گزارش پروژه طراحی پردازنده
درس معماری کامپیوتر
نام استاد: دکتر آتنا عبدی

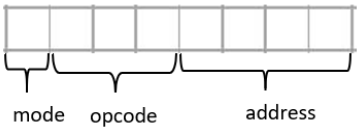
اعضای گروه :

مهدیه رحمتی 9822083

غزل پوراسفندیار بروجنی 9820453

بهار 1400

◄ 6 دستورالعمل داریم. پس می توان با 3 بیت به هر یک opcode نسبت دهیم.
◄ از آنجایی که دستورات فقط حافظه ای هستند، با یک بیت می توان mode آدرس دهی را نشان داد. (I=1 برای غیرمستقیم و I=0 برای مستقیم)
◄ و در نهایت از کل 8 بیت، 4 بیت باقی مانده برای آدرس دهی استفاده می شود.



پس می توان گفت واحد حافظه دارای 16 خط 8 بیتی می باشد.
ثبات هایی که نیاز داریم:

- ثبات PC برای نگهداری آدرس دستورالعمل جاری که نوبت fetch شدن آن است. ----> 4 بیتی
- ثبات IR برای نگهداری دستوری که به تازگی از حافظه fetch شده است. ----> 8 بیتی
- ثبات AR برای نگهداری آدرسی از حافظه که داده در آن قرار دارد. ----> 4 بیتی
- ثبات DR برای نگهداری داده ها ----> 8 بیتی
- ثبات AC برای نگهداری خروجی واحد حساب (از نوع داده) ----> 8 بیتی

Register	PC	AR	IR	DR	AC
Size(bit)	4	4	8	8	8

add	D0	AC ← AC + M[AR] , E← cout
sub	D1	AC ← AC - M[AR]
xor	D2	AC ← AC ⊕ M[AR]
multiply by 2	D3	1 : SHL WITH ALU 2 : AC ← M[AR] DR ← AC AC ← AC + AC
load	D4	AC ← M[AR]
store	D5	M[AR] ← AC
cmp	D6	AC ← M[AR] AC ← ~AC M[AR] ← AC

طبق چرخه فون نیومن، یک دستورالعمل در 4 مرحله انجام می شود.

T0 : AR ← PC (نگهداری آدرس دستورالعمل)

T1 : IR ← M[AR] , PC ← PC+1

T2 : I ← IR(7) , AR ← IR(0-3) (مرحله کدگشایی)

T3 : AR ← M[AR] or NOTHING (تشخیص مود آدرس دهی)

تا به اینجای کار می دانیم که آدرس داده در AR وجود دارد.

اکنون به سراغ تعریف ریز دستورالعمل ها (micro instruction) ها می رویم که هریک از کلاک چهارم به بعد اجرا خواهند شد. همچنین همزمان در 3 بیت برای دستورات opcode تعریف می کنیم.

Instruction	Opcode	Micro instructions
add	000	D0T4 : DR ← M[AR] D0T5 : AC ← AC + DR , SC ← 0
sub	001	D1T4 : DR ← M[AR] D1T5 : AC ← AC - DR , SC ← 0
xor	010	D2T4 : DR ← M[AR] D2T5 : AC ← AC ⊕ DR , SC ← 0
multiply by 2	011	D3T4 : AC ← M[AR] D3T5 : DR ← AC D3T6 : AC ← AC + DR D3T7 : M[AR] ← AC , SC ← 0
load	100	D4T4 : AC ← M[AR] , SC ← 0
store	101	D5T4 : M[AR] ← AC , SC ← 0
cmp	110	D6T4 : AC ← M[AR] D6T5 : AC ← ~AC D6T6 : M[AR] ← AC , SC ← 0

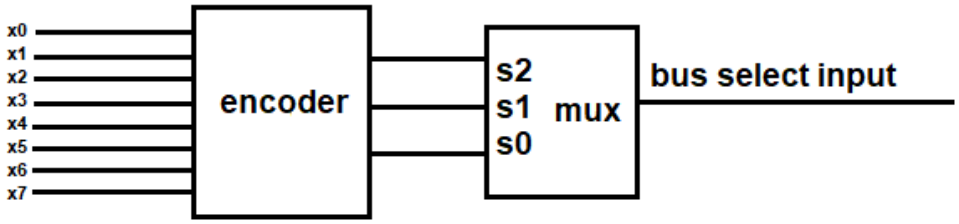
ماکزیمم 7 کلاک داشتیم پس یک sequence counter نیاز داریم که از 0 تا 7 بشمارد. در واقع یه شمارنده 3 بیتی برای ما کافی است.

اکنون نحوه اتصالات را با استفاده از جداول و توضیحات بالا به دست می آوریم:

PC	load	~RT1
IR	load	~RT1
AR	load	~R (T0 + T2) + I T3
DR	load	D0T4 + D1T4 + D2T4 + D3T5 = (D0 + D1 + D2)T4 + D3T5

AC	load	load ندارد چون نمی تواند از باس چیزی بردارد
Memory Unit	read	(D0 + D1 + D2 + D3 + D4 + D5 + D6)T4 + ~R(T1 +I T3)
	write	D3T7 + D5T4 + D6T6

برای مدیریت باس نیاز به یک دیکدر و یک مالتی پلکسر مطابق شکل زیر داریم و باید اتصالات xi ها را بیابیم.

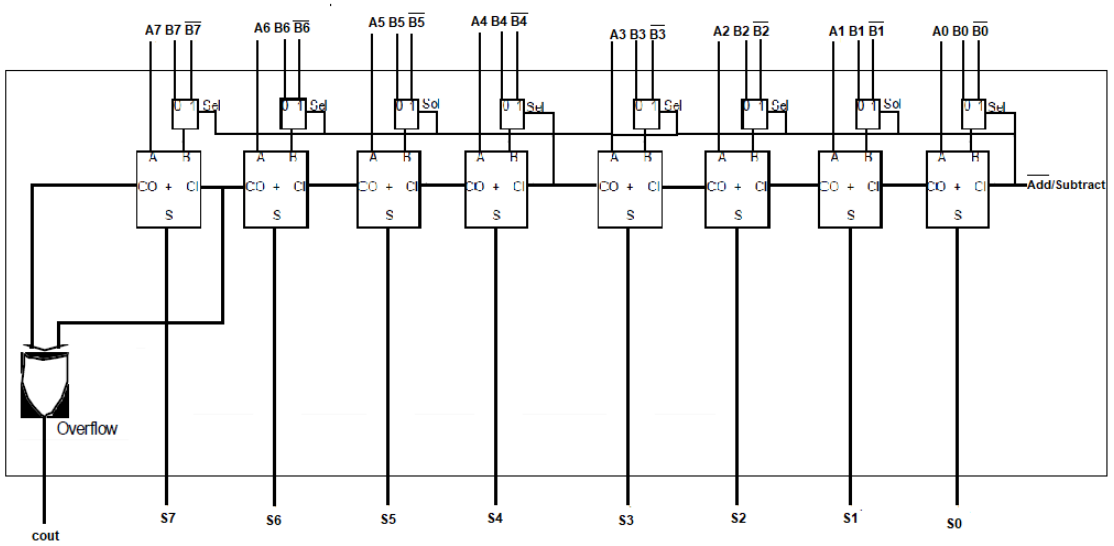


x0	PC	~RT0
x1	IR	~RT2
x2	AR	
x3	DR	D0T5 + D2T5 + D3T6
x4	AC	D3T5 + D3T7 + D5T4 +D6T6
x5	Memory unit	(D0 + D1 + D2 + D3 + D4 + D6)T4 + ~RT1 + I T3

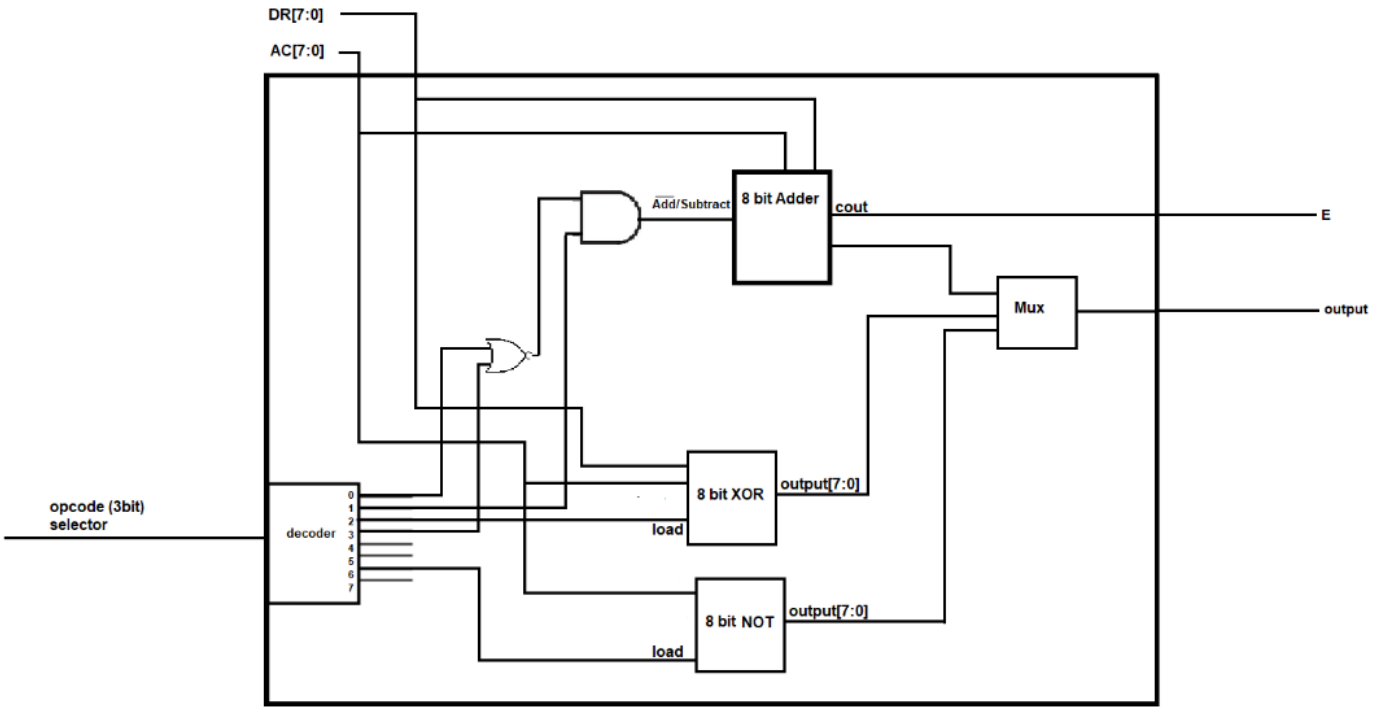
در طراحی واحد ALU به یک جمع/تفریق کننده 8 بیتی(برای عملیات ضرب در 2 هم کافیست عدد را با خودش جمع کنیم.) و یک گیت XOR نیاز داریم.

برای طراحی جمع کننده 8 بیتی، از جمع کننده آبشاری (سریال) استفاده می کنیم. برحسب اینکه عملیات چیست و چه opcode ای دارد، ورودی های جمع کننده تعیین می شوند.

جمع کننده :



واحد ALU:



پیاده سازی به زبان verilog :

تعریف ماژول CPU و reg های مورد نیاز:

```
module CPU(clk);
        input clk;
        //auxiliary regs
        reg [2:0]SC = 0;
        reg [7:0]AC =0 ;
        reg [3:0]PC = 0;
        reg [3:0]AR;
        reg [7:0]IR;
        reg [2:0]opcode;
        reg I;
        reg IR0, IR1, IR2, IR3, IR4, IR5, IR6, IR7;
        reg [7:0]M[0:15]; //16 x 8bit memory
```

مقدار دهی حافظه : خانه های حافظه را به دلخواه پر کردیم. محتویات هر خانه داده 8 بیتی است که دو معنا دارد (یا دستور است که باید آن را کدگشایی کرد و یا مقدارش مد نظر است.).

	binary value	decimal value	instruction description
M[0]	00001000	8	direct address, add, AC = AC + M[8]
M[1]	00011000	24	direct address, sub, AC = AC - M[8]
M[2]	00101000	40	direct address, xor, AC = AC ⊕ M[8]
M[3]	00111000	56	direct address, mul, M[8] = M[8] * 2
M[4]	01001000	72	direct address, load , AC = M[8]
M[5]	01011000	88	direct address, store, M[8] = AC
M[6]	01101000	104	direct address, cmp, M[8] = ~M[8]
M[7]	10001001	137	indirect address, add, AC = AC + M[M[9]]
M[8]	00001001	9	direct address, add, AC = AC + M[9]
M[9]	00001100	12	direct address, add, AC = AC + M[12]
M[10]	00001000	8	direct address, add, AC = AC + M[8]
M[11]	00001001	9	direct address, add, AC = AC + M[9]
M[12]	00001000	8	direct address, add, AC = AC + M[8]
M[13]	00001000	8	direct address, add, AC = AC + M[8]
M[14]	00001001	9	direct address, add, AC = AC + M[9]
M[15]	00001000	8	direct address, add, AC = AC + M[8]

```
13          //assignments
14          initial
15          begin
16                  M[0] = 8'b00001000;
17                  M[1] = 8'b00011000;
18                  M[2] = 8'b00101000;
19                  M[3] = 8'b00111000;
20                  M[4] = 8'b01001000;
21                  M[5] = 8'b01011000;
22                  M[6] = 8'b01101000;
23                  M[7] = 8'b10001001;
24                  M[8] = 8'b00001001;
25                  M[9] = 8'b00001000;
26                  M[10] = 8'b00001000;
27                  M[11] = 8'b00001001;
28                  M[12] = 8'b00001000;
29                  M[13] = 8'b00001000;
30                  M[14] = 8'b00001001;
31                  M[15] = 8'b00001000;
32          end
33
```

مراحل T0 تا T3 را بر اساس شمارنده SC به شرح زیر آوردیم:

```
34          always @ (posedge clk)
35          begin
36                  case(SC)
37                  3'b000: begin
38                          AR[3:0] = PC[3:0];
39                          SC = SC + 1;
40                          end
41                  3'b001: begin
42                          PC = PC + 1;
43                         {IR} = {M[AR]};
44                          SC = SC + 1;
45                          end
46                  3'b010: begin
47                          IR7 = IR[7];
48                          IR6 = IR[6];
49                          IR5 = IR[5];
50                          IR4 = IR[4];
51                          IR3 = IR[3];
52                          IR2 = IR[2];
53                          IR1 = IR[1];
54                          IR0 = IR[0];
55                          I = IR7;
56                          {opcode} = {IR6, IR5, IR4};
57                          {AR} = {IR3, IR2, IR1, IR0};
58                          SC = SC + 1;
59                          end
60                  3'b011: begin
61                          if (I == 1)
62                                  assign {AR} = {M[AR]};
63                          SC = SC + 1;
64                          end
```

در T4 نوبت به چک کردن opcode می رسد و تعیین می شود چه عملیاتی باید انجام شود(در پایان هر عملیات شمارنده صفر می شود تا چرخه تمام شود.) :

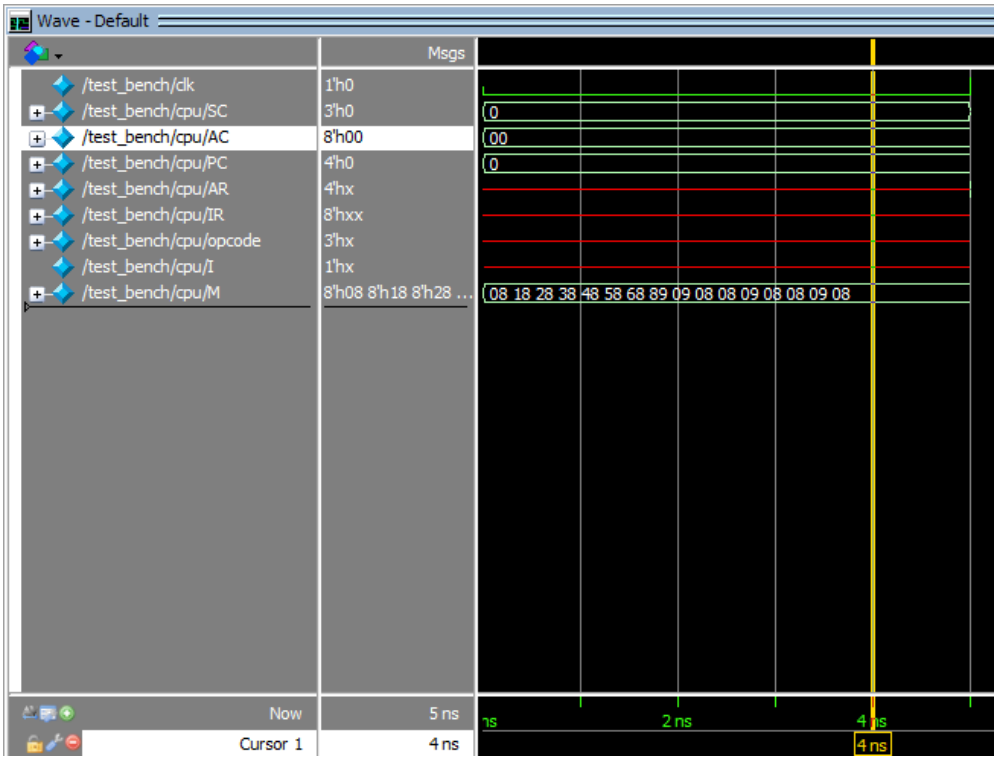
```
65                  3'b100: begin
66                          case(opcode)
67                          3'b000: begin
68                                  AC = AC + M[AR];
69                                  SC = 0;
70                                  end
71                          3'b001: begin
72                                  AC = AC - M[AR];
73                                  SC = 0;
74                                  end
75                          3'b010: begin
76                                  AC = AC ^ M[AR];
77                                  SC = 0;
78                                  end
79                          3'b011: begin
80                                  M[AR] = M[AR] + M[AR];
81                                  SC = 0;
82                                  end
83                          3'b100: begin
84                                  AC = M[AR];
85                                  SC = 0;
86                                  end
87                          3'b101: begin
88                                  M[AR] = AC;
89                                  SC = 0;
90                                  end
91                          3'b110: begin
92                                  M[AR] = ~M[AR];
93                                  SC = 0;
94                                  end
95                          endcase
96                          end
97                  endcase
98          end
99  endmodule
100
```

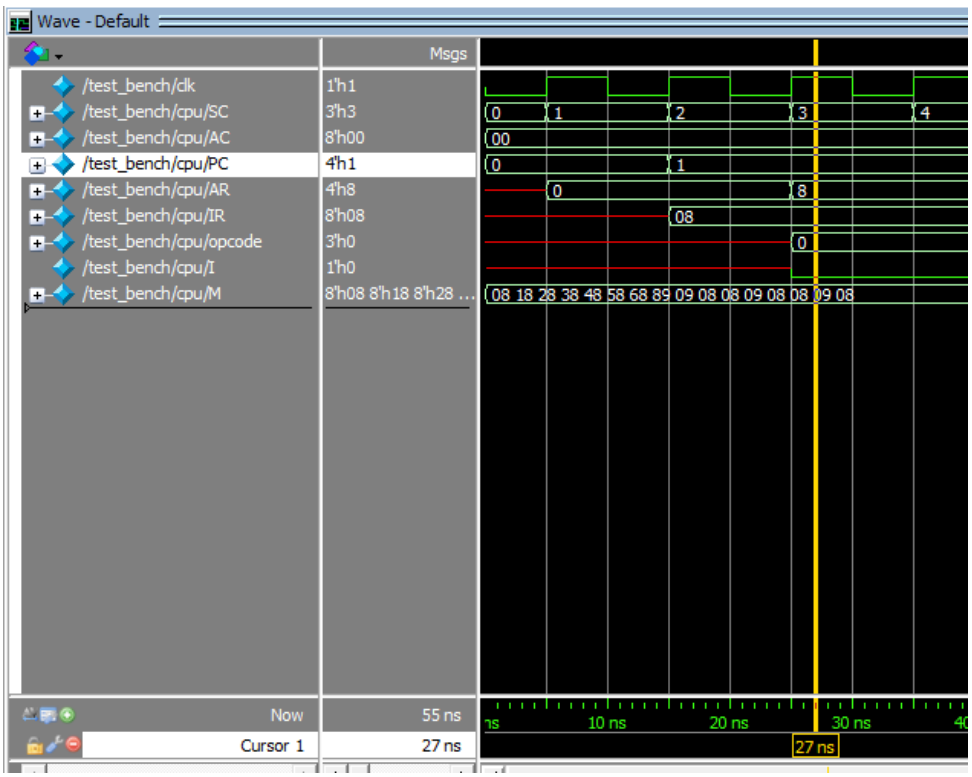
در آخر یک تست بنچ برای چک کردن درستی عملکرد پردازنده نوشتیم: که در آن هر 5 نانوثانیه کلاک تغییر وضعیت می دهد و به مدت 1000 نانو ثانیه برنامه اجرا می گردد:

```
101 module test_bench;
102         reg clk;
103
104         CPU cpu(.clk(clk));
105         initial
106         begin
107                 clk <= 0;
108
109                 #1000
110                 $finish;
111         end
112
113         always #5 clk = ~clk;
114 endmodule
```

شبیه سازی :

در ابتدا مقادیر PC, AC, SC صفر هستند.

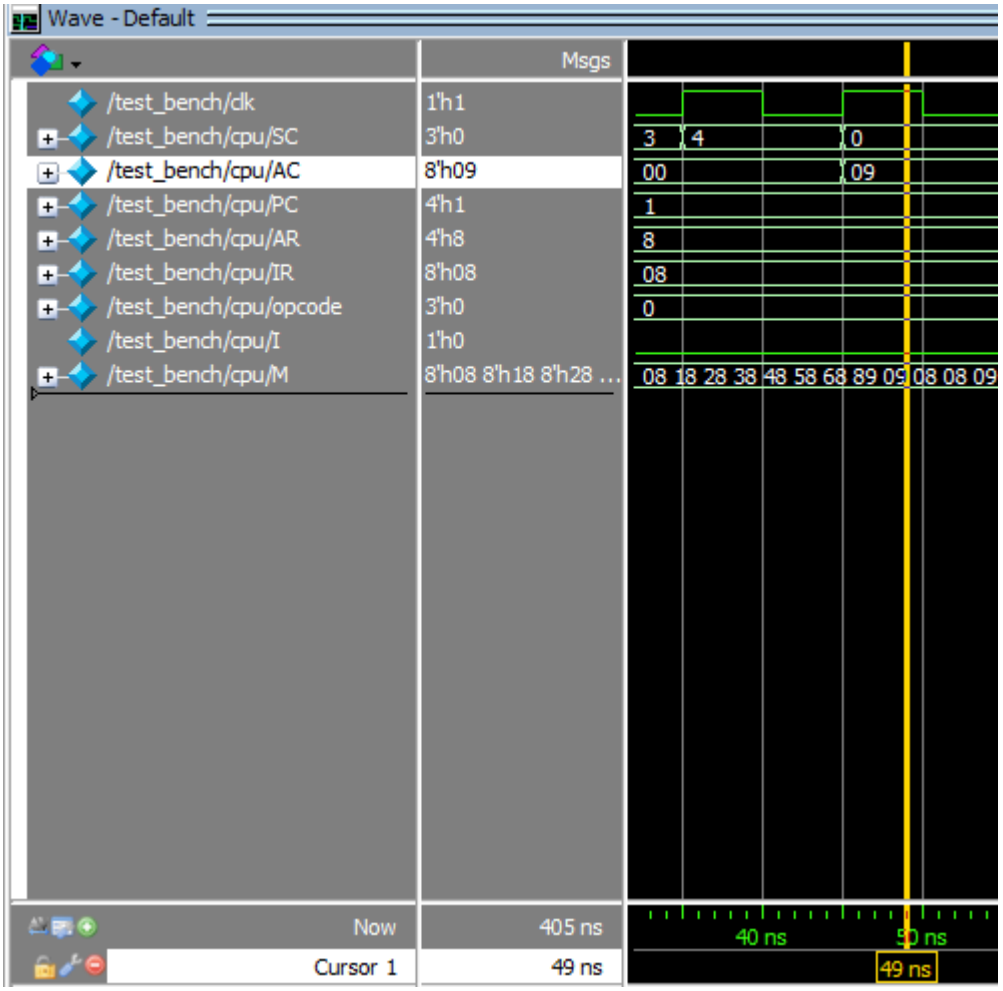




مقدار PC برابر صفر است (دقت شود که مقدار قبلی PC مد نظر است زیرا تا قبل از T4 که نوبت اجرای دستورالعمل است، PC یک واحد اضافه شده است و مقدار فعلی آن مد نظر نیست.). یعنی دستور در خانه شماره 0 حافظه قرار دارد. که یک جمع با آدرس دهی مستقیم است.

M[0]	00001000	8	direct address, add, AC = AC + M[8]

پس مقدار موجود در خانه 8 حافظه که برابر 9 بوده با AC که صفر بوده جمع شده و در AC ذخیره می شود و در نهایت حاصل AC برابر 9 می شود.

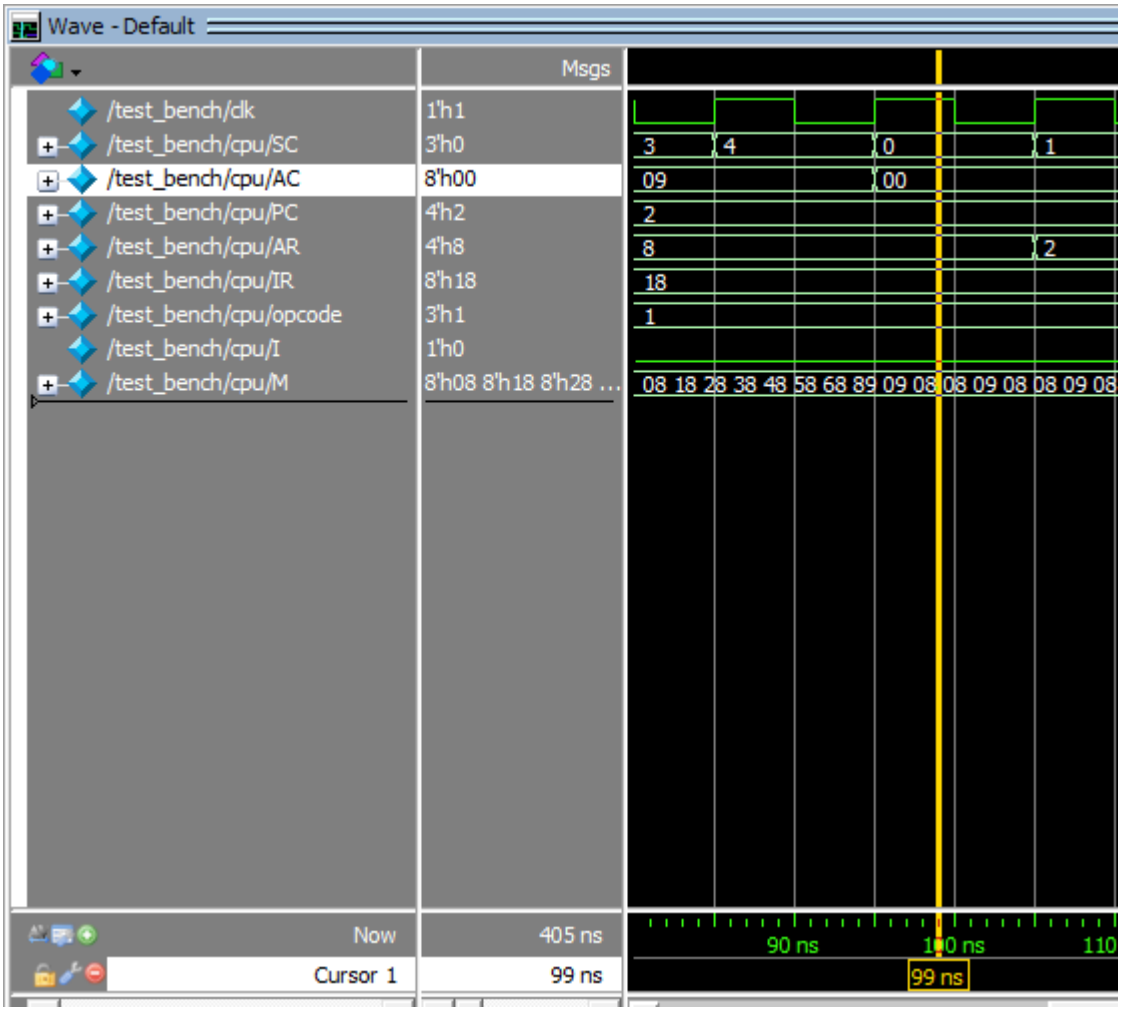


جمع به درستی انجام شد.

اکنون طبق کد، PC یک واحد افزایش می یابد و برابر 1 می شود که این به این معناست که دستور العمل بعدی در خانه شماره 1 حافظه قرار دارد که حاصل تفریق محتوای خانه شماره 8 حافظه با آدرس دهی مستقیم از AC و ذخیره آن در AC می باشد.

M[1]	00011000	24	direct address, sub, AC = AC - M[8]

از AC که قبلا مقدار 9 را داشت اکنون مقدار موجود در خانه 8 حافظه (مقدار9) کم شده و در نهایت صفر می شود.



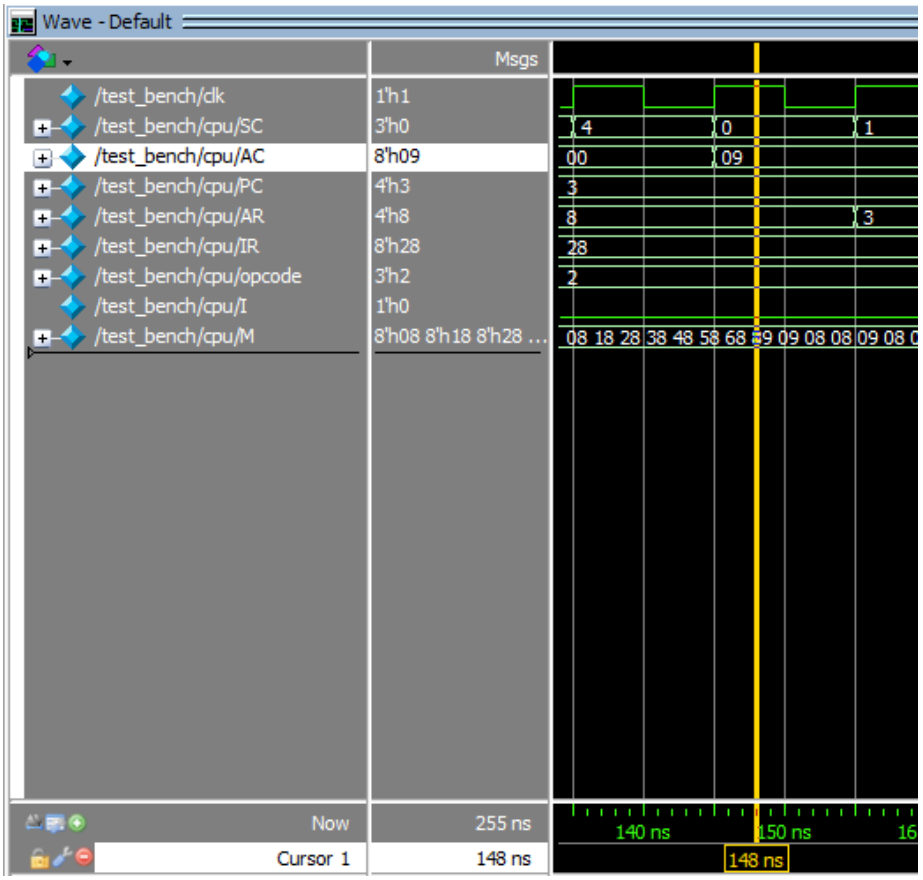
تفریق به درستی انجام شد.

مجددا روال قبل طی شده و این بار دستور خانه شماره 2 انجام می شود. که حاصل xor مقدار موجود در خانه 8 حافظه (مقدار 9) با مقدار AC (که صفر است) می باشد.

M[2]	00101000	40	direct address, xor, AC = AC ⊕ M[8]

طبق رابطه زیر، انتظار می رود حاصل AC برابر 9 شود.

$$0000'1001 \oplus 0000'0000 = 0000'1001$$

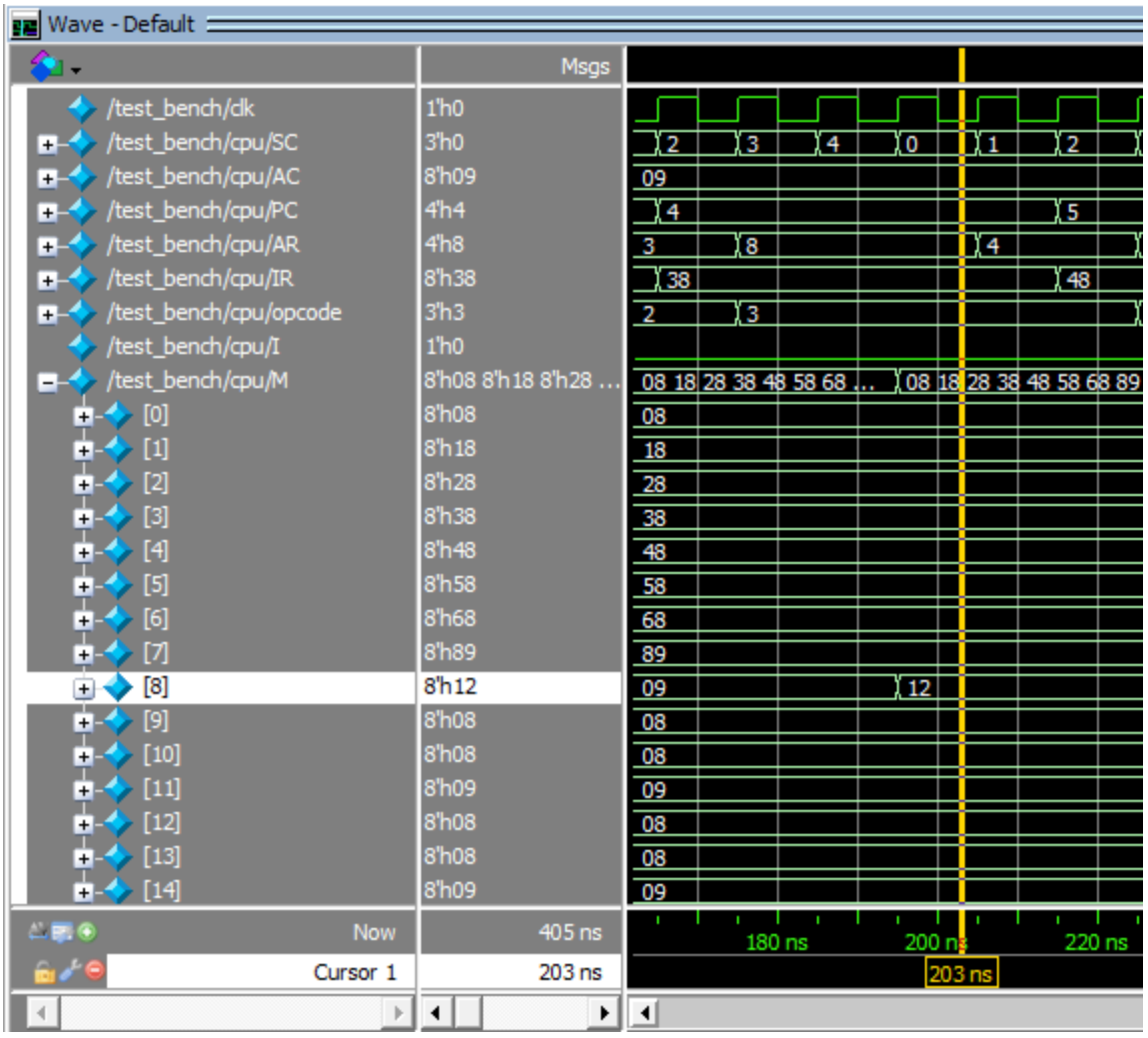


عملیات xor هم به درستی انجام شد.

اکنون که PC مقدار 3 را دارد به سراغ خانه سوم حافظه می رویم. دستور این خانه، ضرب مقدار موجود در خانه شماره AR در 2 است.

M[3]	00111000	56	direct address, mul, M[8] = M[8] * 2

همانطور که پیداست مقدار AR برابر 8 است و محتویات قبلی خانه شماره 8 حافظه برابر 9 بوده است. پس انتظار می رود دوبرابر شده و مقدار 18 در آن ذخیره شود. (اعداد در شبیه سازی در مبنای 16 هستند)



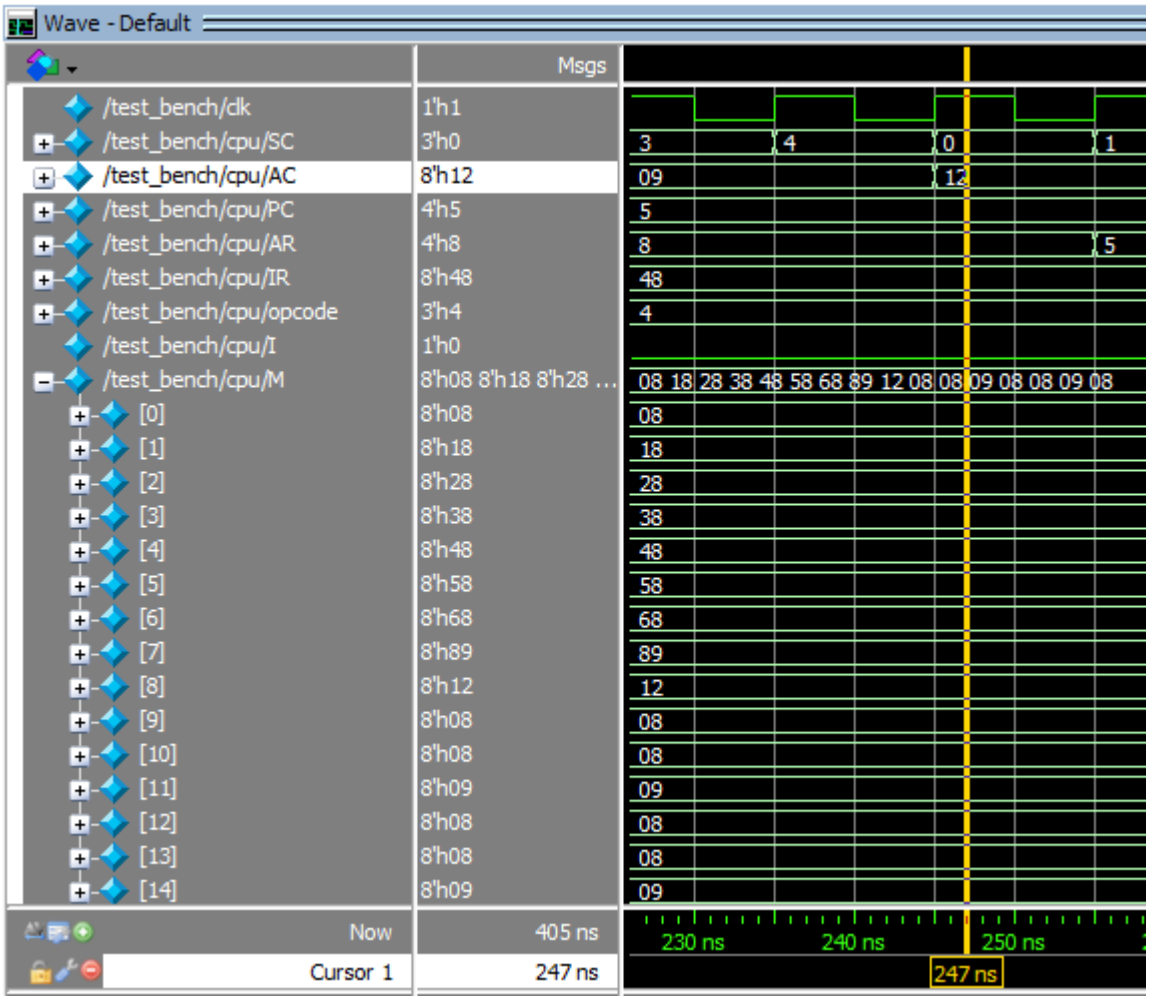
مقدار 12 در مبنای 16 همان 18 = 2 + 16 در مبنای 10 است.

پس عملیات ضرب در 2 به درستی انجام شد.

مقدار PC برابر 4 است. پس دستور خانه شماره 4 حافظه را می خوانیم. که همان load کردن مقداری از آدرس AR در AC است.

M[4]	01001000	72	direct address, load , AC = M[8]

مقدار AR برابر 8 است. پس به سراغ خانه 8 ام حافظه رفته و مقدار موجود در آن را در AC میریزیم. که در اینجا همان مقداری است که در مرحله قبل ذخیره کردیم (18 دسیمال و 12 در مبنای 16)



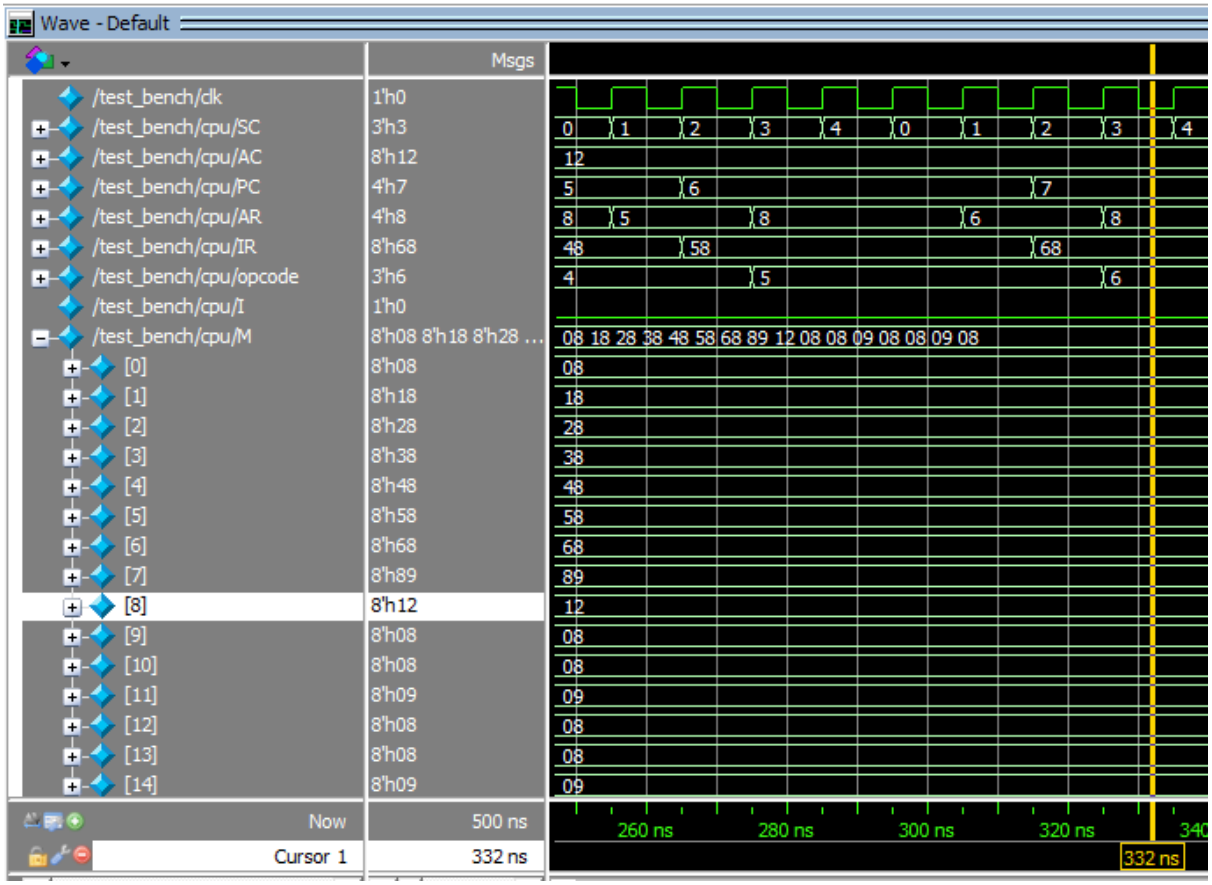
همانطور که انتظار داشتیم مقدار AC هم همان 12 در مبنای 16 می باشد.

پس عملیات load به درستی انجام گرفت.

مقدار PC در اینجا 5 است و باید عملیات store انجام شود و مقدار موجود در AC در خانه شماره AR از حافظه ذخیره شود.

M[5]	01011000	88	direct address, store, M[8] = AC

در اینجا AR همان 8 است پس به خانه شماره 8 حافظه می رویم و مقدار فعلی AC که 12 است را جایگزین مقدار فعلی این خانه (که از قضا آن هم 12 است)، می کنیم. چون این مقدار جایگزین شده ولی به لحاظ مقداری تغییر نکرده است، تغییرات این بخش در شکل موج ها مشخص نیست.

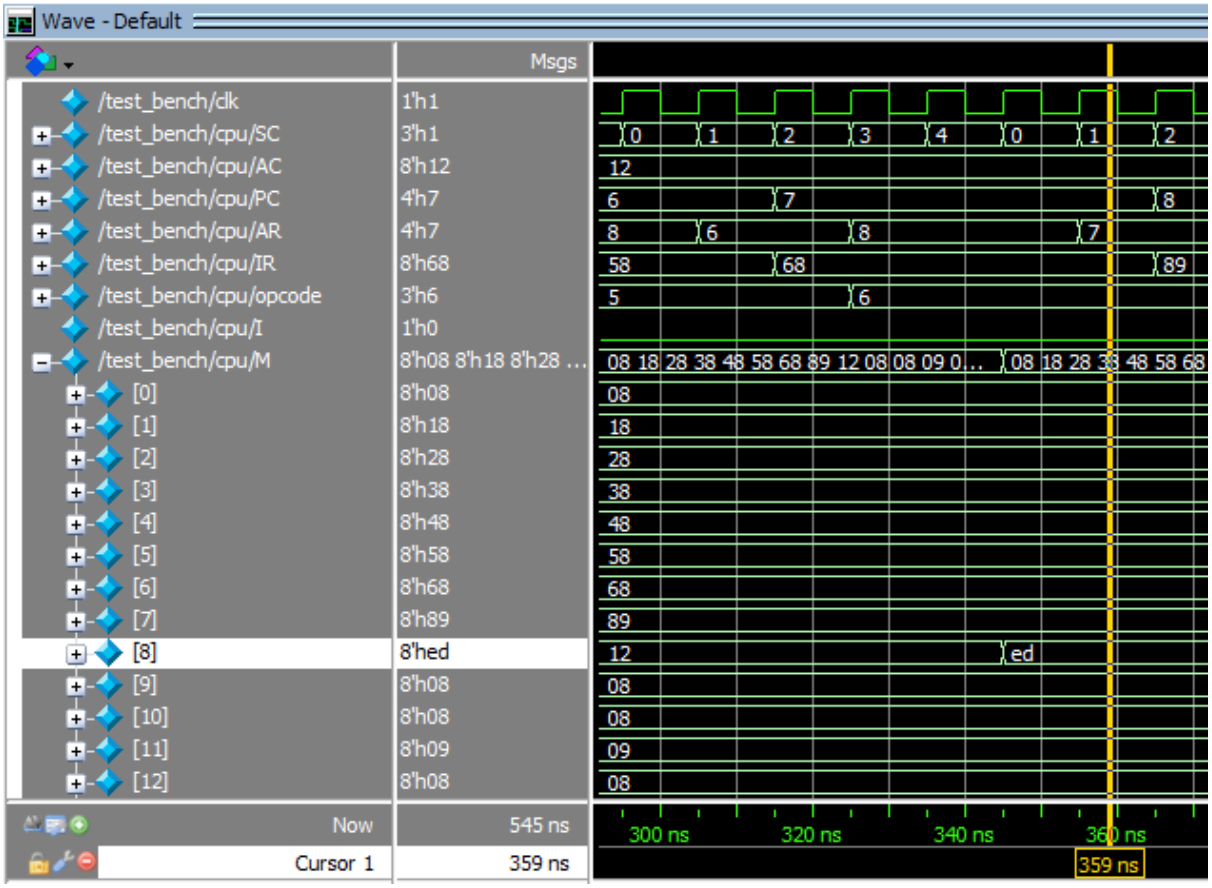


عملیات store هم به درستی انجام گرفت.

مقدار PC در اینجا برابر 6 است که عملیات cmp باید انجام شود به طوری که تک تک بیت های مقدار موجود در خانه شماره AR حافظه، not شوند.

M[6]	01101000	104	direct address, cmp, M[8] = ~M[8]

در اینجا AR همان 8 است. و محتویات آن 12 در مبنای 16 و به باینری 0001'0010 می باشد که not آن، برابر 1110'1101 می باشد که در بنای 16 برابر ed است.



همانطور که انتظار می رفت مقدار ed در خانه شماره 8 حافظه ذخیره شد.

پس دستور cmp هم به درست انجام شد.

برای آنکه مود آدرس دهی غیر مستقیم را هم چک کنیم، مرحله بعد که PC برابر 7 است را اجرا می کنیم. که به شرح زیر است.

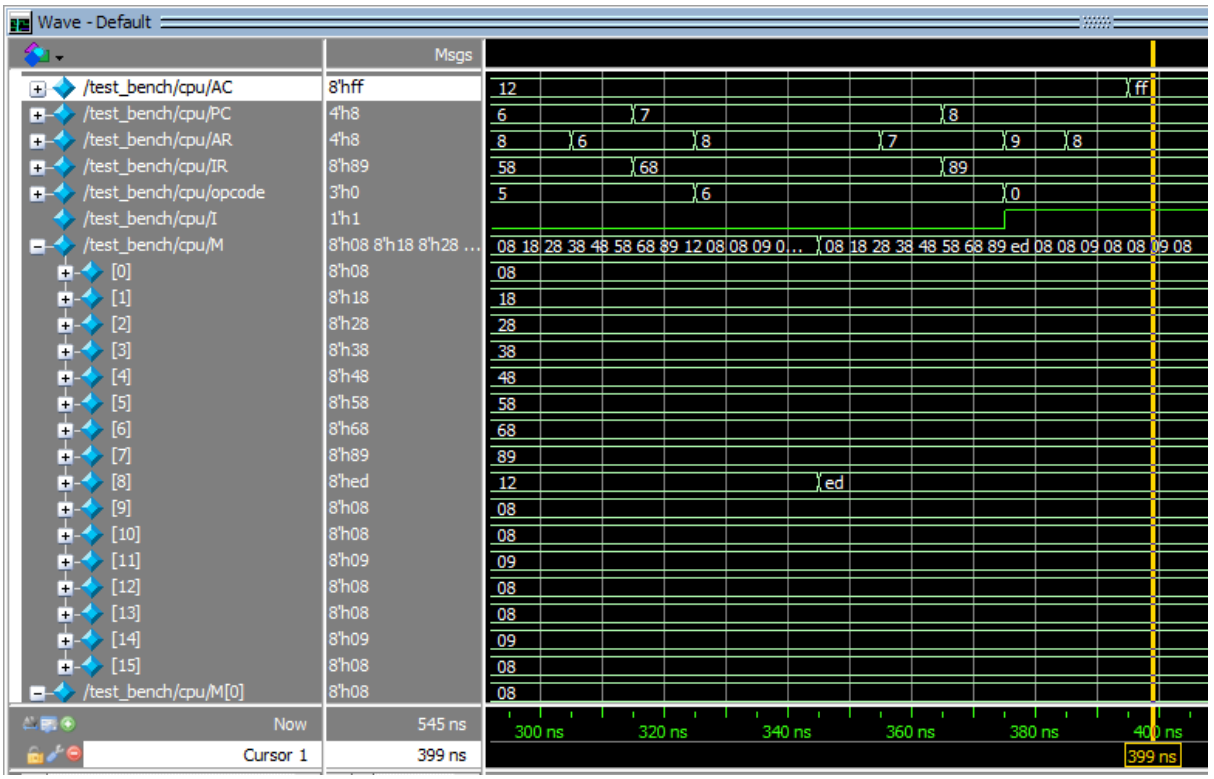
M[7]	10001001	137	indirect address, add, AC = AC + M[M[9]]

M[M[9]] = M[8] = 11101101

مقدار موجود در خانه 9 ام برابر 8 است. پس به سراغ خانه شماره می رویم که مقدار ed در مبنای 16 که همان 1110'1101 در باینری است را در خود دارد. مقدار موجود در AC که برابر 18 دسیمال (12 در مبنای 16) است را با مقدار به دست آمده در بالا جمع می کنیم و در AC ذخیره می کنیم:

1110’1101 + 0001’0010 = 1111’1111

که این مقدار در مبنای 16 برابر با ff است.



نتایج، همانطور است که انتظار می رفت.

پس مود آدرس دهی غیر مستقیم هم به درستی کار می کند.