|  |  |  |  |
| --- | --- | --- | --- |
| Nome: | Kaique da Silva Fernandes | R.A.: | 222210114 |
| Nome: | Matheus Miranda Vieira | R.A.: | 222200172 |

Projeto de Arquitetura de Computadores

# Descrição do Projeto

O código em linguagem Assembly apresentado é destinado à implementação de um sistema de segurança de porta eletrônica. O programa é projetado para ser executado em um microcontrolador 8051 e utiliza um display LCD para interação com o usuário. Ele insere uma senha, que é comparada com uma senha pré-definida e é fornecido o feedback visualmente no display LCD.

# Desenhos esquemáticos

Diagrama, Esquemático

Descrição gerada automaticamente.

# Fluxograma ou Diagrama

1. Mapeamento de Hardware (8051):

O código define duas constantes, RS e EN, que representam os pinos do microcontrolador 8051 conectados ao Registrador de Seleção (RS) e ao Pino Enable (EN) do display LCD, respectivamente.

2. Inicialização de Valores:

A seção "org 0000h" é o ponto de partida do programa e inicializa valores iniciais, como mapeamento de teclas, senha predefinida e variáveis.

3. Endereçamento de Memória:

O código utiliza rótulos simbólicos (e.g., COUNT, SELECT, KEYBOARD\_VALUE, WRITE\_POSITION) para endereçar as variáveis armazenadas na memória do microcontrolador.

4. Função Principal (MAIN):

A função principal inicia com a chamada da função lcd\_init, que configura o display LCD.

5.Funções:

O código inclui várias funções para lidar com diferentes partes do sistema de segurança.

-TYPE\_PASSWORD: Essa função permite que o usuário insira a senha. Ela usa o display para orientar o usuário na entrada da senha e compara a senha inserida com a senha predefinida.

-COMPARE\_PASSWORD: Essa função compara a senha inserida com a senha predefinida, dígito por dígito. Se a senha estiver correta, exibe "Senha Correta" no display; caso contrário, exibe "Senha Incorreta."

-SAVE\_PASSWORD: Armazena o valor do teclado atual na posição correspondente da senha inserida.

-PRINT\_PASSWORD: Exibe a mensagem "Senha" no display para orientar o usuário a inserir a senha.

-PRINT\_MENU: Exibe a mensagem "Digite sua senha" no display como parte do menu principal.

-WAIT\_KEYBOARD: Aguarda a entrada do teclado e armazena o valor pressionado na variável KEYBOARD\_VALUE.

-IF\_KEYBOARD\_VALUE\_RIGHT e clearDisplay: Funções auxiliares para manipulação do display.

-leituraTeclado, colScan, e sendCharacter: Funções relacionadas à leitura do teclado matricial e envio dos caracteres para o display.

-posicionaCursor e retornaCursor: Funções para posicionar o cursor no display.

-delay: Função para introduzir atrasos controlados, necessários para garantir a estabilidade da comunicação com o display.

-clearDisplay: Limpa o display.

6.Strings:

O código define várias strings em memória para exibição no display, como mensagens de menu, senha correta, senha incorreta, entre outras.

7.Espera pelo Teclado:

O código aguarda a entrada do teclado e armazena o valor pressionado na variável KEYBOARD\_VALUE, enquanto exibe "Senha" no display para orientar o usuário.

8.Comparação de Senha:

A senha inserida é comparada com a senha predefinida dígito por dígito. Se as senhas coincidirem, a mensagem "Senha Correta" é exibida; caso contrário, é exibida a mensagem "Senha Incorreta."

9.Funções Auxiliares do Display:

O código possui funções que permitem o controle do display, como posicionar o cursor, retornar o cursor para a primeira posição, limpar o display e exibir caracteres no display.

10. Atrasos (Delay):A função delay é utilizada para introduzir atrasos controlados, necessários para garantir a estabilidade da comunicação com o display.Texto

Descrição gerada automaticamente com confiança baixa

# Imagens da simulação realizada na IDE

Menu:

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

Em caso de senha correta:

Interface gráfica do usuário

Descrição gerada automaticamente com confiança baixa

Em caso de senha incorreta:

Calendário

Descrição gerada automaticamente com confiança média

# Discussões e conclusões

1. Dificuldades

A própria linguagem Assembly foi a nossa maior dificuldade por conta de toda sua limitação. A lógica em si, não foi algo dificil por si só, porém outro empecilho foi o tempo que tivemos para realizar o projeto, por conta de projetos e provas de outras matérias, assim como o trabalho de ambos, sendo o nosso tempo disponível para a realização do código durante os feriados e finais de semana.

1. Aprendizado

Nós aprendemos uma nova linguagem, e aprender melhor o pensamento do passo a passo, por conta da linguagem Assembly. O que ocasionou na melhora do nosso raciocínio lógico

1. Diferença

Assembly é uma linguagem de baixo nível, o que têm seus prós e contras, o que ocasiona de demandar muito tempo para se fazer algo simples. Além que é necessário o entendimento do próprio software utilizado Edsin51, o que também demandou um tempo considerável para o entendimento do Hardware que iríamos utilizar em nosso projeto

# Código-fonte

; --- Mapeamento de Hardware (8051) ---

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

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

;\_\_\_\_\_\_\_\_\_\_\_\_\_Comeco \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

;Incializa��o de valores

org 0000h

;Mapeamento das teclas

MOV 50H, #'#'

MOV 51H, #'0'

MOV 52H, #'\*'

MOV 53H, #'9'

MOV 54H, #'8'

MOV 55H, #'7'

MOV 56H, #'6'

MOV 57H, #'5'

MOV 58H, #'4'

MOV 59H, #'3'

MOV 5AH, #'2'

MOV 5BH, #'1'

;Senha predefinida

MOV 5CH, #'9'

MOV 5DH, #'9'

MOV 5EH, #'9'

MOV 5FH, #'9'

;RESET nas vari�veis

MOV KEYBOARD\_VALUE, #0

MOV WRITE\_POSITION, #66

MOV SELECT, #0

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ADDRESS\_IN\_MEMORIE\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

;Guarda variaveis

COUNT equ 73h

SELECT equ 72h

KEYBOARD\_VALUE equ 70h

WRITE\_POSITION equ 71h

LJMP START

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

org 0030h

START:

ACALL lcd\_init

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_MAIN\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

MAIN:

MAIN\_LOOP:

CALL PRINT\_MENU

CALL TYPE\_PASSWORD

SJMP MAIN\_LOOP

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

;\_\_\_\_\_\_\_\_\_\_\_\_\_FUN��ES\_\_\_\_\_\_\_\_\_\_\_\_

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_TYPE\_PASSWORD\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

TYPE\_PASSWORD:

MOV COUNT, #4

CALL PRINT\_PASSWORD

INICIALIZATE\_PASSWORD:

CALL WAIT\_KEYBOARD

MOV A, #'\*'

ACALL SAVE\_PASSWORD

DJNZ COUNT, INICIALIZATE\_PASSWORD

END\_PASSWORD:

CALL COMPARE\_PASSWORD

RET

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_COMPARE\_PASSWORD\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

COMPARE\_PASSWORD:

MOV COUNT, #4

MOV R6, #0

MOV DPTR, #STR\_PASSWORD

COMPARE\_LOOP:

MOV R2, #64H

MOV R3, #60H

MOV A, R2

SUBB A, COUNT

MOV R0, A

MOV A, R3

SUBB A, COUNT

MOV R1, A

MOV A, @R0

MOV B, @R1

SUBB A, B

INC DPTR

INC R6

JNZ COMPARED\_WRONG

DJNZ COUNT, COMPARE\_LOOP

JMP COMPARED\_RIGHT

COMPARED\_WRONG:

CALL clearDisplay

MOV A, #0

ACALL posicionaCursor

MOV DPTR, #STR\_WRONG\_PASSWORD

ACALL escreveStringROM

ACALL delay

CALL clearDisplay

RET

COMPARED\_RIGHT:

CALL clearDisplay

MOV A, #0

ACALL posicionaCursor

MOV DPTR, #STR\_RIGHT\_PASSWORD

ACALL escreveStringROM

ACALL delay

CALL clearDisplay

RET

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_SAVE\_PASSWORD\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SAVE\_PASSWORD:

MOV A, #64H

SUBB A, COUNT

MOV R0, A

MOV @R0, KEYBOARD\_VALUE

RET

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_PRINT\_PASSWORD\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

PRINT\_PASSWORD:

MOV A, #64

ACALL posicionaCursor

MOV DPTR,#STR\_PW

ACALL escreveStringROM

MOV A, #58

ACALL sendCharacter

RET

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_PRINT\_MENU\_\_\_\_\_\_\_\_\_\_\_\_\_\_

PRINT\_MENU:

MOV A, #0

ACALL posicionaCursor

MOV DPTR,#STR\_MENU

ACALL escreveStringROM

RET

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_PRINT\_KEYBOARD\_VALUE\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

PRINT\_KEYBOARD\_VALUE:

MOV A, WRITE\_POSITION

ACALL posicionaCursor

MOV A, KEYBOARD\_VALUE

ACALL sendCharacter

;Incrementa o write\_position

MOV A, WRITE\_POSITION

INC A

MOV WRITE\_POSITION, A

RET

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_RESET\_WRITE\_POSITION\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

RESET\_WRITE\_POSITION:

MOV WRITE\_POSITION, #66

RET

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_IF\_KEYBOARD\_VALUE\_RIGHT\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

IF\_KEYBOARD\_VALUE\_RIGHT:

;it backs to WAIT\_KEYBOARD to check the next value

MOV A, KEYBOARD\_VALUE

SUBB A, #30H

JZ CONFIRM

RET

CONFIRM:

CALL clearDisplay

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_WAIT\_KEYBOARD\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

;Aguarda valor do teclado and armazena ele na variavel KEYBOARD\_VALUE Wait value in keyboard and store him in VALUE\_KEYBOARD in ASCII

WAIT\_KEYBOARD:

;L� o teclado at� que uma tecla seja pressionada

ACALL leituraTeclado

JNB F0, WAIT\_KEYBOARD

;Captura o valor, compara com a label e armazena a senha

MOV A, #50h;Begin with data

ADD A, R0

MOV R0, A

MOV A, @R0

MOV KEYBOARD\_VALUE, A

CLR F0

CALL delay

;Verifica Input

JMP IF\_KEYBOARD\_VALUE\_RIGHT

;\_\_\_\_\_\_\_\_\_\_\_\_\_STRINGS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

STR\_MENU:

DB "Digite sua senha "

DB 0

STR\_PASSWORD:

DB "9999"

DB 0

STR\_PW:

DB "Senha"

DB 0

STR\_WRONG\_PASSWORD:

DB "Senha Incorreta"

DB 0

STR\_RIGHT\_PASSWORD:

DB "Senha Correta"

DB 0

;Below are all functions given in class

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_escreveStringROM\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

escreveStringROM:

MOV R1, #00h

; Inicia a escrita da String no Display LCD

loop:

MOV A, R1

MOVC A,@A+DPTR ;l� da mem�ria de programa

JZ escreveStringROM\_finish ; if A is 0, then end of data has been reached - jump out of loop

ACALL sendCharacter ; send data in A to LCD module

INC R1 ; point to next piece of data

MOV A, R1

JMP loop ; repeat

escreveStringROM\_finish:

RET

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_leituraTeclado\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

leituraTeclado:

MOV R0, #0 ; clear R0 - the first key is key0

; scan row0

MOV P0, #0FFh

CLR P0.0 ; clear row0

CALL colScan ; call column-scan subroutine

JB F0, finish ; | if F0 is set, jump to end of program

; | (because the pressed key was found and its number is in R0)

; scan row1

SETB P0.0 ; set row0

CLR P0.1 ; clear row1

CALL colScan ; call column-scan subroutine

JB F0, finish ; | if F0 is set, jump to end of program

; | (because the pressed key was found and its number is in R0)

; scan row2

SETB P0.1 ; set row1

CLR P0.2 ; clear row2

CALL colScan ; call column-scan subroutine

JB F0, finish ; | if F0 is set, jump to end of program

; | (because the pressed key was found and its number is in R0)

; scan row3

SETB P0.2 ; set row2

CLR P0.3 ; clear row3

CALL colScan ; call column-scan subroutine

JB F0, finish ; | if F0 is set, jump to end of program

; | (because the pressed key was found and its number is in R0)

finish:

RET

; column-scan subroutine

colScan:

JNB P0.4, gotKey ; if col0 is cleared - key found

INC R0 ; otherwise move to next key

JNB P0.5, gotKey ; if col1 is cleared - key found

INC R0 ; otherwise move to next key

JNB P0.6, gotKey ; if col2 is cleared - key found

INC R0 ; otherwise move to next key

RET ; return from subroutine - key not found

gotKey:

SETB F0 ; key found - set F0

RET ; and return from subroutine

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_lcd\_init\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

; initialise the display

; see instruction set for details

lcd\_init:

CLR RS ; clear RS - indicates that instructions are being sent to the module

; function set

CLR P1.7 ; |

CLR P1.6 ; |

SETB P1.5 ; |

CLR P1.4 ; | high nibble set

SETB EN ; |

CLR EN ; | negative edge on E

CALL delay ; wait for BF to clear

; function set sent for first time - tells module to go into 4-bit mode

; Why is function set high nibble sent twice? See 4-bit operation on pages 39 and 42 of HD44780.pdf.

SETB EN ; |

CLR EN ; | negative edge on E

; same function set high nibble sent a second time

SETB P1.7 ; low nibble set (only P1.7 needed to be changed)

SETB EN ; |

CLR EN ; | negative edge on E

; function set low nibble sent

CALL delay ; wait for BF to clear

; entry mode set

; set to increment with no shift

CLR P1.7 ; |

CLR P1.6 ; |

CLR P1.5 ; |

CLR P1.4 ; | high nibble set

SETB EN ; |

CLR EN ; | negative edge on E

SETB P1.6 ; |

SETB P1.5 ; |low nibble set

SETB EN ; |

CLR EN ; | negative edge on E

CALL delay ; wait for BF to clear

; display on/off control

; the display is turned on, the cursor is turned on and blinking is turned on

CLR P1.7 ; |

CLR P1.6 ; |

CLR P1.5 ; |

CLR P1.4 ; | high nibble set

SETB EN ; |

CLR EN ; | negative edge on E

SETB P1.7 ; |

SETB P1.6 ; |

SETB P1.5 ; |

SETB P1.4 ; | low nibble set

SETB EN ; |

CLR EN ; | negative edge on E

CALL delay ; wait for BF to clear

RET

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_sendCharacter\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

sendCharacter:

SETB RS ; setb RS - indicates that data is being sent to module

MOV C, ACC.7 ; |

MOV P1.7, C ; |

MOV C, ACC.6 ; |

MOV P1.6, C ; |

MOV C, ACC.5 ; |

MOV P1.5, C ; |

MOV C, ACC.4 ; |

MOV P1.4, C ; | high nibble set

SETB EN ; |

CLR EN ; | negative edge on E

MOV C, ACC.3 ; |

MOV P1.7, C ; |

MOV C, ACC.2 ; |

MOV P1.6, C ; |

MOV C, ACC.1 ; |

MOV P1.5, C ; |

MOV C, ACC.0 ; |

MOV P1.4, C ; | low nibble set

SETB EN ; |

CLR EN ; | negative edge on E

CALL delay ; wait for BF to clear

CALL delay ; wait for BF to clear

RET

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_posicionaCursor\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

;Posiciona o cursor na linha e coluna desejada.

;Escreva no Acumulador o valor de endere�o da linha e coluna.

;|--------------------------------------------------------------------------------------|

;|linha 1 | 00 | 01 | 02 | 03 | 04 |05 | 06 | 07 | 08 | 09 |0A | 0B | 0C | 0D | 0E | 0F |

;|linha 2 | 40 | 41 | 42 | 43 | 44 |45 | 46 | 47 | 48 | 49 |4A | 4B | 4C | 4D | 4E | 4F |

;|--------------------------------------------------------------------------------------|

posicionaCursor:

CLR RS

SETB P1.7 ; |

MOV C, ACC.6 ; |

MOV P1.6, C ; |

MOV C, ACC.5 ; |

MOV P1.5, C ; |

MOV C, ACC.4 ; |

MOV P1.4, C ; | high nibble set

SETB EN ; |

CLR EN ; | negative edge on E

MOV C, ACC.3 ; |

MOV P1.7, C ; |

MOV C, ACC.2 ; |

MOV P1.6, C ; |

MOV C, ACC.1 ; |

MOV P1.5, C ; |

MOV C, ACC.0 ; |

MOV P1.4, C ; | low nibble set

SETB EN ; |

CLR EN ; | negative edge on E

CALL delay ; wait for BF to clear

CALL delay ; wait for BF to clear

RET

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_retornaCursor\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

;Retorna o cursor para primeira posi��o sem limpar o display

retornaCursor:

CLR RS

CLR P1.7 ; |

CLR P1.6 ; |

CLR P1.5 ; |

CLR P1.4 ; | high nibble set

SETB EN ; |

CLR EN ; | negative edge on E

CLR P1.7 ; |

CLR P1.6 ; |

SETB P1.5 ; |

SETB P1.4 ; | low nibble set

SETB EN ; |

CLR EN ; | negative edge on E

CALL delay ; wait for BF to clear

RET

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_clearDisplay\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

clearDisplay:

CLR RS

CLR P1.7 ; |

CLR P1.6 ; |

CLR P1.5 ; |

CLR P1.4 ; | high nibble set

SETB EN ; |

CLR EN ; | negative edge on E

CLR P1.7 ; |

CLR P1.6 ; |

CLR P1.5 ; |

SETB P1.4 ; | low nibble set

SETB EN ; |

CLR EN ; | negative edge on E

MOV R6, #40

rotC:

CALL delay ; wait for BF to clear

DJNZ R6, rotC

RET

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_delay\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

delay:

MOV R0, #50

DJNZ R0, $

RET