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 №2 по курсу "Операционные системы"

Тема Защищённый режим

Студент Прянишников А. Н.

Группа ИУ7-55Б

Оценка (баллы) _____

Преподаватели Рязанова Н.А.

Москва — 2021 г.

Код программы

```
.586p

; Структура дескриптора сегмента
descr struct
    limit   DW 0
    base_l  DW 0
    base_m  DB 0
    attr_1  DB 0
    attr_2  DB 0
    base_h  DB 0
descr ends

; Структура дескриптора прерывания
idescr struc
    offs_l  dw 0
    sel     dw 0
    cntr    db 0
    attr    db 0
    offs_h  dw 0
idescr ends

; Сегмент стека.
stack32 segment  para stack 'STACK'
    stack_start db  100h dup(?)
    stack_size = $-stack_start
stack32 ends

; Сегмент данных
data32 segment para 'data'
    gdt_null  descr <>
    ; дескриптор, описывающий сегмент кода для реального режима
    gdt_code16 descr <code16_size-1,0,0,98h>
    gdt_data4gb descr <0FFFFh,0,0,92h,0CFh>
    gdt_code32 descr <code32_size-1,0,0,98h,40h>
    gdt_data32 descr <data_size-1,0,0,92h,40h>
    gdt_stack32 descr <stack_size-1,0,0,92h,40h>
    gdt_video16 descr <3999,8000h,0Bh,92h> ; 0 + B 0000h + 8000h = 0B8000h


    ; Размер таблицы GDT.
    gdt_size=$-gdt_null
    ; DF - выделить поле для псевдодескриптора (6-байт).
    pdescr    df 0

    ; Селекторы - номер (индекс начала) дескриптора в GDT.
```

```
46      code16s=8
47      data4gbs=16
48      code32s=24
49      data32s=32
50      stack32s=40
51      video16s=48
52
53      ; idt - метка начала IDT
54      idt label byte
55
56      idescr_0_12 idescr 13 dup (<0,code32s,0,8Fh,0>)
57
58      idescr_13 idescr <0,code32s,0,8Fh,0>
59
60      ;[14, 31]
61      idescr_14_31 idescr 18 dup (<0,code32s,0,8Fh,0>)
62
63      ; 10001110 - Eh - шлюз прирываний - служит для обработки прерывания.
64      int08 idescr <0,code32s,0,10001110b,0>
65      int09 idescr <0,code32s,0,10001110b,0>
66
67      ; Размер таблицы дескрипторов прерываний.
68      idt_size=$-idt
69
70      ; interruption psevdo descriptor
71      ; DF - выделить поле для псевдодескриптора (6-байт).
72      ipdescr df 0
73
74      ipdescr16 dw 3FFh, 0, 0
75
76      mask_master db 0
77      mask_slave  db 0
78
79      asciimap   db 0, 0, 49, 50, 51, 52, 53, 54, 55, 56, 57, 48, 45, 61, 8, 0
80      db 113, 119, 101, 114, 116, 121, 117, 105, 111, 112, '[', ']', 0, 0, 97,
            115
81      db 100, 102, 103, 104, 106, 107, 108, 59, 39, 96, 7, 92, 122, 120, 99
82      db 118, 98, 110, 109, 44, 46, 47
83
84      flag_enter_pr db 0
85      flag_shift_pr db 0
86
87      start_pos          dd 2 * 80 * 5
88      syml_pos       dd 2 * 80 * 5
89
90      mem_pos=0
```

```
91
92      mem_value_pos=14 + 16
93      mb_pos=30 + 2
94      cursor_pos=80
95      param=1Eh
96
97
98      cursor_symb=219
99      color db 00Fh ; Цвет курсора.
100
101     rm_msg      db 27, '[30;42mNow␣in␣Real␣Mode.␣', 27, '[0m$', '$'
102     pm_msg_wait db 27, '[30;42mPress␣any␣button␣to␣enter␣protected␣mode!',
            27, '[0m$'
103     pm_msg_out  db 27, '[30;42mNow␣in␣Real␣Mode␣again!␣', 27, '[0m$'
104     pm_mem_count db 'Memory:␣'
105
106     data_size = $-gdt_null
107 data32 ends
108
109
110 code32 segment para public 'code' use32
111     assume cs:code32, ds:data32, ss:stack32
112
113 pm_start:
114     mov ax, data32s
115     mov ds, ax
116     mov ax, video16s
117     mov es, ax
118     mov ax, stack32s
119     mov ss, ax
120     mov eax, stack_size
121     mov esp, eax
122
123     sti ; Резрешаем (аппаратные) прерывания
124
125     ; Вывод сообщения "Memory"
126     mov di, mem_pos
127     mov ah, param
128     xor esi, esi
129     xor ecx, ecx
130     mov cx, 8 ; Длина строки
131     print_memory_msg:
132         mov al, pm_mem_count[esi]
133         stosw ; al (символ) с параметром (ah) перемещается в область памяти
                es:di
134         inc esi
```

```
135     loop print_memory_msg
136
137     ; Считаем и выводим кол-во физической памяти, выделенной dosbox'y.
138     call count_memory_proc
139
140     ; Цикл, пока не будет введен Enter
141     proccess:
142         test flag_enter_pr, 1 ; если flag = 1, то выход
143     jz  proccess
144
145     ; Выход из защищенного режима
146     cli ; Запрет аппаратных маскируемые прерывания прерываний.
147
148     db  0EAh ; jmp
149     dd  offset return_rm ; offset
150     dw  code16s ; selector
151
152
153     ; Зашлушка для исключений.
154     except_1 proc
155         iret
156     except_1 endp
157
158     ; Заглушка для 13 исключения.
159     ; Нужно снять со стека код ошибки.
160     except_13 proc uses eax
161         pop eax
162         iret
163     except_13 endp
164
165
166     new_int08 proc uses eax
167         mov edi, cursor_pos ; поместим в edi позицию для вывода
168
169         mov ah, color ; В ah помещаем цвет текста.
170         ror ah, 1
171         mov color, ah
172         mov al, cursor_symb ; Символ, который мы хотим вывести (в моем случа
                e просто квадрат).
173         stosw ; al (символ) с параметром (ah) перемещается в область памяти
                es:di
174
175         mov al, 20h
176         out 20h, al
177
178         iretd ; double - 32 битный iret
```

```
179     new_int08 endp
180
181     ; uses - сохраняет контекст (push + pop)
182     new_int09 proc uses eax ebx edx
183         ;Порт 60h при чтении содержит скан-код последней нажатой клавиши.
184         in  al, 60h ; считываем порт клавы
185         cmp al, 1Ch ; сравниваем с Enter'ом
186
187         jne print_value
188         or flag_enter_pr, 1 ; если Enter, устанавливаем флаг
189         jmp exit
190
191
192         print_value:
193             cmp al, 80h
194             ja exit
195
196             xor ah, ah
197
198             xor ebx, ebx
199             mov bx, ax
200
201             mov dh, param
202             mov dl, asciimap[ebx]
203
204             mov ebx, syml_pos
205             cmp dl, 8
206             je del_symb
207
208             cmp dl, 7
209             je shift_down
210
211             cmp flag_shift_pr, 1
212             jne print
213
214             ; Проверяем, что выводим букву
215             cmp dl, 96
216             jle print
217
218             cmp dl, 123
219             jge print
220
221             sub dl, 32
222             jmp print
223
224         shift_down:
```

```
225             mov dl, '*'
226             cmp flag_shift_pr, 1
227             je shift_up
228 
229             mov flag_shift_pr, 1
230             jmp print
231 
232         shift_up:
233             mov flag_shift_pr, 0
234             jmp print
235 
236         print:
237             mov es:[ebx], dx
238 
239             add ebx, 2
240             mov syml_pos, ebx
241             jmp exit
242 
243         del_symb:
244             sub ebx, 2
245             mov dh, 0
246             mov dl, '␣'
247             mov es:[ebx], dx
248             mov syml_pos, ebx
249             jmp exit
250 
251 
252         exit:
253 
254             mov al, 20h
255             out 20h, al
256 
257             iretd
258     new_int09 endp
259 
260     count_memory_proc proc uses ds eax ebx
261         mov ax, data4gbs ; Селектор, который указывает на дескриптор, описыв
                ающий сегмент 4 Гб.
262         mov ds, ax ; На данном этапе в сегментный регистр помещается селекто
                р data4gbs
263 
264         mov ebx,  100001h ;
265         mov dl,   0AEh
266 
267         mov ecx, 0FFEFFFFEh
268 
```

```
269         count_memory:
270             mov dh, ds:[ebx]
271 
272             mov ds:[ebx], dl
273 
274             cmp ds:[ebx], dl
275 
276             jne print_memory_counter
277 
278             mov ds:[ebx], dh    ; Обратно запиываем считанное значени.
279             inc ebx             ; Увеличиваем счетчик.
280         loop count_memory
281 
282     print_memory_counter:
283         mov eax, ebx
284         xor edx, edx
285 
286         mov ebx, 100000h
287         div ebx
288 
289         mov ebx, mem_value_pos
290         call print_count_memory
291 
292         ; Печать надписи Mb (мегабайты)
293         mov ah, param
294         mov ebx, mb_pos
295         mov al, 'M'
296         mov es:[ebx], ax
297 
298         mov ebx, mb_pos + 2
299         mov al, 'b'
300         mov es:[ebx], ax
301         ret
302 
303     count_memory_proc endp
304 
305     print_count_memory proc uses ecx ebx edx
306 
307         mov ecx, 8
308         mov dh, param
309 
310         print_symbol:
311             mov dl, al
312 
313             and dl, 0Fh
314 
```

```
315             cmp dl, 10
316
317             jl to_ten ; До десяти.
318
319             sub dl, 10
320
321             add dl, 'A'
322             jmp after_ten
323
324
325         to_ten:
326             add dl, '0'  ; Превращаем в строковое представление число.
327         after_ten:
328             mov es:[ebx], dx
329
330             ror eax, 4
331             sub ebx, 2
332         loop print_symbol
333
334         ret
335     print_count_memory endp
336
337     code32_size = $-pm_start
338 code32 ends
339
340
341 code16 segment para public 'CODE' use16
342 assume cs:code16, ds:data32, ss: stack32
343
344 NewLine:
345     xor dx, dx
346     mov ah, 2
347     mov dl, 13
348     int 21h
349     mov dl, 10
350     int 21h
351     ret
352
353 ClearScreen:
354     ; Инструкция очистки экрана
355     mov ax, 3
356     int 10h
357     ret
358
359
360 ; Начало программы.
```

```
361 start:
362     mov ax, data32
363     ; Инициализируем DS сегментом данных.
364     mov ds, ax
365 
366     ; Выводим сообщение, о том, что мы в реальном режиме.
367     mov ah, 09h
368     lea dx, rm_msg
369     int 21h
370     call NewLine
371 
372     ; Вывод сообщения, что мы ожидаем нажатие клавиши.
373     mov ah, 09h
374     lea dx, pm_msg_wait
375     int 21h
376     call NewLine
377 
378     ; Ожидание нажатия кнопки
379     mov ah, 10h
380     int 16h
381 
382     call ClearScreen
383 
384     xor eax, eax
385 
386 
387     mov ax, code16
388     shl eax, 4 ; Получаем линейный адрес
389     mov word ptr gdt_code16.base_l, ax
390     shr eax, 16
391     mov byte ptr gdt_code16.base_m, al
392     mov byte ptr gdt_code16.base_h, ah
393 
394     mov ax, code32
395     shl eax, 4
396     mov word ptr gdt_code32.base_l, ax
397     shr eax, 16
398     mov byte ptr gdt_code32.base_m, al
399     mov byte ptr gdt_code32.base_h, ah
400 
401     mov ax, data32
402     shl eax, 4
403     mov word ptr gdt_data32.base_l, ax
404     shr eax, 16
405     mov byte ptr gdt_data32.base_m, al
406     mov byte ptr gdt_data32.base_h, ah
```

```
407
408     mov ax, stack32
409     shl eax, 4
410     mov word ptr gdt_stack32.base_l, ax
411     shr eax, 16
412     mov byte ptr gdt_stack32.base_m, al
413     mov byte ptr gdt_stack32.base_h, ah
414
415     mov ax, data32
416     shl eax, 4
417
418     add eax, offset gdt_null
419
420     mov dword ptr pdescr+2, eax
421     mov word ptr  pdescr, gdt_size-1
422     lgdt fword ptr pdescr
423
424
425     mov ax, code32
426     mov es, ax
427
428     lea eax, es:except_1
429     mov idescr_0_12.offs_l, ax
430     shr eax, 16
431     mov idescr_0_12.offs_h, ax
432
433     lea eax, es:except_13
434     mov idescr_13.offs_l, ax
435     shr eax, 16
436     mov idescr_13.offs_h, ax
437
438     lea eax, es:except_1
439     mov idescr_14_31.offs_l, ax
440     shr eax, 16
441     mov idescr_14_31.offs_h, ax
442
443
444     lea eax, es:new_int08
445     mov int08.offs_l, ax
446     shr eax, 16
447     mov int08.offs_h, ax
448
449     lea eax, es:new_int09
450     mov int09.offs_l, ax
451     shr eax, 16
452     mov int09.offs_h, ax
```

```
453
454     ; Получаем линейный адрес IDT
455     mov ax, data32
456     shl eax, 4
457     add eax, offset idt
458
459     mov  dword ptr ipdescr + 2, eax
460     ; И размер IDT
461     mov  word ptr  ipdescr, idt_size-1
462
463      ; Сохранение масок
464     in  al, 21h
465     mov mask_master, al
466     in  al, 0A1h
467     mov mask_slave, al
468
469     ; Перепрограммирование контроллера прерываний
470     mov al, 11h
471     out 20h, al
472     mov al, 32
473     out 21h, al
474     mov al, 4
475     out 21h, al
476     mov al, 1
477     out 21h, al
478
479     mov al, 0FCh
480     out 21h, al
481
482     mov al, 0FFh
483     out 0A1h, al
484
485     ; открытие линии A20
486     in  al, 92h
487     or  al, 2
488     out 92h, al
489
490     cli
491
492     ;  lidt - load IDT - загрузить в регистр IDTR
493     ; Наш псевдодискриптор ipdescr, который содержит
494     ; Лин адрес таблицы IDT и её размер.
495     lidt fword ptr ipdescr
496
497     ; Запрет немаскируемых прерываний. NMI
498     mov al, 80h
```

```
499     out 70h, al
500
501     ; Переход в защищенный режим
502     mov eax, cr0 ; Получаем содержимое регистра CR0
503     or eax, 1    ; Установим бит защищённого режима
504     mov cr0, eax ; Обновляем содержимое регистра CR0
505
506     db  66h  ; Префикс изменения разрядности операнда (меняет на противополо
            жный).
507     db  0EAh ; Код команды far jmp.
508     dd  offset pm_start ; Смещение
509     dw  code32s         ; Сегмент
510
511
512 return_rm:
513     ; возвращаем флаг pe
514     mov eax, cr0
515     and al, 0FEh
516     mov cr0, eax
517
518     db  0EAh
519     dw  offset go
520     dw  code16
521
522 go:
523     ; обновляем все сегментные регистры
524     mov ax, data32
525     mov ds, ax
526     mov ax, code32
527     mov es, ax
528     mov ax, stack32
529     mov ss, ax
530     mov ax, stack_size
531     mov sp, ax
532
533     ; возвращаем базовый вектор контроллера прерываний
534     mov al, 11h
535     out 20h, al
536     mov al, 8
537     out 21h, al
538     mov al, 4
539     out 21h, al
540     mov al, 1
541     out 21h, al
542
543     ; восстанавливаем маски контроллеров прерываний
```

```
544     mov al, mask_master
545     out 21h, al
546     mov al, mask_slave
547     out 0A1h, al
548 
549     ; восстанавливаем вектор прерываний (на 1ый кб)
550     lidt    fword ptr ipdescr16
551 
552     ; закрытие линии A20
553     in  al, 70h
554     and al, 7Fh
555     out 70h, al
556 
557     sti ; Резрешаем (аппаратные) прерывания
558 
559     ; Очищаем экран
560     call ClearScreen
561 
562     mov ah, 09h
563     lea dx, pm_msg_out
564     int 21h
565     call NewLine
566 
567     ; Завершаем программу.
568     mov ax, 4C00h
569     int 21h
570 
571     code16_size = $-start
572 code16 ends
573 
574 end start
```

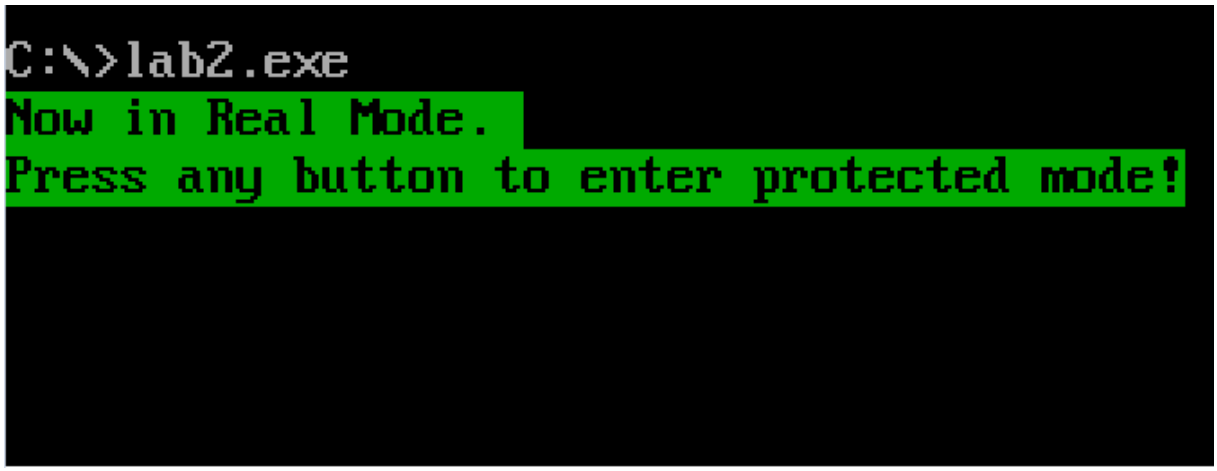
Демонстрация работы программы

В обработчике прерывания отдельно обрабатывается BackSpace и Shift.

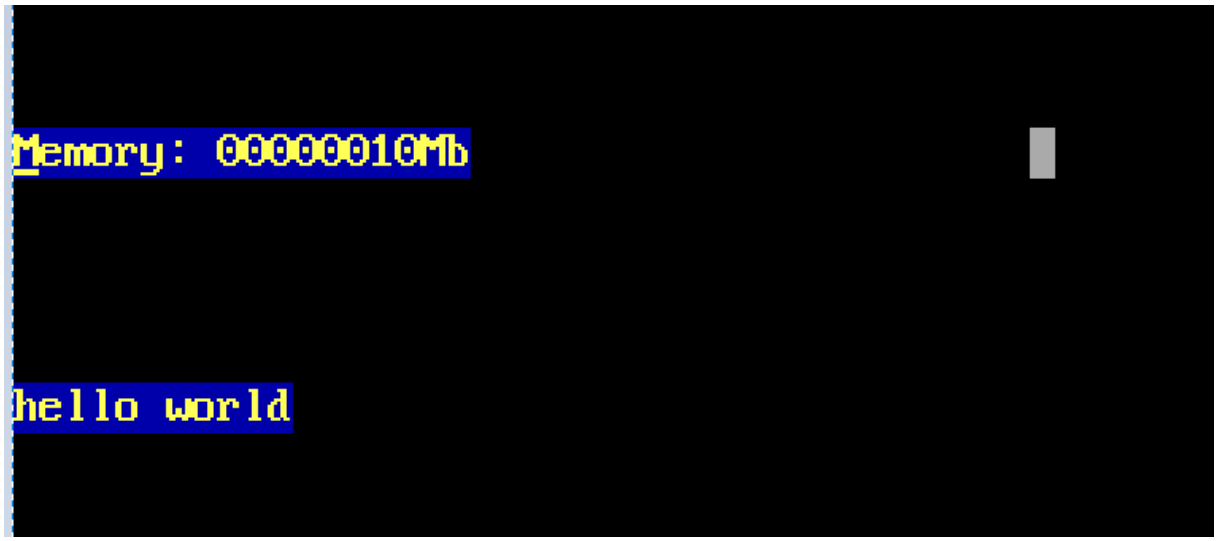
В случае BackSpace последний перед курсором символ стирается.

В случае Shift происходит смена регистра и печатается '*'.

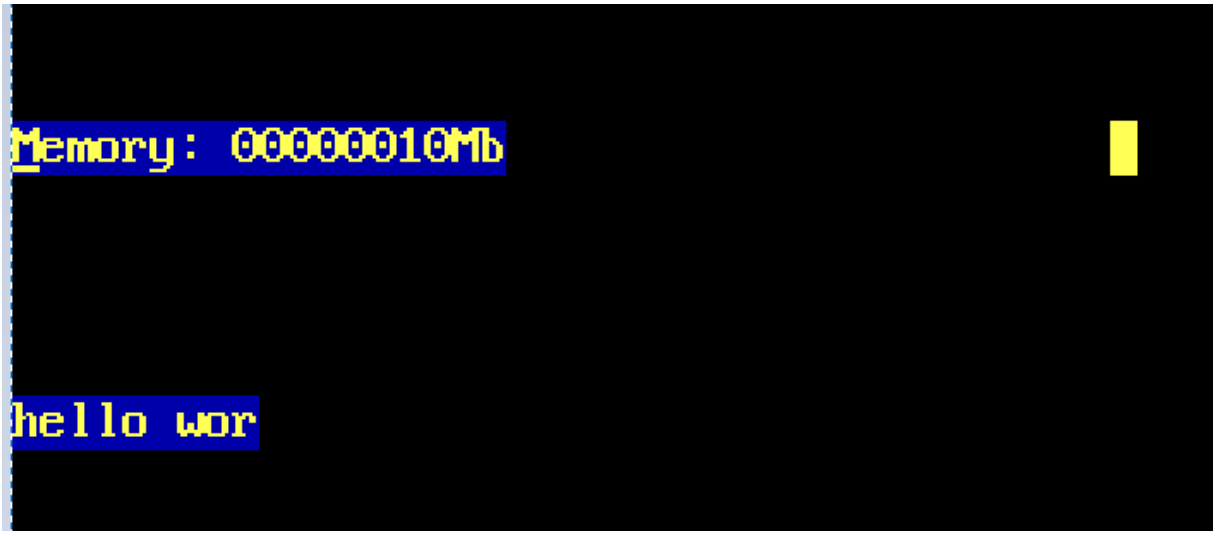
В начале программа стартует в реальном режиме и ждёт нажатия любой клавиши для перехода:

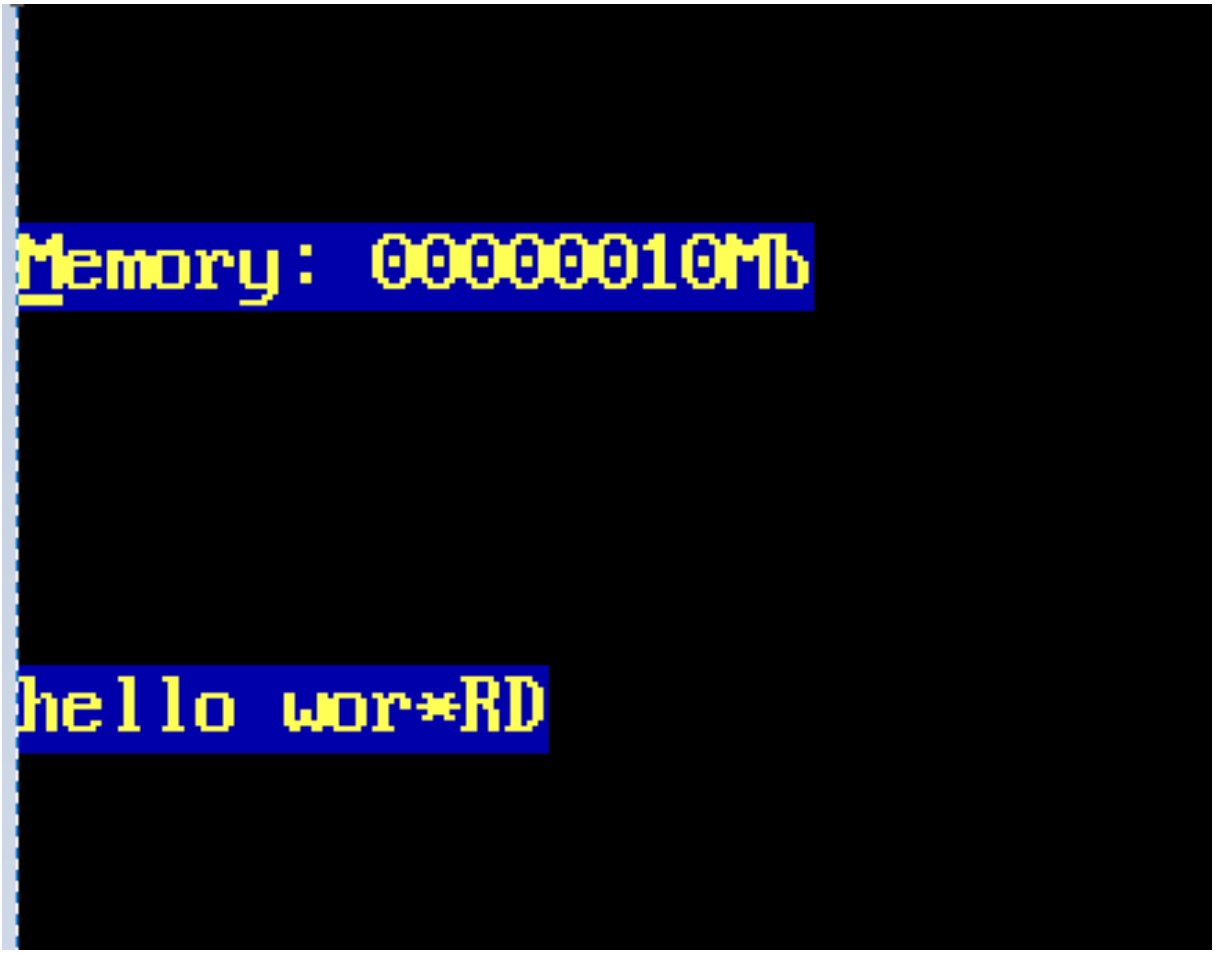


После перехода в защищённый режим программа подсчитывает количество доступной памяти и выводит значение на экран. При этом справа от значения есть квадрат, который постоянно меняет свой цвет по тику. Снизу пользователь может вводить символы на экран:



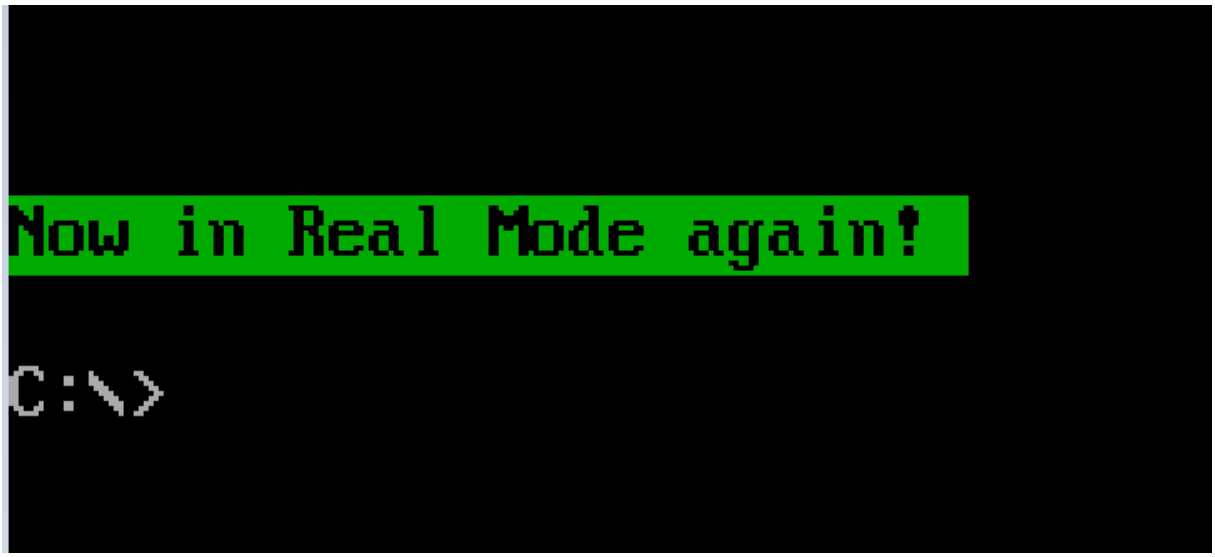
Теперь пользователь ввёл BackSpace и стёр rd. Затем пользователь нажал Shift и дописал буквы RD:


Memory: 00000010Mb
hello wor


Memory: 00000010Mb
hello wor*RD

Полный набор доступных символов представлен на скрине:

В конце программа по нажатию Enter возвращается в реальный режим:



Максимальная память, доступная в DOSBOX

В конфигурационном файле DosBox можно поменять поле memsize, которое отвечает за количество доступной памяти. Несмотря на то, что число может быть больше 64, максимальное количество памяти не превысило 63: