Visvesvaraya National Institute of Technology (VNIT), Nagpur

Advanced microprocessors and interfacing lab (ECP426)

Project

Submitted by :
Shruti Murarka (BT18ECE099)
Kushagra Srivastava (BT18ECE109)
Mayank Bumb (BT18ECE111)
Semester VII

Submitted to :
Dr. Neha K. Nawandar and Dr. Abhisek Pahuja
(Lab Instructors)
Department of Electronics and Communication Engineering,
VNIT Nagpur

Contents

1 Problem Statement and Theory 2
2 Code 3
3 Results and explanation of Code 18
4 Conclusion 21

Problem Statement and Theory

Hangman is a two or more player guessing game. One person (computer) comes up with a word, while the other tries to guess it by putting letters together. The number of letters in the word to guess is represented by a row of dashes. If a player guesses a letter that appears in the word, the algorithm writes it in all of its right placements. If the recommended letter does not appear in the word, the other player makes a tally mark with one of the hangman diagram elements (head, body, 2 hands, 2 legs).

Figure 1: Hangman

The guessing player wins if accurately guesses the entire word before completion of hangman diagram [7 wrong guesses].

The player loses if the hangman diagram is completed before all of the letters are guessed.

Game Description: There is a list of words which we have saved out of which one word is chosen randomly. Your task is to correctly recognise the word in less than 7 chances. On the top right one can see the hanging machine where the human isn't seen initially. At bottom one can see a few blanks which you need to fill in with some letter and gradually move on to successfully recognise the correct word. Then there is a failure wherein you can see the letters you have already chosen and do not fit in any of the blanks. (Be smart and don't repeat letters). As you move on with the game and as the number of incorrect guesses increases the body parts of hangman start appearing and once this count gets to 7 the hangman dies and your task is to save him before that by guessing the correct word. In the end if one wins then the word will appear in green text suggesting his/her success along with hurray else it will appear in red with better luck next time. Happy playing!!!

Code

Listing 1: Hangman

```
1
2  data segment
3
4      square_head db 0,0,15,15,15,15,15,15,15,15,15,15,15,0,0
5                  db 0,0,15,0,0,0,0,0,0,0,0,0,15,0,0
6                  db 0,0,15,0,0,0,0,0,0,0,0,0,15,0,0
7                  db 0,0,15,0,0,0,0,0,0,0,0,0,15,0,0
8                  db 0,0,15,0,0,0,0,0,0,0,0,0,15,0,0
9                  db 0,0,15,0,0,0,0,0,0,0,0,0,15,0,0
10                 db 0,0,15,0,0,0,0,0,0,0,0,0,15,0,0
11                 db 0,0,15,0,0,0,0,0,0,0,0,0,15,0,0
12                 db 0,0,15,0,0,0,0,0,0,0,0,0,15,0,0
13                 db 0,0,15,0,0,0,0,0,0,0,0,0,15,0,0
14                 db 0,0,15,0,0,0,0,0,0,0,0,0,15,0,0
15                 db 0,0,15,0,0,0,0,0,0,0,0,0,15,0,0
16                 db 0,0,15,0,0,0,0,0,0,0,0,0,15,0,0
17                 db 0,0,15,0,0,0,0,0,0,0,0,0,15,0,0
18                 db 0,0,15,15,15,15,15,15,15,15,15,15,15,0,0
19     head_length equ 15
20     win_label_message db "Hurray, you win !!!$"
21     lose_label_message db "Better Luck Next Time :($"
22     file_words db "hangman_words.txt"
23     File_handler dw 0
24     buffer_word db 255 dup (0)
25     random_no db 0
26     word db 20 dup (0)
27     word_len db 0
28     fail_label_message db "fails:$"
29     fail_number db 0
30     correct_word db 20 dup (0)
31     did_win db 0
32     succeed db 0
33     fail_place db 6
34     hangmann_title db " H A N G M A N$"
35
36 ends
37
38 stack segment
39     dw 128 dup(0)
40 ends
41
42 code segment
43     proc initialize_screen
```

```
44          push ax
45          mov ax,13h ;enables screen mode
46          int 10h
47          pop ax
48          ret
49      endp
50
51      proc man_head
52          pusha ;push all the general purpose registers to the stack, prevents ...
                information loss
53          lea si, square_head ;to display the head of the hangman
54          mov ah, 0ch ;change color for a pixel
55          mov cx,213 ;column number
56          mov dx, 55 ; row
57
58          draw_head_loop:
59              cmp dx, 55 + head_length ;check if all rows have been printed
60              je draw_head_exit ;if yes then exit the loop else continue
61              mov al, [si] ;moves the next value to al
62              int 10h ;changes the color of that pixel
63              inc cx ;increment column number
64              inc si ;move to the next value
65              cmp cx, 213 + head_length ;check if all columns have been printed
66              je next_row ;if yes then move to next row
67              jmp draw_head_loop ;else continue is same row
68          next_row:
69              inc dx ;increment row value
70              mov cx, 213 ; start from the first column index again
71              jmp draw_head_loop ;follow the same procedure for the previous row
72
73          draw_head_exit:
74              popa ;pop back all values from the stack
75              ret
76      man_head endp
77
78      proc draw_base
79          pusha
80          mov ah,0ch ;change colour of single pixel
81          mov bh, 0000h
82          mov al,15 ;pixel color value
83          mov cx, 240 ;coloumn number
84          mov dx, 120 ;row number
85
86          part1:
87              int 10h
88              inc cx
89              cmp cx,270
90              jne part1 ;draw a straight line for 30 columns keeping row fixed
91
```

```
92          mov cx,255 ;new column number
93
94      part2:
95          int 10h
96          dec dx
97          cmp dx, 45
98          jne part2 ;draw a straight line for 75 rows keeping column fixed
99
100     part3:
101         int 10h
102         dec cx
103         cmp cx, 220 ;draw a straight line for 35 columns keeping row fixed
104         jne part3
105
106     part4:
107         int 10h
108         inc dx
109         cmp dx,55
110         jne part4 ;draw a straight line for 10 rows keeping columns fixed
111
112     popa
113     ret
114 draw_base endp
115
116 proc draw_main_body ;to be used to indicate progress
117     pusha
118     mov ah,0ch
119     mov al,15
120     mov dx,70 ;starting row
121     mov cx,220 ;starting column
122
123     main_body_loop:
124         int 10h
125         inc dx
126         cmp dx,100 ;draw a straight line for 30 rows keeping column fixe
127         jne main_body_loop
128     popa
129     ret
130 draw_main_body endp
131
132 proc first_hand
133     pusha
134     mov ah,0ch
135     mov al,15
136     mov dx,70 ;starting row
137     mov cx,220 ;starting column
138
139     first_hand_loop:
140         int 10h
```

```
141             inc dx
142             inc cx ;hand will be slanting so incrementing columns
143             cmp dx,85 ;draw a straight line for 15 rows
144             jne first_hand_loop
145         popa
146         ret
147     first_hand endp
148     proc second_hand
149         pusha
150         mov ah,0ch
151         mov al,15
152         mov dx,70 ;starting row
153         mov cx,220 ;starting column
154 
155         second_hand_loop:
156         int 10h
157         inc dx
158         dec cx ;hand will be slanting so decrementing columns
159         cmp dx,85 ;draw a straight line for 15 rows
160         jne second_hand_loop
161         popa
162         ret
163     second_hand endp
164     proc first_leg
165         pusha
166         mov ah,0ch
167         mov al,15
168         mov dx,100 ;starting row
169         mov cx,220 ;starting column
170 
171         first_leg_loop:
172         int 10h
173         inc dx
174         inc cx ;leg will be slanting so incrementing columns
175         cmp dx,115 ;draw a straight line for 15 rows
176         jne first_leg_loop
177         popa
178         ret
179      first_leg endp
180 
181      proc second_leg
182         pusha
183         mov ah,0ch
184         mov al,15
185         mov dx,100 ;starting row
186         mov cx,220 ;starting column
187 
188         second_leg_loop:
189             int 10h
```

```
190             inc dx
191             dec cx ;leg will be slanting so decrementing columns
192             cmp dx,115 ;draw a straight line for 15 rows
193             jne second_leg_loop
194         popa
195         ret
196       endp second_leg
197
198
199     proc win
200         pusha
201
202         lea si, word ; pointer to word
203         mov dl, 4
204         write_win:
205             mov dh, 20 ;Row
206             xor bh, bh ;Display page set to 0
207             mov ah, 02h ;SetCursorPosition
208             int 10h
209             add dl, 4
210
211             mov ah, 09h ; write char
212             mov cx, 1
213             mov al, [si]
214             inc si
215             mov bl, 2h
216             int 10h
217             cmp [si], "$"
218             jne write_win
219
220             lea si, win_label_message ; pointer to win msg
221             mov dl, 0
222             xor cl, cl
223         write_label2:
224             mov dh, 24 ;Row
225             mov bh, 0 ;Display page
226             mov ah, 02h ;SetCursorPosition
227             int 10h
228
229             mov al, [si]
230             inc si
231             ;mov al, '3'
232             mov bl, 2h ;Color is red
233             xor bh, bh ;Display page
234             mov ah, 0Eh ;Teletype
235             int 10h
236             inc dl
237             cmp [si], "$"
238             jne write_label2
```

```
239         popa
240         ret
241     win endp
242
243     proc write_all_word
244         pusha
245
246         lea si, word
247         mov dl, 4
248         write_loop:
249
250             mov dh, 20 ;Row
251             xor bh, bh ;Display page set to 0
252             mov ah, 02h ;SetCursorPosition
253             int 10h
254             add dl, 4
255
256
257             mov ah, 09h ; write char
258             mov cx, 1 ; no of times to write char
259             mov al, [si]
260             inc si
261             mov bl, 4h
262             int 10h
263             cmp [si], "$" ; compare with end of string
264             jne write_loop ; loop till word ends
265
266
267             lea si, lose_label_message ; pointer to lose msg
268             mov dl, 0
269             xor cl, cl
270         write_label1:
271             mov dh, 24 ;Row
272             mov bh, 0 ;Display page
273             mov ah, 02h ;SetCursorPosition
274             int 10h
275
276             mov al, [si]
277             inc si
278             ;mov al, '3'
279             mov bl, 2h ;Color is red
280             xor bh, bh ;Display page
281             mov ah, 0Eh ;Teletype
282             int 10h
283             inc dl
284             cmp [si], "$"
285             jne write_label1
286         popa
287         ret
```

```
288     write_all_word endp
289
290     open_read_file proc
291         pusha ; contents of registers in stack
292         ;opens file
293         mov ah, 3Dh ; open existing file
294         lea dx, file_words ; file pointer
295         xor al, al ; read file
296         int 21h
297         mov offset File_handler, ax ;handler
298
299         ;reads from file
300         mov ah, 3Fh ; read from file
301         mov bx, [File_handler] ; file handle
302         mov cx, 255 ; no. of bytes to read
303         lea dx, buffer_word ; pointer to buffer
304         int 21h
305         popa ; return contents of register
306         ret
307     open_read_file endp
308
309     get_random_number proc
310         pusha
311         mov ah, 2ch ; get system time
312
313         int 21h ; CH = hour. CL = minute. DH = second. DL = 1/100 seconds.
314         mov ax, dx ; get seconds accumulator
315         add ah, al ; add AH and AL
316         xor dx, dx ; clear DX
317         mov bx, 10
318         div bx ; divide A by 10
319         lea bx, random_no ; get pointer for random number
320         mov [bx], ah ; store random number
321         popa
322         ret
323     get_random_number endp
324
325     get_random_word proc
326         pusha
327         lea bx, random_no ; pointer to random number
328         mov cx, [bx] ; get random number in Cx
329         lea bx, buffer_word ; get pointer to buffer
330       loop_until_found: ;0A (new line)
331         inc bx
332         mov ax, [bx]
333         cmp al, 0Ah
334         jne loop_until_found ; loop until new line found
335         dec cx ; decrement count
336         cmp cx,0
```

```
337         jne loop_until_found ; loop for random number times
338
339         inc bx
340
341
342         lea di, word ;pointer to variable word
343        collect_word:
344         mov si, [bx] ; data from buffer to si
345         mov [di], si
346         inc bx ; increment buffer pointer
347         inc di ; increment destination pointer
348         cmp si, 0A0Dh ; 0A0D new line
349         jne collect_word
350
351         dec di
352         mov [di], "$" ; mov $ at the end of string
353
354         popa
355         ret
356     get_random_word endp
357
358     get_word_len proc
359         pusha
360
361         lea bx, word ; pointer to word
362         xor cx, cx ; clear count
363        count:
364         inc cx ; increase count
365         inc bx ; increase pointer
366         cmp [bx], "$" ; compare character with "$" i.e. end of string
367         jne count ; if not equal jump to count
368
369         lea bx, word_len ; pointer to word_length
370         mov [bx], cl ; store word_length to variable
371
372
373         popa
374         ret
375     get_word_len endp
376
377     proc fail_label
378
379         lea si, fail_label_message
380         mov dl, 0
381         xor cl, cl
382        write_label:
383         ;mov dl, cl ;Column start at 3
384         mov dh, 22 ;Row
385         mov bh, 0 ;Display page
```

```
386         mov ah, 02h ;SetCursorPosition
387         int 10h
388 
389         mov al, [si]
390         inc si
391         ;mov al, '3'
392         mov bl, 0Ch ;Color is red
393         xor bh, bh ;Display page
394         mov ah, 0Eh ;Teletype
395         int 10h
396         inc dl
397         cmp [si], "$"
398         jne write_label
399         ret
400     fail_label endp
401 
402     proc draw_lines
403         pusha
404         lea si, word_len
405         mov bx, [si]
406 
407         mov ah, 0ch
408         mov al, 15 ; pixel color
409         mov cx, 27 ; column
410         mov dx, 170 ; row
411 
412        lines:
413         call draw_line
414         dec bx
415         add cx, 12
416         cmp bl, 0
417         jne lines
418 
419         popa
420         ret
421     draw_lines endp
422     proc draw_line
423         push bx
424         xor bx, bx
425        making_line:
426         int 10h
427         inc cx
428         inc bx
429         cmp bx, 20
430         jne making_line
431 
432         pop bx
433         ret
434     draw_line endp
```

```
435
436     proc check_win ; compare correct_word by user with word
437         pusha
438
439         lea si, correct_word ; pointer to user word
440         lea bx, word ; actual word
441         cmp [bx], "$" ; compare with eos
442         je call_win ; if reached eos user won
443         mov al, [si]
444         mov dl, [bx]
445         cmp al, dl
446         je check_loop
447
448
449       check_loop:
450         inc si
451         inc bx
452         cmp [bx], "$"
453         je call_win
454         mov al, [si]
455         mov dl, [bx]
456         cmp al, dl
457         je check_loop
458
459         exit_check:
460         popa
461         ret
462
463       call_win:
464         lea si, did_win ; set did_win variable
465         mov [si], 1
466         call win
467         jmp exit_check
468     check_win endp
469
470     get_letter proc
471         pusha
472         mov ah, 7h ; character input
473         int 21h
474
475         lea si, correct_word
476
477         mov cl, 1 ; count
478
479         lea bx, word ; pointer to word
480         cmp [bx], al ; compare character to input letter
481         je call_write_letter
482
483        check_for_letter:
```

```
484         inc cl ; increase count
485         inc bx ; increase word pointer
486         inc si ; increase correct pointer
487         cmp [bx], "$" ; if string end char not found
488         je call_get_fail
489         cmp [bx], al
490         je call_write_letter
491
492         jne check_for_letter
493
494        call_get_fail:
495         push ax
496         call get_fail ; char not found
497         jmp exit
498        call_write_letter:
499         mov [si], al ; move char to correct variable
500         push ax
501         push cx
502         call write_letter
503         jmp check_for_letter
504
505
506        exit:
507         popa
508         ret
509     write_letter proc
510         mov bp, sp
511         pusha
512
513         lea si, succeed
514         mov [si], 1
515
516         mov cx, [bp+2]
517         mov ax, 4
518         mul cx
519
520
521         mov dl, al ;Column start at al location
522         mov dh, 20 ;Row
523         xor bh, bh ;Display page set to 0
524         mov ah, 02h ;SetCursorPosition in int 10h
525         int 10h
526
527         mov ax, [bp+4]
528         ;mov al, '3'
529         mov bl, 15
530         mov bh, 0 ;Display page
531         mov ah, 0Eh ;Teletype
532         int 10h
```

```
533         popa
534         ret 4
535     write_letter endp
536 
537     get_fail proc
538 
539         mov bp, sp
540         pusha
541 
542         lea si, succeed
543         cmp [si], 1
544         je exit_get_fail
545 
546 
547         lea si, fail_place
548         mov dl, [si]
549         ;mov dl, 6 ;column = 6
550         mov ah, 02h
551         xor bh, bh ;page = 0
552         mov dh, 22 ;row = 24
553         int 10h
554 
555         mov ah, 08h
556         int 10h
557 
558         inc dl
559         mov [si], dl
560 
561         mov al, [bp +2]
562         mov ah, 09h
563         mov bl, 0ch
564         mov cx, 1
565         int 10h
566 
567         lea si, fail_number
568         inc [si]
569 
570        exit_get_fail:
571         lea si, succeed
572         mov [si], 0
573         popa
574         ret 2
575 
576     get_fail endp
577 
578     proc hangman_display
579         pusha
580          lea si, hangmann_title
581         mov dl, 0
```

```
582         xor cl, cl
583        write_label3:
584         ;mov dl, cl ;Column start at 3
585         mov dh, 2 ;Row
586         mov bh, 0 ;Display page
587         mov ah, 02h ;SetCursorPosition
588         int 10h
589
590         mov al, [si]
591         inc si
592         ;mov al, '3'
593         mov bl, 2h ;Color is red
594         xor bh, bh ;Display page
595         mov ah, 0Eh ;Teletype
596         int 10h
597         inc dl
598         cmp [si], "$"
599         jne write_label3
600         popa
601         ret
602     hangman_display endp
603
604     proc game
605         pusha ;push all the general purpose registers to the stack, prevents ...
                information loss
606         call initialize_screen
607         call hangman_display
608         call draw_base
609         call open_read_file
610         call get_random_number
611         call get_random_word
612         call get_word_len
613         call draw_lines
614         call fail_label
615
616         main_game:
617         call get_letter
618         call check_win
619         cmp [did_win],1 ; if win exit game
620         je exit_game
621         lea bx, fail_number
622         cmp [fail_number], 1 ; fail 1 time draw head
623         je call_draw_head
624         cmp [fail_number], 2 ; fail 2 time draw body
625         je call_draw_main_body
626         cmp [fail_number], 3
627         je call_draw_first_hand ;fail3 time draw hand
628         cmp [fail_number], 4
629         je call_draw_second_hand ;fail 4 time draw another hand
```

```
630         cmp [fail_number], 5 ; fail 5 time draw first leg
631         je call_draw_first_leg
632         cmp [fail_number], 6 ; fail 6 time draw seconf leg
633         je call_draw_second_leg
634         cmp [fail_number], 7 ; fail 7 time write all letters and print lose msg
635         je call_write_all_word
636         jmp main_game
637
638
639
640        call_draw_head:
641         call man_head
642         jmp main_game
643
644        call_draw_main_body:
645         call draw_main_body
646         jmp main_game
647
648        call_draw_first_hand:
649         call first_hand
650         jmp main_game
651
652        call_draw_second_hand:
653         call second_hand
654         jmp main_game
655
656        call_draw_first_leg:
657         call first_leg
658         jmp main_game
659
660        call_draw_second_leg:
661         call second_leg
662         jmp main_game
663        call_write_all_word:
664         call write_all_word
665     exit_game:
666         popa
667         ret
668     game endp
669
670
671 start:
672     mov ax,data
673     mov ds,ax
674
675     call game
676
677     mov ax,4c00h
678     int 21h
```

```
679
680 ends
681 end start
```

Results and explanation of Code

Code Explanation: Words for hangman are stored in a hangman_words.tex file. The file is opened using *int 21h/ 3D h*. The contents of file is read using *int 21h/ 3F h*. A random number is generated using system time. Current seconds and milliseconds are added and are divided by 10 to generate a random number. From the file randomnumberth word is selected and stored in word variable. By comparing words character to $ and increasing count, we determine word length. Later screen is initialized in graphics mode and title, hangman base, fails string and dashes for word are displayed on screen.

Later a character is taken from user using *int 21h/ 7h* and is compared with each characters of original word. If letter is present in word, user succeeds and correct_word variable is updated from string of null to character at desired placed and is displayed over the screen. We move on to construct hangman if the character isn't found in the original word. For constructing hangman, a head is drawn using defined array using *int 10h*. Similarly main body is drawn using fixed column and moving 30 rows. For hands columns are incremented as well since slant line is reuired. In similar manner leg and base are defined.

For every step check_win function is executed wherein the program checks if the number of letters in the word are equal to or less than the number of letters which have been correctly guessed so far. If both are not equal then the game continues and if they are equal then we move to the win block which displays the entire word in green text and writes below "HURRAY, YOU WIN!!!" at the bottom showing the users victory else "BETTER LUCK NEXT TIME" shows you lose.

Results: The following are the screenshots of the emulator 8086 and the results:

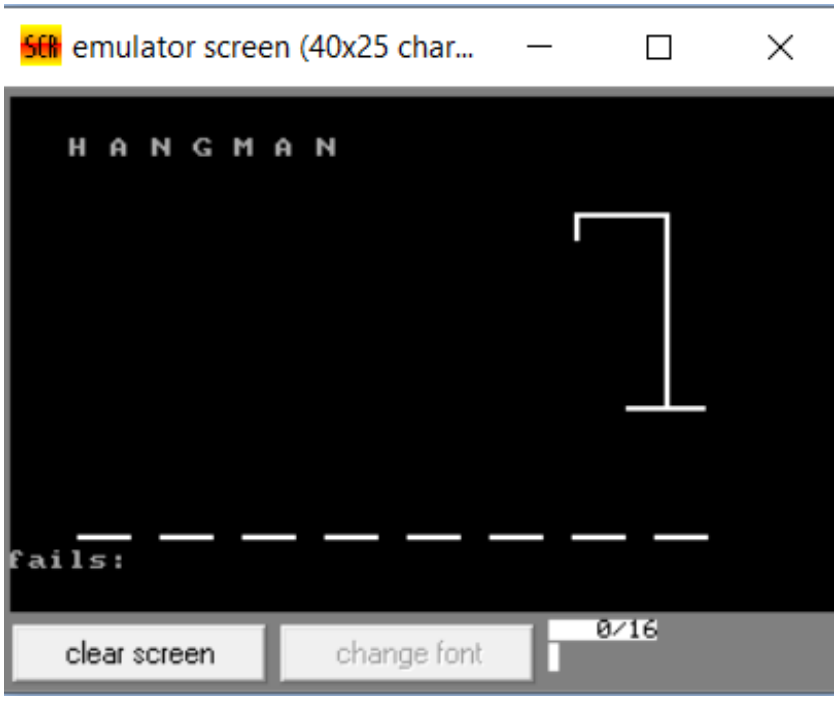

Figure 2: Initial Screen

Figure 3: Display after winning game

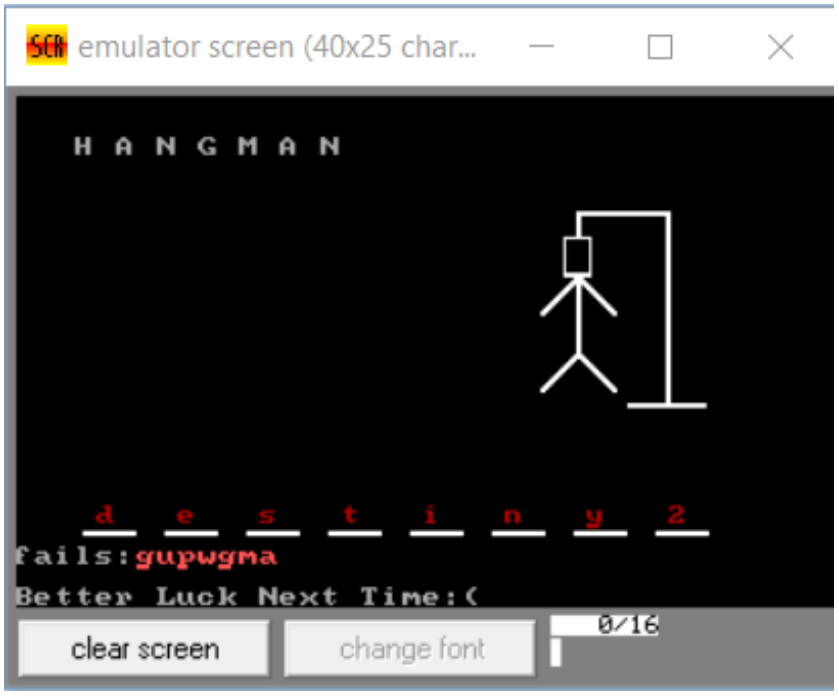

Figure 4: Display after losing Game

Conclusion

We have successfully implemented hangman in assembly language. The game is user interactive and can be played using emu8086. We defined near proc for good code practices and learnt concepts of various interrupts.

For future extension, the game considers repeated characters as well. We can improve it to not take repeated characters. Various levels can be introduced easy, medium and hard etc.