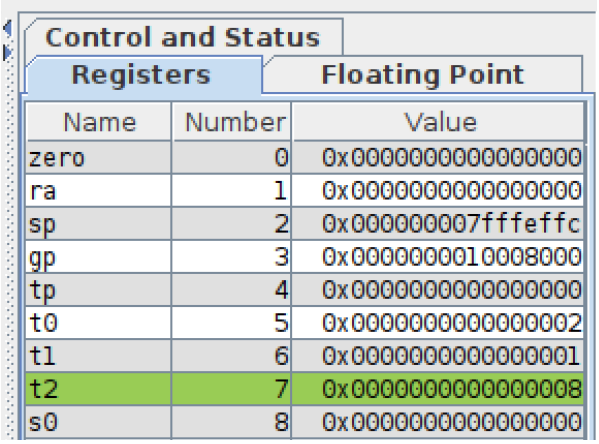
COMP30640: Operating Systems – Finbar Allan

Assignment 1

Question 1

$$Student\ ID: 22208108$$
$$Removing\ First\ Digit: 2208108$$
$$Corresponding\ Arithmetic\ Operations: 2 * 0 \div 1 + 8$$
$$Expected\ Solution: 8$$

Four digits are required to create the Student ID #22208108 and as RARS has a read-only zero value register we require three additional registers to load the remaining digits (2, 1 & 8). We use the load immediate operation (li) to register these digits in the t0, t1 and t2 registers, as displayed in Figure 1.



Name	Number	Value
zero	0	0x0000000000000000
ra	1	0x0000000000000000
sp	2	0x000000007fffeffc
gp	3	0x0000000010008000
tp	4	0x0000000000000000
t0	5	0x0000000000000002
t1	6	0x0000000000000001
t2	7	0x0000000000000008
s0	8	0x0000000000000000

Figure 1: Registers t0, t1, t2 and zero containing values 2, 1, 8 and 0 respectively.

As mentioned above, we begin evaluating the arithmetic expression by using the multiplication operator with the value held in the t0 register and the zero register as arguments. The division operator is then applied to the product of this multiplication with the value held in the t1 register and the addition operator is used with the value held in the t2 register to complete the evaluation of the above arithmetic expression.

t2	7	0x0000000000000008
s0	8	0x0000000000000000
s1	9	0x0000000000000000
a0	10	0x0000000000000000
a1	11	0x0000000000000000
a2	12	0x0000000000000000
a3	13	0x0000000000000000
a4	14	0x0000000000000000
a5	15	0x0000000000000000
a6	16	0x0000000000000000
a7	17	0x0000000000000000
s2	18	0x0000000000000000
s3	19	0x0000000000000008
s4	20	0x0000000000000000

Figure 2: Registers s1, s2 and s3 containing the evaluation of the arithmetic expressions. (note: both the multiplications and division instructions evaluate to zero)

We initiate a system call to print an integer by using the li operation to load a value of 1 into the a7 register, then move the evaluated arithmetic expression from register s2 to register a0 where it will be printed when the ecall operator is executed. Finally, the system call operation to terminate the program is initiated by using the li operation to load a value of 10 into the a7 register and executing the ecall command. Figure 3 displays the outputs of both system calls in the "Run I/O" dialog box below.

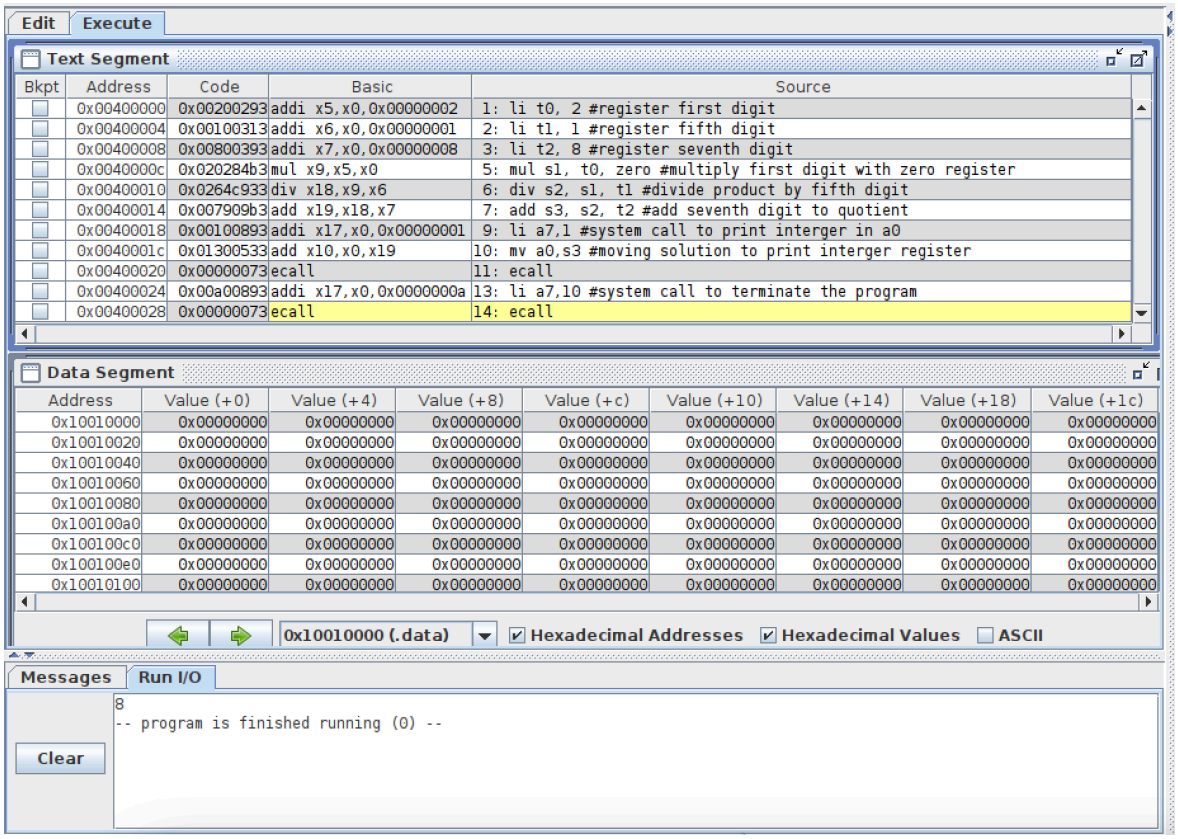


Figure 3: Outputs of the system calls for printing an integer and terminating the program.

With the Student ID value containing two zeros the multiplication and division sections of the arithmetic expression evaluate to zero. This leaves the addition operation as the only significant section of the arithmetic expression. To demonstrate the functionality of the program with non-zero values, Student ID has been updated to replace the zeros with a value of nine giving the following arithmetic expression:

$$\textit{Supplementary Student ID}: 22298198$$
$$\textit{Removing First Digit}: 2298198$$
$$\textit{Corresponding Arithmetic Operations}: 2 * 9 \div 1 \,\%\, 8$$
$$\textit{Expected Solution}: 2$$

The original program has been updated to assign a value of nine to the t1 register and the addition operator has been replaced with the remainder operator for the final step of evaluating the arithmetic expression. The updated registers and evaluations of these expressions are displayed in Figure 4 along with the outputs of the system calls in the "Run I/O" dialog box below. This assembly program has been included in the submission as a supplementary file saved as "22208108_a1q1_supplementary.asm".

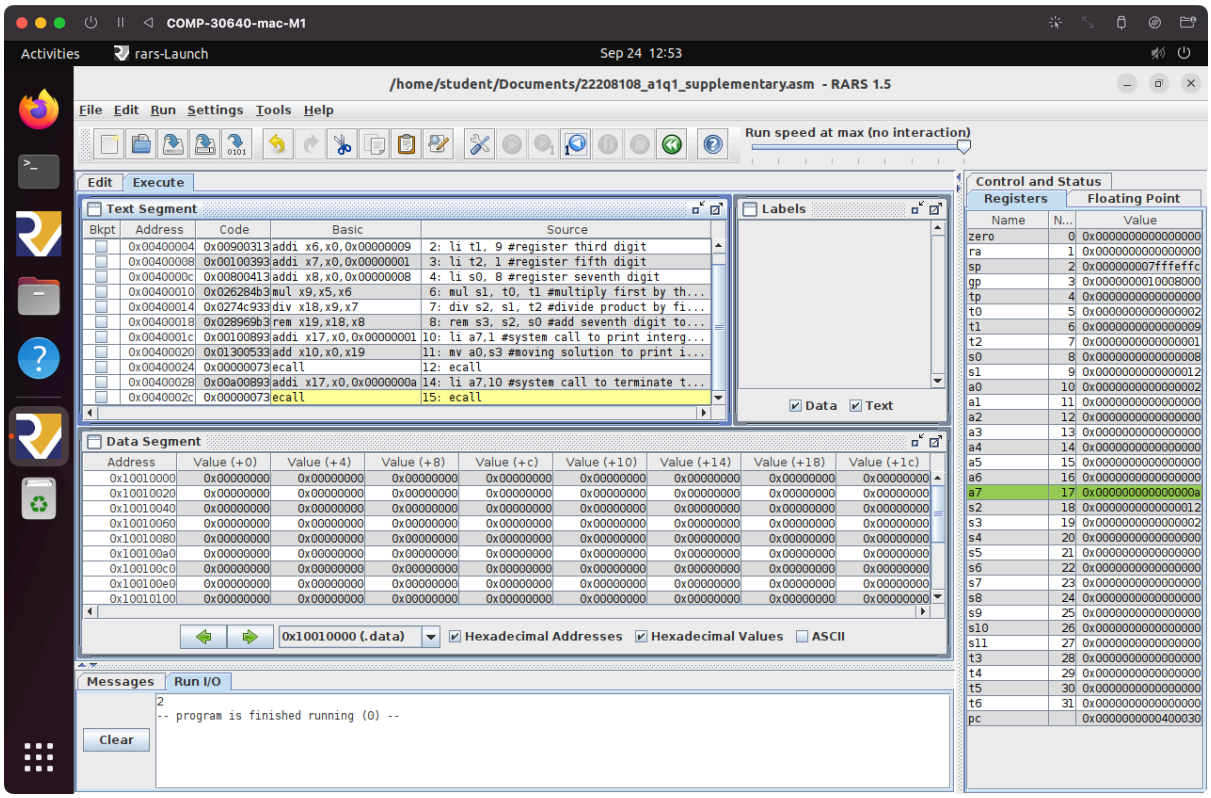


Figure 4: Registers and output screenshot of the supplementary submission.

Question 2

The Digital Lab Sim is initialised by loading the hexadecimal value of 0xFFFF0010 into the t0 register. It is necessary to determine the binary value which corresponds to the correct lighting sequence for each Student ID digit before displaying digits on the 7-segment display in RARS. Taking the number 2 as an example the corresponding binary value would be 1011011 which can be read in hexadecimal by converting from base 2 to base 16, giving a value of 5B. A similar process is applied to the remaining digits to determine the required hexadecimal values to display each on the 7-segment display. These values are shown loaded into the registers in Figure 5.

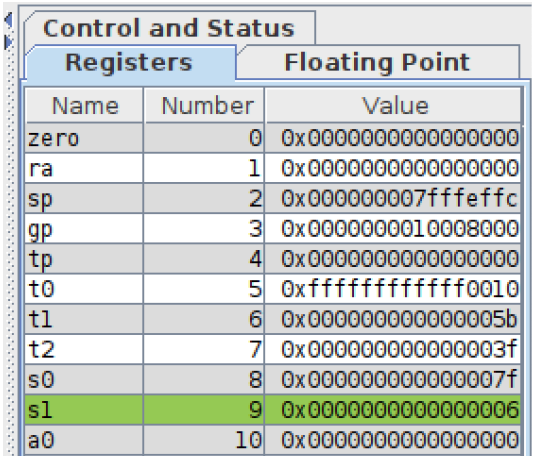


Figure 5: Registers containing hexadecimal values required for 7-segment display.

To display each digit the store byte operation is used to load the corresponding hexadecimal value into the t0 register which updates the 7-segment display with each step through the program showing all values of the Student ID successively. A few examples of these digits being displayed on the 7-segment screen are shown in Figure 6 which also contains a screenshot of the final state of the panel per the assignment instructions.

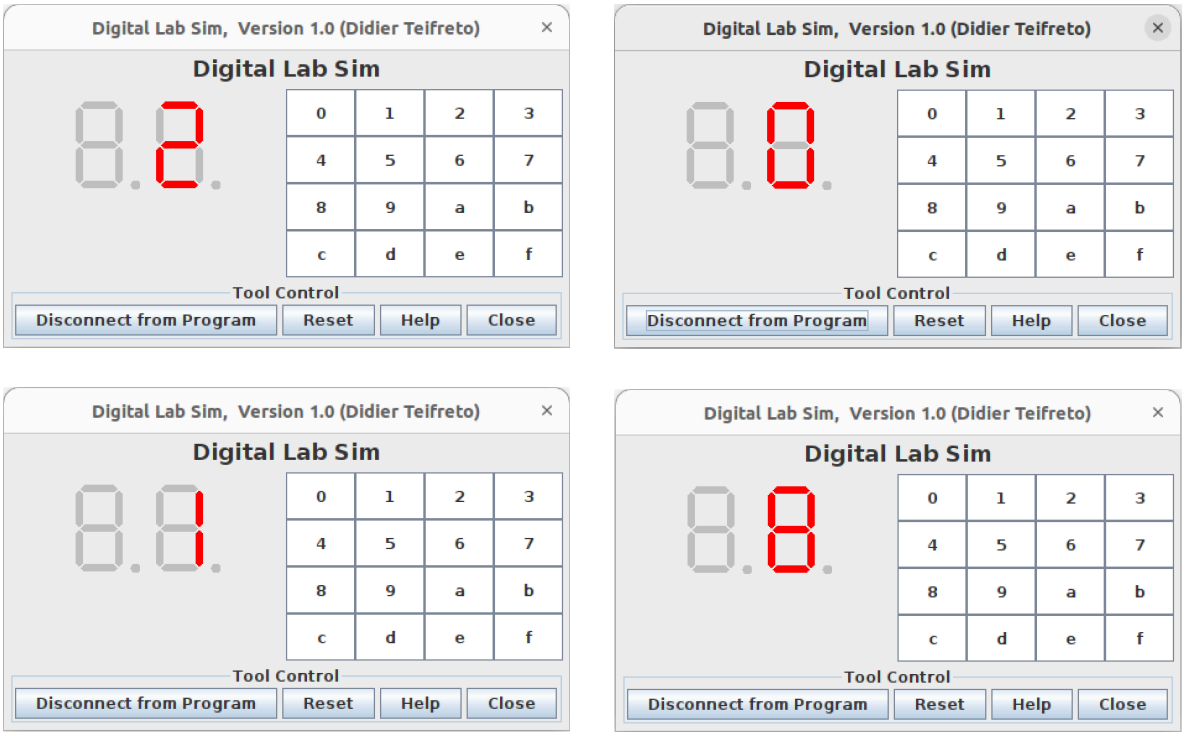


Figure 6: Final state of the 7-segment display panel with a few other examples

As the first three digits of the Student ID are repeated, the first three store byte operations are identical. This gives us as opportunity to introduce a loop which will avoid repeating lines of code at the expense of computational efficiency. As the looping approach is less efficient than the original approach the assembly file has been included in the submission as a supplementary file saved as “22208108_a1q2_supplementary.asm”.

The loop functions by loading a value of 3 into the t3 register then running the line of code required to display 2 on the 7-segment display before incrementally reducing the value in the t3 register while the value is not equal to zero, at which point the loop will be terminated.

Question 3

The linear search algorithm program functions by storing the integer values in memory as a list of “words” in the form of hexadecimal values. The first “word” is then loaded into the t0 register and will be treated as the minimum value until proven otherwise. With this value being loaded into the t0 register the list of remaining options for the minimum value has been reduced to 9, hence, this value is loaded into the t2 register for future use.

The loop begins with an instruction to advance the target by four bytes (as there are 32-bits per word and 8-bits per byte) and load the next word from the list into the t3 register. The key to modifying this program to find the maximum number in the list comes from the

branching instruction which acts as a comparison operator between the values in registers t1 and t3. The original program makes use of the blt operator to skip the assignment of the target value as the new minimum value if the condition evaluates false, to reverse this operation we can replace the blt operator with the bgt operator, as is the case in the revised program below. Alternatively we could swap the positions of the t1 and t3 registers without making any changes to the branching operator to have the program determine the maximum value.

The skip: section of the loop increments the t2 register and evaluates the exit condition until each of the "words" in the list have been evaluated to determine the minimum/maximum value. The program is then completed with two system calls to print the final value and terminate the program.

```
      .data
list: .word 10,3,2,4,6,10,6,5,-5,10
      .text
main: la t0, list # t0 points to the start of the list of numbers (32-bit words)
      lw t1, 0(t0) # load the first number and guess that it the minimal number
      li t2, 9 # 9 items in the list remain beside the first
loop: addi t0,t0,4 # advance t0 to point to the next number in the list
      lw t3,0(t0) # load the next number in the list
      bgt t1,t3,skip # change to branching operator gives maximum value
      mv t1,t3 # otherwise guess that t3 is the new minimal number.
skip: addi t2,t2,-1 # reduce the number of remaining numbers in the list by 1
      bne t2,zero,loop # keep looping while there are still numbers remaining
      li a7,1 # print the last chosen minimal number
      mv a0,t1
      ecall
      li a7,10 # terminate the program
      ecall
```