HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

----***----

ASSEMBLY LANGUAGE AND COMPUTER ARCHITECTURE LAB

FINAL PROJECT REPORT

IT3280E

Instructor: Lê Bá Vui

Group 14

Name	Student ID	Problem
Nguyễn Đình Phúc	20226062	5
Vũ Thái Hưng	20226046	4

Hanoi, 2024

Table of contents

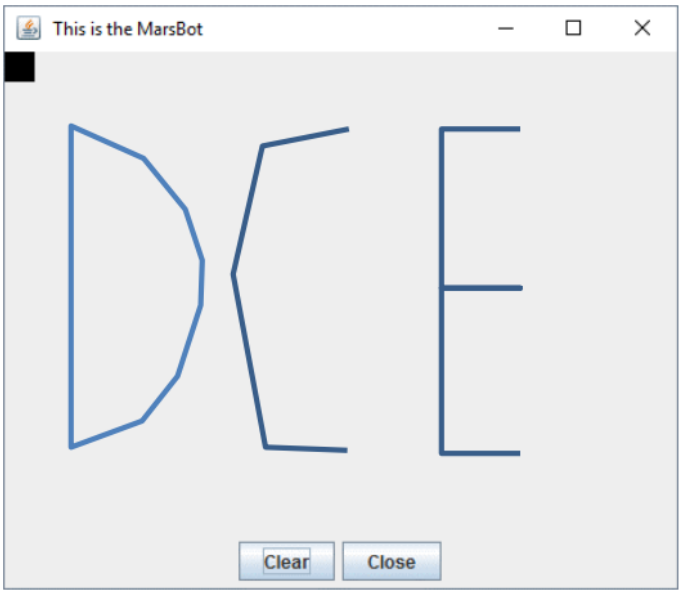
Task 4. Postscript CNC Marsbot

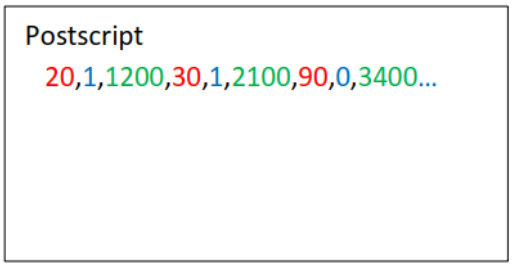
1. Problem description

CNC Marsbot is used to cut the metal panel according to predetermined lines. It has a cutting blade that moves across the metal plate, with the assumption that:
- If the blade moves but does not cut the metal plate, it means the Marsbot moves but does not leave a track.
- If the blade moves and cuts the metal plate, Marsbot will move and leave a track.

To control Marsbot to cut into the desired shape, Marsbot will be loaded a script which is a string consisting of 3 consecutive elements:
- <ANGLE>, <CUT/UNCUT>, <DURATION>
- <ANGLE> is the angle of HEADING command of Marsbot
- <CUT/UNCUT> leave or does not leave a track.
- <DURATION> time for current operation

Create a program that Marsbot can do:
- Cut the metal panel as described above.
- The content of scripts is hardcoded in the source code.
- The source code includes 3 scripts and users can press 0, 4 or 8 in the key matrix to select the script to execute.
- One script should contain DCE. Two remaining scripts are proposed by students (at least 10 lines)





2. Source code and explanation

2.1. Main Source Code:

```
.data
# declaring table
# postscript length = numberOfLines*3
postscript1:    .word       180,1,6000, 90,1,1500, 60,1,1000, 0,1,5000, 300,1,1000, 270,1,1500, 90,0,6500, 270,1,1500, 240,1,1000, 180,1,5000, 120,1,1000, 90,1,1500, 90,0,2500, 270,1,1500, 0,1,3000, 90,1,1500, 270,0,1500, 0,1,3000, 90,1,1500
postscript1_length:     .word   57
postscript2:    .word       180,1,6000, 90,1,1500, 60,1,1000, 0,1,5000, 300,1,1000, 270,1,1500, 90,0,5000, 270,1,1500, 180,1,3000, 90,1,1500, 270,0,1500, 180,1,3000, 90,1,1500, 90,0,1500, 0,1,3000, 135,1,4200, 315,0,4200, 45,1,4200, 225,0,4200, 0,1,3000, 90,0,5000, 180,1,5500, 120,1,1000, 90,1,1000, 60,1,1000, 0,1,5500
postscript2_length:     .word   78
postscript3:    .word       90,1,1500, 270,0,1500, 240,1,1000, 180,1,5000, 120,1,1000, 90,1,1500, 60,1,1000, 0,1,5000, 300,1,1000, 90,0,2500, 180,1,6000, 0,0,6000, 120,1,1500, 60,1,1500, 180,1,6000, 90,0,2000, 0,0,500, 0,1,5000, 60,1,1000, 90,1,1500, 120,1,1000, 180,1,1500, 0,0,1500, 300,0,1000, 270,0,1500, 240,0,1000, 180,0,5000, 120,1,1000, 90,1,1500, 60,1,1000, 0,1,1500, 270,0,1000, 90,1,2000
postscript3_length:     .word   99

Msg:    .asciiz     "Invalid input for postscript!\n"
Msg1:   .asciiz     "You choose to print DCE!\n"
enter:  .asciiz     "\n"
Msg2:   .asciiz     "You choose to print DEKU!\n"
Msg3:   .asciiz     "You choose to print OMG!\n"
space:  .asciiz     " "

.eqv    IN_ADDRESS_HEXA_KEYBOARD    0xFFFF0012
.eqv    OUT_ADDRESS_HEXA_KEYBOARD       0xFFFF0014

.eqv    HEADING     0xffff8010      # Integer: An angle between 0 and 359
                                    # 0 : North (up)
                                    # 90: East (right)
                                    # 180: South (down)
                                    # 270: West (left)
.eqv    MOVING              0xffff8050      # Boolean: whether or not to move
.eqv    LEAVETRACK          0xffff8020      # Boolean (0 or non-0):
                                    # whether or not to leave a track
.eqv    WHEREX              0xffff8030      # Integer: Current x-location of MarsBot
.eqv    WHEREY              0xffff8040      # Integer: Current y-location of MarsBot

.text
        li      $t4, 0              # count number of successful postscript
        li      $t5, 0              # check whether or not have postscript1 done
                                    # 0 - not yet, 1 - done => t4 += 1
                                    # t4 === 3 => all 3 postscript are done => complete
        li      $t6, 0              # check whether or not have postscript2 done
        li      $s5, 0              # check whether or not have postscript3 done
polling:
row1:
        li      $t1, IN_ADDRESS_HEXA_KEYBOARD
        li      $t2, OUT_ADDRESS_HEXA_KEYBOARD
        li      $t3, 0x01                       # check row 1 with key 0, 1, 2, 3
        sb      $t3, 0($t1)                     # must reassign expected row
        lb      $a0, 0($t2)                     # read scan code of key button
        bne     $a0, 0x00000011, row2           # 0 - postscript1
        li      $v0, 4
        la      $a0, Msg1
```

```
        syscall
        la      $t8, postscript1
        bne     $t5, 0, postscript1_already_done
        li      $t5, 1                          # postscript1 done
        addi    $t4, $t4, 1                     # done 1 postscript
postscript1_already_done:
        la      $t7, postscript1_length
        lw      $t7, 0($t7)
        j       main
        nop

row2:
        li      $t1, IN_ADDRESS_HEXA_KEYBOARD
        li      $t2, OUT_ADDRESS_HEXA_KEYBOARD
        li      $t3, 0x02                       # check row 2 with key 4, 5, 6, 7
        sb      $t3, 0($t1)                     # must reassign expected row
        lb      $a0, 0($t2)                     # read scan code of key button
        bne     $a0, 0x00000012, row3           # 4 - postscript2
        li      $v0, 4
        la      $a0, Msg2
        syscall
        la      $t8, postscript2
        bne     $t6, 0, postscript2_already_done
        li      $t6, 1                          # postscript2 done
        addi    $t4, $t4, 1                     # done 1 postscript
postscript2_already_done:
        la      $t7, postscript2_length
        lw      $t7, 0($t7)
        j       main
        nop

row3:
        li      $t1, IN_ADDRESS_HEXA_KEYBOARD
        li      $t2, OUT_ADDRESS_HEXA_KEYBOARD
        li      $t3, 0x04                       # check row 3 with key 8, 9, A, B
        sb      $t3, 0($t1)                     # must reassign expected row
        lb      $a0, 0($t2)                     # read scan code of key button
        bne     $a0, 0x00000014, invalid        # 8 - postscript3
        li      $v0, 4
        la      $a0, Msg3
        syscall
        la      $t8, postscript3
        bne     $s5, 0, postscript3_already_done
        li      $s5, 1                          # postscript3 done
        addi    $t4, $t4, 1                     # done 1 postscript
postscript3_already_done:
        la      $t7, postscript3_length
        lw      $t7, 0($t7)
        j       main
        nop

invalid:
        li      $v0, 4
        la      $a0, Msg
        syscall
sleep_wait:
        li      $a0, 1000                       # sleep 1000ms
        li      $v0, 32
        syscall
        j       polling
```

```
main:
# Go to cut area
        jal     UNTRACK             # no draw track line
        addi    $s2, $zero, 135  # Marsbot rotates given radius and start
start_running:
        jal     ROTATE
        jal     GO
start_sleep:
        addi    $v0,$zero,32        # Keep running by sleeping in 1000 ms
        addi    $a0,$zero,5000
        syscall

        jal     UNTRACK             # keep old track
        #jal    TRACK                       # and draw new track line

        li      $s0, 0              # Set index counter for postscript array
loop:
        beq     $s0, $t7, end_loop  # if i == numberOfLines*3 then quit
        sll     $s1, $s0, 2         # s1 = 4i
        add     $s1, $s1, $t8       # s1 = A[i]'s address
        lw      $s2, 0($s1)         # s2 = A[i]'s value - move radius
        addi    $s1, $s1, 4
        lw      $s3, 0($s1)         # s3 = A[i+1]'s value - cut/not cut
        addi    $s1, $s1, 4
        lw      $s4, 0($s1)         # s4 = A[i+2]'s value - time per move

        jal     TRACK_UNTRACK               # draw track line/ or not
running:
        jal     ROTATE
        jal     GO
sleep:
        addi    $v0,$zero,32        # Keep running by sleeping in 1000 ms
        add     $a0,$zero,$s4
        syscall

        jal     UNTRACK             # keep old track
        #jal    TRACK                       # and draw new track line

        addi    $s0, $s0, 3         # iterate next 3 values
        j       loop
end_loop:
        jal     STOP
        #j      end_main
        beq     $t4, 3, end_main    # done 3 postscript -> done
        j       polling
end_main:
        li      $v0, 10
        syscall

#-------------------------------------------------------
# GO procedure, to start running
# param[in] none
#-------------------------------------------------------

GO:
        li      $at, MOVING         # change MOVING port
        addi    $k0, $zero,1        # to logic 1,
        sb      $k0, 0($at)         # to start running
        jr      $ra
```

```
#-----------------------------------------------------------
# STOP procedure, to stop running
# param[in] none
#-----------------------------------------------------------

STOP:
        li      $at, MOVING             # change MOVING port to 0
        sb      $zero, 0($at)           # to stop
        jr      $ra

#-----------------------------------------------------------
# TRACK procedure, to start drawing line
# param[in] none
#-----------------------------------------------------------

TRACK_UNTRACK:
        li      $at, LEAVETRACK         # change LEAVETRACK port
        sb      $s3, 0($at)             # to start tracking/ or not
        jr      $ra

#-----------------------------------------------------------
# UNTRACK procedure, to stop drawing line
# param[in] none
#-----------------------------------------------------------

TRACK:
        li      $at, LEAVETRACK         # change LEAVETRACK port
        addi    $k0, $zero,1            # to logic 1,
        sb      $k0, 0($at)             # to start tracking
        jr      $ra

UNTRACK:
        li      $at, LEAVETRACK         # change LEAVETRACK port to 0
        sb      $zero, 0($at)           # to stop drawing tail
        jr      $ra
#-----------------------------------------------------------
# ROTATE procedure, to rotate the robot
# param[in] $a0, An angle between 0 and 359
# 0 : North (up)
# 90: East (right)
# 180: South (down)
# 270: West (left)
#-----------------------------------------------------------
ROTATE:
        li      $at, HEADING            # change HEADING port
        sw      $s2, 0($at)             # to rotate robot
        jr      $ra
```

2.2. Explanation:

n04_g14_VuThaiHung.asm

```
1  .data
2  # declaring table
3  # postscript length = numberOfLines*3
4  postscript1:    .word          180,1,6000, 90,1,1500, 60,1,1000, 0,1,5000, 300,1,1000, 270,1,1500, 90,0,6500, 270,1,1500, 240,1,1000, 180,1,5000, 1
5  postscript1_length:    .word   57
6  postscript2:    .word          180,1,6000, 90,1,1500, 60,1,1000, 0,1,5000, 300,1,1000, 270,1,1500, 90,0,5000, 270,1,1500, 180,1,3000, 90,1,1500, 2
7  postscript2_length:    .word   78
8  postscript3:    .word          90,1,1500, 270,0,1500, 240,1,1000, 180,1,5000, 120,1,1000, 90,1,1500, 60,1,1000, 0,1,5000, 300,1,1000, 90,0,2500, 18
9  postscript3_length:    .word   99
10
11 Msg:    .asciiz        "Invalid input for postscript!\n"
12 Msg1:   .asciiz        "You choose to print DCE!\n"
13 enter:  .asciiz        "\n"
14 Msg2:   .asciiz        "You choose to print DEKU!\n"
15 Msg3:   .asciiz        "You choose to print OMG!\n"
16 space:  .asciiz        " "
17
18 .eqv    IN_ADDRESS_HEXA_KEYBOARD       0xFFFF0012
19 .eqv    OUT_ADDRESS_HEXA_KEYBOARD      0xFFFF0014
20
21 .eqv    HEADING         0xffff8010     # Integer: An angle between 0 and 359
22                                        # 0 : North (up)
23                                        # 90: East (right)
24                                        # 180: South (down)
25                                        # 270: West (left)
26 .eqv    MOVING          0xffff8050     # Boolean: whether or not to move
27 .eqv    LEAVETRACK      0xffff8020     # Boolean (0 or non-0):
```

- From lines 1 to 30: initializing the values of postscript and log messages, defining the in/out addresses of the digital lab sim and the components of MarsBot.

```
32 .text
33         li      $t4, 0                  # count number of successful postscript
34         li      $t5, 0                  # check whether or not have postscript1 done
35                                         # 0 - not yet, 1 - done => t4 += 1
36                                         # t4 === 3 => all 3 postscript are done => complete
37         li      $t6, 0                  # check whether or not have postscript2 done
38         li      $s5, 0                  # check whether or not have postscript3 done
```

- Starting the task: initializing the values as follows:
  - $t4 = 0: used to count the number of drawn postscripts.
  - $t5 = 0: used to check if postscript 1 has been executed or not.
  - $t6 = 0: used to check if postscript 2 has been executed or not.
  - $s5 = 0: used to check if postscript 3 has been executed or not.

```
39 polling:
40 row1:
41         li      $t1, IN_ADDRESS_HEXA_KEYBOARD
42         li      $t2, OUT_ADDRESS_HEXA_KEYBOARD
43         li      $t3, 0x01                       # check row 1 with key 0, 1, 2, 3
44         sb      $t3, 0($t1)                     # must reassign expected row
45         lb      $a0, 0($t2)                     # read scan code of key button
46         bne     $a0, 0x00000011, row2           # 0 - postscript1
47         li      $v0, 4
48         la      $a0, Msg1
49         syscall
50         la      $t8, postscript1
51         bne     $t5, 0, postscript1_already_done
52         li      $t5, 1                          # postscript1 done
53         addi    $t4, $t4, 1                     # done 1 postscript
54 postscript1_already_done:
55         la      $t7, postscript1_length
56         lw      $t7, 0($t7)
57         j       main
58         nop
```

```
row2:
        li      $t1, IN_ADDRESS_HEXA_KEYBOARD
        li      $t2, OUT_ADDRESS_HEXA_KEYBOARD
        li      $t3, 0x02                               # check row 2 with key 4, 5, 6, 7
        sb      $t3, 0($t1)                             # must reassign expected row
        lb      $a0, 0($t2)                             # read scan code of key button
        bne     $a0, 0x00000012, row3                   # 4 - postscript2
        li      $v0, 4
        la      $a0, Msg2
        syscall
        la      $t8, postscript2
        bne     $t6, 0, postscript2_already_done
        li      $t6, 1                                  # postscript2 done
        addi    $t4, $t4, 1                             # done 1 postscript
postscript2_already_done:
        la      $t7, postscript2_length
        lw      $t7, 0($t7)
        j       main
        nop
row3:
        li      $t1, IN_ADDRESS_HEXA_KEYBOARD
        li      $t2, OUT_ADDRESS_HEXA_KEYBOARD
        li      $t3, 0x04                               # check row 3 with key 8, 9, A, B
        sb      $t3, 0($t1)                             # must reassign expected row
        lb      $a0, 0($t2)                             # read scan code of key button
        bne     $a0, 0x00000014, invalid                # 8 - postscript3
        li      $v0, 4
        la      $a0, Msg3
        syscall
        la      $t8, postscript3
        bne     $s5, 0, postscript3_already_done
        li      $s5, 1                                  # postscript3 done
        addi    $t4, $t4, 1                             # done 1 postscript
postscript3_already_done:
        la      $t7, postscript3_length
        lw      $t7, 0($t7)
        j       main
        nop

invalid:
        li      $v0, 4
        la      $a0, Msg
        syscall
sleep_wait:
        li      $a0, 1000                               # sleep 1000ms
        li      $v0, 32
        syscall
        j       polling
```

- Processing in the Digital Lab Sim – Key matrix:
  - Load the in/out address values into $t1, $t2, assigning respectively $t3 = 0x01 – row 1, 0x02 – row 2, 0x04 – row 4, then specify the row where we want to press a key by storing $t3 into in address. After pressing the key – corresponding to the desired row (e.g., row 1 – including 0, 1, 2, 3), the value of the key pressed is stored in $a0 (loaded from out address). Check the value of $a0: if $a0 is not equal to 0x00000011 – representing the value of key 0, then the pressed key is not 0. In this case, we move to row 2 to check if the pressed key belongs to row 2 (key 4) with a mechanism similar to the one described. Proceeding, if the pressed key is not 4, we jump to row 3 to check if key 8 was pressed. If the pressed key is not 8, then the pressed key is not within the allowed format (0, 4, 8), and an alert is displayed: "Invalid input for postscript!"
  - One noteworthy aspect is the labels postscript1/2/3_already_done: their purpose is to check if postscript 1/2/3 has been selected for cutting. If it has not been selected for cutting and the corresponding key is pressed, it marks that it has been cut and increments the count of drawn postscripts by 1. o If the user inputs incorrectly and falls into the invalid label, the user must choose a key until the correct format (0, 4, 8) is entered (as shown in the invalid label -> jump back to polling).
  - If the corresponding Key for the postscript is pressed, $t8 is assigned as the address of the postscript array, and $t7 is the number of elements in the array storing the postscripts, then jump to the label main – where metal cutting is performed and MarsBot moves.

```
110 main:
111 # Go to cut area
112         jal     UNTRACK                 # no draw track line
113         addi    $s2, $zero, 135         # Marsbot rotates given radius and start
114 start_running:
115         jal     ROTATE
116         jal     GO
117 start_sleep:
118         addi    $v0,$zero,32            # Keep running by sleeping in 1000 ms
119         addi    $a0,$zero,5000
120         syscall
121
122         jal     UNTRACK                 # keep old track
123         #jal    TRACK                   # and draw new track line
124
125         li      $s0, 0                  # Set index counter for postscript array
```

- The first task upon reaching the main function is to move MarsBot to the cutting area (because if left at the initial position, it would be hard to see – might even lose track due to being out of range). Firstly, UNTRACK is done to avoid marking the path the MarsBot has taken, then it moves downward at a 45-degree angle (labeled 135 in the code due to MarsBot's convention – as indicated in the code) for a distance of 5 seconds = 5000ms. MarsBot's movement: rotate -> move – following the time at label start_sleep -> jal UNTRACK – to save the track that the MarsBot has traversed. If UNTRACK is not present at the end after the robot moves, the track that the robot just moved on will be pulled along with the new track, leading to undesired outcomes.

```
126 loop:
127         beq     $s0, $t7, end_loop      # if i == numberOfLines*3 then quit
128         sll     $s1, $s0, 2             # s1 = 4i
129         add     $s1, $s1, $t8           # s1 = A[i]'s address
130         lw      $s2, 0($s1)             # s2 = A[i]'s value - move radius
131         addi    $s1, $s1, 4
132         lw      $s3, 0($s1)             # s3 = A[i+1]'s value - cut/not cut
133         addi    $s1, $s1, 4
134         lw      $s4, 0($s1)             # s4 = A[i+2]'s value - time per move
135
136         jal     TRACK_UNTRACK           # draw track line/ or not
137 running:
138         jal     ROTATE
139         jal     GO
140 sleep:
141         addi    $v0,$zero,32            # Keep running by sleeping in 1000 ms
142         add     $a0,$zero,$s4
143         syscall
144
145         jal     UNTRACK                 # keep old track
146         #jal    TRACK                   # and draw new track line
147
148         addi    $s0, $s0, 3             # iterate next 3 values
149         j       loop
150 end_loop:
151         jal     STOP
152         #j      end_main
153         beq     $t4, 3, end_main        # done 3 postscript -> done
154         j       polling
```

- Beginning the loop:
  - Set $s0 = 0 = i – representing the index to iterate through the postscript array.
  - If $s0 = $t7 – the size of the postscript array = number of cutting lines (the actual number of cutting lines differs from the observed number of cuts by eyes – explained below) * 3 (at the beginning of the array, there are 3 parts: angle; cut/not cut; time – the length of the cut).
  - To retrieve elements from the array, initially, set $s1 = 4 * $s0, then get $s1 + $t8 - $t8 currently storing the address of the postscript array to retrieve the address of the i-th element in the array -> use lw to fetch

the value at that address. At this point, the value retrieved is the angle – saved into $s2, then get the address of A[i + 1] (A – the postscript array) by taking $s1 – currently storing the address A[i] + 4 (1 word = 4 bytes => + 4 gets to the next word's address), then save the values of A[i + 1] and A[i + 2] into $s3, $s4 respectively ($s3 determines cut/not cut and $s4 determines the length of the cut).

- After obtaining the 3 necessary values for one cut, we begin to execute the MarsBot procedure mentioned above. First, call the subfunction TRACK_UNTRACK (depending on $s3) to determine if we've stored the track of MarsBot's path or not -> ROTATE (based on the value of $s2) -> command MarsBot to GO -> label sleep – deciding how long MarsBot moves -> UNTRACK – saving the path traveled to avoid being overwritten by a new path. NOTE: the line addi $t2,$t2,4 is redundant and unrelated to the code. Once this process is complete, increment the index by 3 to reach the element A[i + 3]– storing the angle of the next cutting line and continue looping until the exit condition is met -> achieving the desired drawing.
- At the label end_loop: stop the MarsBot to prevent it from endlessly following the previous track. Check if all 3 postscripts have been drawn – by checking $t4 === 3 ? true : false, if not, allow the user to continue using it – as the requirement in the task is to draw/cut 3 postscripts.

```
155 end_main:
156         li      $v0, 10
157         syscall
```

- Function end_main: This function concludes the program and exits to prevent infinite looping or address errors when encountering the part of the code that handles the MarsBot procedure.

```
159 #-------------------------------------------------------------
160 # GO procedure, to start running
161 # param[in] none
162 #-------------------------------------------------------------
163
164 GO:
165         li      $at, MOVING             # change MOVING port
166         addi    $k0, $zero,1            # to logic 1,
167         sb      $k0, 0($at)             # to start running
168         jr      $ra
```

- Subfunction handling MarsBot movement by setting MOVING = 1, with no input parameters.

```
170 #-------------------------------------------------------------
171 # STOP procedure, to stop running
172 # param[in] none
173 #-------------------------------------------------------------
174
175 STOP:
176         li      $at, MOVING             # change MOVING port to 0
177         sb      $zero, 0($at)           # to stop
178         jr      $ra
179
```

- Subfunction handling MarsBot movement by setting MOVING = 0, with no input parameters.

```
#-------------------------------------------------------------
# TRACK procedure, to start drawing line
# param[in] none
#-------------------------------------------------------------

TRACK_UNTRACK:
        li      $at, LEAVETRACK         # change LEAVETRACK port
        sb      $s3, 0($at)             # to start tracking/ or not
        jr      $ra

#-------------------------------------------------------------
# UNTRACK procedure, to stop drawing line
# param[in] none
#-------------------------------------------------------------

TRACK:
        li      $at, LEAVETRACK         # change LEAVETRACK port
        addi    $k0, $zero,1            # to logic 1,
        sb      $k0, 0($at)             # to start tracking
        jr      $ra

UNTRACK:
        li      $at, LEAVETRACK         # change LEAVETRACK port to 0
        sb      $zero, 0($at)           # to stop drawing tail
        jr      $ra
```

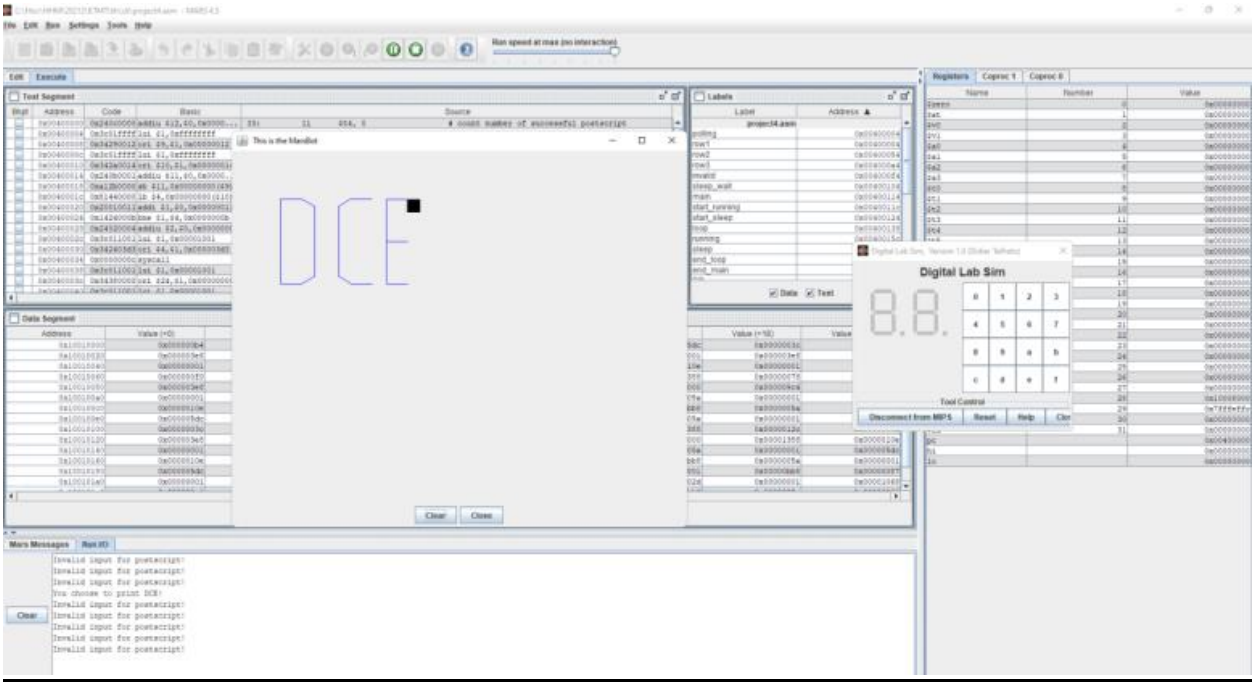
- Function TRACK_UNTRACK: Determines whether to store the trace in the current cut – depends on $s3.
- Function TRACK: Handles MarsBot leaving a trace, without any input parameters.
- Function UNTRACK: Handles MarsBot not leaving a trace, without any input parameters.

```
#-------------------------------------------------------------
# ROTATE procedure, to rotate the robot
# param[in] $a0, An angle between 0 and 359
# 0 : North (up)
# 90: East (right)
# 180: South (down)
# 270: West (left)
#-------------------------------------------------------------
ROTATE:
        li      $at, HEADING            # change HEADING port
        sw      $s2, 0($at)             # to rotate robot
        jr      $ra
```

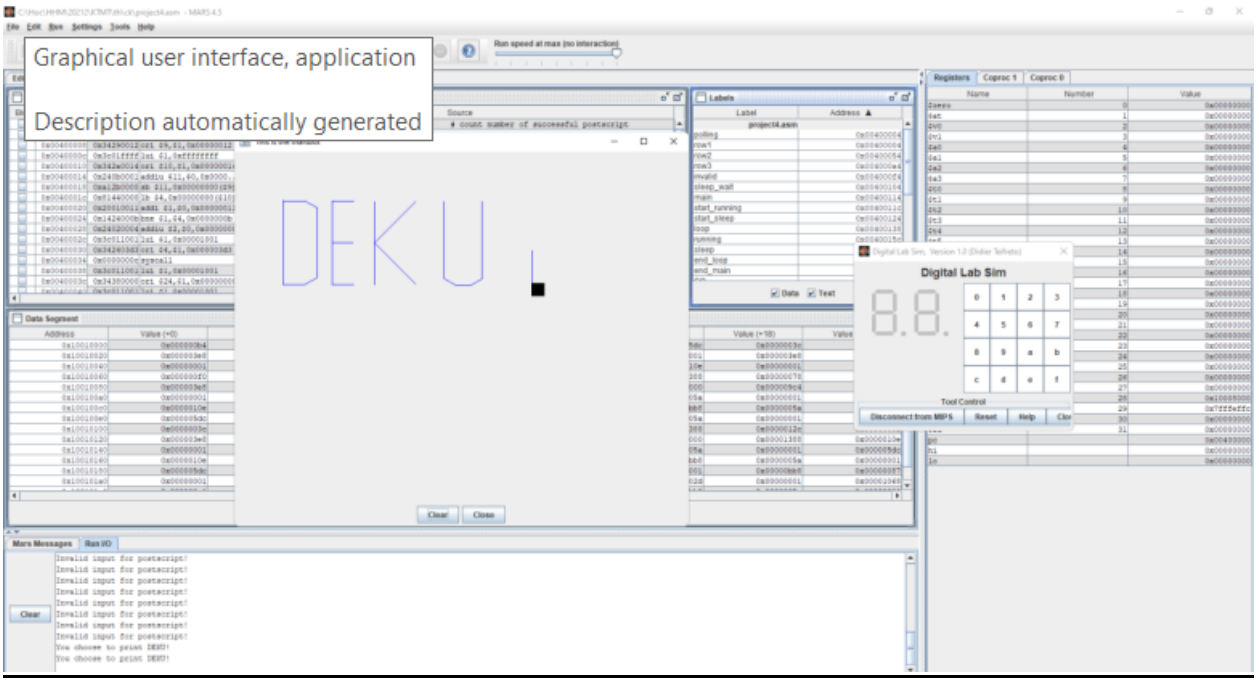
- Function rotate: This function manages the direction of MarsBot's movement, with no input value.

3. Result

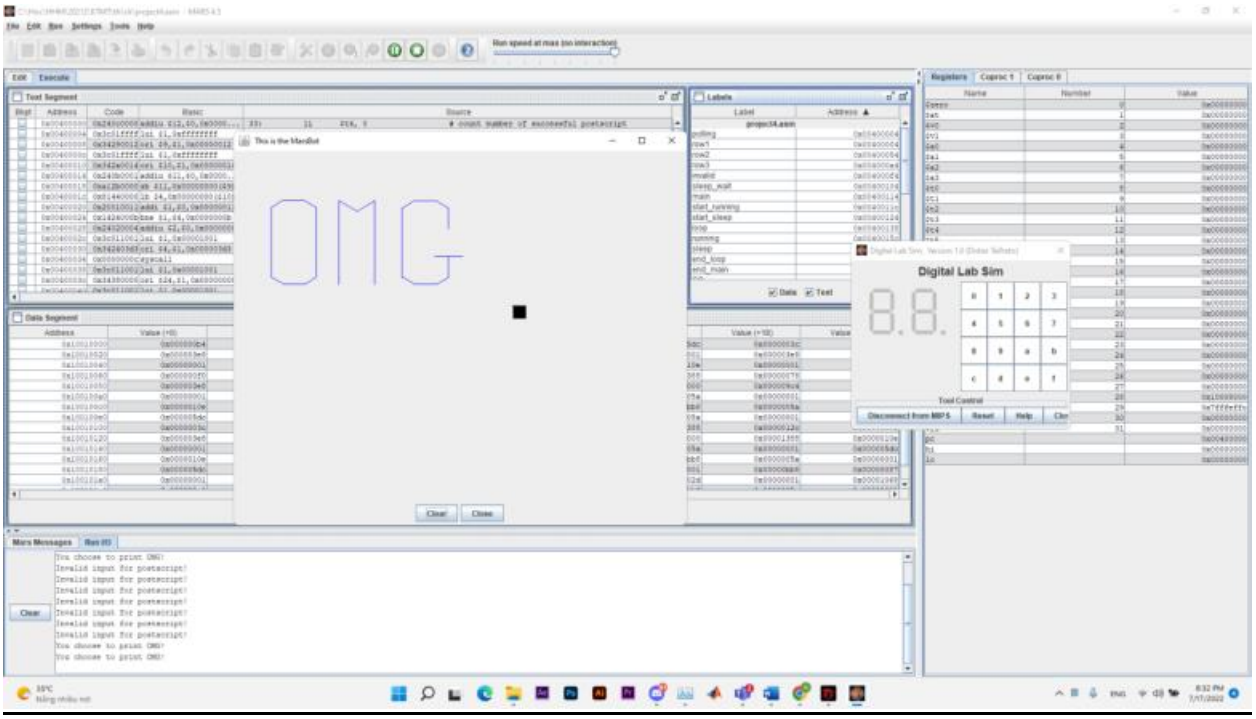
3.1. Postscript 1



3.2. Postscript 2



3.3. Postscript 3



Task 5. Infix and Postfix expression

1. Problem description

Create a program that can calculate an expression by evaluating the postfix expression.
Requirements:
1. Enter an infix expression from the console, for example: 9 + 2 + 8 * 6
2. Print it in the postfix representation, for example: 9 2 + 8 6 * +
3. Calculate and display the result to the console screen.
The operand must be an integer between 0 and 99.
Operators include addition, subtraction, multiplication, division, division with remainder and parenthesis.

2. Source code and explanation

a) Variable setting

```
.data
	infix: .space 256
	postfix: .space 256
	operator: .space 256
	endMsg: .asciiz "Do you want to type in another infix expression?"
	byeMsg: .asciiz "Goodbye!Have a good day"
	errorMsg: .asciiz "Input error!"
	startMsg: .asciiz "Please enter infix expression\nNote: only allowed to use + - * / ()\nRange:Natural number from 00-99"
	postfix_notif: .asciiz "Postfix expression: "
	result_notif: .asciiz "Result: "
	infix_notif: .asciiz "Infix expression: "
	converter: .word 1
	wordToConvert: .word 1
	stack: .float
```

- Setting up the space for the input, the postfix and the stack operator
- Create message variables, float converter and the float stack use to calculate expression in the postfix form.

b) Initialization

```
.text
start:
# Get infix expression
        li $v0, 54
        la $a0, startMsg
        la $a1, infix
        la $a2, 256
        syscall
        beq $a1,-2,end
        beq $a1,-3,start
# Print infix
        li $v0, 4
        la $a0, infix_notif
        syscall
        li $v0, 4
        la $a0, infix
        syscall
        li $v0, 11
        li $a0, '\n'
        syscall
# Status
        li $s7,0                # Status
                                # 0 = initially receive nothing
                                # 1 = receive number
                                # 2 = receive operator
                                # 3 = receive (
                                # 4 = receive )

        li $t9,0                # Count digit
        li $t5,-1               # Postfix top offset
        li $t6,-1               # Operator top offset
        la $t1, infix           # Infix current byte address +1 each loop
        la $t2, postfix
        la $t3, operator
        addi $t1,$t1,-1         # Set initial address of infix to -1
```

- In this code, a status variable is defined in $t7. This variable will be 0 if receive nothing, change to 1 if receive number, 2 if receive operator, 3 if receive open bracket, 4 if receive close bracket.
- Create digit counting variable in $t9, which count the digit that input has been received before encountering a bracket or operator
- Installing $t5: postfix top offset, $t6: operator stack top offset
- Storing the address of the input to $t1 and then point to the 1st element of the input string. The space buffer for postfix and stack operator are also loaded in $t2 and $t3

c) Scanning input and Storing digits, operators and brackets

```
scanInfix:                  # Loop for each character in postfix
# Check all valid input option
        addi $t1,$t1,1              # Increase infix position
        lb $t4, ($t1)               # Load current infix input
        beq $t4, ' ', scanInfix     # If scan spacebar ignore and scan again
        beq $t4, '\n', EOF          # Scan end of input --> pop all operator to postfix
        beq $t9,0,digit1            # If state is 0 digit
        beq $t9,1,digit2            # If state is 1 digit
        beq $t9,2,digit3            # If state is 2 digit
        continueScan:
        beq $t4, '+', plusMinus
        beq $t4, '-', plusMinus
        beq $t4, '*', multiplyDivide
        beq $t4, '/', multiplyDivide
        beq $t4, '(', openBracket
        beq $t4, ')', closeBracket

wrongInput:     # When detect wrong input situation
        li $v0, 55
        la $a0, errorMsg
        li $a1, 2
        syscall
        j ask
```

- Start scanning each character of the input from left to right and check whether it's a digit, operators, brackets or the '\n' keyword.
- If none of them occurs → Invalid character → error box appears.
- If no error there will be **three case** appears below:

First case: The character currently scanning is a digit

```
digit1:
        beq $t4,'0',store1Digit
        beq $t4,'1',store1Digit
        beq $t4,'2',store1Digit
        beq $t4,'3',store1Digit
        beq $t4,'4',store1Digit
        beq $t4,'5',store1Digit
        beq $t4,'6',store1Digit
        beq $t4,'7',store1Digit
        beq $t4,'8',store1Digit
        beq $t4,'9',store1Digit
        j continueScan

digit2:
        beq $t4,'0',store2Digit
        beq $t4,'1',store2Digit
        beq $t4,'2',store2Digit
        beq $t4,'3',store2Digit
        beq $t4,'4',store2Digit
        beq $t4,'5',store2Digit
        beq $t4,'6',store2Digit
        beq $t4,'7',store2Digit
        beq $t4,'8',store2Digit
        beq $t4,'9',store2Digit
        # If do not receive second digit
        jal numberToPost
        j continueScan
digit3:
        # If scan third digit --> error
        beq $t4,'0',wrongInput
        beq $t4,'1',wrongInput
        beq $t4,'2',wrongInput
        beq $t4,'3',wrongInput
        beq $t4,'4',wrongInput
        beq $t4,'5',wrongInput
        beq $t4,'6',wrongInput
        beq $t4,'7',wrongInput
        beq $t4,'8',wrongInput
        beq $t4,'9',wrongInput
        # If do not receive third digit
        jal numberToPost
        j continueScan
```

- digit1: if there are no digits before it → jump to store1Digit
- digit2: if there are 1 digit scanned before it → jump to store2Digit
- digit3: if there are 2 digits scanned before it → the number will have 3 digits →Out of range number 00-99 –> Jump to wrongInput

Second case: The input is an operator "+ - * /"

```
plusMinus:                  # Input is + -
        beq $s7,2,wrongInput        # Receive operator after operator or open bracket
        beq $s7,3,wrongInput
        beq $s7,0,wrongInput        # Receive operator before any number
        li $s7,2                    # Change input status to 1
        continuePlusMinus:
        beq $t6,-1,inputToOp        # There is nothing in Operator stack --> push into
        add $t8,$t6,$t3             # Load address of top Operator
        lb $t7,($t8)                # Load byte value of top Operator
        beq $t7,'(',inputToOp       # If top is ( --> push into
        beq $t7,'+',equalPrecedence # If top is + -
        beq $t7,'-',equalPrecedence
        beq $t7,'*',lowerPrecedence # If top is * /
        beq $t7,'/',lowerPrecedence
multiplyDivide:             # Input is * /
        beq $s7,2,wrongInput        # Receive operator after operator or open bracket
        beq $s7,3,wrongInput
        beq $s7,0,wrongInput        # Receive operator before any number
        li $s7,2                    # Change input status to 1
        beq $t6,-1,inputToOp        # There is nothing in Operator stack --> push into
        add $t8,$t6,$t3             # Load address of top Operator
        lb $t7,($t8)                # Load byte value of top Operator
        beq $t7,'(',inputToOp       # If top is ( --> push into
        beq $t7,'+',inputToOp       # If top is + - --> push into
        beq $t7,'-',inputToOp
        beq $t7,'*',equalPrecedence # If top is * /
        beq $t7,'/',equalPrecedence
```

- Handling error: if $s7, the status variable that contain the status of the last character scanned, equal to the status of operator of an open bracket or operator before any number → **Error!**
- Check for the operator offset $t6 and load address of top operator to $t8, load the value in address $t8 to $t7 →Check value of $t7 ($t7 store the top operator). For the **plusMinus** operator check:
    - If top operator is the open bracket → jump to inputToOp to store in the stack.
    - If top is + or – (Same precedence), →Jump to equalPrecedence
    - If top is * or /(Lower predence), →Jump to lowerPrecedence
- For the multiplyDivide operator check:
    - If top operator is the open bracket → jump to inputToOp to store in the stack.
    - If top is + or – (Lower precedence), →Jump to inputToOp to store in the stack.
    - If top is * or /(Same predence), →Jump to lowerPrecedence

Third case: The input is a bracket

```
openBracket:                    # Input is (
        beq $s7,1,wrongInput            # Receive open bracket after a number or close bracket
        beq $s7,4,wrongInput
        li $s7,3                        # Change input status to 1
        j inputToOp
closeBracket:                   # Input is )
        beq $s7,2,wrongInput            # Receive close bracket after an operator or operator
        beq $s7,3,wrongInput
        li $s7,4
        add $t8,$t6,$t3                 # Load address of top Operator
        lb $t7,($t8)                    # Load byte value of top Operator
        beq $t7,'(',wrongInput          # Input contain () without anything between --> error
        continueCloseBracket:
        beq $t6,-1,wrongInput           # Can't find an open bracket --> error
        add $t8,$t6,$t3                 # Load address of top Operator
        lb $t7,($t8)                    # Load byte value of top Operator
        beq $t7,'(',matchBracket        # Find matched bracket
        jal opToPostfix                 # Pop the top of Operator to Postfix
        j continueCloseBracket          # Then loop again till find a matched bracket or error
```

- Input is open bracket → check if open bracket are not received after open bracket or close bracket, if yes → wrongInput
- Else: Jump to the inputToOp
- CloseBracket: Loop to pop out top element in the stack operator until ‘(‘ appears. If there are open bracket matched → jump to matchBracket to discard that pair of bracket

```
matchBracket:                   # Discard a pair of matched brackets
        addi $t6,$t6,-1                 # Decrement top of Operator offset
        j scanInfix
```

After processing the case, the program will jump to the storing process.
Case 1: Storing operator:

```
equalPrecedence:        # Mean receive + - and top is + - || receive * / and top is * /
        jal opToPostfix                 # Pop the top of Operator to Postfix
        j inputToOp                     # Push the new operator in
lowerPrecedence:        # Mean receive + - and top is * /
        jal opToPostfix                 # Pop the top of Operator to Postfix
        j continuePlusMinus             # Loop again
inputToOp:                      # Push input to Operator
        add $t6,$t6,1                   # Increment top of Operator offset
        add $t8,$t6,$t3                 # Load address of top Operator
        sb $t4,($t8)                    # Store input in Operator
        j scanInfix
opToPostfix:                    # Pop top of Operator in push into Postfix
        addi $t5,$t5,1                  # Increment top of Postfix offset
        add $t8,$t5,$t2                 # Load address of top Postfix
        addi $t7,$t7,100                # Encode operator + 100
        sb $t7,($t8)                    # Store operator into Postfix
        addi $t6,$t6,-1                 # Decrement top of Operator offset
        jr $ra
```

- **EqualPrecedence:** Pop up operator at the top and push it to the postfix and push the scanned operator in.
- **lowerPrecedence:** Pop the top element and then continue to loop until it has no greater or equal precedence operator.
- **inputToOp:** push Input to operator stack
- **opToPostfix:** pop the top operator in stack and insert to the postfix expression. Storing the length of the postfix in $t5

Case 2: Storing number

```
store1Digit:
        beq $s7,4,wrongInput            # Receive number after )
        addi $s4,$t4,-48                # Store the number: the actual ASCII Code of a digit = the ASCII code of the digit in character form -48
        add $t9,$zero,1                 # Change status to 1 digit
        li $s7,1                        # Change the Receiving status to 1
        j scanInfix                     # Jump back to scanning procedure
store2Digit:
        beq $s7,4,wrongInput            # Receive number after )
        addi $s5,$t4,-48                # Store the number: the actual ASCII Code of a digit = the ASCII code of the digit in character form -48
        mul $s4,$s4,10

        add $s4,$s4,$s5                 # Stored number = first digit * 10 + second digit
        add $t9,$zero,2                 # Change status to 2 digit
        li $s7,1                        # Change the Receiving status to 1
        j scanInfix                     # Jump back to scanning procedure
numberToPost:
        beq $t9,0,endnumberToPost
        addi $t5,$t5,1
        add $t8,$t5,$t2
        sb $s4,($t8)                    # Store number in Postfix
        add $t9,$zero,$zero             # Change status to 0 digit
        endnumberToPost:
        jr $ra  #If no digits received -- jump back to continueScan.
```

- Store1Digit in $s4
- Store2Digit: the number stored = 10 * first digit + the second digit
- numberToPost: Push the number to postfix expression and reset the count digit to 0.

d)Printing postfix

```
finishScan:
# Print postfix expression
        # Print prompt:
        li $v0, 4
        la $a0, postfix_notif
        syscall
        li $t6,-1               # Load current of Postfix offset to -1
printPost:
        addi $t6,$t6,1          # Increment current of Postfix offset
        add $t8,$t2,$t6         # Load address of current Postfix
        lbu $t7,($t8)           # Load value of current Postfix
        bgt $t6,$t5,finishPrint # Print all postfix --> calculate
        bgt $t7,99,printOp      # If current Postfix > 99 --> an operator
        # If not then current Postfix is a number
        li $v0, 1
        add $a0,$t7,$zero
        syscall
        li $v0, 11
        li $a0, ' '
        syscall
        j printPost             # Loop
        printOp:
        li $v0, 11
        addi $t7,$t7,-100       # Decode operator
        add $a0,$t7,$zero
        syscall
        li $v0, 11
        li $a0, ' '
        syscall
        j printPost             # Loop
finishPrint:
        li $v0, 11
        li $a0, '\n'
        syscall
```

- Set postfix offset $t6 to -1, scan the postfix and print element
- If $t6 = $t5 (the length of the postfix) →printed all the expression

d) Calculating results using the postfix expression

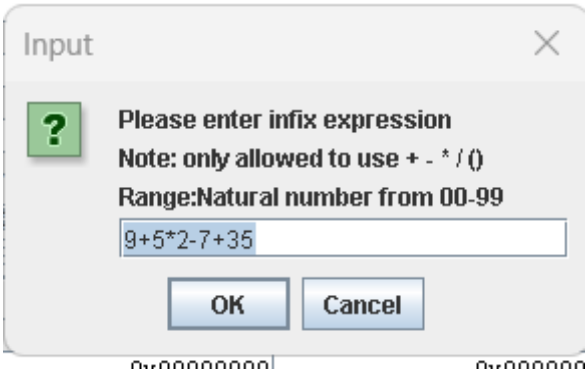
```
# Calculate
        li $t9,-4               # Set top of stack offset to -4
        la $t3,stack            # Load stack address
        li $t6,-1               # Load current of Postfix offset to -1
        l.s $f0,converter       # Load converter
calPost:
        addi $t6,$t6,1          # Increment current of Postfix offset
        add $t8,$t2,$t6         # Load address of current Postfix
        lbu $t7,($t8)           # Load value of current Postfix
        bgt $t6,$t5,printResult # If $t6(current postfix offset = $t5(the length of the postfix calculated in the scanning and storing procedure) --> Calculate for all postfix --> print
        bgt $t7,99,calculate    # If current Postfix > 99 --> an operator --> popout 2 number to calculate
        # If not then current Postfix is a number
        addi $t9,$t9,4          # Current stack top offset
        add $t4,$t3,$t9         # Current stack top address
        sw $t7,wordToConvert
        l.s $f10,wordToConvert  # Load number to coproc1 to convert to float
        div.s $f10,$f10,$f0
        s.s $f10,($t4)          # Push number into stack
        sub.s $f10,$f10,$f10    # Reset f10
        j calPost               # Loop
        calculate:
                # Pop current top number
                add $t4,$t3,$t9 #Assign $t4 to the address of the current top element of the stack
                l.s $f3,($t4) #load the value on address $t4 to the address $f3 to calculate in float number
                # Pop next number
                addi $t9,$t9,-4 #Move to next element
                add $t4,$t3,$t9 #Assign $t4 to the address of that element
                l.s $f2,($t4) #Store that value in $f2
                # Decode operator
                beq $t7,143,plus
                beq $t7,145,minus
                beq $t7,142,multiply
                beq $t7,147,divide
```

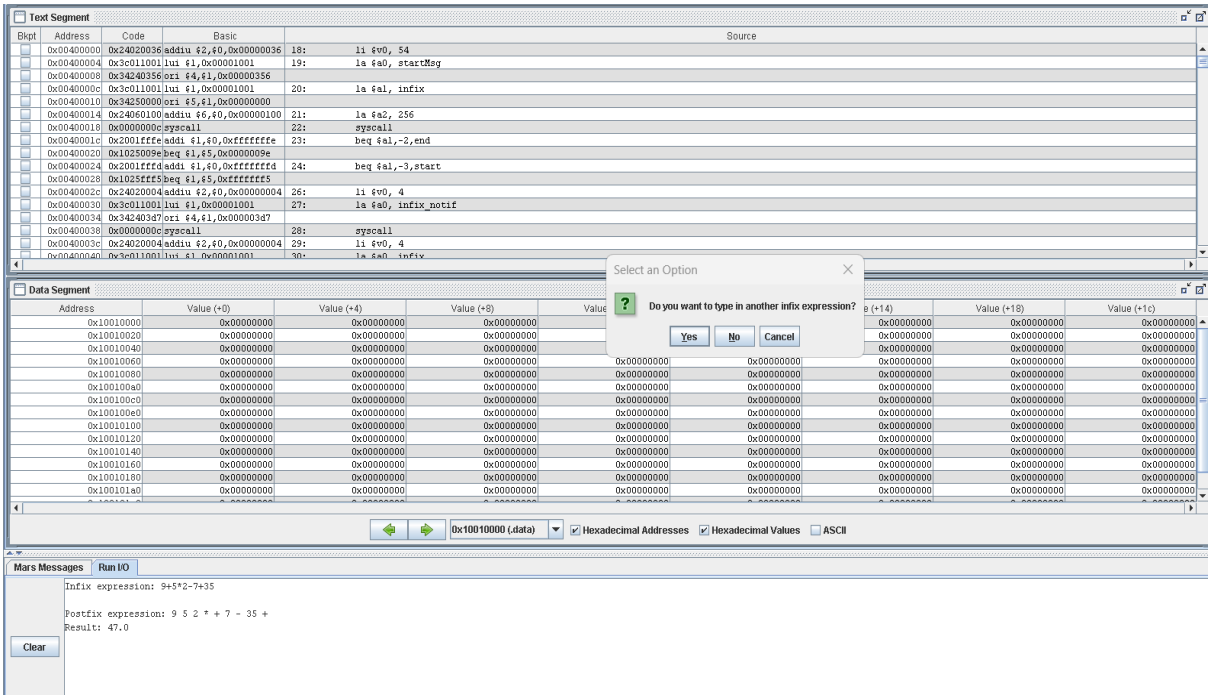
```
plus:
        add.s $f1,$f2,$f3
        s.s $f1,($t4)
        sub.s $f2,$f2,$f2       # Reset f2 f3 to the value of 0
        sub.s $f3,$f3,$f3
        j calPost
minus:
        sub.s $f1,$f2,$f3
        s.s $f1,($t4)
        sub.s $f2,$f2,$f2       # Reset f2 f3 to the value of 0
        sub.s $f3,$f3,$f3
        j calPost
multiply:
        mul.s $f1,$f2,$f3
        s.s $f1,($t4)
        sub.s $f2,$f2,$f2       # Reset f2 f3 to the value of 0
        sub.s $f3,$f3,$f3
        j calPost
divide:
        div.s $f1,$f2,$f3
        s.s $f1,($t4)
        sub.s $f2,$f2,$f2       # Reset f2 f3 to the value of 0
        sub.s $f3,$f3,$f3
        j calPost

printResult:
        li $v0, 4
        la $a0, result_notif
        syscall
        li $v0, 2
        l.s $f12,($t4) #value to print is in the $t4 address,  loaded in $f12
        syscall
        li $v0, 11
        li $a0, '\n'
        syscall
ask:                    # Ask user to continue or not
        li $v0, 50
        la $a0, endMsg
        syscall
        beq $a0,0,start #If yes --> back to start
```

- Scan the postfix operator using $t6 offset.
- If encounter an operator, pop up the top two element, do the calculation and push it to the stack.
- If encounter an operand, pop operand from the postfix to the stack
- The result is in the float stack

3. **Result**

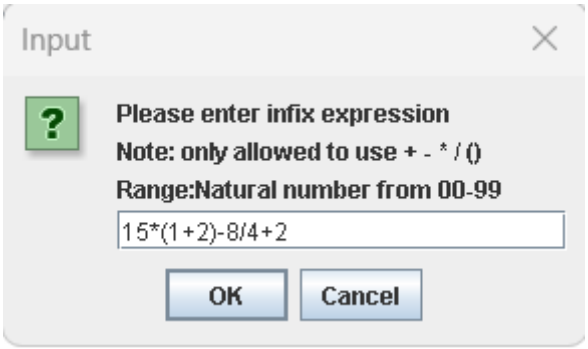
case 1: 9+5*2-7+35 (Valid infix expression)

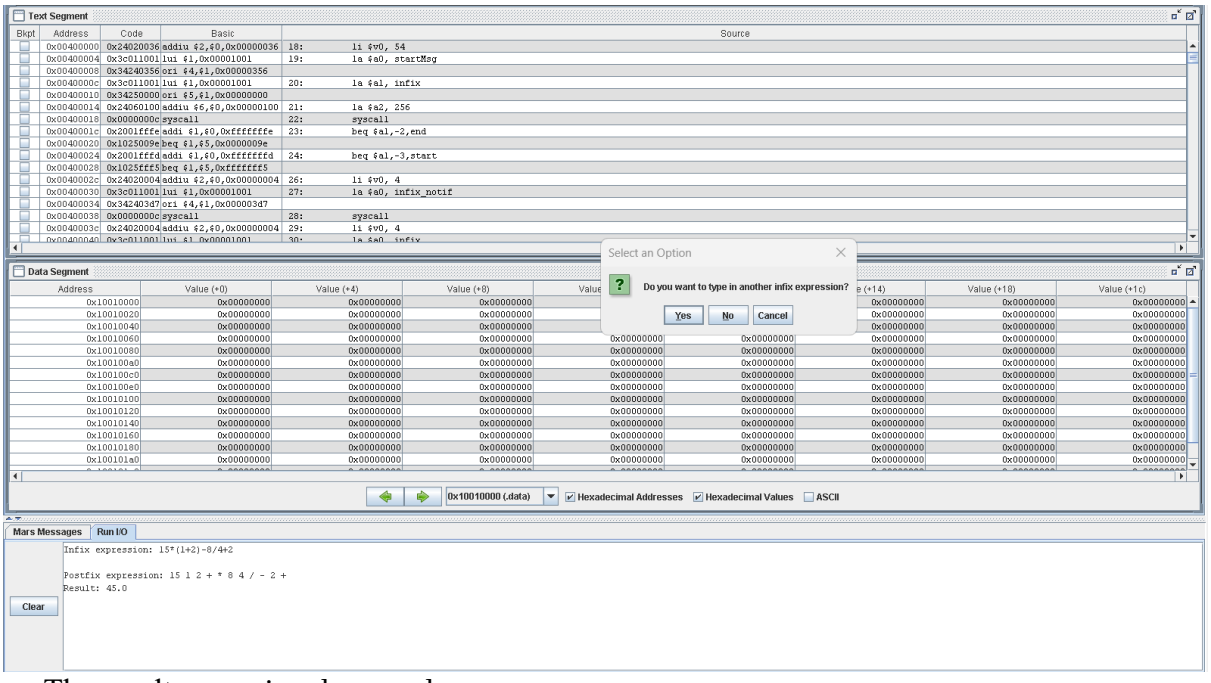




→ The result printed correctly

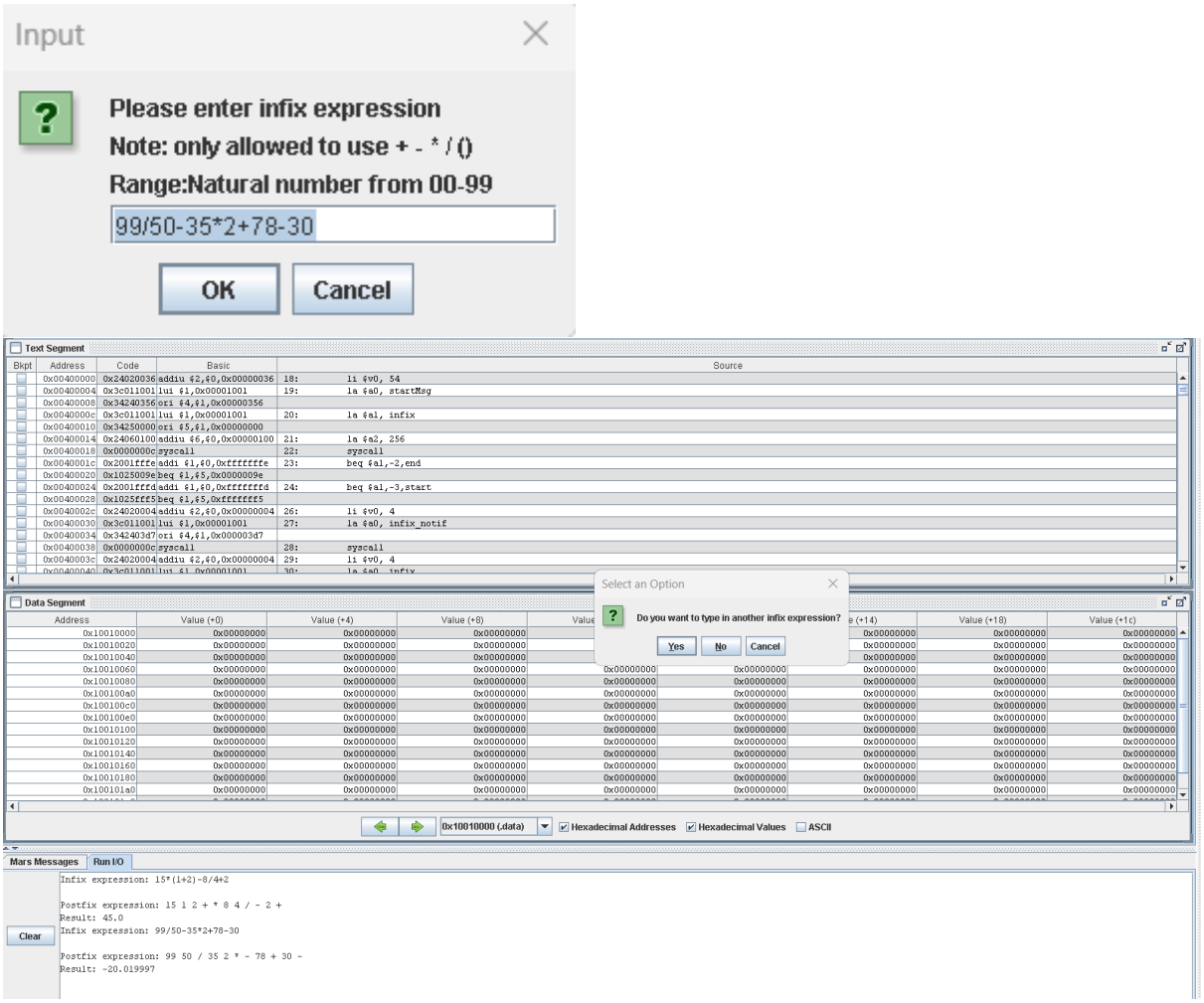
case 2: 15*(1+2)-8/4+2





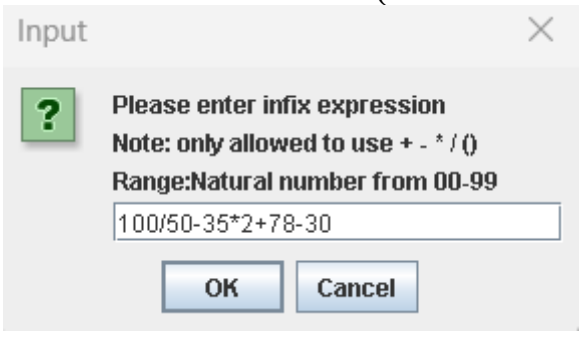
→ The result was printed correctly

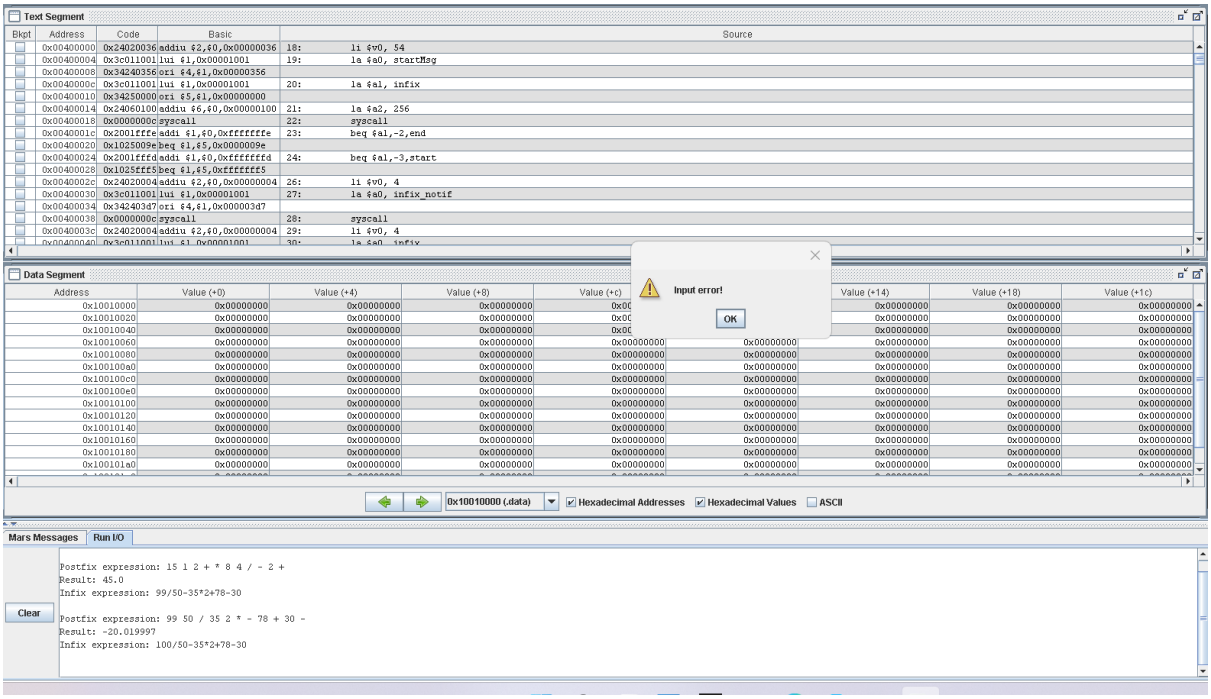
Case 3: 99/50-35*2+78-30



→ The result was printed correctly

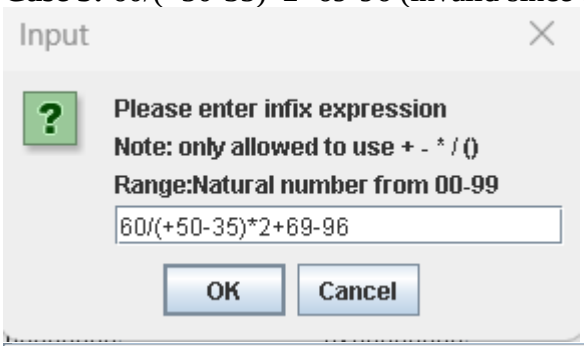
Case 4: 100/50-35*2+78-30 (invalid infix expression)

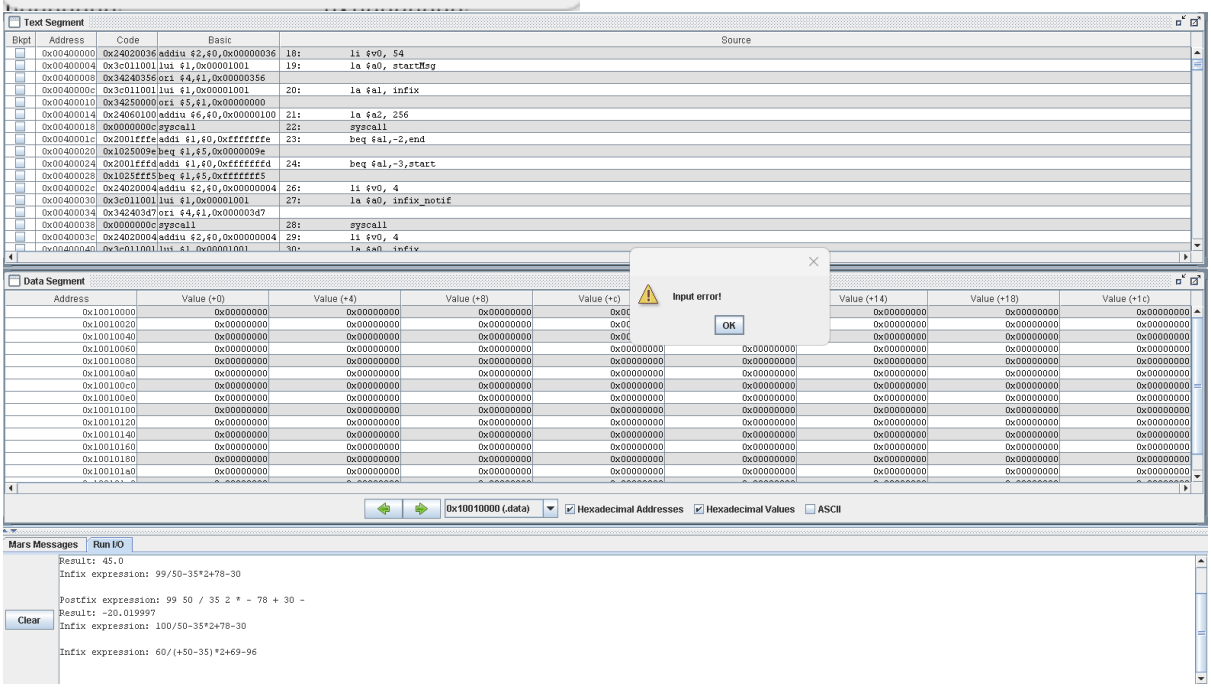




→ The error Number out of range was handled.

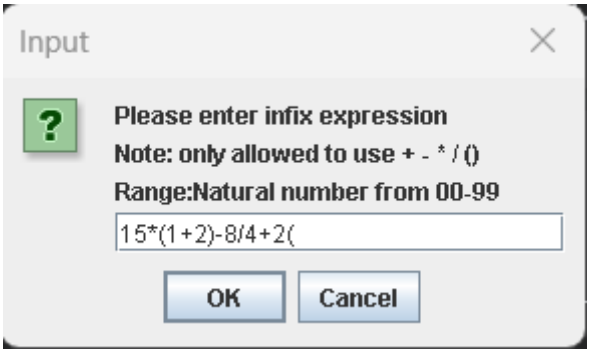
Case 5: 60/(+50-35)*2+69-96 (invalid since operator right begin open bracket)

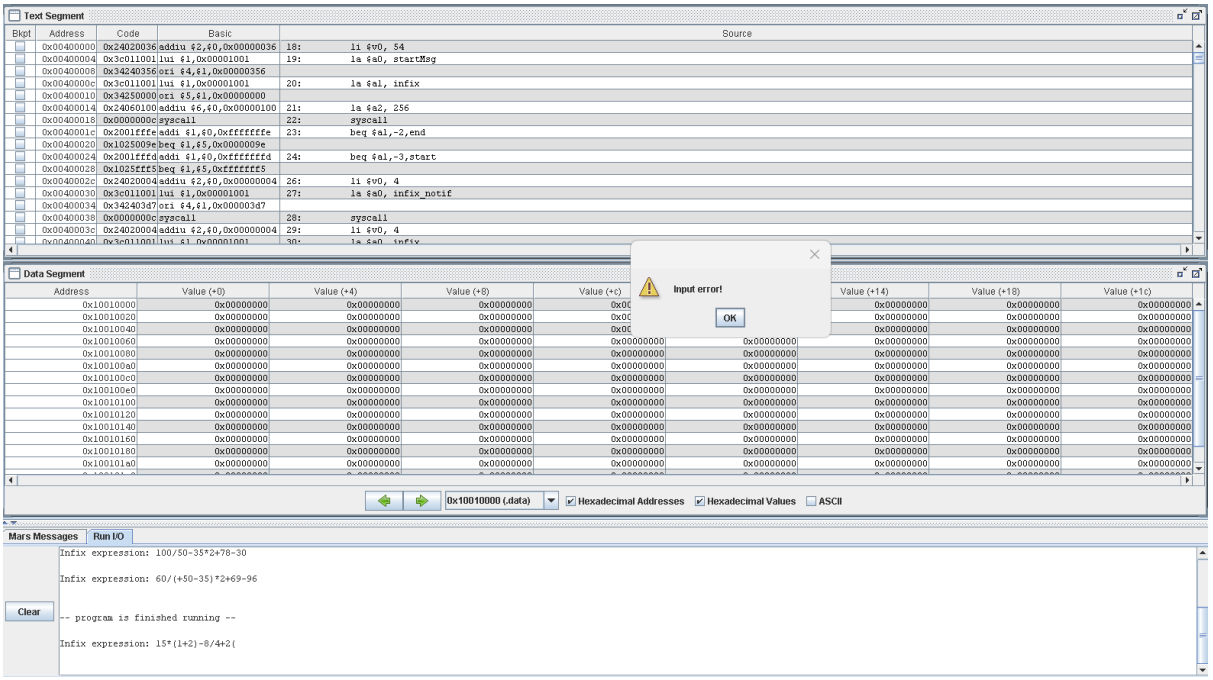




→ The error was handled.

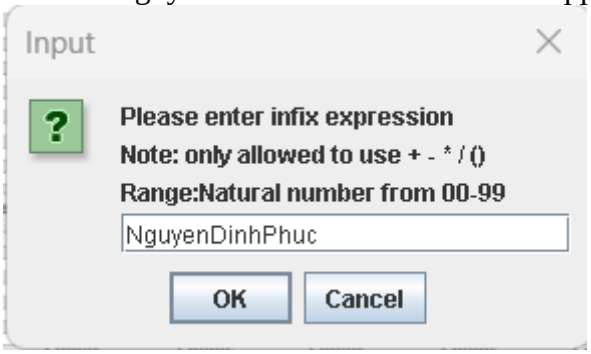
Case 6: 15*(1+2)-8/4+2(
Error: end with an open bracket

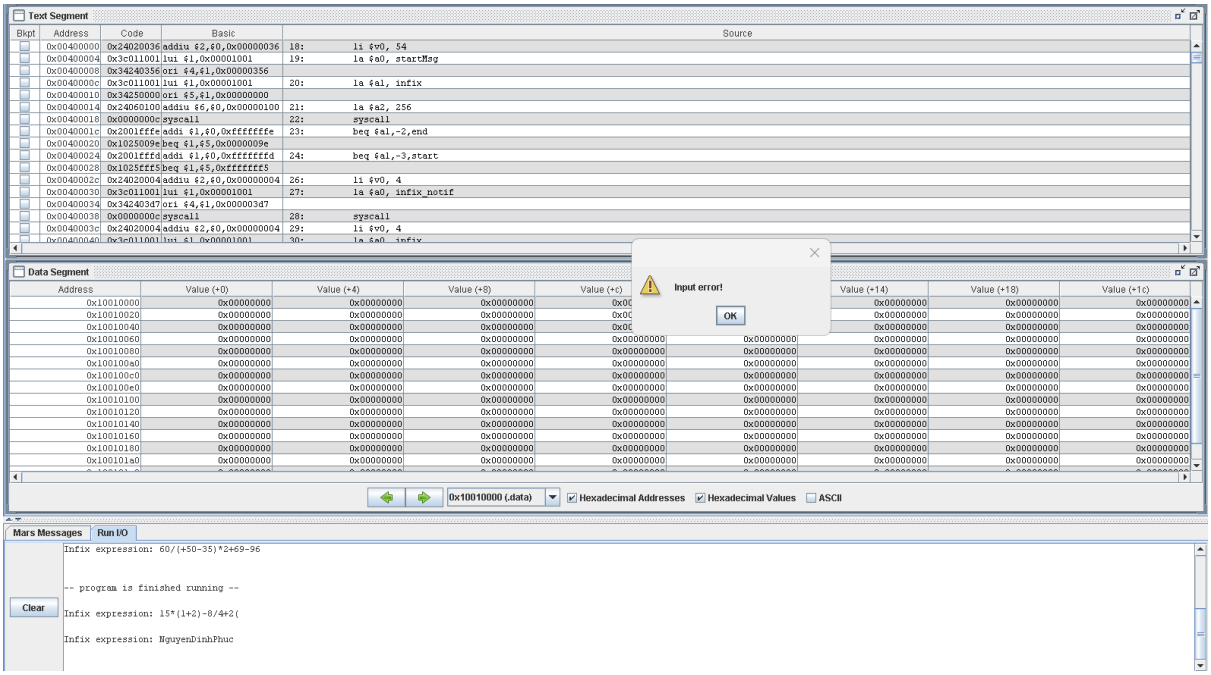




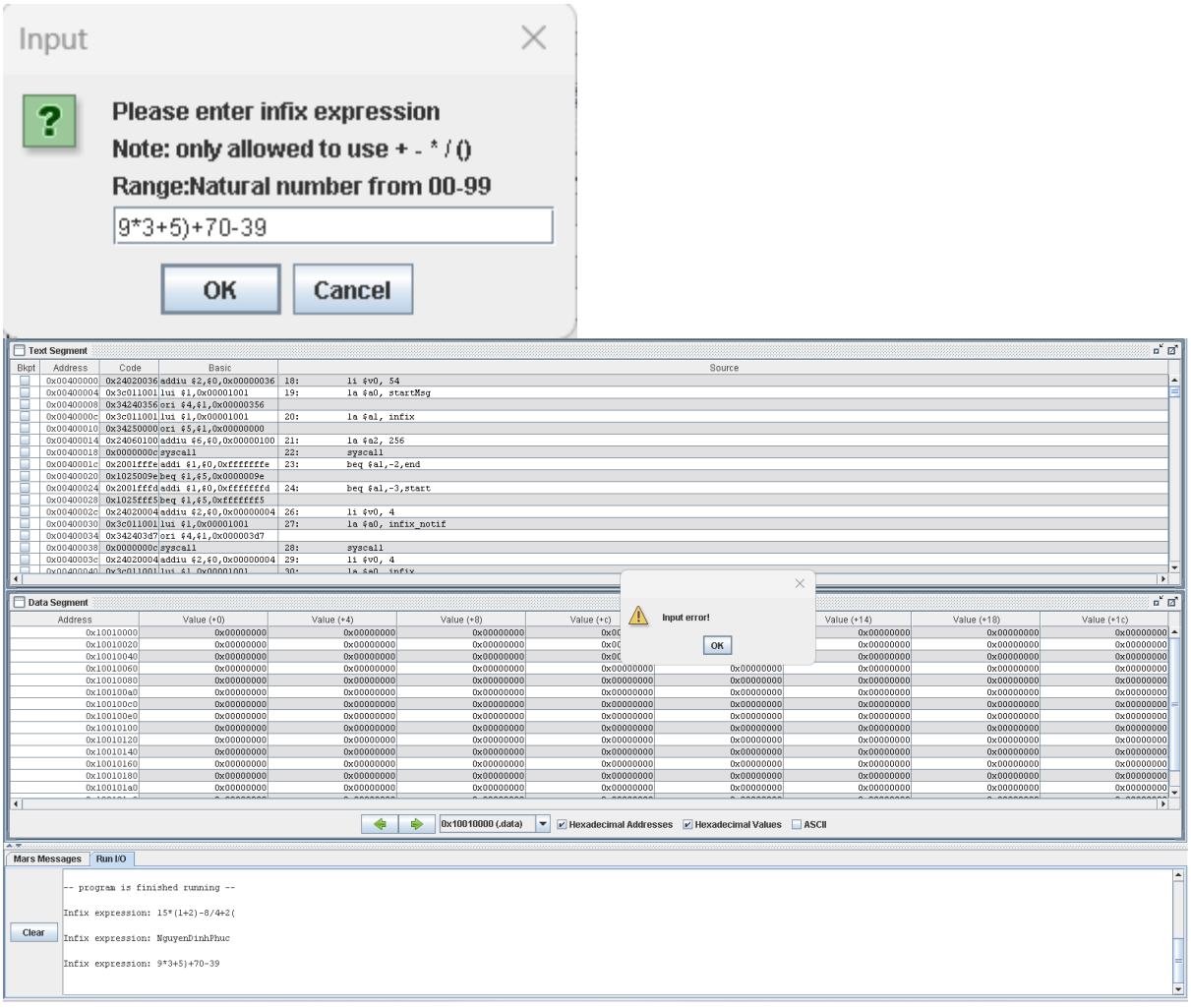
→ The error was handled

Case 7: NguyenDinhPhuc → The error inappropriate character handled



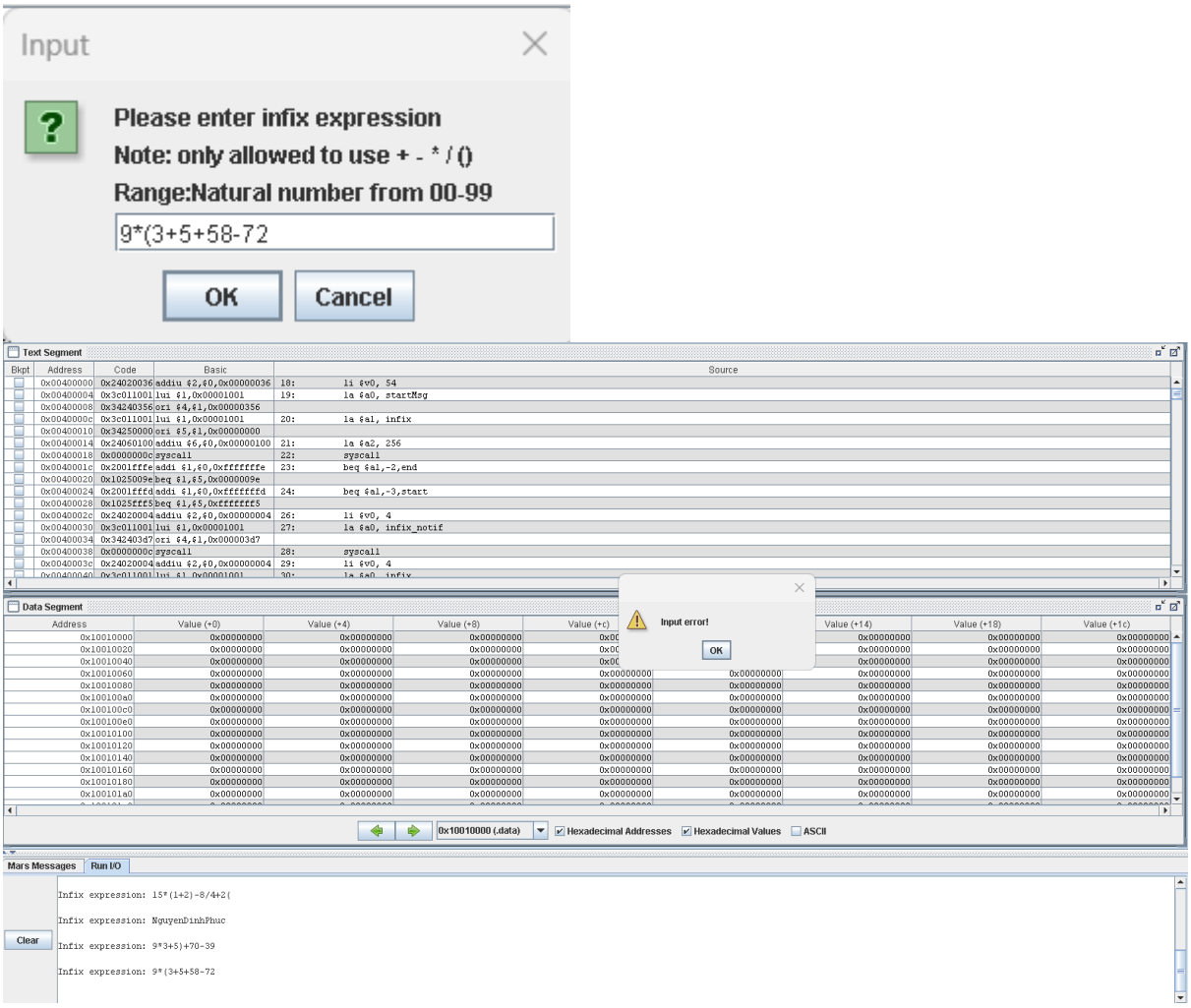


Case 8: 9*3+5)+70-39



→ The error: “No open bracket fits with the close one!”handled

Case 9: 9*(3+5+58-72
Error: no close bracket



→ The error was handled