تمرین ۸

سوال ۱

رابطه مربوط به AMAT به صورت بازگشتی برای سیستمی که یک لایه cache, یک memory و یک virtual memory داشته باشد به صورت زیر محاسبه می‌شود.

$$AMAT = t_{\text{cache}} + MR_{\text{cache}}(t_{MM} + MR_{MM}t_{VM}) \qquad (8.2)$$

برای این سوال دو لایه cache داریم اما virtual memory نداریم, که در این صورت رابطه بازگشتی به شکل زیر تبدیل خواهد شد:

```
AMAT = Tcl1 + MRcl1 (Tcl2 + MRcl2.Tmm)
=> AMAT = 1ns + 0.05(5ns + 0.15.(100ns))
=> AMAT = 2ns
```

سوال ۲

ساختار cache:

V	Tag	Data	V	Tag	Data

Tag	Set	Byte Offset
28 bits	2 bits	2 bits(00)

تعداد کل بلاک‌ها در حافظه نهان(ا)	تعداد مجموعه‌ها(ب)	بلوک‌ها در مجموعه(ب)	index(ج) بیت‌های	tag(ج) بیت‌های
۸	۴	۲	۲	۲۸

د) بررسی تعداد و نرخ hit و miss:

```
int i,j,sum=0;

  for (i=0 ; i< SIZE ; i++)
      for (j=0 ; j < SIZE ; j++)
          array[i][j] = i+j+1;
```

V	Tag	Data	V	Tag	Data
1	(Tag of i in stack)	0-7		(Tag of an array cell)	
1	(Tag of j in stack)	0-7		(Tag of an array cell)	
1	(Tag of sum in stack)	0		(Tag of an array cell)	
	(Tag of an array cell)			(Tag of an array cell)	

در این قسمت از کد با هر write روی i و j مقادیر این دو متغییر قبل از فراخوانی بروزرسانی شده‌اند, بنابراین تعداد miss ها برابر صفر خواهد بود. همچنین خانه‌های آرایه هم به همین ترتیب با هر write روی cache نوشته میشوند اما در این قسمت از کد هیچ دستوری مقادیر درون آرایه را فراخوانی نمیکند.

به این ترتیب تعداد hit ها:

```
first for loop:
 8 times hit on loading variable i on i < SIZE

second for loop:
 8 times hit on loading variable j on j < SIZE

array[i][j] = i+j+1; :
 64 times hit each time for i,j * 8


=>


hits= 8 + 8 + 64*8 = 528
```

```
for (i=0 ; i < SIZE ; i++)
      for (j=0 ; j < SIZE ; j++)
          sum+=  array[i][j];
```

at the beginning:

V	Tag	Data	V	Tag	Data
1	(Tag of i in stack)	0	1	(Tag of array cell[7][4])	12
1	(Tag of j in stack)	0	1	(Tag of array cell[7][5])	13
1	(Tag of array cell[7][2])	10	1	(Tag of array cell[7][6])	14
1	(Tag of array cell[7][3])	11	1	(Tag of array cell[7][7])	15

در این قسمت هیچکدام از خانه‌های آرایه hit نخواهند شد چرا که هر خانه فقط یکبار فراخوانی می‌شود و دیگر مورد استفاده قرار نخواهد گرفت. متغییر های i , j در سمت دیگر, در هر بار فراخوانی hit خواهند شد. حتی در اولین بار فراخوانی به دلیل sw که در i=0 , j=0 در اولین دستور حلقه انجام می‌شود مقادیر آنها درست خواهد بود. متغییر sum بار اول miss و در باقی فراخوانی‌ها hit می‌شود.

به این ترتیب تعداد hit ها:

```
first for loop:
 8 times hit on loading variable i on i < SIZE

second for loop:
 8 times hit on loading variable j on j < SIZE

sum+=  array[i][j]; :
```

```
   1 miss and 63 hits for sum,
   64 misses on every array invocation
```

=>

```
  hits= 8 + 8 + 63 = 79
  misses= 64
```

OVERALL RESULTS:

```
  hits= 79+528 = 607
  misses = 64
  all = 671
  hit rate = 607/671 = 90.46 %
  miss rate = 9.53 %
```

سوال ۳

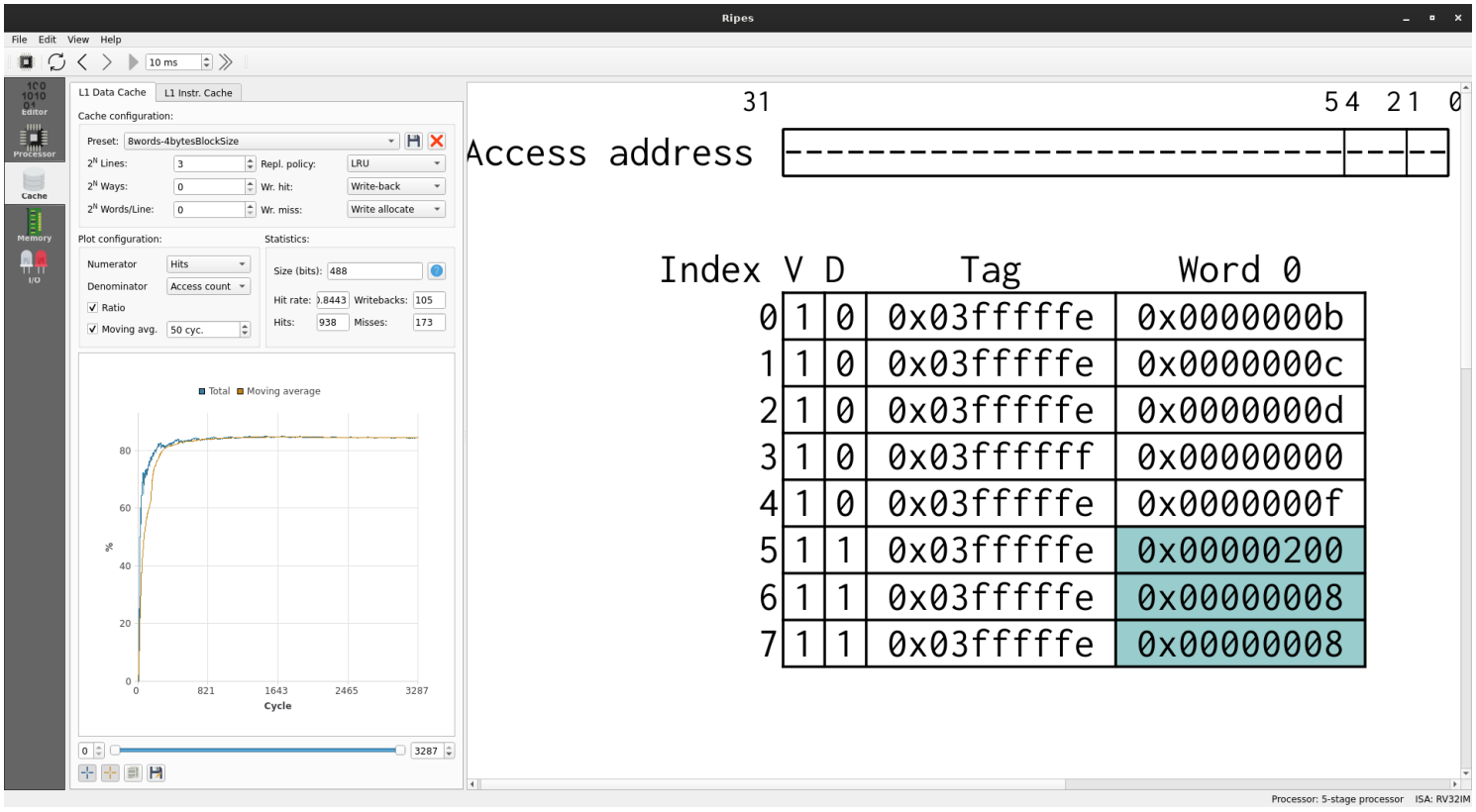
کد اسمبلی (با استفاده از ابزار https://godbolt.org) :

```
  main:
          addi    sp,sp,-288
          sw      s0,284(sp)
          addi    s0,sp,288
          sw      zero,-28(s0)
          sw      zero,-20(s0)
          j       L2
  L5:
          sw      zero,-24(s0)
          j       L3
  L4:
          lw      a4,-20(s0)
          lw      a5,-24(s0)
          add     a5,a4,a5
          addi    a4,a5,1
          lw      a5,-20(s0)
          slli    a3,a5,3
          lw      a5,-24(s0)
          add     a5,a3,a5
          slli    a5,a5,2
          addi    a5,a5,-16
          add     a5,a5,s0
          sw      a4,-268(a5)
          lw      a5,-24(s0)
          addi    a5,a5,1
          sw      a5,-24(s0)
  L3:
          lw      a4,-24(s0)
          li      a5,7
          ble     a4,a5,L4
          lw      a5,-20(s0)
          addi    a5,a5,1
          sw      a5,-20(s0)
```

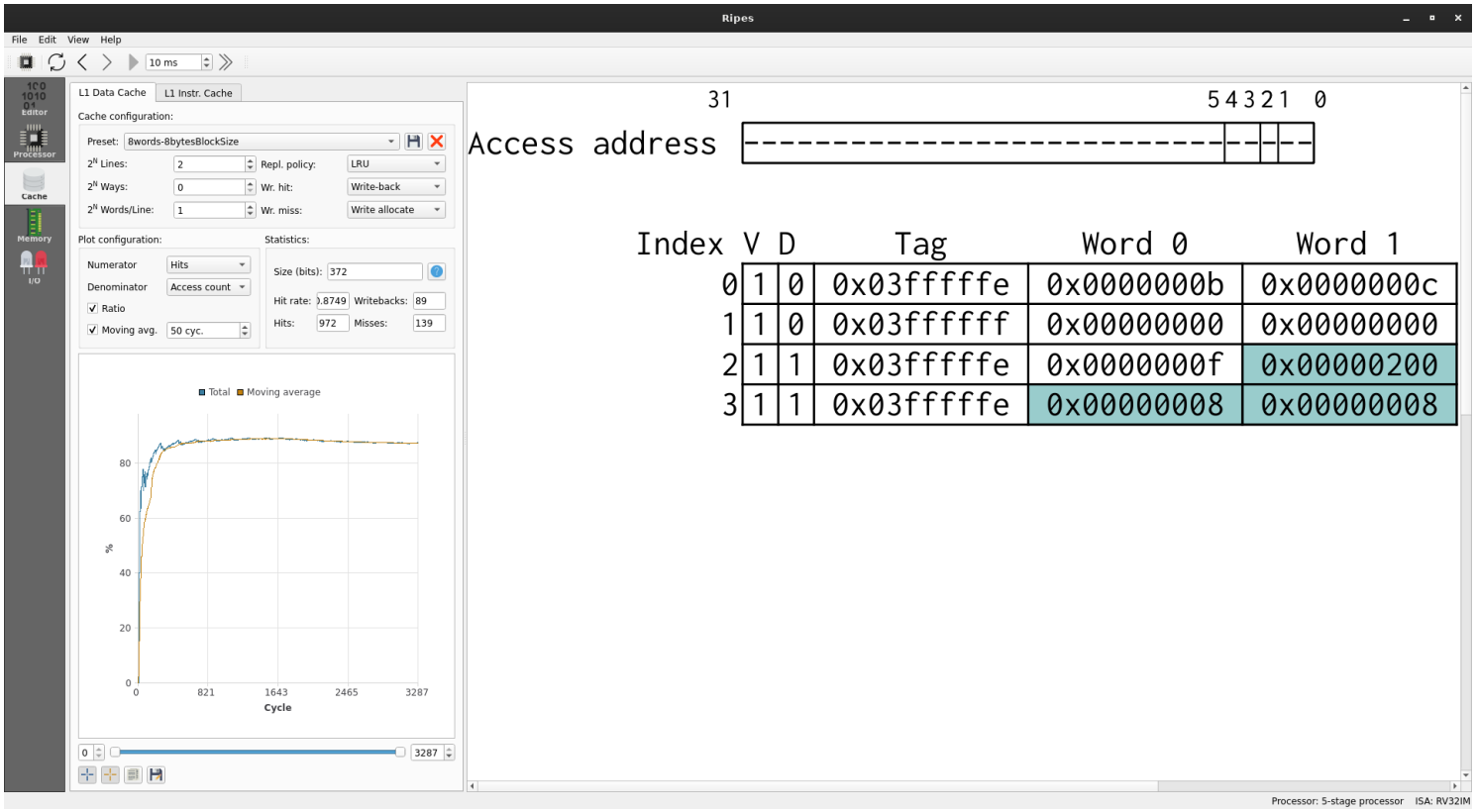
```
L2:
        lw      a4,-20(s0)
        li      a5,7
        ble     a4,a5,L5
        sw      zero,-20(s0)
        j       L6
L9:
        sw      zero,-24(s0)
        j       L7
L8:
        lw      a5,-20(s0)
        slli    a4,a5,3
        lw      a5,-24(s0)
        add     a5,a4,a5
        slli    a5,a5,2
        addi    a5,a5,-16
        add     a5,a5,s0
        lw      a5,-268(a5)
        lw      a4,-28(s0)
        add     a5,a4,a5
        sw      a5,-28(s0)
        lw      a5,-24(s0)
        addi    a5,a5,1
        sw      a5,-24(s0)
L7:
        lw      a4,-24(s0)
        li      a5,7
        ble     a4,a5,L8
        lw      a5,-20(s0)
        addi    a5,a5,1
        sw      a5,-20(s0)
L6:
        lw      a4,-20(s0)
        li      a5,7
        ble     a4,a5,L9
        li      a5,0
        mv      a0,a5
        lw      s0,284(sp)
        addi    sp,sp,288
        #jr      ra
```

الف) هر ۴ bytes = word

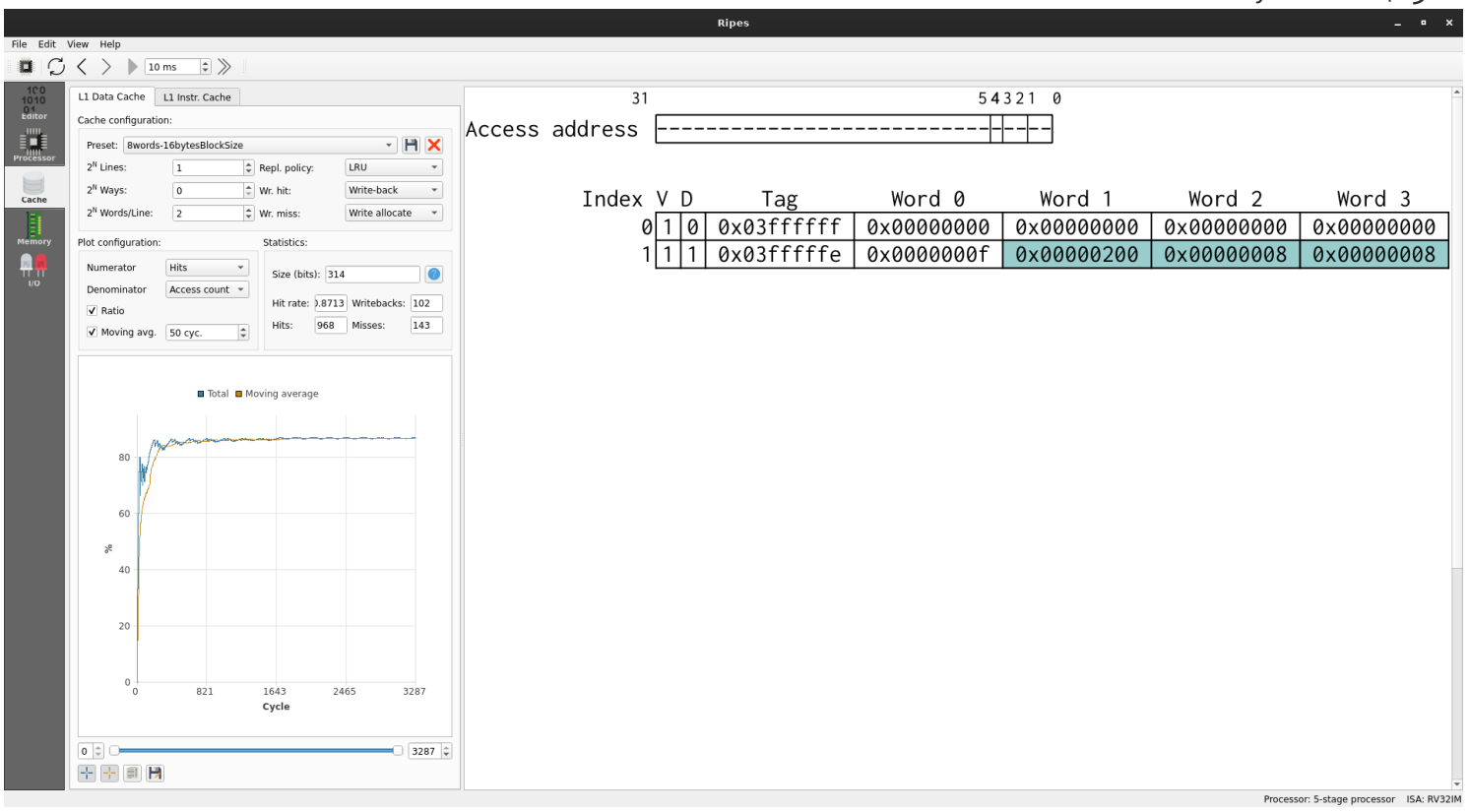
اندازه بلاک 4 bytes:



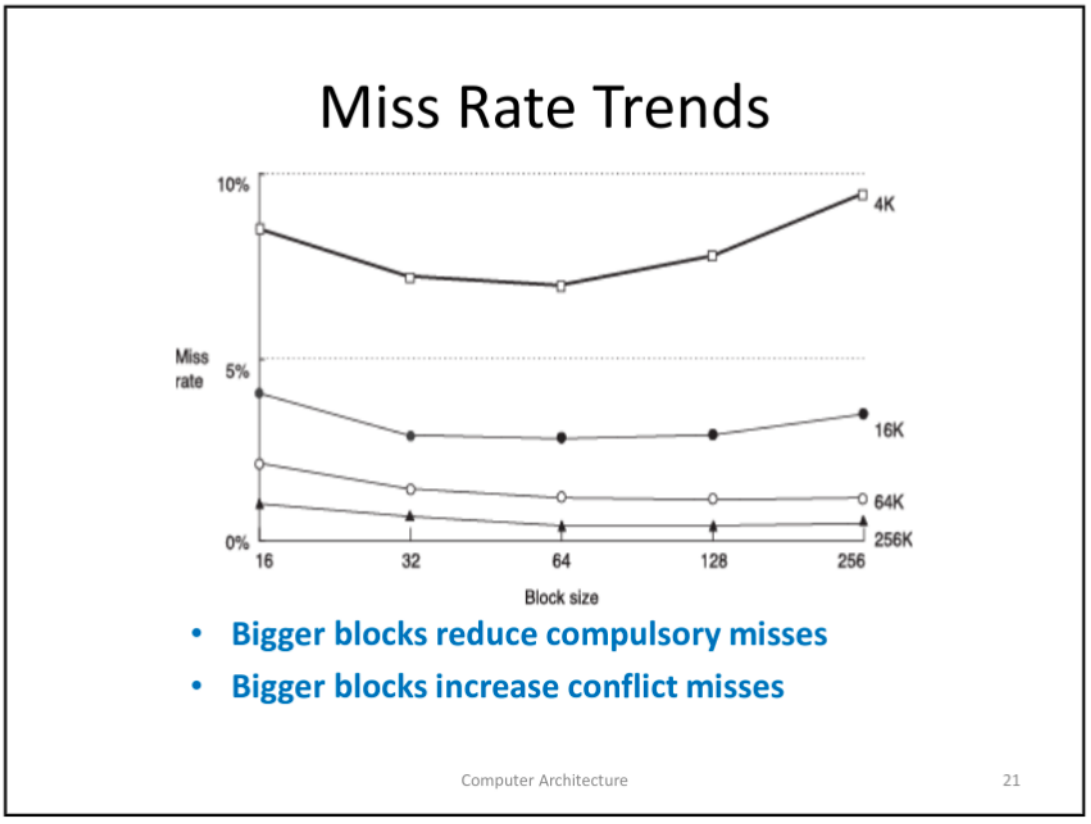
اندازه بلاک 8 bytes:



اندازه بلاک 16 bytes:



با افزایش اندازه بلاک‌ها نرخ miss تا حدی کاهش می‌یابد چرا که با این روش از **هم‌محلیت فضایی** بهتر استفاده میکنیم چرا که پیمایش آرایه عمده کار این کد می‌باشد و بدین ترتیب موجب کاهش تعداد **Compulsory miss** ها میشویم. این مورد در افزایش اندازه بلاک از ۴ به ۸ قابل مشاهده است.



اما در نقطه مقابل در افزایش بعدی اندازه بلاک, همان طور که در نمودار بالا قابل مشاهده است, این افزایش موجب افزایش **Conflict miss** ها خواهد شد. در نتیجه بهره‌وری کمتری از فضای حافظه نهان در اندازه بلاک ۱۶ برده‌ایم.

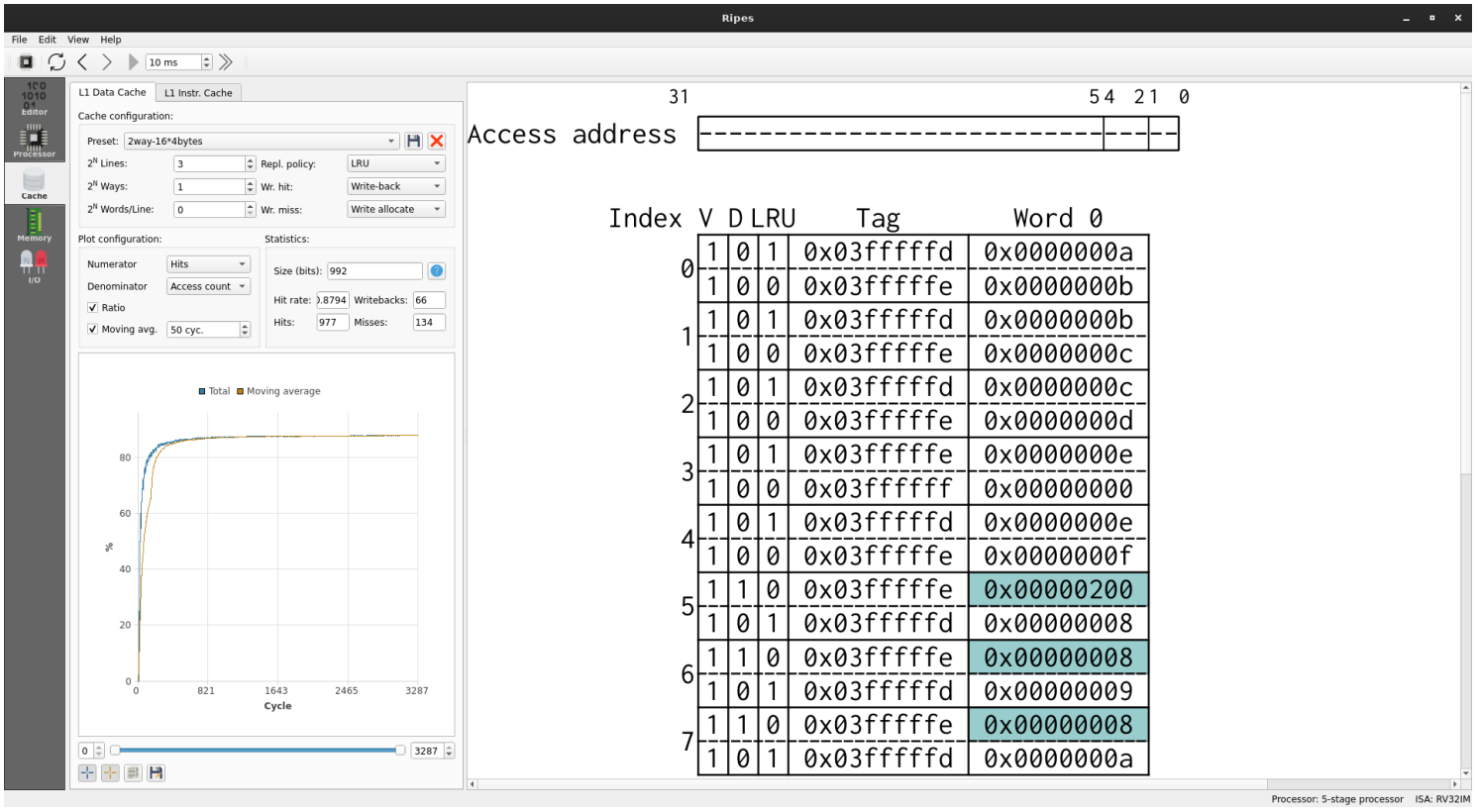
b	Hit rate
4	84.43 %
8	87.49 %
16	87.13 %

ب)

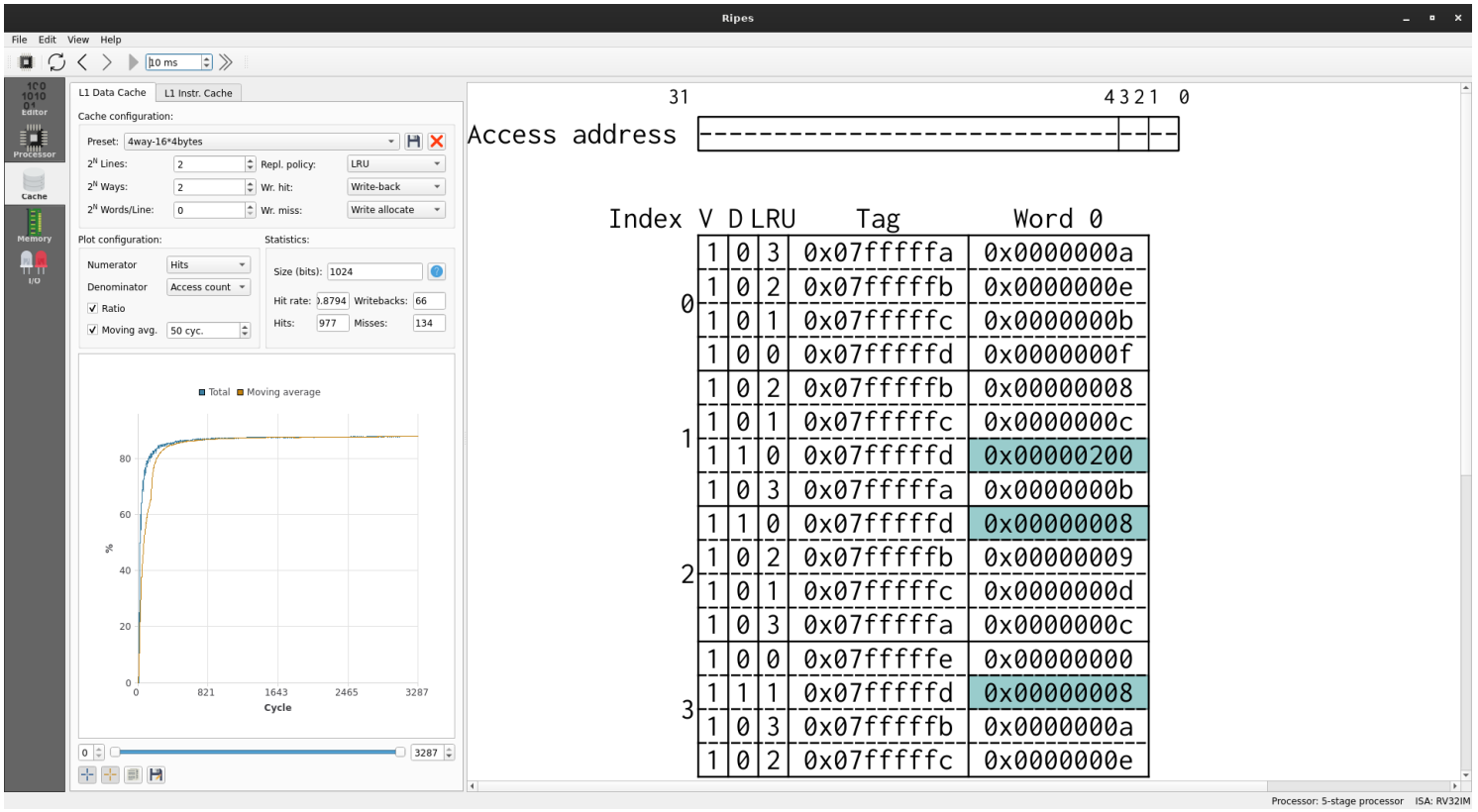
ظرفیت: 4*16 bytes

اندازه بلاک: 4

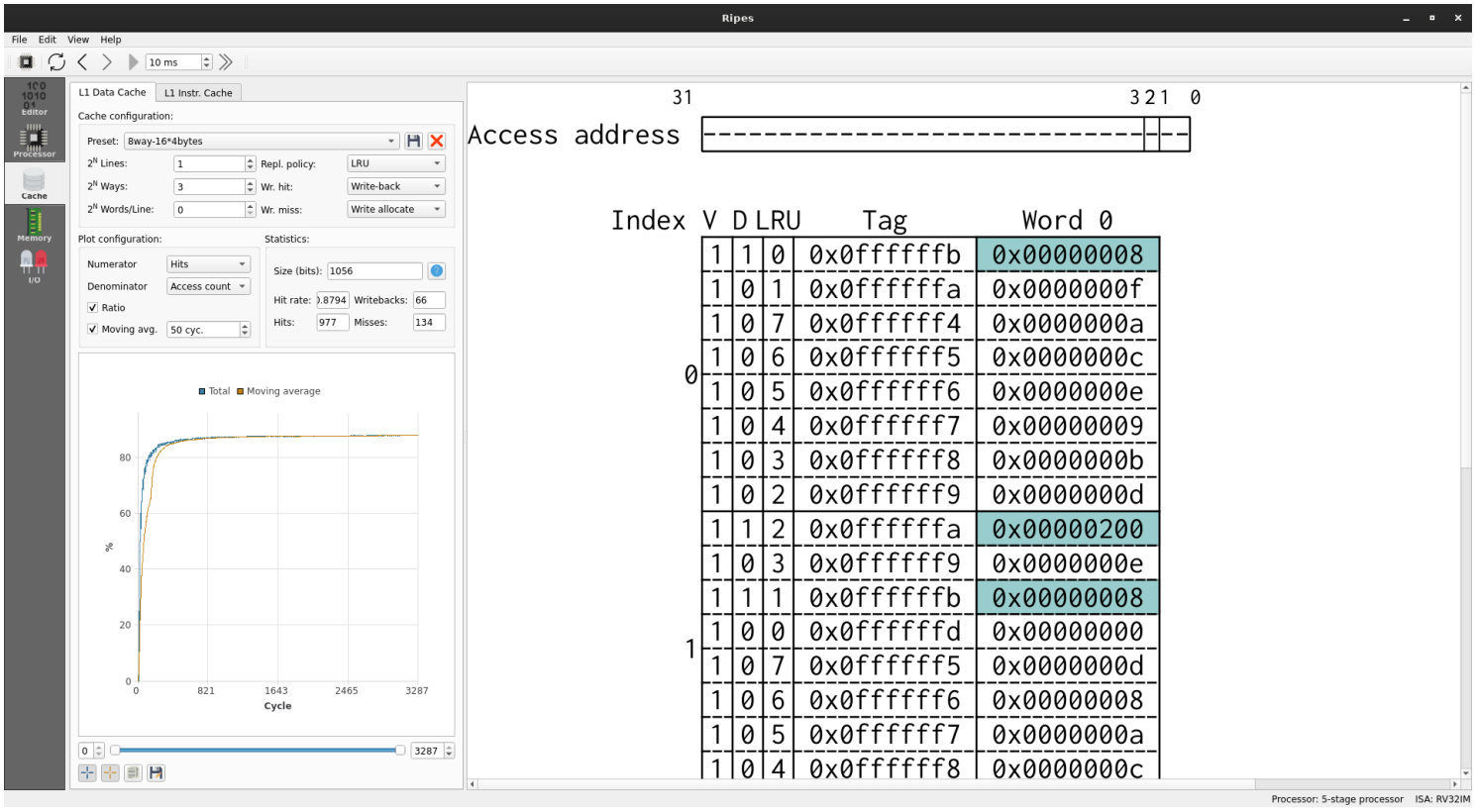
2-way:


Ripes
File Edit View Help
10 ms
L1 Data Cache
L1 Instr. Cache
Cache configuration:
Preset: 2way-16*4bytes
2^N Lines: 3
Repl. policy: LRU
2^N Ways: 1
Wr. hit: Write-back
2^N Words/Line: 0
Wr. miss: Write allocate
Plot configuration:
Numerator Hits
Denominator Access count
Ratio
Moving avg. 50 cyc.
Statistics:
Size (bits): 992
Hit rate: .8794
Writebacks: 66
Hits: 977
Misses: 134
Total
Moving average
%
Cycle
0
821
1643
2465
3287
Access address
31
54 21 0
Index V D LRU Tag Word 0
0 1 0 1 0x03fffffd 0x0000000a
1 0 0 0x03ffffffe 0x0000000b
1 1 0 1 0x03fffffd 0x0000000b
1 0 0 0x03ffffffe 0x0000000c
2 1 0 1 0x03fffffd 0x0000000c
1 0 0 0x03ffffffe 0x0000000d
3 1 0 1 0x03ffffffe 0x0000000e
1 0 0 0x03fffffff 0x00000000
4 1 0 1 0x03fffffd 0x0000000e
1 0 0 0x03ffffffe 0x0000000f
5 1 1 0 0x03ffffffe 0x00000200
1 0 1 0x03fffffd 0x00000008
6 1 1 0 0x03ffffffe 0x00000008
1 0 1 0x03fffffd 0x00000009
7 1 1 0 0x03ffffffe 0x00000008
1 0 1 0x03fffffd 0x0000000a
Processor: 5-stage processor ISA: RV32IM

4-way:


Ripes
File Edit View Help
10 ms
L1 Data Cache
L1 Instr. Cache
Cache configuration:
Preset: 4way-16*4bytes
2^N Lines: 2
Repl. policy: LRU
2^N Ways: 2
Wr. hit: Write-back
2^N Words/Line: 0
Wr. miss: Write allocate
Plot configuration:
Numerator Hits
Denominator Access count
Ratio
Moving avg. 50 cyc.
Statistics:
Size (bits): 1024
Hit rate: .8794
Writebacks: 66
Hits: 977
Misses: 134
Total
Moving average
%
Cycle
0
821
1643
2465
3287
Access address
31
4321 0
Index V D LRU Tag Word 0
0 1 0 3 0x07fffffa 0x0000000a
1 0 2 0x07fffffb 0x0000000e
1 0 1 0x07fffffc 0x0000000b
1 0 0 0x07fffffd 0x0000000f
1 1 0 2 0x07fffffb 0x00000008
1 0 1 0x07fffffc 0x0000000c
1 1 0 0x07fffffd 0x00000200
1 0 3 0x07fffffa 0x0000000b
2 1 1 0 0x07fffffd 0x00000008
1 0 2 0x07fffffb 0x00000009
1 0 1 0x07fffffc 0x0000000d
1 0 3 0x07fffffa 0x0000000c
3 1 0 0 0x07ffffffe 0x00000000
1 1 1 0x07fffffd 0x00000008
1 0 3 0x07fffffb 0x0000000a
1 0 2 0x07fffffc 0x0000000e
Processor: 5-stage processor ISA: RV32IM

8-way:



با افزایش درجه associativity می‌توان miss هایی از جنس **conflict** را پوشش داد. اما ظاهرا در مورد این کد, افزایش این درجه تاثیری چندانی روی بهبود نرخ miss ندارد چرا که miss هایی که از جنس conflict بوده‌اند در مرحله اول مهاجرت از نگاشت مستقیم برطرف شده و سایز فعلی حافظه نهان با افزایش درجه associativity نمی‌تواند کمکی به اجرای سریع‌تر برنامه کند.

N	Hit rate
2	87.94 %
4	87.94 %
8	87.94 %

سوال ۴

الف)

قبل از افزودن TLB:

```
AMAT = ATc + MRc(ATmm + ATmm + MRmm(AThd))
=> AMAT = 1 + 0.02(100 + 100 + 0.000003(1000000) )
=> AMAT = 5.06 cycles
```

بعد از افزودن TLB:

```
AMAT = ATc + MRc(ATtlb + MRtlb(ATmm) + ATmm + MRmm(AThd))
=> AMAT = 1 + 0.02( 1 + 0.0005(100) + 100 + 0.000003(1000000) )
=> AMAT = 3.081 cycles
```

ب)

حافظه مجازی توانایی آدرس دهی تا ۲ به توان ۳۲ را دارد این یعنی فضای آدرس حافظه مجازی در 32 بیت خلاصه میشود که از آدرس `0x0000_0000` شروع و به `0xFFFF_FFFF` ختم می‌شود.

فضای آدرس حافظه فیزیکی هم در ۲۳ بیت قرار میگیرد.

اندازه هر page برابر با 4KB است در نتیجه ۱۲بیت مربوط به `Page offset` خواهد بود. ۳۲-۱۲=۲۰ بیت مربوط به VPN و ۲۳-۱۲=۱۱ بیت هم مربوط به PPN خواهد بود.

ساختار هر سطر ورودی و بیت‌های مورد نیاز آن به این شکل ترسیم می‌شود:

V	VPN	PPN
1	20	11

```
TLB_Size= 64*(1+20+11) = 2048 bits
```

ج)

V /1	VPN /20	PPN /11

.

.

64

.

.

د) برای ساخت این TLB نیاز به یک SRAM که فضای آدرس ۶۴ تایی داشته باشد داریم که در هر آدرس 4byte را در خود جای دهد. یعنی در نهایت 256B حافظه SRAM.