|  |  |  |  |
| --- | --- | --- | --- |
| Nome: | João Vitor Dias | R.A.: | 22.119.006-9 |
| Nome: | Thiago Andrade | R.A.: | 22.119.080-4 |

Projeto de Arquitetura de Computadores

# Descrição do Projeto

Link para o GitHub: <https://github.com/JoaoDias-223/Simple-Assembly-Calculator>

Link para o YouTube: <https://youtu.be/uWuBCXIWHsk>

O projeto consiste numa calculadora capaz de realizar as 4 operações básicas: adição, subtração, produto e divisão. Além disso, ela também será capaz de realizar operações como potenciação, radiciação e logaritmo na base 10. Ademais, o usuário utilizará o keyboard do simulador Edsim51 para escolher os números e visualizará no display LCD o resultado das operações.

Abaixo está um *checklist* das funcionalidades que serão implementadas:

Operação de Adição

Operação de Subtração

Operação de Produto

Operação de Divisão

Operação de Potência

Operação de Radiciação

Operação de Logaritmo na Base 10

Conexão entre o keyboard e os registradores

Conexão entre o resultado e o display LCD

OBS: O ‘x’ nas caixas de seleção, significa que a funcionalidade em questão já foi implementada.

**Nota 09/11:** Conseguimos pegar o input do keyboard com sucesso e salvá-los na memória para fazermos os cálculos. Além disso conseguimos mostrar o resultado das operações de adição e subtração na tela, porém, infelizmente, não conseguimos fazer as operações de produto, divisão, potência, radiciação e logaritmo.

# Desenhos esquemáticos

Diagrama, Esquemático

Descrição gerada automaticamente

Figura 1: Diagrama esquemático do circuito.

# Fluxograma ou Diagrama

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

Figura 2: O retângulo vermelho é a representação dos caracteres “0, 1, 2, 3, 4, 5, 6, 7, 8, 9, C, =” em ASCII, da esquerda para a direita.

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

Figura 3: Os retângulos vermelhos contêm os endereços responsáveis por guardar os cursores dos dois números no display LCD e guardar o número de algarismos dos números.

- Limpar memória RAM

- Inicializar caracteres na memória RAM

- Inicializar variáveis da calculadora na memória RAM

- Guardar cada dígito do segundo número na memória RAM.

- Realizar a operação

- Escanear o teclado para pegar a tecla pressionada.

- Guardar cada dígito do primeiro número na memória RAM.

# Código-fonte

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

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

org 0000h

MOV R0, #7Fh

JMP CLEAR\_RAM

org 0030h

START:

CALL SET\_OPERATIONS

CALL INITIALIZE\_CHARACTERS

CALL INITIALIZE\_POINTERS

JMP MAIN

MAIN:

CALL lcd\_init

JMP ROTINA

ADDITION:

SETB P2.7

CLR P3.1

MOV 5AH, #01H

RET

SUBTRACTION:

SETB P2.7

CLR P3.2

MOV 5AH, #02H

RET

PRODUCT:

SETB P2.7

CLR P3.3

MOV 5AH, #03H

RET

DIVISION:

SETB P2.7

CLR P3.4

MOV 5AH, #04H

RET

FIND\_OP:

JNB P2.0, ADDITION

JB P2.7, finish

JNB P2.1, SUBTRACTION

JB P2.7, finish

JNB P2.2, PRODUCT

JB P2.7, finish

JNB P2.3, DIVISION

JB P2.7, finish

RET

ROTINA:

CLR C

; CJNE P2, #0FH, FIND\_OP

CALL FIND\_OP

CALL WHICH\_NUMBER

CALL leituraTeclado

JNB F0, ROTINA ;if F0 is clear, jump to ROTINA

CALL CHOOSE\_LINE

CALL posicionaCursor

MOV A, #40h

ADD A, R0

MOV R0, A

MOV A, @R0

CALL sendCharacter

CLR F0

JMP ROTINA

WHICH\_NUMBER:

MOV C, P3.0

ORL C, P2.7

MOV P3.0, C

CLR P2.7

RET

CHOOSE\_LINE:

JNB P3.0, FIRST\_LINE

JB P3.0, SECOND\_LINE

RET

FIRST\_LINE:

MOV A, 51H

RET

SECOND\_LINE:

MOV A, 50H

RET

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

; scan row1

SETB P0.0 ; set row0

CLR P0.1 ; clear row1

CALL colScan ; call column-scan subroutine

JB F0, finish

; scan row2

SETB P0.1 ; set row1

CLR P0.2 ; clear row2

CALL colScan ; call column-scan subroutine

JB F0, finish

; scan row3

SETB P0.2 ; set row2

CLR P0.3 ; clear row3

CALL colScan ; call column-scan subroutine

JB F0, finish

finish:

RET

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

CJNE R0, #0AH, VALIDACAO\_DO\_CHARACTER

CALL clearDisplay

CLR F0

RET ; and return from subroutine

VALIDACAO\_DO\_CHARACTER:

JC CHARACTER\_IS\_NUM

JNC EQUAL\_OP

JMP ROTINA

CHARACTER\_IS\_NUM:

CLR C

JB P3.0, GUARDA\_NUM2

JNB P3.0, GUARDA\_NUM1

JMP ROTINA

EQUAL\_OP:

CLR F0

CLR P3.0

CALL DO\_OP

MOV A, R0

MOV R2, A

MOV A, R1

MOV R3, A

MOV R4, 58H

MOV 30H, 58H

CLR C

CALL MOV\_FIRST\_TO\_RESULT

CALL clearDisplay

CALL CALLS\_LONG\_DELAY

;CALL delay

CALL MOV\_RESULT\_TO\_FIRST

CLR C

MOV R0, #3FH

CALL SHOW\_RESULT

JMP ROTINA

SHOW\_RESULT:

INC 51H

INC 50H

CALL CHOOSE\_LINE

CALL posicionaCursor

CLR C

MOV A, #3FH

SUBB A, 30H

INC A

MOV R0, A

MOV A, @R0

ADD A, #30H

CALL sendCharacter

DJNZ 30H, SHOW\_RESULT

RET

GUARDA\_NUM1:

INC 51H

MOV A, R0

SETB RS0

SETB RS1

CALL SORT\_FIRST\_NUMBER\_START

INC 54H

MOV 58H, 54H

MOV @R0, A

DEC 56H

CLR RS0

CLR RS1

RET

GUARDA\_NUM2:

INC 50H

MOV A, R0

SETB RS0

SETB RS1

CALL SORT\_SECOND\_NUMBER\_START

INC 55H

MOV 59H, 55H

MOV @R1, A

DEC 57H

CLR RS0

CLR RS1

RET

MOV\_FIRST\_TO\_RESULT:

MOV A, 52H

SUBB A, 58H

MOV R0, A

SUBB A, #30H

MOV R1, A

MOV B, @R0

MOV @R1, B

DJNZ 58H, MOV\_FIRST\_TO\_RESULT

MOV A, 52H

SUBB A, 58H

MOV R0, A

SUBB A, #30H

MOV R1, A

MOV B, @R0

MOV @R1, B

RET

MOV\_RESULT\_TO\_FIRST:

MOV A, 52H

SUBB A, R4

MOV R0, A

SUBB A, #30H

MOV R1, A

MOV A, @R1

MOV R0, A

DJNZ R4,MOV\_RESULT\_TO\_FIRST

MOV A, 52H

SUBB A, R4

MOV R0, A

SUBB A, #30H

MOV R1, A

MOV A, @R1

MOV R0, A

RET

DO\_OP:

MOV A, R0

MOV R6, A

MOV A, R1

MOV R7, A

MOV R0, 18H

MOV R1, 19H

MOV R4, #00H

MOV A, 54H

MOV R5, 55H

CALL PRE\_NUMEROS\_DIFERENTES

CLR C

CALL OPS

MOV A, R6

MOV R0, A

MOV A, R7

MOV R1, A

RET

OPS:

JNB P3.1, ADDITION\_OP

JNB P3.2, SUBTRACTION\_OP

RET

PRE\_NUMEROS\_DIFERENTES:

CJNE A, 55H, NUMEROS\_DIFERENTES

NUMEROS\_DIFERENTES:

JNC PRIMEIRO\_MAIOR\_OU\_IGUAL

RET

PRIMEIRO\_MAIOR\_OU\_IGUAL:

MOV R5, 54H

SETB C

JMP NUMEROS\_DIFERENTES

ADDITION\_OP:

MOV A, @R0 ;first digit

ADD A, @R1 ;second digit

ADD A, R4 ;carry from previous op

MOV B, #0AH

DIV AB

MOV @R0, B

MOV R4, A

DEC R0

DEC R1

DJNZ R5, ADDITION\_OP

CJNE R4, #00H, ADD\_CARRY

RET

ADD\_CARRY:

MOV A, R4

MOV @R0, A

RET

SUBTRACTION\_OP:

MOV A, @R0 ;first digit

SUBB A, @R1 ;second digit

CALL PRE\_CHANGE\_ORDER

MOV @R0, A

DEC R0

DEC R1

DJNZ R5, SUBTRACTION\_OP

RET

PRE\_CHANGE\_ORDER:

JC CHANGE\_ORDER

RET

CHANGE\_ORDER:

MOV A, @R0

ADD A, #0AH

SUBB A, @R1

MOV B, A

MOV A, R0

MOV R7, A

DEC R0

CALL PRE\_SUB\_R0

CLR C

MOV A, R7

MOV R0, A

MOV A, B

RET

PRE\_SUB\_R0:

CJNE @R0, #00H, SUB\_R0

RET

SUB\_R0:

DEC @R0

RET

lcd\_init:

CLR RS

CLR P1.7

CLR P1.6

SETB P1.5

CLR P1.4

SETB EN

CLR EN

CALL delay

SETB EN

CLR EN

SETB P1.7

SETB EN

CLR EN

CALL delay

CLR P1.7

CLR P1.6

CLR P1.5

CLR P1.4

SETB EN

CLR EN

SETB P1.6

SETB P1.5

SETB EN

CLR EN

CALL delay

CLR P1.7

CLR P1.6

CLR P1.5

CLR P1.4

SETB EN

CLR EN

SETB P1.7

SETB P1.6

SETB P1.5

SETB P1.4

SETB EN

CLR EN

CALL delay

RET

sendCharacter:

SETB RS

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

SETB EN

CLR EN

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

SETB EN

CLR EN

CALL delay

CALL delay

RET

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

;|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

SETB EN

CLR EN

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

SETB EN

CLR EN

CALL delay

CALL delay

RET

clearDisplay:

MOV R0, #54H

CALL CLEAR\_MEMORY\_ROW

MOV 56h, #6FH

MOV 57h, #7FH

CALL SET\_P3

CLR P3.0

MOV 51H, 4EH

MOV 50H, 4FH

CLR RS

CLR P1.7

CLR P1.6

CLR P1.5

CLR P1.4

SETB EN

CLR EN

CLR P1.7

CLR P1.6

CLR P1.5

SETB P1.4

SETB EN

CLR EN

CALL delay

RET

CLEAR\_MEMORY\_ROW:

MOV @R0, #00H

INC R0

CJNE R0, #80H, CLEAR\_MEMORY\_ROW

RET

delay:

MOV R7, #50

DJNZ R7, $

RET

CALLS\_LONG\_DELAY:

MOV R5, #50h

CALL LONG\_DELAY

RET

LONG\_DELAY:

CALL delay

DJNZ R5, LONG\_DELAY

RET

CLEAR\_RAM:

MOV @R0, #00h

DJNZ R0, CLEAR\_RAM

JMP START

SET\_OPERATIONS:

SETB P2.0

SETB P2.1

SETB P2.2

SETB P2.3

CLR P2.4

CLR P2.5

CLR P2.6

CLR P2.7

RET

SET\_P3:

SETB P3.0

SETB P3.1

SETB P3.2

SETB P3.3

SETB P3.4

SETB P3.5

SETB P3.6

SETB P3.7

RET

INITIALIZE\_CHARACTERS:

; put data in RAM

MOV 40H, #'0'

MOV 41H, #'1'

MOV 42H, #'2'

MOV 43H, #'3'

MOV 44H, #'4'

MOV 45H, #'5'

MOV 46H, #'6'

MOV 47H, #'7'

MOV 48H, #'8'

MOV 49H, #'9'

MOV 4AH, #'C'

MOV 4BH, #'B'

RET

INITIALIZE\_POINTERS:

;LCD Cursor Position

MOV 4EH, #01h ;backup first line

MOV 4FH, #41h ;backup second line

MOV 50H, #41H ;position second line

MOV 51H, #01h ;position first line

MOV 52H, #6FH ;Backup position of the start of the first number

MOV 53H, #7FH ;Backup position of the start of the second number

MOV 54h, #00h ;Backup Number of digits of the first number

MOV 55h, #00h ;Backup Number of digits of the second number

MOV 56h, #6FH ;First Number Cursor

MOV 57h, #7FH ;Second Number Cursor

MOV 58H, 54H ;Number of Digits in First Number

MOV 59H, 55h ;Number of Digits in Second Number

SETB RS0

SETB RS1

MOV R0, 52H ;First number digits

MOV R1, 53H ;Second number digits

CLR RS0

CLR RS1

CLR P3.0 ;Checks whether it is the first

;or the second number being typed

RET

SORT\_FIRST\_NUMBER\_START:

MOV B, R0

MOV R4, B

MOV B, R1

MOV R5, B

MOV R6, A

MOV A, 58H

JZ FINISH2

CALL SORT\_FIRST\_NUMBER

MOV B, R4

MOV R0, B

MOV B, R5

MOV R1, B

MOV A, R6

RET

SORT\_FIRST\_NUMBER:

MOV A, 52H

SUBB A, 58H

MOV R0, A

INC A

MOV R1, A

MOV A, @R1

MOV @R0, A

DJNZ 58H, SORT\_FIRST\_NUMBER

RET

SORT\_SECOND\_NUMBER\_START:

MOV B, R0

MOV R4, B

MOV B, R1

MOV R5, B

MOV R6, A

MOV A, 59H

JZ FINISH2

CALL SORT\_SECOND\_NUMBER

MOV B, R4

MOV R0, B

MOV B, R5

MOV R1, B

MOV A, R6

RET

SORT\_SECOND\_NUMBER:

MOV A, 53H

SUBB A, 59H

MOV R0, A

INC A

MOV R1, A

MOV A, @R1

MOV @R0, A

DJNZ 59H, SORT\_SECOND\_NUMBER

RET

FINISH2:

MOV B, R4

MOV R0, B

MOV B, R5

MOV R1, B

MOV A, R6

RETImagens da simulação realizada na IDE

Interface gráfica do usuário

Descrição gerada automaticamente

Figura 4: Inicialização da calculadora.

Interface gráfica do usuário

Descrição gerada automaticamente

Figura 5: Inserção do número 143.

Interface gráfica do usuário

Descrição gerada automaticamente

Figura 6: Inserção do número 66.

Interface gráfica do usuário

Descrição gerada automaticamente

Figura 7: Operação de adição entre 143 e 66, resultando em 209.

Interface gráfica do usuário

Descrição gerada automaticamente

Figura 7: Operação de subtração entre 143 e 66, resultando em 77.