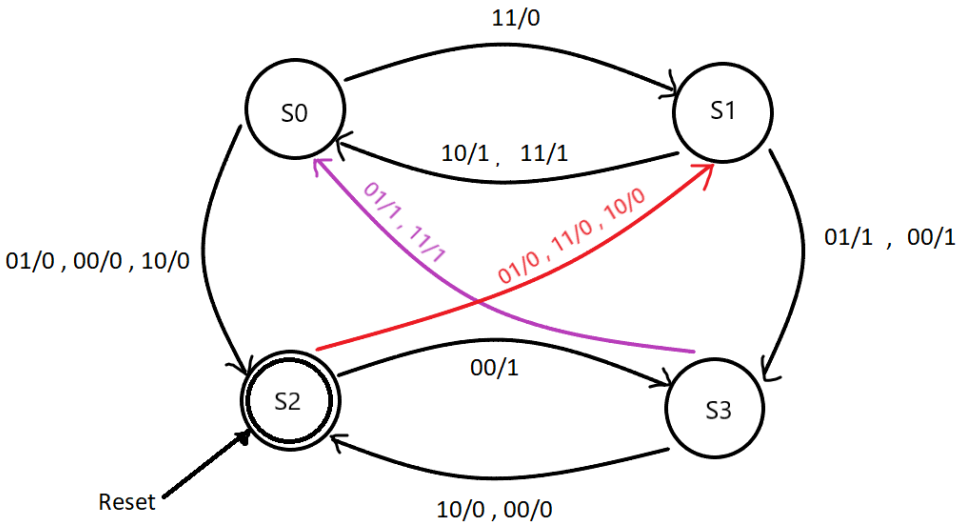
تمرین دوم معماری کامپیوتر

سوال یک)

الف)دیاگرام نشان داده شده ماشین موری را نشان میدهد زیرا خروجی فقط وابسته به وضعیت فعلی است.

ب) از انجایی که دیاگرام اصلی ماشین مور است برای کمترین تغییرات میتوان حالت استیت را روی یال ها به شکل زیر نوشت.

پ.ن: رنگ های مختلف معنی خاصی ندارند و صرفا برای واضح بودن مسیر ها، از رنگ های مختلفی استفاده کردیم.



ج) ماژول مور را می‌سازیم. سیگنال کلاک و ریسیت و عددی به نام num را به عنوان ورودی ماشین تعریف می‌کنیم. عدد num با توجه به دیاگرام، دو بیتی است.

همچنین این ماشین یک خروجی y دارد.

با استفاده از اینام دیتاتایپی به نام statetype می‌سازیم و دو متغیر از نوع statetype با نام های state که نشان دهنده استیت فعلی و nextstate که نشان دهنده استیت ثانویه است می‌سازیم. سپس با استفاده از always_ff، مشخص می‌کنیم تا در لبه بالارونده کلاک، استیت عوض شود و در لبه بالارونده rst، ماشین reset شود.

سپس در always_comb(که می‌دانیم همان at(*) است)، مدار ترکیبی را اجرا میکنیم. در این بلوک ما یک کیس تعریف کردیم که با توجه به استیت فعلی، استیت بعدی را با توجه به شرط های دیاگرام داده

شده مشخص می‌کند. خروجی y, یا ۰ و یا ۱ است. با توجه به دیاگرام فقط زمانی یک است که در یکی از استیت های S0 یا S3 باشیم.

در اخر نیز به کمک نرم افزار کوارتوس دیاگرام این کد را رسم کردیم:)

(فایل کد به در فایل زیپ، پیوست شده است)

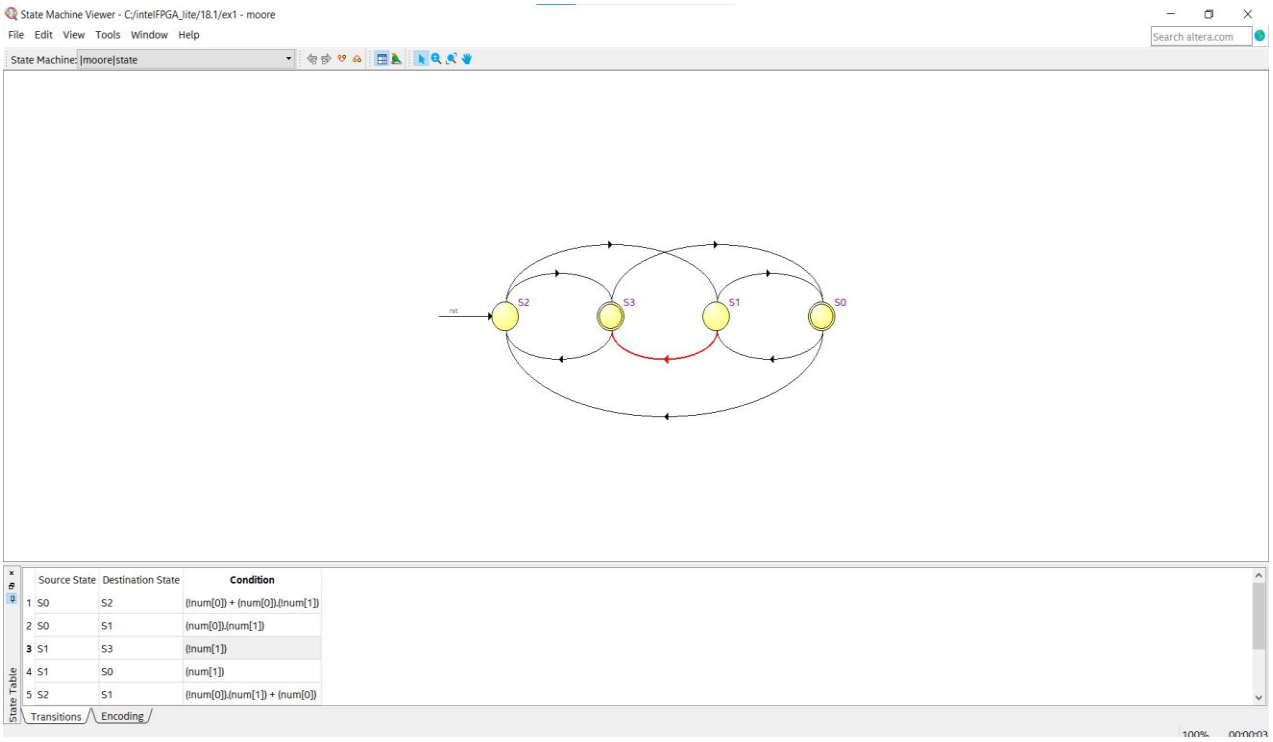
```
module moore(input logic clk,rst , input logic [1:0] num , output logic y);

    typedef enum logic [1:0] {S0, S1, S2, S3} statetype;

    statetype state, nextstate;
    always_ff@(posedge clk , posedge rst)
     if(rst) state <= S2;
     else state <= nextstate;

    always_comb
        case(state)
        S0: if (num == 2'b11) nextstate = S1;
            else nextstate = S2;
        S1: if (num == 2'b11 | num==2'b10) nextstate = S0;
            else nextstate = S3;
        S2: if (num == 2'b00) nextstate = S3;
            else nextstate = S1;
        S3: if (num == 2'b11 | num==2'b01) nextstate = S0;
            else nextstate = S2;
         endcase

    assign y = (state == S0) | (state == S3);
endmodule
```



سوال دو)

الف) ابتدا اعداد را به باینری تبدیل میکنیم.میدانیم هر رقم در مبنای ۸، برابر ۴ رقم در مبنای ۲ است:

۱-$(4B002525)_{16}$= 0100 1011 0000 0000 0010 0101 0010 0101

2- $(5010B66E)_{16}$ =0101 0000 0001 0000 1011 0110 0110 1110

مرتب کردن اعداد بر اساس استاندارد *IEEE754* :

رقم اول در سمت چپ نشان دهنده علامت، هشت رقم بعدی توان و رقم های بعد نشان دهنده بخش اعشاری عدد هستند.

میدانیم که در این استاندارد توان با یک عدد **بایاس** جمع میشود. در سیستم ۳۲ بیتی، بایاس ۱۲۷ میباشد. پس رابطه زیر برای توان واقعی برقرار است:

Exponent + bias = biased-exponent

همچنین میدانیم که برای بخش اعضاری باید مانتیسا را با ۱ جمع کنیم. با توجه به توضیحات داده شده خواهیم داشت.

1-

Sign = 0 , biased - exponent = 10010110 =$(150)_{10}$, mantissa=00000000010010100100101

Exponent = 150-127 = 23 , Mantissa = 1.00000000010010100100101

2-

Sign = 0 , biased - exponent = 10100000=$(160)_{10}$, mantissa = 001 0000 1011 0110 0110 1110

Exponent = 160 - 127 = 33 , Mantissa = 1. 00100001011011001101110

بعد از تبدیل هر دو عدد به سیستم گفته شده، برای جمع انها باید توان انها رو مقایسه کنیم تا عدد با توان کوچکتر را شیفت بدهیم.

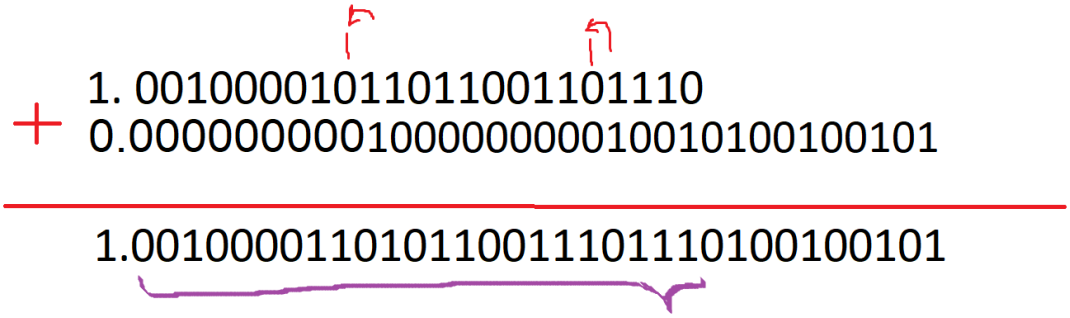
همانطور که در بالا نوشته ایم، توان عدد دوم بزرگتر است پس باید به اندازه اخلاف انها عدد کوچکتر(عدد اول) را به سمت راست شیفت بدهیم.

33 – 23 = 10

پس عدد اول را با ۱۰ بیت شیفت به راست، به صورت زیر بازنویسی میکنیم:

Exponent = ۳3 , Mantissa = ۰/۰۰۰۰۰۰۰۰۰۰00000000010010100100101

حال بخش های اعشاری دو عدد را به شکل زیر جمع میکنیم:



چون در سیستم ۳۲ بیتی کار میکنیم، ۳۲ بیت بعد از ممیز را جدا میکنیم. بیت ۲۴ م بعد از ممیز، صفر است پس نیازی به گرد کردن نیست.

در جواب نهایی با توجه به اینکه هر دو عدد مثبت هستند، بیت علامت ۰ خواهد شد. و بیت های توان و مانتسیا به شکل زیر خواهند بود:

Sign =0

Exponent + bias = biased- exponent ---> biased-exponent = 33 + 127 = $(160)_{10}$ =10100000

Mantissa = 00100001101011001110111

Sum = 01010000000100001101011001110111

ب)این کار برای راحتی در عملیات بین دو یا چند عدد انجام میشود. برای نمونه در همین مثال در قسمت الف سوال دیدیم که برای انجام عملیات جمع، باید توان های این دو عدد را مقایسه میکردیم. مقایسه دو عدد مکمل دو سخت است و زمان و عملیات نسبتا زیادی را از سیستم میگیرد. ولی با اضافه کردن بایاس فقط کافی است که دو عدد مثبت را مقایسه کنیم و با چالش مقایسه کردن اعداد در مکمل دو برخورد نمیکنیم. درواقع وقتی هر دو عدد را با بایاس جمع کنیم هر دو قطعا نامنفی میشوند(در صورت عدم وجود اورفلو و اندرفلو) و میتوانیم آنها را به روش lexicographical مقایسه کنیم. یعنی از بیت سمت راست مقایسه را شروع کنیم و در اولین بیتی که متفاوت بودند، عدد بزرگتر را مشخص کنیم.

توانایی نمایش و ذخیره عدد صفر در کامپیوتر ها اهمیت زیادی دارد. گاهی در محاسبات کامپیوتری، باید حالت های خاصی مانند تقسیم بر صفر را نیز در نظر بگیریم تا در صورت نیاز ارور مناسب را نشان دهیم. زمان هایی هم بعضی از اعداد د ر بازه حقیقی تعریف شده نیستند و باید راهی برای نشان دادن آنها پیدا کنیم. گاهی هم اعداد مورد استفاده در محاسبات خیلی بزرگ و یا خیلی کوچک هستند به طوری که سیستم های ممیز شناور ۳۲ بیت یا حتی ۶۴ بیت و ... برای نگهداری آن ها مناسب نیستند. پس در صورت ظهور این مشکلات باید راهی برای آن ها بیابیم.

نمایش عدد صفر: از آنجایی که در سیستم ممیز شناور همیشه بخش اعشاری را با ۱ جمع می‌کنیم. اکنون این سوال پیش می‌آید پس ۰ چه؟ در این سیستم قرداد می‌کنیم که اگر تمام بیت های توان و مانیسا صفر باشند، مستقل از بیت علامت(چون برای صفر علامت خاصی در نظر نمی‌گیریم)، عدد را صفر در نظر بگیریم.

نمایش اعداد نامعتبر(NaN): NaN برای اعدادی که عضو مجموعه حقیقی اعداد نیستند و همچنین در زمانهایی که عملیات های نامعتبر مانند تقسیم بر صفر یا جمع و تفریق دو بینهایت را انجام می‌دهیم، استفاده می‌شود. اگر تمام بیت های توان ۱ باشند و بیت های مانیتسا حداقل یک، ۱ داشته باشد، مستقل از بیت علامت این عدد نامعتبر تلقی می‌شود.

نمایش بی نهایت: اگر تمام بیت های توان ۱ باشند و تمام بیت های مانیتسا ۰ داشته باشد، اگر بیت علامت ۰ باشد این عدد مثبت بی‌نهایت و اگر بیت علامت ۱ باشد، این عدد منفی بی‌نهایت تلقی می‌شود.

Single Precision		Double Precision		Object Represented
Exponent	Mantissa	Exponent	Mantissa	
0	0	0	0	zero
0	nonzero	0	nonzero	± denormalized number
1-254	anything	1-2046	anything	± normalized number
255	0	2047	0	± infinity
255	nonzero	2047	nonzero	NaN (Not a Number)

ج)برای شروع و مثال اولیه با تعداد بیت ۳۲ شروع به مقایسه میکنیم و سپس با کمک این فرض برای حالت کلی نتیجه گیری میکنیم. در اولین نگاه با توجه به اینکه هر دو ۳۲ بیت دارند، میتوان گفت که از آنجایی که برای هر بیت دو حالت ۰و ۱ داریم میتوان در هر دو حالت ۲۸۳۲ عدد نوشت.(البته در حالت ممیز شناور به علت مشخص کردن حالت های مختلف برای بینهایت و صفر تعداد کمتری عدد داریم که دراینجا به آنها نمی‌پردازیم.)

ممیز شناور:

بازه اعداد نمایشی در سیستم ممیز شناوردر بازه مثبت ها از **1.17549435E-38** تا **3.40282347E+38** و ۰ و در بازه اعداد منفی از **-3.40282347E+38** تا **-1.17549435E-38** می‌باشد که دقت خیلی بالایی به حساب نمی‌آید و حتی در جمع و تفریق کردن بعضی اعداد خیلی کوچک خطای محاسباتی به چشم می‌آید(مثلا اگر C0123987 را با 81C56AC6 در سیستم ممیز شناور ۳۲ بیتی جمع کنیم حاصل برابر C0123987 می‌شود.)

البته باید گفت همانطور که در بالا ذکر شد، ممیز شناور در نزدیک ترین عدد مثبتی که ممیز شناور نزدیک به صفر نگه میدارد بسیار خوب عمل می‌کند.

اعداد در سیستم ممیز شناور به بالا رند می‌شوند و برای همین اعداد گاهی به طور دقیق نگهداری نمی‌شوند.

نکته ای که در اعداد ممیز شناور توجه ما را جلب می‌کند این است که فاصله اعداد تولید شده برابر نیست یا اصطلاحا epsilon ثابت نیست. به مثال زیر دقت کنید:

Scientific Notation	Binary	Decimal
1.000×2^{-1}	0.1	0.5
1.001×2^{-1}	0.1001	0.5625
1.010×2^{-1}	0.101	0.625
1.011×2^{-1}	0.1011	0.6875
1.100×2^{-1}	0.11	0.75
1.101×2^{-1}	0.1101	0.8125
1.110×2^{-1}	0.111	0.875
1.111×2^{-1}	0.1111	0.9375

میبینیم که فاصله هر دو عدد مجاور ۰/۰۶۲۵ میباشد.

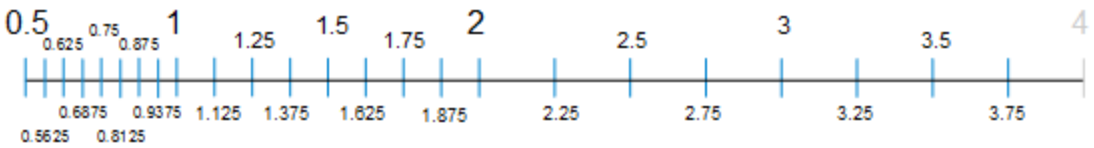
1.000×2^{0}	1	1
1.001×2^{0}	1.001	1.125
1.010×2^{0}	1.01	1.25
1.011×2^{0}	1.011	1.375
1.100×2^{0}	1.1	1.5
1.101×2^{0}	1.101	1.625
1.110×2^{0}	1.11	1.75
1.111×2^{0}	1.111	1.875

می‌بینیم که فاصله هر دو عدد مجاور ۰/۱۲۵ می‌باشد.

1.000×2^1	10	2
1.001×2^1	10.01	2.25
1.010×2^1	10.1	2.5
1.011×2^1	10.11	2.75
1.100×2^1	11	3
1.101×2^1	11.01	3.25
1.110×2^1	11.1	3.5
1.111×2^1	11.11	3.75

می‌بینیم که فاصله هر دو عدد مجاور ۰/۲۵ می‌باشد.

همانطور که مشاهده کردیم، با افزایش عددِ توان، فاصله بین هر دو عدد مجاور بیشتر می‌شود و اپسیلون ثابت نیست و وابسته به توان است.



ممیز ثابت:

در ممیز ثابت، بسته به اینکه ممیز را کجا قرار دهیم بازه متفاوتی از اعداد میتوانیم تولید کنیم.

برای مثال در ۳۲ بیت تصور می‌کنیم که ۸ رقم را به اعضار بدهیم و ۲۳ رقم به بخش صحیح و ۱ رقم به بخش علامت بدهیم. در این حالت بزرگترین عدد ما ۲۸۲۳ یا ۸۳۸۸۶۰۸ است و دقت نیز ۸-۲۸ یا تقریبا ۰/۰۴ است.

در سیستم ممیز ثابت دقت یا epsilon همواره ثابت است.

البته یک مزیتی که برای سیستم ممیز شناور می‌توانیم در نظر بگیریم این است که می‌توان تعداد بیت اختصاصی به ممیز را، بسته به کاربرد، تغییر داد. برای مثال هنگام کارکردن در ابعاد اتم، می‌توان تعداد خیلی زیادی (مثلا ۳۰ بیت) به بخش اعشاری داد.

با مقایسه توضیحات داده شده درباره ممیز ثابت و ممیز شناور می‌توان به این نتیجه رسید که، در حالت کلی تعداد اعدادی که تولید میشوند برابر با ۲ به توان تعداد بیت هاست و تقریبا در هر دو حالت برابر است.

درباره مقایسه بازه تولیدی اعداد در دو حالت نیز باید گفت که سیستم ممیز شناور با اختلاف از سیستم ممیز ثابت بهتر عمل می‌کند و بازه بیشتری را پوشش می‌دهد. در سیستم ممیز شناور فاصله دو عدد مجاور می‌تواند حتی تا10E-45 باشد ولی برای ممیز ثابت این عدد بیشتر است(در مثال ما 10E-3 بود.)در

مثال ۳۲ بیتی، حتی اگر هیچ بیتی را به اعشار اختصاص مدهیم و یک بیت علامت داشه باشیم باز هم بازه تولیدی اعداد در سیستم ممیز شناور بزرگتر است.

مزیت ممیز ثابت نسبت به ممیز شناور این است که اعداد را دقیق نگه می‌داردر حالی که در ممیز شناور، برنامه نویس باید به خطای احتمالی ناشی از نادقیق نگه داشتن اعداد توجه کند. همچنین اپسیلون در ممیز ثابت، همواره ثابت است و فاصله هر دو عدد مجاور ثابت است ولی در میز شناور این گونه نیست.