```
@
@ craps.s
@   Written in ARM Assembly by S. Caruthers, 2021
@   Basic dice game of Pig
@   Requires functions in dice_functions.s
@
@   Registers used consistently:
@       r5  -  Current player total score (banked + hand)
@       r6  -  Current Player: 0 = Human, 1 = Computer
@
@   Assemble the program using Gnu C Compiler:
@       gcc -o pig pig_dice.s dice_functions.s
@
@   Execute the program using command:
@       ./pig
@
@   Due to ASCI graphics, the terminal is assumed
@   to be using a monospaced font and is at least
@   50 cols wide by 28 rows tall.

@ Define my Raspberry Pi
        .cpu    cortex-a53
        .fpu    neon-fp-armv8

@ ---------------------------------
@   Constants
@ ---------------------------------

.equ STDOUT, 1                          @ Linux output console
.equ EXIT,   1                          @ Linux syscall
.equ WRITE,  4                          @ Linux syscall
.equ sys_nanosleep, 162                 @ high-resolution sleep
.equ HUMAN, 0                           @ Flag for Human Player
.equ ROLL, 1                            @ Flag to roll again
.equ HOLD, 2                            @ Flag to hold and bank hand


@ ---------------------------------
@   Data Section
@ ---------------------------------

    .data
    .balign 4

@ Strings for messages, printf and scanf formats:

ascii_banner1:
                .ascii "     _     _.....__     _   _        \n"
                .ascii "  '-(-'  .'         '.  | \\_/ |      \n"
                .ascii "     `'-/             \\/       \\     \n"
                .ascii "       |                `    6 6 |_   \n"
                .ascii "       |                        /..\\  \n"
                .ascii "        \\          |          ,_\\__/  \n"
                .ascii "         /       /     /  __.--'     \n"
                .ascii "        <     .-;'----`\\  \\ \\        \n"
                .ascii "         \\   \\\  \\\        \\\  \\\ \\\      \n"
                .asciz "          \\___\\__\\\       \\__\\_\\     \n\n"
```

```
58  ascii_banner2:
59                      .ascii "                                    \n"
60                      .ascii "                                    \n"
61                      .ascii "                                    \n"
62                      .ascii "                                    \n"
63                      .ascii "                                    \n"
64                      .asciz "                                    \n\n"
65  welcome_msg:        .ascii "\033[1;97mWelcome to the Dice Game of Pig!!\n\033[0m"
66                      .ascii "The goal is to be the first to total 100 points.\n"
67                      .ascii "Points are earned by rolling the die repeatedly.\n"
68                      .ascii "For each roll, the amount is added to the total\n"
69                      .ascii "for the hand.  You can HOLD at any time to lock in\n"
70                      .ascii "the total for the hand.  But if you roll a 1, you\n"
71                      .ascii "lose the points for the hand.  Keep repeating\n"
72                      .asciz "until the banked total is 100 or more.\n\n"
73  cont_string:        .asciz "Hit <Enter> to continue..."
74  human_turn_str:     .asciz "    Human's Turn              \n\n"
75  comp_turn_str:      .asciz "    Computer's Turn           \n\n"
76  comp_think_str:     .asciz "    Computer is playing...    \n\033[K\n\033[K\n\033[K"
77  comp_holds_str:     .asciz "    Computer holds...         \n\033[K\n\033[K\n\033[K"
78  human_win_msg:      .asciz "\007\033[1;32m\n\n             !! You Win !!\n\n"
79  comp_win_msg:       .asciz "\033[1;31m\n\n           Computer Wins\n\n"
80  erase_string:       .asciz "\033[F\033[K \n"   @ go to previous line & delete it
81  int_format:         .asciz "%d"
82  roll_choice_str:    .ascii "What would you like to do?\n"
83                      .ascii "1. Roll Again\n"
84                      .ascii "2. Hold and Bank Hand\n"
85                      .asciz "   Enter the # of your choice: "
86  roll_choice:        .int   0
87
88     @ Constructs for player score printout.
89     @ _asc is actually the location to write the value as ascii string.
90  plyr_score:         .int  0          @ actual int values of scores
91  comp_score:         .int  0          @ for player and computer
92  hand_score:         .int  0
93  plyr_score_str:     .ascii "\033[12;5HYour Score: "
94  plyr_score_asc:     .asciz "0      "
95  plyr_score_prog:    .ascii "\033[12;23H"
96  plyr_score_bar:     .asciz "|----------|"
97  comp_score_str:     .ascii "\033[13;5HComp Score: "
98  comp_score_asc:     .asciz "0      "
99  comp_score_prog:    .ascii "\033[13;23H"
100 comp_score_bar:     .asciz "|----------|\n"
101 curr_play_window:   .asciz "\033[14;1H"
102 curr_hand_str:      .ascii "\033[19;12HCurrent Hand: "
103 hand_score_asc:     .asciz "0      "
104 query_window:       .asciz "\033[21;1H"
105
106
107 timespec:                           @ time structure for sleep
108     timespecsec:    .word   0
109     timespecnano:   .word   1000000
110
111 @ ---------------------------------
112 @   Code Section
113 @ ---------------------------------
114
```

```
115     .text
116     .global main                    @ make main callable by all
117
118 @ ---------------------------------
119 @   Code Section -- MACROS
120 @ ---------------------------------
121
122 @-----------------------------
123 @ modulo - Macro to calculate the remainder between two numbers
124 @                   Input Parameters are
125 @                       the dividend register (numerator) and
126 @                       the divisor register (denominator)
127 @                   Output Parameter
128 @                       r0 : the return register (to hold remainder)
129
130 .MACRO  modulo      dividend, divisor
131                 udiv     r0, \dividend, \divisor @ calculate qutotient -> r0
132                 mul      r0, r0, \divisor        @ temp to find remain -> r0
133                 sub      r0, \dividend, r0       @ remainder -> r0
134 .ENDM
135
136 @-----------------------------
137 @ msSleep - Macro to sleep ms milliseconds
138 @                   Input Parameters are ms
139
140 .MACRO  msSleep     ms
141                 push     {r0-r2,r7}
142                 mov      r2, #\ms
143             1:
144                 ldr      r0, =timespecsec
145                 ldr      r1, =timespecsec
146                 mov      r7, #sys_nanosleep
147                 svc      0
148                 subs     r2, #1
149                 bhi      1b
150                 pop      {r0-r2,r7}
151 .ENDM
152
153 @ ---------------------------------
154 @   Code Section -- Main
155 @ ---------------------------------
156
157 @ ---------------------------------
158 @   main:  Dice game of PIG
159 @           param: none
160 @           requires functions in dice_functions.s
161 @           returns: nothing
162
163 main:
164
165         push    {ip, lr}            @ push return address and ip for alignment
166
167          @ Initialize things:
168          bl     seedRandom          @ call function to seed random number generator
169          bl     clrscrn             @ clear the screen
170
171          @ Print Banner and Welcome Message:
```

```
172             bl      pinkText            @ make banner text pink
173             ldr     r0, =ascii_banner1  @ point to graphic banner part 1
174             bl      my_print            @ and print it.
175             ldr     r0, =ascii_banner2  @ point to graphic banner part 2
176             bl      my_print            @ and print it.
177
178             bl      resetText            @ make Welcome text system default
179             ldr     r0, =welcome_msg     @ point to the welcome message
180             bl      my_print
181
182             bl      waitEnter           @ wait for user to clear welcome screen
183
184             @ Print Inital Score Screen Banner:
185             bl      clrscrn             @ clear the screen
186             bl      pinkText            @ make banner text pink
187             ldr     r0, =ascii_banner1  @ point to graphic banner part 1
188             bl      my_print            @ and print it.
189             bl      resetText           @ return to system default font
190
191             @ Initialize Progress Bars & Print Scores...
192             bl      update_human_score  @ ...for human
193             bl      update_comp_score   @ ...and computer
194
195
196     top_loop:
197
198             @ Randomly choose first player by rolling die.
199             @ If 1-3, the Human, else, Computer
200             @ Then print who it is
201
202             bl    rollDie               @ r0 gets value
203             cmp   r0, #3
204             movle r6, #HUMAN             @ if <= 3, then Human (0)
205             movgt r6, #(HUMAN+1)         @ if >3, then Computer (1)
206
207     new_player_loop:
208             bl    println
209             cmp   r6, #HUMAN            @ Depending on which player
210             ldreq r0, =human_turn_str   @ pick the right string
211             ldrne r0, =comp_turn_str
212             bleq  greenText             @ make text green for human, and
213             blne  pinkText              @ pink for computer
214             bl    my_print              @ and print it.
215             bl    resetText
216
217             bl    waitEnter
218
219             @ zero out current hand score before starting
220             bl    zero_hand_total
221
222             @ Get current player score into r5 to check for running total > 100
223             cmp   r6, #HUMAN
224             ldreq r5, =plyr_score       @ point to the right variable
225             ldrne r5, =comp_score
226             ldr   r5, [r5]              @ load score into r5
227
228     curr_hand_loop:
```

```
229             ldr   r0, =curr_play_window
230             bl    my_print              @ move cursor to same spot each time
231
232             bl    rollDie               @ roll a Die and put value in r0
233
234             @ Print the die (with animation):
235             mov   r1, #14               @ column to put die into
236             bl    animate_die
237
238             cmp   r0, #1                @ was the roll a 1?
239             beq   quit_hand             @ if so, this hand is over
240                                         @ else,
241             add   r5, r5, r0            @ add roll to temporary running total Score
242             bl    calc_hand_total       @ add this roll to the current hand and return
                in r0
243             bl    print_hand_score
244
245             @ Has player reached 100? If so, end play
246             cmp   r5, #100              @ has player exceeded 100 for the win?
247             bge   its_a_winner          @ If so, quit
248
249             @ Get player decision to bank or roll
250             mov   r0, r6                @ put current player into r0 for call
251             bl    get_action_choice     @ go to general routine to get choice in r0:
252                                         @ 1 means keep rolling (#ROLL)
253                                         @ 2 means bank and quit (#HOLD)
254             cmp   r0, #HOLD
255             beq   hold_bank_hand        @ if 2, bank it
256
257             bal   curr_hand_loop        @ else, roll again
258
259     hold_bank_hand:
260             mov   r0, r6            @ put current player in r0
261             bl    update_player_score
262
263     quit_hand:
264             msSleep  500                  @ pause 1/2 sec
265             bl    zero_hand_total
266             bl    print_hand_score
267             ldr   r0, =(erase_string+3)   @ point to erase to end of line
268             bl    my_print
269
270             eor   r6, r6, #1              @ XOR with 1 to toggle Player
271             bal   new_player_loop
272
273     its_a_winner:
274             mov   r0, r6                  @ put current player into r0
275             bl    show_winner_message     @ print the right message
276
277             msSleep 500
278             pop   {ip, pc}
279
280
281  @ ---------------------------------
282  @   Code Section -- Subroutines
283  @ ---------------------------------
284
```

```
285 @ ----------------------------------
286 @   waitEnter(): prompts user to hit Enter, then waits
287
288 waitEnter:
289         @ param:  nothing
290         @ returns nothing, and does not use anything the user inputs
291         @ relies on two strings in memory.  One for the prompt, the other to erase it.
292
293         push    {r0-r4, lr}             @ protect some of the registers
294
295         ldr     r0, =cont_string        @ print the prompt string
296         bl      my_print
297
298     wait_char_loop:
299         bl      getchar
300         cmp     r0, #0x0A               @ did user type more than <enter> ?
301         bne     wait_char_loop          @ keep reading characters until CR.
302
303         ldr     r0, =erase_string       @ erase the prompt
304         bl      my_print
305
306         pop     {r0-r4, pc}
307
308
309 @ ----------------------------------
310 @   display text using svc (with size calculation)
311
312 my_print:
313       @ param: r0 contains the address of the null-terminated string
314       @ returns nothing
315
316         push    {r0,r1,r2,r7,lr}                @ save  registers
317         mov     r2,#0                           @ counter length */
318   my_print_loop:                                @ loop length calculation
319         ldrb    r1,[r0,r2]                      @ read octet start position + index
320         cmp     r1,#0                           @ if 0 it is over
321         addne   r2,r2,#1                        @ else add 1 to the length
322         bne     my_print_loop                   @ and loop
323                                                 @ so here r2 contains the length of ↵
                                                    the message
324         mov     r1,r0                           @ address message in r1
325         mov     r0, #STDOUT                     @ code to write to the standard ↵
            output Linux
326         mov     r7, #WRITE                      @ code call system "write"
327         svc #0                                  @ call system
328         pop     {r0,r1,r2,r7,pc}                @ restore registers & return
329
330
331 @ ----------------------------------
332 @     convert an unsigned int into ascii string
333
334 my_itoa:
335       @ Input:
336       @       r0 contains the integer
337       @       r1 contains the address of buffer for string output
338       @ Output:
339       @       r0 contains # of characters in string
```

```
340         @      [r1] address of buffer contains null terminated string
341         @ Warning:
342         @      r1 must point to an address with enough allocated memory
343         @      such that it can hold a string long enough for the number
344         @      of digits in the int to be converted.
345         @      No error checking for this is present!!
346
347           push    {r1-r5, lr}
348           mov     r2, #10                     @ use base 10
349           mov     r3, r0                      @ hold the number
350           mov     r4, r1                      @ hold the buffer address
351           mov     r5, #1                      @ counter for num chars
352       itoa_loop1:
353           modulo  r3, r2                      @ get remainder in r0
354           add     r0, r0, #'0'                @ add to ascii '0'
355           push    {r0}                        @ least sig digit, save it
356           udiv    r3, r3, r2
357           cmp     r3, #0
358           beq     itoa_quit                   @ if result is zero, done extracting
359           add     r5, #1                      @ otherwise, add one to counter
360           bal     itoa_loop1                  @ and go do another digit
361       itoa_quit:
362           mov     r0, r5                      @ first save the count of digits to return
363       itoa_loop2:
364           pop     {r3}                        @ get the saved digits and
365           strb    r3, [r4],#1                 @ put into buffer, inc pointer
366           subs    r5, #1                      @ decrement digit count
367           beq     itoa_exit                   @ if zero, we are done so exit
368           bal     itoa_loop2                  @ otherwise get the next
369       itoa_exit:
370           mov     r3, #0
371           strb    r3, [r4]                    @ put a null terminator
372           pop     {r1-r5, pc}
373
374   @ ------------------------------------
375   @     update the Current Player Score and Display it
376
377   update_player_score:
378      @ Params:
379      @     r0 - Current Player, 0 = Human, 1 = Computer (typically from r6)
380      @ Outputs:
381      @     Returns nothing
382      @     Updates the global variable for the Player Score
383      @     then, calls the appropriate subrouting to update screen
384      @
385      @  Protects all registers, including r0
386
387           push     {r0-r4, lr}
388
389           @ First retrieve the score for current hand
390           ldr      r4, =hand_score
391           ldr      r4, [r4]
392
393           @ now figure out which player and go to that section
394           cmp      r0, #HUMAN              @ if player is NOT human (0),
395           bne      comp_player_calc        @ then go to comp section,
396                                            @ otherwise continue to human
```

```
397         human_player_calc:
398            @ add current score to players total score, then update screen
399            ldr      r1, =plyr_score         @ player is human
400            ldr      r2, [r1]                @ get player score
401            add      r2, r2, r4              @ add current hand to it
402            str      r2, [r1]                @ put it back to memory
403            bl       update_human_score      @ then update screen
404            bal      exit_update_player_score
405
406         comp_player_calc:
407            @ add current score to players total score, then update screen
408            ldr      r1, =comp_score         @ player is computer
409            ldr      r2, [r1]                @ get player score
410            add      r2, r2, r4              @ add current hand to it
411            str      r2, [r1]                @ put it back to memory
412            bl       update_comp_score       @ then update screen
413
414         exit_update_player_score:
415            pop      {r0-r4, pc}
416
417   @ ----------------------------------
418   @     update the Score and progress bar for Human Player
419
420   update_human_score:
421      @ Params: none, but relies on global variables of plyr_score and bar
422      @ Returns nothing, but updates screen with Player current score and progress bar
423
424            push   {r0-r1, lr}
425
426            @ Update Progress Bars & Print Scores (after converting int to ascii):
427            ldr    r0, =plyr_score
428            ldr    r0, [r0]              @ put score into r0, and
429            ldr    r1, =plyr_score_bar   @ progress bar into r1
430            bl     update_progress_bar
431
432            ldr    r1, =plyr_score_asc   @ point to score string
433            bl     my_itoa               @ assume r0 still holds score!
434
435            ldr    r0, =plyr_score_str   @ point to player score string construct
436            bl     my_print
437            ldr    r0, =plyr_score_prog  @ point to progress bar
438            bl     my_print
439
440            pop    {r0-r1, pc}
441
442
443   @ ----------------------------------
444   @     update the Score and progress bar for Computer Player
445
446   update_comp_score:
447      @ Params: none, but relies on global variables of comp_score and bar
448      @ Returns nothing, but updates screen with Computers current score and progress bar
449
450            push  {r0-r1, lr}
451
452            ldr    r0, =comp_score
453            ldr    r0, [r0]              @ put score into r0, and
```

```
454             ldr     r1, =comp_score_bar     @ progress bar into r1
455             bl      update_progress_bar
456
457             ldr     r1, =comp_score_asc     @ point to score string
458             bl      my_itoa                 @ assume r0 still holds score!
459
460             ldr     r0, =comp_score_str     @ point to comp score str construct
461             bl      my_print
462             ldr     r0, =comp_score_prog    @ point to progress bar
463             bl      my_print
464
465             pop    {r0-r1, pc}
466
467
468     @ ------------------------------------
469     @     update the Score progress bar
470
471     update_progress_bar:
472           @ Params:
473           @       r0 contains the score
474           @       r1 contains the address of the string with progress bar to be updated
475           @ Returns nothing and preserves all registers
476
477           @ Progress bar is a string of 12 characters.
478           @ Char [0] and [11] are the end points, Chars [1] - [10] are % progress markers
479           @ Since winning score is 100, there is 1 marker per 10 points.
480
481              push     {r0-r4, lr}
482
483              mov      r3, #0                     @ counter for # of 10s
484              mov      r4, #'#'                   @ hold character for progress bar
485
486        upb_count_loop:
487              subs     r0, #10                    @ subtract 10 from score
488              ble      upb_done_count             @ if <= 0, then we are done
489              add      r3, r3, #1                 @ otherwise, add one to counter
490              bal      upb_count_loop             @ go back and repeat
491
492        upb_done_count:
493              cmp      r3, #0                     @ if counter is 0, nothing to do
494              beq      upb_exit
495
496        upb_update_loop:
497              strb     r4, [r1, r3]               @ put "progress" char at location r3
                 inside string
498              subs     r3, #1                     @ then decrement counter and repeat...
499              bne      upb_update_loop            @ if not equal to 0
500
501        upb_exit:
502              pop      {r0-r4, pc}
503
504     @ ------------------------------------
505     @     Update the total of current Hand
506
507     calc_hand_total:
508        @ Params:
509        @        r0 contains the current roll to add to the sum
```

```
510     @        uses global addresses for:
511     @           int hand_score
512     @           string hand_score_asc
513     @ Returns:
514     @        r0 contains the new total for the hand,
515     @           also updates the glogal variables
516
517           push     {r1-r2, lr}              @ protect everything except r0
518
519           ldr      r1, =hand_score          @ point to current score
520           ldr      r2, [r1]                 @ and put it in r2
521
522           add      r0, r0, r2               @ add current roll to current sum
523           mov      r2, r0                   @ store a copy in r2
524
525           str      r0, [r1]                 @ put it back in global
526           ldr      r1, =hand_score_asc      @ point to string global of score
527           bl       my_itoa                  @ and put the new sum, r0, in
528
529           mov      r0, r2                   @ move it back to r0 for return
530           pop      {r1-r2, pc}
531
532   @ ---------------------------------
533   @    Zero out the total of current Hand
534
535   zero_hand_total:
536      @ Params:
537      @     none
538      @ Returns nothing
539
540           push  {r0-r1, lr}
541           mov   r0, #0
542           ldr   r1, =hand_score            @ store 0 in hand_score int variable
543           str   r0, [r1]
544           ldr   r1, =hand_score_asc
545           bl    my_itoa                    @ and in the string variable
546           pop   {r0-r1, pc}
547
548   @ ---------------------------------
549   @    Print out the total of current Hand
550
551   print_hand_score:
552           push  {r0-r1, lr}
553           bl    whiteBldText
554           ldr   r0, =curr_hand_str   @ point to string for current hand score
555           bl    my_print
556           bl    resetText
557           pop   {r0-r1, pc}
558
559   @ ---------------------------------
560   @     General subroutines to get decision to bank or roll
561
562   get_action_choice:
563      @ Params:
564      @     r0 - current player (from r6)
565      @ Returns:
566      @     r0 - Choice (1 to roll, 2 to hold)
```

```
567     @     Independent of player (Human or Computer)
568
569           push  {r1-r2, lr}
570           cmp   r0, #HUMAN          @ is the current user Human?
571           beq   gac_human           @ if yes, go ask the results
572           bne   gac_computer        @ else, go get computer response
573
574     gac_human:
575           bl    get_user_choice
576           bal   gac_exit
577     gac_computer:
578           bl    get_comp_choice
579
580     gac_exit:
581           @ r0 should have the choice
582           pop   {r1-r2, pc}
583
584   @ ---------------------------------
585
586   get_user_choice:
587           push  {r1-r2, lr}
588
589     guc_loop:
590           @ Ask user to bank or roll again
591           ldr   r0, =query_window    @ point to string to locate quesitons
592           bl    my_print
593           ldr   r0, =roll_choice_str @ point to string to ask what to do
594           bl    my_print
595
596           @ get choice into global variable
597           ldr     r1, =roll_choice    @ ptr for answer
598           ldr     r0, =int_format
599           bl      scanf               @ user response in [r1], (get it later)
600
601     clear_buffer:                     @ scanf leaves \n in buffer, so
602           bl      getchar             @ go through the buffer, before exit
603           cmp     r0, #0x0A           @ is it a CR ?
604           bne     clear_buffer        @ keep reading characters until CR.
605
606           @ Erase the questions
607           ldr   r0, =query_window
608           bl    my_print
609           ldr   r0, =(erase_string+3)   @ point to erase to end of line
610           bl    my_print
611           bl    my_print
612           bl    my_print
613           bl    my_print
614           bl    my_print
615
616           @ get user choice back in r0 to return
617           ldr   r0, =roll_choice
618           ldr   r0, [r0]
619
620           @ do some error checking
621           cmp   r0, #0                  @ if <= 0, error.
622           ble   guc_loop                @ ask again
623           cmp   r0, #2                  @ if >2, error
```

```
624             bgt   guc_loop                @ ask again
625
626             pop   {r1-r2, pc}             @ otherwise, just return
627
628     @ ---------------------------------
629
630     get_comp_choice:
631             push  {r1-r3, lr}
632
633             @ Print a message about computer turn
634             ldr   r0, =query_window    @ point to string to locate message
635             bl    my_print
636             ldr   r0, =comp_think_str @ point to string to print
637             bl    my_print
638             msSleep  400
639
640             @ Read the global variables of interest
641             ldr   r1, =hand_score   @ get current hand...
642             ldr   r1, [r1]          @ into r1
643             ldr   r2, =comp_score   @ get computers score...
644             ldr   r2, [r2]          @ into r2
645             ldr   r3, =plyr_score   @ get opponent score...
646             ldr   r3, [r3]          @ into r3.
647
648             @ Here is the logic to roll or hold:
649             @ Generally, hold if hand is >20.
650             @ But, if opponent is about to win, always roll (within reason).
651
652             @@ Check if this is a long roll streak and never go beyond 48!
653             cmp   r1, #48           @ do not press your luck beyond 48
654             movge r1, #HOLD         @ If curr hand >= 48, hold
655             bge   gcc_logic_done    @ and exit
656
657             @@ Check if opponent is about to win...
658             cmp   r3, #90           @ if opponent will win next hand,
659             movge r1, #ROLL         @ then set choice to roll
660             bge   gcc_logic_done    @ and exit
661
662             @@ Check if my score is low, and be slightly more aggressive
663             @@ Otherwise, always hold when roll gets to 20
664             cmp   r2, #10           @ if comp score is < 10,
665             movle r3, #25           @   then keep rolling until 25
666             movgt r3, #20           @   otherwise, use 20 as the cut off
667             cmp   r1, r3            @ is it the threshold set above?
668             movlt r1, #ROLL         @ if <20, roll again
669             movge r1, #HOLD         @ if >=20, bank it
670
671          gcc_logic_done:
672             cmp   r1, #ROLL         @ if not rolling, print hold message
673             beq   skip_hold_msg
674             ldr   r0, =query_window
675             bl    my_print
676             ldr   r0, =comp_holds_str     @ point to computer holding str...
677             bl    my_print
678             msSleep  1000
679
680       skip_hold_msg:
```

```
681             @ Erase the message
682             ldr   r0, =query_window
683             bl    my_print
684             ldr   r0, =(erase_string+3)   @ point to erase to end of line
685             bl    my_print
686             bl    my_print
687             bl    my_print
688             bl    my_print
689
690             mov   r0, r1            @ put choice from r1 into r0 to return
691
692             pop   {r1-r3, pc}
693
694     @ -----------------------------------
695     @    Print out the WINNER message for current player (in r0)
696
697     show_winner_message:
698        @ Params:
699        @     r0 - has the value for player (from r6)
700        @ Returns nothing, but prints message to screen
701             push  {r0-r1, lr}
702
703             cmp   r0, #HUMAN
704             ldreq r0, =human_win_msg
705             ldrne r0, =comp_win_msg
706             bl    my_print
707             bl     resetText
708
709             pop   {r0-r1, pc}
710
```