|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

*ФАКУЛЬТЕТ «Информатика и системы управления»*

*КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»*

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 02 |

**Название:**

***Защищенный режим.***

**Дисциплина:  *Операционные системы***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ***ИУ7И-52Б*** |  |  | Чыонг Н.В.У. |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | **Рязанова Н. Ю.** |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

*Москва, 2021*

* **Задание:**

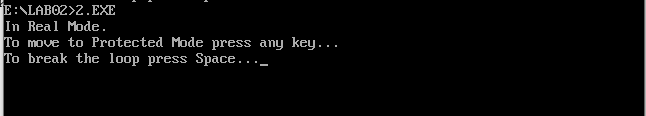
Написать программу, переводящую компьютер в защищенный режим (32-разраядный режим работы компьютеров на базе процессоров Intel).

Программа начинает работать в реальном режиме. Для перевода в защищенный режим выполняются необходимые действия. В защищенном режиме программа работает на нулевом уровне привилегий.

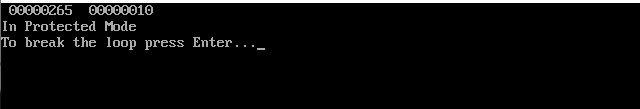
В защищенном режиме программа должна:

* определить объем доступной физической памяти;
* осуществить ввод с клавиатуры строки с выводом введенной строки на экран;
* получить информацию на экране от системного таймера или в виде мигающего курсора, или в виде количества тиков с момента запуска программы на выполнение, или в виде значения реального времени.

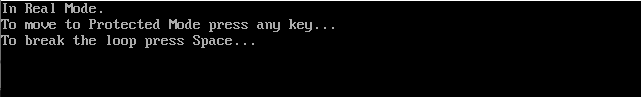
Затем программа корректно возвращается в реальный режим с соответствующими сообщениями.



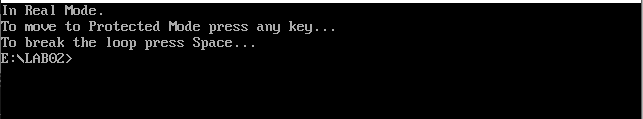
Press Key



Press Enter



Press Space



1. .386p
3. descr struc
4. lim         dw 0
5. base\_l  dw 0
6. base\_m  db 0
7. attr\_1  db 0
8. attr\_2  db 0
9. base\_h  db 0
10. descr ENDS
12. int\_descr struc
13. offs\_l  dw 0
14. sel         dw 0
15. counter db 0
16. attr    db 0
17. offs\_h  dw 0
18. int\_descr ENDS
20. ; Protected mode
21. PM\_seg  SEGMENT PARA PUBLIC 'DATA' USE32
22. ASSUME  CS:PM\_seg
24. ; Таблица дескрипторов сегметов GDT
25. GDT     label   byte
27. ; нулевой дескриптор
28. gdt\_null    descr <>
30. ; 32-битный 4-гигабайтный сегмент с базой = 0
31. gdt\_flatDS  descr <0FFFFh,0,0,92h,11001111b,0>    ; 92h = 10010010b
33. ; 16-битный 64-килобайтный сегмент кода с базой RM\_seg
34. gdt\_16bitCS descr <RM\_seg\_size-1,0,0,98h,0,0> ; 98h = 10011010b
36. ; 32-битный 4-гигабайтный сегмент кода с базой PM\_seg
37. gdt\_32bitCS descr <PM\_seg\_size-1,0,0,98h,01000000b,0>
39. ; 32-битный 4-гигабайтный сегмент данных с базой PM\_seg
40. gdt\_32bitDS descr <PM\_seg\_size-1,0,0,92h,01000000b,0>
42. ; 32-битный 4-гигабайтный сегмент данных с базой stack\_seg
43. gdt\_32bitSS descr <stack\_l-1,0,0, 92h, 01000000b,0>
45. gdt\_size = $-GDT ; размер нашей таблицы GDT+1байт (на саму метку)
47. gdtr    df 0
49. ; имена для селекторов
50. SEL\_flatDS     equ   8
51. SEL\_16bitCS    equ   16
52. SEL\_32bitCS    equ   24
53. SEL\_32bitDS    equ   32
54. SEL\_32bitSS    equ   40
56. ; Таблица дескрипторов прерываний IDT
57. IDT label   byte
59. ; первые 32 элемента таблицы
60. **trap\_f int\_descr 12 dup (<0, SEL\_32bitCS, 0, 8Eh, 0>)**; 12 первые элемента
61. **trap\_13 int\_descr <0, SEL\_32bitCS, 0, 8Eh, 0>**; 13oe исключениe
62. **trap\_s int\_descr 19 dup (<0, SEL\_32bitCS, 0, 8Eh, 0>)**; 19 остальные элемента
63. ; дескриптор прерывания от таймера
64. int08 int\_descr <0, SEL\_32bitCS,0, 8Eh, 0>
66. ; дескриптор прерывания от клавиатуры
67. int09 int\_descr <0, SEL\_32bitCS,0, 8Eh, 0>

70. idt\_size = $-IDT ; размер нашей таблицы IDT+1байт (на саму метку)
72. idtr    df 0 ; переменная размера 6 байт как Регистр таблицы дескрипторов прерываний IDTR
74. idtr\_real dw    3FFh,0,0 ; содержимое регистра IDTR в реальном режиме
76. master      db 0            ; маска прерываний ведущего контроллера
77. slave       db 0            ; ведомого
79. EScape      db 0            ; флаг - в реальный режим, если ==1
80. time\_08     dd 0            ; счетчик прошедших тиков таймера
82. msg1 db 'IN ReAL Mode.$'
83. msg2 db 13, 10, 'To Move to Protected Mode press any key...'
84. db 13, 10, 'To break the loop press Space...$'
85. msg3 db 13, 10, 'IN Protected Mode'
86. db 13, 10, 'To break the loop press Enter...$'
88. ; Таблица символов ASCII для перевода из скан кода в код ASCII.
89. ; Номер скан кода = номеру соответствующего элемента в таблице:
90. ASCII\_table db 0, 0, 49, 50, 51, 52, 53, 54, 55, 56, 57, 48, 45, 61, 0, 0
91. db 81, 87, 69, 82, 84, 89, 85, 73, 79, 80, 91, 93, 0, 0, 65, 83
92. db 68, 70, 71, 72, 74, 75, 76, 59, 39, 96, 0, 92, 90, 88, 67
93. db 86, 66, 78, 77, 44, 46, 47
94. OUT\_position    dd 1E0h ; Позиция печати вводимого текста



99. print\_str macro str
100. MOV AH,9
101. MOV DX, str
102. **INT**  21h
103. endm
105. create\_number macro
106. local number1
107. CMP  DL, 10
108. JL number1
109. ADD DL, 'A' - '0' - 10
110. number1:
111. ADD DL, '0'
112. endm
114. print\_EAX macro
115. local prcyc1
116. PUSH ECX
117. PUSH DX
119. MOV ECX,8
120. ADD EBP,0B8010h

123. prcyc1:
124. MOV DL, AL
125. AND DL, 0Fh
126. create\_number 0
127. MOV ES:[EBP],DL
128. ROR EAX,4
130. SUB EBP,2
131. LOOP prcyc1
133. SUB EBP,0B8010h
134. POP DX
135. POP ECX
136. endm

139. ; точка входа в 32-битный защищенный режим
140. PM\_entry:
141. MOV AX,SEL\_32bitDS
142. MOV DS,AX
143. MOV AX,SEL\_flatDS
144. MOV ES,AX
145. MOV AX,SEL\_32bitSS
146. MOV EBX,stack\_l
147. MOV SS,AX
148. MOV ESp,EBX
150. ; разрешить прерывания, запрещенные ранее ещё в реальном режиме
151. STI
153. CALL    compute\_memory
155. work:
156. TEST    EScape, 1
157. JZ  work
159. goback:
160. CLI
162. db  0EAh
163. dd  offset RM\_return
164. dw  SEL\_16bitCS
166. new\_INT08:
167. PUSH EAX
168. PUSH EBP
169. PUSH ECX
170. PUSH DX
171. MOV  EAX,time\_08
173. PUSH EBP
174. MOV EBP, 0
175. print\_EAX 0
176. POP EBP
178. INC EAX
179. MOV time\_08,EAX
181. POP DX
182. POP ECX
183. POP EBP
185. MOV AL,20h
186. OUT 20h,AL
187. POP EAX
189. IRETD
191. new\_INT09:
192. PUSH EAX
193. PUSH EBX
194. PUSH EBP
195. PUSH EDX
197. IN  AL,60h       ; Получаем скан-код нажатой клавиши из порта клавиатуры
199. CMP AL,1Ch       ; Сравниваем с кодом Enter
200. JNE not\_leave
201. MOV EScape,1
202. JMP leav
203. not\_leave:
204. CMP  AL,80h
205. JA leav
206. XOR AH,ah
207. MOV BP,AX
208. MOV DL, ASCII\_table[EBP]
209. MOV EBP,0B8000h
210. MOV EBX,OUT\_position
211. MOV ES:[EBP+EBX],DL
213. ADD EBX,2
214. MOV OUT\_position,EBX
215. leav:
217. IN  AL,61h
218. OR  AL,80h
219. OUT 61h,AL
221. ; Посылаем сигнал EOI:
222. MOV AL,20h
223. OUT 20h,AL
225. POP EDX
226. POP EBP
227. POP EBX
228. POP EAX
230. ; Выходим из прерывания:
231. IRET
233. compute\_memory  proc
235. push    DS
236. MOV AX, SEL\_flatDS
237. MOV DS, AX
238. MOV EBX, 100001h
239. MOV DL,     10101010b
241. MOV ECX, 0FFEFFFFEh
242. check:
243. MOV DH, DS:[EBX]
244. MOV DS:[EBX], DL
245. CMP DS:[EBX], DL
246. jnz end\_of\_memory
247. MOV DS:[EBX], DH
248. INC EBX
250. LOOP    check
251. end\_of\_memory:
252. POP DS
253. xor EDX, EDX
254. MOV EAX, EBX
255. MOV EBX, 100000h
256. div EBX
258. PUSH EBP
259. MOV EBP,20
260. print\_EAX 0
261. POP EBP
262. RET
263. compute\_memory  ENDP
265. **except\_1 proc**
266. **iret**
267. **except\_1 endp**
269. **except\_13 proc**
270. **pop eax**
271. **iret**
272. **except\_13 endp**

275. PM\_seg\_size = $-GDT
276. PM\_seg  ENDS
278. stack\_seg   SEGMENT  PARA STACK 'STACK' use32
279. stack\_start db  100h dup(?)
280. stack\_l = $-stack\_start
281. stack\_seg   ENDS
283. RM\_seg  SEGMENT PARA PUBLIC 'CODE' USE16
284. ASSUME CS:RM\_seg, DS:PM\_seg, SS:stack\_seg
286. start:
288. MOV   AX,PM\_seg
289. MOV   DS,AX
291. MOV AH, 09h
292. MOV EDX, offset msg1
293. **INT**  21h
295. MOV AH, 09h
296. MOV EDX, offset msg2
297. **INT**  21h
299. ;ожидаем ввода клавиатуры
300. PUSH EAX
301. MOV AH,10h
302. **INT**  16h
304. CMP  AL, 20h
305. JE exit
306. POP EAX
308. ; очистить экран
309. MOV AX,3
310. **INT**     10h
312. PUSH PM\_seg
313. POP DS
315. xor EAX,EAX
316. MOV AX,RM\_seg
317. shl EAX,4
318. MOV word ptr gdt\_16bitCS.base\_l,AX
319. shr EAX,16
320. MOV byte ptr gdt\_16bitCS.base\_m,AL
321. MOV AX,PM\_seg
322. shl EAX,4
323. PUSH EAX        ; для вычисления адреса idt
324. PUSH EAX        ; для вычисления адреса gdt
325. MOV word ptr GDT\_32bitCS.base\_l,AX
326. MOV word ptr GDT\_32bitSS.base\_l,AX
327. MOV word ptr GDT\_32bitDS.base\_l,AX
328. shr EAX,16
329. MOV byte ptr GDT\_32bitCS.base\_m,AL
330. MOV byte ptr GDT\_32bitSS.base\_m,AL
331. MOV byte ptr GDT\_32bitDS.base\_m,AL
333. ; вычислим линейный адрес GDT
334. POP EAX
335. ADD EAX,offset GDT
336. MOV dword ptr gdtr+2,EAX
337. MOV word ptr gdtr, gdt\_size-1
338. LGDT fword ptr gdtr
340. ; вычислим линейный адрес IDT
341. POP EAX
342. ADD EAX,offset IDT
343. MOV dword ptr idtr+2,EAX
344. MOV word ptr idtr, idt\_size-1
346. ; Заполним смещение в дескрипторах прерываний
347. MOV EAX, offset new\_INT08
348. MOV int08.offs\_l, AX
349. SHR EAX, 16
350. MOV int08.offs\_h, AX
351. MOV EAX, offset new\_INT09
352. MOV int09.offs\_l, AX
353. SHR EAX, 16
354. MOV int09.offs\_h, AX
356. **MOV EAX, offset except\_1**
357. **MOV trap\_f.offs\_l, AX**
358. **SHR EAX, 16**
359. **MOV trap\_f.offs\_h, AX**
361. **MOV EAX, offset except\_1**
362. **MOV trap\_s.offs\_l, AX**
363. **SHR EAX, 16**
364. **MOV trap\_s.offs\_h, AX**
366. **MOV EAX, offset except\_13**
367. **MOV trap\_13.offs\_l, AX**
368. **SHR EAX, 16**
369. **MOV trap\_13.offs\_h, AX**
371. ; сохраним маски прерываний контроллеров
372. IN  AL, 21h
373. MOV master, AL
374. IN  AL, 0A1h
375. MOV slave, AL
376. MOV AL, 11h
377. OUT 20h, AL
378. MOV AL, 20h
379. OUT 21h, AL
380. MOV AL, 4
382. OUT 21h, AL
383. MOV AL, 1
384. OUT 21h, AL
386. MOV AH, 09h
387. MOV EDX, offset msg3
388. **INT**  21h
390. ; Запретим все прерывания в ведущем контроллере, кроме IRQ0 и IRQ1
391. MOV AL, 0FCh
392. OUT 21h, AL
394. MOV AL, 0FFh
395. OUT 0A1h, AL
397. ; загрузим IDT
398. LIDT    fword ptr idtr

401. ; если мы собираемся работать с 32-битной памятью, стоит открыть A20
402. ; А20 - линия ("шина"), через которую осуществляется доступ ко всей
403. IN  AL,92h
404. OR  AL,2
405. OUT 92h,AL
407. CLI
409. IN  AL,70h
410. OR  AL,80h
411. OUT 70h,AL
413. MOV EAX,cr0
414. OR  AL,1
415. MOV cr0,EAX
417. db  66h
418. db  0EAh
419. dd  offset PM\_entry
420. dw  SEL\_32bitCS
422. RM\_return:
423. MOV EAX,cr0
424. AND AL,0FEh
425. MOV cr0,EAX
427. db  0EAh
428. dw  $+4
429. dw  RM\_seg
431. MOV AX,PM\_seg
432. MOV DS,AX
433. MOV ES,AX
434. MOV AX,stack\_seg
435. MOV BX,stack\_l
436. MOV SS,AX
437. MOV sp,BX
439. ;перепрограммируем ведущий контроллер обратно на вектор 8
440. MOV AL, 11h
441. OUT 20h, AL
442. MOV AL, 8
443. OUT 21h, AL
444. MOV AL, 4
445. OUT 21h, AL
446. MOV AL, 1
447. OUT 21h, AL
449. MOV AL, master
450. OUT 21h, AL
451. MOV AL, slave
452. OUT 0A1h, AL
454. LIDT fword ptr idtr\_real
456. IN  AL,70h
457. AND AL,07FH
458. OUT 70h,AL
460. STI
462. MOV AX,3
463. **INT** 10h
465. JMP start
467. MOV AH, 09h
468. MOV EDX, offset msg1
469. **INT**  21h
470. exit:
471. MOV AH,4Ch
472. **INT** 21h
474. RM\_seg\_size = $-start
475. RM\_seg  ENDS
476. END start