**PROJECT REPORT**

**Assembly Language and Computer Architecture Lab – IT3280e**

**Group 2**

Ngụy Quang Sơn – 20225998

Lò Đức tài – 20225999

1. **Problem 8: Pocket Calculator**
2. **Problem description**

Use a keypad and two 7-segment displays (Digital Lab Sim) to build a simple calculator supporting operations +, -, \*, /, and % on integers.

**Requirements:**

* Use keypad keys for operations:
* **a** for addition (+)
* **b** for subtraction (–)
* **c** for multiplication (\*)
* **d** for division (/)
* **e** for modulo (%)
* **f** for equals (=)
* When entering a number, display the last two digits of the number on the two 7-segment displays.

Example of entering 123: 1 → 01, 2 → 12, 3 → 23.

* After entering a number, select an operation (+, –, etc.) and press f to calculate.
* Allow consecutive calculations (like the Windows Calculator app)

1. **How to use**

* Open Digital Lab Sim (connect to program) and run the program
* Click the number in the numpad and the corresponding number will appear on the

Seven-Segment Display LED (LED on the right side)

* Choose an operator (see the problem description to see what key is corresponding to what character. Example: a is addition, b is subtraction,…)
* Choose the number again for calculating. (Exception: If the user does not click any number key, the second number will be default to zero)
* Click f to compute the expression. The result will be displayed on the screen. (

\*Notice: The next calculation will depend on previous calculation (the first number = result of previous calculation) so remember to enter next operator before enter the number (if not the operator is set default for addition calculation)

**3**. **How the program work**

Data section:

* Number: Array of 7-segment display codes for digits 0-9.
* Numpad: Array of key codes for a 4x4 keypad.
* Message: Various messages to be displayed.

Text section:

* **init**: initializes the registers and sets up the initial state of the calculator.
* **main:** Sets the initial display of two LEDs to 0
* Enter Number Loop:
* **enterNumber**: Scans the numpad on each row to detect which key is pressed.
* **loop\_enter**: Iterates through the rows of the numpad.
* **checkNumPad**: Determines which key was pressed by comparing the input with the NumPad array (the number pressed is t2)
* **process\_key**: Determines if a number or an operator key was pressed and jumps to the appropriate label.
* Arithmetic Operation:
* **addition**, **subtraction**, **multiplication**, **division**, **modulo**: Set the operator and change state
* **equals**: Checks which operator was set and jumps to the corresponding calculation.
* Calculation Functions:
* **calc\_add**, **calc\_sub**, **calc\_mul**, **calc\_div**, **calc\_mod**: Perform the respective arithmetic operations, check for overflow, and display the result.
* Display Number part:
* **Display\_Number**: Update two LEDs with the enter number.
* **update\_number1**: Update the first number (s4)
* **update\_number2**: Update the second number (s5)
* **sleep**: pause the program for 100ms
* **reset\_row**: resets the row scanning to the first row.
* **next\_cal**: Prepares for the next calculation by updating s4 with the result and resetting s5.
* **error\_handling**: handle the case like divide, mod by 0 or overflow occur.

**4. Source Code**

.eqv IN\_ADDRESS\_HEXA\_KEYBOARD 0xFFFF0012 # Receive row and column of the key pressed, 0 if not key pressed

.eqv OUT\_ADDRESS\_HEXA\_KEYBOARD 0xFFFF0014

.eqv LEFT\_7SEGMENT\_LED 0xFFFF0011

.eqv RIGHT\_7SEGMENT\_LED 0xFFFF0010

.data

Number: .word 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F

NumPad: .word 0x11, 0x21, 0x41, 0x81, 0x12, 0x22, 0x42, 0x82, 0x14, 0x24, 0x44, 0x84, 0x18, 0x28, 0x48, 0x88

NumPadEnd: .word

mess0: .asciz " and "

mess1: .asciz " is: "

addMess: .asciz "The addition of "

subMess: .asciz "The subtraction of "

mulMess: .asciz "The multiplication of "

divMess: .asciz "The division of "

modMess: .asciz "The modulo of "

newline: .asciz "\n"

prompt1: .asciz "\nContinue the calculation....\n"

prompt2: .asciz "\nCan not divide by 0. Reset the calculator.\n"

prompt3: .asciz "\nCan not mod 0. Reset the calculator.\n"

prompt4: .asciz "\nOverflow occurred. Reset the calculator.\n"

.text

init:

li s0, IN\_ADDRESS\_HEXA\_KEYBOARD

li s1, OUT\_ADDRESS\_HEXA\_KEYBOARD

li s2, LEFT\_7SEGMENT\_LED

li s3, RIGHT\_7SEGMENT\_LED

li s4, 0 #number1 : x1

li s5, 0 #number2 : x2

li s7, 0 #check state

li s8, 10 # operator: + = 10, - = 11, \* = 12, / = 13, % = 14

main:

#store previous value

la a1, Number

lw s6, 0(a1) #start previous led = 0

sb s6, 0(s2) #default left Led display 0

sb s6, 0(s3) #default right Led display 0

enterNumber:

li t0, 0x01 #check row i (start from row i)

li t1, 0x10 #if = row 5 then loop again

loop\_enter:

beq t0, t1, reset\_row

sb t0, 0(s0) #reassign each row

lbu a1, 0(s1) #read enter button

beq a1, zero, update\_loop\_enter

la a2, NumPad # a2 = address(Numpad[0])

la a3, NumPadEnd

addi a3, a3, -4 # a3 = address(NumPad[n-1])

li t2, 0 #number clicked

checkNumPad: #function to find which number was clicked (determined by t2)

lw t3, 0(a2) #t3 = Numpad[i]

beq a1, t3, process\_key #if button clicked is Numpad[i] then go find which button was clicked

addi a2, a2, 4

addi t2, t2, 1

j checkNumPad

process\_key:

li t3, 10 #number range

blt t2, t3, Display\_Number #if t2 < 10 then number was clicked

#else the operator is clicked

beq t2, t3, addition

addi t3, t3, 1

beq t2, t3, subtraction

addi t3, t3, 1

beq t2, t3, multiplication

addi t3, t3, 1

beq t2, t3, division

addi t3, t3, 1

beq t2, t3, modulo

addi t3, t3, 1

beq t2, t3, equals

update\_loop\_enter:

slli t0, t0, 1 #row i + 1

j loop\_enter

addition:

li s8, 10 # operator +

li s7, 1 #change state

j main

subtraction:

li s8, 11 #operator -

li s7, 1 #change state

j main

multiplication:

li s8, 12 #operator x

li s7, 1 #change state

j main

division:

li s8, 13 #opearor /

li s7, 1 #change state

j main

modulo:

li s8, 14 #operator %

li s7, 1 #change state

j main

equals: #check operator

li t3, 10

beq s8, t3, calc\_add

addi t3, t3, 1

beq s8, t3, calc\_sub

addi t3, t3, 1

beq s8, t3, calc\_mul

addi t3, t3, 1

beq s8, t3, calc\_div

addi t3, t3, 1

beq s8, t3, calc\_mod

calc\_add:

add a0, s4, s5

blt a0, s4, error\_overflow #check for overflow

blt a0, s5, error\_overflow #check for overflow

li a7, 4

la a0, addMess

ecall

li a7, 1

add a0, zero, s4

ecall

li a7, 4

la a0, mess0

ecall

li a7, 1

add a0, zero, s5

ecall

li a7, 4

la a0, mess1

ecall

li a7, 1

add a0, s4, s5

ecall

j next\_cal

calc\_sub:

sub a0, s4, s5

bgt a0, s4, error\_overflow #check for overflow

li a7, 4

la a0, subMess

ecall

li a7, 1

add a0, zero, s4

ecall

li a7, 4

la a0, mess0

ecall

li a7, 1

add a0, zero, s5

ecall

li a7, 4

la a0, mess1

ecall

li a7, 1

sub a0, s4, s5

ecall

j next\_cal

calc\_mul:

mul a0, s4, s5

div t3, a0, s5

bne t3, s4, error\_overflow #check for overflow

li a7, 4

la a0, mulMess

ecall

li a7, 1

add a0, zero, s4

ecall

li a7, 4

la a0, mess0

ecall

li a7, 1

add a0, zero, s5

ecall

li a7, 4

la a0, mess1

ecall

li a7, 1

mul a0, s4, s5

ecall

j next\_cal

calc\_div:

beq s5, zero, error\_division

li a7, 4

la a0, divMess

ecall

li a7, 1

add a0, zero, s4

ecall

li a7, 4

la a0, mess0

ecall

li a7, 1

add a0, zero, s5

ecall

li a7, 4

la a0, mess1

ecall

li a7, 1

div a0, s4, s5

ecall

j next\_cal

calc\_mod:

beq s5, zero, error\_modulo

li a7, 4

la a0, modMess

ecall

li a7, 1

add a0, zero, s4

ecall

li a7, 4

la a0, mess0

ecall

li a7, 1

add a0, zero, s5

ecall

li a7, 4

la a0, mess1

ecall

li a7, 1

rem a0, s4, s5

ecall

j next\_cal

Display\_Number:

#print number entered on LED

la a1, Number #a1 = address of Number[0]

slli t3, t2, 2

add a1, a1, t3

lw a2, 0(a1)

sb a2, 0(s3) #Right Led display the value entered

sb s6, 0(s2) #Left Led display the previous value entered

mv s6, a2 #update previous value = value entered

#update value base on state

li t3, 10

beq s7, zero, update\_number1 #update the first Number

j update\_number2 #update the second Number

update\_number1:

mul s4, s4, t3 #x1 \*= 10

add s4, s4, t2 #x1 += x (value was entered)

j sleep

update\_number2:

mul s5, s5, t3 #x2 \*= 10

add s5, s5, t2 #x2 += x (value was entered)

j sleep

sleep:

li a0, 100 # sleep 100ms

li a7, 32

ecall

reset\_row:

li t0, 0x01

j loop\_enter

next\_cal:

mv s4, a0 #update x1 = calculation result

li s5, 0 #reset x2 = 0

li a7, 4

la a0, prompt1

ecall

li s8, 10 #reset operator to default "+"

j main

#handle case divide or mod by 0

error\_handling:

error\_division:

li a7, 4

la a0, prompt2

ecall

j init

error\_modulo:

li a7, 4

la a0, prompt3

ecall

j init

#handle overflow

error\_overflow:

li a7, 4

la a0, prompt4

ecall

j init

1. **Code Demo**

A screenshot of a computer

Description automatically generatedStart implement:

A screenshot of a computer

Description automatically generated

Press button: 1

A screenshot of a computer

Description automatically generatedPress button: 2

A screenshot of a computer

Description automatically generatedPress button: b (perfrom subtraction)

When performing operator, reset LED to 0 for entering next number

A screenshot of a computer

Description automatically generatedPress button: 5

Press button: f (equals)

A screenshot of a computer

Description automatically generatedAfter press, it will print out the value on the screen and continue the next calculation.