SYS-102 Basic Computer Architecture
Assessment Activity SYS-102:00080
Assembly Programming

Name: (Win) Thanawin Pattanaphol **Student ID**: 01324096

Code Analysis

Main section initializes user input by loading user_input to the register t1 and calls the secret_operation section.

The secret_operation sections is the main part that manipulate values where it stores and retrieves values from the stack, applies bitwise operations such as: XOR and OR, math calculations using instructions like: addi, sub, add, and slli (shift left) and builds up result that will be compared to user_input. A code snipping for each example:

```
secret_operation:
    addi sp, sp, -28
    li t2, 0x10
    addi t2, t2, 1
    xori t2, t2, 0x4
    sw t2, 0(sp)
```

Figure 1: Snipping of data manipulation

We can see all the examples in figure 1:

- The yellow highlighted instruction uses the addi instruction which basically says: sp = sp + (-28), sp in this case is the stack pointer and this operation essentially shifts the stack pointer by -28 which is a way to allocate space.
- The orange highlighted instruction demonstrates the retrieve process where li (load immediate) loads the value 0x10 to the t2 register
- Blue highlighted instruction demonstrates similar action as the yellow highlighted line as the instruction addi is used meaning: t2 = t2 + 1
- The green highlighted instruction does a bitwise operation where the instruction XORs t2 and 0x04
- The red highlighted instruction stores a word (4 bytes) *whose data is from the memory at the address stored in t0* into the register t2.

Another instruction that was used is the beqz instruction in the main section where it is used to compare if the value of a0 (return value of secret_operation) is zero – if not, jump to secure exit.

```
main:
    la t0, user_input
    lw t1, 0(t0)
    jal ra, secret_operation
    beqz a0, secure_exit
```

Figure 2: Main section

Key elements and operations

a) The code transforms parts of the passcode by using instructions such as, silli, andi, xori, addi and ori to manipulate or combine values in the various t registers. Some examples include the use of the addi instruction above to increment the value of t2 register by 1 and the use of the xori instruction to store a new value to the t2 register.
b) The values in the stack are immediate and temporary, that means, these values are temporarily stored and can be immediately retrieved which will be used for calculations later on.
c)

```
sw t3, 8(sp)
li t4, 0xAB
sw t4, 12(sp)
li t5, 0x60
addi t5, t5, -8
andi t5, t5, 0xFF
sw t5, 16(sp)
li t6, 0x37
sw t6, 20(sp)
li t7, 0x99
sw t7, 24(sp)
lw t3, 8(sp)
andi t3, t3, 0xFF
slli t3, t3, 8
add t2, x0, t3
lw t4, 16(sp)
add t2, t2, t4
lw t6, 20(sp)
addi t6, t6, -7
andi t6, t6, 0x7F
lw t3, 0(sp)
xori t3, t3, 0x4
slli t3, t3, 16
add t2, t2, t3
sub a0, t1, t2
addi sp, sp, 28
ret
```

Figure 3: Instructions leading up to the creation of the final passcode

As seen here, the t registers are being in the stack via the sw instructions which is then loaded out using lw, then used for value manipulation and finally storing them in the t2 register.

Correct user_input

Before we can find the correct user_input, we would need to see how the value is validated. The last line of Figure 2, we can see that the beqz instruction is checking if the value of the a0 register is zero and if so, the secure exit will be executed. Now that we have checked the validation process, we would now have to check the instruction that modifies the value of a0, which is the instruction “sub a0, t1, t2” in Figure 3 in which the value in a0 will be equal to the user_input (t1) subtracted by the value in register t2.

We can now create an equation to solve for t1:

Step 1: Let a0 = 0
0 = t1 – t2

Step 2: Make t1 the subject by adding t2 on both sides (cancels)
t1 = t2

These two equations show that the value of t1 has to equal the value of t2 so that a0 can be zero.

To determine the final value for the t2 register, we would need to trace the lines that comes before the sub a0, t1, t2 instruction and as a result, the value of t2 would be 0x112358, thus, the user input needs to be 0x112358.

Questions

What transformations are applied to each part of the passcode before storage?

```
secret_operation:
        addi sp, sp, -28
        li t2, 0x10
        addi t2, t2, 1
        xori t2, t2, 0x4
        sw t2, 0(sp)
        li t3, 0xDE
        sw t3, 4(sp)
        li t3, 0x20
        addi t3, t3, 3
        ori t3, t3, 0x1
```

Figure 6 – Code snipping displaying data transformation before storing

The code contains a wide range of transformations to the parts of the passcode before storing them in the stack. This makes it more difficult to find out the final password directly.

For example the code in Figure 6, the instruction li t3, 0x20 which gives t3 is value of 0x20 but is then followed by the the addi instruction that adds t3 with 3 and stores the result in t3.

There are also other ways of transformations such as the Bitwise shifts used in the following line:

```
slli t3, t3, 16
```

Here, the instruction is used to shift values into another position during reconstruction.

How does the code handle dummy values, and what purpose might they serve?

```
sw t3, 4(sp)
...
li t7, 0x99
sw t7, 24(sp)
```

Figure 7 – Dummy Values

Dummy values and regular values are handled in the same way as the program still store them in the stack, however, the dummy values are not used to form the final passcode. Examples in Figure 7 where several registers are being stored with some data but is not used to calculate the final outcome.

Explain how the code reconstructs the passcode before comparing it with the user_input

The code goes through several sets of loading, shifting, masking and adding values and then finally stores the final passcode in the t2 register. The code then continues by subtracting the value of the t2 register with the user_input and store the result in the register a0.

As mentioned above, the code will check whether the result of that subtraction will be 0 or not. 0 means that the numbers are exactly the same and means that the passcode is correct and a secure_exit will be executed, if not, the program will end in an infinite loop.

Summary

In summary, the code supplied essentially tries to match the user_input with the passcode by reconstructing the passcode via several sets transformations and derivations and eventually comparing it with the given value in the t2 register and when compared, the process is done via subtraction by subtracting the user_input and the value of the t2 register which either results in 0 or a non-zero value; the former indicating that the user_input is indeed correct and ends the program gracefully and the latter indicates that the user_input is incorrect and goes into an infinite loop.