המכללה האקדמית של תל-אביב-יפו
121113 – ארגון המחשב ושפת-סף
קיץ תשס"ב

מבחן סופי – מועד א' - פתרון

תאריך הבחינה: 9 באוקטובר 2002.

משך הבחינה: 3 שעות.

חומר עזר מותר: מצורפים דפי עזר. מותר שימוש במחשבונים (אך לא במחשב אישי).

מרצה: אליאב גנסין

ניקוד: המבחן מחולק לשני חלקים
חלק א' – 4 שאלות חובה, כל אחת 10 נקודות – סה"כ 40 נקודות
חלק ב' – שאלת חובה, 3 סעיפים – סה"כ 60 נקודות

הנחיות: לשאלות 1-4, ו 5א' יש לענות על דף הבחינה. לשאלות 5ב' ו 5ג' יש לענות במחברת הבחינה.

במבחן זה 9 עמודים, כולל דף זה, לא כולל דפי העזר.

בהצלחה !

תעודת זהות/מס' סטודנט: _____

<u>חלק א'</u>

1. (10 נקודות)

נתון כי ערכו של רגיסטר DS הוא 1100h.

נתונות השורות הבאות מתוך תכנית אסמבלר:

```
.data
VAR1=12
VAR2 DW 4 DUP(?,?),3
VAR3=1Fh
VAR4 DW VAR1,7
VAR5 DD VAR4
VAR6 DB VAR3
```

א. המשתנה VAR6 יימצא בכתובת פיזית 1 1 0 1 A h

ב. לאיזה ערך יאותחל המשתנה VAR6? (כתוב/כתבי בבסיס הקסדצימלי)

1Fh

ג. לאיזה ערך יאותחל המשתנה VAR5? (כתוב/כתבי בבסיס הקסדצימלי)

11000012h

2. (10 נקודות)

נתון קטע הקוד הבא:

```
.code

          mov   [mybyte], 5    ; /* (א) */
          mov   sp,0468h
          xor   ax,ax
here:     add   al,[mybyte]
          push  ax
          dec   BYTE PTR [mybyte]
          jnz   here
          pop   es
          nop
```

א. מלא/י את החסר בתכנית שלמעלה כך שכשנגיע לפקודה nop ערכו של sp יהיה 0460h (4 נק')

ב. מלא/י את הטבלה הבאה, כך שתייצג את תוכן הזיכרון כאשר אנו מגיעים לפקודה nop: (6 נק')
אם לא ידוע איזה ערך יש בתא זיכרון כלשהו, יש לרשום סימן שאלה (?)

Address	Contents (hex)
SS:045Eh	?
SS:045Fh	?
SS:0460h	0E
SS:0461h	00
SS:0462h	0C
SS:0463h	00
SS:0464h	09
SS:0465h	00
SS:0466h	05
SS:0467h	00

3. (10 נקודות)

מגנון הפסיקות ב 8086 מבוסס על וקטור פסיקות בעל תאים של 4 בתים, הנמצא בתחילת הזיכרון.

א. מדוע נדרשים 4 בתים עבור כל איבר בוקטור הפסיקות? (2 נק')

מאחר ובוקטור הפסיקות מאוחסנות כתובות FAR של הפונקציות, יש צורך לאחסן גם Offset וגם Segment ולכן נדרשים 16X2 ביט=32 ביט=4 בתים

ב. אחד ממהנדסי המערכת הציע לבטל את וקטור הפסיקות ולהחליף את פקודת ה int בפקודה חדשה בשם jmi – jump to interrupt, שקופצת באופן ישיר לפונקציה המטפלת בפסיקה, ואינה משתמשת בוקטור הפסיקות. ציין/צייני לפחות 2 חסרונות של רעיון זה. (4 נק')

1. אין יכולת להחליף פונקציה המטפלת בפסיקה כי בתכנית המשתמשת בפונקציה "צרובה" הכתובת שאליה קופצים.
2. מבנה זה מחייב כל מערכת הפעלה לשים את הפונקציות המטפלות בפסיקות באותן כתובות בדיוק.
3. לא ניתן לשרשר פסיקות בקלות.

ג. בקר ה PIC (Programmable Interrupt Controller) מטפל בפסיקות מהתקנים חיצוניים. הסבר/הסבירי כיצד מעביר בקר זה את מספר הפסיקה אל המעבד. (4 נק')

כאשר יש להעביר את מספר הפסיקה למעבד, הבקר מסמן למעבד באמצעות קו ייעודי (INT) כי ברצונו להעביר את המספר, המעבד מסמן בקו Ack כי מאושר לבקר לכתוב ל data bus, והבקר כותב את מספר הפסיקה ל data bus. המעבד קורא מספר זה, וכך יודע איזו פונקצית פסיקה להפעיל.

4. (10 נקודות)

נתון קטע התכנית הבא:

```
Mac1 MACRO
     REPT 8
          val=val+1
          ex=ex+2
          tp=tp*2
          Mac2 %ex
     ENDM
ENDM

Mac2 MACRO lc
     Tow&lc dd val
ENDM

.data
val=2
tp=2
ex=2
Mac1
```

איך נראית הפרישה של המאקרו הנ"ל ?

הערה: הכוונה היא כי תרשמו איך הייתה נראית הכרזה מקבילה של חלק ה data. ללא שימוש במאקרו.

אין להראות תמונת זיכרון.

```
Tow4 dd 3
Tow6 dd 4
Tow8 dd 5
Tow10 dd 6
Tow12 dd 7
Tow14 dd 8
Tow16 dd 9
Tow18 dd 10
```

<u>חלק ב'</u>

5. (60 נקודות)

להלן הכרזה של איבר בודד בעץ בינארי:

סעיפי שאלה 5 נוגעים לעץ בינארי בעל איברי בסיס אלו

```
T1    db 99          ; value
      dw T7          ; left
      dw -1          ; right (null)
```

א. (10 נקודות)

DFS (Depth-First-Search) הוא אלגוריתם חיפוש רקורסיבי בעץ בינארי.

להלן תכנית ב C++ המתארת את הפונקציה הרקורסיבית לחיפוש בעץ:

```
void dfs (TreeElement *t) {
  print(t->value);
  if (t->left!=null)  dfs(t->left);
  if (t->right!=null) dfs(t->right);
}
```

להלן מימוש באסמבלר של פונקציה זו:

```
dfsrec    proc near
          mov al,[bx]
          mov ah,0              1
```
הואיל ו printax מדפיסה מספר בן 16 ביט, ו t->value הוא בן 8 ביט, יש לאפס את הבית העליון
```
          call printax
          mov ax,-1
          cmp ax,[bx+1]
          jne re_call1          2
```
אם התנאי t->left!=null מתקיים, קפוץ לביצוע dfs(t->left)
```
sec_comp: mov ax,-1
          cmp ax,[bx+3]
          jne re_call2          3
```
אם התנאי t->right!=null מתקיים, קפוץ לביצוע dfs(t->right)
```
          jmp done
re_call1: push bx               4
```
שמור את t, כך שנוכל להשתמש בו כשנחזור מהרקורסיה
```
          mov bx,[bx+1]
          call dfsrec
          pop bx
          jmp sec_comp          5
```
אם ביצענו את dfs(t->left) עדיין עלינו לבדוק אם יש לבצע גם את t->right
```
re_call2: push bx
          mov bx,[bx+3]         6
```
העבר את t->right ל BX וקרא לרקורסיה
```
          call dfsrec
          pop bx
done:     ret
dfsrec    endp
```

הסבר במילים ליד כל אחת מן השורות המסומנות את תפקידה של השורה.

שימו לב – יש להסביר את תפקידה של הפקודה בהקשר של מימוש הפונקציה dfs (לדוגמא, עבור הפקודה push bx, לא תתקבל תשובה כמו "לדחוף את bx למחסנית"). ניתן ומומלץ להסביר את תפקיד השורות באמצעות תכנית ה C++.

ב. (25 נקודות)

כתוב סדרת פונקציות המממשות תור. יש לממש את הפונקציות הבאות:

הכנס לתור - enqueue(BX)

הוצא מהתור – BX=dequeue()

האם התור ריק – AX=isempty() (AX שווה לאחד אם התור ריק או אפס אם יש בו איברים)

הנחיות ורמזים לצורך הכתיבה:

1. ניתן להשתמש במקום בזכרון בגודל קבוע מראש (לדוגמא, להכריז על שטח בגודל 100 בתים ולתמוך בתור בעל מקום איחסון זה בלבד).
2. ניתן להשתמש במשתנים גלובליים (לדוגמא, לצורך מצביעים לתחילת וסוף התור).
3. יש לזכור כי התור ציקלי, ולבצע את הבדיקות המתאימות.
4. פרט לפונקציות, רשום גם את ההכרזות אותן אתה מבצע ב data segment.

הנחות יסוד:

tail – מצביע על האיבר הבא שיוצא מהתור בעקבות dequeue

head – מצביע על המקום הבא בו ייכנס האיבר בעקבות enqueue

במימוש זה נניח שקיים מקום אחסון בעל 256 בתים ושמספר האיברים המקסימלי לאחסון הוא 127.

להלן peudo-code המתאר את הפונקציות הממומשות:

```
void enqueue(BX) {
  if (((head-2) mod 256)==(tail mod 256)) {
     call error_full;
     endproc;
  }
  [head]=BX;
  head=(head-2) mod 256;
}

BX dequeue {
  if isempty() {
     call error_empty;
     endproc;
  }
  BX=[tail];
  tail=(tail-2) mod 256;
}

AX isempty() {
  AX=(head==tail);
}
```

להלן המימוש באסמבלר:

```
.data
queue dw 128 dup (?); queue is in DS:0
head db 0;
tail db 0;

proc      isempty
            push ax
            mov al,head
            cmp al,tail
            bne notempty
            mov ax,1
            jmp exitie
notempty:   mov ax,0
exitie:     pop ax
            ret
endproc   isempty

proc      enqueue
            push ax
            push bx
            push cx
            mov al,head
            mov cl,tail
            dec al
            dec al
            cmp al,cl
            beq errorfull
            mov ax,bx
            mov bl,head
            mov bh,0
            mov queue[bx],ax
            dec bx
            dec bx
            mov head,bl
            jmp exitenq
errorfull:  call printerrfull
exitenq:    pop cx
            pop bx
            pop ax
            ret
endproc   enqueue

proc      dequeue
            push ax
            push si
            call isempty
            cmp ax,1
            beq errorempty
            mov si,tail
```

```
            mov bx,queue[si]
            dec si
            dec si
            mov ax,si
            mov tail,al
            jmp exitdeq
errorempty: call printerrempty
exitdeq:    pop si
            pop ax
            ret
endproc     dequeue
```

ג. (25 נקודות)

BFS (Breadth-First-Search) הנו אלגוריתם לחיפוש בעץ בינארי.

להלן מימוש ב C++ של אלגוריתם זה:

```
// At the start the queue is empty
void bfs (TreeElement *t) {
  enqueue(t);
  while (queue is not empty) {
    t = dequeue();
    print(t->value);
    if (t->left!=null) enqueue(t->left);
    if (t->right!=null) enqueue(t->right);
}
```

כתוב פונקציה המממשת אלגוריתם חיפוש זה, ומדפיסה את האיברים של עץ בינארי.

אם מימשת בשיטה שונה את האלגוריתם, אנא רשום/רשמי בנוסף את קוד ה C++ המתאים לשיטה שלך.

יש להשתמש בפונקציות של סעיף ב'.

ניתן להניח כי קיימת הפונקציה printax שמדפיסה את תוכן רגיסטר ax.

```
proc      dfs
            call enqueue
whileloop:  call isempty
            cmp ax,1
            beq exitdfs
            call dequeue
            mov al,[bx]
            mov ah,0
            call printax
            mov ax,[bx+1]
            cmp ax,-1
            beq nextcheck
            push bx
            mov bx,[bx+1]
            call enqueue
            pop bx
nextcheck:  mov ax,[bx+3]
            cmp ax,-1
            beq nextloop
            mov bx,[bx+3]
            call enqueue
nextloop:   jmp whileloop
exitdfs:    ret
endproc   dfs
```