



گزارش آزمایش هشتم طراحی سیستم‌های دیجیتال

گروه شش

اعضا:

احمد سلیمی

همیلا میلی

درنا دهقانی

شرح آزمایش

در این آزمایش می‌خواهیم یک کامپیوتر پایه طراحی کنیم که از چند بخش تشکیل شده‌است:

الف) بخش جمع و تفریق اعداد مختلط.

ب) بخش ضرب اعداد مختلط.

ج) بخش خوانش و اجرای دستور از حافظه که باید پایپلاین باشد.

در طراحی حافظه‌ی این کامپیوتر، از ۳۲ کلمه‌ی ۱۶ بیتی استفاده می‌شود. یعنی برای هر بخش (حقیقی و موهومی) ۸ بیت در نظر گرفته شده‌است.

فایل macros.v

با توجه به اینکه برخی توابع یا مقداردهی‌ها در طول طراحی آزمایش ثابت بودند، در یک فایل وریلاگ این مقادیر را توسط ``define` مشخص کردیم و در سایر ماژول‌ها از آنها استفاده شد. این مقادیر عبارتند از:

- WL: مقداری ثابت برابر با ۸.
- Complex: مشخص‌کننده ابتدا و انتهای هر `reg` یا `wire` که حاوی عددی مختلط است.
- Re(c): تابعی برای برگرداندن مقدار حقیقی عدد مختلط، یعنی ۸ بیت پرارزش‌تر.
- Im(c): تابعی برای برگرداندن مقدار موهومی عدد مختلط، یعنی ۸ بیت کم‌ارزش‌تر.
- sRe(c): تابعی برای برگرداندن مقدار حقیقی علامت‌دار عدد مختلط، یعنی ۸ بیت پرارزش‌تر.
- sIm(c): تابعی برای برگرداندن مقدار موهومی علامت‌دار عدد مختلط، یعنی ۸ بیت کم‌ارزش‌تر.

این فایل در تمامی ماژول‌ها ``include` می‌شود.

کد وریلاگ این بخش به شکل زیر است:

```
1 `define WL 8
2 `define complex [2*`WL-1:0]
3 `define Re(c) c[2*`WL-1:`WL]
4 `define Im(c) c[`WL-1:0]
5 `define sRe(c) $signed(`Re(c))
6 `define sIm(c) $signed(`Im(c))
```

ماژول‌ها

Addsub

ورودی‌ها:

- a: عدد ورودی اول.
- b: عدد ورودی دوم.
- op: مشخص‌کننده‌ی عملیات جمع یا تفریق.

خروجی‌ها:

- s: حاصل جمع یا تفاضل دو ورودی.

در این ماژول بر حسب مقدار ورودی op، مقدار حقیقی ورودی b با مقدار حقیقی ورودی a جمع یا تفریق شده و در مقدار حقیقی خروجی s قرار می‌گیرد. به طور مشابه مقدار موهومی ورودی b با مقدار موهومی ورودی a جمع یا تفریق شده و در مقدار موهومی خروجی s قرار می‌گیرد.

کد وریلاگ این بخش به شکل زیر است:

```
1  `include "macros.v"
2
3  module addsub (
4      input  `complex  a,
5      input  `complex  b,
6      input          op,    // 0 for addition, 1 for subtraction
7      output `complex  s
8  );
9
10 assign `Re(s) = `sRe(a) + (op ? -1 : 1) * `sRe(b);
11 assign `Im(s) = `sIm(a) + (op ? -1 : 1) * `sIm(b);
12
13 endmodule
```

Addsub_TB

در این ماژول عملکرد ماژول addsub بررسی می‌شود. کد وریلاگ آن به شکل زیر است:

```
1  `include "macros.v"
2  module addsub_TB ();
3
4  reg    `complex  a, b;
5  reg    op;
6  wire   `complex  s;
7
8  addsub  ADDSUB(a, b, op, s);
9  wire [7:0] op_char;
10 assign op_char = op ? "-" : "+";
11 integer i;
12 initial begin
13     $monitor("(%d, %d) %s (%d, %d) = (%d, %d)",
14             `sRe(a), `sIm(a), op_char, `sRe(b), `sIm(b), `sRe(s), `sIm(s));
15
16     `Re(a) = -10;
17     `Im(a) = 15;
18     `Re(b) = 13;
19     `Im(b) = -18;
20     op = 1;
21     #10;
22     `Re(a) = 16;
23     `Im(a) = 3;
24     `Re(b) = 12;
25     `Im(b) = -64;
26     op = 0;
27     #10;
28     `Re(a) = 2;
29     `Im(a) = 18;
30     `Re(b) = -50;
31     `Im(b) = 32;
32     op = 0;
33     #10;
34     `Re(a) = 54;
35     `Im(a) = 31;
36     `Re(b) = -12;
37     `Im(b) = -37;
38     op = 1;
39     #10;
40     $stop;
41 end
42 endmodule
```

حاصل شبیه‌سازی این بخش از آزمایش به شکل زیر است:

```
VSIM8> run 5000
# ( -10, 15) - ( 13, -18) = ( -23, 33)
# ( 16, 3) + ( 12, -64) = ( 28, -61)
# ( 2, 18) + ( -50, 32) = ( -48, 50)
# ( 54, 31) - ( -12, -37) = ( 66, 68)
```

/addsub_TB/a	0011011000011111	0001000000000011	0000001000010010					0011011000011111											
/addsub_TB/b	1111010011011011	0000110011000000	1100111000100000					1111010011011011											
/addsub_TB/op	1																		
/addsub_TB/s	0100001001000100	0001110011000011	1101000000110010					0100001001000100											
/addsub_TB/op_char	-							+											
/addsub_TB/i	x																		

Mul

ورودی‌ها:

- a: عدد ورودی اول.
- b: عدد ورودی دوم.

خروجی‌ها:

- s: حاصل ضرب دو ورودی.

در این ماژول طبق فرمول $(a + bi)(c + di) = (ac - bd) + (bc + ad)i$ ، مقدار حقیقی و موهومی حاصل ضرب محاسبه می‌شوند.

کد وریلاگ این بخش به شکل زیر است:

```

1  `include "macros.v"
2
3  module mul (
4      input  `complex  a,
5      input  `complex  b,
6      output `complex  s
7  );
8
9  assign `Re(s) = `sRe(a) * `sRe(b) - `sIm(a) * `sIm(b);
10 assign `Im(s) = `sRe(a) * `sIm(b) + `sIm(a) * `sRe(b);
11
12 endmodule

```

Mul_TB

در این ماژول عملکرد ماژول mul بررسی می‌شود. کد وریلاگ این بخش به شکل زیر است:

```

1  `include "macros.v"
2
3  module mul_TB ();
4
5  reg    `complex  a, b;
6  wire   `complex  s;
7
8  mul    MUL(a, b, s);
9
10 initial begin
11     $monitor("(%d, %d) * (%d, %d) = (%d, %d)",
12             `sRe(a), `sIm(a), `sRe(b), `sIm(b), `sRe(s), `sIm(s));
13
14     `Re(a) = -10;
15     `Im(a) = 5;
16     `Re(b) = 3;
17     `Im(b) = -8;
18     #10;
19     `Re(a) = 6;
20     `Im(a) = 3;
21     `Re(b) = 2;
22     `Im(b) = -6;
23     #10;
24     `Re(a) = 2;
25     `Im(a) = 8;
26     `Re(b) = -0;
27     `Im(b) = 2;
28     #10;
29     `Re(a) = 4;
30     `Im(a) = 1;
31     `Re(b) = -2;
32     `Im(b) = -7;
33     #10;
34     $stop;
35 end
36
37 endmodule




```

حاصل شبیه‌سازی این بخش از آزمایش به شکل زیر است:

```

VSIM 19> run 1000
# (-10, 5) * ( 3, -8) = ( 10, 95)
# ( 6, 3) * ( 2, -6) = ( 30, -30)
# ( 2, 8) * ( 0, 2) = (-16, 4)
# ( 4, 1) * (-2, -7) = (-1, -30)

```

 /mul_TB/a	0000010000000001	1111011000000101	000001100000011	0000001000001000	0000010000000001
 /mul_TB/b	1111111011111001	0000001111111000	0000001011111010	0000000000000010	1111111011111001
 /mul_TB/s	1111111111100010	000010100101111	0001111011100010	1111000000000100	1111111111100010

Alu

ورودی‌ها:

- a: عدد ورودی اول.
- b: عدد ورودی دوم.
- op: مشخص‌کننده‌ی عملیات جمع یا تفریق یا ضرب.

خروجی‌ها:

- s: حاصل جمع یا تفاضل یا ضرب دو ورودی.

در این ماژول ابتدا حاصل جمع یا تفریق (برحسب مقدار op[0]) دو ورودی و مقدار حاصل ضرب دو ورودی محاسبه می‌شوند. سپس با توجه به مقدار op[1] که مشخص می‌کند باید ضرب یا جمع/تفریق انجام بگیرد، مقدار مورد نظر در خروجی قرار می‌گیرد.

کد وریلاگ این بخش به شکل زیر است:

```
1  `include "macros.v"
2
3  module alu (
4      input  `complex  a,
5      input  `complex  b,
6      input  [1:0]     op,
7      output `complex  s
8  );
9
10 wire `complex  addsub_res, mul_res;
11 addsub ADDSUB (a, b, op[0], addsub_res);
12 mul     MUL (a, b, mul_res);
13
14 assign s = op[1] ? mul_res : addsub_res;
15
16 endmodule
```

Memory

ورودی‌ها:

- Raddr1: آدرس اولین ورودی که می‌خواهیم مقدار آن را بخوانیم. چون حافظه ۳۲ کلمه‌ایست، به ۵ بیت برای مشخص کردن آن نیاز داریم.
- Raddr2: آدرس دومین ورودی که می‌خواهیم مقدار آن را بخوانیم. چون حافظه ۳۲ کلمه‌ایست، به ۵ بیت برای مشخص کردن آن نیاز داریم.
- Wdata: مقداری که می‌خواهیم در حافظه ذخیره کنیم.

- Waddr: آدرس مقداری که می‌خواهیم ورودی wdata در آن ذخیره شود.

خروجی‌ها:

- Rdata1: مقداری که در حافظه در خانه‌ی raddr1 قرار دارد.
- Rdata2: مقداری که در حافظه در خانه‌ی raddr2 قرار دارد.

این ماژول، حافظه‌ی فرضی کامپیوتر ساده‌ی ماست و از ۳۲ کلمه‌ی ۱۶ بیتی ساخته شده است. در این ماژول همواره مقداری از حافظه که در خانه‌ی raddr1 قرار دارد، در خروجی rdata1 و مقداری از حافظه که در خانه‌ی raddr2 قرار دارد در خروجی rdata2 قرار می‌گیرد. ضمناً با هر تغییر ورودی‌های wdata یا waddr، مقدار wdata در خانه‌ای از حافظه با اندیس waddr قرار می‌گیرد.

کد وریلاگ این بخش از مدار به شرح زیر است:

```

1  `include "macros.v"
2
3  module memory #(
4      parameter DEPTH = 32,
5      parameter A_LEN = 5
6  ) (
7      input  [A_LEN:1]    raddr1,
8      input  [A_LEN:1]    raddr2,
9      input  `complex     wdata,
10     input  [A_LEN:1]    waddr,
11     output `complex     rdata1,
12     output `complex     rdata2
13 );
14
15 reg `complex    mem [DEPTH-1:0];
16
17 assign rdata1 = mem[raddr1];
18 assign rdata2 = mem[raddr2];
19 always @(*) mem[waddr] <= wdata;
20
21 endmodule

```

Inst_fetch

ورودی‌ها:

- Clk
- rstN

خروجی‌ها:

- op: بیانگر عملیاتی‌ست که در این دستور انجام می‌گیرد. در ماژول alu مشاهده شد که باید ۲ بیتی باشد.
- Waddr: آدرس خانه‌ای از حافظه است که حاصل انجام دستور، باید در آن قرار گیرد.
- Raddr1: آدرس خانه‌ای از حافظه که مقدار ورودی اول در آن قرار دارد.
- Raddr2: آدرس خانه‌ای از حافظه که مقدار ورودی دوم در آن قرار دارد.

در این ماژول به واکشی دستورات ۱۷ بیتی که در حافظه‌ی دستورات قرار دارند می‌پردازیم. ابتدا متغیر pc را برای پیمایش این حافظه مشخص می‌کنیم. در هر دستور که قرار گرفته باشیم، همواره مقدار خروجی‌های op، waddr، raddr1، raddr2 به ترتیب برابر با بیت‌های اول تا دوم آن دستور، بیت‌های سوم تا هفتم آن دستور، بیت‌های هشتم تا دوازدهم آن دستور و بیت‌های سیزدهم تا هفدهم آن دستور می‌باشند. ضمناً با هر لبه‌ی بالارونده‌ی کلاک یا پایین‌رونده‌ی ریست، اگر مقدار ورودی rstN ۱ باشد، مقدار pc برابر با ۰ و در غیر اینصورت مقدار pc یک واحد افزایش می‌یابد.

کد وریلاگ این بخش از آزمایش به شرح زیر است:

```

1 module inst_fetch (
2     input          clk,
3     input          rstN,
4     output [1:0]   op,
5     output [4:0]   waddr,
6     output [4:0]   raddr1,
7     output [4:0]   raddr2
8 );
9
10 localparam DEPTH = 32;
11 localparam A_LEN = 5;
12
13 reg [1:17] mem [DEPTH-1:0];
14 reg [A_LEN:1] pc;
15
16 assign op      = mem[pc][1:2];
17 assign waddr   = mem[pc][3:7];
18 assign raddr1  = mem[pc][8:12];
19 assign raddr2  = mem[pc][13:17];
20
21 always @(posedge clk or negedge rstN)
22     pc <= rstN ? (pc + 1) : 0;
23
24 endmodule

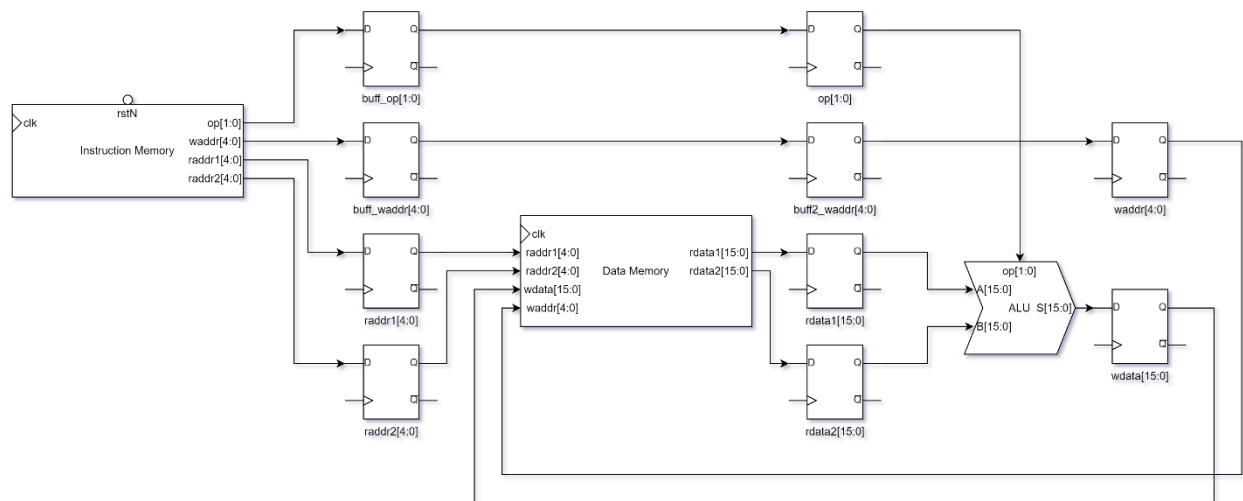
```

Pipeline

ورودی‌ها:

- Clk
- rstN

دیاگرام این ماژول به شکل زیر است:



این ماژول، ماژول اصلی آزمایش است. در این ماژول هر یک از ۳ ماژول inst_fetch, memory, alu یک شیء می‌سازیم:

- IF: شیء ماژول inst_fetch که همان ورودی‌های ماژول pipeline را می‌گیرد و خروجی‌هایش را در چهار wire قرار می‌دهد.
- MEM: شیء ماژول memory که چهار ورودی‌اش را از چهار reg می‌گیرد و دو خروجی‌اش را در دو wire برمی‌گرداند.
- ALU: شیء ماژول alu که سه ورودی‌اش را از سه reg می‌گیرد و خروجی‌اش را در یک wire قرار می‌دهد.

سپس به یک always block که به لبه‌ی بالارونده کلاک و لبه‌ی پایین‌رونده ریست حساس است وارد می‌شویم. در صورت ۱ بودن مقدار rstN، در هر کلاک مقادیر خروجی سه شیء فوق که در wire بودند، به reg منتقل می‌شوند.

خروجی i_op از IF در بافر buff_op قرار می‌گیرد، سپس مقدار op که ورودی alu می‌باشد، برابر با buff_op می‌شود که با اینکه در کلاک جدید مقداردهی شده، اما چون مقداردهی non-blocking است، تا پایان کلاک مقدار سابق خود را حفظ می‌کند و باعث می‌شود کامپیوتر به صورت پایپلاین عمل کند. به طور مشابه مقدار خروجی i_waddr از IF ابتدا در بافر buff_waddr و سپس در بافر buff2_waddr قرار می‌گیرد تا پس از انجام محاسبات در ALU، نتیجه صحیح بتواند در آن خانه از حافظه ذخیره شود و خللی در عملیات پایپلاین وارد نشود.

در واقع به کمک این بافرها و رجیسترها، هر دستور در هر کلاک که در مرحله‌ی محاسبات و ذخیره در حافظه که قرار داشته باشد، دستور پیشین آن در یک مرحله قبل‌تر یعنی خواندن از حافظه و دستور پیش از پیشین آن در دو مرحله قبل‌تر یعنی واکشی قرار دارند.

کد وریلاگ این ماژول به شرح زیر است:

```

1  `include "macros.v"
2  module pipeline (
3      input      clk,
4      input      rstN
5  );
6      wire [1:0] i_op;
7      wire [4:0] i_waddr, i_raddr1, i_raddr2;
8      wire `complex m_rdata1, m_rdata2, alu_out;
9
10     reg [1:0] buff_op, op;
11     reg [4:0] buff_waddr, buff2_waddr, raddr1, raddr2, waddr;
12     reg `complex rdata1, rdata2, wdata;
13
14     inst_fetch IF(clk, rstN, i_op, i_waddr, i_raddr1, i_raddr2);
15     memory    MEM(raddr1, raddr2, wdata, waddr, m_rdata1, m_rdata2);
16     alu        ALU(rdata1, rdata2, op, alu_out);
17     always @(posedge clk or negedge rstN) begin
18         if (rstN) begin
19             // IF
20             buff_op <= i_op;
21             buff_waddr <= i_waddr;
22             raddr1 <= i_raddr1;
23             raddr2 <= i_raddr2;
24             // MEM
25             rdata1 <= m_rdata1;
26             rdata2 <= m_rdata2;
27             op <= buff_op;
28             buff2_waddr <= buff_waddr;
29             // ALU
30             waddr <= buff2_waddr;
31             wdata <= alu_out;
32
33             $display("#%d\tbuff_op=%b, buff_waddr=%d, raddr1=%d, raddr2=%d", $time,
34                 buff_op, buff_waddr, raddr1, raddr2);
35             $display("#%d\ttop=%b, buff2_waddr=%d, rdata1=(%d, %d), rdata2=(%d, %d)", $time,
36                 op, buff2_waddr, $Re(rdata1), $Im(rdata1), $Re(rdata2), $Im(rdata2));
37             $display("#%d\twaddr=%d, wdata1=(%d, %d)\n", $time,
38                 waddr, $Re(wdata), $Im(wdata));
39         end
40     end
41 endmodule

```

Pipeline_TB

در این ماژول کارکرد کلی ماژول اصلی بررسی می‌شود. پس ابتدا یک شیء با نام PIPELINE از ماژول pipeline ساخته می‌شود. سپس مقادیر موجود در دو فایل inst_mem و initial_mem که به ترتیب مجموعه دستورات و مقادیر اولیه ذخیره شده در حافظه می‌باشند در آرایه mem در بخش IF از ماژول PIPELINE و در آرایه mem در بخش MEM از ماژول PIPELINE قرار می‌گیرند.

مقادیر موجود در initial_mem عبارت است از:

```

1 00011001_11100100 // (25, -28)
2 00001110_00001001 // (14, 9)
3 00000101_11110010 // (5, -14)
4 00010101_00000111 // (21, 7)
5 11001111_00010001 // (-49, 17)
6 00011101_11101001 // (29, -23)
7 00010111_00001111 // (23, 15)
8 11001101_11111010 // (-51, -6)
9 11110110_00001010 // (-10, 10)
10 11111110_11011101 // (-2, -35)
11 00000001_00000010 // (1, 2)
12 11111111_00100000 // (-1, 32)
13 01000101_10111010 // (69, -70)
14 11110111_10011111 // (-9, -97)
15 11111111_01100110 // (-1, 102)
16 00010111_11111001 // (23, -7)
17 00110100_00000000 // (52, 0)
18 01011001_00110110 // (89, 54)
19 11000100_11100101 // (-60, -27)
20 11110110_11001101 // (-10, -51)
21 11111111_10011001 // (-1, -103)
22 01100001_10101101 // (97, -83)
23 01101111_11100011 // (111, -29)
24 11111110_11111110 // (-2, -2)
25 01001011_10101110 // (75, -82)
26 11001111_10000110 // (-49, -122)
27 10011101_11111011 // (-99, -5)
28 10010001_01111000 // (-111, 120)
29 00111010_01011110 // (58, 94)
30 00101011_01111101 // (43, 125)
31 01011010_00110001 // (90, 49)
32 00111101_00000110 // (61, 6)

```

مقادیر موجود در inst_mem عبارت است از:

```

1 10_00111_01010_10111 // mul $00111 $01010 $10111
2 10_00011_10111_00001 // mul $00011 $10111 $00001
3 10_01011_00001_10111 // mul $01011 $00001 $10111
4 10_11011_01010_01001 // mul $11011 $01010 $01001
5 00_00010_00111_10010 // add $00010 $00111 $10010
6 00_01000_11011_01111 // add $01000 $11011 $01111
7 00_00010_00111_01001 // add $00010 $00111 $01001
8 00_00101_01011_10001 // add $00101 $01011 $10001
9 01_00110_00001_01000 // sub $00110 $00001 $01000
10 01_10110_10011_01000 // sub $10110 $10011 $01000
11 01_11100_11111_00101 // sub $11100 $11111 $00101
12 01_11010_01111_00010 // sub $11010 $01111 $00010
13 01_01011_00100_00101 // sub $01011 $00100 $00101
14 01_01111_10001_01000 // sub $01111 $10001 $01000
15 01_00101_10100_00101 // sub $00101 $10100 $00101

```

با توجه به این که ۱۵ دستور داریم و روند اجرا پایلین است، حداقل به ۱۷ کلاک نیاز داریم. پس به محض رسیدن به کلاک ۱۸م، اجرا متوقف می‌شود.

نهایتاً آرایه mem در بخش MEM از ماژول PIPELINE که اکنون برخی خانه‌های آن مقادیر جدیدی دارند، به عنوان حافظه نهایی در فایل final_mem ذخیره می‌شود:

```

1 // memory data file (do not edit the following line - required for mem load use)
2 // instance=/pipeline_TB/PIPELINE/MEM/mem
3 // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4 0001100111100100
5 0000111000001001
6 0000000011010111
7 1111011011010010
8 1100111100010001
9 1011000010010001
10 1011001100110111
11 0000001011111010
12 0101101111010010
13 1111111011011101
14 0000000100000010
15 10000000000001001
16 0100010110111010
17 1111011110011111
18 1111111101100110
19 1111111001100100
20 0011010000000000
21 0101100100110110
22 1100010011100101
23 1111011011001101
24 1111111110011001
25 0110000110101101
26 1001101111111011
27 1111111011111110
28 0100101110101110
29 1100111110000110
30 0001011100100010
31 0100010011011001
32 1110111011111110
33 0010101101111101
34 0101101000110001
35 0011110100000110

```

مقادیر حقیقی و موهومی این اعداد به شرح زیر است:

0001100111100100	// (25, -28)
0000111000001001	// (14, 9)
0000000011010111	// (0, -41)
1111011011010010	// (-10, -46)
1100111100010001	// (-49, 17)
1011000010010001	// (-80, -111)
1011001100110111	// (-77, 55)
0000001011111010	// (2, -6)
0101101111010010	// (91, -46)
1111111011011101	// (-2, -35)

0000000100000010	// (1, 2)
1000000000001001	// (-128, 9)
0100010110111010	// (69, -70)
1111011110011111	// (-9, -97)
1111111101100110	// (-1, 102)
1111111001100100	// (-2, 100)
0011010000000000	// (52, 0)
0101100100110110	// (89, 54)
1100010011100101	// (-60, -27)
1111011011001101	// (-10, -51)
1111111110011001	// (-1, -103)
0110000110101101	// (97, -83)
1001101111111011	// (-101, -5)
1111111011111110	// (-2, -2)
0100101110101110	// (75, -82)
1100111110000110	// (-49, -122)
0001011100100010	// (23, 34)
0100010011011001	// (68, -39)
1110111011111110	// (-18, -2)
0010101101111101	// (43, 125)
0101101000110001	// (90, 49)
0011110100000110	// (61, 6)

کد وریلاگ این ماژول به شرح زیر است:

```

1  module pipeline_TB ();
2
3  reg rstN = 0, clk = 1;
4  pipeline PIPELINE(clk, rstN);
5
6  always #10 clk = ~clk;
7  initial begin
8      $readmemb("data/inst_mem.txt", PIPELINE.IF.mem, 0, 32);
9      $readmemb("data/initial_mem.txt", PIPELINE.MEM.mem, 0, 32);
10
11     #40 rstN = 1;
12     wait(PIPELINE.IF.pc == 18);
13     $writememb("data/final_mem.txt", PIPELINE.MEM.mem);
14     $stop;
15 end
16
17 endmodule

```

با شبیه‌سازی این ماژول توسط modelsim می‌توان مشاهده کرد که طبق انتظار، در هر کلاک (به جز کلاک‌های ابتدایی و انتهایی که تنها یک دستور در حال اجرا دارند) هم‌زمان یک دستور در حال واکنشی، یک دستور در حال خوانده شدن از حافظه و یک دستور در حال محاسبه و ذخیره است:

```

# Time: 0 ns Iteration: 0 Instance: /pipeline_TB
#
# 40 buff_op=xx, buff_waddr= x, raddr1= x, raddr2= x
# 40 op=xx, buff2_waddr= x, rdatal=( x, x), rdata2=( x, x)
# 40 waddr= x, wdatal=( x, x)
#
# 60 buff_op=10, buff_waddr= 7, raddr1=10, raddr2=23
# 60 op=xx, buff2_waddr= x, rdatal=( x, x), rdata2=( x, x)
# 60 waddr= x, wdatal=( x, x)
#
# 80 buff_op=10, buff_waddr= 3, raddr1=23, raddr2= 1
# 80 op=10, buff2_waddr= 7, rdatal=( 1, 2), rdata2=( -2, -2)
# 80 waddr= x, wdatal=( x, x)
#
# 100 buff_op=10, buff_waddr=11, raddr1= 1, raddr2=23
# 100 op=10, buff2_waddr= 3, rdatal=( -2, -2), rdata2=( 14, 9)
# 100 waddr= 7, wdatal=( 2, -6)
#
# 120 buff_op=10, buff_waddr=27, raddr1=10, raddr2= 9
# 120 op=10, buff2_waddr=11, rdatal=( 14, 9), rdata2=( -2, -2)
# 120 waddr= 3, wdatal=( -10, -46)
#
# 140 buff_op=00, buff_waddr= 2, raddr1= 7, raddr2=18
# 140 op=10, buff2_waddr=27, rdatal=( 1, 2), rdata2=( -2, -35)
# 140 waddr=11, wdatal=( -10, -46)
#
# 160 buff_op=00, buff_waddr= 8, raddr1=27, raddr2=15
# 160 op=00, buff2_waddr= 2, rdatal=( 2, -6), rdata2=( -60, -27)
# 160 waddr=27, wdatal=( 68, -39)
#
# 180 buff_op=00, buff_waddr= 2, raddr1= 7, raddr2= 9
# 180 op=00, buff2_waddr= 8, rdatal=( 68, -39), rdata2=( 23, -7)
# 180 waddr= 2, wdatal=( -58, -33)
#
# 200 buff_op=00, buff_waddr= 5, raddr1=11, raddr2=17
# 200 op=00, buff2_waddr= 2, rdatal=( 2, -6), rdata2=( -2, -35)
# 200 waddr= 8, wdatal=( 91, -46)
#
# 220 buff_op=01, buff_waddr= 6, raddr1= 1, raddr2= 8
# 220 op=00, buff2_waddr= 5, rdatal=( -10, -46), rdata2=( 89, 54)
# 220 waddr= 2, wdatal=( 0, -41)

```

