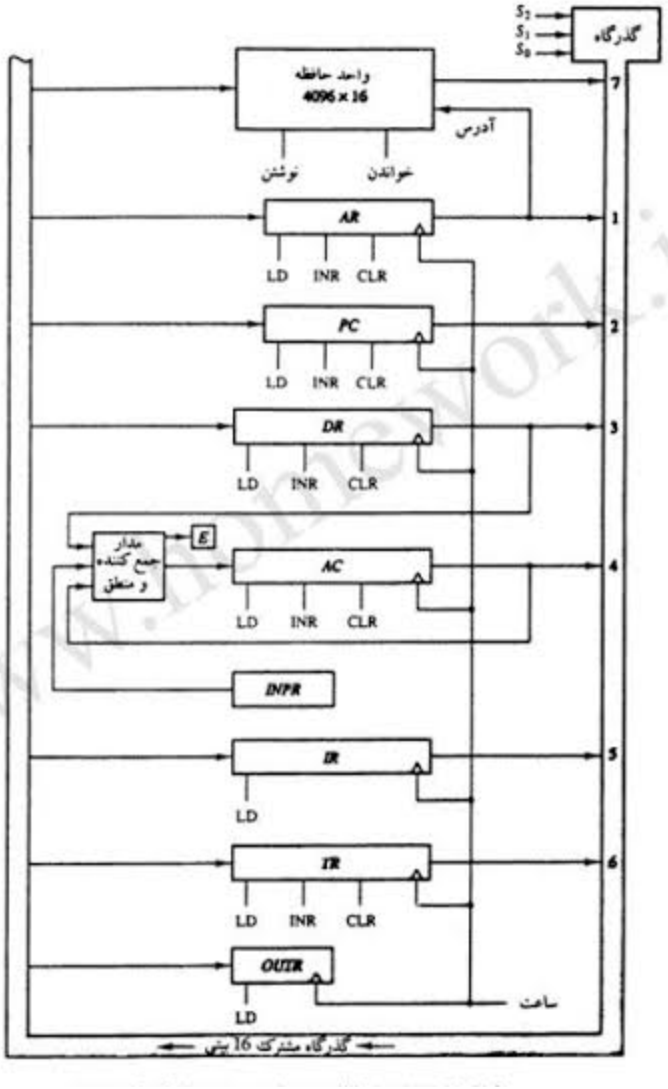
در این گزارش، برنامه نوشته شده برای گذرگاه کامپیوتر پایه که در شکل ۱ نشان داده شده است تشریح شده است. برای طراحی بخشهای مختلف از کتاب مرویس مانو و توضیحات مربوطه استفاده شده است.



شکل ۱: شماتیک گذرگاه(باس) مشترک در کامپیوتر پایه.

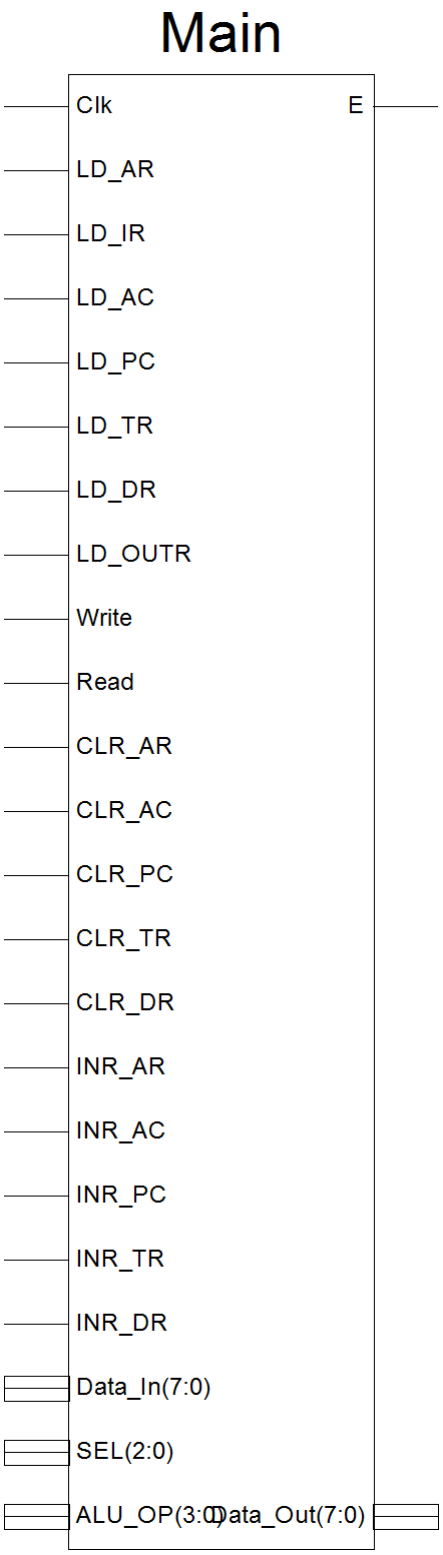
در این شبیه سازی برنامه ها به زبان Verilog نوشته شده و برای هر بخش مدار یک ماژول مجزا تعریف شده و کد مربوطه نوشته شده است. در واقع ماژولهای Memory، AR، PC، DR، AC، INPR، IR، TR و OUTR به صورت

ماژولهای جداگانه نوشته شده اند. کد مربوطه در نرم‌افزار ISE14.2 که مربوطه به شرکت Xilinx می باشد نوشته شده است.

در نهایت تمام این ماژولها در یک ماژول اصلی به نام ماژول Main فرخوانی یا Instantiate شده و به هم متصل گشته اند. برای کنترل باس ۱۶ بیتی نیز در ماژول Main کد مربوطه با استفاده از دستور Assign و به فرم شرطی نوشته شده است.

ماژول Main دارای ورودی کلاک (Clk)، ورودیهای بار کردن (LD) برای ماژولهای مختلف، ورودیهای پاک کردن (Clr) برای ماژولهای مربوطه، ورودی افزایش (Inr) برای ماژولها، ورودی انتخاب ۳ بیتی (SEL) برای انتخاب ورودی باس(همانند شکل مدار)، ورودی داده ۸ بیتی برای رجیتسر INPR و نیز خروجی ۸ بیتی برای داده های خروجی رجیستر OUTR می‌باشد. در واقع داده های ورودی مظابق با مطالب کتاب، به ورودی رجیستر INPR اعمال شده و خروجیهای رجیستر OUTR نیز به عنوان خروجی مجموعه لحاظ می شوند. علاوه بر این برای کنترل ALU نیز یک ورودی ۴ بیتی با نام ALU_OP لحاظ شده است.

برای طراحی ALU نیز طبق جدول دستورات کتاب موری مانو، یک ALU طراحی شده که دو ورودی ۱۶ بیتی از DR و AC به همراه یک ورودی ۸ بیتی از INPR دریافت کرده و دارای یک خروجی ۱۶ بیتی و یک خروجی تک بیتی با نام E می باشد. عملیات انجام شده در ALU شامل ۹ عمل مختلف می‌باشد که براساس جدول دستورات کتاب موریس مانو انتخاب شده اند. این عملیات شامل AND و ADD کردن دو ورودی ۱۶ بیتی، بار کردن یکی از ورودیهای ۱۶ بیتی (DR)، بار کردن ورودی ۸ بیتی، پاک کردن خروجی E، شیفت به چپ خروچی ۱۶ بیتی به همراه خروجی تک بیتی، شیفت به راست خروچی ۱۶ بیتی به همراه خروجی تک بیتی، معکوس (Complement) کردن خروجی ۱۶ بیتی و معکوس کردن خروجی E می باشد. برای انجام این عملیات از یک ورودی ۴ بیتی استفاده شده تا عملیات مورد نظر را انتخاب نماید.



شکل ۲: ماژول Main که ماژول اصلی برنامه می باشد.

در شکل فوق شمای ماژول Main نشان داده شده است.

در ادامه برنامه نوشته شده برای ماژول Main توضیح داده می شود.

این بخش از برنامه که در زیر دیده می شود برای تعریف ورودی و خروجیهای ماژول می باشد. ورودیها شامل ورودیهای بارگذاری، افزایش و پاک کردن ماژولهای مختلف که به اسم هر ماژول و تک بیتی آورده شده اند. همچنین شامل داده ورودی ۸ بیتی، ورودی ۳ بیتی برای انتخابگر ورودی باس، ورودی کلاک مجموعه، ورودی خواندن و نوشتن در حافظه و ورودی انتخاب عملیات ALU می باشد. خروجیها شامل داده خروجی ۸ بیتی و خروجی تکبیتی مربوطه به خروجی E یا همان کری ALU می باشد. همچنین ۳ خط آخر مربوط به تعریف سیگنالهای داخل مجموعه است. این سیگنالها شامل سیگنال ۱۶ بیتی باس، سیگنالهای ۱۶ بیتی خروجی ماژولهای IR، AC، DR، TR و ALU، سیگنالهای ۱۲ بیتی خروجی ماژولهای AR و PC می باشد.

```
module Main(
  input [7:0] Data_In,
  output [7:0] Data_Out,
  input Clk,
  input [2:0] SEL,
  input LD_AR, input LD_IR,     input LD_AC,    input LD_PC,
        input LD_TR,    input LD_DR,    input LD_OUTR,
  input Write,  input Read,
        input CLR_AR,  input CLR_AC,   input CLR_PC,   input CLR_TR,   input CLR_DR,
        input INR_AR,  input INR_AC,   input INR_PC,   input INR_TR,   input INR_DR,
        input [3:0] ALU_OP,     output E
  );
wire    [15:0]   Bus;
wire    [15:0]   Data_IR,Data_AC,Data_DR,Data_TR,Data_ALU;
wire    [11:0]   Data_AR,Data_PC;
```

بخش بعدی برنامه مربوط به کنترل ورودی باس می باشد. برای این عمل برنامه نوشته شده از دستور assign شرطی استفاده نموده و بسته به مقدار ورودی SEL یکی از داده های رجیسترهای AR، PC، DR، AC، IR، TR یا خروجی حافظه را در باس قرار می دهد. برای نوشتن این بخش از برنامه از جدول خود کتاب موری مانو استفاده

شده است. از آنجاییکه خروجی رجیسترهای AR و PC ض۲ بیتی بوده و باس ۱۶ بیتی است لذا مطابق با خود کتاب رویس مانو، در هنگام قرار دادن خروجی این رجیسترها در باس ۴ بیت پرارزش باس صفر قرار داده می شود({4'b0000,Data_AR} ,{4'b0000,Data_PC}).

```
assign Bus = (SEL == 3'b111) ? Data_RAM:
                               (SEL == 3'b110) ? Data_TR:
                               (SEL == 3'b101) ? Data_IR:
                               (SEL == 3'b100) ? Data_AC:
                               (SEL == 3'b011) ? Data_DR:
                               (SEL == 3'b010) ? {4'b0000,Data_PC}:
                               (SEL == 3'b001) ? {4'b0000,Data_AR}:
                               Bus;
```

بخش نهایی برنامه مربوط به فراخوانی ماژولهای مختلف و متصل نمودن آنها به هم می باشد.

```
Memory      U_RAM  (.Din(Bus),.Dout(Data_RAM),.Clk(Clk),.Write(Write),.Read(Read),.Address(Data_AR));
AR          U_AR   (.Din(Bus[11:0]),.Dout(Data_AR),.Clk(Clk),.Ld(LD_AR),.Clr(CLR_AR),.Inr(INR_AR));
PC          U_PC   (.Din(Bus[11:0]),.Dout(Data_PC),.Clk(Clk),.Ld(LD_PC),.Clr(CLR_PC),.Inr(INR_PC));
DR          U_DR   (.Din(Bus),.Dout(Data_DR),.Clk(Clk),.Ld(LD_DR),.Clr(CLR_DR),.Inr(INR_DR));
AC          U_AC   (.Din(Data_ALU),.Dout(Data_AC),.Clk(Clk),.Ld(LD_AC),.Clr(CLR_AC),.Inr(INR_AC));
INPR        U_INPR (.Din(Data_In),.Dout(Data_INPR));
IR          U_IR   (.Din(Bus),.Dout(Data_IR),.Clk(Clk),.Ld(LD_IR));
TR          U_TR   (.Din(Bus),.Dout(Data_TR),.Clk(Clk),.Ld(LD_TR),.Clr(CLR_TR),.Inr(INR_TR));
OUTR        U_OUTR(.Din(Bus[7:0]),.Dout(Data_Out),.Clk(Clk),.Ld(LD_OUTR));
ALU   U_ALU (.DR(Data_DR),.AC(Data_AC),.INPR(Data_INPR),.Clk(Clk),.Op(ALU_OP),.E(E),.Dout(Data_ALU));
```

هر خط از برنامه فوق برای فراخوانی یک ماژول بوده که در ابتدا اسم ماژول، سپس نام انتخابی برای آن در ماژول Main و سپس اتصال ورودی و خروجیهای ماژول به ورودی و خروجی یا سیگنالهای ماژول Main می باشد. برای مثال نام ماژول حافظه Memory بوده و در ماژول Main برای آن از اسم U_RAM استفاده شده است.

برنامه نوشته شده برای رجیسترهای AR، PC، DR، AC و TR مشابه هم می باشد. در ابتدا نام ماژول و ورودی و خروجیهای مربوطه تعریف شده اند. در ادامه و هدد سیگنال با استفاده از دستورات wire و reg تعریف شده اند. از دستور reg به این دلیل استفاده شده است که در این رجیسترها نیاز به استفاده از فلیف فلاپ و یا به عبارتی کد نویسی sequential در زبان Verilog داریم. در ادامه با استفاده از دستور شرطی assign سیگنال D مقدار دهی شده است. سپس با استفاده از دستور Always سیگنال Q که در واقع شامل فلیپ فلاپ است مقدار دهی گشته استو مقدار دهی سیگنال Q وابسته به کلاک ورودی می باشد و با لبه بالا رونده کلاک انجام می شود. در نهایت مقدار Q به خروجی انتقال داده می شود. تنها لازم به ذکر است که در رجیسترهای AR و PC سیگنالها و ورودیها ۱۲ بیت هستند.

```
module DR(
  input [15:0] Din,
  output [15:0] Dout,
  input Ld,   input Clr,
  input Inr,   input Clk
  );
wire    [15:0]   D;
reg     [15:0]   Q;
assign  D       =       ({Ld,Clr,Inr} == 3'b100)    ?       Din:
        ({Ld,Clr,Inr} == 3'b010)    ?       16'd0:
        ({Ld,Clr,Inr} == 3'b001)    ?       (Q+1):
        D;
always  @(posedge Clk)
                Q       <=      D;
assign  Dout    =       Q;
endmodule
```

رجیسترها یا ماژولهای IR و OUTR مشابه هم بوده و تنها ورودی بارگذاری و کلاک دارند. در این ماژولها همانند ماژولهای فوق، ابتدا نام ماژول و ورودی و خروجیها تعریف شده و سپس سیگنالهای D و Q تعریف شده اند. در ادامه سیگنال D تنها با استفاده از شرط برروی ورودی بارگذاری (LD) با استفاده از دستور شرطی assign مقدار

دهی شده و سپس سیگنال Q با دستور Always و در لبه بالا رونده کلاک مقدار دهی شده و در نهایت مقدار سیگنال Q به خروجی انتقال یافته است.

```
module IR(
  input [11:0] Din,
  output [11:0] Dout,
  input Ld,   input Clk
  );
wire    [11:0]   D;
reg     [11:0]   Q;
assign  D       =       (Ld == 1)?      Din:    D;
always  @(posedge Clk)
                Q       <=      D;
assign  Dout    =       Q;
endmodule
```

رجیستر یا ماژول INPR تنها یک بافر بوده و ورودی را دریافت و در خروجی قرار می دهد.

برای ماژول حافظه نیز همانند روشهای قبلی ورودی و خروجیها تعریف شده و سپس سیگنال D تعریف شده است. به حای سیگنال Q در این برنامه یک آرایه دوبعدی با استفاده از دستور reg و به ابعاد 16*4096 تعریف شده است. در ادامه داده های دردون این آرایه با استفاده از دستور Always و در لبه رونده کلاک و با شرط برروی ورودی Write با توجه به مقدار ورودی Address مقدار ورودی را دریافت می نمایند. در واقع Address به اندیس مربوطه از آرایه RAM اشاره می نماید. در ادامه با استفاد از دستور Always و در لبه بالا روند کلاک و شرط برروی ورودی Read سیگنال D مقدار خانه ای از آرایه دوبعدی که ورودی Address به آن اشاره می کند را دریافت می نماید. در نهایت مقدار سیگنال D به خروجی انتقال می یابد.

```
module Memory(
  input [15:0] Din,   output [15:0] Dout,
  input Write,   input Read,
  input [11:0] Address,  input Clk
  );
```

```
reg     [15:0]  D;
reg     [15:0]  RAM     [4095:0];
always  @(posedge Clk)
                if      ({Write,Read} == 2'b10)
                        RAM[Address]    <=      Din;
always  @(posedge Clk)
                if      ({Write,Read} == 2'b01)
                        D       <=      RAM[Address];
                else
                        D       <=      D;
assign  Dout    =       D;
endmodule
```