ארגון המחשב ומערכות הפעלה

אביב תשפ"ד

תרגול 13 – סיכום

תרגיל 1

ווה ו-ווי הם אמא ובן יפנים אשר גרים בלב יער יפני מסורתי.

הם מקבלים מעמי ותמי אספקת ממתקים, ואוכלים מהם כל אחד בתורו.

מכיוון ש-ווה בהיריון (עם תאומים), היא תאכל את שני הממתקים הראשונים בכל סיבוב. אחרי זה יהיה תורו של ווי לאכול ממתק, ולבסוף ווה תקבל ממתק נוסף (הפעם עבור עצמה), וזה יסגור סיבוב.

תרגיל 1 – סעיף א'

בהנחה כי בסוף כל סיבוב ווה שולחת את ווי לישון, והסיבוב הבא יתחיל עם בוקר יום חדש – תזמנו את תהליכי צריכת הממתקים של ווה ו-ווי **ללא שימוש במערכת ההפעלה**.

דמו אכילת ממתק באמצעות הדפסת שמו של מי שאכל את הממתק, ובאמצעות ירידת שורה את הפעולה של שינה למשך לילה

תרגיל 1 – סעיף א' : פתרון

ווי:

```
While (1){
        if (cnt == 0){
                printf("Wi");
                cnt = 3;
        }
}
```

ווה:

```
While (1){
        if (cnt != 0){
                printf("Wa");
                if (cnt ==3)
                        printf("\n");
                cnt--;
        }
}
```

נאתחל משתנה:

Cnt = 2

פלט:

WaWaWiWa

WaWaWiWa

…

WaWaWiWa

תרגיל 1 – סעיף א' : פתרון

ווה יכולה להיכנס לקטע הקריטי רק אם הערך של cnt הוא לא 0, בעוד ש ווי יכול להיכנס לקטע הקריטי רק אם הערך שלו הוא 0.

מניעה הדדית - שני התהליכים מעדכנים את הערך של cnt בסוף הקטע הקריטי שלהם. זה אומר שמיד לאחר עדכון הערך אף תהליך לא נמצא בתוך הקטע הקריטי, והבא שייכנס אליו יהיה זה שיעמוד בתנאי. כמו שכבר אמרנו – עבור כל ערך של cnt, יש בדיוק אחד שיכול לעבור את התנאי.

אין כאן חבק - על כל ערך של cnt יש תהליך כלשהו שיכול לעבור את תנאי הכניסה לקטע הקריטי.

אין כאן הרעבה – ווה בכל כניסה לקטע הקריטי מורידה את הערך שלcnt ב-1. זה אומר שמתישהו הוא יהיה 0 ויאפשר ל-ווי להיכנס לקטע הקריטי שלו. מכיוון ש-ווי הוא היחיד שמעלה את הערך של cnt והוא עושה את זה בכל פעם לערך 3, אז בוודאי שהתור יחזור אליו בתוך זמן סופי.

תרגיל 1 – סעיף ב'

בהנחה כי סיבוב מתחיל בכל פעם שעמי ותמי מגיעים, תזמנו את תהליכי צריכת הממתקים של ווה ו-ווי באמצעות סמפורים.

הנחות:

1. בכל פעם שעמי ותמי מגיעים הם מביאים 4 ממתקים בדיוק
2. בכל פעם שעמי ותמי מגיעים, הם מבצעים פעולת signal לסמפור ששמו sem1

תרגיל 1 – סעיף ב' : פתרון

```
Sem sem1 = s_create(0);
Sem sem2 = s_create(0);
Sem sem3 = s_create(0);
```

ווי:

```
while (1){
        wait(sem2);
        printf(“Wi”);
        signal(sem3);
}
```

ווה:

```
while (1){
        wait(sem1);
        printf(“Wa”);
        printf(“Wa”);
        signal(sem2);
        wait(sem3);
        printf(“Wa\n”);
}
```

תרגיל 2

נתונה המטריצה: float M[8][8]

ובנוסף, נתון הקוד הבא:

```
float fsum(float M[][8]){
  int i,j;
  float sum = 0;
  for (i=0; i<8; i++){
        for (j=0; j<8; j++){
              sum += M[i][j]; //(*)
        }
  }
  return (sum);
}
```

תרגיל 2 – סעיף א'

חשבו את מדד ה- CPE של הפונקציה fsum.

הניחו כי המעבד עליו רצה הפונקציה הינו

פנטיום 3, וחוץ מן המטריצה כל המשתנים

שמורים על הרגיסטרים.

התעלמו ממימדי המטריצה הקבועים,

וחשבו עבור מימד n כלשהו.

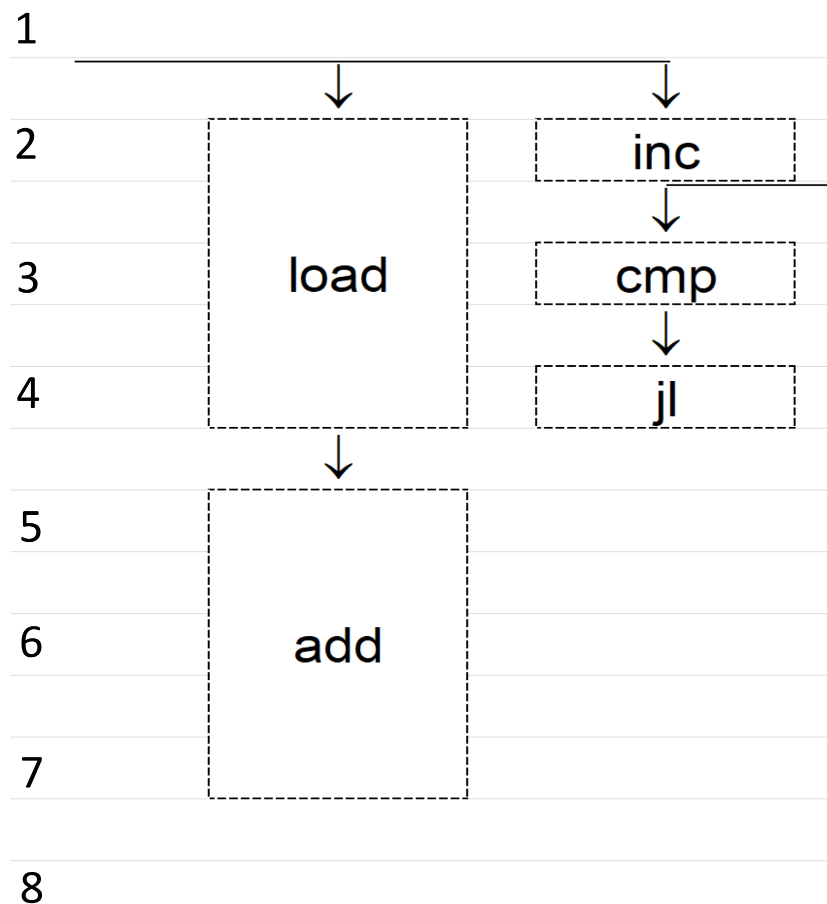
האם ניתן לשפר אותו?

```
float fsum(float M[][8]){
   int i,j;
   float sum = 0;
   for (i=0; i<8; i++){
          for (j=0; j<8; j++){
                    sum += M[i][j];
          }
   }
   return (sum);
}
```

ניזכר בטבלה

instruction	Latency	Cycles / Issue
Load / Store	3	1
Integer Add / Subtruct	1	1
Integer Multiply	4	1
Integer Division	36	36
Double/Single FP Add	3	1

תרגיל 2 – סעיף א' : פתרון

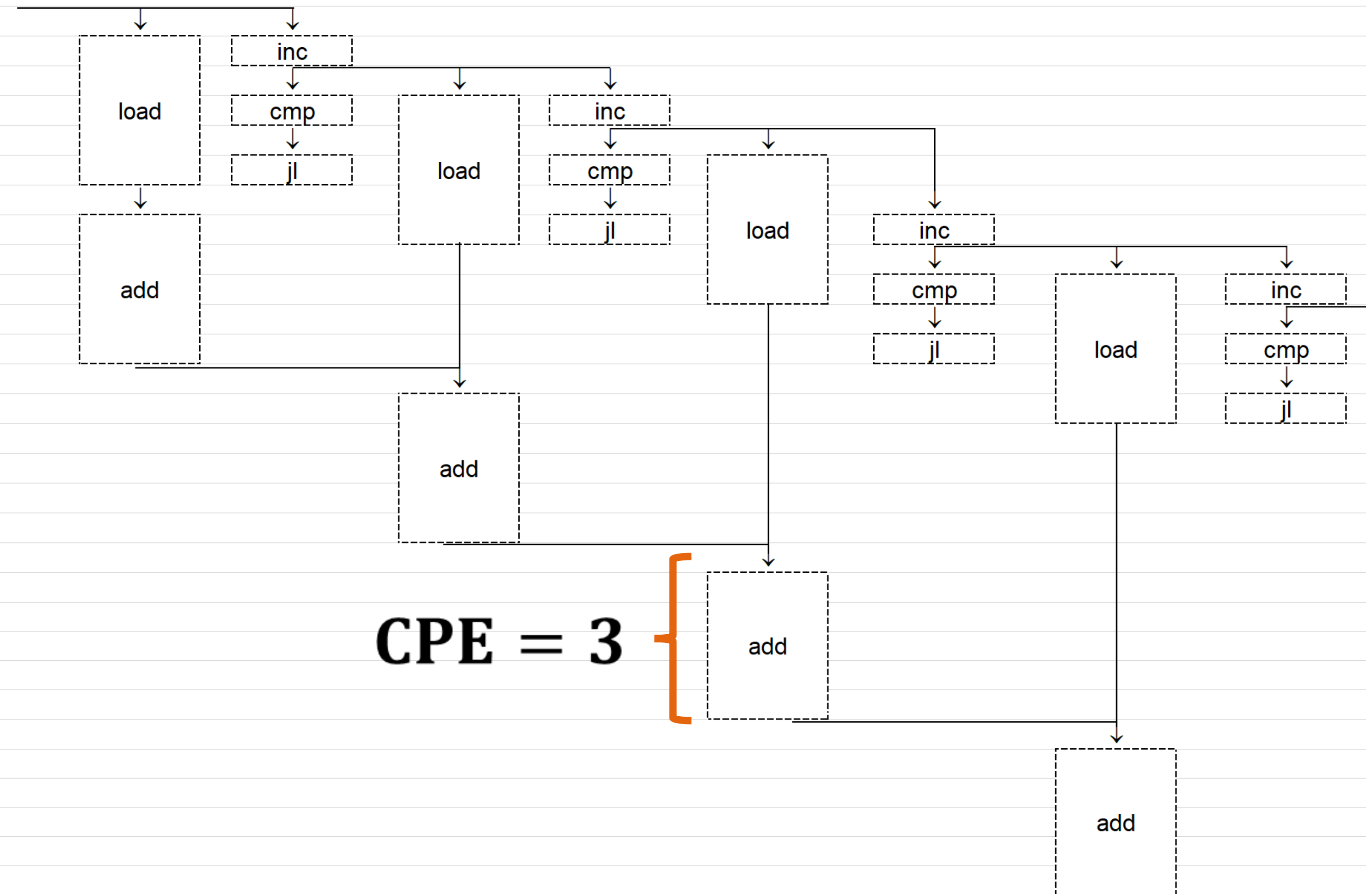



```
float fsum(float M[][8]){
   int i,j;
   float sum = 0;
   for (i=0; i<8; i++){
            for (j=0; j<8; j++){
                        sum += M[i][j];
            }
   }
   return (sum);
}
```

(*) נזכור שבמקרה הזה פעולת add היא על float ולכן היא מעסיקה את הרכיב של FP addition (ולא int)

תרגיל 2 – סעיף א' : פתרון

```
float fsum(float M[][8]){
  int i,j;
  float sum = 0;
  for (i=0; i<8; i++){
        for (j=0; j<8; j++){
              sum += M[i][j];
        }
  }
  return (sum);
}
```



תרגיל 2 – סעיף א' : פתרון

```
float fsum(float M[][8]){
  int i,j;
  float sum = 0;
  for (i=0; i<8; i++){
        for (j=0; j<8; j++){
              sum += M[i][j];
        }
  }
  return (sum);
}
```

כעת ננסה לשפר את ה CPE. איך?

תרגיל 2 – סעיף א' : פתרון

```
float fsum(float M[][8]){
  int i,j;
  float sum1 = 0;
  float sum2 = 0;
  float sum3 = 0;
  for (i=0; i<8; i++){
        for (j=0; j<8; j++){
              sum1 += M[i][j];
              sum2 += M[i][j+1];
              sum3 += M[i][j+2];
        }
  }
  return (sum1 + sum2 + sum3);
}
```

כעת ננסה לשפר את ה CPE.

נבצע loop unrolling ונפרוס על פני 3 אלמנטים.

במובן שצריך לעשות מעט התאמות נוספות לקוד
כאשר פורסים את הלולאה, כמו למשל לדאוג
שעוברים על כל השורה בלי גלישות גם כאשר
הגודל שלה לא כפולה של 3, אבל כמו שתראו – הן
לא הולכות להשפיע על ה- CPE

תרגיל 2 – סעיף א' : פתרון

load
add
load
add
load
add
add
cmp
jl
load
add
load
add
load
add
add
cmp
jl

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

תרגיל 2 – סעיף ב'

מריצים את הפקודה: `float f = fsum(M);`. נתון:

cache בגודל 64 בתים.

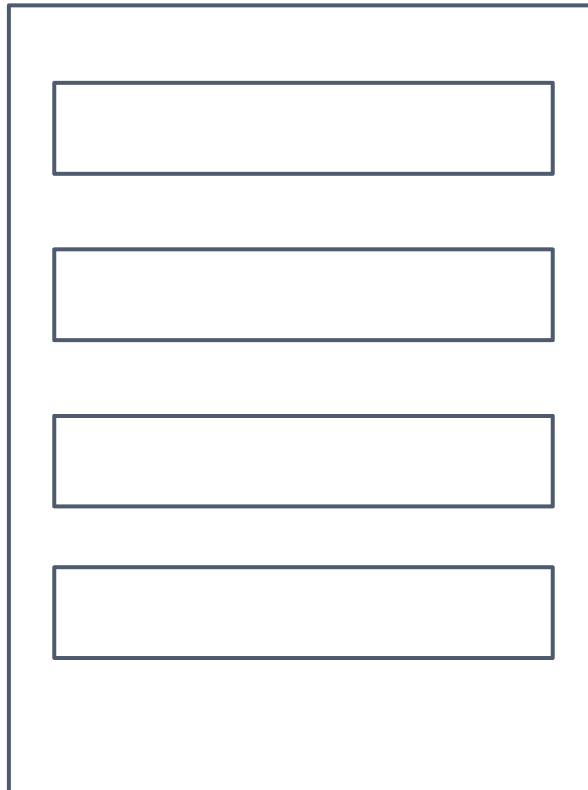
גודל כל בלוק הוא 16 בתים.

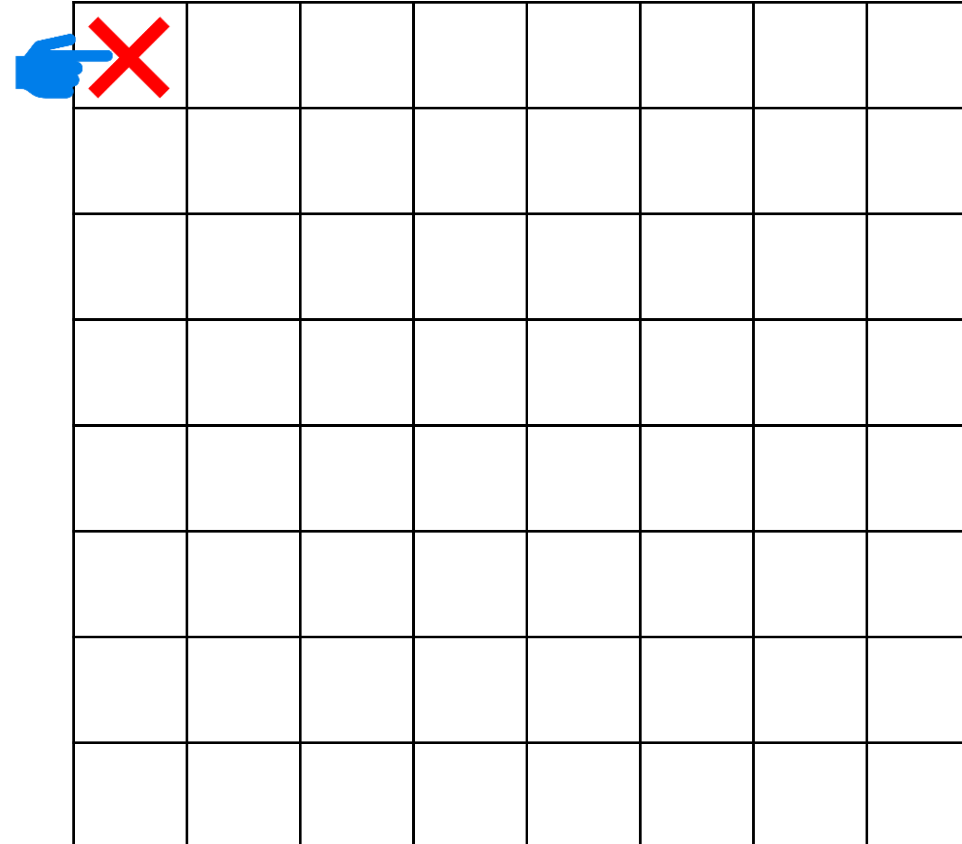
cold cache בתחילת ריצת הפונקציה.

1) חשבו את ה- miss rate של הפונקציה.

תרגיל 2 – סעיף ב' 1 : פתרון

- בכל בלוק יש מקום ל-4 ערכי משתנים מטיפוס float.
- ב-cache יש מקום ל-4 בלוקים.

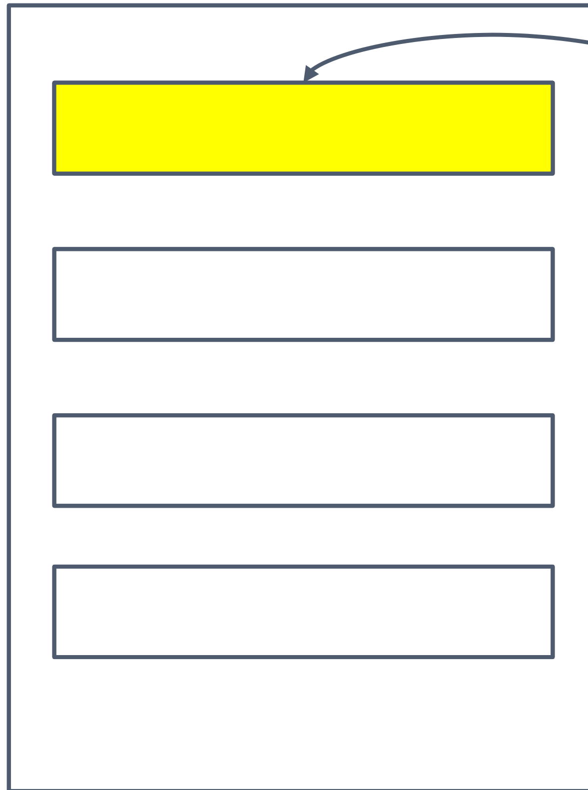




תרגיל 2 – סעיף ב' 1 : פתרון

- בכל בלוק יש מקום ל-4 ערכי משתנים מטיפוס float.
- ב-cache יש מקום ל-4 בלוקים.

cache

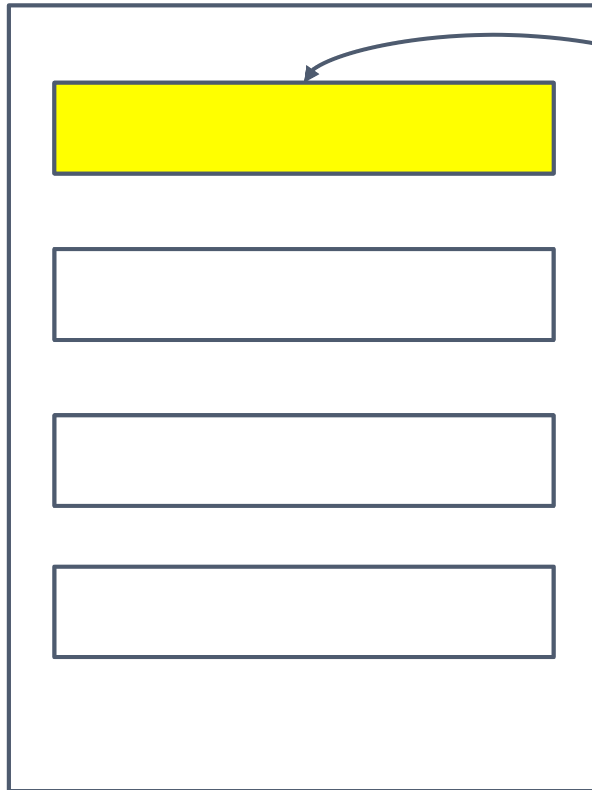


M

תרגיל 2 – סעיף ב' 1 : פתרון

- בכל בלוק יש מקום ל-4 ערכי משתנים מטיפוס float.
- ב-cache יש מקום ל-4 בלוקים.

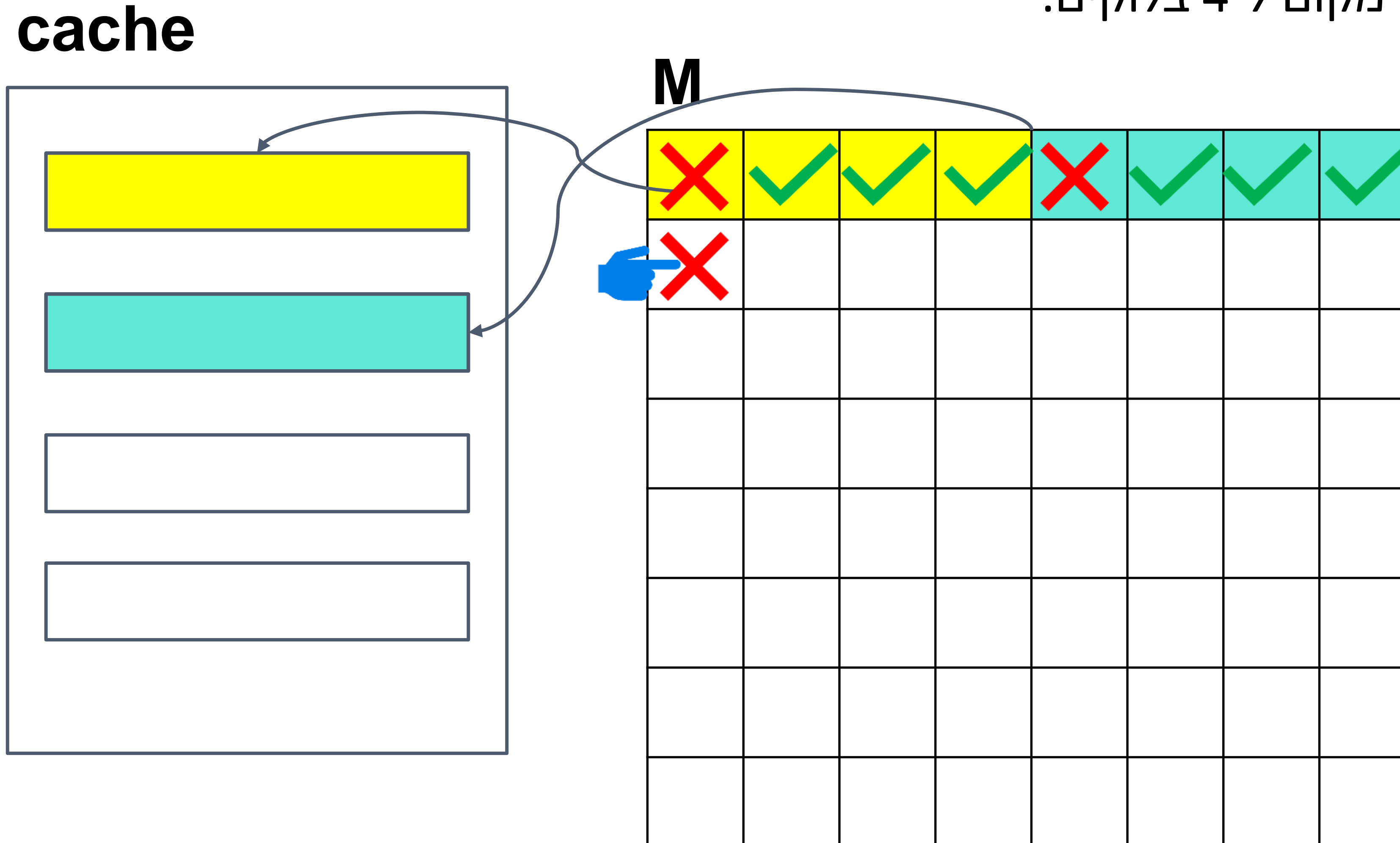
cache



M

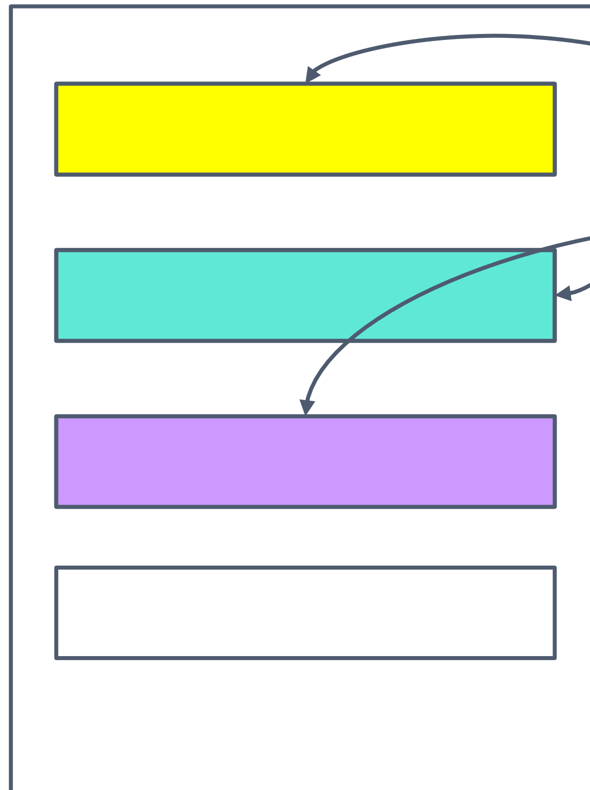
תרגיל 2 – סעיף ב' 1 : פתרון

- בכל בלוק יש מקום ל-4 ערכי משתנים מטיפוס float.
- ב-cache יש מקום ל-4 בלוקים.



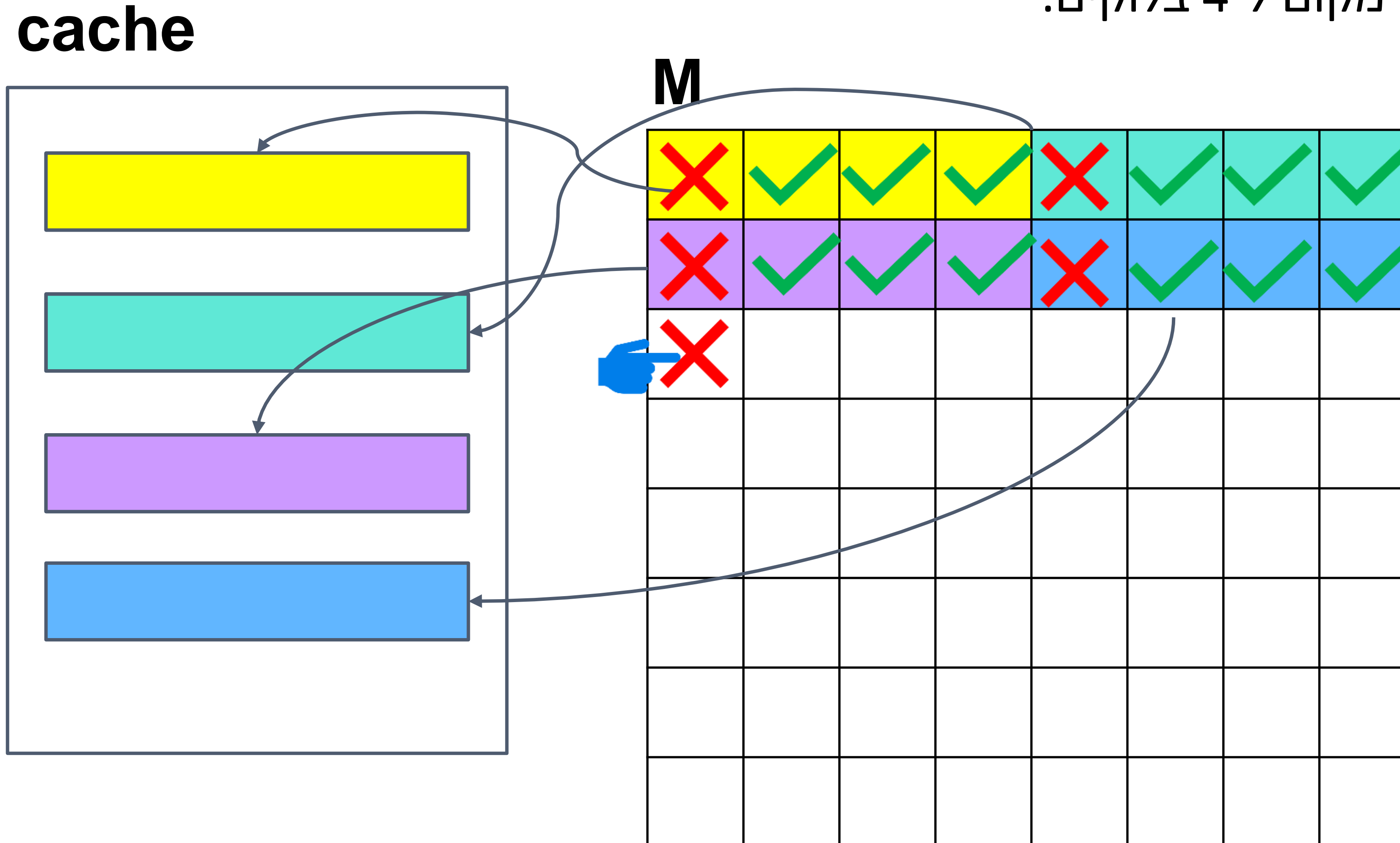
תרגיל 2 – סעיף ב' 1 : פתרון

- בכל בלוק יש מקום ל-4 ערכי משתנים מטיפוס float.
- ב-cache יש מקום ל-4 בלוקים.



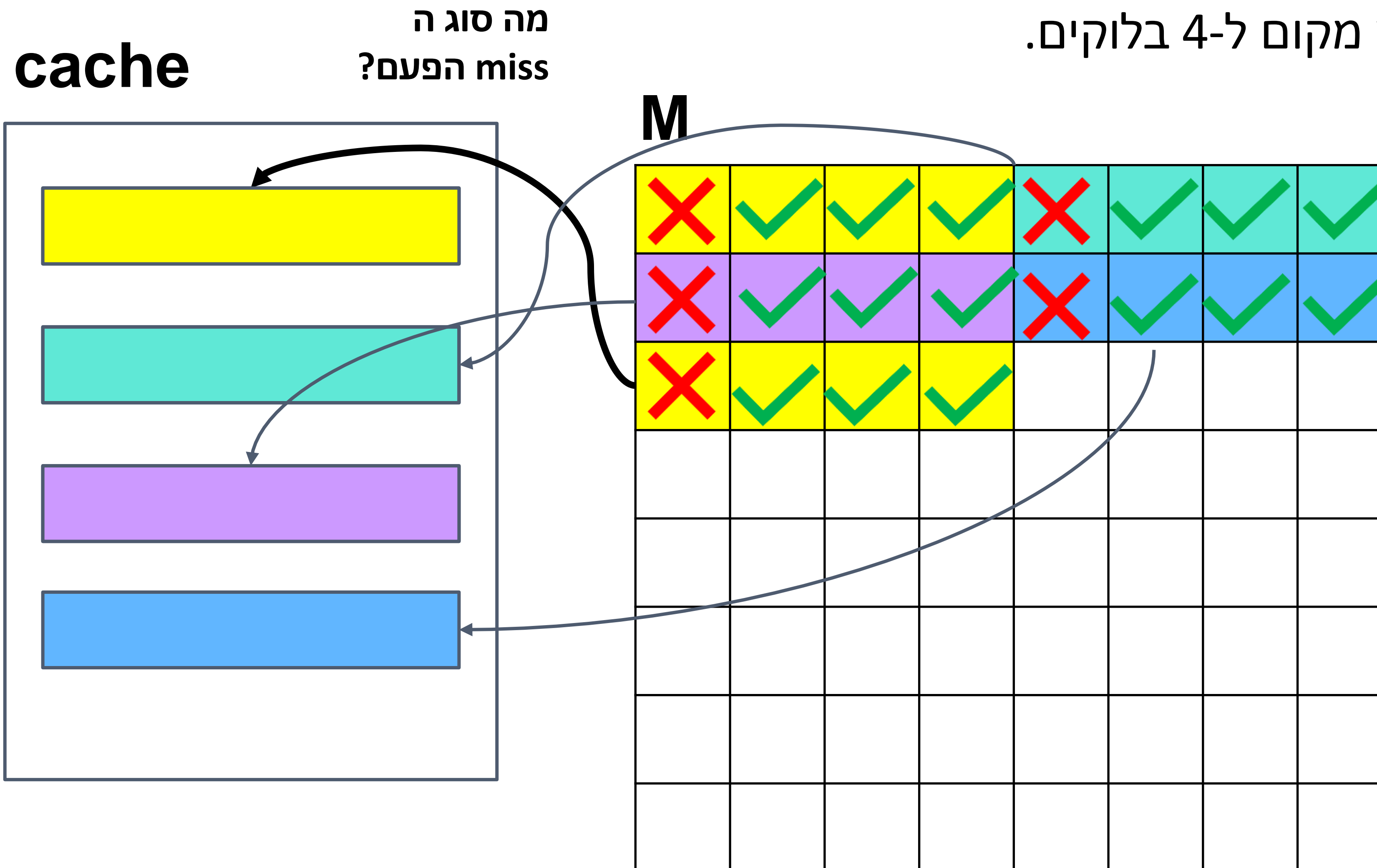
תרגיל 2 – סעיף ב' 1 : פתרון

- בכל בלוק יש מקום ל-4 ערכי משתנים מטיפוס float.
- ב-cache יש מקום ל-4 בלוקים.



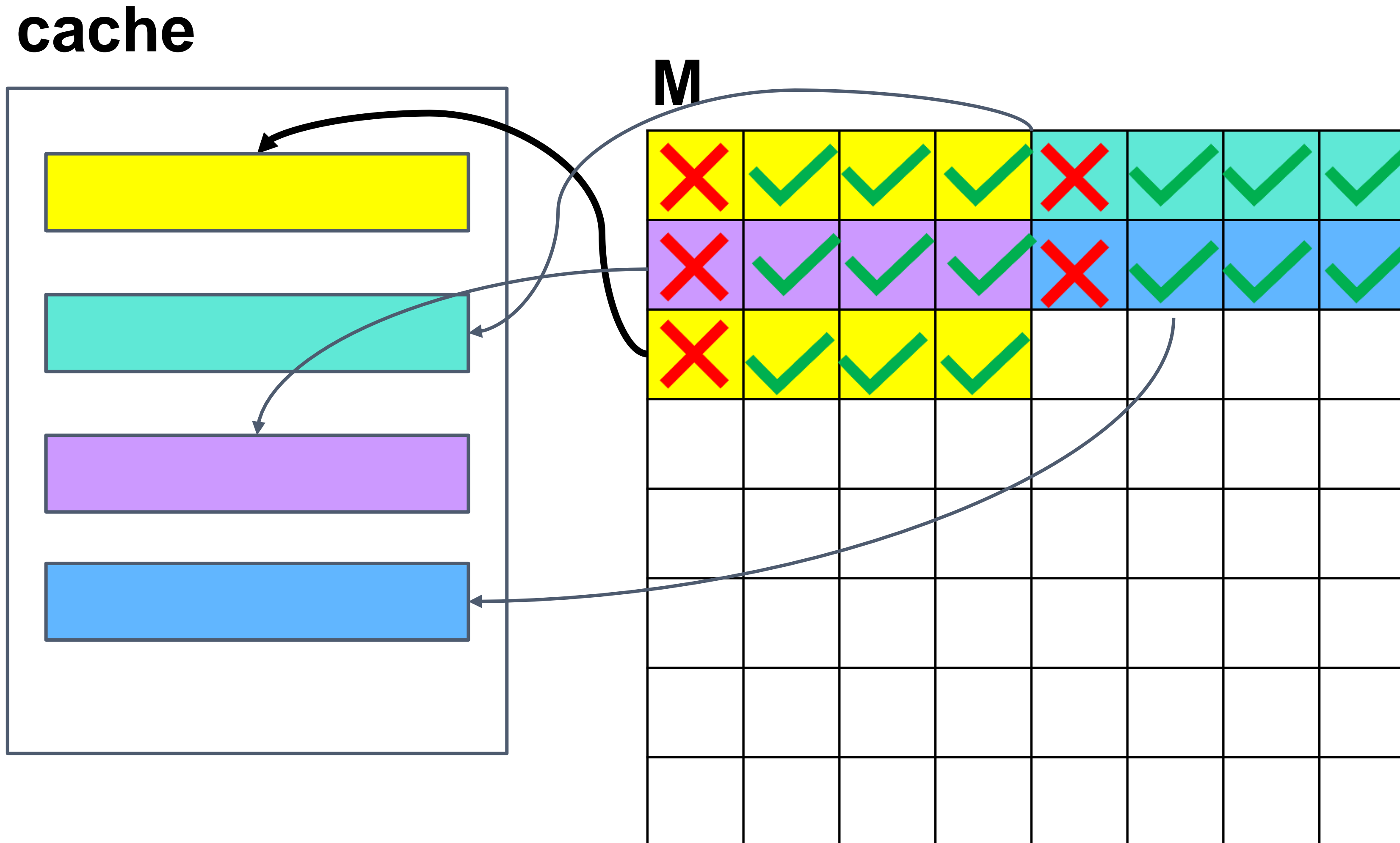
תרגיל 2 – סעיף ב' 1 : פתרון

- בכל בלוק יש מקום ל-4 ערכי משתנים מטיפוס float.
- ב-cache יש מקום ל-4 בלוקים.



תרגיל 2 – סעיף ב' 1 : פתרון

תשובה: $miss\ rate = \frac{1}{4}$



תרגיל 2 – סעיף ב'

בעת נשנה את

הקוד:

```
float fsum(float M[][8]){
  int i,j;
  float sum = 0;
  for (i=0; i<8; i++){
        for (j=0; j<8; j++){
              sum += M[i][j];
        }
  }
  return (sum);
}
```

```
float fsum(float M[][8]){
  int i,j;
  float sum = 0;
  for (i=0; i<8; i++){
        for (j=0; j<8; j++){
              sum += M[j][i];
        }
  }
  return (sum);
}
```

תרגיל 2 – סעיף ב'

2) מה צריך להיות גודל ה- cache המינימלי כדי שה- miss rate לא יעלה?

האם הקטנת גודל הבלוק תשיג את המטרה הזאת?

תרגיל 2 – סעיף ב' 2 - פתרון

- מבלי לשנות את גודל ה cache, ה miss rate הוא 1.
- כדי שנוכל לקבל cache hit כאשר חוזרים לאיבר שכבר הובא בעבר אל ה cacher בחלק מבלוק שהעלנו, צריך להיות מקום ב cache לבלוקים לפחות במספר השורות (למה?)
- לכן, גודל ה cache המינימלי שלא יפגע לנו ב miss rate יהיה 8*16 = 128 bytes.
- הקטנת מספר הבלוקים אמנם יכולה לשפר את ה miss rate, אבל לא תשיג את המטרה

תרגיל 2 – סעיף ג'

נתונים:

- Float M[8][8] = {4}; M[7][7] = 4.5;
- float f = fsum(M); int d = fsum(M);
- char c = fsum(M); unsigned u = fsum(M);

עבור כל אחד מההיגדים הבאים, קבעו האם הוא נכון או לא:

- u == f
- u == d
- *(int *)(&f) == d
- c == d
- c <= u
- (c – 1) <= u

תרגיל 2 – סעיף ג' : פתרון

- u == f   **לא נכון**.

בהמרה מ-float לטיפוס של ערכים שלמים (כמו unsigned), מתבצע עיגול הערך כלפי מטה.

- u == d   **נכון**.

נובע ממה שרשום בשורה הקודמת.

בשני האגפים יש טיפוסים שלמים שהוכנס אליהם ערך חיובי שעבר את אותה ההמרה.

- *(int *)(&f) == d   **לא נכון**.

קוראים מהכתובת של f ערך int (בעצם אומרים לקומפיילר לגשת לכתובת מסוימת כאילו היא כתובת ל-int – לקרוא 4 בתים ולפרש אותם לפי 2's complement.

בהמרה (casting) מ-float ל-int יכול (ובמעט תמיד זה קורה) ממש להשתנות הערך הביטי!

(כי הרי 256.0 לפי floating point זה ממש לא זהה ל-256 לפי 2's complement.

תרגיל 2 – סעיף ג' : פתרון

- c == d **לא נכון**.
256 לא נמצא בטווח של char (ולכן הערך שיהיה בו הוא 0), אבל כן בטווח של int (ולכן זה הערך שיש בו). בהמרה של הערך ב-c ל-int (שתתרחש לצורך ביצוע פעולת ההשוואה), תהיה הרחבה מ-8 ל-32 ביטים, כאשר הביטים ה"חדשים" יהיו שכפול של הביט השמאלי, ולכן במקרה הזה הערך יישאר 0.
- c <= u **נכון**.
גם כאן ההרחבה של הערך של c ל-32 ביטים תתבצע באותו אופן כמו בשורה הקודמת, והוא יישאר 0 (שבכל מקרה זהו ה-unsigned הקטן ביותר).
- (c – 1) <= u **לא נכון**.
אם הערך של c הוא 0, אז (c-1) הוא 1...11 בבינארי (בביטים). הערך הזה צריך לעבור המרה ל-unsigned, וכחלק ממנה הוא צריך להתרחב מ-8 ל-32 ביטים. מכיוון ש-char הוא טיפוס signed (להבדיל מ-unsigned char), אז בהרחבה ישוכפל הביט השמאלי, ובעצם יתקבל כאן ה-unsigned הגדול ביותר. אנחנו כבר יודעים שהערך של u הוא 256.