1- کامپیوتری را که دارای حافظه اصلی دارای ظرفیت 16 گیگابایت و حافظه نهان به بزرگی 2^{12} خط (Line) و اندازه هر بلاک 2^9 بیت و هر کلمه (Word) 4 بایت است ، را در نظر بگیرید. با فرض استفاده از روش نگاشت مستقیم و خالی بودن حافظه نهان در ابتدا، به سوالات زیر پاسخ دهید:

الف) هرکدام از میدان‌های word ، block ، Index و tag چند بیت به خود اختصاص می‌دهند؟

پاسخ:

نکته ای که در این سوال باید در نظر گرفته شود این هست که هر Word 4 بایت است پس یعنی خط آدرس حافظه اصلی بر اساس تعداد کلمات هست.

$$Main\ Memory = 16\ G\ Byte = 2^{34}\ Byte = \frac{2^{34}}{2^2} = 2^{32}\ Word \rightarrow m = 32\ bit\ address$$

$$Cache = 2^{12}\ Line = 2^{12} * 16 = 2^{16}\ Word \rightarrow c = 16\ bit\ address$$

$$Block = 2^9\ bit = \ 2^6\ byte = \frac{2^6}{2^2} = 2^4\ Word \rightarrow \ b = 4\ bit\ address$$

tag	Line / Block	Offset / Word
16 bit	12 bit	4 bit
	Index	
	16 bit	
Main Memory Address Field		
32 bit		

ب) نرخ موفقیت و وضعیت موفقیت (hit) یا (miss) را برای سری آدرس‌های زیر (به ترتیب از چپ به راست) به طور جداگانه مشخص کنید:

1) سری اول آدرس ها: (مبنای 10)

180, 179, 182, 177, 175, 179, 181, 190, 201, 200, 201, 173, 162, 168, 191, 189, 176, 177, 179, 180

چون اندازه آدرس های درخواستی بیشتر از 8 بیت نیست پس نیازی به بررسی tag وجود ندارد.
برای راحتی در محاسبات بهتر از همه آدرس های بالا را در مبنای 16 بنویسیم.
B4, B3, B6, B1, AF, B3, B5, BE, C9, C8, C9, AD, A2, A8, BF, BD, B0, B1, B3, B4
سپس داریم:
M , H , H , H , M , H , H , H , M , H , H , H , H , H , H , H , H , H , H , H

نرخ موفقیت = $\frac{17}{20} * 100 = 85\%$

2) سری دوم آدرس ها: (مبنای 16)
BFAD, BECD, CD87, CD8F, BFA1, BECA, DD81, CD88, CD81, BFAE, BFAD, BFAE, CD87
سپس داریم:
M , M , M , H , H , H , M , M , H , H , H , H , H

نرخ موفقیت = $\frac{8}{13} * 100 = 61.5\%$

ج) دو عاملی که باعث افزایش نرخ موفقیت می‌شوند را در فراخوانی های بالا مشخص بکنید.(می توانید آدرس ها را در مبنای 16 بنویسید).

همجواری مکانی High Light

همجواری زمانی Text Color

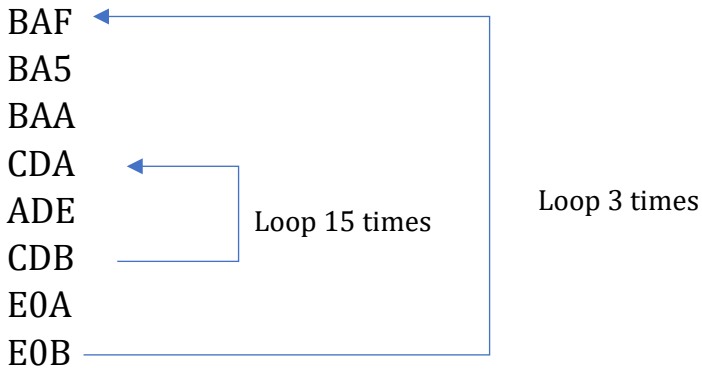
سری اول آدرس ها:

B4, B3, B6, B1, AF, B3, B5, BE, C9, C8, C9, AD, A2, A8, BF, BD, B0, B1, B3, B4

سری دوم آدرس ها:

BFAD, BECD, CD87, CD8F, BFA1, BECA, DD81, CD88, CD81, BFAE, BFAD, BFAE, CD87

2- برنامه‌ای را در نظر بگیرید که درخواست‌های دسترسی به حافظه‌ی آن به صورت زیر باشد.



تصور کنید قرارست این برنامه روی سیستمی با ویژگی‌های زیر اجرا شود: 1) حافظه‌ی نهان با 4 بلوک (بلوک 64 کلمه)، 2) حافظه‌ی اصلی 256 بلوکی و 3) نگاشت مستقیم در حافظه‌ی نهان.

الف) پس از اجرای کامل برنامه فوق، نرخ موفقیت حافظه‌ی نهان چقدر خواهد بود؟

ب) آیا وجود حافظه نهان در اجرای برنامه تاثیرگذار است یا خیر؟

ج) چطور می‌توان حافظه نهان را طوری تغییر داد که نرخ موفقیت افزایش چشمگیری داشته باشد (تغییر باید هزینه کمی داشته باشد)؟

د) آیا همواره افزایش اندازه بلوک باعث افزایش نرخ موفقیت می‌شود؟

پاسخ:

الف) طبق فرضیات سوال داریم:

tag	Line / Block	Offset / Word
4 bit	2 bit	6 bit
	Index	
	8 bit	
Main Memory Address Field		
12 bit		

BAF	BA5	BAA	CDA		ADE	CDB	E0A	E0B
M	H	H	M	H	M	M	M	H
			1t	14t	15 times	15 times		
H	H	H	H		M	M	H	H
			15 times		15 times	15 times		
H	H	H	H		M	M	H	H
			15 times		15 times	15 times		

s

نرخ موفقیت = $\frac{57}{150} * 100 = 38\%$

ب) با توجه به نرخ موفقت بدست آمده در قسمت قبل وجود حافظه نهان تاثیر چندانی در اجرای برنامه ندارد.

ج) همانطوری که می‌دانیم با تغییر اندازه بلاک می‌توان مقدار نرخ موفقیت را تغییر داد.

در دسترسی‌هایی که برنامه بالا داشته است همانطور که طبق بخش "الف" متوجه می‌شویم در حلقه داخلی به علت Miss شدن زیاد نرخ موفقیت کاهش یافته است. پس برای افزایش چشمگیر نرخ موفقیت باید 3 دسترسی CDA ، ADE و CDB در یک بلاک از حافظه نهان قرار بگیرند.

CDA			ADE			CDB		
1100	11	011010	1010	11	011110	1100	11	011011

همانطور که می‌بینیم دلیل Miss شدن حافظه نهان به دلیل یکی نبودن tag و یکی بودن شماره بلاک در حافظه‌ی نهان می‌باشد.

دو راه برای بهبود این موضوع وجود دارد:

راه حل اول افزایش اندازه حافظه‌ی نهان که هزینه زیادی دارد.

راه حل دوم تغییر اندازه بلاک برای تغییر پدیده همجواری مکانی در حافظه نهان که هزینه کمتری نسبت به حالت اول دارد. چون استفاده از حلقه باعث ایجاد دسترسی های متوالی می‌شود. و چون این دسترسی ها باعث Miss شدن شده است پس باید پدیده همجواری مکانی را افزایش بدهیم.

طبق جدول صفحه قبل برای 3 دسترسی چون شماره tag ها یکی نیست پس باید با تغییر اندازه بلاک(offset) شماره بلاک ها(Line) را در حافظه نهان همجواری مکانی را افزایش بدهیم. برای اینکار باید اندازه بلاک را به مقداری تغییر دهیم. تا شماره بلاک ها در 3 دسترسی قبلی یکی نباشد همچنین باید حواسمان باشد که این مقدار را به قدری تغییر دهیم که باعث کاهش hit های دسترسی های دیگر نشود یا حداقل مقدار ممکن شود.

چون 3 دسترسی داریم باید حداقل 2 بیت متفاوت در شماره بلاک آنها داشته باشیم. همانطور که می‌بینم 2 بیت پر ارزش در اندازه بلاک(offset) دارای مقدار های متفاوتی هستند. پس اگر ساختار میدان ها را به صورت زیر تغییر دهیم داریم:

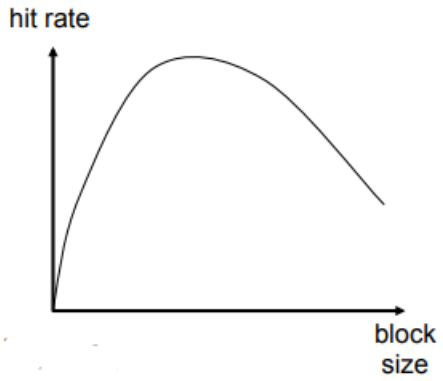
tag	Line / Block	Offset / Word
4 bit	6 bit	2 bit
	Index	
	8 bit	
Main Memory Address Field		
12 bit		

CDA			ADE			CDB		
1100	110110	10	1010	110111	10	1100	110110	11

BAF	BA5	BAA	CDA		ADE		CDB	E0A	E0B
M	M	M	M	H	M	H	H	M	H
			1t	14t	1t	14t	15 times		
H	H	H	H		H		H	H	H
			15 times		15 times		15 times		
H	H	H	H		H		H	H	H
			15 times		15 times		15 times		

نرخ موفقیت $= \frac{144}{150} * 100 = 96\%$

د) همانطور که در بخش "ج" ملاحظه کردیم با کاهش اندازه کلمه‌های داخل یک بلاک نرخ موفقیت اندازه چشمگیری داشت پس می‌توان گفت که افزایش اندازه یک بلاک می‌تواند تا مقدار مشخصی باعث افزایش نرخ موفقیت شود بعد از آن باعث کاهش نرخ موفقیت می‌شود. مطابق شکل زیر:



3- دو تابع first و second تعریف شده‌اند که مجموع مقادیر موجود در دو آرایه بزرگ A و B را حساب کنند. کدام یک از پیاده سازی‌ها از منظر سخت افزاری بهتر هست؟ این بهتر بودن چه نتیجه‌ای دارد؟ برای پاسخ خود دلیل بیاورید.

```
function first (*A, *B):
      for (int i = 0; i < 100000; i++)
          sum += A[i]

      for (int i = 0; i < 100000; i++)
          sum += B[i]

      return sum

function second (*A, *B):
      for (int i = 0; i < 100000; i++)
          sum += A[i]
          sum += B[i]

      return sum
```

پاسخ:

پیاده سازی تابع first بهتر است زبان‌های برنامه‌نویسی از دو روش برای ذخیره سازی آرایه دو بعدی استفاده می‌کنند که می‌توانید در اینجا بیشتر در مورد آنها مطالعه بکنید.

آرایه های این سوال یک بعدی هستند و اعضای یک آرایه با هم همجواری مکانی دارند بنابر این بهتر است که عملیات جمع به صورت جداگانه روی اعضای آرایه ها صورت بگیرد تا نرخ موفقیت بالا بماند و سرعت اجرای کد بیشتر شود.