גליון 1

פתרון לשאלה 1

סעיף א

בכל שנייה, המעבד עובד $16.7M$ מחזורי שעון, ואם קיים $Coprocessor$, אזי $CPI = 10$, כלומר כל 10 מחזורי שעון מתבצעת פקודה אחת, ולכן

$$MIPS_1 = \frac{16.7M}{10}\frac{1}{M} = 1.67$$

ללא $Coprocessor$, נתון כי $CPI = 7$, ולכן במקרה זה נקבל

$$MIPS_2 = \frac{16.7M}{7}\frac{1}{M} = 2.38$$

סעיף ב

עבור הרצה עם רכיב ה $Coprocessor$, התוכנית רצה למשך $1.08[\sec]$, ולכן נמשכה $CC = 16.7M \cdot 1.08 = 18.036M$ ($Clock\ Count$) מחזורי שעון. לכן, הורצו סה"כ

$$IC = \frac{CC}{CPI} = \frac{18.036M}{10} = 1.8036M$$

($Instruction\ Count$) פקודות.

עבור הרצה ללא רכיב ה $Coprocessor$, התוכנית רצה למשך $13.6[\sec]$, ולכן נמשכה $CC = 16.7M \cdot 13.6 = 227.12M$ מחזורי שעון. לכן, הורצו סה"כ

$$IC = \frac{CC}{CPI} = \frac{227.12M}{7} = 32.44M$$

פקודות.

סעיף ג

ראשית, נספור את מספר פקודות FP:

$$FPIC = 82,014 + 8,226 + 73,220 + 21,399 + 6,006 + 4,710 = 195,575$$

עם יחידת $Coprocessor$, הורצו סה"כ $1,803,600$ פקודות, ולכן ישנן $1,803,600 - 195,575 = 1,608,025$ פקודות $Integer$ בתוכנית.

ללא יחידת ה $Coprocessor$ הורצו $32.44M$ פקודות. לאחר החסרת מספר פקודות ה $Integer$ שבתוכנית, הרי שמתבצעות $32,440,000 - 1,608,025 = 30,831,975$ פקודות $Integer$ שהחליפו את פקודות ה FP. לכן, בממוצע, מספר פקודות ה $Integer$ הדרושות למימוש פקודות FP, הוא

$$\frac{30,831,975}{FPIC} = \frac{30,831,975}{195,575} \cong 158$$

פתרון לשאלה 2

נניח כי מחזור השעון הוא T.

בגישה הראשונה ניתן להעביר פקודה אחת כל מחזור שעון, ולכן

$BW_1 = 1$.

בגישה השנייה, ב 20% מהפעמים העברנו אפקטיבית רק מילה אחת בשני מחזורי שעון, וב 80% מהפעמים העברנו שתי מילים בשני מחזורי שעון. לכן

$$BW_2 = 80\% \cdot \frac{2}{2} + 20\% \cdot \frac{1}{2} = 0.8 + 0.1 = 0.9 .$$

פתרון לשאלה 3

בכל השאלה נניח כי מספר הבלוקים גדל באופן ליניארי עם רדיוס הדיסק.

סעיף א

נחשב את מהירות הקריאה של בלוק, בהנתן שבחרנו את מסילה ($Track$) X .
במסילה x , כאשר $x=1,2,3,...N$, ישנם סה"כ

$$B(x)=1500+900\frac{x-N}{N-1}$$

בלוקים. לכן, הזמן שיקח לקרוא את בלוק יחיד ממסילה זו הוא

$$T(x)=\frac{1}{7200}\frac{1}{1500+900\frac{x-N}{N-1}}[\min]=\frac{60}{7200}\frac{1}{1500+900\frac{x-N}{N-1}}[\sec]$$

מספר הבלוקים הכולל:

$$TB=\sum_{x=1}^{N}1500+900\frac{x-N}{N-1}=1500N+\frac{900}{N-1}\left(\left(\sum_{x=1}^{N}x\right)-N^2\right)$$

$$=1500N+\frac{900}{N-1}\left(\left(\sum_{x=1}^{N}x\right)-N^2\right)=1500N+\frac{900}{N-1}\left(\frac{N(N+1)}{2}-N^2\right)$$

$$=1500N+\frac{900}{N-1}\frac{N-N^2}{2}=1500N-450N=1050N$$

כעת, X מ"א כאשר ההסברות לבחור את מסילה x פרופורציונית לכמות הבלוקים $B(X=x)$ שבמסילה זו, ולכן

$$P(X=x)=\frac{B(X=x)}{TB}=\frac{1500+900\frac{x-N}{N-1}}{1050N}$$

ולכן

$$ET=\sum_{x=1}^{N}T(x)P(X=x)=\sum_{x=1}^{N}\frac{60}{7200}\frac{1}{1500+900\frac{x-N}{N-1}}\cdot\frac{1500+900\frac{x-N}{N-1}}{1050N}$$

$$=\frac{60}{7200}\frac{1}{1050N}N=\frac{60}{7200}\frac{1}{1050}=7.9\mu[\sec]$$

סעיף ב

אם נבחר ראשית מסילה באקראי, בממוצע נבחר את מסילה $\frac{N}{2}$, ולכן זמן הקריאה הממוצע יהיה זמן הקריאה של בלוק ממסילה זו. במסילה זו 1,050 בלוקים. הדיסק הקשיח יעבור על 1,050 בלוקים אלו במשך $\frac{60}{7,200}[\sec]$, ולכן זמן הקריאה הממוצע של בלוק יחיד במסלול יהיה

$$ET=\frac{1}{1,050}\frac{60}{7,200}[\sec]=7.9\mu[\sec]$$

סעיף ג

בשתי שיטות החישוב הגענו לאותה תוצאה. זאת מכיוון שמספר הבלוקים גדל באופן ליניארי ברדיוס הדיסק ($B(x)=1500+900\frac{x-N}{N-1}$ - כאן המסילה מייצגת את הרדיוס) וכמו כן גם מהירות הקריאה של הבלוקים גדלה באופן ליניארי ברדיוס הדיסק(מהירות הקריאה היא $v=\omega\cdot r$). לכן קיבלנו שזמן הקריאה של בלוק אקראי הוא זמן קריאה של בלוק הנמצא ברדיוס אקראי.

את המספר המתאים לדיסק שלנו כבר קיבלנו בשני הסעיפים הקודמים: $7.9\mu[\sec]$. נוסחה כללית:

$$ET=\frac{60}{RPM}\frac{2}{B_{first}+B_{last}}$$

כאשר B_{last} מספר הבלוקים במסילה האחרונה, ו B_{first} מספר הבלוקים במסילה הראשונה.

פתרון לשאלה 4

סעיף א

כאשר $S' = NS_{baseline}$ הוא השטח שיתפוס המעבר המשופר:

- בשיפור ארכיטקטוני, SC, נקבל

$$T_{SC} = \frac{T_{baseline}}{\sqrt{N}}$$

- בשימוש ב CMP, נקבל

$$T_{CMP} = \alpha \frac{T_{baseline}}{\frac{N \cdot S_{baseline}}{S_{baseline}}} + (1-\alpha) T_{baseline} = \left(\frac{\alpha}{N} + 1 - \alpha \right) T_{baseline}$$

- בשימוש ב $hCMP$, נקבל

$$T_{hCMP} = \alpha \frac{T_{baseline}}{\frac{(N-M) S_{baseline}}{S_{baseline}} + 1} + (1-\alpha) \frac{T_{baseline}}{\sqrt{M}} = \left(\frac{\alpha}{N-M+1} + \frac{1-\alpha}{\sqrt{M}} \right) T_{baseline}$$

השיפור הארכיטקטוני משפר את הביצועים הכוללים של המעבד, ואילו השימוש ב CMP משפר רק את הביצועים המקביליים. לכן, ככל ש α, החלק המקבילי של התוכנית, גדל, נעדיף את השימוש ב CMP.
ניתן לחשב את α הקריטי:

$$T_S = T_{CMP} \Rightarrow \frac{T_{baseline}}{\sqrt{N}} = \left(\frac{\alpha_c}{N} + 1 - \alpha_c \right) T_{baseline} \Rightarrow \frac{1}{\sqrt{N}} = \frac{\alpha_c}{N} + 1 - \alpha_c$$

$$\Rightarrow \frac{1}{\sqrt{N}} - 1 = \alpha_c \left(\frac{1}{N} - 1 \right) \Rightarrow \alpha_c = \frac{\sqrt{N}}{1+\sqrt{N}}$$

עבור $\alpha > \alpha_c$ נעדיף את ריבוי המעבדים.

בשימוש ב $hCMP$, נשפר את החלקים הסידרתיים והמקביליים בו זמנית, ע"י שליטה בפרמטר M.

$$T_S = T_{hCMP} \Rightarrow \frac{T_{baseline}}{\sqrt{N}} = \left(\frac{\alpha}{N-M+1} + \frac{1-\alpha}{\sqrt{M}} \right) T_{baseline} \Rightarrow \frac{1}{\sqrt{N}} = \frac{\alpha}{N-M+1} + \frac{1-\alpha}{\sqrt{M}}$$

$$\Rightarrow \frac{1}{\sqrt{N}} = \frac{\alpha}{N-M+1} + \frac{1-\alpha}{\sqrt{M}} \Rightarrow \frac{1}{\sqrt{N}} - \frac{1}{\sqrt{M}} = \alpha \left(\frac{1}{N-M+1} - \frac{1}{\sqrt{M}} \right)$$

$$\Rightarrow \alpha = \frac{\frac{1}{\sqrt{N}} - \frac{1}{\sqrt{M}}}{\frac{1}{N-M+1} - \frac{1}{\sqrt{M}}} = \frac{\sqrt{\frac{M}{N}} - 1}{\frac{\sqrt{M}}{N-M+1} - 1}$$

סעיף ב

שיטת $hCMP$ היא שילוב של שתי השיטות הקודמות, בכך שהיא מצליחה לשפר את שני חלקי התוכנית באופן גמיש, ע"פ החלטה על הפרמטרים N ו M. לכן, שיטה זו תתאים לכל המצבים בהם אנו לא יכולים לדעת בוודאות מה אחוז הפעילות המקבילית במעבד, או כאשר אחוזים אלו ידועים ולא קבועים.

©2006 אבי בנדל www.technion.co.il

גליון 2

פתרון לשאלה 1

סעיף א

מכיוון שגודל ה $Cache$ אינסופי, לאחר n הגישות הראשונות לזיכרון, כל $f(n)$ הבלוקים יהיו ב $Cache$, ולכן ה $Miss\ Rate$ יהיה מורכב רק מחלק ה $Compulsary$. סה"כ פעולות הגישה לזיכרון הם 30% מסך כל הפקודות, כלומר $n = 0.3IC$. ולכן

$$Miss\ Rate = \frac{f(n)}{n} = \frac{f(0.3IC)}{0.3IC}$$

מכיוון שיש שימוש ב $Write\ Allocate$, יכלנו לתייחס רק ל $Compulsory\ Miss$ - יהיה מקרה יחיד של $Write\ Miss$ לכל בלוק. לכן התייחסנו לאוסף הפקודות של גישה לזיכרון בתור גוש פקודות אחד המהווה 30% מכלל הפקודות.

סעיף ב

$$\begin{aligned}\frac{Execution\ Time}{CCT} &= \frac{IC \cdot CPI_{eff} \cdot CCT}{CCT} = IC \cdot CPI_{eff} \\ &= IC \cdot \left(CPI_{Ideal} + \frac{Memory\ Accesses}{IC} \cdot Miss\ Rate \cdot Miss\ Penalty \right) \\ &= IC \cdot \left(1 + 0.3 \cdot \frac{f(0.3IC)}{0.3IC} \cdot 16 \right)\end{aligned}$$

סעיף ג

רוחב פס כתובות של 32 ביט יתן לנו $2^{32} = 4,294,967,296[Bytes]$ או $4GB$ של נפח זכרון (כאשר $1GB = 2^{30}$).
מכיוון שכל בלוק מכיל 8 בתים, נזדקק ל $Offset$ ברוחב 3 ביטים. מכיוון שזיכרון המטמון הוא מסוג $Fully\ Associative$, הרי ששאר 29 הביטים ישמשו כשדה Tag.

$Tag[32:3]$	$Offset[2:0]$	$Data[0:64]$							

פתרון לשאלה 2

סעיף א

בשימוש ב $Write-Through$ אנו נשתמש ב Bus עבור כל כתיבה של מילה לבלוק זה, ולכן נקבל

$$D_{WT} = n$$

$$T_{WT} = n(A+B)$$

כאשר הבלוק יזרק מזכרון המטמון, לא תהיה כלל פעילות על ה Bus עבור הזריקה הזו.

סעיף ב

בשימוש ב $Write-Back$ אנו נשתמש ב Bus פעם אחת, כאשר הבלוק נזרק מזכרון המטמון, ואז נעביר את כל הבלוק על ה Bus, ולכן נקבל

$$D_{WB} = L$$

$$T_{WB} = A + BL$$

סעיף ג

עבור מדד D, נעדיף את שיטת $Write-Through$ כאשר $n < L$, ונעדיף את שיטת $Write-Back$ כאשר $n > L$ (במקרה שבו $n = L$ אין שיטה עדיפה).
עבור מדד T, נבדוק מתי שיטת $Write-Through$ עדיפה:

$$T_{WT} < T_{WB} \ \Rightarrow\ n(A+B) < A + BL \ \Rightarrow\ n < \frac{A+BL}{A+B}$$

כלומר, עבור $n < \frac{A+BL}{A+B}$, נעדיף את שיטת $Write-Through$, אחרת נעדיף את שיטת $Write-Back$.
שיטת $Write-Back$ תהיה עדיפה בחיתוך בין שני המקרים, כלומר כאשר

$$n > \frac{A+BL}{A+B} \cup n > L \Rightarrow n > \max\left\{\frac{A+BL}{A+B}, L\right\}$$

שיטת $Write-Through$ תהיה עדיפה בחיתוך בין שני המקרים האחרים, כלומר כאשר

$$n < \frac{A+BL}{A+B} \cup n < L \Rightarrow n < \min\left\{\frac{A+BL}{A+B}, L\right\}$$

בתחומי הביניים יהיו עדיפויות שונות לפי שני הקריטריונים, כפי שתוארו בתחילת השאלה.

פתרון לשאלה 3

סעיף א

עבור כל ארכיטקטורת $Cache$, נרצה לפחות $L = M \cdot N$ בתים של $Cache$, כדי שכל המטריצה תוכל להכנס לזכרון המטמון. במקרה זה, בכל הארכיטקטורות, נקבל $Hit\ Rate = 1$.

סעיף ב

לפי נתוני השאלה, כל מטריצה A, ורק היא, מאוכסנות בזכרון המטמון לפני תחילת חיבור מטריצה B ל A. נשים לב כי מתבצעות רק קריאות עבור מטריצות A ו B, ורק כתיבות עבור מטריצה C.

ראשית נביט בקריאות.

עבור ארכיטקטורת $Direct-Mapped$:

הקריאה של המילה הראשונה של מטריצה A תהיה פגיעה (Hit), אך מכיוון שהמטריצות מאוכסנות ברצף, קריאה של המילה הראשונה של מטריצה B תגרום בוודאות להחלפת הבלוק הראשון של מטריצה A ב $Cache$ לבלוק הראשון של מטריצה B. כל עוד אנחנו עוברים על הבלוק הראשון של המטריצות, תהיה החלפה של הבלוק המתאים ב $Cache$ בכל קריאה, לכן מכיוון שבכל בלוק 8 מילים, לאחר הפגיעה הראשונה יהיו 15 החטאות ($Misses$) - 7 קריאות של מטריצה A ו 8 קריאות של מטריצה B. דבר זה יחזור על עצמו לכל בלוק בזכרון, ולכן באופן כולל נקבל

$$Hit\ Rate = \frac{1}{16}$$

עבור ארכיטקטורת $2-Way\ Set\ Associative$:

הקריאה של המילה הראשונה של מטריצה A תהיה פגיעה, מכיוון שהמטריצות מאוכסנות ברצף, קריאה של המילה הראשונה של מטריצה B תגרום בוודאות להחלפת הבלוק השני (בגלל שיטת LRU) שב Set הראשון. 7 הקריאות הבאות של מילים ממטריצה A יהיו פגיעות, כי הבלוק המתאים עדיין בזיכרון, וכמו כן גם 7 הקריאות של המילים הבאות של מטריצה B, לכן קיבלנו 15 פגיעות על 16 הקריאות הראשונות. לאחר שעברנו בקריאות על חצי מהמטריצות, אף בלוק של מטריצה A לא נמצא ב $Cache$, מכיוון שהחלפנו את כל הבלוקים השייכים לחצי השני של מטריצה A עם בלוקים ממטריצה B. לכן כעת על כל קריאה של בלוק, ממטריצה A או B, תהיה החטאה אחת, כלומר 2 החטאות על כל 16 מילים נקראות. לסיכום נקבל

$$Hit\ Rate = \frac{1}{2}\frac{15}{16} + \frac{1}{2}\frac{14}{16} = \frac{29}{32}$$

עבור ארכיטקטורת $Fully\ Associative$:

בקריאה מילים מהבלוק הראשון של מטריצה A יהיו רק פגיעות. קריאה של מילה מהבלוק הראשון של מטריצה B יגרום להחלפת הבלוק השני של מטריצה A, בהתאם לשיטת LRU. מכאן והלאה, תהיה החטאה אחת בכל גישה לבלוק חדש של מטריצה A או B. לכן יהיה $Hit\ Rate$ של $\frac{7}{8}$ בתוספת פגיעה אחת לכל שתי המטריצה (זו המילה הראשונה של מטריצה A) גודל כל מטריצה הוא NM ולכן

$$Hit\ Rate = \frac{7}{8} + \frac{1}{2MN}$$

שנית נביט בכתיבות.

מכיוון שאף בלוק של מטריצה C אינו ב $Cache$ מתחילת הריצה, ומכיוון שאנו פועלים בשיטת $Write\ Around$, נקבל רק החטאות לגבי הכתיבות, כלומר $Hit\ Rate_{writing} = 0$.

סעיף ג

בחלק הראשון של התוכנית, נהיה חייבים לספוג S החטאות, מכיוון שה $Cache$ ריק.
אם נקרא את מערך A מהסוף להתחלה, נוכל לנצל את מנגנון ה LRU בכך שבלוקים חדשים שמובאים ל $Cache$ עבור קריאת מערך B יחליפו את האיברים האחרונים של מערך A, ולא את הראשונים (במקרה שהיינו קוראים את A מההתחלה לסוף).
סה"כ מתבצעות $3S$ קריאות מהזיכרון. S קריאות של A בחלק הראשון, ובחלק השני S קריאות ממערך A ו S קריאות ממערך B. בנוסף ל S ההחטאות של החלק הראשון, בחלק השני נספוג עוד S החטאות עבור קריאת מערך B. מכיוון שקראנו בחלק הראשון את מערך A מהסוף להתחלה, אז יהיו לנו $\frac{S}{2}$ פגיעות בקריאת מערך A, ולכן לסיכום:

$$Miss\ Rate = \frac{(S)+\left(S+\frac{1}{2}S\right)}{3S} = \frac{5}{6}$$

והתוכנית, בשפת c:

```
sum = 0;
pi = 1;
for (i=S ; i>0 ; i--) sum += a[i];
for (i=0 ; i<S ; i++) pi *= a[i] + b[i];
```

פתרון לשאלה 4

סעיף א

כדי להביא למינימום את הביטוי

$$Cache\ Price = \left[\left(associativity\right)^2 + 4\left(Cache\ Size\right)\right]\cdot\left(C+5\right)$$

נרצה פשוט להקטין את כל הפרמטרים, ולכן נבחר:

- אסוציאטיביות: 1
- גודל זכרון מטמון: $8\left[Kbyte\right]$
- טכנולוגיה: $WriteThrough$

סעיף ב

חישוב זמן הגישה (כמות $Clock\ Cycles$) בטכנולוגית WT יעשה ע"פ:

$$\begin{aligned} Access\ Time &= 0.9\left[Hit\ Rate\cdot Hit\ Time + Miss\ Rate\cdot\left(Hit\ Time + Miss\ Penalty\right)\right] + 0.1\cdot Hit\ Time \\ &= Hit\ Time + 0.9\left[Miss\ Rate\cdot Miss\ Penalty\right] = 1 + 0.9\cdot 10\cdot Miss\ Rate \end{aligned}$$

חישוב זמן הגישה (כמות $Clock\ Cycles$) בטכנולוגית WB יעשה ע"פ:

$$\begin{aligned} Access\ Time &= Hit\ Rate\cdot Hit\ Time + Miss\ Rate\cdot\left(Hit\ Time + Miss\ Penalty + 0.3\cdot Dirty\ Penalty\right) \\ &= Hit\ Rate\cdot Hit\ Time + Miss\ Rate\cdot\left(Hit\ Time + Miss\ Penalty + 0.3\cdot Miss\ Penalty\right) \\ &= Hit\ Time + 1.3\cdot Miss\ Rate\cdot Miss\ Penalty = 1 + 1.3\cdot 10\cdot Miss\ Rate \end{aligned}$$

מספר	WB/WT	$Cache\ Size$	$Associativity$	$Miss\ Rate$	זמן גישה ($Clocks$)	עלות
1	WT	$8\left[KByte\right]$	$1-Way$	0.087	1.783	165
2	WB				2.131	495
3	WT		$2-Way$	0.069	1.621	180
4	WB				1.897	540
5	WT		$4-Way$	0.065	1.585	240
6	WB				1.845	720
7	WT	$16\left[KByte\right]$	$1-Way$	0.066	1.594	325
8	WB				1.858	975
9	WT		$2-Way$	0.054	1.486	340
10	WB				1.702	1020
11	WT		$4-Way$	0.049	1.441	400
12	WB				1.637	1200

<u>סעיף ג</u>

נחשב את כמות הגישות לזכרון. עבור טכנולוגית $Write-Through$:

$$Memory\ Accesses = 0.1 \cdot IC_{Mem} + 0.9 \cdot IC_{Mem} \cdot Miss\ Rate = IC_{Mem}\left(0.1 + 0.9 \cdot Miss\ Rate\right)$$

עבור טכנולוגית $Write-Back$:

$$Memory\ Accesses = IC_{Mem} \cdot \left[Miss\ Rate\left(1 + Dirty\ Rate\right)\right] = IC_{Mem} \cdot 1.3 \cdot Miss\ Rate$$

מכיוון שקיבלנו נוסחאות זהות לנוסחאות זמן הגישה, שחישבנו בסעיף הקודם, עד כדי פקטור 10, נבחר בקונפיגורצית הזכרון הנמצאת בשורה 11 בטבלה זו.

<u>סעיף ד</u>

מהטבלה שבסעיף ב', אנו רואים כי בהנתן תקציב של $1000\$$, נעדיף את הקונפיגורציה של שורה 11 מבחינת זמן גישה ממוצע.

עבור רוחב סרט מינימלי, צריך מספר מינימלי של גישות לזיכרון. בכזה מקרה, עדיף להשתמש ב $Write-Back$ ואז הבחירה המוצלחת, בהנתן תקציב של $1000\$$, היא מס' 8 בטבלה.

פתרון לשאלה 5

<u>סעיף א</u>

בכל בלוק ב $Cache$ יש $2^5 = 32$ בתים, ולכן גודל ה $Offset$ הוא 5 ביטים.

לכל אחד משני חלקי ה $Cache$ יש 512 בלוקים.

עבור החלק ה $2Way\ Set\ Associative$, לכל Way יש 256 שורות, ולכן מספיקים 8 ביטים כדי לקבוע את ה Way.
כדי לזהות חד-חד-ערכית כתובת בזיכרון בסגמנט A יש צורך ב 14 ביטים, ולכן עבור שדה ה Tag נשאר ביט בודד.
כל שורה ב $Cache$ תראה כך:

$32\ Bytes\ of\ data$	$Offset[4:0]$	$Index[12:5]$	$Tag[13]$	2'b00

עבור חלק ה $Direct-Mapped$, כדי לקבוע מיקומו של בלוק, יש לבחור שורה מתוך $2^9 = 512$ השורות, ולכן יש להקצות שדה $Index$ של 9 ביטים. כדי לזהות חד-חד-ערכית כתובת בזיכרון בסגמנטים C, D יש צורך ב 15 ביטים, ולכן עבור שדה ה Tag נשאר ביט בודד.
כל שורה ב $Cache$ תראה כך:

$32\ Bytes\ of\ data$	$Offset[4:0]$	$Index[13:5]$	$Tag[14]$	1'b1

בסגמנט A יש סיבית Tag יחידה וכדי לממש את מנגנון LRU יש לשמור סיבית נוספת לכל Set. נקבל סה"כ

$$Cache\ Bits_A = 1 \cdot 512 + 1 \cdot 256 = 768[Bit]$$

בסגמנטים C, D יש סיבית Tag יחידה נקבל סה"כ

$$Cache\ Bits_{C,D} = 1 \cdot 512 = 512[Bit]$$

בנוסף, יש להקצות סיבית $Valid$ לכל בלוק, כלומר 1,024 סיביות נוספות.
וסה"כ

$$Cache\ Bits = 768 + 512 + 1,024 = 2,304[Bit]$$

אם ה $Cache$ הוא מסוג $Write-Back$, נצטרך ביט $Dirty$ נוסף לכל בלוק ואז נקבל סה"כ

$$Cache\ Bits = 2,304 + 1,024 = 3,328[Bit]$$

©2006 אבי בנדל www.technion.co.il

סעיף ב

1. עבור סגמנט A, כל 512 הבלוקים שלו יכולים להמצא ב $Cache$, ולכן $Miss\ Rate_{Segment\ A} = 0$.
עבור סגמנט B, אין מה לדבר על פגיעות לפי ההגדרה, ולכן $Miss\ Rate_{Segment\ B} = 1$.
עבור סגמנטים C, D, מכיוון שהקריאה היא פשוט סדרתית, נקבל החטאה כל 32 קריאות, כלומר $Miss\ Rate = \frac{1}{32}$.
2. החישוב:

$$AMAT = \underbrace{\frac{1}{4}\cdot Hit\ Time}_{Segment\ A\ reads} + \underbrace{\frac{1}{4}\left(Hit\ Time + Miss\ Rate \cdot Miss\ Penalty\right)}_{Segment\ C,D\ reads}$$
$$+ \underbrace{\frac{2}{4}\left(Hit\ Time + Miss\ Rate \cdot Miss\ Penalty\right)}_{Segment\ C,D\ reads}$$
$$= \frac{1}{4} + \frac{1}{4}(1+10) + \frac{2}{4}\left(1 + \frac{1}{32}\cdot 10\right) = 3.65[Cycles]$$

סעיף ג

קפיצה ב32 בתים מבטיחה גישה לבלוק חדש בכל צעד.

ג.1

בהתסברות P_n החטיאו בגישה מס' n, ובהסתברות P_n 0.5 הביאו לאחר ההחטה את הבלוק לcache.
ז"א, רק אם לא הביא בפעם שעברה את הבלוק אחרי גישה n, תהיה החטאה בגישה n+1.
לכן:

$$P_{n+1} = 1 - 0.5P_n$$

ג.2

נרשום P_n עבור n-ים ראשונים, ונסיק עבור n כללי.

$$\left.\begin{aligned} P_0 &= p \\ P_1 &= 1 - 0.5P_0 = 1 - 0.5p \\ P_2 &= 1 - 0.5P_1 = 1 - 0.5(1 - 0.5p) = 1 - 0.5 + 0.5^2 p \\ P_3 &= 1 - 0.5P_2 = 1 - 0.5(1 - 0.5 + 0.5^2 p) = 1 - 0.5 + 0.5^2 - 0.5^3 p \end{aligned}\right\} P_n = (-1*0.5)^n p + \sum_{k=0}^{n-1}(-1*0.5)^k$$

ג.3

נמצא מה הגבול של P_n עבור n גדולים:

$$P_n = (-1*0.5)^n p + \sum_{k=0}^{n-1}(-1*0.5)^k$$
$$P_{n\to\infty} = (-1*0.5)^{n\to\infty} p + \sum_{k=0}^{\infty}(-1*0.5)^k = 0 + \frac{1}{1-(-0.5)} = \frac{1}{1.5} = \frac{2}{3}$$

לכן, במצב יציב, **miss rate = 66.66%**

סעיף ד

ד.1.

סגמנט A:
הסתברות ההחטאה שווה ל1 באיטרציה הראשונה, וקטנה פי 2 עבור כל איטרציה נוספת.
באיטרציה הראשונה, בהכרח יש החטאה, באיטרציה שניה – יש הסתברות 0.5 ש2 הבלוקים של הסט התמפו ל ways שונים, והסתברות 0.5 שאחד דרס את השני – לכן בגישה שניה, הסתברות חצי להחטאה. בגישה שלישית, תהיה החטאה, רק אם הייתה החטאה בגישה הקודמת, וגם כן, בהסתברות 0.5 (סה"כ – 0.25), וכן הלאה...

סה"כ - $P_{MissA}(n_{iteration}) = 0.5^{n-1}$

סגמנט B:
בגלל שלא ממופה, תמיד מחטיאים

$$P_{MissB}(n_{iteration}) = 1$$

סגמנטים C וD:
יש תמיד החטאות, כי כל פעם בלוק מC דורס בלוק מD שאמור להיות ממופה לאותו המקום בcache.

$$P_{MissC,D}(n_{iteration}) = 1$$

סה"כ:

$$P_{Miss}(n_{iteration}) = 0.25 * P_{MissA}(n_{iteration}) + 0.25 * P_{MissB}(n_{iteration}) + 0.5 * P_{MissC,D}(n_{iteration}) = 0.25 * 0.5^{n-1} + 0.75$$

ד.2

סגמנט A:
עבור מצב יציב – 0% החטאה.
באיטרציה ראשונה, 100% להחטאה, וגם באיטרציה השניה, כי בסוף האיטרציה הראשונה, ערכי A נדרסו ע"י ערכים מC וD. באיטרציה השלישית ערכי A יכנסו לway השני, ולא ידרסו יותר.

סגמנטים B, C, D:
כמו בסעיף הקודם, 100% החטאה.
לכן, עבור מצב יציב:

$$P_{Miss}(n_{iteration}) = 0.25 * P_{MissA}(n_{iteration}) + 0.25 * P_{MissB}(n_{iteration}) + 0.5 * P_{MissC,D}(n_{iteration}) = 0.75$$

גליון 3

פתרון לשאלה 1

תוצאות הריצות:

- דרגת אסוציאטיביות: 2

```
> dinero -b1 -u24 -a2 -rl <$c/trace4.din


---Dinero III by Mark D. Hill.
---Version 3.3, Released May 1989.

---Execution begins.

CMDLINE: dinero -b1 -u24 -a2 -rl
CACHE (bytes): blocksize=1, sub-blocksize=0, Usize=24, Dsize=0, Isize=0.
POLICIES: assoc=2-way, replacement=l, fetch=d(1,0), write=c, allocate=w.
CTRL: debug=0, output=0, skipcount=0, maxcount=10000000, Q=0.

---Simulation begins.
---Simulation complete.

 Metrics                 Access Type:
 (totals,fraction)       Total   Instrn   Data    Read   Write    Misc
 ------------------      ------  ------  ------  ------  ------  ------
 Demand Fetches            5000    5000       0       0       0       0
                          1.0000  1.0000  0.0000  0.0000  0.0000  0.0000

 Demand Misses              622     622       0       0       0       0
                          0.1244  0.1244  0.0000  0.0000  0.0000  0.0000

 Words From Memory          155
 ( / Demand Fetches)      0.0310
 Words Copied-Back            0
 ( / Demand Writes)       0.0000
 Total Traffic (words)      155
 ( / Demand Fetches)      0.0310

---Execution complete.
> _
```

- דרגת אסוציאטיביות: 3

```
> dinero -b1 -u24 -a3 -rl < $c/trace4.din


---Dinero III by Mark D. Hill.
---Version 3.3, Released May 1989.

---Execution begins.

CMDLINE: dinero -b1 -u24 -a3 -rl
CACHE (bytes): blocksize=1, sub-blocksize=0, Usize=24, Dsize=0, Isize=0.
POLICIES: assoc=3-way, replacement=l, fetch=d(1,0), write=c, allocate=w.
CTRL: debug=0, output=0, skipcount=0, maxcount=10000000, Q=0.

---Simulation begins.
---Simulation complete.

 Metrics                 Access Type:
 (totals,fraction)       Total   Instrn   Data    Read   Write    Misc
 ------------------      ------  ------  ------  ------  ------  ------
 Demand Fetches            5000    5000       0       0       0       0
                          1.0000  1.0000  0.0000  0.0000  0.0000  0.0000

 Demand Misses              821     821       0       0       0       0
                          0.1642  0.1642  0.0000  0.0000  0.0000  0.0000

 Words From Memory          205
 ( / Demand Fetches)      0.0410
 Words Copied-Back            0
 ( / Demand Writes)       0.0000
 Total Traffic (words)      205
 ( / Demand Fetches)      0.0410

---Execution complete.
>
```

- דרגת אסוציאטיביות : 4

```
> dinero -b1 -u24 -a4 -rl < $c/trace4.din


---Dinero III by Mark D. Hill.
---Version 3.3, Released May 1989.

---Execution begins.

CMDLINE: dinero -b1 -u24 -a4 -rl
CACHE (bytes): blocksize=1, sub-blocksize=0, Usize=24, Dsize=0, Isize=0.
POLICIES: assoc=4-way, replacement=l, fetch=d(1,0), write=c, allocate=w.
CTRL: debug=0, output=0, skipcount=0, maxcount=10000000, Q=0.

---Simulation begins.
---Simulation complete.

 Metrics                 Access Type:
 (totals,fraction)       Total    Instrn    Data     Read    Write     Misc
 -----------------       ------   ------   ------   ------   ------   ------
 Demand Fetches            5000     5000        0        0        0        0
                         1.0000   1.0000   0.0000   0.0000   0.0000   0.0000

 Demand Misses             1020     1020        0        0        0        0
                         0.2040   0.2040   0.0000   0.0000   0.0000   0.0000

 Words From Memory          255
 ( / Demand Fetches)     0.0510
 Words Copied-Back            0
 ( / Demand Writes)      0.0000
 Total Traffic (words)      255
 ( / Demand Fetches)     0.0510

---Execution complete.
```

- דרגת אסוציאטיביות : 6

```
> dinero -b1 -u24 -a6 -rl < $c/trace4.din


---Dinero III by Mark D. Hill.
---Version 3.3, Released May 1989.

---Execution begins.

CMDLINE: dinero -b1 -u24 -a6 -rl
CACHE (bytes): blocksize=1, sub-blocksize=0, Usize=24, Dsize=0, Isize=0.
POLICIES: assoc=6-way, replacement=l, fetch=d(1,0), write=c, allocate=w.
CTRL: debug=0, output=0, skipcount=0, maxcount=10000000, Q=0.

---Simulation begins.
---Simulation complete.

 Metrics                 Access Type:
 (totals,fraction)       Total    Instrn    Data     Read    Write     Misc
 -----------------       ------   ------   ------   ------   ------   ------
 Demand Fetches            5000     5000        0        0        0        0
                         1.0000   1.0000   0.0000   0.0000   0.0000   0.0000

 Demand Misses             1418     1418        0        0        0        0
                         0.2836   0.2836   0.0000   0.0000   0.0000   0.0000

 Words From Memory          354
 ( / Demand Fetches)     0.0708
 Words Copied-Back            0
 ( / Demand Writes)      0.0000
 Total Traffic (words)      354
 ( / Demand Fetches)     0.0708

---Execution complete.
>
```

- דרגת אסוציאטיביות : 8

```
> dinero -b1 -u24 -a8 -rl < $c/trace4.din


---Dinero III by Mark D. Hill.
---Version 3.3, Released May 1989.

---Execution begins.

CMDLINE: dinero -b1 -u24 -a8 -rl
CACHE (bytes): blocksize=1, sub-blocksize=0, Usize=24, Dsize=0, Isize=0.
POLICIES: assoc=8-way, replacement=l, fetch=d(1,0), write=c, allocate=w.
CTRL: debug=0, output=0, skipcount=0, maxcount=10000000, Q=0.

---Simulation begins.
---Simulation complete.

 Metrics                 Access Type:
 (totals,fraction)       Total   Instrn   Data    Read   Write    Misc
 -----------------       ------  ------  ------  ------  ------  ------
 Demand Fetches            5000    5000       0       0       0       0
                          1.0000  1.0000  0.0000  0.0000  0.0000  0.0000

 Demand Misses             1816    1816       0       0       0       0
                          0.3632  0.3632  0.0000  0.0000  0.0000  0.0000

 Words From Memory          454
 ( / Demand Fetches)     0.0908
 Words Copied-Back            0
 ( / Demand Writes)      0.0000
 Total Traffic (words)      454
 ( / Demand Fetches)     0.0908

---Execution complete.
>
```

- דרגת אסוציאטיביות : 12

```
> dinero -b1 -u24 -a12 -rl < $c/trace4.din


---Dinero III by Mark D. Hill.
---Version 3.3, Released May 1989.

---Execution begins.

CMDLINE: dinero -b1 -u24 -a12 -rl
CACHE (bytes): blocksize=1, sub-blocksize=0, Usize=24, Dsize=0, Isize=0.
POLICIES: assoc=12-way, replacement=l, fetch=d(1,0), write=c, allocate=w.
CTRL: debug=0, output=0, skipcount=0, maxcount=10000000, Q=0.

---Simulation begins.
---Simulation complete.

 Metrics                 Access Type:
 (totals,fraction)       Total   Instrn   Data    Read   Write    Misc
 -----------------       ------  ------  ------  ------  ------  ------
 Demand Fetches            5000    5000       0       0       0       0
                          1.0000  1.0000  0.0000  0.0000  0.0000  0.0000

 Demand Misses             2612    2612       0       0       0       0
                          0.5224  0.5224  0.0000  0.0000  0.0000  0.0000

 Words From Memory          653
 ( / Demand Fetches)     0.1306
 Words Copied-Back            0
 ( / Demand Writes)      0.0000
 Total Traffic (words)      653
 ( / Demand Fetches)     0.1306

---Execution complete.
>
```

אנו רואים שדרגת האסוציאטיביות מגדילה את קצב ההחטאות.סיכום הסימולציות :

אסוציאטיביות	2	3	4	6	8	12
Miss Rate	0.1244	0.1642	0.2040	0.2836	0.3632	0.5224

בהסתכלות על trace4.din, גילינו שסימלצנו על 5000 פקודות שניגשות באופן מחזורי ל 0x19 כתובות עוקבות (25).
באופן אנליטי, כאשר נניח מצב יציב, נקבל

$$MR = \frac{1+n}{25}$$

כאשר n דרגת האסוציאטיביות.
וכאשר נוסיף את ההחטאות המתחייבות (*Compulsory Misses*) נקבל

$$MR = \frac{1+n}{25} + \frac{24}{5000}$$

ולכן באופן אנליטי, קיבלנו:

אסוציאטיביות	2	3	4	6	8	12
Miss Rate	0.1248	0.1648	0.2048	0.2848	0.3648	0.5248

ההבדלים הקטנים נובעים מאי-אינסופיות של התוכנית.

פתרון לשאלה 2

קובץ trace2.din, מדיניות החלפה LRU

אנו פונים באופן מחזורי לשלוש פקודות רציפות.

באופן אנליטי, קל לראות ש $MR = 1$ במצב יציב.

```
> dinero -b1 -u2 -a2 -rl < $c/trace2.din


---Dinero III by Mark D. Hill.
---Version 3.3, Released May 1989.

---Execution begins.

CMDLINE: dinero -b1 -u2 -a2 -rl
CACHE (bytes): blocksize=1, sub-blocksize=0, Usize=2, Dsize=0, Isize=0.
POLICIES: assoc=2-way, replacement=l, fetch=d(1,0), write=c, allocate=w.
CTRL: debug=0, output=0, skipcount=0, maxcount=10000000, Q=0.

---Simulation begins.
---Simulation complete.

 Metrics                Access Type:
 (totals,fraction)      Total   Instrn   Data    Read   Write    Misc
 -----------------      ------  ------  ------  ------  ------  ------
 Demand Fetches            900     900       0       0       0       0
                        1.0000  1.0000  0.0000  0.0000  0.0000  0.0000

 Demand Misses             900     900       0       0       0       0
                        1.0000  1.0000  0.0000  0.0000  0.0000  0.0000

 Words From Memory         225
 ( / Demand Fetches)    0.2500
 Words Copied-Back           0
 ( / Demand Writes)     0.0000
 Total Traffic (words)     225
 ( / Demand Fetches)    0.2500

---Execution complete.
```

קובץ trace5.din, מדיניות החלפה LRU

אנו פונים באופן אקראי לשלוש פקודות רציפות.

במצב יציב, נאמר שלכל פקודה אקראית יש סיכוי של $\frac{2}{3}$ להמצא ב $Cache$. לכן, בסיכוי $\frac{1}{3}$ יש החטאה, כלומר $MR = 0.333$.

```
> dinero -b1 -u2 -a2 -rl < $c/trace5.din


---Dinero III by Mark D. Hill.
---Version 3.3, Released May 1989.

---Execution begins.

CMDLINE: dinero -b1 -u2 -a2 -rl
CACHE (bytes): blocksize=1, sub-blocksize=0, Usize=2, Dsize=0, Isize=0.
POLICIES: assoc=2-way, replacement=l, fetch=d(1,0), write=c, allocate=w.
CTRL: debug=0, output=0, skipcount=0, maxcount=10000000, Q=0.

---Simulation begins.
---Simulation complete.

 Metrics                Access Type:
 (totals,fraction)      Total    Instrn    Data     Read    Write     Misc
 -----------------      ------   ------   ------   ------   ------   ------
 Demand Fetches            900      900        0        0        0        0
                        1.0000   1.0000   0.0000   0.0000   0.0000   0.0000

 Demand Misses             282      282        0        0        0        0
                        0.3133   0.3133   0.0000   0.0000   0.0000   0.0000

 Words From Memory          70
 ( / Demand Fetches)    0.0778
 Words Copied-Back           0
 ( / Demand Writes)     0.0000
 Total Traffic (words)      70
 ( / Demand Fetches)    0.0778

---Execution complete.
```

קובץ trace2.din, מדיניות החלפה Random

אנו פונים באופן מחזורי לשלוש פקודות רציפות.

תמיד ישנן שתי פקודות ב $Cache$. בין קריאה של בלוק מסויים לקריאתו הבאה ישנן שתי קריאות של בלוקים אחרים. כדי לחשב את ה $Hit\ Rate$, נרצה שבשתי קריאות אלו אותו בלוק ישאר ב $Cache$. לכן, חייבת להיות החטאה אחת בשתי הקריאות הללו. נרצה שהבלוק שיוחלף בהחטאה זו לא יהיה הבלוק שלנו, וזה יקרה בהסתברות $\frac{1}{2}$. לכן קיבלנו

$$HR = \frac{MR}{2}, \qquad HR + MR = 1 \ \Rightarrow \ MR = \frac{2}{3}$$

```
> dinero -b1 -u2 -a2 -rr < $c/trace2.din


---Dinero III by Mark D. Hill.
---Version 3.3, Released May 1989.

---Execution begins.

CMDLINE: dinero -b1 -u2 -a2 -rr
CACHE (bytes): blocksize=1, sub-blocksize=0, Usize=2, Dsize=0, Isize=0.
POLICIES: assoc=2-way, replacement=r, fetch=d(1,0), write=c, allocate=w.
CTRL: debug=0, output=0, skipcount=0, maxcount=10000000, Q=0.

---Simulation begins.
---Simulation complete.

 Metrics                Access Type:
 (totals,fraction)      Total    Instrn    Data     Read    Write     Misc
 -----------------      ------   ------   ------   ------   ------   ------
 Demand Fetches            900      900        0        0        0        0
                        1.0000   1.0000   0.0000   0.0000   0.0000   0.0000

 Demand Misses             601      601        0        0        0        0
                        0.6678   0.6678   0.0000   0.0000   0.0000   0.0000

 Words From Memory         150
 ( / Demand Fetches)    0.1667
 Words Copied-Back           0
 ( / Demand Writes)     0.0000
 Total Traffic (words)     150
 ( / Demand Fetches)    0.1667

---Execution complete.
```

קובץ trace5.din, מדיניות החלפה Random

אנו פונים באופן אקראי לשלוש פקודות אקראיות.

מכיוון שהפניות לכל פקודת הן באופן אקראי, אזי ההסתברות להחטאה היא ההסתברות שהבלוק לא נמצא בזיכרון ה *Cache*, כלומר

$$MR = \frac{1}{3}$$

```
> dinero -b1 -u2 -a2 -rr < $c/trace5.din


---Dinero III by Mark D. Hill.
---Version 3.3, Released May 1989.

---Execution begins.

CMDLINE: dinero -b1 -u2 -a2 -rr
CACHE (bytes): blocksize=1, sub-blocksize=0, Usize=2, Dsize=0, Isize=0.
POLICIES: assoc=2-way, replacement=r, fetch=d(1,0), write=c, allocate=w.
CTRL: debug=0, output=0, skipcount=0, maxcount=10000000, Q=0.

---Simulation begins.
---Simulation complete.

 Metrics                 Access Type:
 (totals,fraction)       Total   Instrn   Data    Read   Write    Misc
 -----------------       ------  ------  ------  ------  ------  ------
 Demand Fetches             900     900       0       0       0       0
                         1.0000  1.0000  0.0000  0.0000  0.0000  0.0000

 Demand Misses              305     305       0       0       0       0
                         0.3389  0.3389  0.0000  0.0000  0.0000  0.0000

 Words From Memory           76
 ( / Demand Fetches)     0.0844
 Words Copied-Back            0
 ( / Demand Writes)      0.0000
 Total Traffic (words)       76
 ( / Demand Fetches)     0.0844

---Execution complete.
```

פתרון לשאלה 3

סעיף א

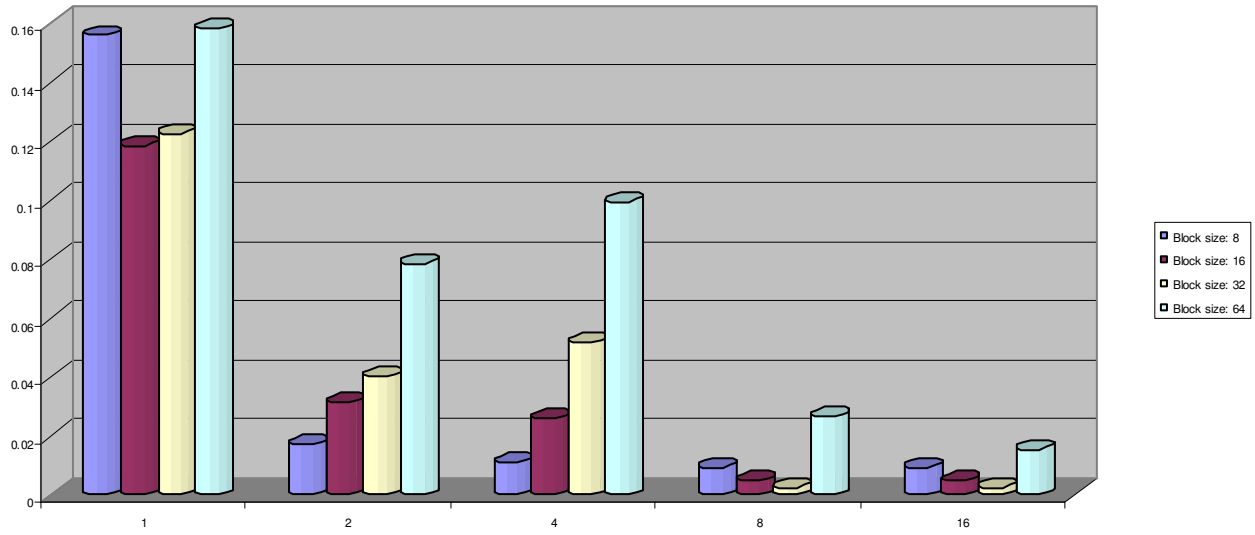
התוצאות בטבלה הבאה:

Unified Cache					
Block Size	8				
Assoc Level	1	2	4	8	16
Miss Rate	0.1559	0.0171	0.011	0.0094	0.0094
Block Size	16				
Assoc Level	1	2	4	8	16
Miss Rate	0.1185	0.0315	0.0259	0.0048	0.0048
Block Size	32				
Assoc Level	1	2	4	8	16
Miss Rate	0.1225	0.0401	0.0517	0.0026	0.0025
Block Size	64				
Assoc Level	1	2	4	8	16
Miss Rate	0.1581	0.0785	0.0996	0.0266	0.0153

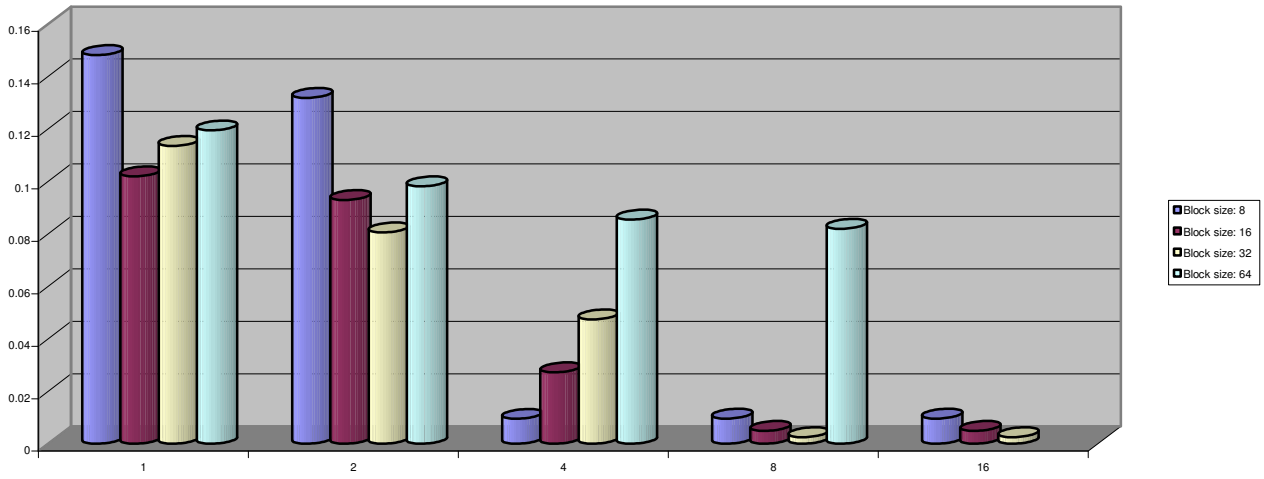
Non Unified Cache					
Block Size	8				
Assoc Level	1	2	4	8	16
Miss Rate	0.1481	0.1318	0.0095	0.0096	0.0096
Block Size	16				
Assoc Level	1	2	4	8	16
Miss Rate	0.1019	0.0929	0.0272	0.0049	0.0049
Block Size	32				
Assoc Level	1	2	4	8	16
Miss Rate	0.1135	0.0805	0.0474	0.0025	0.0025
Block Size	64				
Assoc Level	1	2	4	8	16
Miss Rate	0.1194	0.0981	0.0854	0.0819	

קצב ההחטאה האופטימלי המתקבל הוא 0.25% והוא מתקבל במספר קונפיגורציות, כמודגש בטבלה.

עבור זכרון משולב של 1024 מילים, להלן קצב ההחטאה כפונקציה של דרגת האסוציאטיביות, לגדלי בלוק שונים:



עבור זכרון נפרד של 512 בתים עבור נתונים ו 512 בתים עבור פקודות, להלן קצב ההחטאה כפונקציה של דרגת האסוציאטיביות, לגדלי בלוק שונים:



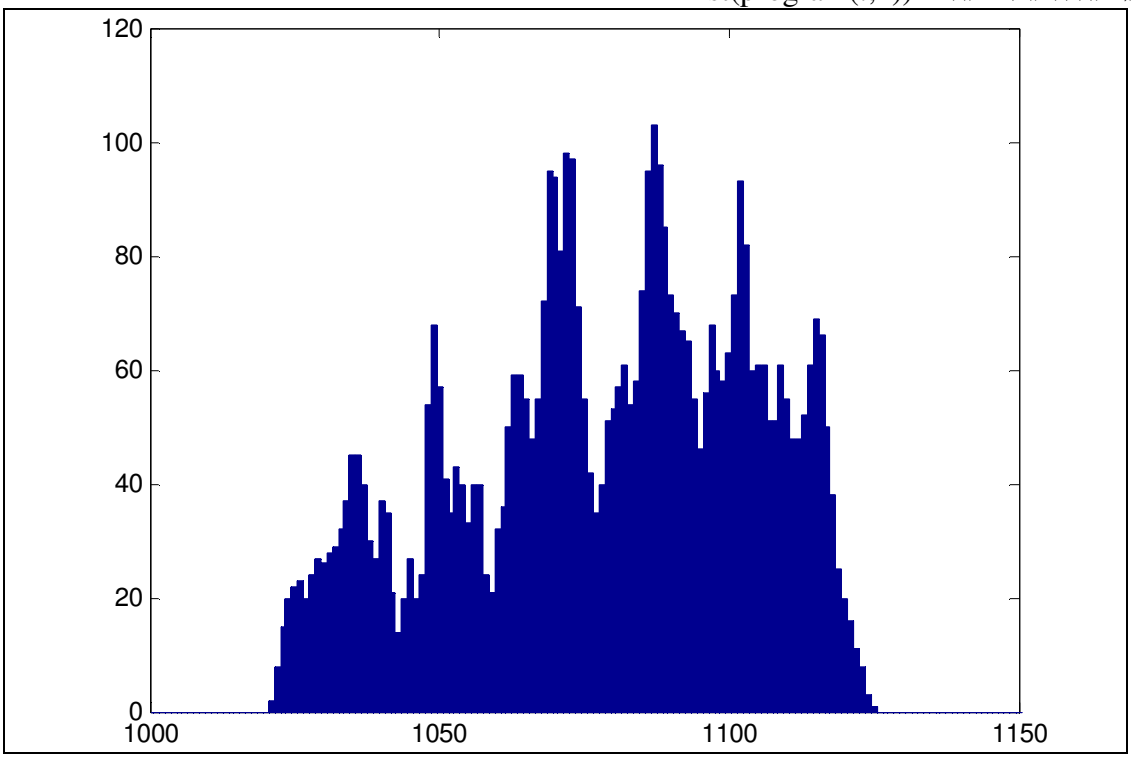
תוצאות הסימוליה מראות כי הגדלת האסוציאטיביות משפרת את ביצועי ה *Cache* , חוץ מאשר במקרה של זיכרון משולב עם גודל בלוק של 64 בתים, וזו תוצאה ישירה של עקרון הלוקליות בזכרון הנתונים.

סעיף ב

1. הילוך שיכור

```
size_of_program  = 5000;

% first command:
program(1,1)      = 2;
program(1,2)      = 2^10;

% paramaters
d                 = 1;

for index = 2:size_of_program
   a = rand;
   program (index,1) = 2;
   if (a>0.5)
      program (index,2) = program (index-1,2) + d;
   else
      program (index,2) = program (index-1,2) - d;
   end
end
```

ההיסטוגרמה המתאימה – hist(program(:,2)):

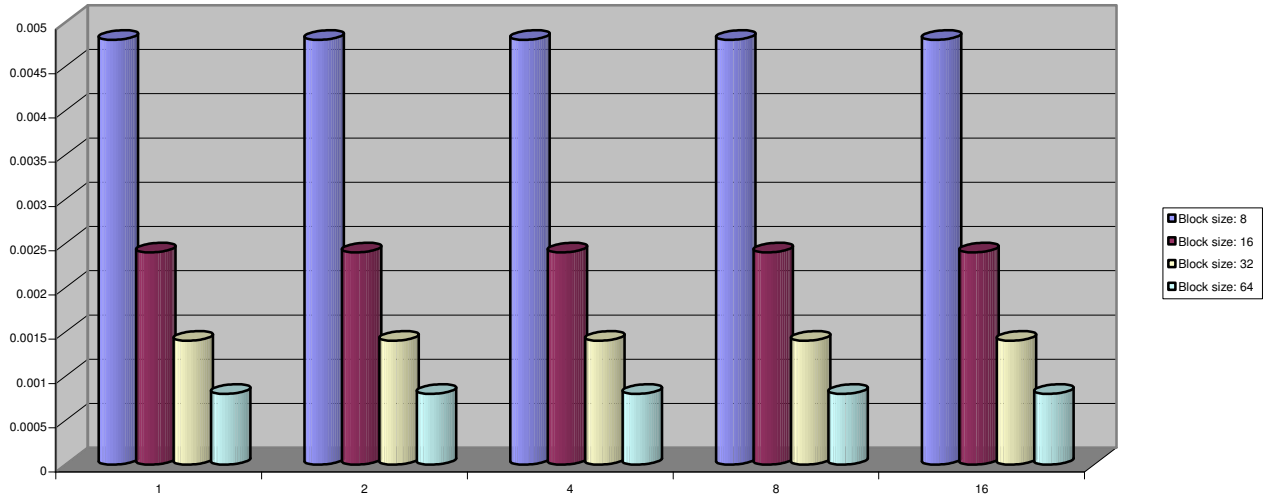


ריכוז תוצאות הריצות בטבלה:

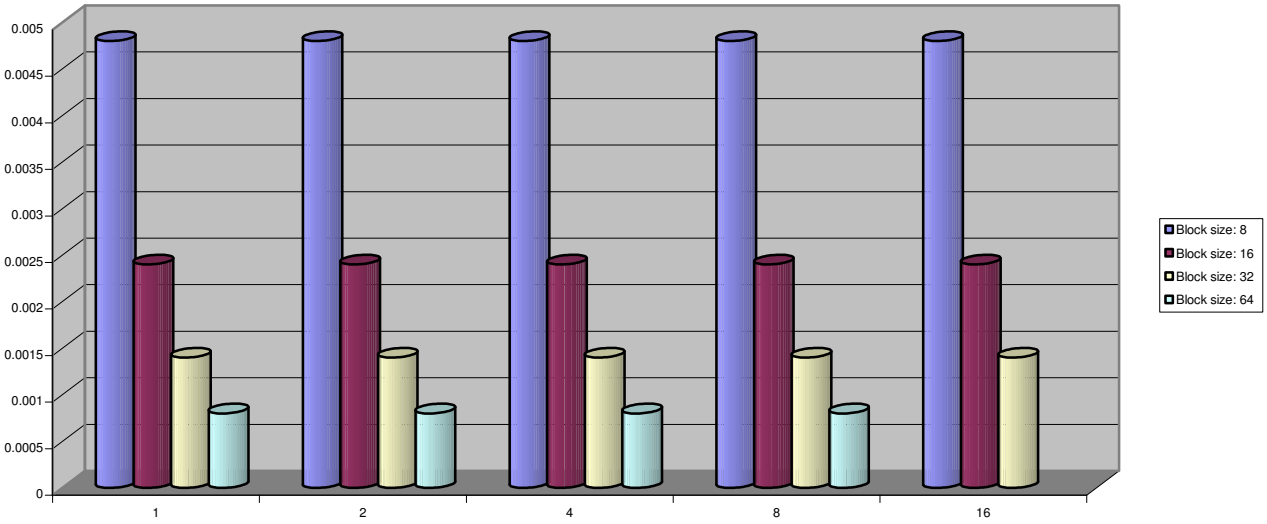
Unified Cache					
Block Size	8				
Assoc Level	1	2	4	8	16
Miss Rate	0.0048	0.0048	0.0048	0.0048	0.0048
Block Size	16				
Assoc Level	1	2	4	8	16
Miss Rate	0.0024	0.0024	0.0024	0.0024	0.0024
Block Size	32				
Assoc Level	1	2	4	8	16
Miss Rate	0.0014	0.0014	0.0014	0.0014	0.0014
Block Size	64				
Assoc Level	1	2	4	8	16
Miss Rate	0.0008	0.0008	0.0008	0.0008	0.0008

Non Unified Cache					
Block Size	8				
Assoc Level	1	2	4	8	16
Miss Rate	0.0048	0.0048	0.0048	0.0048	0.0048
Block Size	16				
Assoc Level	1	2	4	8	16
Miss Rate	0.0024	0.0024	0.0024	0.0024	0.0024
Block Size	32				
Assoc Level	1	2	4	8	16
Miss Rate	0.0014	0.0014	0.0014	0.0014	0.0014
Block Size	64				
Assoc Level	1	2	4	8	16
Miss Rate	0.0008	0.0008	0.0008	0.0008	

ובגרפים:
זכרון משולב בגודל 1024 בתים, להלן קצב ההחטאה כפונקציה של דרגת האסוציאטיביות, לגדלי בלוק שונים:



עבור זכרון נפרד של 512 בתים עבור נתונים ו 512 בתים עבור פקודות, להלן קצב ההחטאה כפונקציה של דרגת האסוציאטיביות, לגדלי בלוק שונים:



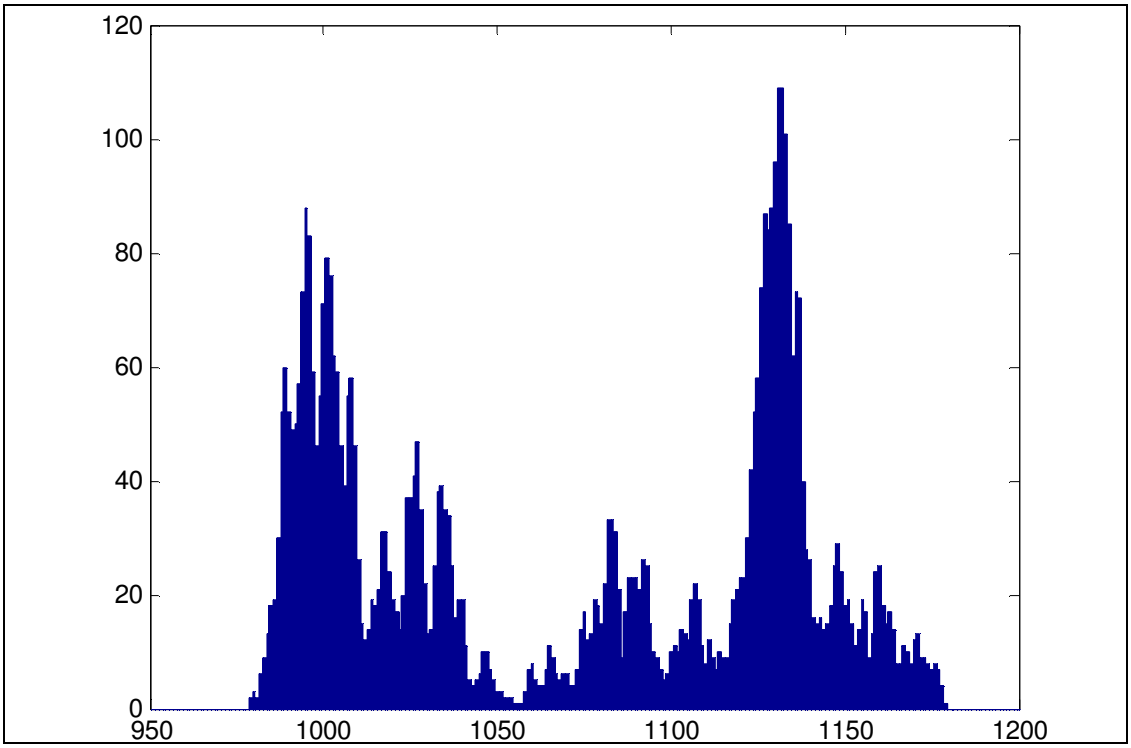
אנו רואים בבירור שאין משמעות להפרדת הזיכרון לפקודות ונתונים, מכיוון שטיפלנו רק בסוג אחד של פניות, וגודל של 512 בתים לא היה קטן דיו כדי להשפיע על ביצועי ה *Cache*.
עוד אנו רואים בבירור כי קצב ההחטאה כלל אינו פונקציה של דרגת האסוציטיביות, וזאת עקב הלוקאליות הרבה שבתוכניתנו.

2. הילוך שיכור עם קפיצות

```
size_of_program  = 5000;

% first command:
program(1,1)      = 2;
program(1,2)      = 2^20;

% paramaters
d                 = 1;
L                 = 200;
p                 = 0.01;

for index = 2:size_of_program
   a = rand;
   program (index,1) = 2;
   if (a>0.5)
      if (a>1-p)
         program (index,2) = program (index-1,2) + L;
      else
         program (index,2) = program (index-1,2) + d;
      end
   else
      if (a<p)
```

```
        program (index,2) = program (index-1,2) - L;
      else
        program (index,2) = program (index-1,2) - d;
      end
    end
end
```

ההיסטוגרמה המתאימה – hist(program(:,2)):



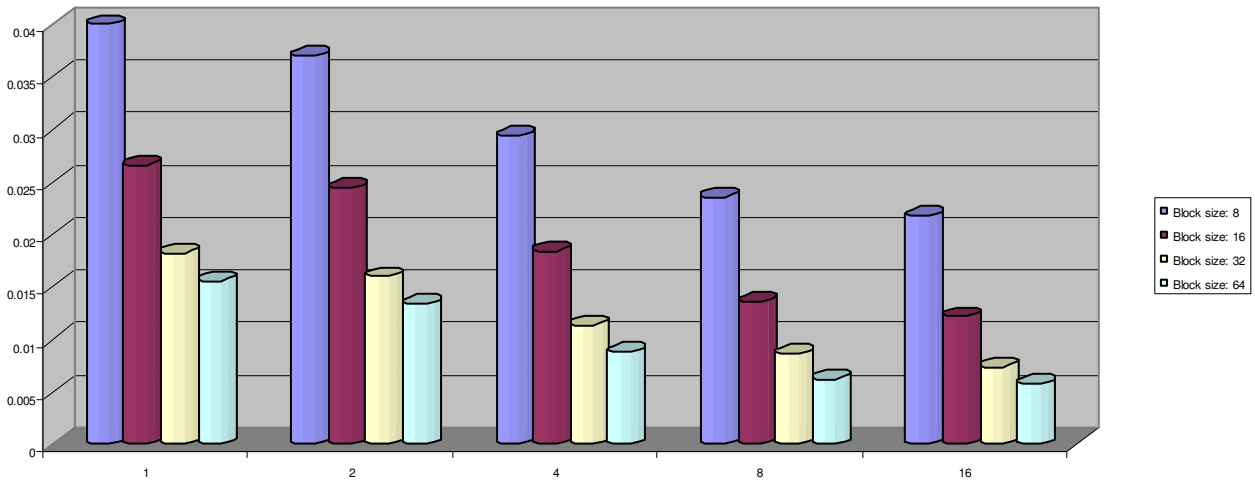
ריכוז תוצאות הריצות בטבלה:

Unified Cache					
Block Size	8				
Assoc Level	1	2	4	8	16
Miss Rate	0.04	0.037	0.0294	0.0236	0.0218
Block Size	16				
Assoc Level	1	2	4	8	16
Miss Rate	0.0266	0.0244	0.0184	0.0136	0.0122
Block Size	32				
Assoc Level	1	2	4	8	16
Miss Rate	0.0182	0.016	0.0114	0.0086	0.0074
Block Size	64				
Assoc Level	1	2	4	8	16
Miss Rate	0.0156	0.0134	0.0088	0.0062	0.0058

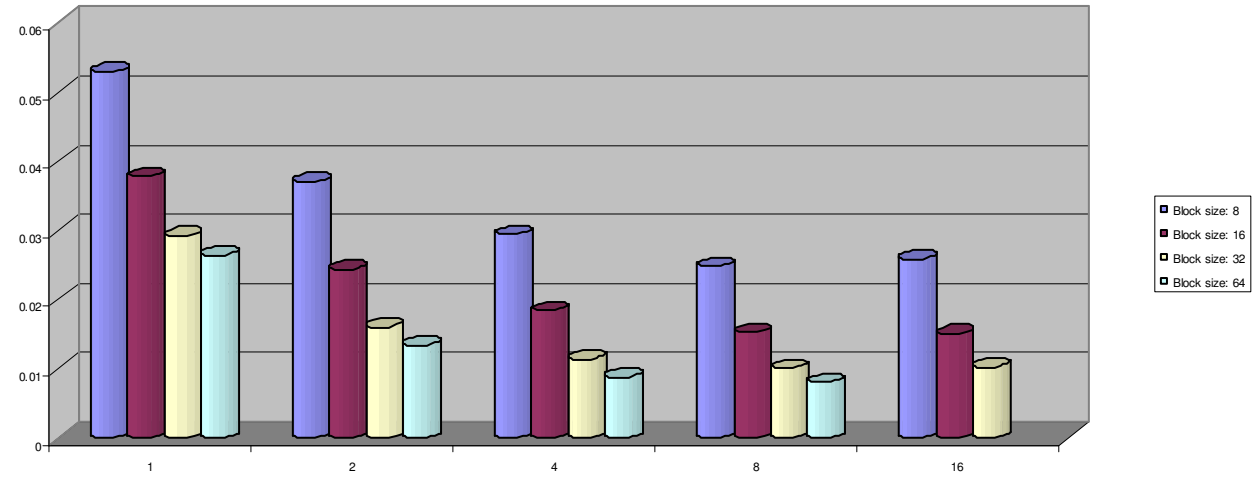
Non Unified Cache					
Block Size	8				
Assoc Level	1	2	4	8	16
Miss Rate	0.053	0.037	0.0294	0.0248	0.0258
Block Size	16				
Assoc Level	1	2	4	8	16
Miss Rate	0.0378	0.0244	0.0184	0.0152	0.015
Block Size	32				
Assoc Level	1	2	4	8	16
Miss Rate	0.0292	0.016	0.0114	0.01	0.0102
Block Size	64				
Assoc Level	1	2	4	8	16
Miss Rate	0.0264	0.0134	0.0088	0.008	

ובגרפים:

עבור זכרון משולב בגודל 1024 בתים, להלן קצב ההחטאה כפונקציה של דרגת האסוציאטיביות, לגדלי בלוק שונים:



עבור זכרון נפרד של 512 בתים עבור נתונים ו 512 בתים עבור פקודות, להלן קצב ההחטאה כפונקציה של דרגת האסוציאטיביות, לגדלי בלוק שונים:



בשונה מהילוך שיכור רגיל, כאן יש משמעות לדרגת האסוציטיביות, וזאת מכיוון שאין כאן לוקליות הדוקה כמו בהילוך שיכור – אלא ישנן הקפיצות בגודל L, המאלצות את לעבוד עם בלוקים מרוחקים זה מזה. לכן הגדלת האסוציאטיביות תעזור לביצועים במקרה זה.

שלא כמו בהילוך שיכור רגיל, יש משמעות להפרדת הזיכרון לפקודות ונתונים: אנו מבחינים כי עבור בלוקים בגודל 8 בתים השיפור בביצועים, ע"י הגדלת דרגת האסוציאטיביות, אינו קורה ואף ישנה החרפה בביצועים. תופעה זאת קורית בדרגות אסוציאטיביות גדולות, מכיוון שאלה לא מאפשרות טיפול טוב במסגרת עקרון הלוקליות.

גליון 4

פתרון לשאלה 1

סעיף א

המעבד נדרש למנוע מתהליך שימוש בבלוקים שנשארו מתהליך קודם משום שהcache מאורגן לפי הכתובות הוירטואליות, שיכולות לחפוף בין 2 תהליכים שונים לחלוטין. לכן, כדי שתהליך אחד לא יבצע בטעות קוד של תהליך אחר, הוא לא אמור לגשת לבלוקים שבהם נשמר מידע מאותו התהליך.

סעיף ב

שיטה 1:

$$\underbrace{4}_{invalidation} + \underbrace{0.5F}_{\text{for each dirty block}} \left(\underbrace{1}_{\text{bus arbitration}} + \underbrace{0.4*10}_{\text{bus isn't immediately free}} + \underbrace{8}_{\text{write back}} \right) = 4 + 6.5F$$

שיטה 2:

$$\underbrace{4}_{invalidation} + \underbrace{\left(\underbrace{1}_{\text{bus arbitration}} + \underbrace{0.4*40}_{\text{bus isn't immediately free}} \right)}_{\text{first access to bus only}} + \underbrace{0.5F*8}_{\text{for each dirty block}} = 21 + 4F$$

שיטה 3:

$$\underbrace{0}_{\text{no invalidation}} + \underbrace{(2F - C)}_{\text{for each removed block}} * \underbrace{0.5}_{\text{dirty \%}} * \left(\underbrace{1}_{\text{bus arbitration}} + \underbrace{0.4*10}_{\text{bus isn't immediately free}} + \underbrace{8}_{\text{write back}} \right) = 6.5(2F - C)$$

סעיף ג

הבעיה העיקרית של burst mode, היא שהיתרון שלו הוא בכתיבה רציפה כאשר הקוד נמצא בצורה רציפה בזיכרון.
הבעיה היא, שהcache ממופה לפי כתובות וירטואליות, ואנחנו לא יכולים לדעת מראש האם הקוד נמצא בכתובות פיזיות צמודות (סביר להניח שלא).
גם במקרה של מעבר לcache הממופה לכתובות פיזיות, עדיין אי אפשר להבטיח שבלוקים יהיו בכתובות עוקבות, אלא אם כן מערכת ההפעלה טטען את הקוד מהדיסק הקשיח בצורה רציפה עד כמה שניתן.

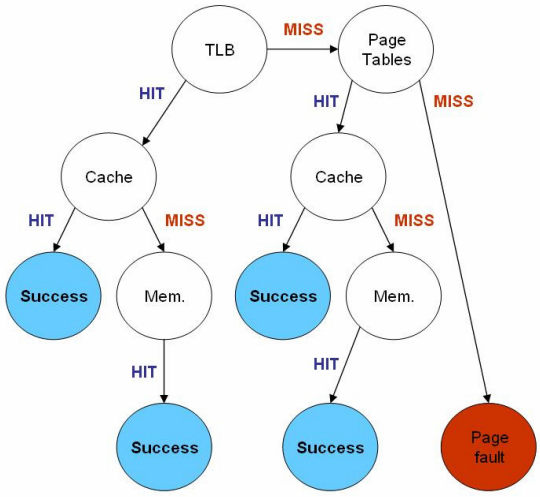
פתרון לשאלה 2

סעיף א

הסבר	TLB miss	Cache miss	Page Fault
קיים – המקרה הרצוי – אין החטאות כלל.	-	-	-
לא קיים – אם לא הייתה החטאה בcache, סימן שהמידע בזכרון הראשי. כנ"ל לגבי הTLB.	-	-	+
קיים. המידע בזיכרון, אך לא ממופה בcache.	-	+	-
לא קיים - אם לא הייתה החטאה בTLB, סימן שהמידע בזכרון הראשי.	-	+	+
קיים. המידע בזיכרון, אך לא ממופה בTLB.	+	-	-
לא קיים – אם לא הייתה החטאה בcache, סימן שהמידע בזכרון הראשי. כנ"ל לגבי הTLB.	+	-	+
קיים – הבלוק לא ממופה לא בcache ולא בTLB. (נמצא בזכרון הראשי)	+	+	-
קיים – הבלוק נמצא בדיסק הקשיח.	+	+	+

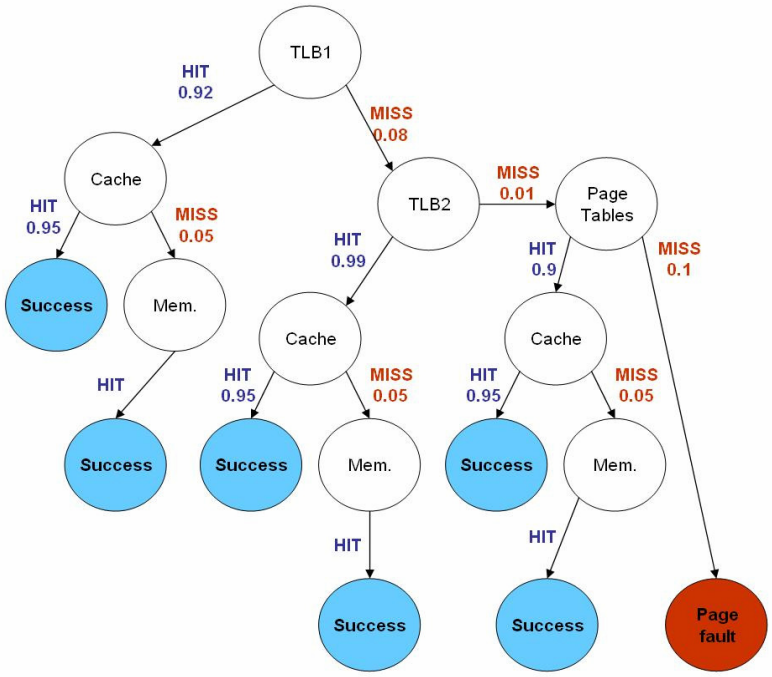
מסקנה – Page Fault יכול להתרחש רק אם היה קודם לכן cache miss וגם TLB miss.

דיאגרמת מצבים:



סעיף ב

1. **דיאגרמת מצבים**:



2. **חישוב זמן הגישה המשוקלל**:

.

$$\underbrace{P(TLB1_{hit})\cdot P(Cache_{hit})(1_{cache\_hit})}_{0.92\cdot 0.95}+$$

$$\underbrace{P(TLB1_{hit})\cdot P(Cache_{miss})(1_{cache\_access}+10_{cache\_penalty})}_{0.92\cdot 0.05\cdot 11}+$$

$$\underbrace{P(TLB1_{miss})\cdot P(TLB2_{hit})\cdot P(Cache_{hit})(1_{tlb2\_access}+1_{cache\_hit})}_{0.08\cdot 0.99\cdot 0.95\cdot 2}+$$

$$\underbrace{P(TLB1_{miss})\cdot P(TLB2_{hit})\cdot P(Cache_{miss})(1_{tlb2\_access}+1_{cache\_access}+10_{cache\_penalty})}_{0.08\cdot 0.01\cdot 0.05\cdot 12}+$$

$$\underbrace{P(TLB1_{miss})\cdot P(TLB2_{miss})\cdot P(Table_{hit})\cdot P(Cache_{hit})(1_{tlb2\_access}+1_{cache\_hit}+10_{page\_acsess})}_{0.08\cdot 0.01\cdot 0.9\cdot 0.95\cdot 12}+$$

$$\underbrace{P(TLB1_{miss})\cdot P(TLB2_{miss})\cdot P(Table_{hit})\cdot P(Cache_{miss})(1_{tlb2\_access}+1_{cache\_access}+10_{page\_acsess})}_{0.08\cdot 0.01\cdot 0.9\cdot 0.05\cdot 12}+$$

$$\underbrace{P(TLB1_{miss})\cdot P(TLB2_{miss})\cdot P(Table_{miss})(1_{tlb2\_access}+10_{page\_acsess}+100_{memory\_access})}_{0.09\cdot 0.01\cdot 0.1\cdot 111}$$

פתרון לשאלה 3

סעיף א

1. חלוקת הכתובת הוירטואלית

19 bits = TAG	1 bit = index	12 bit = offset

גודל דף הוא 4KB – מיוצג ע"י 12 ביט.
לcache של 16KB נכנסים 4 דפים ב2 ways – משמעות – דרוש ביט 1 לאינדקס.
שאר הביטים הולכים לTAG, (32-1-12=19).

2. חלוקת הכתובת הפיזית בTLB

13 bits = TAG	7 bit = index	12 bit = offset

גודל דף הוא 4KB – מיוצג ע"י 12 ביט.
לTLB יש 128 כניסות direct mapped – מיוצגות ע"י 7 ביט index.
שאר הביטים הולכים לTAG,(32-7-12=13)

סעיף ב

1.

I: גישה לcache רק אחרי הגישה לTLB: לצורך גישה לcache צריך 13 ביט (offset + index), ולפני הגישה לTLB יש לנו רק 12 ביטים של offset.
זמן גישה לcache הוא $\lceil 1+0.5 \rceil = 2$

II: גישה לcache במקביל לגישה לTLB: לcache של 16KB נכנסים 4 דפים ב4 ways – משמעות – לא דרושים ביטים לאינדקס. לכן: לצורך גישה לcache צריך 12 ביט, כמו הoffset של TLB.
זמן גישה לcache הוא $\lceil \max(1,0.5) \rceil = 1$

III. גישה לcache רק אחרי הגישה לTLB: : לcache של 32KB נכנסים 8 דפים ב4 ways – משמעות –דרוש ביט אחד לאינדקס. לכן: לצורך גישה לcache צריך 13 ביט), ולפני הגישה לTLB יש לנו רק 12 ביטים של offset.
זמן גישה לcache הוא $\lceil 1+0.5 \rceil = 2$

2.

זמן גישה ממוצע בI:

$$amat(I) = \underbrace{1}_{\text{cache and TLB access together}} + \underbrace{0.05}_{\text{cahce miss rate}} \cdot 16 = 1.8$$

$$amat(II) = \underbrace{2}_{\text{cache and TLB access NOT together}} + \underbrace{0.03}_{\text{cahce miss rate}} \cdot 16 = 2.48$$

$$amat(I) < amat(II)$$

פתרון לשאלה 4

סעיף א

באופן כללי, נרשום

$$CPI = CPI_{Ideal} + \frac{LoadInsctruction}{IC}\left(ReadHitTime + MissRate \cdot \left(ReadPenalty + DirtyRate \cdot DirtyPenalty\right)\right) + \frac{WriteInsctruction}{IC}\left(WriteHitTime + MissRate \cdot \left(WritePenalty + DirtyRate \cdot DirtyPenalty\right)\right)$$

שיטה א:

$$CPI_A = CPI_{Ideal} + 18\%\left(1 + M\left(\underbrace{20+7}_{ReadingTheBlock} + 50\%\left(\underbrace{20+8}_{WritingDirtyBlock}\right)\right)\right) + 8\%\left(2 + M\left(\underbrace{1}_{TagChecking} + \underbrace{20}_{WritingTheWord} + \underbrace{20+7}_{ReadingTheBlock} + 50\%\left(\underbrace{20+7}_{WritingDirtyBlock}\right)\right)\right)$$

שיטה ב:

בשיטה זו נחסוך בממוצע מחצית מזמן הקריאה של הבלוק המוחטא מהזיכרון הראשי:

$$CPI_B = CPI_{Ideal} + 18\%\left(1 + M\left(\left(\underbrace{20+7}_{ReadingTheBlock}\right)\cdot\frac{1}{2} + 50\%\left(\underbrace{20+7}_{WritingDirtyBlock}\right)\right)\right) + 8\%\left(2 + M\left(\underbrace{1}_{TagChecking} + \underbrace{20}_{WritingTheWord} + \left(\underbrace{20+7}_{ReadingTheBlock}\right) + 50\%\left(\underbrace{20+7}_{WritingDirtyBlock}\right)\right)\right)$$

שיטה ג:

בשיטה זו לא נחכה עד הבאת 7 המילים הנוספות של הבלוק המוחטא מהזיכרון הראשי:

$$CPI_C = CPI_{Ideal} + 18\%\left(1 + M\left(\left(\underbrace{20}_{ReadingTheCritialWord}\right) + 50\%\left(\underbrace{20+7}_{WritingDirtyBlock}\right)\right)\right) + 8\%\left(2 + M\left(\underbrace{1}_{TagChecking} + \underbrace{20}_{WritingTheWord} + \left(\underbrace{20+7}_{ReadingTheBlock}\right) + 50\%\left(\underbrace{20+7}_{WritingDirtyBlock}\right)\right)\right)$$

<u>סעיף ב</u>

חלק 1

ראשית עבור קונפיגורציה של $16KB,\ 4\,way$: גודל בלוק הוא 8 מילים, ולכן גודל שדה ה $Offset$ הוא $\log_2 8 = 3$ ביטים. מספר ה $Sets$ הוא $Sets = \frac{Cache\ Size}{Associativity} = \frac{16KB}{4} = 4KB$, ולכן גודל שדה ה $Index$ הוא $\log_2\left(4\cdot 1024\right) = 12$ ביטים. לבסוף, גודל שדה ה Tag הוא $32-3-12=17$ ביטים.

שנית עבור קונפיגורציה של $32KB,\ 2\,way$: גודל בלוק הוא 8 מילים, ולכן גודל שדה ה $Offset$ הוא $\log_2 8 = 3$ ביטים. מספר ה $Sets$ הוא $Sets = \frac{Cache\ Size}{Associativity} = \frac{32KB}{2} = 16KB$, ולכן גודל שדה ה $Index$ הוא $\log_2\left(16\cdot 1024\right) = 14$ ביטים. לבסוף, גודל שדה ה Tag הוא $32-3-14=15$ ביטים

חלק 2

ראשית, מכיוון שקצב הפגיעה ב TLB הוא 100%, עלינו רק להוסיף את זמן תרגום הכתובת ב TLB ל CPI_C שחושב בסעיף הקודם, וכך נקבל, עבור הקונפיגורציה הראשונה והשנייה בהתאמה:

$$CPI_1 = CPI_{Ideal} + 18\%\left(1 + M_1\left(\left(\underbrace{20}_{ReadingTheCritialWord}\right) + 50\%\left(\underbrace{20+7}_{WritingDirtyBlock}\right)\right)\right) + 8\%\left(2 + M_1\left(\underbrace{1}_{TagChecking} + \underbrace{20}_{WritingTheWord} + \left(\underbrace{20+7}_{ReadingTheBlock}\right) + 50\%\left(\underbrace{20+7}_{WritingDirtyBlock}\right)\right)\right) + 1$$

$$= 1.5 + \frac{18}{100}\left(1 + \frac{2}{100}\left(20 + \frac{50}{100}\cdot 27\right)\right) + \frac{8}{100}\left(2 + \frac{2}{100}\left(1 + 20 + 27 + \frac{50}{100}27\right)\right) + 1$$

$$= 2.938$$

$$CPI_2 = CPI_{Ideal} + 18\%\left(1 + M_2\left(\left(\underbrace{20}_{ReadingTheCritialWord}\right) + 50\%\left(\underbrace{20+7}_{WritingDirtyBlock}\right)\right)\right) + 8\%\left(2 + M_2\left(\underbrace{1}_{TagChecking} + \underbrace{20}_{WritingTheWord} + \left(\underbrace{20+7}_{ReadingTheBlock}\right) + 50\%\left(\underbrace{20+7}_{WritingDirtyBlock}\right)\right)\right) + 1$$

$$= 1.5 + \frac{18}{100}\left(1 + \frac{1.4}{100}\left(20 + \frac{50}{100}\cdot 27\right)\right) + \frac{8}{100}\left(2 + \frac{1.4}{100}\left(1 + 20 + 27 + \frac{50}{100}27\right)\right) + 1$$

$$= 2.719$$

הבחירה תהיה בקונפיגורציה השנייה, בה CPI קטן יותר.

פתרון לשאלה 5

סעיף א

גודל כל טבלה בזיכרון זהה לגודל דף, $4KB$. גודל כל כניסה בטבלה הוא $8B$, ולכן מספר הכניסות בטבלה אחת הוא

$$\frac{4K}{8} = \frac{2^2 \cdot 2^{10}}{2^3} = 2^9$$

ולכן כל טבלה תוכל למפות 9 ביטים, לכל היותר, מתוך הכתובת הוירטואלית כולה.
כדי למפות כתובת וירטואלית לפיזית יהיה צורך ב

$$\left\lceil \frac{Virtual\ Address\ Bits - Offset\ Bits}{9} \right\rceil = \left\lceil \frac{48-12}{9} \right\rceil = \left\lceil \frac{36}{9} \right\rceil = 4$$

טבלאות, ולכן מיפוי דף וירטואלי חדש ייקח 4 יחידות זמן.

סעיף ב

נרצה להגריל בעזרת פונקצית הערבול כתובת בטבלת המיפוי, עד אשר נמצא כתובת פנוייה.

גודל טבלת המיפוי הסופית הוא

$$S = k \cdot Pysical\ Pages = k \cdot \frac{Pysical\ Bytes}{Page\ Size} = k\frac{4G}{4K} = k\frac{4 \cdot 2^{30}}{4 \cdot 2^{10}} = 2^{20}k$$

מכיוון שהזיכרון הפיסי מלא, יש 2^{20} שורות מלאות בטבלה. ההסתברות שנגריל בעזרת פונקצית הערבול שורה ריקה היא:

$$p = \frac{2^{20}(k-1)}{2^{20}k} = \frac{k-1}{k}$$

נסמן ב N את ההסתברות למצוא שורה ריקה בהגרלה (של פונקצית ערבול) ה n. N יהיה משתנה אקראי גיאומטרי:

$$P\{N = n\} = (1-p)^{n-1}\ p$$

ולכן

$$EN = \frac{1}{p} = \frac{k}{k-1}$$

וזהו מספר יחידות הזמן (כאשר כל יחידה משמעה זמן הגרלת מספר בעזרת פונקצית הערבול), בממוצע, שיקח למפות דף וירטואלי חדש.

<u>סעיף ג</u>

במחשב רגיל, ניקח את מדד זמן הגישה הממוצע. לכן נעדיף את שיטה א׳ כאשר $6 < \frac{k}{k-1}$, או $k < \frac{6}{5}$.

לכן, עבור $k = 4$ נעדיף את שיטה ב׳.

לבקרת מערכת זמן אמת, נחפש את זמן הגישה המקסימלי הקטן ביותר.
ההסתברות שזמן הגישה בשיטה ב׳ גדול מ t יחידות זמן:

$$P\{N > t\} = \sum_{n=t+1}^{\infty} (1-p)^n \; p = p\frac{(1-p)^{t+1}}{1-(1-p)} = (1-p)^{t+1} = \left(1 - \frac{k-1}{k}\right)^{t+1} = \frac{1}{k^{t+1}}$$

הסתברות שקטנה אספוננציאלית, ולכן מעשית עבור $t = 6$, כדי להשוות עם שיטה א׳, נקבל $\frac{1}{k^7}$ ונעדיף את שיטה ב׳ בד״כ, כלומר עבור k מספיק גדולים. עבור $k = 4$, נקבל כי הסיכוי לקבלת זמן גישה יותר גדול מ 3 יחידות זמן הוא $\frac{1}{4^4} \sim 0.4\%$, והסיכוי לקבל זמן גישה יותר גדול מ 4 יחידות זמן הוא $\frac{1}{4^5} \sim 0.09\%$, כלומר הסתברותית, זמני הגישה בשיטה ב׳ קטנים יותר.

<u>סעיף ד</u>

נעדיף את שיטה ב׳ מכיוון שכאשר הערבול נכשל, לערבול הבא יש יותר שורות לבחור מהם. בשיטה ג׳ אנו רק הולכים לתא הבא, ולכן בממוצע אנחנו נחפש תאים פנויים רק מתוך מחצית מגודל הטבלה בעוד בשיטה ב׳ נגריל שורה מתוך כל השורות של הטבלה.

בנוסף, נראה כי ככל ש k גדול יותר, נעדיף את שיטה ג׳, משום שכך יהיו, באופן התסברותי, יותר מרווחים ואז במקרה של ערבול כושל הסיכוי שהתא הבא יהיה פנוי גדול יותר. ומאידך, k גדול יותר יעזור להסתברות מציאת דף פנוי גם בשיטה ב׳.

לכן נשאר עם העמדה הקודמת – שיטה ב׳ עדיפה.

©2006 אבי בנדל www.technion.co.il

פתרון לשאלה 6

סעיף א

מכיוון שטבלת התרגום של מרחב ה *User* נמצאת במרחב הוירטואלי של כתובות ה *System* , הרגיסטרים יכילו כתובת וירטואליות, שמצביעות למרחב ה *System* .

סעיף ב

רצוי כי טבלת התרגום של מרחב ה *System* תשאר תמיד בזכרון, בגלל שתהליכי *System* הם תכופים מאוד בפעולת המחשב. מאותה סיבה, כלומר מכיוון שתהליכי ה *User* הם פחות תדירים, אין צורך להשאיר את טבלת התרגום של מרחב ה *User* בזכרון.

סעיף ג

כן, ע"פ הסתכלות בשדות ה *PTE* אנו רואים שאין התייחסות ל *PID* , ולכן נסיק כי טבלת התרגום אחידה לכל התהליכים. ה *PID* מקודד, כנראה, בכתובת הוירטואלית או שנעשה שימוש באוגר נוסף שמכיל את ה *PID* .

סעיף ד

מכיוון שטבלת התרגום של מרחב ה *User* נמצאת במרחב הזיכרון הוירטואלי של ה *System* , אזי לשום תהליך שאינו של המערכת אין גישה לטבלה זו.

סעיף ה

במקרה הגרוע ביותר יתכנו 4 קריאות לזיכרון הפיזי:

1. קריאת שלוש שורות של *PTE* המתאימות לגישה מטבלת התרגום, שנמצאות בזיכרון הוירטואלי, כאשר כתובתה של השורה הראשונה נמצאת באוגרים *P1BR* או *P0BR* ושתי שורות ה *PTE* העוקבות נמצאות בזכרון הוירטואלי.
   כל קריאה של שורת *PTE* גוררת קריאה של *PTE* ממרחב ה *System* .
   לסיכום, לאחר הבנת תוכן *P1BR* או *P0BR* וקריאה של כתובת זו, יש לבצע לכל היותר שלוש קריאות נוספות של *PTE* ממרחב ה *User* , כלומר סה"כ 6 קריאות של זכרון.
   מתוך 6 קריאות אלו, קריאות הטבלאות של מרחב ה *User* הן אלו שיכולות לגרור *Page Faults* , ולכן סה"כ יכולים להיות 3 מקרים של *Page Faults* .
2. קריאה של הנתון עצמו: קריאה פיסית נוספות היכולה לגרור *Page Fault* .

סה"כ קיבלנו 7 קריאות זכרון, כאשר מתוכן 4 יכולות לגרור *Page Faults* .

גליון 5

פתרון לשאלה 1

סעיף א

החיזוי החשוב יותר הוא כיוון הקפיצה משום שחיזוי זה בעצם יקבע אם יש צורך לקפוץ, ובכך יכול לחסוך את מחזורי ה *Stall* שיש להכניס לביצוע התוכנית המקורית, וזוהי המהות של חיזוי הקפיצה.

סעיף ב

על מנת להחליט מאין יש לקחת את חיזוי כיוון הקפיצה, על המעבד לדעת:

- השדה הרלוונטי ב *OpCode* לגבי ה *Software Prediction*

על מנת להחליט מאין יש לקחת את חיזוי כתובת הקפיצה, על המעבד לדעת

- האם התוכנית מבצעת כרגע פונקציה? אם כן עלינו להתשמש ב *RSB*.
- האם פקודת הקפיצה היא עקיפה? אם כן עלינו להשתמש ב *Indirect Address Cache*.
- אחרת, נשתמש ב *BTB*.

סעיף ג

יתרונו של חיזוי התוכנה הוא בזה שהקומפיילר מסתכל על כל קוד התוכנית מראש ולכן יכול לחזות יותר טוב את התנהגותה.

חסרונות:

- זמן קומפילציה ארוך יותר ותכנות מסובך יותר של הקומפיילר
- הצורך להגדיל את גודל הפקודות הקבוע בשתי סיביות

יתרונו הגדול של חיזוי התוכנה הוא המוטיבציה לשימוש בו – העובדה כי הקומפיילר "מסתכל" על כל התוכנית א-פריורי ולכן יכול לחזות די טוב את הקפיצות. כדאי להתעלם מחיזוי החומרה כשאפשר כדי לחסוך זמן – חיזוי התוכנה הוא מיידי עבור החומרה (מסתמך על קריאת שני ביטים מקוד הפקודה) וחיזוי החומרה הוא תהליך ארוך יותר.

דוגמה לתוכנית/הסבר	פעולה	*Opcode*
If (R1>R2) { … } כאן עדיף לחזות ע"י החומרה, כלומר בעזרת הדינאמיקה של התוכנית עצמה, משום לתוכנה לא יהיה חיזוי "חכם" לתת.	בטל חיזוי תוכנה – חיזוי באמצעות חומרה בלבד	00
While (!eof) { … } ברוב המקרים, קפיצה זו תלקח.	תוכנה חוזה כי הקפיצה נלקחת	01
If (1>2) { … } קפיצה שתמיד לא תלקח.	תוכנה חוזה כי הקפיצה לא נלקחת	10
כאשר אנו יוצרים אובייקטים חדשים, בתכנות OOP, או כאשר אנ קוראים יחידות מידע ממסד נתונים, ועל סמך הנתונים החדשים הללו אנו צריכים לבצע קפיצה, אזי ההסתברות לקפיצה היא $\frac{1}{2}$ ולכן אין העדפה לקפיצה או ללא קפיצה, ולכן לא נחזה כי כך רק נבזבז זמן על החיזוי.	לא ניתן לחזות גם ע"י חומרה	11

סעיף ד

Type	%	Address Prediction		Direction Prediction		Penalty
		Predictor	Success	Predictor	Success	
Jump						
Direct	15%	BTB	95%	Taken	100%	ניקוי הצינור עד $DEC2$ שלב, כולל, כי כיוון הקפיצה ידוע תמיד.
Return	5%	RSB	97%	Taken	100%	
Indirect	5%	IBC (50%) BTB (50%)	100% 0%	Taken	100%	
Brach (opcode)						
00 none	60%	BTB	95%	BP	90%	ניקוי עד $EX4$ אם חיזוי הכיוון לא הצליח. אם כן הצליח, ניקוי עד $DEC2$ אם חיזוי הכתובת לא הצליח.
01 taken	7%	BTB	95%	Taken	98%	
10 not taken	7%	BTB	95%	Not taken	98%	
11 unpredicted	1%	None used	50%	None used	50%	

$$CPI = CPI_{Ideal} + \underbrace{15\%\left(2\cdot 5\%\right)+5\%\left(2\cdot 3\%\right)+5\%\left(2\cdot 50\%+0\cdot 50\%\right)}_{unconditional\ jumps} + \left(\begin{array}{l} 60\%\left(6\cdot 10\%+2\cdot 5\%\cdot 90\%\right) \\ +7\%\left(6\cdot 2\%+2\cdot 5\%\cdot 98\%\right) \\ +7\%\left(6\cdot 2\%+2\cdot 5\%\cdot 98\%\right) \\ \underbrace{+1\%\left(6\cdot 50\%+2\cdot 50\%\cdot 50\%\right)}_{conditional\ branches} \end{array}\right)$$

$$= 1.5 + 0.068 + 0.09 + 2\cdot 0.01526 + 0.035 = 1.5 + 0.22352 = 1.72352$$

פתרון לשאלה 2

סעיף א

נניח כי גודל כתובת פקודה הוא 32 סיביות. לכן שדה הכתובת לקפיצה, *Target Address* הינו גם ברוחב 32 סיביות.

עבור כל שורה (כל פקודת קפיצה ב *BTB*) יש לשמור 2^k שורות הסטוריה הכוללות את שתי סיביות החיזוי (*2 – bit saturated counter*) עבור כל הסטוריה בנפרד, ובנוסף יש לשמור את ההיסטוריה עצמה, כלומר את k הביטים. לכן, לסיכום נקבל

$$BTB\ Size = n\left(32+32+2^k\cdot 2+k\right) = n\left(64+2^{k+1}+k\right)$$

סעיף ב

מספר	היסטוריה	מצב מונה	חיזוי	ת'כלס	מצב חדש	הערה
1	000	SNT	לא נלקח	נלקח	WNT	חיזוי לא עבד
2	001	SNT	לא נלקח	נלקח	WNT	חיזוי לא עבד
3	011	SNT	לא נלקח	נלקח	WNT	חיזוי לא עבד
4	111	SNT	לא נלקח	לא נלקח	SNT	
5	110	SNT	לא נלקח	נלקח	WNT	חיזוי לא עבד
6	101	SNT	לא נלקח	נלקח	WNT	חיזוי לא עבד
7	011	WNT	לא נלקח	נלקח	WT	חיזוי לא עבד
8	111	SNT	לא נלקח	לא נלקח	SNT	
9	110	WNT	לא נלקח	נלקח	WT	חיזוי לא עבד
10	101	WNT	לא נלקח	נלקח	WT	חיזוי לא עבד

11	011	WT	נלקח	נלקח	ST	
12	111	SNT	לא נלקח	לא נלקח	SNT	
13	110	WT	נלקח	נלקח	ST	
14	101	WT	נלקח	נלקח	ST	
15	011	ST	נלקח	נלקח	ST	
16	111	SNT	לא נלקח	לא נלקח	SNT	
17	110	ST	נלקח	נלקח	ST	
18	101	ST	נלקח	נלקח	ST	
19	011	ST	נלקח	נלקח	ST	
20	111	SNT	לא נלקח	לא נלקח	SNT	

מהטבלה אנו רואים כי מספר החיזויים השגויים עבור 10 המופעים הראשונים של פקודה 10 הוא 8.

<u>סעיף ג</u>

ננתח כל פקודה בנפרד, בגלל הרישום ההיסטורי הנפרד.
עבור פקודה 10, סה"כ ראינו 8 טעויות בחיזוי, אך החל מההופעה ה 11 של הפקודה, אין יותר טעויות חיזוי.
עבור שאר פקודות הקפיצה, במצב יציב לא תהינה טעויות בחיזוי מלבד פעם אחת שפקודה 13 לא תלקח בסוף ריצת התוכנית כולה, ולכן

$$BMR = \frac{1}{Dynamic\ Branches} = \frac{1}{10+5\cdot 4+10} = \frac{1}{40}$$

<u>סעיף ד</u>

ככל שישנן יותר סיביות הסטוריה, ניתן לזכור יותר מקרי הסטוריה וכך איכות החיזוי עולה. במקרה שלנו נוכל לזכור לכל היותר דפוס התנהגות באורך של $2^3 = 8$, ואורך הלולאה בסעיף ב' הוא 4 ולכן יכלנו, לאחר "תקופת הסתגלות" של החזאי, לחזות במדוייק את התנהגות פקודה 10.
מצד שני, ככל שישנן יותר סיביות הסטוריה, יקח לחזאי יותר זמן ללמוד את מצב הלולאה, והוא יהפוך ליעיל רק אחרי מספר רב יותר של איטרציות.

<u>סעיף ה</u>

1. הוספת רמת אמון (*Confidence Level*) לחזאי תגרום לפחות חיזויים ספקולטיביים ולכן תגרום לצריכה נמוכה יותר של אנרגיה, ולכן תפחית את ה *EPI* .
2. הוספת רמת אמון לחזאי תקטין את מדד *MIPS* מכיוון שננסה לחזות פחות קפיצות, ולכן הסתברותית יש סכוי גדול יותר שנזדקק ל *Flush* של הצינור. כבר ראינו שהוספת רמת אמון תפחית את ה *EPI* , ולכן ההשפעה על מדד $\frac{MIPS}{EPI}$ לא חד-משמעי. אם נצליח ליצור רמות אמון כאלה שלא יפחיתו את ה *MIPS* יותר מדי, אזי נוכל לקבל שיפור כולל במדד $\frac{MIPS}{EPI}$.
3. במעבד התומך ב *Multi – Threading* , ההקטנה ב *MIPS* לא תהיה משמעותית כמו במעבד שבשאלה 2, זאת מכיוון שניתן תמיד למלא את הצינור בפקודות של *Thread* אחר. לכן תשובתנו כעת היא שמדד $\frac{MIPS}{EPI}$ יראה שיפור טוב יותר מאשר בשאלה 2.

©2006 אבי בנדל www.technion.co.il

גליון 6

חלק I – הערכת הסימולטור

סעיף א

1. המעבד מניח שלא תהיה קפיצה, וטוען את הפקודה הבאה אחרי פקודת הbranch בזכרון.

2. זוהי אינה מדיניות טובה עבור התוכנית, משום שהקפיצה תתבצע כמעט תמיד (לא תהיה קפיצה רק באיטרציה האחרונה).

3. ניתן להשתמש בdelayed branch, כאשר בפקודה שאחרי הbranch נשים את הפקודה הראשונה של הלולאה (מדיניות "from target", שמתאימה למקרה בו רוב הקפיצות נלקחות).

סעיף ב

1. אם הפקודה הבאה בתור אחרי פקודת FP משתמשת בתוצאת החישוב של פקודת הFP, היא תחכה בstall, עד שהALU יחשב את תוצאת החישוב.

2. פקודת הכפל השניה תחכה בstall עד שהפקודה הראשונה תסיים את פעולת החישוב בALU (וכל הפקודות שנטענו לאחר מכן יחכו גם, כמובן). התופעה תקרה גם אם אין קשר בין האופרנדים של שתי פעולות הכפל – בגלל מחסור במשאבים.

3. ניתן להוסיף עוד יחידת כפל של מספרים ממשיים. כל עוד אין קשר בין אופרנדים של 2 הפקודות הרציפות (Data Hazard), הפקודה השניה לא תתעכב.

4. אם יש יותר מיחידת עיבוד אחת עבור סוג חישוב נתון (כפל או חילוק), המעבד יהיה מסוגל לחשב במקביל מספר פקודות FP גדול יותר. אם יתרחש Data Hazard, המעבד יצטרך בכל זאת להוסיף stalls בין הפקודות, אם הן צמודות מדי.

חלק II – תכנון ארכיטקטוני

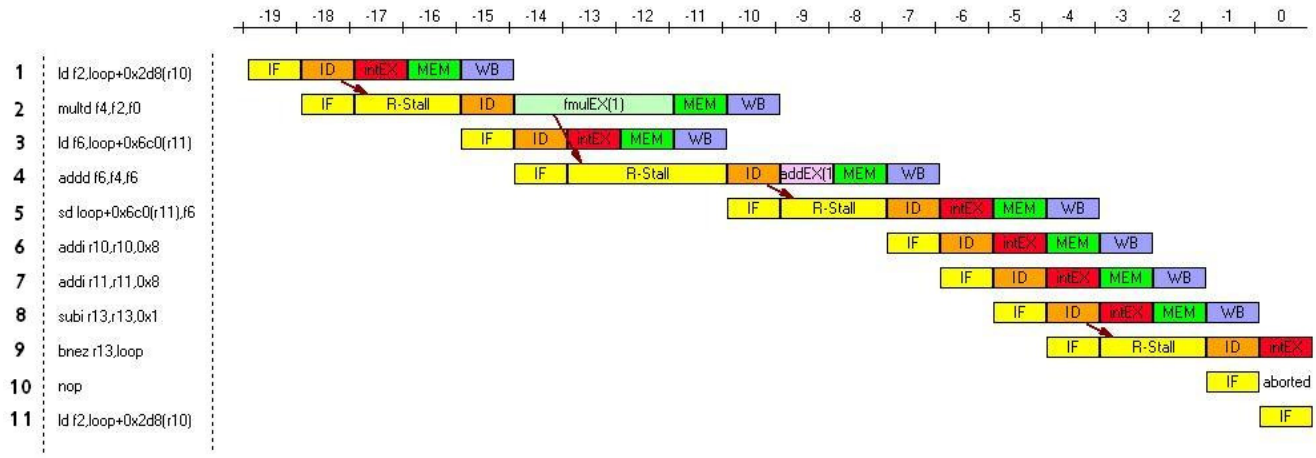
סעיף א

1. כל התוכנית רצה במשך 389 מחזורים. מספר מחזורי השעון עבור איטרציה אחת במצב יציב – 19, כאשר מספר הפקודות שמתבצע הוא 10 (כולל nop שמוכנס אחרי פקודת הbranch, ז"א יש רק 9 פקודות אמיתיות).

2. יש מספר סיבות לכך שבוזבזו 10 מחזורים:
- בגלל שאין Forwarding, כל פעם שיש Data Hazard, צריך לחכות עד סיום שלב הWB של הפקודה שמחשבת אופרנד שנחוץ לפקודה אחרת.
- בגלל מדיניות שמניחה שלא תהיה קפיצה – מבצעים את פעולת הnop, כאשר היה ניתן להמנע מכך במדיניות מתאימה יותר.

3. כדי להקטין את השפעת 2 הסיבות שהזכרנו בסעיף הקודם ניתן:
- לתמוך בביצוע Forwarding, מה שיקטין את מספר המחזורים בהן הפקודות יחכו לערכי אופרנדים.
- לשנות את המדיניות עבור Control Hazards, כפי שהצענו בחלק I, סעיף א3. פתרון אחר הוא Loop Unrolling, אבל הוא יעיל פחות.

4. נתבונן בתרשים זרימת הפקודות עבור איטרציה 2 (מצב יציב):



נסכם את הארועים המתקבלים בכל מחזור שעון בטבלה הבאה:

מחזור (אינדקס לפי התרשים)	ארועים
-19	Inst. 1 in IF.
-18	Inst. 1 ID, Inst. 2 Mult. in IF .
-17	No new inst. because of STALL (Inst. 2 needs Inst. 1 to do WB).
-16	Same as -17.
-15	Inst. 3 in IF, Inst. 1 in WB.
-14	Inst. 4 in IF, Inst. 2 started calculating FP mult.
-13	Inst. 4 in STALL because it needs result of FP mult. Of Inst. 2.
-12	Same as -13.
-11	Same as -12, Inst. 3 in WB, Inst. 2 finished calculating Mult. In ALU.
-10	Inst. 4 receives the result of Mult., STALL ended, Inst. 5 in IF. Inst. 2 in WB.
-9	Inst. 5 in STALL because it needs result of Inst. 4.
-8	Same as -9.
-7	Inst. 5 gets the result from Inst. 4, Inst. 6 in IF. Inst. 4 in WB.
-6	Inst. 7 in IF.
-5	Inst. 8 in IF.
-4	Inst. 9 in IF. Inst. 5 in WB.
-3	Inst. 9 (Branch) in STALL because it needs the result of Inst. 8. Inst. 6 in WB.
-2	Same as -3. Inst. 7 in WB.
-1	Inst. 10 (nop) in IF. Inst. 8 in WB.
0	Inst. 10 aborted because Branch taken. Inst. 11 in IF (same as Inst. 1)

<u>סעיף ב</u>

1. ביצועי המערכת השתפרו (269 מחזורים לביצוע כל התוכנית, לעומת 389).

2. יש שיפור משמעותי משמעותי, משום שרוב המחזורים שבוזבזו בריצה הקודמת התרחשו בגלל הצורך לחכות לשלב הWB של הפקודה הקודמת במקרה של Data Hazard-RAW. עדיין מתרחשים שני Structural Hazards ו Data Hazard אחד.

3. מספר המחזורים שלוקחת איטרציה, כאשר הForwarding קיים הוא 13. זהו כמובן שיפור לעומת הריצה הקודמת.

4. $speedup = \frac{19}{13} = 1.46$

<u>סעיף ג</u>

1. התוכנית רצה 389 מחזורים – בדיוק כמו התוכנית עבור הקונפיגורציה של סעיף א'.
למרות שפעולת הכפל מסתיימת יותר מהר, הפקודה שקודם חיכתה לחישוב המכפלה להסתיים, מחכה עכשיו לסיום הפעולה שלאחר הכפל. זאת אומרת, שלמרות שפעולת הכפל עצמה מסתיימת יותר מהר, הפקודה אותה היא עיכבה מסתיימת באותו הזמן.

2. כמו שנאמר בסעיף 1, יש שיפור בפעולת הכפל בלבד.

3. זמן ביצוע של איטרציה בודדת הוא 19 מחזורים.

4. מספר מחזורי ריצה של איטרציה זהה, לכן ה speedup שווה ל1.

©2006 אבי בנדל www.technion.co.il

חלק III – שיקולים בכתיבת התוכנית

<u>סעיף א</u>

```
      ;**************  saxpy_s.s ***************
        .text
        .global main
main:
        ;*** Initialization
        lw       r10, zero      ; init x vector pointer
        lw       r11, zero      ; init y vector pointer
        lw       r13, rounds    ; Load vector size
        ld       f0, a          ; load a

        ;*** Start the loop
loop:   ld      f2,1000(r10)    ; f2 <= X[r10]
        multd   f4, f2, f0      ; f4 <= f2 * a
        ld      f6,2000(r11)    ; f6 <= Y[r11]
        addi    r10, r10, #8
        subi    r13,r13,#1
        addd    f6, f4, f6      ; f6 <= f4 + f6
        sd      2000(r11), f6   ; Y[r11] <= f6
        addi    r11, r11, #8
        bnez    r13,loop
        nop
        trap    #0
        nop
```

1. מספר מחזורי השעון שלקחה איטרציה במצב יציב הוא 15.

2. ה-speedup יחסית לקונפיגורציה זהה, ללא שינוי סדר הקוד היא: $\frac{19}{15}=1.26$

3. מתוצאות הריצה, ניתן לראות שהשגנו שיפור יחסית לקונפיגורציה ללא שינוי סדר הפקודות. הסיבה לכך היא שהעברנו פקודות לתוך מקומות בהם הן עדיין מבצעות פעולה חוקית מבחינת האלגוריתם, וגם חוצצות בין שתי פקודות שמשתמשות באופרנדים זהים (ויוצרים Data Hazards). זהו פתרון חלקי, כי לא תמיד יש מספיק פקודות להעביר כדי למלא את המחזורים שמתבזבזים על המתנה לחישוב אופרנד.

זאת אנו רואים כי תוצאות השימוש בforwarding ללא שינוי סדר קוד היו יותר מוצלחות ($\frac{13}{15}=0.86$).

©2006 אבי בנדל www.technion.co.il

סעיף ב

```
      ;**************  saxpy_u.s ***************
        .text
        .global main
main:
        ;*** Initialization
        lw       r10, zero      ; init x vector pointer
        lw       r12, zero      ; init x vector 2nd pointer
        lw       r11, zero      ; init y vector pointer
        lw       r13, zero      ; init y vector 2nd pointer
        addi     r12, r10, #8   ; adding 8 to 2nd x pointer
        addi     r13, r10, #8   ; adding 8 to 2nd y pointer
        lw       r14, rounds    ; Load vector size
        ld       f0, a          ; load a

        ;*** Start the loop
loop:   ld      f2,1000(r10)    ; f2  <= X[r10]
        ld      f4,1000(r12)    ; f4  <= X[r12]
        subi    r14, r14, #2
        multd   f6, f2, f0      ; f6  <= f2 * a
        ld      f10,2000(r11)   ; f10 <= Y[r11]
        ld      f12,2000(r13)   ; f12 <= Y[r13]
        multd   f8, f4, f0      ; f8  <= f4 * a
        addi    r10, r10, #16
        addi    r12, r12, #16
        addd    f10, f6, f10    ; f10 <= f6 + f10
        addd    f12, f8, f12    ; f12 <= f8 + f12
        sd      2000(r11), f10  ; Y[r11] <= f10
        sd      2000(r13), f12  ; Y[r13] <= f12
        addi    r11, r11, #16
        addi    r13, r13, #16
        bnez    r14,loop
        nop
        trap    #0
        nop
```

1. איטרציה אחת במצב מתמיד לוקחת 19 מחזורים. (17 פקודות ו2 מחזורי stall).

2. למרות שאיטרציה אחרי loop unrolling לוקחת יותר מחזורים, צירך להתחשב בעובדה שהיא מבצעת פעולות מקבילות ל2 איטרציות של הקוד לפני פריסת הלולאה. לכן הspeedup יהיה שווה ל: $\frac{19}{0.5 \cdot 19} = 2$
זהו כמובן שיפור משמעותי, של 100% בביצוע התוכנית לעומת הקונפיגורציה וכתיבת הקוד בסעיף IIא.

3. קיבלנו תוצאות טובות אפילו מחלק II, סעיף ב', בו הפעלנו את הforwarding. הסיבה לכך היא שהוספת הפקודות החדשות מהאיטרציה השניה מאפשרות לנו למלא יותר רווחים בהם אפילו ה forwarding לא עזר, ועדיין לשמור על נכונות האלגוריתם.

©2006 אבי בנדל www.technion.co.il

סעיף ג

```
      ;**************  saxpy_p.s ***************
        .text
        .global main
main:
        lw      r10, zero       ; init x vector pointer
        lw      r11, zero       ; init y vector pointer
        lw      r13, rounds     ; Load vector size
        ld      f0, a           ; Load a
        ld      f2,1000(r10)    ; f2 <= X(0)
        multd   f6, f2, f0      ; f6 <= f2 * a (calculating aX(0))
        ld      f4,2000(r11)    ; f4 <= Y(0)
        addd    f4, f6, f4      ; f4 <= f4 + f6 (calculating Y(0)=aX(0)+Y(0))
        ld      f2,1008(r10)    ; f2 <= X(1)

loop:   sd      2000(r11), f4   ; Y[r11] <= f4 (inserting Y(i-1))
        ld      f4,2008(r11)    ; f4 <= Y[r11 + 8] (getting Y(i))
        multd   f6, f2, f0      ; f6 <= f2 * a  (calculating X(i))
        subi    r13,r13,#1      ; decreasing loop counter
        ld      f2,1016(r10)    ; f2 <= X[r10+16] (getting  X(i+1))
        addi    r11, r11, #8    ; increasing pointer
        addi    r10, r10, #8    ; increasing pointer
        addd    f4, f6, f4      ; f4 <= f4 + f6 (calculating Y(i)=aX(i)+Y(i))
        bnez    r13,loop

        sd      2000(r11), f4   ; Y[r11] <= f4 (inserting Y(rounds))
        trap    #0
        nop
```

1. במצב מתמיד, איטרציה אחת (9 פקודות) לוקחת 11 מחזורי שעון.

2. משני מחזורי השעון שבוזבזו, אחד נובע מstall ואחד בגלל שמניחים שהקפיצה לא מתרחשת.
יכול להיות שהוספת מחזורים של software pipe תפתור את ה stall האחרון, אך סיבוכיות הפתרון וההוראות שיתווספו מחוץ ללולאה הופכות את הדבר ללא כדאי.

3. ניתן להשוות ישירות בין מספר המחזורים של איטרציה בסעיף זה לבין חלק II, סעיף א', למרות שבסעיף הזה, החישובים בתוך האיטרציה הם על שלבים שונים בוקטורים, אך התוכן של החישובים כמעט זהה.
לכן : $speedup = \frac{19}{11} = 1.72$

4. קיבלנו תוצאות יותר טובות מאשר הפעלת ה forwarding בחלק II, סעיף ב', הסיבה היא שהצלחנו לרופף את הקשר בין הפקודות השונות בלולאה, כאשר ה forwarding פתר את הבעיה הזאת בצורה חלקית.

5. התוצאות בחלק II, סעיף ג' היו לא טובות בכל מקרה, לכן השיפור כאן הוא משמעותי לעומתן (speedup זהה לסעיף 3).

סעיף ד

מהאפשרויות השונות שניסינו, נראה כי loop unrolling היה הכי כדאי עבור התוכנית הנתונה.
נשווה את התוצאות של loop unrolling עם software pipelining, הפתרון השני בטיבו :
loop unrolling היה מעט יותר מוצלח, משום שהיו בו פחות איטרציות (ובכ"א מהן היו 3 מחזורים מבוזבזים לעומת 2 בלולאות היותר קטנות של ה software pipelining).

©2006 אבי בנדל www.technion.co.il

סעיף ה

1. השילוב המוצלח ביותר הוא loop unrolling עם forwarding.

2. במקרה כזה אנו מקבלים סיום של איטרציה ב18 מחזורים (מחזור אחד יותר טוב מאשר אותה קונפיגורציה ללא forwarding). היחס בין מספר הפקודות שהופעלו בתוכנית למספר המחזורים הוא:
$\frac{\text{Inst. Count}}{\text{Num.of cycles}} = \frac{170}{193} = 0.88$, וזהו הקרוב הכי טוב ליחס של 1, האופטימלי בpipeline עם מסלול יחיד.

3. נסתכל על היחס $\frac{\text{Inst. Count}}{\text{Num.of cycles}}$ עבור 2 הקונפיגורציות בנפרד:

$\frac{\text{Inst. Count}}{\text{Num.of cycles}}(\text{Only loop unrolling}) = \frac{170}{203} = 0.837$	$\frac{\text{Inst. Count}}{\text{Num.of cycles}}(\text{Only forwarding}) = \frac{189}{269} = 0.702$

ניתן לראות ששימוש בloop unrolling לבד נותן תוצאות פחות טובות ב5% בלבד, לכן אולי השקעה ב2 מנגנונים אינה משתלמת כל כך, ועדיף להשאר עם מנגנון ה loop forwarding בלבד.

4. ניתן לחסוך עוד מספר מחזורים אם נשתמש במדיניות delayed branch, כאשר בפקודה שאחרי הbranch נשים את הפקודה הראשונה של הלולאה (מדיניות "from target", שמתאימה למקרה בו רוב הקפיצות נלקחות), ובמקרה כזה, לא נצטרך גם את פקודת הnop אחרי פקודת הbranch.

גליון 7

פתרון לשאלה 1

סעיף א

1. ישולם $Branch\,Penalty$ בכל מקרה (גם כאשר הקפיצה נלקחת וגם כאשר אינה נלקחת) והוא באורך של שלושה שלבי צינור
2. נחשב:

$$CPI = CPI_{Ideal} + \frac{15}{100}\cdot 3 = 1.45$$

סעיף ב

1. זהו ה $Branch\,Penanlty$ שבסעיף הקודם, כלומר שלושה מחזורי שעון.
2. התוספת ל CPI האידיאלי היא $CPI_{extra} = (1-p_{BTB})\frac{15}{100}\cdot 3$ ולכן נדרוש:

$$(1-p_{BTB})\frac{15}{100}\cdot 3 < 0.1 \;\Rightarrow\; 1-p_{BTB} < 0.22 \;\Rightarrow\; p_{BTB} > 0.78$$

סעיף ג

(1) העובדה כי הפקודה המייצרת את R_X לא משפיעה, מכיוון שאין יחידת $Forwaring$ במעבד זה, מה שמאלץ אותנו לחכות עד שלב ה WB של כל פקודה כזו.
נגדיר DH_i כמרחק בין הפקודה שלנו, $ALU\,R_3,R_2,R_1$, לבין המרחק בין הפקודה האחרונה שמשתמשת באוגר i הנחוץ לנו. נגדיר N_1 מספר ה $Stalls$ שיש להוסיף כאשר פקודת ה ALU משתמשת באוגר יחיד שמחושב ע"י פקודות קודמות, ונגדיר N_2 מספר ה $Stalls$ שיש להוסיף כאשר פקודת ה ALU משתמשת משני אוגרים שמחושבים ע"י פקודות קודמות.
על פי הנתונים, הרי ש

$$P\{DH_k = i\} = \begin{cases} 0.15, & i=1 \\ 0.10, & i=2 \\ 0.05, & i=3 \\ 0.70, & i\geq 4 \end{cases}$$

מכיוון שמחזור ה WB יכול להתבצע במקביל למחזור ה ID של פקודה אחרת, מספר ה $Stalls$ המרבי שנזדק לו הוא 2. עבור אופרנד בעייתי יחיד נקבל

$$P\{N_1 = n\} = \begin{cases} P\{DH_2 \geq 3\}, & n=0 \\ P\{DH_1 = 2\}, & n=1 \\ P\{DH_1 = 1\}, & n=2 \\ 0, & n\geq 3 \end{cases} = \begin{cases} 0.75, & n=0 \\ 0.10, & n=1 \\ 0.15, & n=2 \\ 0, & n\geq 3 \end{cases}$$

עבור שני אופרנדים בעייתים שונים, מחוסר התלות בין מאורעות $Data\,Hazard$ בין אוגרים שונים, נקבל

$$P\{N_2 = n\} = \begin{cases} 1-P\{1\leq N_2\leq 2\}, & n=0 \\ 2\big(P\{DH_1=2\}\cdot P\{DH_2>2\}\big), & n=1 \\ 2\big(P\{DH_1=1\}\cdot P\{DH_2>1\}\big), & n=2 \\ 0, & n=3 \end{cases} = \begin{cases} 1-0.405, & n=0 \\ 2\cdot 0.10\cdot 0.75, & n=1 \\ 2\cdot 0.15\cdot 0.85, & n=2 \\ 0, & n=3 \end{cases} = \begin{cases} 0.595 & n=0 \\ 0.15, & n=1 \\ 0.255, & n=2 \\ 0, & n=3 \end{cases}$$

(2) בשימוש בסעיף הקודם, נקבל

$$CPI_{extra} = 55\%\cdot\big(20\%\cdot N_1 + 80\%\cdot N_2\big)$$

ובממוצע נקבל

$$E[CPI_{extra}] = \frac{11}{20}\cdot\left(\frac{1}{5}E[N_1] + \frac{4}{5}E[N_2]\right) = \frac{11}{20}\cdot\left(\frac{1}{5}0.4 + \frac{4}{5}0.66\right) = 0.3344$$

(3) עם כל הקידומים האפשריים, הבעייה היחידה שיכולה לגרום הוספת מחזור $Stall$ יחיד הינה פקודת ALU שבאה מייד לאחר פקודת $Load$ שטוענת נתון לאחד מאוגרי המקור של פקודת ה ALU. במצב זה נקבל:

$$CPI_{extra} = 55\% \cdot 40\% \cdot 1 = 0.22$$

פתרון לשאלה 2

סעיף א

התוכנית תרוץ באופן שגוי, בגלל ארכיטקטורת ה $Delayed\ Brach$. השגיאה תהיה ההרצה התמידית של פקודה 7 לאחר פקודת ה $Branch$. נוכל לתקן זאת ע"י העברה של פקודה 5 להיות אחרי פקודה 6.

סעיף ב

ב 50% מהמקרים, ישנה פקודה ב $Delayed\ Slot$ שאינה $no-op$ לאחר פקודת הקפיצה המותנית, לכן נקבל

$$BP_{cond} = 2 + 50\% \cdot 1 = 2.5$$

עבור קפיצות לא מותנות, תמיד נשלם את 3 המחזורים הלוקחים לחשב את כתובת הקפיצה, כלומק

$$BP_{uncond} = 3$$

ואז נקבל

$$CPI = CPI_{Ideal} + 25\% \cdot 80\% \cdot BP_{cond} + 5\% \cdot BP_{uncond} = 1.65$$

סעיף ג

(1) קידום זה אפשרי. יש לקחת את הנתון שנקרא מהזיכרון, או חושב ע"י ה ALU ולהעבירו לזכרון:

$Load\ R1, 0(R2)$

$Sub\ R5, R6, R4$

$Store\ R3, 0(R1)$

(2) קידום זה אפשרי רק אם יש להעביר תוצאות חישוב, כלומר נתון מהזכרון לא יוכל יהיה זמין בשביל הקידום:

$Add\ R1, R2, R2$

$Sub\ R5, R6, R4$

$Add\ R3, R1, R1$

(3) קידום זה אפשרי:

$Load\ R1, 0(R2)$

$Sub\ R5, R6, R4$

$Add\ R3, R1, R1$

סעיף ד

הבעייתיות היא בין פקודה 2 ל 3 – שום יחידת קידום לא תעזור שם, ובנוסף הקפיצות שבד"כ נלקחות:

1, 2, stall, stall, 3, 4, 5, stall, stall, 2, stall, stall, 3, 4, 5, stall, stall, 2, stall, stall, 3, 4, 5, 6, 7

סה"כ 25 מחזורים. למילוי הצינור יש להמתין עוד 6 מחזורים, ונקבל סה"כ 31 מחזורי שעון.

פתרון לשאלה 3

סעיף א

#	IF	ISSUE	RegRead	EX	MEM	WB	Remark
1	1 2						**U pipe** **V pipe**
2	3 4	1 2					
3	5 6	3 4	1 2				
4	7 8	5 6	4 3	1 2			פקודה 4 צריכה אופרנד מפקודה 1 פקודה 3 צריכה אופרנד מפקודה 2
5	9 10	7 8	5 6	4 3	1 2		
6	9 10	7 8	5 6	4 3	stall stall	1 2	פקודות 3 ו4 מחכות לסיום 1 ו2 יש forwarding של תוצאות 1 ו2 ל3 ו4
7	10 11	8 9	7 6	5 stall		4 3	פקודה 6 צריכה תוצאה של 5
8	11 12	9 10	8 6	7 stall		5	
9	12 13	10 11	9 6	8 stall		7	פקודה 7 מחכה לסיום של 6 פקודה 5 עידכנה רגיסטרים של V, פקודה 6 תמשיך במחזור הבא
10	14 15	12 13	10 11	9 6		7,8	פקודות 7 ו8 מחכות לסיום של 6 Forwarding של תוצאת חישוב 9 לפקודה 10
11	16 17	14 15	12 13	10 11		7,8,9 6	חישוב תוצאת קפיצה. פקודות 7,8,9 מחכות לסיום 6
12	1 2					8,9	Flash של הpipes, קריאה של פקודות 1 ו2
13	3 4	1 2					
14	5 6	3 4	1 2				
15	7 8	5 6	4 3	1 2			פקודה 4 צריכה אופרנד מפקודה 1 פקודה 3 צריכה אופרנד מפקודה 2
16	9 10	7 8	5 6	4 3	1 2		
17	9 10	7 8	5 6	4 3	stall stall	1 2	פקודות 3 ו4 מחכות לסיום 1 ו2 של תוצאות 1 ו2 ל3 ו4forwardingיש

<u>סעיף ב</u>

למרות שאנו מבצעים retire של פקודות ע"פ סדר הגעתן, עדכון הזכרון לאו דווקא נעשה ע"פ הסדר הזה. הסיבה היא שאנו מסדרים את הפקודות בשלב הWB, כאשר חלק מהפקודות עדכנו את הזכרון אחרי שסיימו את שלב הMEM וחלק אחרי שלב הWB.

לדוגמא:

1	Ld r1,0(r11)	Load command
2	Add r2,r2,r2	Add command (doesn't need results of 1 as operands)
3	Add r3,r1,r1	Add command (needs as operand the result of 1)
4	St 0(r12),r13	Store command

ניתן לראות שפעולה 4 תגיע לשלב הMEM לפני שפעולה 3 תגיע לשלב הWB, משום שפעולה 3 תהיה בstall עד שתקבל את האופרנד r1. ובמצב כזה, פעולה 4 תכתוב לזיכרון לפני פעולה 3, מה שנוגד לסדר הפעולות בתוכנית.

הצעות לפיתרון:

- לאפשר לפקודות לכתוב לזיכרון רק בשלב יחיד – WB.
- כבר קיים מנגנון שמבצע retire בשלב הWB ע"פ הסדר.

<u>סעיף ג</u>

- ב Register File חלוקת ה ports היא כלהלן:
  2 קריאת פקודות, 1 כתיבת תוצאה, 2 – קשר בין 2 הclusters.
  סה"כ – 5
- במקרה ש num. Clusters = N, מספר הports יהיה:
  2 קריאת פקודות, 1 כתיבת תוצאה, 2*(N-1) קשר בין N הclusters.
  סה"כ – 2N+1
- ניתן לצמצם את מספר ה ports, אם נקדיש לכל cluster רק סיגנל עידכון אחד, ולא 2, לפי ההצעה המקורית. הדבר ידרוש בקרה, שתמנע כתיבה של שני הclusters המחוברים בו זמנית.

©2006 אבי בנדל www.technion.co.il

פתרון לשאלה 4

סעיף א

.1

	ALU 1	ALU 2	MEM 1	MEM2	FADD	FMUL
1			`ld f1, 0(r1)`	`ld f2, 0(r2)`		
2	`add r1, r1, 4`	`add r2, r2, 4`	`ld f3, 0(r3)`			
3	`add r3, r3, 4`	`sub r4, r4, 1`				
4						`fmul f4, f2, f1`
5						
6						
7						
8			`st f4, 0(r1)`		`fadd f5, f4, f3`	
9						
10						
11						
12	`bnez r4, loop`		`st f5, 0(r3)`			

.2

איטרציה במצב יציב לוקחת 17 מחזורים (הפקודה האחרונה באיטרציה מופקת במחזור 12 ומסתיימת ב16).

.3

מתוך 16 מחזורי האיטרציה, הALU מבצע 2 חישובי נקודה צפה.

לכן מדד הflops/cycle שווה ל2/16 = 0.125

סעיף ב

נממש את הקוד הבא:

1	`st f5, 0(r3)`	`Store C(i-1)(from F5)`
2	`fadd f5, f4, f3`	C(i) = A(i)+C(i)
3	`fmul f4, f2, f1`	A(i+1)=A(i+1)*B(i+1)
4	`st f4, 8(r1)`	Store A(i+1) (from F4)
5	`ld f1, 12(r1)`	F1 = a(i+2)
6	`ld f2, 12(r2)`	F2 = b(i+2)
7	`ld f3, 12(r3)`	F3 = c(i+1)
8	`add r1, r1, 4`	Inc. a pointer
9	`add r2, r2, 4`	Inc. b pointer
10	`add r3, r3, 4`	Inc. c pointer
11	`sub r4, r4, 1`	i--
12	`bnez r4, loop`	Branch cmd.

	ALU 1	ALU 2	MEM 1	MEM2	FADD	FMUL
1			`st f5, 0(r3)`		`fadd f5, f4, f3`	`fmul f4, f2, f1`
2		`sub r4, r4, 1`	`ld f1, 12(r1)`	`ld f2, 12(r2)`		
3	`add r1, r1, 4`	`add r2, r2, 4`	`ld f3, 12(r3)`			
4	`add r3, r3, 4`					
5	`bnez r4, loop`		`st f4, 8(r1)`			

1. יש באיטרציה כרגע 5 פקודות.
2. אנו מבצעים בכל רגע חלקים של 4 איטרציות: i-1, i, i+1, i+2 (הסימון של האינדקסים הפוך, כי הcounter יורד מערך התחלתי ל0).
3. האיטרציה מסתיימת לאחר 8 מחזורים הALU מבצע 2 חישובי נקודה צפה.
   לכן מדד הflops/cycle שווה 2/8 = 0.25.

סעיף ג

מספר הפעמים המינימלי שצריך לבצע loop unrolling הוא 4. (256 איטרציות הופכות ל64 איטרציות).

הסבר:

נוכל לנצל את העובדה שפעולות נקודה צפה מבוצעות בצורה מצונרת במשך 4 מחזורים.

לכן, כדי לנצל בצורה מקסימלית את החומרה של FADD וFMUL צריך שיבוצעו 4 פעולות כפל ו4 פעולות חיבור באיטרציה (מה שיקרה עם נאחד 4 איטרציות מקוריות).

משום שאנו יכולים לבצע במקביל 2 פעולות MEM וInteger ALU, אז חישוב 4 איטרציות במקביל יתן לנו את האופציה למלא את ה nopים הרבים שהיינו מוכרחים להשתמש בהם כאשר לא ביצענו loop unrolling או Software pipeline.

מספר פקודות נקודה צפה שנסיים הוא 8 (4 חיבור ו4 כפל).

מספר הפקודות הכללי יהיה בערך כפול ממס' הפקודות בסעיף ב', ואם נניח שלא יהיו בה Nopים, היא תסתיים אחרי 13 מחזורים (5 פקודות * 2 + 3 מחזורים לריקון הpipe).

לכן, ניתן להעריך את מדד הflops/cycle ב8/13 = 0.61

סעיף ד

הקונפיגורציה החדשה לא תוכל לעזור לקומפיילר לשפר את ביצועי התוכנית. הסיבה לכך היא שאי אפשר לדעת מראש מתי יהיה hit בcache, ומתי לא. לכן חייבים לסדר את הקוד כדי שיוכל להתמודד עם המקרה הגרוע ביותר, ועדיין יהיה חוקי – מה שמשאיר אותנו בדיוק אם אותו הקוד. זו בעיה כללית למעבד, ולאו דווקא לתוכנית אותה אנו בודקים בשאלה.

בנוסף, לא יכול להיות שיפור בזמן הריצה, כי אין שום מנגנון interlock, והמעבד סומך על הקומפיילר שיסדר את הקוד כך שלא יווצרו hazards.

©2006 אבי בנדל www.technion.co.il