תוכניות דוגמא gcd4.asm, fib1.asm, fib2.asm, fib3.asm, fib3.asm

התוכנית gcd4.asm הינה מימוש בשיטה ה"יעילה" של הלולאת ה-while בתוכנית gcd3.asm, שהיא המימוש ה"אינטואיטיבי" שלה. כלומר ב-gcd4.asm הבדיקה בסוף הלולאה ומסתעפים אליה בהתחלה.

התוכניות fib1.asm, fib2.asm, fib3.asm, fib4.asm ממחישות מימוש for לחישוב מספרי פיבונצי. fib1.asm, fib2.asm עושות זאת בדרך "הכללית" fib3.asm ו-fib4.asm נעזרת בפקודה LOOP ו-LOOPD.

fib1.asm מממש את הלולאה בדרך ה"אינטואיטבית" ו-fib2.asm בדרך ה"יעילה". בהערות יש את האלגוריתם בקוד C שקוד האסמבלי הוא כביכול "התרגום" שלה.

fib4.asm משתמש בפקודות הלולאה המורחבות של ה-386. שימו לב שהלולאה נשלטת על ידי ECX ולא CX, האגר ECX הוא שמקבל את n ושהפקודות JECXZ ו-LOOPD החליפו את JCX ו-LOOP.

```
;
; gcd4.asm - Compute greatest common divisor, 386 version
;
     .MODEL SMALL
     .STACK 100h
     .DATA
X    DD  881790
Y    DD  188955
Gcd  DD  ?
     .CODE
     .386             ;  Enable 386 code
     MOV AX,@DATA     ; Program prefix
     MOV DS,AX        ; Set DS to point to data segment
                      ;
     MOV EAX,X        ;  Fisrt operand
     MOV EBX,Y        ;  Second operand
     JMP TestNext     ;                                    ***** C *****
Do1:                  ;                                  while (eax != ebx)
                      ;                                    {
                      ;                                         if (eax<ebx)
       JAE Xhigh      ; Skip XCHG IF EAX=>EBX                     {
         XCHG EAX,EBX ; EAX < EBX, Swap them                       temp = eax;
                      ;                                           eax = ebx;
                      ;                                           ebx = temp;
       Xhigh:         ;                                         }
         SUB EAX,EBX  ; EAX := EAX - EBX                        eax = eax - ebx;
TestNext:             ;
     CMP EAX,EBX      ;
     JNE Do1          ; Exit if EAX = EBX
End1p:                ;                                    } /* while */
     MOV Gcd,EAX      ; Store result                      gcd = eax;
                      ;
     MOV AH,4Ch       ; Set terminate option for int 21h
     INT 21h          ; Return to DOS (terminate program)
     END
```

```
;
; fib1.asm - Compute Fibonacci n
;
     .MODEL SMALL
     .STACK 100h
     .DATA
n        DD  20
Fibo_n   DD  ?
      .CODE
      .386              ;  Enable 386 code
      MOV AX,@DATA      ; Program prefix
      MOV DS,AX         ; Set DS to point to data segment
                        ;
      MOV ECX,n          ;              *****  C  *****
      MOV ESI,3          ;              ecx = n;
      MOV EBX,1          ;
      MOV EDX,1          ;              ebx = edx = 1;
Do1:                     ;              for(esi = 3; esi <= ecx; esi++)
      CMP ESI,ECX        ;               {
      JA Endlp           ;                 eax = ebx + edx;
      MOV EAX,EBX        ;                 edx = ebx;
      ADD EAX,EDX        ;                 ebx = eax;
                         ;               }
      MOV EDX,EBX        ;
      MOV EBX,EAX         ;
                          ;
      INC ESI             ;
      JMP Do1
                          ;
Endlp:                    ;
      MOV Fibo_n,EAX     ; Store result    fibo_n = eax
                        ;
      MOV AH,4Ch        ; Set terminate option for int 21h
      INT 21h           ; Return to DOS (terminate program)
      END
```

```
;
; fib2.asm - Compute Fibonacci n
;
     .MODEL SMALL
     .STACK 100h
     .DATA
n        DD  20
Fibo_n   DD  ?
      .CODE
      .386              ;  Enable 386 code
      MOV AX,@DATA      ; Program prefix
      MOV DS,AX         ; Set DS to point to data segment
                        ;
      MOV ECX,n          ;              *****  C  *****
      MOV ESI,3          ;              ecx = n;
      MOV EBX,1          ;
      MOV EDX,1          ;              ebx = edx = 1;
      JMP TestNext       ;
Do1:                     ;              for(esi = 3; esi <= ecx; esi++)
      MOV EAX,EBX        ;                {
      ADD EAX,EDX        ;                  eax = ebx + edx;
                         ;                  edx = ebx;
      MOV EDX,EBX        ;                  ebx = eax;
      MOV EBX,EAX         ;               }
                          ;
      INC ESI             ;
                          ;
TestNext:                 ;
      CMP ESI,ECX         ;
      JNA Do1             ;
Endlp:                    ;
      MOV Fibo_n,EAX      ; Store result    fibo_n = eax
                        ;
      MOV AH,4Ch        ; Set terminate option for int 21h
      INT 21h           ; Return to DOS (terminate program)
      END
```

```
;
; fib3.asm - Compute  Fibonacci n
;
     .MODEL SMALL
     .STACK 100h
     .DATA
n        DW  20
Fibo_n   DD  ?
      .CODE
      .386              ;  Enable 386 code
      MOV AX,@DATA      ; Program prefix
      MOV DS,AX         ; Set DS to point to data segment
                        ;
      MOV CX,n          ;              *****  C  *****
      SUB CX,2          ;               cx = n-2;
      JS  Endlp         ;
      MOV EBX,1          ;
      MOV EDX,1          ;               eax = ebx = edx = 1;
      MOV EAX,1          ;
      JCXZ EndLp         ;
Do1:                     ;               for(cx = n-3; cx > 0; cx-- )
      MOV EAX,EBX        ;                 {
      ADD EAX,EDX        ;                    eax = ebx + edx;
                         ;                    edx = ebx;
      MOV EDX,EBX        ;                    ebx = eax;
      MOV EBX,EAX         ;                 }
                          ;
      LOOP Do1            ;
                          ;
Endlp:                    ;
      MOV Fibo_n,EAX     ; Store result      fibo_n = eax
                        ;
      MOV AH,4Ch        ; Set terminate option for int 21h
      INT 21h           ; Return to DOS (terminate program)
      END
```

```
;
; fib4.asm - Compute fibo
;
     .MODEL SMALL
     .STACK 100h
     .DATA
n        DD  20
Fibo_n   DD  ?
      .CODE
      .386              ;  Enable 386 code
      MOV AX,@DATA      ; Program prefix
      MOV DS,AX         ; Set DS to point to data segment
                        ;
      MOV ECX,n          ;              *****  C  *****
      SUB ECX,2          ;
      JS Endlp           ;
      MOV EBX,1          ;
      MOV EDX,1          ;               eax = ebx = edx = 1;
      MOV EAX,1          ;
      JECXZ EndLp        ;
Do1:                     ;               for(ecx = n-2; ecx > 0; ecx-- )
      MOV EAX,EBX        ;                 {
      ADD EAX,EDX        ;                   eax = ebx + edx;
                         ;                   edx = ebx;
      MOV EDX,EBX        ;                   ebx = eax;
      MOV EBX,EAX         ;                }
                          ;
      LOOPD Do1           ;
                          ;
Endlp:                    ;
      MOV Fibo_n,EAX     ; Store result      fibo_n = eax
                        ;
      MOV AH,4Ch        ; Set terminate option for int 21h
      INT 21h           ; Return to DOS (terminate program)
      END
```

JXX - Jump Instructions Table

Mnemonic	Meaning	Jump Condition
JA	Jump if Above	CF=0 and ZF=0
JAE	Jump if Above or Equal	CF=0
JB	Jump if Below	CF=1
JBE	Jump if Below or Equal	CF=1 or ZF=1
JC	Jump if Carry	CF=1
JCXZ	Jump if CX Zero	CX=0
JE	Jump if Equal	ZF=1
JG	Jump if Greater (signed)	ZF=0 and SF=OF
JGE	Jump if Greater or Equal (signed)	SF=OF
JL	Jump if Less (signed)	SF != OF
JLE	Jump if Less or Equal (signed)	ZF=1 or SF != OF
JMP	Unconditional Jump	unconditional
JNA	Jump if Not Above	CF=1 or ZF=1
JNAE	Jump if Not Above or Equal	CF=1
JNB	Jump if Not Below	CF=0
JNBE	Jump if Not Below or Equal	CF=0 and ZF=0
JNC	Jump if Not Carry	CF=0
JNE	Jump if Not Equal	ZF=0
JNG	Jump if Not Greater (signed)	ZF=1 or SF != OF
JNGE	Jump if Not Greater or Equal (signed)	SF != OF
JNL	Jump if Not Less (signed)	SF=OF
JNLE	Jump if Not Less or Equal (signed)	ZF=0 and SF=OF
JNO	Jump if Not Overflow (signed)	OF=0
JNP	Jump if No Parity	PF=0
JNS	Jump if Not Signed (signed)	SF=0
JNZ	Jump if Not Zero	ZF=0
JO	Jump if Overflow (signed)	OF=1
JP	Jump if Parity	PF=1
JPE	Jump if Parity Even	PF=1
JPO	Jump if Parity Odd	PF=0
JS	Jump if Signed (signed)	SF=1
JZ	Jump if Zero	ZF=1

תקציר מספר 5

המחסנית

מחסנית באופן כללי היא מבנה נתונים המאחסן ומנפק נתונים בסדר LIFO - Last In First Out נאי"ר - נכנס אחרון יוצא ראשון. המוסכמה היא שהפעולות על מחסנית נקראים PUSH ו-POP. לדוגמא הסידרה הבאה של פעולות מחסנית יוצרת (פולטת) את הסידרת התוים 'CEDBFA':

```
PUSH('A')
PUSH('B')
PUSH('C')
POP
PUSH('D')
PUSH('E')
POP
POP
POP
PUSH('F')
POP
POP
```

למחסנית כמבנה נתונים יש שימושים שונים - למשל ביטויים אריתמטיים מממושים ע"י מחסנית. אבל אנחנו נדון כאן במחסנית המערכת - מחסנית שיש לה תמיכה בחומרה (בתוך ה-CPU). המחסנית הזאת נועדה למימוש גורמים שונים במערכת ההפעלה - כמו העברת פרמטרים, משתנים לוקליים אוטומטיים, רקורסיה ותהליכים. השימוש במחסנית לא יהיה רק מחסנית כפשוטו, כלומר לא רק ע"י הפקודות PUSH ו-POP בלבד.

התמיכה למחסנית המערכת בחומרה

ב-8086, המחסנית ממומשת בזכרון הרגיל של המחשב. זאת בניגוד למחשבים מסוימים אחרים, שלמחסניות שלהם זכרון מיוחד משלהן.

עיקר התמיכה בחומרה למחסנית היא כלהלן:

אוגר סגמנט SS.

אוגר מצביע SP.

אוגר מצביע BP.

וכן פקודות מכונה PUSH, POP, PUSHF, POPF.

ניהול המחסנית

המחסנית ב-8086 מנוהלת באופן הבא:

נניח שבראש התוכנית מופיע

```
.STACK 100h
```

אזי עם התחלת הריצה, המערכת דואגת להיווצרות המצב הבא:

כלומר ל-SP יש את ה-OFFSET של הבית הראשון שלאחר המחסנית.

אופרנד PUSH - קודם באופן אוטומטי SP = SP - 2.
ואחר כך דוחף מילה (2 בתים) למחסנית.

לדוגמא:

```
PUSH AX

PUSH Mem16

PUSH 1053
```

האופרנד היחיד יכול להיות אוגר 16 ביט או זכרון 16 ביט או קבוע 16 ביט.

אופרנד POP - קודם שולף מילה (2 בתים) מהמחסנית
ואחר כך באופן אוטומטי SP = SP + 2.

לדוגמא:

```
POP AX

POP Mem16
```

האופרנד היחיד יכול להיות אוגר 16 ביט או זכרון 16 ביט.

PUSHF - ללא אופרנדים - קודם באופן אוטומטי SP = SP - 2.
ואחר כך דחיפת אוגר הדגלים (Flag Register)
לתוך המחסנית.

POPF - ללא אופרנדים - קודם שליפת אוגר הדגלים (Flag Register)
מתוך המחסנית.
אחר כך באופן אוטומטי SP = SP + 2.

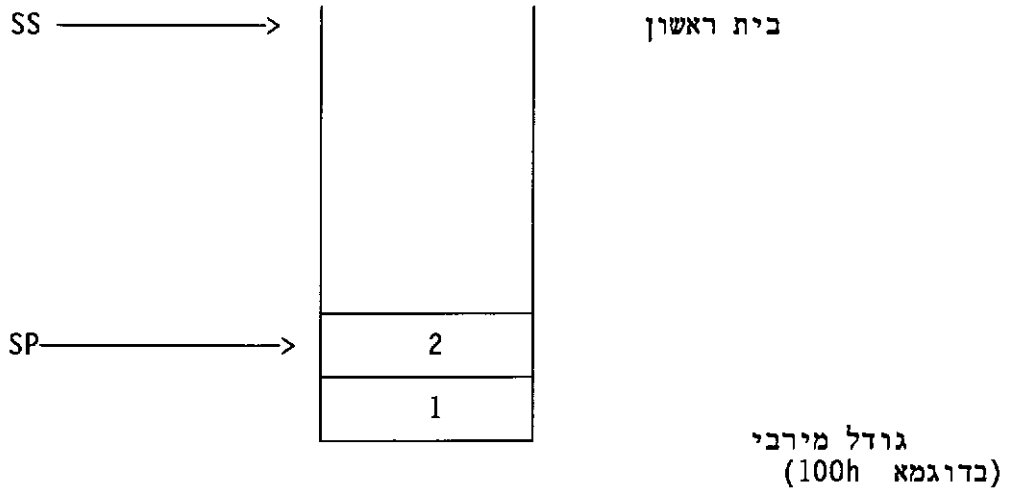
כפי שמשתמע לעיל ב-286, 8086 PUSH ו-POP תמיד 2 בתים. אי אפשר לעשות PUSH ל-byte אחד.

דבר נוסף הוא שהמחסנית מתמלאת מהכתובות הגבוהות לכיוון הכתובות הנמוכות. להמחשה, כאשר המחסנית מלאה נתונים עד הסוף, SP = 0. כאשר המחסנית ריקה מנתונים (כל הזכרון של המחסנית פנוי לשימוש) - למשל בתחילת הריצה של התוכנית - הגודל המירבי של המחסנית = SP.

לדוגמא, אם נבצע את הפעולות הבאות:

```
MOV DX,0102h
PUSH DX
```

אז המחסנית תראה כך:



זאת משום שב-x86 הבית המשמעותי פחות מקדים תמיד את המשמעותי יותר (במובן מסוים ההיפך מאיך שלנו מקובל לחשוב).

כלומר, SP תמיד מצביע על המילה הראשונה לשליפה (תא לא פנוי ראשון).

המחסנית ב-386 ואילך

כל המתואר קודם ממשיך כמובן להתקים. האוגרים SS, SP ו-BP ממשיכים להתקיים, וכמו כן הפקודות PUSH, ו-POP על אופרנדים 16 ביט.

ב-386 נוספו הדברים הבאים:

1. האוגרים SP ו-BP הורחבו ל-32 ביט ESP ו-EBP. לפיכך ההיסטים של המחסנית יכולים להיות (אם מעונינים בכך) בגודל 32 ביט (במגבלות שציינתי קודם, כלומר ב-DOS בדרך כלל רק עד 65536).
2. פעולות הדחיפה ושליפה מהמחסנית יכולים להיות על, בנוסף ל-16 ביט, גם 32 ביט (4 בתים) של אינפורמציה.

הפקודות PUSH, POP תומכת גם פעולות דחיפה של מילים כפולות (DWORD):

לדוגמא:

```
PUSH EAX

PUSH Mem32

PUSH 100000

POP EAX

POP Mem32
```

כאן בצורה דומה,

PUSH מפחית ESP = ESP - 4 לפני כתיבה,

POP קודם קורא ואח"כ מקדם ESP = ESP + 4.

קיימות גם פקודות

PUSHFD, POPFD ששומרים את אוגר הדגלים המורחב 32bit.

יש גם פקודות מחסנית נוספות (PUSHAD, POAD, ENTER, LEAVE) שלא ניכנס אליהן כאן.

<u>קריאת המחסנית ללא שליפה</u>

תהליך <u>חיוני</u> <u>בשפה עלית כמו C</u> שבו <u>הפרמטרים והמשתנים הלוקליים</u> ממומשים

במחסנית ויש צורך לקרוא ולכתוב את תוכן המחסנית בדרך שאינה שליפה ודחיפה.

ב-8086 הדבר נעשה בעיקר דרך האוגר BP.

התהליך הוא

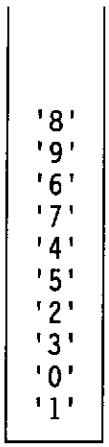
MOV BP,SP
קדם או הפחת את BP לפי הצורך והתיחס
דרכו לזכרון.

[BP + k], [BP - k] התיחסות יחסית ל-SS במחסנית עם ל-BP כ-offset.

תוכנית דוגמא stack3.asm

התוכנית stack3.asm עושה משהו דומה ל-memory4.asm רק שחלק מהתוכן המודפס מועתק מהמחסנית. המטרה של התוכנית הזו היא להמחיש את האפשרויות של קריאת מידע מתוך המחסנית <u>ללא</u> שליפה (שימוש ב-POP) באמצעות האוגר BP המשמש פוינטר טבעי למחסנית.

שים לב שבשל האופי המיוחד של פעולת ה-PUSH והעובדה שהמחסנית הזו מתמלאת מהכתובות הגבוהות לנמוכות תוכן המחסנית עם סיום לולאת ה-PUSH-ים (StackStore) נראה ככה:



שנים מהלולאות (StackLoop1 ו-StackLoop2) של התוכנית הם סריקות (בשני הכיוונים) של המחסנית באמצעות BP וזה מסביר את תוצאות הפלטים שלהם ("8967452301" ו-"1032547698" בהתאמה).

```
;
;   stack3.asm - demonstrate hardware stack usage.
;
     .MODEL SMALL
     .STACK 100h
     .DATA
Separator        EQU '#'
DisplayString    DB  64 DUP('$')
Numbs            DB '0123456789'
Stack_Size       DW (?)
      .CODE
ProgStart:
     MOV AX,@DATA                    ; Set DS to point ...
     MOV DS,AX                       ; ... to data segment
                                     ;
     MOV DI,OFFSET DisplayString     ; Have DI point to start ...
                                     ; ... of DisplayString
     MOV Stack_Size,SP                ; Save stack size
                                     ;
     XOR BX,BX                       ;  BX := 0;
     MOV CX,5                        ;  Size(Numbs) = 2*5
                                     ;
StackStore:
        MOV DX,WORD PTR Numbs[BX]    ; Read two bytes in Numbs
        PUSH DX                      ; Store in Stack
        MOV [DI],DX                  ; Store in DisplayString
        ADD BX,2                     ; BX point to next word in Numbs
        ADD DI,2                     ; DI point to next available position
       LOOP StackStore               ; Repeat 5 times
                                     ;
     MOV DL,Separator                ;
     MOV [DI],DL                     ; Store '#' ...
     INC DI                          ;   ... to separate
                                     ;
     MOV CX,Stack_Size               ;
     SUB CX,SP                       ; Set CX to equal ...
                                     ;  ... number of relevant ...
                                     ;  ... bytes in stack
     MOV BP,SP                       ;
StackLoop1:                          ; Print Stack downward
        MOV BL,[BP]                  ;
        INC BP                       ;
        MOV [DI],BL                  ; Store in DisplayString
        INC DI                       ; point to next available position
     LOOP StackLoop1                 ;
                                     ;
     MOV DL,Separator                ;
     MOV [DI],DL                     ; Store '#' ...
     INC DI                          ;   ... to separate
                                     ;
     MOV BX,[BP-3]                   ;  Retrieve eighth and ninth bytes ...
     MOV [DI],BX                     ;   ... in stack
     ADD DI,2                        ;
                                     ;
     MOV DL,Separator                ;
     MOV [DI],DL                     ; Store '#' ...
     INC DI                          ;   ... to seperate
                                     ;
```

```
      MOV BP,SP                          ;  Retrieve third and forth bytes ...
      MOV BX,[BP+2]                      ;    ... in stack
      MOV [DI],BX                        ;
      ADD DI,2                           ;
                                         ;
      MOV DL,Separator                   ;
      MOV [DI],DL                        ; Store '#' ...
      INC DI                             ;   ... to seperate
                                         ;
      MOV CX,Stack_Size                  ;
      SUB CX,SP                          ; Set CX to equal ...
                                         ;  ... number of relevant ...
                                         ;  ... bytes in stack
      MOV BP,Stack_Size                  ;
      DEC BP                             ; Set BP to point to last ...
                                         ; ... byte in stack
StackLoop2:                              ; Print stack, upward
         MOV BL,[BP]                     ;
         DEC BP                          ;
         MOV [DI],BL                     ; Store in DisplayString
         INC DI                          ; point to next available position
      LOOP StackLoop2                    ;
                                         ;
      MOV AH,9                           ; Set print option for int 21h
      MOV DX,OFFSET DisplayString        ;  Set  DS:DX to point to DisplayString
      INT 21h                            ;  Print DisplayString
                                         ;
      MOV AH,4Ch                         ; Set terminate option for int 21h
      INT 21h                            ; Return to DOS (terminate program)
      END  ProgStart
```

```
E:\>stack3
0123456789#8967452301#30#67#1032547698
E:\>
```