

Ассемблер Intel 8086

Определение идентификаторов

Для присваивания символических имён различным выражениям существуют специальные директивы.

1. **EQU** – директива, предназначенная для присваивания символического имени символам, переменным, текстам, выражениям.

Примеры:

```
K EQU 1024 ; K = 1024 = const
TABLE EQU DS:[BP+SI] ; TABLE – это текст
SPEED EQU RATE ; SPEED и RATE – синонимы
COUNTER EQU CX
DOUBLE_SPEED EQU 2*SPEED
END_OF_DATA EQU ‘!’
```

После таких определений можно записать:

```
mov AX, K ; mov AX, 1024
mov TABLE, BX ; mov DS:[BP+SI], BX
cmp AL, END_OF_DATA ; cmp AL, ‘!’
```

Ассемблер Intel 8086

Определение идентификаторов

2. `=` – директива, предназначенная для присваивания символического имени только числовым выражениям.

Пример:

```
TEMP = 10
DW TEMP      ; DW 10
TEMP = TEMP + 10
DW TEMP      ; DW 20
```

Ассемблер Intel 8086

Определение идентификаторов

Между директивами EQU и = есть **важное отличие**:

EQU определяет символьное имя, которое не может быть переопределено в программе;

директива = предназначена только для числовых выражений, но символьные имена, созданные с помощью этой директивы могут переопределяться в программе.

Пример:

```
TEMP = 10
DW TEMP      ; DW 10
TEMP = TEMP + 10
DW TEMP      ; DW 20
```

Ассемблер Intel 8086

Многомодульные программы

Реальный проект состоит из нескольких модулей. Для согласования модулей между собой используются следующие директивы:

- 1) **PUBLIC** – указывает на метки текущего модуля, к которым могут иметь доступ другие модули проекта:
PUBLIC <метка> [, <метка>]
- 2) **EXTRN** – объявление меток из других модулей, которые необходимы для работы данного модуля:
EXTRN <объявление> [, <объявление>]
где <объявление> - запись вида <метка>:<тип>
- 3) **GLOBAL** – директива, которая интерпретируется как PUBLIC, если объект определён в данном модуле, и как EXTRN, если определение объекта в данном модуле отсутствует:
GLOBAL <объявление> [, <объявление>]
где <объявление> - запись вида <метка>:<тип>, по формату совпадающая с такой записью в директиве EXTRN
- 4) **INCLUDE** – включение содержимого указанного файла в текущий файл:
INCLUDE <имя файла>

Ассемблер Intel 8086

Многомодульные программы

В качестве типа принимаемого объекта могут указываться следующие:

ABS – имя постоянной величины;

BYTE – имя переменной величины байтового типа (1 байт);

WORD – имя переменной величины типа WORD (2 байта);

DWORD – имя переменной величины типа DWORD (4 байта);

FWORD – имя переменной величины типа FWORD (6 байтов);

QWORD – имя переменной величины типа QWORD (8 байтов);

TWORD – имя переменной величины типа TWORD (10 байтов);

NEAR – имя ближней процедуры или команды;

FAR – имя дальней процедуры или команды.

Ассемблер Intel 8086

Многомодульные программы: пример

;первый модуль

.Data

PUBLIC MemVar, Array1, Array_Length

Array_Length EQU 100

MemVar DW 10

Array1 DB Array_Length DUP(?)

...

.Code

PUBLIC NearProc, FarProc

NearProc PROC Near

...

NearProc ENDP

FarProc PROC Far

...

FarProc ENDP

;второй модуль

.Data

EXTRN MemVar: WORD, Array1: BYTE,
Array_Length: ABS

...

.Code

EXTRN NearProc: NEAR, FarProc: FAR

...

mov ax, [MemVar]

mov bx, OFFSET Array1

mov cx, Array_Length

...

call NearProc

...

call FarProc

```
;первый модуль
dosseg
.model small
.stack 200h
.data

.code

PUBLIC p1

p1 Proc
    push bp
    mov bp,sp
    mov ax,word[bp+6]
    add ax,word[bp+4]
    mov bx,word[bp+2]
    mov [bx],ax
    pop bp
    ret 6
p1 endp

end
```

```
;второй модуль
dosseg
.model small
.stack 200h
.data
    a dw ?
    b dw ?
    c dw ?
    q dw ?
    r dw ?
    e dw ?

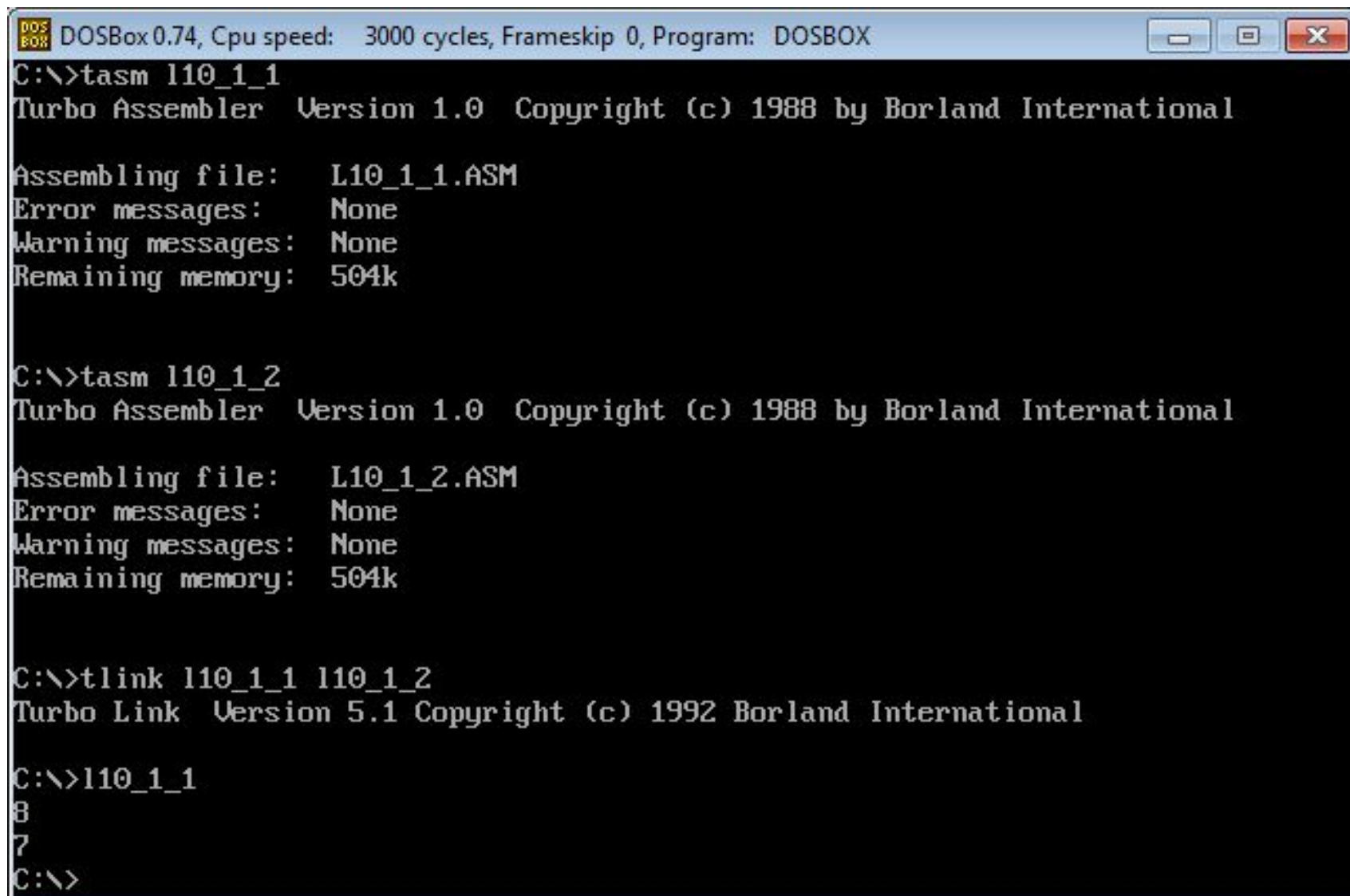
.code

EXTRN p1: NEAR

Begin:
    mov ax,@Data
    mov ds, ax

    mov a,3
    mov b,5
    push a
    push b
    mov ax,OFFSET c
    push ax
    call p1
```

Многомодульные программы: пример



```
DOSBOX 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
C:\>tasm l10_1_1
Turbo Assembler Version 1.0 Copyright (c) 1988 by Borland International

Assembling file: L10_1_1.ASM
Error messages: None
Warning messages: None
Remaining memory: 504k

C:\>tasm l10_1_2
Turbo Assembler Version 1.0 Copyright (c) 1988 by Borland International

Assembling file: L10_1_2.ASM
Error messages: None
Warning messages: None
Remaining memory: 504k

C:\>tlink l10_1_1 l10_1_2
Turbo Link Version 5.1 Copyright (c) 1992 Borland International

C:\>l10_1_1
8
7
C:\>
```



```

;первый модуль
dosseg
.model small
.stack 200h
.data

.code

;PUBLIC p1

p1 Proc
    push bp
    mov bp,sp
    mov ax,word[bp+6]
    add ax,word[bp+4]
    mov bx,word[bp+2]
    mov [bx],ax
    pop bp
    ret 6
p1 endp

end

```

```

;второй модуль
dosseg
.model small
.stack 200h
.data
    a dw ?
    b dw ?
    c dw ?
    q dw ?
    r dw ?
    e dw ?

.code

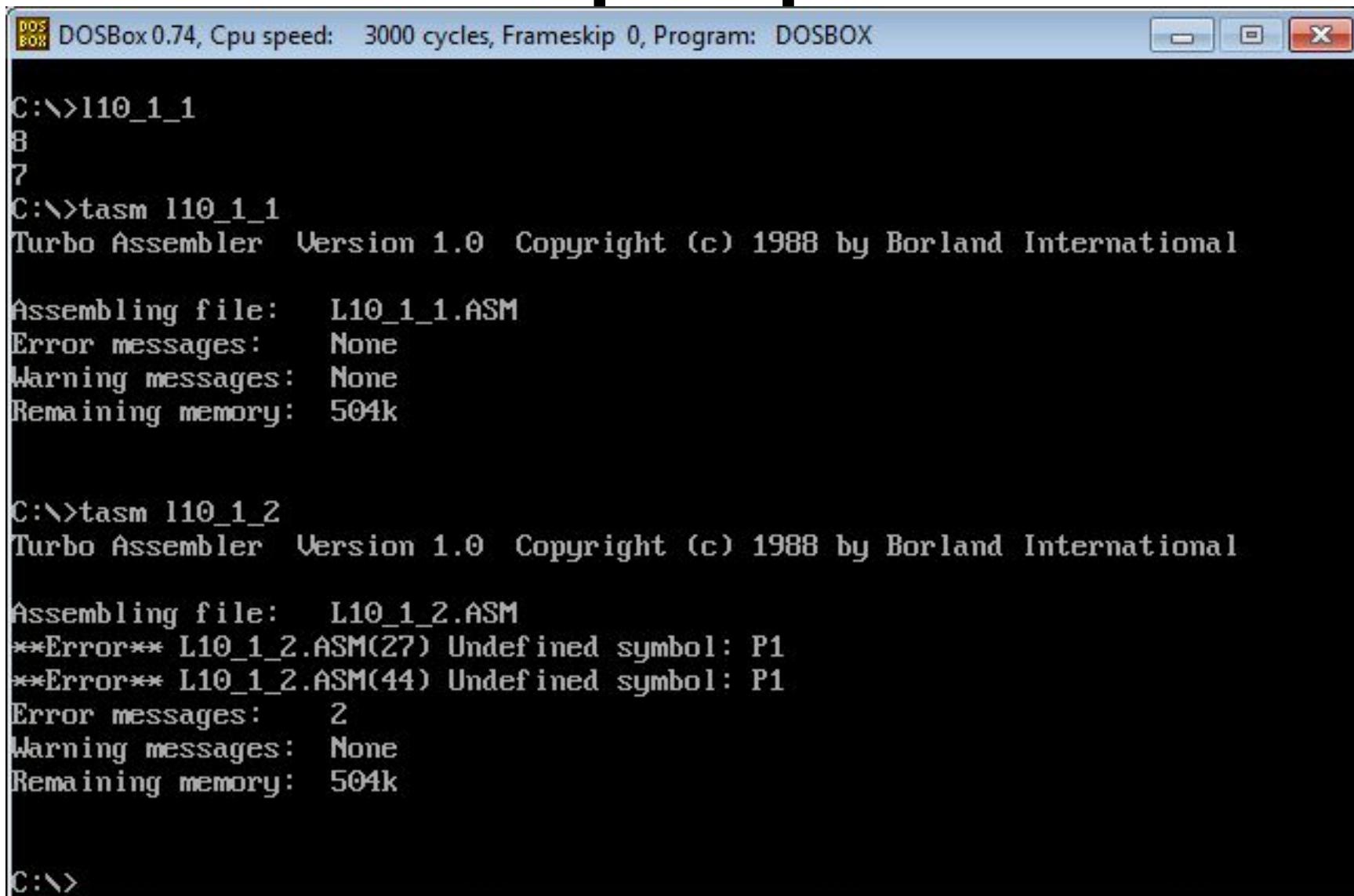
;EXTRN p1: NEAR

Begin:
    mov ax,@Data
    mov ds, ax

    mov a,3
    mov b,5
    push a
    push b
    mov ax,OFFSET c
    push ax
    call p1

```

Многомодульные программы: пример



```
DOS  
BOX DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX  
C:\>l10_1_1  
8  
7  
C:\>tasm l10_1_1  
Turbo Assembler Version 1.0 Copyright (c) 1988 by Borland International  
Assembling file: L10_1_1.ASM  
Error messages: None  
Warning messages: None  
Remaining memory: 504k  
C:\>tasm l10_1_2  
Turbo Assembler Version 1.0 Copyright (c) 1988 by Borland International  
Assembling file: L10_1_2.ASM  
**Error** L10_1_2.ASM(27) Undefined symbol: P1  
**Error** L10_1_2.ASM(44) Undefined symbol: P1  
Error messages: 2  
Warning messages: None  
Remaining memory: 504k  
C:\>
```

Ассемблер Intel 8086

Сегментные директивы

Программа может быть написана с использованием определений каждого сегмента в явном виде. Для этого предусмотрены сегментные директивы:

- 1) **SEGMENT** – указывает начало и атрибуты каждого сегмента программы. Это структурная директива, имеющая следующий вид:

```
label SEGMENT align combine class  
label ENDS
```

где **label** – имя сегмента;

align – тип выравнивания сегмента (BYTE, WORD, DWORD, PARA, PAGE);

combine – способ объединения нескольких сегментов (PRIVATE, PUBLIC, COMMON, STACK, MEMORY, AT exp);

class – имя класса, к которому будет отнесён данный сегмент (заключается в апострофы или кавычки).

Ассемблер Intel 8086

Сегментные директивы

Программа может быть написана с использованием определений каждого сегмента в явном виде. Для этого предусмотрены сегментные директивы:

- 2) **GROUP** – директива, предназначенная для объединения различных сегментов таким образом, чтобы была возможной адресация внутри этих сегментов с помощью одного сегментного регистра, т.е. объединённый сегмент будет занимать не более 64 Кбайт памяти. Директива имеет следующий вид:

```
name GROUP <segname> [, <segname>]
```

где **name** – имя объединённого сегмента;

segname – имена сегментов, которые будут объединены.

- 3) **ASSUME** – описание назначения сегментных регистров. Вид директивы:

```
ASSUME <reg>: <name>[, <reg>: <name>]
```

```
ASSUME <reg>: NOTHING
```

```
ASSUME NOTHING
```

где **reg** – имя сегментного регистра;

name – имя сегмента или группы сегментов.

Ассемблер Intel 8086

Сегментные директивы: пример

```
ASSUME CS: Code
Code SEGMENT
    Fix DB 25
Begin:
    mov ax, Code
    mov ds, ax
    mov al, [Fix]
    mov ah, 4ch
    int 21h
Code ENDS
Stack_seg SEGMENT STACK
    DB 100h DUP(?)
Stack_seg ENDS
END Begin
```

Обращение к переменной Fix требует воспользоваться номером сегмента, в котором объявлена метка Fix. Несмотря на то что во время выполнения в этот номер находится в регистре DS, во время компиляции этот факт не был известен (по умолчанию предполагается директива ASSUME DS: NOTHING), следовательно, была сгенерирована команда с префиксом: `mov al, [cs:Fix]`.

Этого можно избежать, если написать директиву `ASSUME CS: Code, DS:Code`

Сегментные директивы: пример

- ASSUME CS: Code
- Code SEGMENT
- Fix DB 25
- Begin:
 - mov ax, Code
 - mov ds, ax
 - mov al, [Fix]
 - mov ah, 4ch
 - int 21h
- Code ENDS
- Stack_seg SEGMENT STACK
 - DB 100h DUP(?)
- Stack_seg ENDS
- END Begin

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: TD

File Edit View Run Breakpoints Data Options

[] CPU 80486

cs:0001	B8ED48	mov	ax, 48ED	ax 0
cs:0004	8ED8	mov	ds, ax	bx 0
cs:0006	2EA00000	mov	al, cs:[0000]	cx 0
cs:000A	B44C	mov	ah, 4C	dx 0
cs:000C	CD21	int	21	si 0
cs:000E	0000	add	[bx+si], al	di 0
cs:0010	0000	add	[bx+si], al	bp 0
cs:0012	0000	add	[bx+si], al	sp 0
cs:0014	0000	add	[bx+si], al	ds 4
cs:0016	0000	add	[bx+si], al	es 4
cs:0018	0000	add	[bx+si], al	ss 4
cs:001A	0000	add	[bx+si], al	cs 4
cs:001C	0000	add	[bx+si], al	ip 0

ds:0000	CD 20 FF 9F 00 EA FF FF	= f Ω	
ds:0008	AD DE E4 01 C9 15 AE 01	⏏ ⏏ ⏏ ⏏ ⏏ ⏏ ⏏ ⏏	
ds:0010	C9 15 80 02 24 10 92 01	⏏ ⏏ ⏏ ⏏ ⏏ ⏏ ⏏ ⏏	ss:0
ds:0018	01 01 01 00 02 FF FF FF	⏏ ⏏ ⏏ ⏏ ⏏ ⏏ ⏏ ⏏	ss:0

Alt: F2-Bkpt at F3-Close F4-Back F5-User F6-Undo F7-I

Сегментные директивы: пример

- ASSUME CS: Code, DS:Code
- Code SEGMENT
- Fix DB 25
- Begin:
 - mov ax, Code
 - mov ds, ax
 - mov al, [Fix]
 - mov ah, 4ch
 - int 21h
- Code ENDS
- Stack_seg SEGMENT STACK
 - DB 100h DUP(?)
- Stack_seg ENDS
- END Begin

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: TD
File Edit View Run Breakpoints Data Opt
[ ]=CPU 80486
cs:0001 B8ED48 mov ax,48ED
cs:0004 8ED8 mov ds,ax
cs:0006 A00000 mov al,[0000]
cs:0009 B44C mov ah,4C
cs:000B CD21 int 21
cs:000D 0000 add [bx+si],al
cs:000F 0000 add [bx+si],al
cs:0011 0000 add [bx+si],al
cs:0013 0000 add [bx+si],al
cs:0015 0000 add [bx+si],al
cs:0017 0000 add [bx+si],al
cs:0019 0000 add [bx+si],al
cs:001B 0000 add [bx+si],al
ds:0000 CD 20 FF 9F 00 EA FF FF = f R
ds:0008 AD DE E4 01 C9 15 AE 01
ds:0010 C9 15 80 02 24 10 92 01
ds:0018 01 01 01 00 02 FF FF FF
```

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-