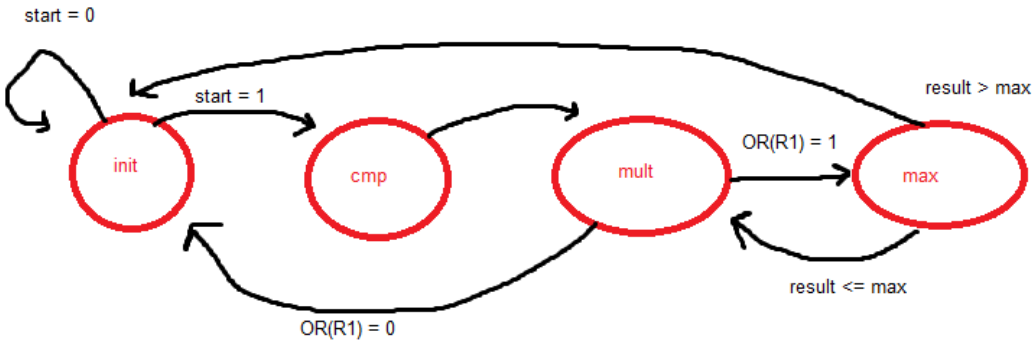
```
module Datapath (
input wire clk,input wire [31:0] first_operand , , load_controller , multiplicand_controller , decrement_controller, [31:0] max_controller, select_controller , multiplicand_select , reset_result, [31:0] second_operand, load_result, load_controller, load_final , final_select ,output wire multipicant_or , result_gt_max, multiplier_gt_multiplicand, [31:0] final);
	wire [31:0] the_result_1 , main_input_1 , main_result_1, main_input_2, max, result_data, adder_result, final_register_input;
	register multipicant(clk, multiplicand_controller ,decrement_controller , main_input_1 , the_result_1);
	register max_reg(clk, load_controller , max_controller , max);
	register multiplier(clk, load_controller , main_input_2 ,main_result_1);
	register final_register(clk, load_final , final_register_input ,final);
	register result(clk, load_result , reset_result ,adder_result , result_data);
	Mux mux_1(first_operand, main_result_1 , multiplicand_select ,main_input_1);
	Mux mux_2(second_operand, the_result_1 , select_controller ,main_input_2);
	Mux mux_3(result_data , max, final_select ,final_register_input);
	Adder adder(result_data , main_result_1 , adder_result);
	Comparator comtor(main_result_1 , the_result_1 , multiplier_gt_multiplicand);
	Comparator maxComprator(result_data , max,result_gt_max);
	DO_OR or1(the_result_1 , multipicant_or);
endmodule
```

حال به بخش control unit می رسیم که مانند چیزی که در حالت اولی گفته شد، باید Asm block ها را مشخص کنیم که مانند زیر است:



حال باز هم مانند چیزی که در حالت اولیه گفته شد، باید نمودار حالت را بکشیم:

برای اینکه از init به cmp برویم، واضحا باید start برابر با یک شود و اگر صفر بماند، یک طوقه روی خود init داریم. حال از cmp بدون شرطی به mul می رویم. حال اگر در mul دریابیم که OR(R1) همان صفر است، در جا به init إر می گردیم و اگر یک باشد، به بخش max می رویم. حال اگر نتیجه از max کمتر باشد، به mult بر می گردیم وگرنه صاف به init می رویم.



حال از روی این مورد کنترل یونیت را طراحی می کنیم و طراحی دقیقا مانند طراحی خامی است که بالاتر قرار دادیم

```
module ControlUnit (input wire clk,reset ,start ,greater_than_max, main_or,
is_greater,output reg ready ,in_1_l, result_r, in_2_l , final_s, multiplicand_d , ,in_2_s,
max_l ,result_l, in_1_s ,final_l);
	reg [1:0] p_state , n_state;
	reg is_ready = 0;
	localparam [1:0] init=2'b00, cmp=2'b01, mul=2'b10, max=2';
	always @(*) begin
		n_state = init;
		case (p_state)
			init: begin
				if (start) begin
					{in_1_l, in_2_l, in_1_s, in_2_s, max_l, result_r, final_l,
is_ready, final_s} = 9'b110011000;
					n_state = cmp;
				end
			end
			cmp: begin
				result_r = 0;
				if (is_greater) begin
					{in_1_s, in_2_s, in_1_l, in_2_l} = 4'b1111;
					n_state = mul;
				end else begin
					{in_1_l, in_2_l} = 2'b00;
					n_state = mul;
```

```
                end
            end
            mul: begin
                in_1_l = 0;
                in_2_l = 0;
                if (main_or) begin
                    result_l = 1;
                    multiplicand_d = 1;
                    n_state = max;
                end else begin
                    {result_l, multiplicand_d, final_l, is_ready} = 4'b0011;
                    n_state = init;
                end
            end
            max: begin
                result_l = 0;
                multiplicand_d = 0;
                if (greater_than_max) begin
                    is_ready = 1;
                    final_s = 1;
                    final_l = 1;
                    n_state = init;
                end else begin
                    n_state = mul;
                end
            end
        endcase
    end
    always @(posedge clk) begin
        if (reset)
            p_state <= init;
        else
            p_state <= n_state;
        ready <= is_ready;
    end
endmodule
```

پس الان طراحی کا از دو بخش اصلی تکمیل شد و به این ترتیب، طراحی ما کامل می شود و مدار قابل استفاده است.