```
;;; Projeto IAC
;;; Master Mind
;;; Primeira Entrega
;;; 27/10/2017

;;; Baltasar Dinis 89416
;;; Vasco Rodrigues 89557


;zona 1: constantes

filtro          EQU     0007h ; filtro para os primeiros 3 bits
mascara         EQU     1000000000010110b ; mascara para usar no modulo random

SP_INICIAL      EQU     FDFFh ; stack pointer
IO_READ         EQU     FFFFh ; input janela de texto
IO_WRITE        EQU     FFFEh ; output janela de texto
IO_STATUS       EQU     FFFDh ; testa se foi premida tecla na janela de texto
IO_CONTROL      EQU     FFFCh ; posiciona o cursor na janela de texto
IO_SW           EQU     FFF9h ; 8 interruptores (alavancas)
IO_LEDS         EQU     FFF8h ; 16 leds (1 por bit)
LCD_WRITE       EQU     FFF5h ; escreve caracter no display
LCD_CONTROL     EQU     FFF4h ; colocar o cursor no display
IO_DISPLAY      EQU     FFF0h ; display de 7 leds
TEMP            EQU     FFF7h ; start/stop temporizador
TEMP_VALUE      EQU     FFF6h ; valor para gerar interrupÃ§Ã£o
INT_MASK_ADDR   EQU     FFFAh
INT_MASK        EQU     8000h
SPACE           EQU     ' ' ; codigo ascii do caracter de espaÃ§o
ZERO            EQU     '0' ; codigo ascii do zero
NROUNDS_LOST    EQU     13 ; pontuacao correspondente a uma jogada perdida; acab
a a contagem quando o contador for igual a pontuacao de perder o jogo ; default
= 13
NROUNDS_MAX     EQU     12 ; pontuacao minima, correspondente ao numero maximo d
e rondas ; default = 12
TEMP_STEP       EQU     5 ; numero que e colocado no temporizador. Atua como reg
ulador da dificuldade do jogo. Quanto mais pequeno mais dificil; default = 5

;zona 2: variaveis

                ORG     8000h

; os valores utilizados pelo jogo sao declarados
; como variaveis para evitar a volatilidade dos registos
; sao utilizados apenas como meio inicial de passagem de parametros
; e como meio final de retorno de parametros

; o utilizador introduz, pelo meio da placa do P3, a jogada, e esta fica
; guardada em memoria
;
; o codigo, sempre que recalculado, e guardado em memoria
;
; o resultado (da verificacao) fica guardado em memoria
;
; a semente e acedida e escrita pela funcao random, para garantir que,
; no caso de multiplas chamadas, o numero e diferente
; a funcao get_seed inicializa a semente com o valor do temporizador
;
; a melhor pontuacao e inicializada em memoria com o valor
; mais baixo que o jogador pode ter, correspondendo ao valor de uma derrota
;
; a pontuacao atual e guardada em memoria
```



```
;
; o tempo restante para o jogador efetuar a sua jogada e guardado em memoria
;
; o temporizador esta guardado em memoria
;
; cursor mimica o funcionamento do porto de controlo da janela de texto

jogada          TAB     1
codigo          TAB     1
resultado       TAB     1
semente         TAB     1
melhor_pont     WORD    NROUNDS_LOST
pont_atual      WORD    1
tempo_restante  WORD    FFFFh
temporizador    TAB     1
cursor          WORD    0000h


; mensagens
inicia_jogo     STR     'Carregue no botao IA para iniciar|'
ganhou          STR     'You won!|'
perdeu          STR     'You lost... Good luck next time|'


;zona 3: interrupcoes

                ORG     FE01h
INT1            WORD    INT1F
INT2            WORD    INT2F
INT3            WORD    INT3F
INT4            WORD    INT4F
INT5            WORD    INT5F
INT6            WORD    INT6F

                ORG     FE0Ah
INTA            WORD    INTAF

                ORG     FE0Fh
INTF            WORD    INTFF


;zona 4: codigo

                ORG     0000h
                JMP     inicio

; interrupcoes

; INT1 coloca o valor 1 no R3
INT1F:          MOV     R3, 1
                RTI

; INT2 coloca o valor 2 no R3
INT2F:          MOV     R3, 2
                RTI

; INT3 coloca o valor 3 no R3
INT3F:          MOV     R3, 3
                RTI

; INT4 coloca o valor 4 no R3
INT4F:          MOV     R3, 4
```

```
                RTI

; INT5 coloca o valor 5 no R3
INT5F:          MOV     R3, 5
                RTI

; INT6 coloca o valor 6 no R3
INT6F:          MOV     R3, 6
                RTI

; INTA coloca o valor do temporizador em R1, recomeÃ§ando o temporizador
INTAF:          MOV     R1, M[temporizador]
                CALL    reset_timer
                CALL    start_timer
                RTI

; INTFF incrementa o temporizador,
INTFF:          PUSH    R1 ; preserva o R1
                INC     M[temporizador] ;incrementa o temporizador de 100ms em 1
00ms segundo
                MOV     R1, 1 ;reinicia o temporizador
                MOV     M[TEMP_VALUE], R1
                MOV     M[TEMP], R1
                POP     R1
                RTI
                RTI

; modulo check
; faz a validacao de uma jogada, excrevendo o resultado em memoria

;; check_play: void
;;
;; modulo que estrutura a verificacao da jogada no ciclo de execucao do jogo
;; serve apenas para aumentar a legibilidade do codigo no cilco de execucao
check_play:     MOV     R2, M[jogada]
                MOV     R3, M[codigo]

                PUSH    R0
                PUSH    R2 ; jogada
                PUSH    R3 ; codigo
                CALL    check
                POP     M[resultado]

                RET


;; check: (int, int) -> int
;;
;; verifica o resultado da jogada, comparando-a com o codigo
;;
;; input: jogada e codigo
;; output: resultado, com o numero de X's no octeto menos significativo e
;; o numero de O's no octeto mais significativo
check:          PUSH    R1 ; preserva R1
                PUSH    R2 ; preserva R2
                MOV     R1, M[SP + 5] ; R1 <- jogada

                CMP     R1,  M[SP + 4] ; compara a jogada com o codigo
                BR.Z    check_direct

                MOV     R2, M[SP + 4] ; R2 <- codigo
                PUSH    R0 ; Resposta
```



```
                PUSH    R1 ; jogada
                PUSH    R2 ; codigo
                CALL    check_xes
                POP     R1
                MOV     M[SP + 6], R1 ; X's
                MOV     R1, M[SP + 5] ; R1 <- jogada

                PUSH    R0 ; resposta
                PUSH    R1 ; jogada
                PUSH    R2 ; codigo
                CALL    check_oes
                POP     R1
                SUB     R1, M[SP + 6] ; retira os valores que foram contados em
check_xes
                ROR     R1, 8 ; coloca a resposta no octeto mais significativo d
e R1
                MVBH    M[SP + 6], R1 ; O's

check_ret:      POP     R2 ; recupera R2
                POP     R1 ; recupera R1
                RETN    2

check_direct:   MOV     R1, 4
                ADD     M[SP + 6], R1 ; <8 bits a 0> + <'X's>
                BR      check_ret


;; check_xes: (int, int) -> int
;;
;; calcula quanto digitos na posicao certa estao na jogada
;;
;; input: jogada e codigo
;; output: numero de pares na mesma posicao
check_xes:      PUSH    R1 ; preserva R1
                PUSH    R2 ; preserva R2
                PUSH    R3 ; preserva R3
                PUSH    R4 ; preserva R4
                PUSH    R5 ; preserva R5

                MOV     R1, 4 ; contador ciclo
                MOV     R2, R0 ; contador X's
                MOV     R3, filtro ; filtro

ciclo_xes:      MOV     R4, M[SP + 8] ; recupera a jogada
                MOV     R5, M[SP + 7] ; impede R2 de ser destuido / recupera o c
odigo

                AND     R4, R3 ; aplica o filtro
                AND     R5, R3 ; aplica o filtro
                CMP     R4, R5
                BR.NZ   salto_xes

                INC     R2 ; incrementa o contador

salto_xes:      ROL     R3, 3 ; muda o filtro de posicao
                DEC     R1
                BR.NZ   ciclo_xes

                MOV     M[SP + 9], R2 ; retorna a resposta

                POP     R5 ; recupera R5
                POP     R4 ; recupera R4
```

```
                POP     R3 ; recupera R3
                POP     R2 ; recupera R2
                POP     R1 ; recupera R1
                RETN    2


;; check_oes: (int, int) -> int
;;
;; verifica quantos pares em comum o codigo tem com a jogada
;; independentemente da posicao em que se encontram
;;
;; input: jogada e codigo
;; output : numero de pares de digitos em comum entre o codigo e a jogada
check_oes:      PUSH    R1 ; preserva R1
                PUSH    R2 ; preserva R2
                PUSH    R3 ; preserva R3
                PUSH    R4 ; preserva R4
                PUSH    R5 ; preserva R5

                MOV     R1, 4; contador do ciclo
                MOV     R2, M[SP + 7] ; R4 <- codigo
                MOV     R4, R0 ; contador dos O's
                MOV     R5, M[SP + 8] ; R5 <- jogada

ciclo_oes:      MOV     R3, 8 ; ao dividir, o resto fica em R3

                DIV     R5, R3 ; isola um digito
                PUSH    R0
                PUSH    R2 ; codigo (nao e preservado para guardar as mudancas e
 nao repetir a contagem)
                PUSH    R3 ; digito
                CALL    check_digit
                POP     R3
                ADD     R4, R3 ; atualiza o resultado

                DEC     R1

                BR.NZ   ciclo_oes

                MOV     M[SP + 9], R4 ; escreve o valor de retorno
                POP     R5 ; recupera R5
                POP     R4 ; recupera R4
                POP     R3 ; recupera R3
                POP     R2 ; recupera R2
                POP     R1 ; recupera R1
                RETN    2

;; check_digit: (int, int, int) -> bool
;;
;; verifica se o digito, obtido pela aplicacao do filtro, esta presente no codig
o
;;
;; input: codigo, digito e filtro
;; output: 1 se o digito existir; 0 caso contrario
check_digit:    PUSH    R1 ; preserva R1 ; nao preserva de R2, para saber quais
digitos ja foram retirados
                PUSH    R3 ; preserva R3
                PUSH    R4 ; preserva R5
                PUSH    R5 ; preserva R4


                MOV     R1, 4 ; contador
```



```
                MOV     R2, M[SP + 7] ; R2 <- codigo - valores previamente encon
trados na funcao
                MOV     R4, M[SP + 6]
                MOV     R5, filtro

ciclo_digit:    MOV     R3, R2 ; preserva localmente o codigo
                AND     R3, R5 ; aplica filtro

                CMP     R3, R4 ; verifica se o digito e igual
                BR.NZ   salto_digit
                INC     M[SP + 8] ; incrementa o valor de retorno, passa a 1
                OR      R2, R5 ; atualiza o codigo -> coloca-se o digito a 7
                BR      ret_digit

salto_digit:    ROL     R5, 3 ; atualiza a posicao do filtro
                ROL     R4, 3 ; atualiza a posicao do digito
                DEC     R1
                BR.NZ   ciclo_digit

ret_digit:      POP     R5 ; recupera R5
                POP     R4 ; recupera R4
                POP     R3 ; recupera R3
                POP     R1 ; recupera R1
                RETN    2


; modulo print
; imprime o output do programa

;; print: void
;;
;; modulo que estrutura a impressao do output do programa no ciclo de execucao d
o jogo
;; serve apenas para aumentar a legibilidade do codigo no ciclo de execucao
print:          MOV     R2, R0 ; limpa o R2
                MOV     R3, R0 ; limpa o R2
                MVBH    R2, M[resultado] ; Oes
                ROR     R2, 8 ; coloca o octeto na posicao correta
                MVBL    R3, M[resultado] ; Xes
                PUSH    R2
                PUSH    R3
                PUSH    M[jogada]
                PUSH    R1 ; coloca o contador para impressao
                CALL    print_line  ; imprime a linha

                RET


;; print_char: char -> void
;;
;; imprime um caracter na JV
;;
;; input: caracter a imprimir
print_char:     PUSH    R1 ; preserva R1
                MOV     R1, M[cursor]
                MOV     M[IO_CONTROL], R1 ; coloca o cursor no lugar correto
                MOV     R1, M[SP + 3]  ;coloca o caracter em R1
                MOV     M[IO_WRITE], R1  ;imprime
                INC     M[cursor] ; atualiza o cursor
                POP     R1  ; recupera o R1
                RETN    1
```

```
;; newline: -> void
;;
;; introduz uma nova linha
newline:        PUSH    R1 ; preserva R1
                MOV     R1, M[cursor]
                AND     R1, FF00h ; coloca o cursor na coluna zero
                ADD     R1, 0100h ; adiciona 1 a linha
                MOV     M[cursor], R1
                MOV     M[IO_CONTROL], R1 ; atualiza o cursor
                POP     R1 ; recupera R1
                RET

;; print_str: str -> void
;;
;; imprime uma cadeia de caracteres (terminada pelo caracter '|') numa linha da
JV
;;
;; input: endereco da cadeia de caracteres a imprimir
print_str:      PUSH    R1 ; preserva R1
                PUSH    R2 ; preserva R2

                MOV     R1, M[SP + 4] ; coloca o endereco da string em R1
                MOV     R2, '|' ; coloca o valor ascii do caracter terminal em R
2
ciclo_str:      CMP     M[R1], R2 ; verifica se o caracter e o terminal
                BR.Z    fim_str
                PUSH    M[R1]
                CALL    print_char ; imprime o caracter
                INC     R1 ; incrementa o indice
                BR      ciclo_str

fim_str:        CALL    newline ; imprime uma nova linha
                POP     R2 ; recupera R2
                POP     R1 ; recupera R1
                RETN    1

;; print_digit: int -> void
;;
;; imprime um digito na JV
;;
;; input: digito decimal inteiro a imprimir
print_digit:    PUSH    R1  ; preserva R1

                MOV     R1, M[cursor]
                MOV     M[IO_CONTROL], R1 ; coloca o cursor no lugar correto
                MOV     R1, M[SP + 3]  ;coloca o caracter em R1
                ADD     R1, '0'  ;converte para ascii
                MOV     M[IO_WRITE], R1  ;imprime
                INC     M[cursor] ; atualiza o cursor
                POP     R1  ; recupera o R1
                RETN    1

;; print_jogada: int -> void
;;
;; imprime um codigo de 4 digitos (em 12 bits) na JV
;;
;; input: jogada
print_jogada:   PUSH    R1 ; preserva R1
                PUSH    R2 ; preserva R2
                PUSH    R3 ; preserva R3

                MOV     R1, M[SP + 5] ; coloca o parametro em R1
```



```
                ROL     R1, 4 ; "encosta" o nÃºmero
                MOV     R2, 4  ; contador

guess_ciclo:    ROL     R1, 3 ; coloca o digito mais significativo na posiÃ§Ã£o
menos significativa
                MOV     R3, R1 ; preserva o R1
                AND     R3, filtro ; coloca em R3 o bit menos significativo
                PUSH    R3
                CALL    print_digit ;imprime R3

                DEC     R2
                BR.NZ   guess_ciclo

                POP     R3 ; recupera o R3
                POP     R2 ; recupera o R2
                POP     R1 ; recupera o R1
                RETN    1

;; print_count: int -> void
;;
;; imprime o contador (decimal com 2 digitos)
;;
;; input: contador (decimal com 2 digitos)
print_count:    PUSH    R1 ; preserva R1
                PUSH    R2 ; preserva R2

                MOV     R1, M[SP + 4] ; coloca o parametro em R1
                MOV     R2, ah
                DIV     R1, R2 ; em R1 estÃ¡ o algarismo das dezenas e em R2 as
unidades

                BR.Z    unidades ; verifica se o algarismo das dezenas e zero
                PUSH    R1
                CALL    print_digit ; imprime algarismo das dezenas

unidades:       PUSH    R2
                CALL    print_digit ; imprime algarismo das unidades

                POP     R2 ; recupera o R2
                POP     R1 ; recupera o R1
                RETN    1

;; print_result: (int, int) -> void
;;
;; imprime o resultado da jogada
;;
;; input: numero de X's a imprimir e numero de O's a imprimir
print_result:   PUSH    R1 ; preserva R1
                PUSH    R2 ; preserva R2
                PUSH    R3 ; preserva R3

                MOV     R1, M[SP + 6] ; R1 <- #Xes
                MOV     R2, M[SP + 5] ; R2 <- #Oes
                MOV     R3, 4 ; numero de tracos a colocar
                SUB     R3, R1 ; subtrai o numero de X's

                SUB     R3, R2 ; subtrai o numero de O's

print_xes:      CMP     R1, R0
                BR.Z    print_oes ; verifica se o contador esta em 0

                PUSH    'X'
```

```
                CALL    print_char ; imprime 'X'
                DEC     R1
                BR      print_xes

print_oes:      CMP     R2, R0
                BR.Z    print_dash  ; verifica se o contador esta em 0

                PUSH    'O'
                CALL    print_char  ; imprime 'O'
                DEC     R2
                BR      print_oes

print_dash:     CMP     R3, R0
                BR.Z    ret_result  ; verifica se o contador esta em 0

                PUSH    '-'
                CALL    print_char ; imprime '-'
                DEC     R3
                BR      print_dash

ret_result:     CALL    newline ; imprime uma nova linha

                POP     R3 ; recupera o R3
                POP     R2 ; recupera o R2
                POP     R1 ; recupera o R1
                RETN    2

;; print_line:   (int, int, int, int) -> void
;;
;; imprime uma linha, ou seja, o output de uma ronda do programa
;;
;; input: contador, jogada, numero de x's e numero de o's
print_line:     PUSH    R1 ; preserva R1
                PUSH    R2 ; preserva R2

                MOV     R1, M[SP + 4] ; contador de jogada
                PUSH    R1
                CALL    print_count ; imprime o contador

                PUSH    ':'
                CALL    print_char

                PUSH    ' '
                CALL    print_char

                MOV     R1, M[SP + 5] ; guess
                PUSH    R1
                CALL    print_jogada ; imprime a jogada

                PUSH    ':'
                CALL    print_char

                PUSH    ' '
                CALL    print_char

                MOV     R1, M[SP + 6] ; R1 <- #Xes
                MOV     R2, M[SP + 7] ; R2 <- #Oes
                PUSH    R1
                PUSH    R2
                CALL    print_result ; imprime resultado

                POP     R2 ; recupera o R2
```



```
                POP     R1 ; recupera o R1
                RETN 4

;; limpa_jt: void -> void
;;
;; limpa a janela de texto
limpa_jt:       PUSH    R1 ; preserva R1

                MOV     M[IO_CONTROL], R0 ; coloca o cursor no inicio
                MOV     M[cursor], R0 ; atualiza o cursor
                MOV     R1, 10000 ; numero arbitrariamente grande
clear_ciclo:    PUSH    SPACE
                CALL    print_char
                DEC     R1
                BR.NZ   clear_ciclo

                MOV     M[IO_CONTROL], R0 ; coloca o cursor no inicio
                MOV     M[cursor], R0 ; atualiza o cursor
                POP     R1 ; recupera R1
                RET

; modulo random
; gera um numero pseudoaleatorio

;; random: void -> int
;;
;; gera um numero pseudoaleatorio, sendo a seed enderecada por semente
;;
;; output: numero aleatorio hexadecimal em que cada digito esta entre 1 e 6
random:         PUSH    R1 ; preserva R1
                PUSH    R2 ; preserva R2
                PUSH    R3 ; preserva R3
                PUSH    R4 ; preserva R4
                PUSH    R5 ; preserva R5
                PUSH    R6 ; preserva R6

                MOV     R1, 4 ; contador
                MOV     R2, M[semente]
                MOV     R3, R0 ; numero gerado
                MOV     R4, mascara

rand_cycle:     ROL     R3, 3 ; gera espaco para o proximo digito
                MOV     R5, 6 ; nÃºmero mÃ¡ximo

                TEST    R2, 1
                BR.Z    jump
                XOR     R2, R4

jump:           ROR     R2, 1
                MOV     R6, R2 ; preserva R2
                DIV     R6, R5 ; garante que o valor esta entre 0 e 5
                INC     R5 ; garante que o resultado esta entre 1 e 6
                ADD     R3, R5 ;Devolve o resultado para o numero de 4 algarismo
s

                DEC     R1
                BR.NZ   rand_cycle

                MOV     M[semente], R3
                MOV     M[SP + 8], R3
                POP     R6 ; recupera R6
                POP     R5 ; recupera R5
```

```
                POP     R4 ; recupera R4
                POP     R3 ; recupera R3
                POP     R2 ; recupera R2
                POP     R1 ; recupera R1
                RET

;; get_seed: -> int
;;
;; imprime a mensagem de inicio de jogo, inicia o temporizador e aguardando pela
 interruÃ§Ã£o A
;; quando esta acontecer, para o temporizador e devolve o seu valor
;; limpa depois a janela de texto
;;
;; output: semente para a geracao de um numero pseudoaleatorio
get_seed:       PUSH    R1 ; preserva o R1

                MOV     R1, INT_MASK
                ADD     R1, 1024 ; permite a interrupcao INTA
                MOV     M[INT_MASK_ADDR], R1

                MOV     R1, R0

                CALL    reset_timer
                CALL    start_timer

                ENI
                PUSH    inicia_jogo
                CALL    print_str ; imprime a mensagem para iniciar o jogo
wait_for_seed:  CMP     R1, R0 ; como convencionado em INTAF, R1 e alterado
                BR.Z    wait_for_seed ; para ficar com o valor do temporizador q
uando INTA for premido
                DSI

                CALL    limpa_jt
                MOV     M[SP + 3], R1 ; escreve o resultado no stack
                MOV     R1, INT_MASK
                MOV     M[INT_MASK_ADDR], R1 ; bloqueia a interrupcao INTA
                POP     R1 ; recupera R1
                RET

; modulo interface
; faz a gestao dos perifericos, tanto para o input como para o output

;; display: void
;;
;; modulo que estrutura o display, para tornar a funcao da jogada de uma ronda
;; mais legivel
display:        MOV     M[pont_atual], R1 ; guarda o valor atual em memÃ³ria
                CALL    display_current
                CALL    display_best
                RET



;; diplay_best: int -> void
;;
;; imprime o melhor resultado no display LCD
;;
;; input: a melhor jogada, endereÃ§ada melhor_pont
display_best:   PUSH    R1 ; preserva o R1
                PUSH    R2 ; preserva o R2
                MOV     R2, 8020h
```



```
                MOV     M[LCD_CONTROL], R2; liga o LCD e limpa-o

                MOV     R1, M[melhor_pont]
                MOV     R2, 10
                DIV     R1, R2 ; R1, segundo digito - R2, primeiro digito
                ADD     R1, ZERO
                ADD     R2, ZERO
                MOV     M[LCD_WRITE], R1 ; imprime o segundo digito

                MOV     R1, 8001h
                MOV     M[LCD_CONTROL], R1 ; posiciona o LCD na segunda coluna
                MOV     M[LCD_WRITE], R2 ; imprime o segundo digito

                POP     R2 ; recupera R2
                POP     R1 ; recupera R1
                RET

;; diplay_current: int -> void
;;
;; imprime o resultado atual no display de 7 segmentos
;;
;; input: o resultado atual, endereÃ§ado em pont_atual
display_current:PUSH    R1 ; preserva o R1
                PUSH    R2 ; preserva o R2
                PUSH    R3 ; preserva o R3

                MOV     R1, M[pont_atual]
                MOV     R2, 10
                DIV     R1, R2 ; R1, segundo digito - R2, primeiro digito

                MOV     R3, IO_DISPLAY
                MOV     M[R3], R2
                INC     R3
                MOV     M[R3], R1

                POP     R3 ; recupera R3
                POP     R2 ; recupera R2
                POP     R1 ; recupera R1
                RET


;; diplay_time: int -> void
;;
;; mostra o tempo que falta nos LEDS
;;
;; input: o tempo que falta, endereÃ§ado em tempo_restante
display_time:   PUSH    R1 ; preserva o R1
                MOV     R1, M[tempo_restante]
                MOV     M[IO_LEDS], R1
                POP     R1 ; recupera R1
                RET

;; get_input: -> int
;;
;; recebe um input da placa
;; output: uma jogada, guardada em 12 bits, onde cada 3 bits correspondem a um d
igito
get_input:      PUSH    R1 ; preserva R1
                PUSH    R2 ; preserva R2
                PUSH    R3 ; preserva R3
                PUSH    R4 ; preserva R4
                PUSH    R5 ; preserva R5
```

```
                MOV     R1, INT_MASK
                ADD     R1, 1111110b ; ativa as interrupcoes INT1 - INT6
                MOV     M[INT_MASK_ADDR], R1
                MOV     R1, FFFFh
                MOV     M[tempo_restante], R1 ; inicializa o tempo que falta
                MOV     R1, 4 ; contador
                MOV     R2, R0 ; resultado
                MOV     R4, M[tempo_restante]
                MOV     R5, TEMP_STEP ; para comparacao

                ENI
                CALL    display_time
                CALL    reset_timer
                CALL    start_timer

input_macro:    MOV     R3, R0 ; digito, por convencao [ver INT1F - INT6F]
input_micro:    CMP     M[temporizador], R5
                BR.NZ   wait_timer ; se o temporizador chegar aos 5 (500ms), atu
aliza o tempo restante
                SHR     R4, 1 ; apaga um led
                MOV     M[tempo_restante], R4 ; atualiza o tempo restante
                CALL    display_time ; atualiza o display
                CMP     R4, R0
                BR.Z    return_input ; se o tempo for excedido, retorna 0
                CALL    reset_timer
                CALL    start_timer

wait_timer:     CMP     R3, R0
                BR.Z    input_micro
                ROL     R2, 3 ; cria espaÃ§o para o digito
                ADD     R2, R3 ; adiciona o digito
                DEC     R1
                BR.NZ   input_macro

                MOV     R1, INT_MASK
                MOV     M[INT_MASK_ADDR], R1 ; impede as interrupcoes INT1 - INT
6
                DSI
                MOV     M[SP + 7], R2 ; devolve o resultado

return_input:   POP     R5 ; recupera R5
                POP     R4 ; recupera R4
                POP     R3 ; recupera R3
                POP     R2 ; recupera R2
                POP     R1 ; recupera R1
                RET


; modulo temporizador
; temporizador

;; start_timer: void
;;
;; inicia o temporizador
start_timer:    PUSH    R1 ; preserva o R1
                MOV     R1, 1
                MOV     M[TEMP_VALUE], R1
                MOV     R1, 1
                MOV     M[TEMP], R1
                POP     R1 ; recupera R1
                RET
```



```
;; reset_timer: void
;;
;; coloca o temporizador a 0 e desativa o temporizador
reset_timer:    MOV     M[temporizador], R0
                MOV     M[TEMP], R0 ; desativa o temporizador
                RET


;; modulo jogo

;; ronda: void
;;
;; permite jogar uma ronda; estrutura geral do jogo
;; recebe os inputs
;; faz o display e a impressao na janela de texto de todos os outputs
ronda:          PUSH    R1 ; preserva R1
                PUSH    R2 ; preserva R2
                PUSH    R3 ; preserva R3

                PUSH    R0
                CALL    get_seed ; gera uma semente
                POP     M[semente] ; guarda a semente em memoria

                PUSH    R0
                CALL    random ; gera o codigo pseudoaleatorio
                POP     M[codigo] ; armazena-o no endereco codigo

                MOV     R1, 1 ; contador
                CALL    display ; apresenta o display inicial

ciclo_ronda:    PUSH    R0
                CALL    get_input ; recebe input
                POP     M[jogada] ; guarda jogada
                CMP     M[jogada], R0
                BR.Z    etiq_perdeu ; verifica se o tempo foi excedido

                CALL    check_play ; calcula o resultado da jogada
                CALL    print ; imprime a jogada
                CALL    display ; atualiza os perifericos


                MOV     R2, 4
                CMP     M[resultado], R2
                BR.Z    etiq_ganhou ; verifica se o jogo foi ganho

                INC     R1
                CMP     R1, NROUNDS_LOST ; o contador comeca a 1; acaba a contag
em quando o contador for igual a pontuacao de perder o jogo
                BR.NZ   ciclo_ronda

etiq_perdeu:    PUSH    perdeu
                MOV     R1, NROUNDS_LOST
                MOV     M[pont_atual], R1; caso tenha perdido por excesso de tem
po, atualiza a pontuacao adequadamente
                CALL    print_str ; imprime mensagem de derrota
                BR      return_ronda

etiq_ganhou:    PUSH    ganhou
                CALL    print_str ; imprime mensagem de vitoria

return_ronda:   CALL    update_best ; atualiza o recorde
```

```
                POP     R3 ; recupera R3
                POP     R2 ; recupera R2
                POP     R1 ; recupera R1
                RET

;; update_best: (input, input) -> input
;;
;; verifica se a pontuacao da jogada bateu o recorde; caso se verifique,
;; atualiza o recorde adequadamente
;;
;; como as variaveis em questao so existem em memoria, os inputs sao
;; passados diretamente pela memoria
;;
;; inputs: pontuacao atual e recorde
;; output: recorde atualizado
update_best:    PUSH    R1
                PUSH    R2

                MOV     R1, M[pont_atual]
                MOV     R2, M[melhor_pont]

                CMP     R1, R2
                BR.NN   dont_update ; se a diferenca for negativa, e preciso atu
alizar
                MOV     M[melhor_pont], R1
dont_update:    POP     R2 ; recupera R2
                POP     R1 ; recupera R1
                RET


;; main: execucao principal do programa;

inicio:         MOV     R1, SP_INICIAL
                MOV     SP, R1

                MOV     R1, FFFFh
                MOV     M[IO_CONTROL], R1 ; inicializa o porto de controlo da ja
nela de texto

joga_sempre:    CALL    ronda
                BR      joga_sempre


fim:            BR      fim
```