|  |  |  |  |
| --- | --- | --- | --- |
| Nome: | Enzo Bozzani Martins | R.A.: | 24.122.020-1 |
| Nome: | Luca Anequini Antoniazzi | R.A.: | 24.122.032-6 |

Projeto de Arquitetura de Computadores

# Descrição do Projeto

**Tema**: Jogo de memorização de uma sequência de números.

**Descrição**: Criação de um jogo de memorização utilizando uma sequência de números. Ao dar “*Run*” , a sequência é mostrada no display hexadecimal. O usuário, após exibição da sequência, deve acertá-la, digitando seus números no teclado matricial. Caso erre algum número da sequência, é mostrada a mensagem no LCD indicando que o usuário perdeu. Após isso, ele deve dar “*Run*” novamente para tentar acertar a sequência.

**Atividades**:

**10/10**:

- Adição das funções já existentes para uso do LCD e do teclado.

- Desenvolvimento das sub-rotinas para printar os números no display, printar a sequência no display (usando as sub-rotinas de printar números), printar no LCD o título do jogo e iniciado o desenvolvimento da sub-rotina que compara a leitura do teclado com a sequência.

**17/10**:

- Desenvolvimento da funcionalidade de comparar os dados lidos do teclado com a sequência, onde caso o usuário erre algum número da sequência ele perde e, caso acerte todos, ganha o jogo.

- Adicionada funcionalidade de informar ao usuário que ele venceu.

- Consertado o problema de que tanto o LCD quanto o display hexadecimal estarem usando a mesma porta.

**24/10:**

- Adicionado o segundo nível do jogo da memória

- Adicionada funcionalidade dos números digitados aparecerem na memória

- Correção de problemas relacionados à implementação do nível 2 e também à sub-rotinas estarem fora do alcance do pulo do comando CJNE

- Finalizado completamente o desenvolvimento do projeto

# Desenhos esquemáticos

Diagrama, Esquemático

Descrição gerada automaticamente

# Fluxograma ou Diagrama

Diagrama

Descrição gerada automaticamente

Fluxograma da lógica do projeto

Uma imagem contendo Calendário

Descrição gerada automaticamente

No endereço de memória 50H, é armazenado o valor da tecla que o usuário pressiona.

# Imagens da simulação realizada na IDE

Interface gráfica do usuário

Descrição gerada automaticamente

Número da sequência do nível 1 sendo exibido no display hexadecimal

Interface gráfica do usuário

Descrição gerada automaticamente

Número da sequência do nível 2 sendo exibido no display hexadecimal

Interface gráfica do usuário

Descrição gerada automaticamente

Impressão da mensagem de vitória no LCD

Interface gráfica do usuário

Descrição gerada automaticamente

Impressão da mensagem de derrota no LCD

# Discussões e conclusões

O projeto foi dividido em várias etapas de desenvolvimento, incluindo a adição das funções existentes, a implementação das sub-rotinas para exibir números, o desenvolvimento da lógica de comparação e a expansão do jogo com um segundo nível.

Dentre as principais dificuldades, é possível citar a configuração correta do edsim, por exemplo, indicando quais pinos deviam ser usados para que não houvesse conflito. Outra dificuldade foi da implementação da lógica de comparação entre os valores informados pelo o usuário com os números da sequência.

Durante o desenvolvimento, pudemos aprender mais sobre o funcionamento de hardware, assim como obter conhecimentos acerca da linguagem assembly.

# Código-fonte

**Repositório GitHub contendo o código-fonte completo:** [**https://github.com/EnzoBozzani/memory-game**](https://github.com/EnzoBozzani/memory-game)

**Código com as partes mais importantes comentadas:**

RS equ P1.3 ;Reg Select ligado em P1.3

EN equ P1.2 ;Enable ligado em P1.2

org 0000h

LJMP START

org 0030h

;atribuídos os valores para uso do teclado matricial

START:

MOV 50H, #0

MOV 40H, #'#'

MOV 41H, #'0'

MOV 42H, #'\*'

MOV 43H, #'9'

MOV 44H, #'8'

MOV 45H, #'7'

MOV 46H, #'6'

MOV 47H, #'5'

MOV 48H, #'4'

MOV 49H, #'3'

MOV 4AH, #'2'

MOV 4BH, #'1'

ACALL MAIN

;sub rotina que imprime no LCD que o usuário perdeu

printDerrota0:

MOV A, #44

ACALL posicionaCursor

MOV A, #' '

ACALL sendCharacter

MOV A, #'P'

ACALL sendCharacter

MOV A, #'E'

ACALL sendCharacter

MOV A, #'R'

ACALL sendCharacter

MOV A, #'D'

ACALL sendCharacter

MOV A, #'E'

ACALL sendCharacter

MOV A, #'U'

ACALL sendCharacter

SJMP $

;ponte para que funcione o pulo do CJNE

ponte0:

ACALL printDerrota0

MAIN:

;início do LCD e impressão da sequência no display hexadecimal

ACALL lcd\_init

ACALL printTitulo

ACALL printSeq

;rotinas que leem o teclado e comparam com os valores da sequência

;para cada rotina, é lido o valor, comparado com a sequência.

;Se for diferente, pula para o printDerrota

;Se for igual, exibe no endereço 50H o valor e continua a ler o teclado

ROTINA:

ACALL leituraTeclado

JNB F0, ROTINA

MOV A, #40h

ADD A, R0

MOV R0, A

MOV A, @R0

CJNE A, #'8', printDerrota0

SUBB A, #30h

MOV 50H, A

CLR F0

ACALL delay

ROT2:

ACALL leituraTeclado

JNB F0, ROT2

MOV A, #40h

ADD A, R0

MOV R0, A

MOV A, @R0

CJNE A, #'3', printDerrota0

SUBB A, #30h

MOV 50H, A

CLR F0

ACALL delay

ROT3:

ACALL leituraTeclado

JNB F0, ROT3

MOV A, #40h

ADD A, R0

MOV R0, A

MOV A, @R0

CJNE A, #'5', ponte0

SUBB A, #30h

MOV 50H, A

CLR F0

ACALL delay

ROT4:

ACALL leituraTeclado

JNB F0, ROT4

MOV A, #40h

ADD A, R0

MOV R0, A

MOV A, @R0

CJNE A, #'9', printDerrota

SUBB A, #30h

MOV 50H, A

CLR F0

ACALL delay

ROT5:

ACALL leituraTeclado

JNB F0, ROT5

MOV A, #40h

ADD A, R0

MOV R0, A

MOV A, @R0

CJNE A, #'1', printDerrota

SUBB A, #30h

MOV 50H, A

CLR F0

ACALL delay

;caso o usuário não perca em nenhum momento, é

;exibido no LCD que o nivel 2 irá iniciar

MOV A, #44

ACALL posicionaCursor

MOV A, #'N'

ACALL sendCharacter

MOV A, #'I'

ACALL sendCharacter

MOV A, #'V'

ACALL sendCharacter

MOV A, #'E'

ACALL sendCharacter

MOV A, #'L'

ACALL sendCharacter

MOV A, #' '

ACALL sendCharacter

MOV A, #'2'

ACALL sendCharacter

ACALL nivel2

JMP $

;sub rotina que imprime no LCD que o usuário perdeu

printDerrota:

MOV A, #44

ACALL posicionaCursor

MOV A, #' '

ACALL sendCharacter

MOV A, #'P'

ACALL sendCharacter

MOV A, #'E'

ACALL sendCharacter

MOV A, #'R'

ACALL sendCharacter

MOV A, #'D'

ACALL sendCharacter

MOV A, #'E'

ACALL sendCharacter

MOV A, #'U'

ACALL sendCharacter

SJMP $

;ponte para que funcione o pulo do CJNE

ponte:

ACALL printDerrota

;nível 2 seguindo a mesma lógica que o nível 1,

;porém com 10 números ao invés de 5

nivel2:

ACALL printSeq2

loop:

ACALL leituraTeclado

JNB F0, loop

MOV A, #40h

ADD A, R0

MOV R0, A

MOV A, @R0

CJNE A, #'3', printDerrota

SUBB A, #30h

MOV 50H, A

CLR F0

ACALL delay

ROT6:

ACALL leituraTeclado

JNB F0, ROT6

MOV A, #40h

ADD A, R0

MOV R0, A

MOV A, @R0

CJNE A, #'7', printDerrota

SUBB A, #30h

MOV 50H, A

CLR F0

ACALL delay

ROT7:

ACALL leituraTeclado

JNB F0, ROT7

MOV A, #40h

ADD A, R0

MOV R0, A

MOV A, @R0

CJNE A, #'9', printDerrota

SUBB A, #30h

MOV 50H, A

CLR F0

ACALL delay

ROT8:

ACALL leituraTeclado

JNB F0, ROT8

MOV A, #40h

ADD A, R0

MOV R0, A

MOV A, @R0

CJNE A, #'2', ponte

SUBB A, #30h

MOV 50H, A

CLR F0

ACALL delay

ROT9:

ACALL leituraTeclado

JNB F0, ROT9

MOV A, #40h

ADD A, R0

MOV R0, A

MOV A, @R0

CJNE A, #'5', ponte

SUBB A, #30h

MOV 50H, A

CLR F0

ACALL delay

ROT10:

ACALL leituraTeclado

JNB F0, ROT10

MOV A, #40h

ADD A, R0

MOV R0, A

MOV A, @R0

CJNE A, #'4', printDerrota2

SUBB A, #30h

MOV 50H, A

CLR F0

ACALL delay

ROT11:

ACALL leituraTeclado

JNB F0, ROT11

MOV A, #40h

ADD A, R0

MOV R0, A

MOV A, @R0

CJNE A, #'1', printDerrota2

SUBB A, #30h

MOV 50H, A

CLR F0

ACALL delay

ROT12:

ACALL leituraTeclado

JNB F0, ROT12

MOV A, #40h

ADD A, R0

MOV R0, A

MOV A, @R0

CJNE A, #'0', printDerrota2

SUBB A, #30h

MOV 50H, A

CLR F0

ACALL delay

ROT13:

ACALL leituraTeclado

JNB F0, ROT13

MOV A, #40h

ADD A, R0

MOV R0, A

MOV A, @R0

CJNE A, #'6', printDerrota2

SUBB A, #30h

MOV 50H, A

CLR F0

ACALL delay

ROT14:

ACALL leituraTeclado

JNB F0, ROT14

MOV A, #40h

ADD A, R0

MOV R0, A

MOV A, @R0

CJNE A, #'8', printDerrota2

SUBB A, #30h

MOV 50H, A

CLR F0

ACALL delay

;informado ao usuário que ele venceu

MOV A, #44

ACALL posicionaCursor

MOV A, #' '

ACALL sendCharacter

MOV A, #'V'

ACALL sendCharacter

MOV A, #'E'

ACALL sendCharacter

MOV A, #'N'

ACALL sendCharacter

MOV A, #'C'

ACALL sendCharacter

MOV A, #'E'

ACALL sendCharacter

MOV A, #'U'

ACALL sendCharacter

RET

;sub rotina que imprime no LCD que o usuário perdeu

printDerrota2:

MOV A, #44

ACALL posicionaCursor

MOV A, #' '

ACALL sendCharacter

MOV A, #'P'

ACALL sendCharacter

MOV A, #'E'

ACALL sendCharacter

MOV A, #'R'

ACALL sendCharacter

MOV A, #'D'

ACALL sendCharacter

MOV A, #'E'

ACALL sendCharacter

MOV A, #'U'

ACALL sendCharacter

SJMP $

;sub rotina que imprime no LCD o título

printTitulo:

MOV A, #1

ACALL posicionaCursor

MOV A, #'J'

ACALL sendCharacter

MOV A, #'O'

ACALL sendCharacter

MOV A, #'G'

ACALL sendCharacter

MOV A, #'O'

ACALL sendCharacter

MOV A, #' '

ACALL sendCharacter

MOV A, #'D'

ACALL sendCharacter

MOV A, #'A'

ACALL sendCharacter

MOV A, #' '

ACALL sendCharacter

MOV A, #'M'

ACALL sendCharacter

MOV A, #'E'

ACALL sendCharacter

MOV A, #'M'

ACALL sendCharacter

MOV A, #'O'

ACALL sendCharacter

MOV A, #'R'

ACALL sendCharacter

MOV A, #'I'

ACALL sendCharacter

MOV A, #'A'

ACALL sendCharacter

RET

;sub rotina que printa a sequência 1 no display

; (usando sub rotinas que imprimem cada número no

; display, setando ou não os bits de P2)

printSeq:

ACALL PRINT8

ACALL delay

ACALL PRINT3

ACALL delay

ACALL PRINT5

ACALL delay

ACALL PRINT9

ACALL delay

ACALL PRINT1

ACALL delay

MOV P2, #0FFH

RET

;sub rotina que printa a sequência 2 no display

; (usando sub rotinas que imprimem cada número no

; display, setando ou não os bits de P2)

printSeq2:

ACALL PRINT3

ACALL delay

ACALL PRINT7

ACALL delay

ACALL PRINT9

ACALL delay

ACALL PRINT2

ACALL delay

ACALL PRINT5

ACALL delay

ACALL PRINT4

ACALL delay

ACALL PRINT1

ACALL delay

ACALL PRINT0

ACALL delay

ACALL PRINT6

ACALL delay

ACALL PRINT8

ACALL delay

MOV P2, #0FFH

RET

leituraTeclado:

;...

colScan:

;...

gotKey:

;...

sendCharacter:

;...

;Limpa o display

clearDisplay:

;…

delay:

;...

PRINT0:

MOV P2, #0

SETB P2.6

SETB P2.7

RET

PRINT1:

SETB P2.0

CLR P2.1

SETB P2.3

CLR P2.2

SETB P2.4

SETB P2.5

SETB P2.6

SETB P2.7

RET

PRINT2:

CLR P2.0

CLR P2.1

SETB P2.2

CLR P2.3

CLR P2.4

SETB P2.5

CLR P2.6

SETB P2.7

RET

PRINT3:

CLR P2.0

CLR P2.1

CLR P2.2

CLR P2.3

SETB P2.4

SETB P2.5

CLR P2.6

SETB P2.7

RET

PRINT4:

SETB P2.0

CLR P2.1

CLR P2.2

SETB P2.3

SETB P2.4

CLR P2.5

CLR P2.6

SETB P2.7

RET

PRINT5:

CLR P2.0

SETB P2.1

CLR P2.2

CLR P2.3

SETB P2.4

CLR P2.5

CLR P2.6

SETB P2.7

RET

PRINT6:

CLR P2.0

SETB P2.1

CLR P2.2

CLR P2.3

CLR P2.4

CLR P2.5

CLR P2.6

SETB P2.7

RET

PRINT7:

CLR P2.0

CLR P2.1

CLR P2.2

SETB P2.3

SETB P2.4

SETB P2.5

SETB P2.6

SETB P2.7

RET

PRINT8:

MOV P2, #0

SETB P2.7

RET

PRINT9:

MOV P2, #0

SETB P2.4

SETB P2.7

RET