תוכניות הדוגמא

תוכנית הדוגמא של החוברת הם תוכניות אסמבלי או C שקומפלו ב-Turbo Assembler, Turbo C++ for DOS או Borland C++. כולם הורצו בעבר ב-DOS אך עדיין מתקמפלים ורצים תחת הגרסאות השונות של Windows בתוך חלונות command (כל הגירסאות) או cmd (XP, NT, 2000). החלונות הללו משמשות סביבות הדמיה (אמולציה) של DOS, כלומר תוכניות ה-DOS המורצות בחלונות הללו "חושבות" שהן עדיין רצות תחת DOS ובכל התוכניות שבחוברת עובדות בהצלחה בסביבה המודמית.

משמעות המילה "אמולציה" פירושה בהקשר שלנו "הדמיה במובן של תחליף". יש לך A המנסה להיות תחליף ל-B למרות שהוא לא B. אם יש לך "אמולציה" של משהו פירושו שאתה כאילו לא צריך את הדבר האמיתי (למרות שאמולציה מושלמת ב- 100% היא בדרך כלל אידאל שאפשר רק לשאוף אליו). המונח הקשור "סימולציה" פירושה גורם הדמיה במובן של מציאות מדומה (ולא תחליף אפקטיבי). למשל מערכת לאימון טייסים להתמודדות עם מצבי חירום הם סימולציות משום שהם אינם מטוסים כלל ומדמים טיסות רק בהקשר הצר של אימון.

הסיבה שמערכת ההפעלה DOS ממשיכה להיות הבסיס לתוכניות הדוגמא והתרגילים של הקורס משום שחלק מהן, אותן תוכניות שקשורות למנגנון הפסיקות והתוכניות המבצעות קלט/פלט, עושות פעולות הנחשבות היום מיוחסות. במערכת DOS כל תוכנית יכולה היה לעשות פעולות כאלו אך במערכות החדשות הדבר לא ניתן, אך תוכנות האמולציה מדמות איכשהוא גם את האספקטים הללו. מעבר לזה סביבות הפיתוח וניפוי השגיאות הקיימים כאן הם מאד נוחות לעבודה. ההשלכות של תחת איזה מערכת ההפעלה עובדים התוכניות הן די שוליות מהבחינה הזו שהנושאים העיקריים הנלמדים בקורס אינם תלויים בהן.

תוכנית דוגמא hellola.asm

תוכנית הדוגמא הראשונה שלנו תמחיש חלק משמעותי מהמבנה של תוכניות האסמבלי לפי השיטה שנעבוד בהם בשלב זה - תוכניות אסמבלי טהורות תוך שימוש בהנחיות הסגמנטים המקוצרות. בשיטה הזו האסמבלר פורש בשבילכם באופן אוטומטי הגדרות שונות ופותר אתכם מזה, וזה נוח בשלב המוקדם הזה.

- תזכורת: כל תוכנית הדוגמא כתובות בצורה כזו שכל מילה על טהרת האותיות הגדולות הם מילים שמורות בשפת האסמבלי. כל מילה שמכילה גם אותיות קטנות הם בחזקת מזהים שיכולים להיות מוחלפים בכל מילה אחרת.

- הנחיות לאסמבלר: מדובר בשורות בתוכנית שמגדירות לאסמבלר איך לפרש שורות פקודה שמתחתיהן אך אינם משתקפים ישירות בקובץ הבינארי.

- בכל שורה מה שנכתב לאחר ה-";" הינו <u>הערה</u>. האסמבלר פשוט יתעלם מכל תוכן שיש שם עד לשורה הבאה.

- ההנחיה

```
.MODEL SMALL
```

אומרת לאסמבלר לבחור ערכים מסוימים מבין אפשריות שונות שיש. לא ניכנס לכך בשלב זה.

- ההנחיה

```
.STACK 100h
```

מנחה את האסמבלר להגדיר מחסנית בגודל 100 הקסדצימל (256 עשרוני) בתים. למה זה נחוץ נלמד בהמשך הקורס. הגדרה שקולה לחלוטין תהיה:

```
.STACK 256
```

בכל הקשור לקבועים, אם התו האחרון בקבוע הוא H או D או B הקבוע

מפורש כהקסהדצימלי, עשרוני או בינארי בהתאמה. אם הוא על טהרת המספרים הוא יפורש כעשרוני. מספר הקסה שמתחיל באות (כמו B800h) חייב להכתב עם אפס מוביל (0B800h בדוגמא) כי אחרת האסמבלר ינסה לפרש את המספר כשם של משתנה.

- ההנחיה

.DATA

אומר שמה שמתואר להלן יהיה תאור של שטח מידע - משתנים במושגים של שפת תכנות נוסח C. גם את התמונה המלאה לזה נראה מאוחר יותר.

- שורת הפקודה

```
DisplayString  DB 'Hello World!',13,10,'$'
```

הינה הגדרת משתנה בשם DisplayString שהוא למעשה מערך של תוים (בית בודד כל אחד).
הסיבה שמדובר במערך של בתים נקבעת על ידי ההנחיה DB. האפשרויות הם:

DB - בית אחד

DW - 2 בתים

DD - 4 בתים

DF, DP - 6 בתים

DQ - 8 בתים

DT - 10 בתים

באופן עקרוני איברים במערך מוגדרים בין פסיקים (...3,5,4,6...) אבל במחרזות ניתן להגדיר בין פסיקים ('Hello' שקול ל-'H','e','l','l','o'). האסמבלר מקצה מקום לקבועים הללו ודואג שיאותחלו במה שהמתכנת מצין.

הערכים 13,10 הם "עבור לשורה הבאה". ה-'$' הוא לצורכי סיום הדפסה - עוד נדבר על כך.

- ההנחיה

```
.CODE
```

מנחה את האסמבלי לכך שמה שמתואר להלן הוא החלק הביצועי של התוכנית.

- השורה

```
Begin:
```

הוא סמן (label) הנותן שם לפקודה. במקרה הזה אנחנו עושים זאת בכדי לסמן לאסמבלר מי הפקודה הראשונה לביצוע של התוכנית הזאת. הנקודה שבו מוכרזת הפקודה הזו כראשונה לביצוע היא בהנחיה

```
END Begin
```

ה-END מנחה את האסמבלר שזהו סוף התוכנית ובמקרה הזה גם מציין ש-Begin הוא הפקודה הראשונה לביצוע של התוכנית.

- הפקודות

```
MOV AX,@DATA
MOV DS,AX
```

גורמים לאוגר המיוחד DS להצביע על ששח המשתנים של התוכנית (מה משמעות הדבר ולמה זה נחוץ נעמוד בהמשך).

- הפקודה INT 21h וקלט פלט

בתוכנית אסמבלי ביצוע קלט/פלט הוא לכאורה עינין טכני מאד הכרוך

בידעה מדויקת של מנגנון הקלט/פלט של המחשב אולם יש דרך להתחמק מכך וזה להסתמך על רוטינות קיימות במחשב. במקרה הזה אנחנו מסתמכים על מערכת ההבעלה DOS. INT 21h היא פניה לספרית רוטינות של DOS, הרוטינה המדויקת נקבעת לפי הערך של AH ברגע הקריאה. אנחנו מסתמכים על הרוטינות:

INT 21h, AH = 9

"הדפס למסך תוים מהנקודה DS:DX עד שתתקל בתו '$'".

INT 21h, AH = 4Ch

"סים ריצה והחזר שליטה ל-DOS".

הפקודה INT היא אחת מפקודות ההסתעפות (שמשנות את הפקודה הבאה לביצוע). היא פקודת הסתעפות די מיוחדת במספר מובנים, בין השאר שהיא מצינת איכשוא איך לחזור לתוכנית. INT 21h היא הסתעפות לשטח זיכרון שמור ל-DOS שבו הוא מאכסן רוטינות קלט/פלט שלו. הרוטינות הללו משמשים את שורת הפקודה אך עומדות גם לרשות תוכנית אפליקציה.

בתוכנית אסמבלי עצירת התוכנית היא פקודה שהתוכנית חייבת לבצע אותה (שום דבר בתוכנית לא תבצע את זה אוטומטית). אם לא בצע את "פקודה החזרה" הזו, התוכנית תמשיך לקרוא זיכרון ולנסות לפרש את התוכן כתאור של פקודות מכונה ולנסות לבצע אותם, דבר שבמקרה הטוב יתקע את התוכנית.

- הפקודה

MOV DX,OFFSET DisplayString

מציב ל-DX חלק מהכתובת של המשתנה DisplayString (ל-DS כבר דאגנו בשתי הפקודות הראשונות של התוכנית). זו איננה הדרך היחידה (או אפילו העיקרית) לחישוב כתובות, אנחנו נלמד על הפקודה LEA בשלב יותר מאוחר.

אם נניח ש- Var1 הוא שם של משתנה (2 בתים נניח) אז צריך להבדיל

```
MOV AX,OFFSET Var1
```

שמציבה ל-AX את הכתובת של Var1 לבין

```
MOV AX,Var1
```

המציבה ל-AX את התוכן של Var1. זה בערך כמו ההבדל בין ;x = &y לבין x = y; בשפת C.

- הידור התוכנית

בשלב הזה, אחרי שנקליד תוכנית נוסח hellola.asm באמצעות תוכנת עריכה (editor) נתרגם אותם לקובץ exe בשני שלבים, תוך יצירת קובץ ביניים עם סיומת obj:

```
tasm hellola.asm
```

ו-

```
tlink hellola.obj
```

ההרצה עצמה תהיה הרצת הקובץ hellola.exe.

במידה ויש שגיאות ה-tasm יודיע על כך ולא יווצר קובץ ה-obj.

אם אנחנו רוצים להשתמש ב-Turbo Debugger בכדי לאתר שגיאות נריץ:

```
tasm /zi hellola.asm
tlink /v hellola.obj
td hellola.exe
```

```
;
;  hello1a.asm - send message 'Hello World!' to the screen.
;
     .MODEL SMALL
     .STACK 100h
     .DATA
DisplayString DB 'Hello World!',13,10,'$'
                              ;
      .CODE
Begin:
      MOV AX,@DATA      ; DS can be written to only through a register
      MOV DS,AX         ; Set DS to point to data segment
      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 Begin
```

```
E:\>tasm hello1a.asm
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland International

Assembling file:   hello1a.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  389k


E:\>tlink hello1a.obj
Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

E:\>hello1a.exe
Hello World!

E:\>
```

תוכנית דוגמא hello4.asm

זוהי תוכנית דומה עקרונית ל-hellola.asm אך שואלת את המשתמש אם השעה היא אחרי 12 בצהריים ולפי תגובת מהמשתמש מדפיסה או את "Good afternoon world!" או "Good morning world!". אם התגובת המשתמש היא 'y' או 'Y' יודפס "Good afternoon world!" ובמקרה של כל תגובה אחרת יודפס "Good morning world!". על מנת שהתוכנית לא תבדיל בין 'y' ל-'Y' היא חייבת לבצע את שתי ההשוואות.

בנוסף לרוטינות INT 21h עליהם הסתמכנו ב-hellola.asm אנחנו מסתמכים על הרוטינה:

INT 21h, AH = 1

שמשמעותה "המתן ללחיצת מקש (עם קוד Ascii) מהמקלדת והחזר את קוד ה-Ascii ב-AL".

סימו לב למבנה של קטעי התוכנית

```
   MOV AH,1
   INT 21h
   CMP AL,'y'
   JE IsAfternoon

     .....

   MOV DX,OFFSET GoodMorningMessage
   JMP DisplyGreeting
IsAfternoon:
   MOV DX,OFFSET GoodAfternoonMessage
DisplayGreeting:
   MOV AH,9
   INT 21h
```

כאן אנחנו למעשה רואים פחות או יותר איך בפועל ממומשים תוכנית

בשפת מכונה ובאסמבלי: כל הפקודות שעשויות להתבצע נמצאות בגוף התוכנית. פקודות ההסתעפות של התוכנית דואגות לכך שה-CPU יעקוף את הפקודות שלפי הנסיבות אמורות שלא להתבצע ולהגיע אל אלה שכן. הדבר נעשה ע"י שילוב של פקודת השוואה (CMP) ופקודות הסתעפות (JE, JMP). במושגים של שפת C זה כאילו יש בשפה רק "if" ו-"goto" אבל אין מבנים כמו "{" ו-"}" שלא לדבר על while וכו'. אפקט ה-"if" מתקבל ע"י פקודות הסתעפות מותנות שבו שינוי הפקודה הבאה מתבצעת רק אם תנאי מסוים מתקים, ואחרת הפקודה הבאה לביצוע היא הפקודה העוקבת בזיכרון. בתוכנית הזו משתמשים בפקודה "JE" שמשמעותה "הסתעף במקרה של שוויון". המבנה שמדובר כאן הוא שהפקודה

```
CMP AL,'y'
```

משווה את תוכן AL עם 'y' ושומר את התוצאה היכן שהוא. הפקודה

```
JE IsAfternoon
```

מבצעת את הסתעפות לנקודה האמורה (שינוי הפקודה הבאה לביצוע) רק אם הם אכן שווים. במידה ולא הפקודה הבאה לביצוע תישאר הפקודה העוקבת בזיכרון, ואז מתבצעים הפקודות העוקבות תוך עקיפת הפקודות של המקרה השני על ידי פקודת ההסתעפות הבלתי מותנית JMP.

```
;
; hello4.asm - Conditional response.
;
     .MODEL SMALL
     .STACK 100h
     .DATA
TimePrompt DB 'Is it after 12 noon (y/n)?',13,10,':$'
GoodAfternoonMessage DB 13,10
         DB 'Good afternoon, world!',13,10,'$'
GoodMorningMessage DB 13,10
         DB 'Good morning, world!',13,10,'$'
                              ;
                              ;
     .CODE
ProgStart:
     MOV AX,@DATA              ; DS can be written to only through a register
     MOV DS,AX                 ; Set DS to point to data segment
     MOV AH,9                  ; Set print option for int 21h
     MOV DX,OFFSET TimePrompt  ;  Set  DS:DX to point to TimePrompt
     INT 21h                   ;  Print TimePrompt
     MOV AH,1                  ; DOS get character function #
     INT 21h                   ; Get a single character from keyboard
     CMP AL,'y'                ; AL has input. Compare with 'y'
     JE  IsAfternoon           ; If AL = 'y' then go to IsAfternoon
     CMP AL,'Y'                ; Compare with 'Y'
     JE  IsAfternoon           ; If AL = 'Y' then go to IsAfternoon
IsMorning:
     MOV  DX,OFFSET GoodMorningMessage  ; Point display message to morning
     JMP  DisplayGreeting               ; Avoid following code
IsAfternoon:
     MOV  DX,OFFSET GoodAfternoonMessage ; Point display message to afternoon
DisplayGreeting:
     MOV AH,9                          ; Set print option for int 21h
     INT 21h                           ; Print  chosen message
     MOV AH,4Ch                ; Set terminate option for int 21h
     INT 21h                   ; Return to DOS (terminate program)
    END ProgStart
```

```
E:\>tasm hello4.asm
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland International

Assembling file:   hello4.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  376k


E:\>tlink hello4.obj
Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

E:\>hello4.exe

Is it after 12 noon (y/n)?
:y
Good afternoon, world!

E:\>
```

TABLE 5.4 *(Continued)*

AH	Purpose	Type	Description
3A	Remove Subdirectory	Disk	RMDIR function; DS:DX points to string containing drive and path names.
3B	Change Current Directory	Disk	CHDIR function; DS:DX points to string containing drive and path names.
3C	Create File	File	CREATE function; if file pointed to in DS:DX does not exist the file is opened.
3D	Open File	File	DS:DX points to file; AL = 0 (read only), 1 (write only), or 2 (read/write). (See *DOS Technical Reference* manual).
3E	Close File Handle	File	BX contains file handle; file closed, directory updated, and internal file buffers removed.
3F	Read from File/Device	File	BX = file handle, CX = number bytes to read, DS:DX = buffer to be loaded. After call, AX = number bytes read.
40	Write to File/Device	File	Inverse of 3F.
41	Delete File from Directory	File	Removes a directory entry associated with the filename pointed to in DS:DX.
42	Move File Read/Write Pointer	File	See *DOS Technical Reference* manual.
43	Change File Mode	File	See *DOS Technical Reference* manual.
44	I/O Control for Devices	I/O	See *DOS Technical Reference* manual.
45	Duplicate File Handle	File	On entry BX = file handle, on exit AX = duplicate.
46	Force Duplicate File Handle	File	Forces the handle in CX to refer to the same file at the same position as the handle in BX.
47	Get Current Directory	Disk	DL = drive number; DS:SI = pointer to 64 byte user area to contain directory; and AX returns error codes.
48	Allocate Memory	Memory	BX = number paragraphs and AX:0000 points to the allocated blocks.
49	Free Allocated Memory	Memory	Frees the memory allocated with 48H.
4A	Modify Allocated Memory Blocks	Memory	Modifies blocks to contain new block size. ES = block segment and BX = new block size in paragraphs.
4B	Load/Execute Program	Control	Provides for overlaying. DS:DX points to program and ES:BX points to parameter block for load.
4C	Terminate Process	Control	Exits to invoking process.
4D	Get Return Code	Misc.	See *DOS Technical Reference* manual.
4E	Find First Matching File	File	Finds the first filename that matches the file pointed to in DS:DX. CX = attribute used in search.
4F	Find Next Matching File	File	Same as 4E except finds second match. The DTA contains information from 4EH or previous 4FH.
54	Get Verify Setting	Misc.	Returns the value of verify set with 2EH, in AL.
56	Rename File	File	Renames the file in DS:DX with ES:DI.

57	Get/Set File Date and Time	Misc.	On entry: AL = 0 (get) or 1 (set), BX = file handle, CX = time, and DX = date.
59	Get Extended Error (DOS 3.00 & 3.10)	Error	Returns additional error information (see *DOS Technical Reference* manual).
5A	Create Unique File (DOS 3.00 & 3.10)	File	Generates file pointed to by DS:DX (path ends with/) and CX = attribute.
5B	Create New File (DOS 3.00 & 3.10)	File	Creates new file pointed to by DS:DX with attribute in CX.
5C	Lock/Unlock File Access (DOS 3.00/3.10)	File	AL = 0 (lock) or 1 (unlock), BX = file handle, CX = byte offset high, DX = byte offset low, SI = length high and DI = length low.
5E00	Get Machine Name (DOS 3.10)	Misc.	DS:DX points to location where computer name returned.
5E02	Set Printer Setup (DOS 3.10)	Network	BX = redirection list index, CX = length string, and DS:SI points to string to be put in front of all print files.
5E03	Get Printer Setup	Network	Reverse 5E02.
5F02	Get Redirection List Entry (DOS 3.10)	Network	Returns nonlocal network assignments.
5F03	Redirect Device (DOS 3.10)	Network	Principally for networking.
5F04	Cancel Redirection	Network	Principally for networking.
62	Get PSP (DOS 3.00 & 3.10)	Misc.	Returns the Program Segment Prefix in BX.

One goal of this subsection was to provide the reader with an introduction to techniques for communicating with the various peripherals attached to the system. INT 10H served to provide graphics communication. Now, INT 21H provides the remaining I/O needed to write flexible modular, general-purpose programs. No longer will it be necessary to use DEBUG as the only source of input and output to assembled programs.

Figure 5.11 illustrates a program that uses INT 21H for keyboard, printer, and display communication. The program first clears the screen. Next it writes the message

Input characters (terminate with ENTER)

to the screen using function 9. This function outputs an entire string that is pointed to by the starting address in DX. The string, in this case, is defined in the data segment starting at MESOUT. In order to delimit the string, DOS looks for a $ as the last character. Once the string has been passed to the display, function 2 is called for control. Next the keyboard is read using function 0AH and loaded into CHAR in the data segment. CHAR is preceded by two bytes in the data segment: BUFFMX, which is initialized to the maximum character count allowed in the input string, and BUFFLN, a byte that will contain the number of characters read from the keyboard. After more screen control instructions, the program modifies the string for output to the screen. The last character input in the string was a carriage return and

TABLE 5.4 INT 21H OPTIONS (DOS FUNCTION CALLS)

AH	Purpose	Type	Description
0	Program Terminate	Control	Terminates the execution of a program.
1	Keyboard Input	Keyboard	Waits for keyboard input, displays it, and returns it in AL.
2	Display Output	Display	Displays the character in DL.
3	Auxiliary Input	Misc.	Waits for a character from the COM port and puts it in AL.
4	Auxiliary Output	Misc.	Outputs the character in DL to the COM port.
5	Printer Output	Printer	Outputs the character in DL to the printer.
6	Direct Console I/O	Keyboard	Waits for a character from the keyboard (no Ctrl-Break check).
7	Direct Console Input with No Echo	Keyboard	Waits for a character from the keyboard and puts it in AL.
8	Console Input without Echo	Keyboard	Waits for a character from the keyboard, returns it in AL, and executes an interrupt for Ctrl-Break.
9	Print String	Display	Outputs the string to the display. String must end with $.
A	Buffered Keyboard Input	Keyboard	Reads characters from the keyboard into a buffer. DS:DX points to buffer. 1st byte = max characters, and 2nd byte = number characters read.
B	Check Standard Input Status	Keyboard	Checks to see if a character is available from the keyboard.
C	Clear Keyboard Buffer and Invoke Keyboard Function	Keyboard	Clears the keyboard buffer and executes the function call in AL (only 01H, 06H, 07H, 08H, or 0AH).
D	Disk Reset	Disk	All files not closed are lost.
E	Select Disk	Disk	Selects the drive in DL as default (0 = A, 1 = B, etc.).
F	Open File	File	Searches the directory for the file pointed to in DS:DX. AL = FFH (not found) or 00H (found). If found the FCB is filled.
10	Close File	File	Closes the file after a write. DS:DX points to FCB.
11	Search for First Entry	Disk	Searches the directory for the first matching filename. AL = FFH if none found.
12	Search for Next Entry	Disk	After a filename has been found, this call searches for the next occurrence.
13	Delete File	File	Deletes all directory entries that match DS:DX pointer.
14	Sequential Read	Disk	Loads the record addressed by the current block and record at the DTA and increments the record address.
15	Sequential Write	Disk	Opposite 14H.
16	Create File	File	Searches the directory for a matching entry, if found it is reused, if not found opens a file.
17	R[illegible]me File	File	[illegible]

19	Current Disk	Disk	Determines the default drive and returns it in AL.
1A	Set Disk DTA	Disk	Sets the disk transfer address to DS:DX.
1B	Allocation Table Information	Disk	Returns DS:BX = pointer to media descriptor byte, DX = number allocation units, AL = numbersectors/allocation unit, and CX = size physical sector.
1C	Allocation Table Info for Drive	Disk	DL = drive number; this function returns the same parameters as 1CH.
21	Random Read	Disk	Reads the record addressed by the current block and record fields into memory at the DTA.
22	Random Write	Disk	Opposite 21H.
23	File Size	File	Searches the directory for entry matching DS:DX and sets the FCB random record field equal to the number records in file.
24	Set Relative Record Field	File	Sets the random record field to the same address as the current block and record fields.
25	Set Interrupt Vector	Misc.	The interrupt vector in AL is set to address DS:DX.
26	Create New Program Segment	Misc.	This call should not be used.
27	Random Block Read	Disk	Reads the number of records in CX from DS:DX into DTA.
28	Random Block Write	Disk	Opposite 27H.
29	Parse Filename	File	See *DOS Technical Reference* manual.
2A	Get Date	Misc.	Returns: AL = day of week, CX = year, DH = month, and DL = day of month.
2B	Set Date	Misc.	Reverse 2AH.
2C	Get Time	Misc.	Returns: CH = hour, CL = minutes, DH = seconds, and DL = hundredths of a second.
2D	Set Time	Misc.	Reverse 2CH.
2E	Set/Reset Verify Switch	Misc.	When set, DOS performs a verify operation for each disk write. AL = 0 (off) and AL = 1 (on).
2F	Get DTA	Disk	Returns the disk transfer address in ES:BX.
30	Get DOS Version No.	Misc.	Returns: DOS major version number (AL) and minor version (AH).
31	Terminate Process/Remain Resident	Control	See *DOS Technical Reference* manual.
33	Ctrl-Break Check	Control	Requests/sets BREAK, AL = 0(request) or 1(set) and DL = 0 (off) or 1 (on).
35	Get Vector	Misc.	For interrupt number in AL, it returns the pointer in ES:BX.
36	Get Disk Free Space	Disk	Returns for DL (drive) the available clusters (BX), clusters/drive (DX), bytes/sector (CX), and sectors/cluster (AX).
38	Country Dependent Information	Misc.	See *DOS Technical Reference* manual.
39	Create Subdirectory	Disk	Generates the MKDIR function with DS:DX pointing to an ASCIIZ string containing drive

2.5.1 General Structure of the i386

The i386 also follows the general structure of other microprocessors, as shown in Figure 2.3. Besides the control unit it has several registers. Additionally, several registers for memory management are available which are, however, important only in protected mode. Figure 2.5 shows all the implemented registers.

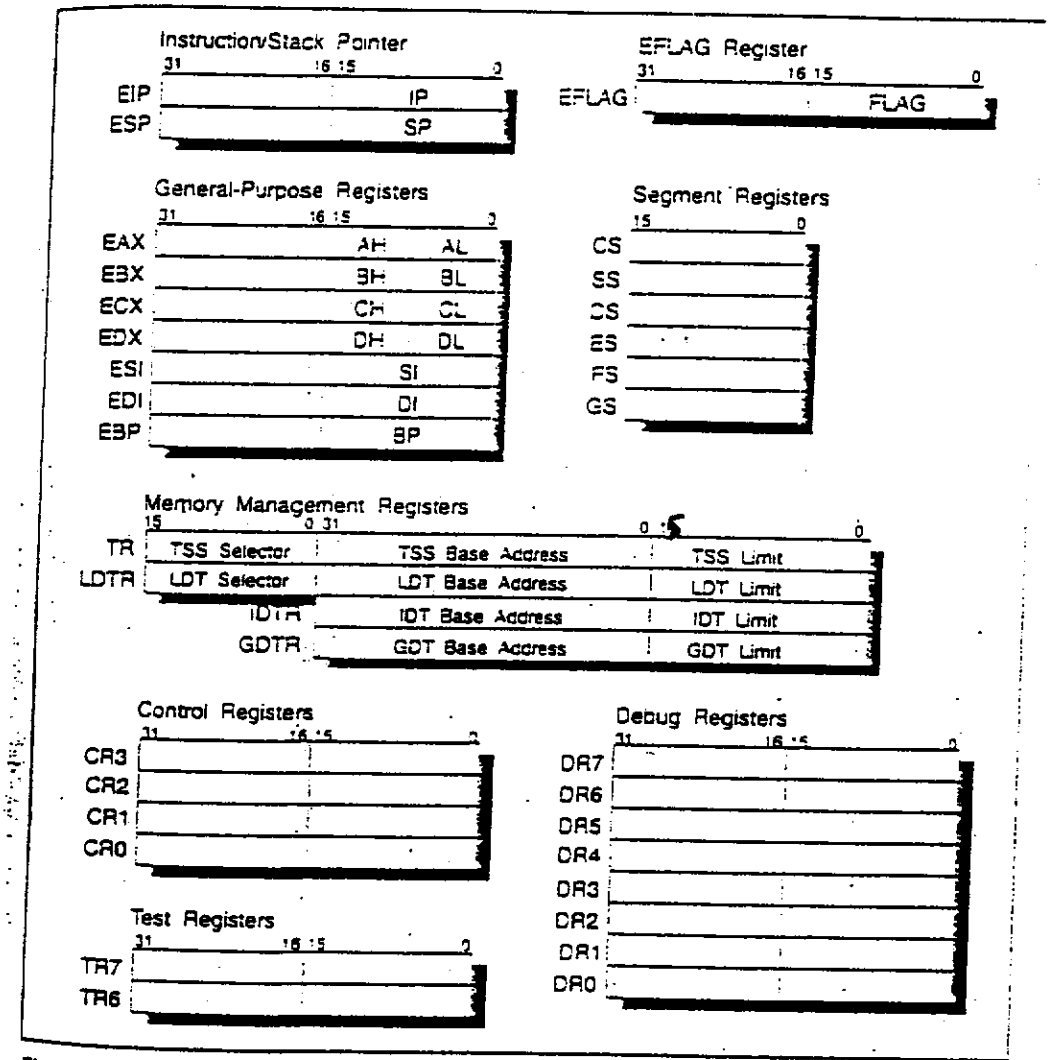

Figure 2.5: i386 processor registers. The general-purpose registers of the i386 are 32 bits wide, but can also be accessed as 16-bit or 8-bit registers. Additionally, there are six segment registers and an instruction pointer, which addresses the instruction to be executed next, as well as a flag register storing the current processor status flags and various control and test registers.

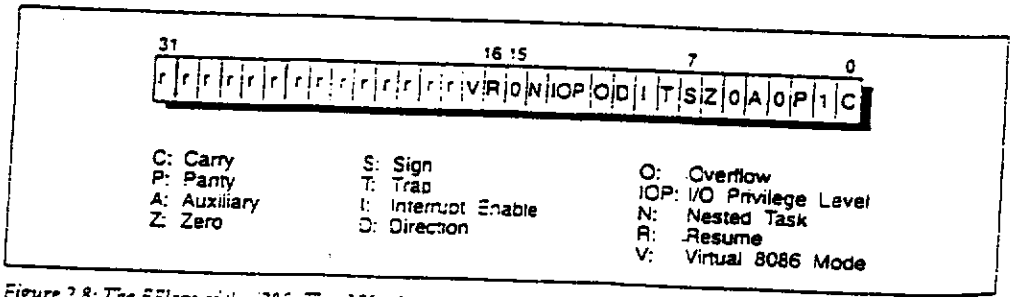

Figure 2.8: The EFlags of the i386. The i386 microprocessor comprises several flags which indicate the result of the previous operation or the current processor status. With the new operation modes of the i386, the number of flags has also increased compared to the 8086.