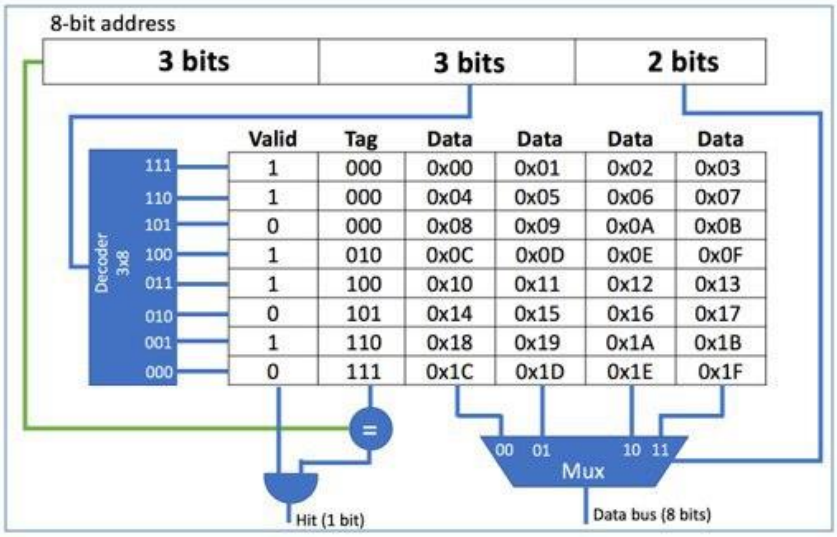
گزارش بخش 3

عرفان رفیعی – فربد فولادی – پارسا نوری (گروه 1)

است که برای جلوگیری از تأخیر DRAM یا SRAM یا حافظه نهان نوعی از حافظهٔ (CACHE) حافظه کش زمان پردازش بین پردازنده و حافظهٔ رم کامپیوتر استفاده می‌شود. حافظه کش معمولاً ظرفیت کمتر و سرعت بیشتری .نسبت به دیگر حافظه‌های مورداستفاده در کامپیوتر دارد.

زمانی که پردازنده از حافظه درخواست خواندن دارد، ابتدا محتویات حافظه کش سیستم بررسی می‌شود. اگر دادهٔ مدنظر در حافظه CACHE سیستم وجود داشته باشد، دیگر نیازی به دسترسی به حافظهٔ رم وجود ندارد و به این ترتیب، سرعت فرآیند بیشتر خواهد شد.

طبق عکسی که در کانال گذاشته شد حافظه cache را پیاده میکنیم:



توضیح :

چهار رجیستر برای هر ستون از داده‌های هشت بیتی ساختیم که هر کدام هشت خط دارند و در نام آن‌ها ، واضح است که نشان‌دهنده‌ی کدام ستون هستند.

هشت خط تگ و هشت بیت هم valid در نظر گرفته شده است.

```
module cache(
    input clk,
    input [7:0] address,
    input [31:0] data,
    output reg hit,
    output reg [7:0] out
    );

    reg [7:0] m00 [0:7];
    reg [7:0] m01 [0:7];
    reg [7:0] m10 [0:7];
    reg [7:0] m11 [0:7];
    reg [2:0] tag [0:7];
    reg [7:0] v;

    initial begin
    v=8'b00000000;
    end
```

ابتدا تمام بیت‌های valid را صفر می‌کنیم که ورودی‌های ما همیشه در اولین اجرا در کش ذخیره شوند و درواقع مموری ما خالی از اطلاعات (در ابتدا) باشد.

در ادامه ، برنامه‌ی طراحی شده را پیاده می‌کنیم و این پیاده‌سازی بسیار ساده خواهد بود. فقط کافیست بررسی کنیم که آدرس ورودی در کش ذخیره شده یا خیر.

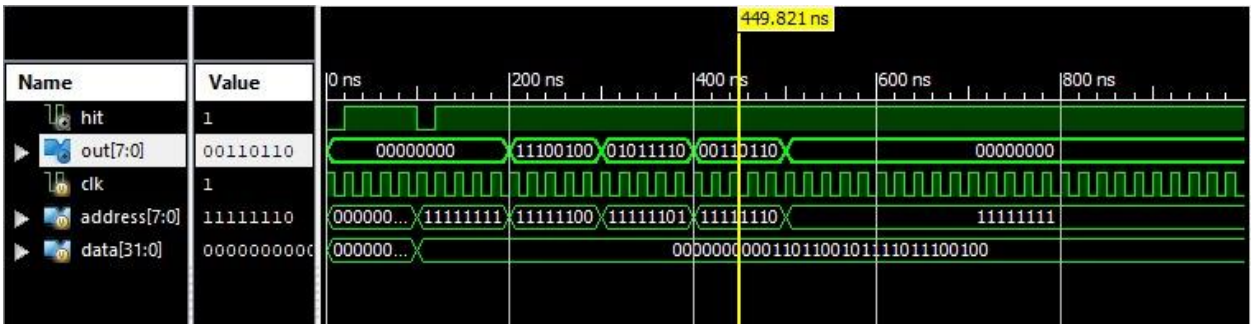
```
always @(posedge clk)
if(clk) begin
  if ( address [7:5] == tag [ address [4:2] ] && v [ address [7:5] ] == 1 ) begin
     hit <= 1;
     if( address[1:0] == 0 ) begin
        out <= m00[ address [7:5] ];
     end else if( address[1:0] == 1 ) begin
        out <= m01[ address [7:5] ];
     end else if( address[1:0] == 2 ) begin
        out <= m10[ address [7:5] ];
     end else begin
        out <= m11[ address [7:5] ];
     end
  end
  else begin
     v [ address [4:2] ] <= 1;
     tag [ address [4:2] ] <= address [7:5];
     hit <= 0;
     m00[ address [7:5] ] = data [7:0];
     m01[ address [7:5] ] = data [15:8];
     m10[ address [7:5] ] = data [23:16];
     m11[ address [7:5] ] = data [31:24];
     if( address[1:0] == 0 ) begin
        out <= m00[ address [7:5] ];
     end else if( address[1:0] == 1 ) begin
        out <= m01[ address [7:5] ];
     end else if( address[1:0] == 2 ) begin
        out <= m10[ address [7:5] ];
     end else begin
        out <= m11[ address [7:5] ];
     end
  end
end
```

سپس با توجه به دو بیت کم‌ارزش آدرس ، خروجی را تنظیم می‌کنیم:

```
initial begin
  // Initialize Inputs
  clk = 0;
  address = 0;
  data = 0;

  // Wait 100 ns for global reset to finish
  #100;

  // Add stimulus here
  address = 8'b11111111;
  data = 3563236;
  #100;

  address = 8'b11111100;
  #100;

  address = 8'b11111101;
  #100;

  address = 8'b11111110;
  #100;

  address = 8'b11111111;
  #100;
```

برای تست هم به سادگی می‌توان یک آدرس را درنظر گرفت و دو بیت انتهایی آن‌را (کم‌ارزش) تغییر داد تا صحت عملکرد هیت و خروجی بررسی شود:

کلاک این تست ، هر 100 نانو ثانیه تغییر می‌کند. در این تست ما یک ورودی به کش می‌دهیم و در ادامه هشت بیت هشت بیت آن را در خروجی به نمایش می‌گذاریم:



می‌بینیم که خروجی ما به ترتیب هشت بیت کم‌ارزش ورودی ، هشت بیت دوم و سوم و هشت بیت پرارزش آن را نمایش می‌دهد. هیت نیز در تمام طول اجرای این تست (بعد از گرفتن ورودی) یک است زیرا آدرس داده شده تغییری نکرده و همان آدرس قبلی باقی‌مانده.

بحران زمانی نیز در جدول زیر معلوم است:

Clock clk to Pad

Destination	clk (edge) to PAD	Internal Clock(s)	Clock Phase
hit	7.145(R)	clk_IBUF	0.000
out<0>	7.197(R)	clk_IBUF	0.000
out<1>	7.243(R)	clk_IBUF	0.000
out<2>	7.265(R)	clk_IBUF	0.000
out<3>	7.263(R)	clk_IBUF	0.000
out<4>	7.438(R)	clk_IBUF	0.000
out<5>	7.703(R)	clk_IBUF	0.000
out<6>	7.580(R)	clk_IBUF	0.000
out<7>	7.319(R)	clk_IBUF	0.000