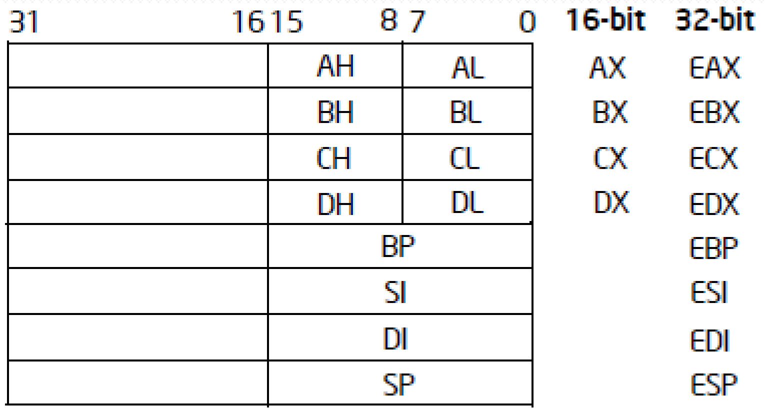
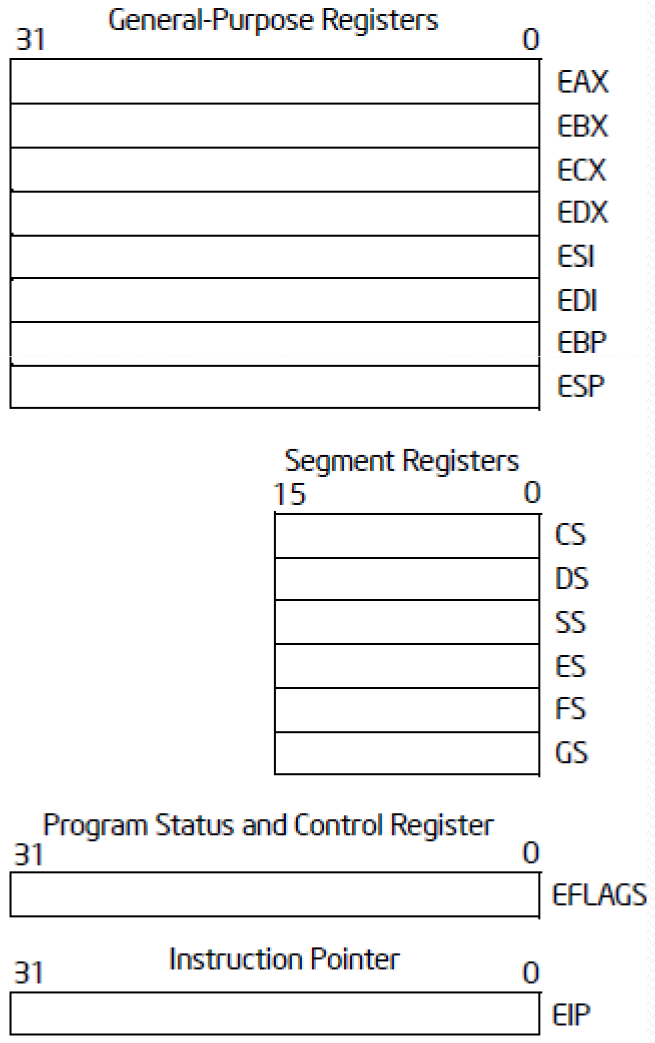
EC325 Microprocessors Introduction to Assembly

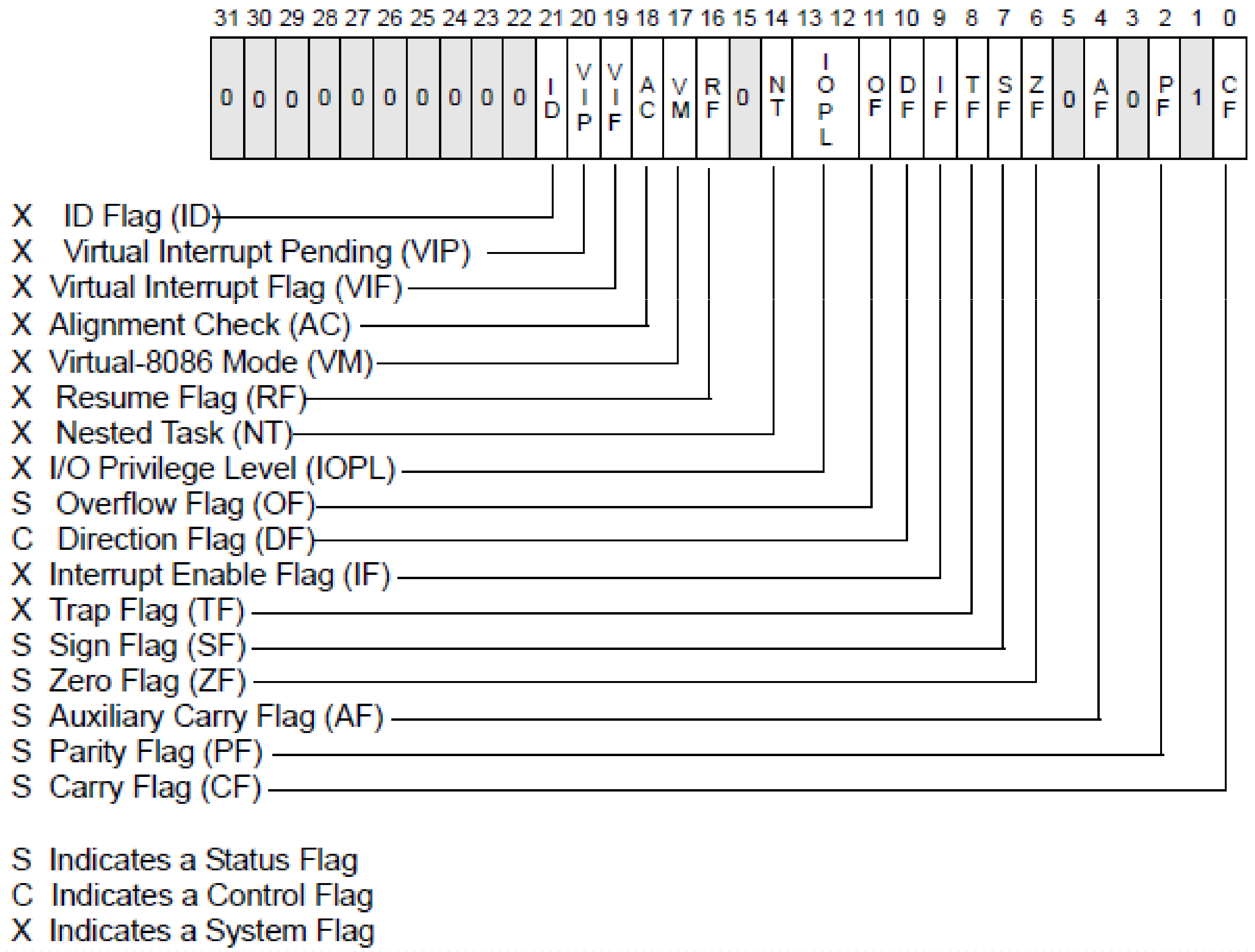
Yasser F. O. Mohammad

REMIDER 1: GPRs (32 bits)

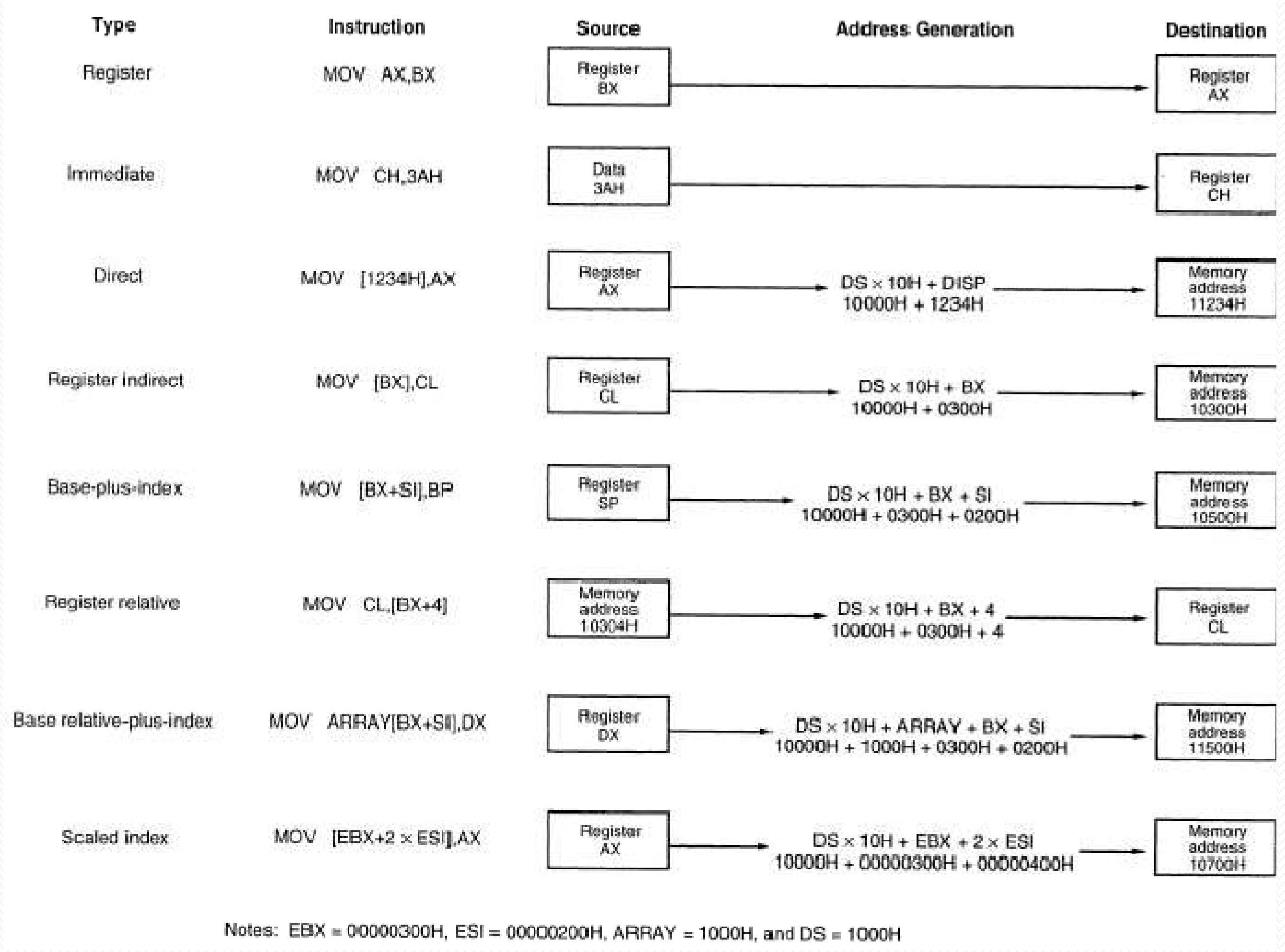




REMINDER 2: EFLAGS



REMINDER 3:Addressing Modes



Type	Instruction	Source	Address Generation	Destination
Register	MOV AX,BX	Register BX		Register AX
Immediate	MOV CH,3AH	Data 3AH		Register CH
Direct	MOV [1234H],AX	Register AX	DS × 10H + DISP 10000H + 1234H	Memory address 11234H
Register indirect	MOV [BX],CL	Register CL	DS × 10H + BX 10000H + 0300H	Memory address 10300H
Base-plus-index	MOV [BX+SI],BP	Register SP	DS × 10H + BX + SI 10000H + 0300H + 0200H	Memory address 10500H
Register relative	MOV CL,[BX+4]	Memory address 10304H	DS × 10H + BX + 4 10000H + 0300H + 4	Register CL
Base relative-plus-index	MOV ARRAY[BX+SI],DX	Register DX	DS × 10H + ARRAY + BX + SI 10000H + 1000H + 0300H + 0200H	Memory address 11500H
Scaled index	MOV [EBX+2 × ESI],AX	Register AX	DS × 10H + EBX + 2 × ESI 10000H + 00000300H + 00000400H	Memory address 10700H

Notes: EBX = 00000300H, ESI = 00000200H, ARRAY = 1000H, and DS = 1000H

Advantages of Assembly

- Full Control
- Understanding the microprocessor
- Appreciating Compilers!!
- Code for tiny microcontrollers (8-bits)

Disadvantages of Assembly

- Complexity
- Mis-optimization!!

Contents of Assembly File

- Statements
  - Up to 512 chars (MASM 6.1)
  - Can be multiline with ‘\’ added at the end of each (except last)

- Comments
  - Starts with a semicolon ‘;’ and extends to EOL

Types of Statements

- Instructions
  - Translated to machine code by the assembler
  - Example: add AX,10
- Directives
  - Command to the assembler (not translated)
  - Example: .NOLIST
- Macros
  - Shorthand to a sequence of statements (not directly translated)

Statement Anatomy

- [name]   [mnemonic]   [operand(s)]   [; comment]
- Zerocount: mov   ecx,0   ; initialize counter

- Name
  - Ends with colon ':' for instructions but not directives

- Mnemonic
  - Indicates what the statement is about

- Operands
  - Optional and separated with comas

- Comment
  - Optional and starts with a semicolon

Identifier rules

- Cannot start with a number
- Allowable special characters (‘_’, ‘?’, ‘$’, ‘@’)
- Up to 247 characters
- Cannot be a reserved name

Our rules

- A comment EVERY 1 to 2 lines

- All names start at column 1
- All mnemonics start at column 13

- All operands start at column 21

Our First Program

File Header

Prototypes

Macros

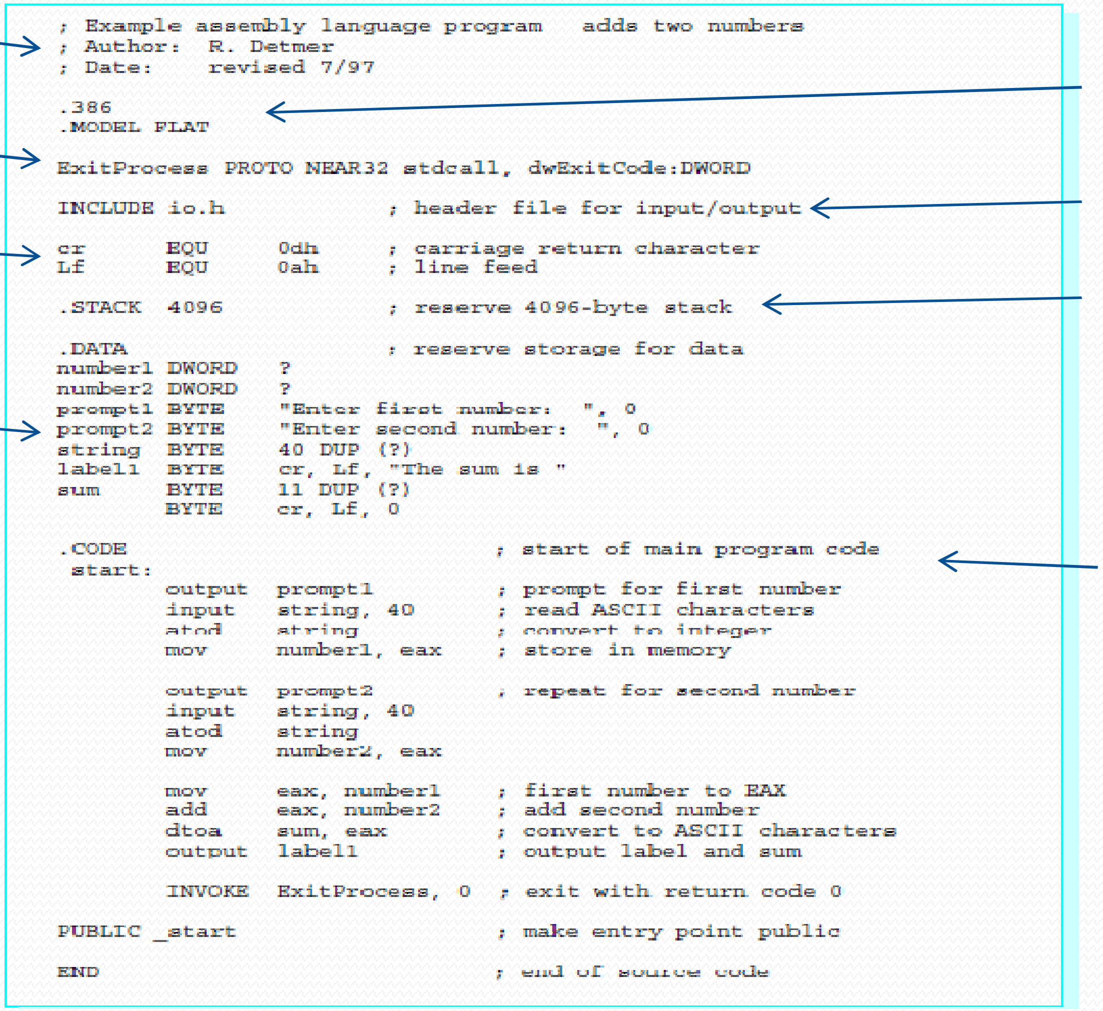
Data Alloc.

Directives

Inclusion

Stack Alloc.

CODE



```
; Example assembly language program   adds two numbers
; Author:  R. Detmer
; Date:    revised 7/97

.386
.MODEL FLAT

ExitProcess PROTO NEAR32 stdcall, dwExitCode:DWORD

INCLUDE io.h            ; header file for input/output

cr      EQU     0dh     ; carriage return character
Lf      EQU     0ah     ; line feed

.STACK  4096            ; reserve 4096-byte stack

.DATA                   ; reserve storage for data
number1 DWORD   ?
number2 DWORD   ?
prompt1 BYTE    "Enter first number:  ", 0
prompt2 BYTE    "Enter second number:  ", 0
string  BYTE    40 DUP (?)
label1  BYTE    cr, Lf, "The sum is "
sum     BYTE    11 DUP (?)
        BYTE    cr, Lf, 0

.CODE                           ; start of main program code
 start:
        output  prompt1         ; prompt for first number
        input   string, 40      ; read ASCII characters
        atod    string          ; convert to integer
        mov     number1, eax    ; store in memory

        output  prompt2         ; repeat for second number
        input   string, 40
        atod    string
        mov     number2, eax

        mov     eax, number1    ; first number to EAX
        add     eax, number2    ; add second number
        dtoa    sum, eax        ; convert to ASCII characters
        output  label1          ; output label and sum

        INVOKE  ExitProcess, 0  ; exit with return code 0

PUBLIC _start                   ; make entry point public

END                             ; end of source code
```

Instruction Set Selection

- Default
  - 8086/8088
- .386
  - Until 80386 without privileged instructions
- .386P
  - 80386 with privileged instructions
- .486
  - Until 80486
- .586
  - Until Pentium

Model

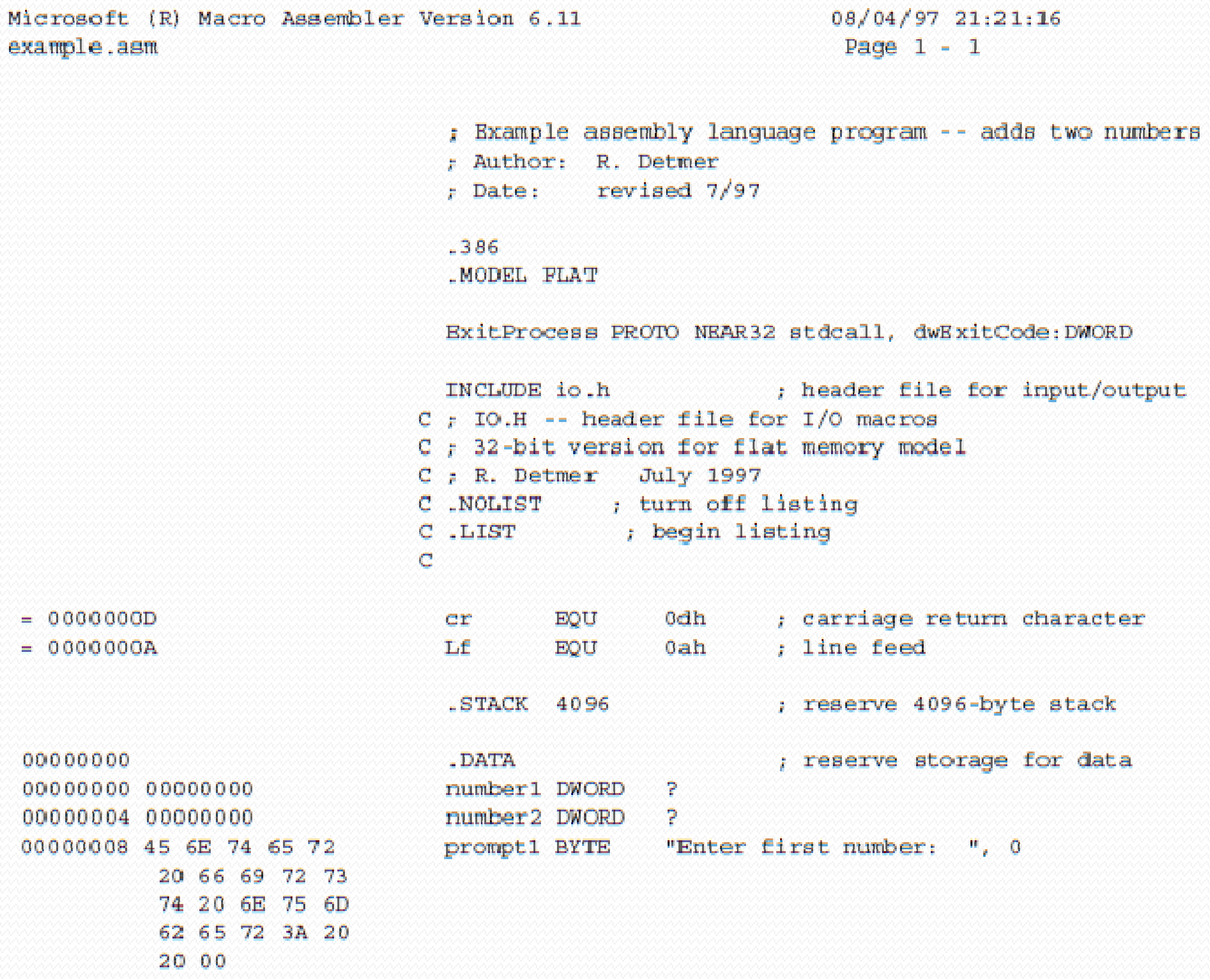
- .MODEL memorymodel [[, langtype]] [[, stackoption]]
- Memorymodel:
  - TINY
  - SMALL
  - COMPACT
  - MEDIUM
  - LARGE
  - HUGE
  - FLAT
- Langtype:
  - C, BASIC, FORTRAN, PASCAL, SYSCALL, or STDCALL
- Stackoption:
  - NEARSTACK or FARSTACK (not allowed with FLAT)

How To Assemble, Run, and Debug

- In the section!!!

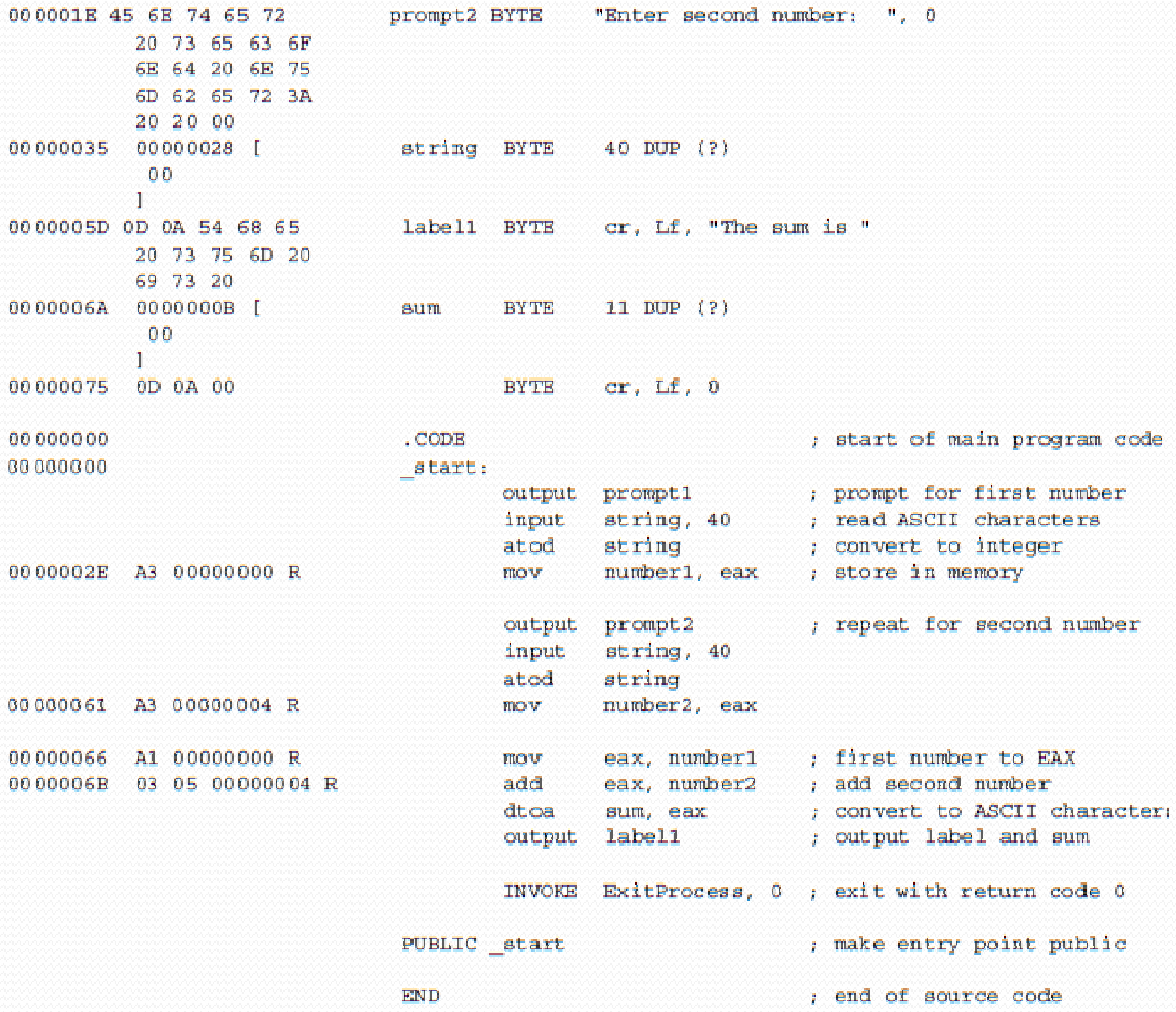
.LST file

- Assemble using /Fl switch



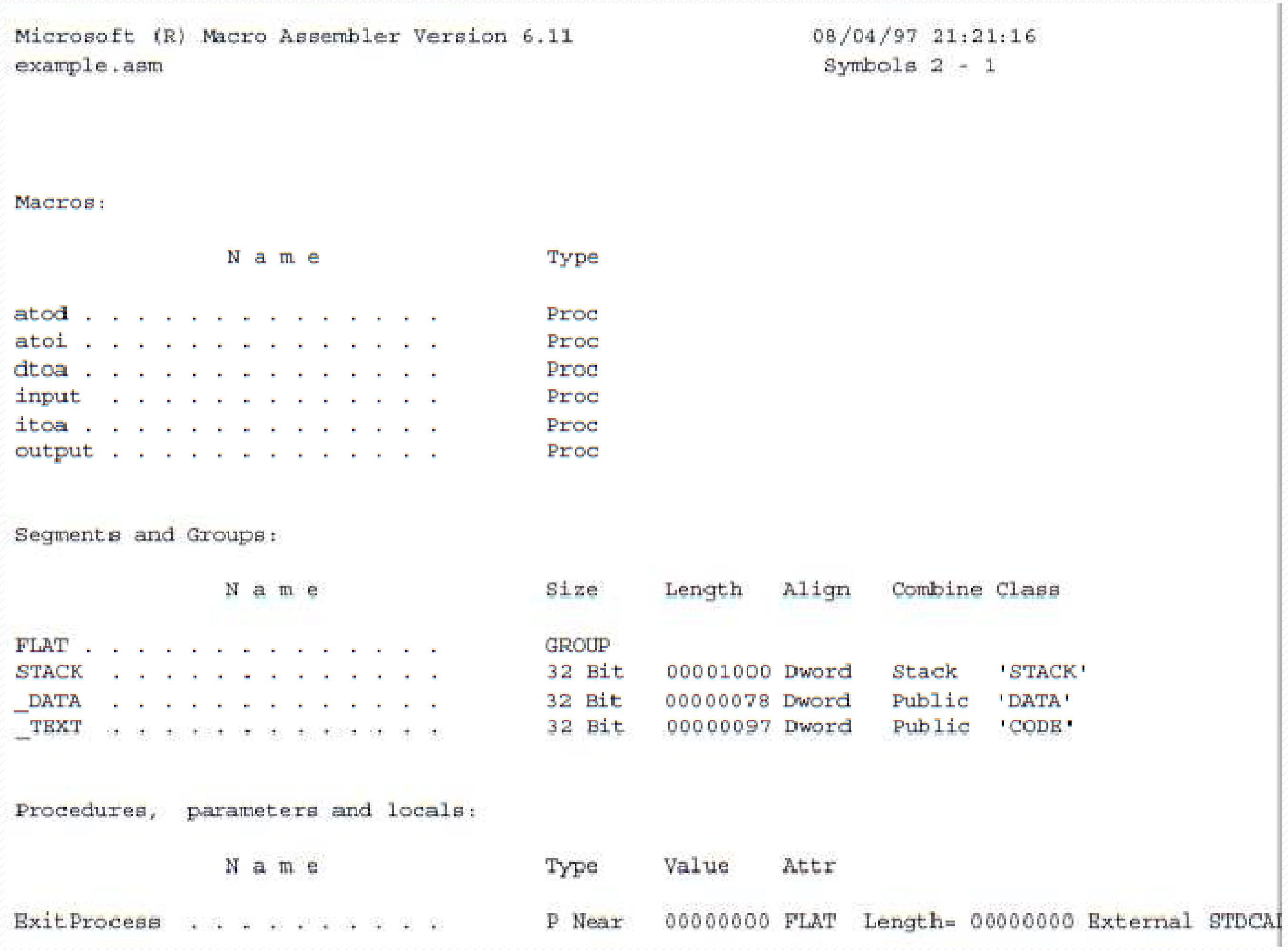
```
Microsoft (R) Macro Assembler Version 6.11                    08/04/97 21:21:16
example.asm                                                    Page 1 - 1


                                 ; Example assembly language program -- adds two numbers
                                 ; Author:  R. Detmer
                                 ; Date:    revised 7/97

                                 .386
                                 .MODEL FLAT

                                 ExitProcess PROTO NEAR32 stdcall, dwExitCode:DWORD

                                 INCLUDE io.h            ; header file for input/output
                               C ; IO.H -- header file for I/O macros
                               C ; 32-bit version for flat memory model
                               C ; R. Detmer   July 1997
                               C .NOLIST     ; turn off listing
                               C .LIST        ; begin listing
                               C

 = 0000000D                      cr      EQU     0dh     ; carriage return character
 = 0000000A                      Lf      EQU     0ah     ; line feed

                                 .STACK  4096            ; reserve 4096-byte stack

 00000000                        .DATA                   ; reserve storage for data
 00000000 00000000               number1 DWORD   ?
 00000004 00000000               number2 DWORD   ?
 00000008 45 6E 74 65 72         prompt1 BYTE    "Enter first number:  ", 0
          20 66 69 72 73
          74 20 6E 75 6D
          62 65 72 3A 20
          20 00
```

.LST



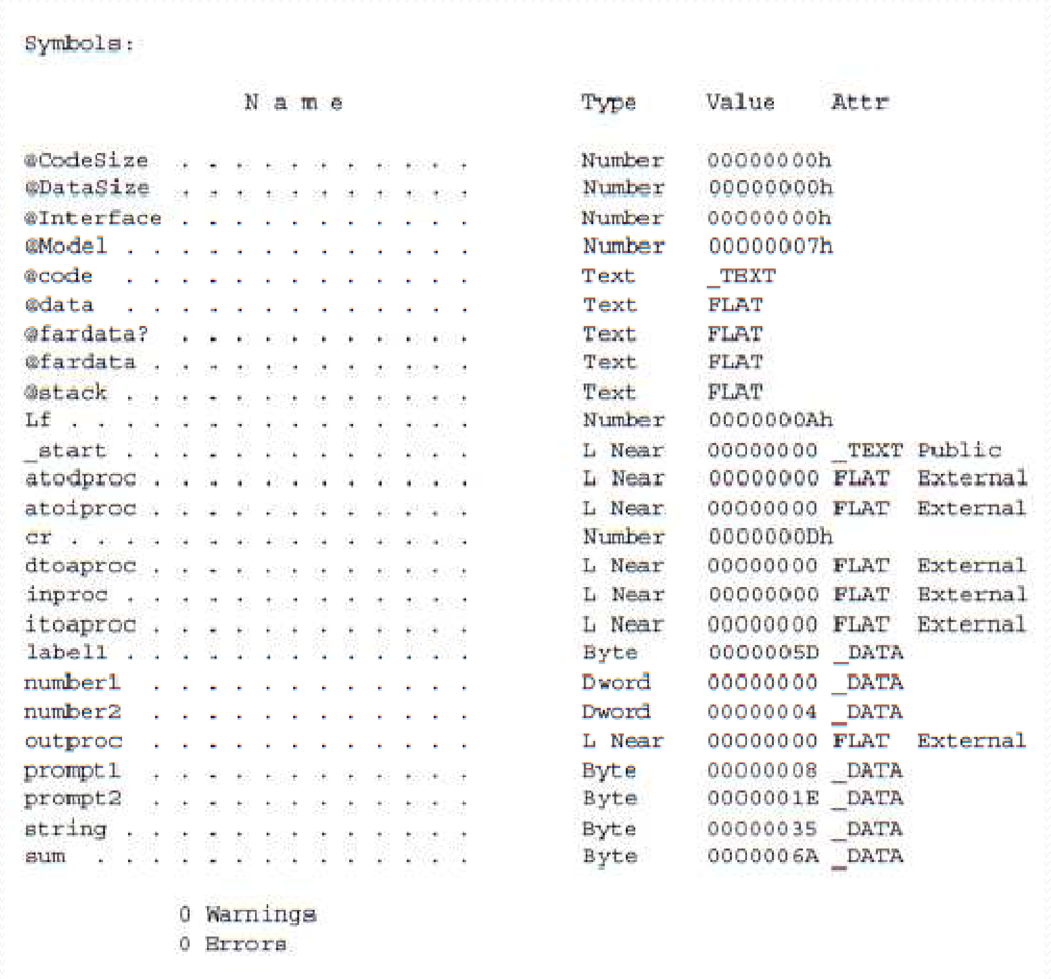
```
00 00001E 45 6E 74 65 72          prompt2 BYTE    "Enter second number:  ", 0
           20 73 65 63 6F
           6E 64 20 6E 75
           6D 62 65 72 3A
           20 20 00
00 000035  00000028 [              string  BYTE    40 DUP (?)
            00
           ]
00 00005D 0D 0A 54 68 65           label1  BYTE    cr, Lf, "The sum is "
           20 73 75 6D 20
           69 73 20
00 00006A  0000000B [              sum     BYTE    11 DUP (?)
            00
           ]
00 000075  0D 0A 00                        BYTE    cr, Lf, 0

00 000000                          .CODE                           ; start of main program code
00 000000                          _start:
                                           output  prompt1         ; prompt for first number
                                           input   string, 40      ; read ASCII characters
                                           atod    string          ; convert to integer
00 00002E  A3 00000000 R                   mov     number1, eax    ; store in memory

                                           output  prompt2         ; repeat for second number
                                           input   string, 40
                                           atod    string
00 000061  A3 00000004 R                   mov     number2, eax

00 000066  A1 00000000 R                   mov     eax, number1    ; first number to EAX
00 00006B  03 05 00000004 R                add     eax, number2    ; add second number
                                           dtoa    sum, eax        ; convert to ASCII character
                                           output  label1          ; output label and sum

                                           INVOKE  ExitProcess, 0  ; exit with return code 0

                                   PUBLIC _start                   ; make entry point public

                                   END                             ; end of source code
```

.LST



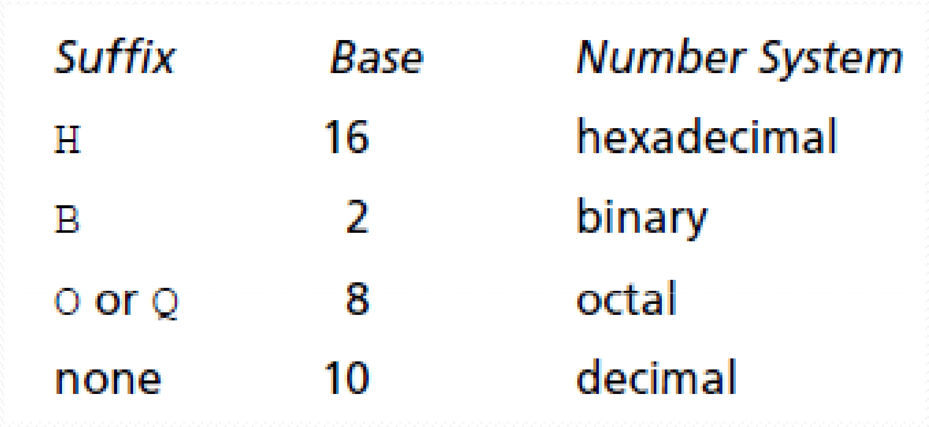
```
Microsoft (R) Macro Assembler Version 6.11                 08/04/97 21:21:16
example.asm                                                 Symbols 2 - 1



Macros:

                N a m e                 Type

atod . . . . . . . . . . . . . .        Proc
atoi . . . . . . . . . . . . . .        Proc
dtoa . . . . . . . . . . . . . .        Proc
input  . . . . . . . . . . . . .        Proc
itoa . . . . . . . . . . . . . .        Proc
output . . . . . . . . . . . . .        Proc


Segments and Groups:

                N a m e                 Size     Length   Align    Combine Class

FLAT . . . . . . . . . . . . . .        GROUP
STACK  . . . . . . . . . . . . .        32 Bit   00001000 Dword    Stack    'STACK'
_DATA  . . . . . . . . . . . . .        32 Bit   00000078 Dword    Public   'DATA'
_TEXT  . . . . . . . . . . . . .        32 Bit   00000097 Dword    Public   'CODE'


Procedures,  parameters and locals:

                N a m e                 Type     Value    Attr

ExitProcess  . . . . . . . . . .        P Near   00000000 FLAT  Length= 00000000 External STDCAL
```

.LST



```
Symbols:

                N a m e                 Type     Value    Attr

@CodeSize . . . . . . . . . . .         Number   00000000h
@DataSize . . . . . . . . . . .         Number   00000000h
@Interface . . . . . . . . . . .        Number   00000000h
@Model . . . . . . . . . . . . .        Number   00000007h
@code  . . . . . . . . . . . . .        Text     _TEXT
@data  . . . . . . . . . . . . .        Text     FLAT
@fardata? . . . . . . . . . . .         Text     FLAT
@fardata . . . . . . . . . . . .        Text     FLAT
@stack . . . . . . . . . . . . .        Text     FLAT
Lf . . . . . . . . . . . . . . .        Number   0000000Ah
_start . . . . . . . . . . . . .        L Near   00000000 _TEXT  Public
atodproc . . . . . . . . . . . .        L Near   00000000 FLAT   External
atoiproc . . . . . . . . . . . .        L Near   00000000 FLAT   External
cr . . . . . . . . . . . . . . .        Number   0000000Dh
dtoaproc . . . . . . . . . . . .        L Near   00000000 FLAT   External
inproc . . . . . . . . . . . . .        L Near   00000000 FLAT   External
itoaproc . . . . . . . . . . . .        L Near   00000000 FLAT   External
label1 . . . . . . . . . . . . .        Byte     0000005D _DATA
number1  . . . . . . . . . . . .        Dword    00000000 _DATA
number2  . . . . . . . . . . . .        Dword    00000004 _DATA
outproc  . . . . . . . . . . . .        L Near   00000000 FLAT   External
prompt1  . . . . . . . . . . . .        Byte     00000008 _DATA
prompt2  . . . . . . . . . . . .        Byte     0000001E _DATA
string . . . . . . . . . . . . .        Byte     00000035 _DATA
sum  . . . . . . . . . . . . . .        Byte     0000006A _DATA

           0 Warnings
           0 Errors
```

Constants



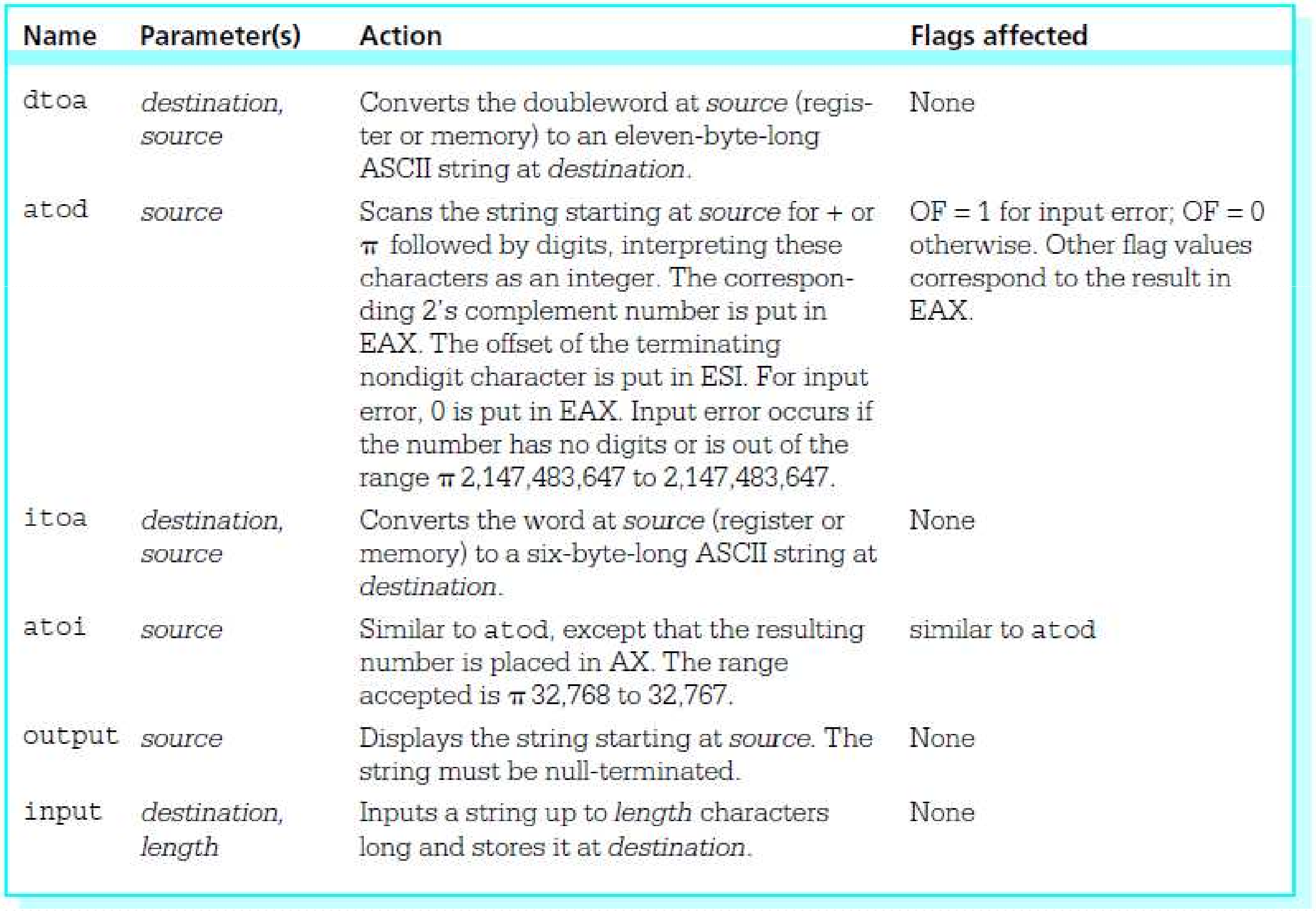
Suffix	*Base*	*Number System*
H	16	hexadecimal
B	2	binary
O or Q	8	octal
none	10	decimal

‘A’ means the ASCII code of *A*

Data Reservation

- name STORAGE_SIZE [value | n DUP(init)] [,value | n DUP(init) …]
- Name
  - Identifier
- STORAGE_SIZE
  - BYTE, WORD, DWORD, QWORD, TBYTE, REAL4, REAL8, REAL10
- DUP = duplicate *n* times and initialize to *init*
- *"abc"* is a shorthand to 'a','b','c'

Macros in IO.H



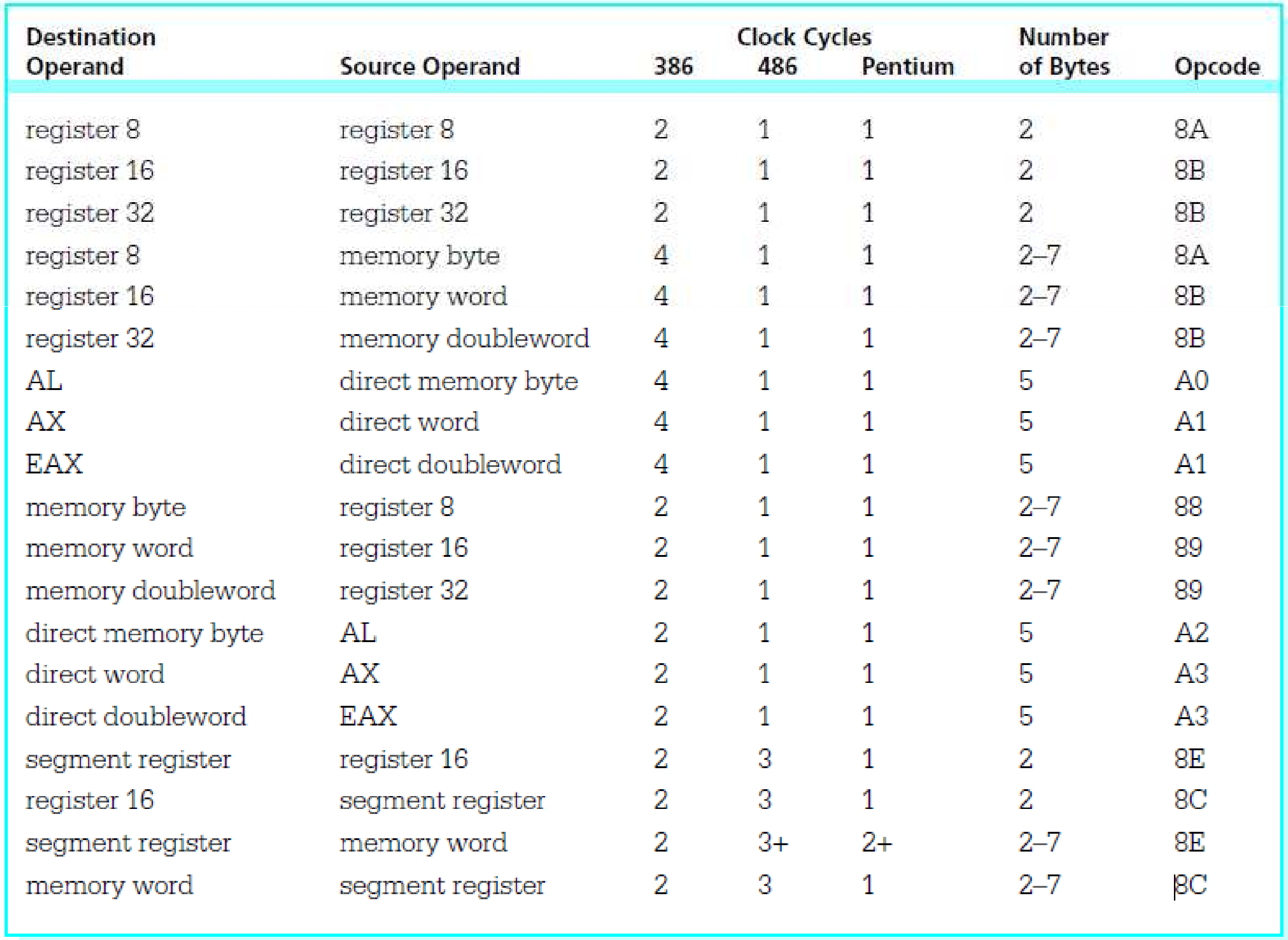
Name	Parameter(s)	Action	Flags affected
`dtoa`	*destination, source*	Converts the doubleword at *source* (register or memory) to an eleven-byte-long ASCII string at *destination*.	None
`atod`	*source*	Scans the string starting at *source* for + or π followed by digits, interpreting these characters as an integer. The corresponding 2's complement number is put in EAX. The offset of the terminating nondigit character is put in ESI. For input error, 0 is put in EAX. Input error occurs if the number has no digits or is out of the range π 2,147,483,647 to 2,147,483,647.	OF = 1 for input error; OF = 0 otherwise. Other flag values correspond to the result in EAX.
`itoa`	*destination, source*	Converts the word at *source* (register or memory) to a six-byte-long ASCII string at *destination*.	None
`atoi`	*source*	Similar to `atod`, except that the resulting number is placed in AX. The range accepted is π 32,768 to 32,767.	similar to `atod`
`output`	*source*	Displays the string starting at *source*. The string must be null-terminated.	None
`input`	*destination, length*	Inputs a string up to *length* characters long and stores it at *destination*.	None

MOV

- mov destination , source
- Immediate:
  - mov EAX, 5

- Direct:
  1. mov EAX, [5]
  2. mov EAX,num1 ; num1 DWORD ?

MOV Times



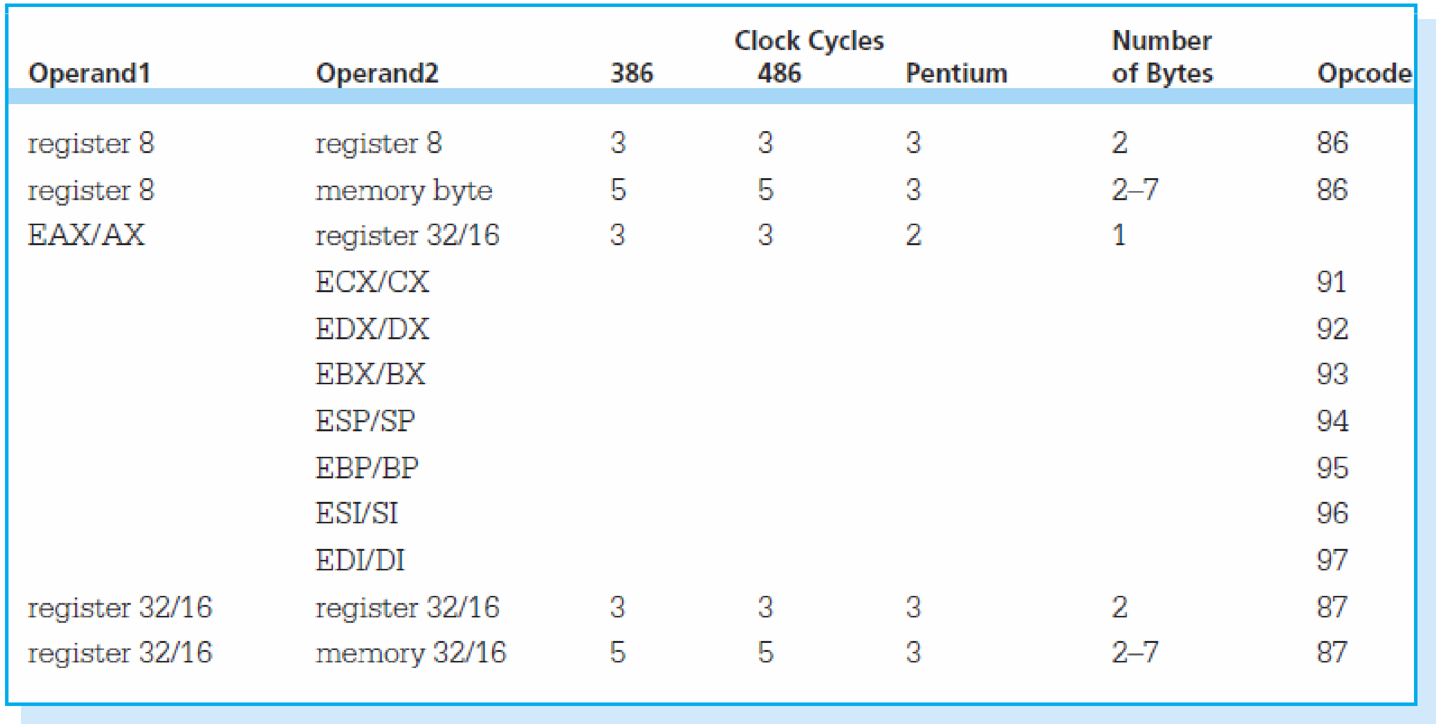
Destination Operand	Source Operand	Clock Cycles 386	486	Pentium	Number of Bytes	Opcode
register 8	register 8	2	1	1	2	8A
register 16	register 16	2	1	1	2	8B
register 32	register 32	2	1	1	2	8B
register 8	memory byte	4	1	1	2–7	8A
register 16	memory word	4	1	1	2–7	8B
register 32	memory doubleword	4	1	1	2–7	8B
AL	direct memory byte	4	1	1	5	A0
AX	direct word	4	1	1	5	A1
EAX	direct doubleword	4	1	1	5	A1
memory byte	register 8	2	1	1	2–7	88
memory word	register 16	2	1	1	2–7	89
memory doubleword	register 32	2	1	1	2–7	89
direct memory byte	AL	2	1	1	5	A2
direct word	AX	2	1	1	5	A3
direct doubleword	EAX	2	1	1	5	A3
segment register	register 16	2	3	1	2	8E
register 16	segment register	2	3	1	2	8C
segment register	memory word	2	3+	2+	2–7	8E
memory word	segment register	2	3	1	2–7	8C

MOVs not allowed

- Source and destination in memory
- To and from FLAG register
- To and from IP
- Source and destination are segment registers
- Immediate to segment register
- Operands are not same size

XCHG

- xchg   source1,source2
- Exchange the two values
- Equals 3 moves



			Clock Cycles		Number	
Operand1	Operand2	386	486	Pentium	of Bytes	Opcode
register 8	register 8	3	3	3	2	86
register 8	memory byte	5	5	3	2–7	86
EAX/AX	register 32/16	3	3	2	1	
	ECX/CX					91
	EDX/DX					92
	EBX/BX					93
	ESP/SP					94
	EBP/BP					95
	ESI/SI					96
	EDI/DI					97
register 32/16	register 32/16	3	3	3	2	87
register 32/16	memory 32/16	5	5	3	2–7	87