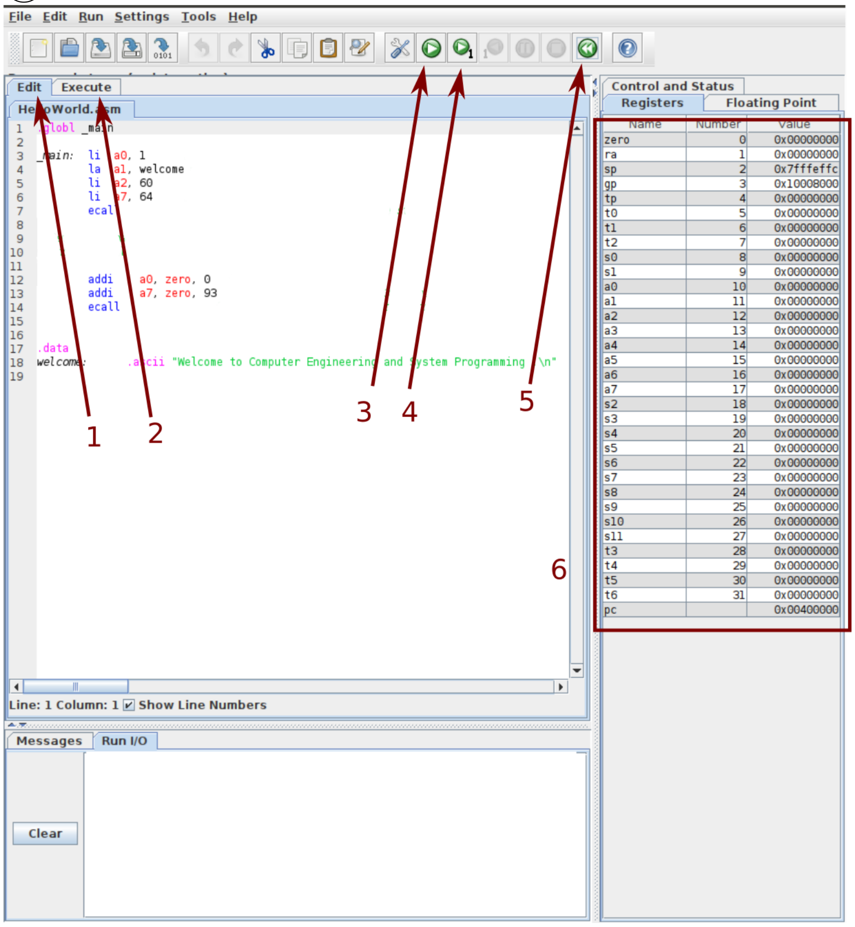
Exercise Sheet 2: Basic RISC-V Assembly Language Programming

Before starting call
cesp_setup ↵
Material for this exercise is in $HOME/cesp_course/exercises/02_basic_assembler

Task 1. Run your first RISC V program.

These are the control elements of RARS that you will need for your first program:

(1) Edit tab
(2) Execute tab
(3) Run button
(4) Run1 button
(5) Reset button
(6) Register values



```
.globl _main

_main:  li   a0, 1
        la   a1, welcome
        li   a2, 60
        li   a7, 64
        ecall



        addi    a0, zero, 0
        addi    a7, zero, 93
        ecall


.data
welcome:        .ascii "Welcome to Computer Engineering and System Programming  \n"
```

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400000

1. Open Rars instruction level simulator from the terminal:
   `rars`

2. Open `hello_world.asm`
3. Select Run→Assemble from the context menu.
4. Click Run1 button multiple times until a message appears in the Run I/O window ... Congrats ... You just execute your first RISC-V program.
5. Modify the program to print **Hello World**, then select Run→Assemble, press the Run button.
6. If the Run I/O window displays the **Hello World** message you are done!

Task 2. Unit Test of Simple Addition

1. Open RARS
   `rars` ↩
2. Extend the file to `simple_addition.asm` and follow the instructions in the file.
3. Throughout the course we will strongly use unit tests.
   A unit test executes a program and compares its results against predefined values.
   The tool `cesp_utest` will help you here. It executes the content in `.json` files.
4. Familiarize yourself with the unit test file
   `$HOME/cesp_course/exercises/01_get_tools_running/hello_world.json` ↩
   What it does:
   - Executon of asm file:
     `java -jar /home/cesp/cesp_course/tools/rars/rars.jar a0 a2 a7 hello_world.as`
     ↩
     This is the same thing you did before by pressing the *Run* button. The parameters a0 a2 a7 specify which register values to print after the execution.
   - Comparison of output of program against the value specified in the `expected_result` field of the `.json` file.
5. Create a unit test `simple_addition.json` file that checks for the correct result.
6. Run
   `cesp_utest` ↩

   Once your unit test succeeds you are done.

Function Calls

In mathematics, the Fibonacci numbers are a sequence where each number is the sum of the two preceding numbers (starting with two ones):

$$1, 1, 2, 3, 5, 8, 13, 21, 34, ... \qquad (1)$$

Task 3. Iterative implementation of Fibonacci Sequence

Implement a function in RISC-V assembly language that calculates the Fibonacci numbers in an iterative fashion and run it in the RARS simulator. Follow the steps in Fib01.asm from the course website that helps you to implement the functionality defined by the following C code.
Please use comments to describe the functionality of your assembler code.

```c
void fib(int* result, int n){
    if (n > 0)
        result[0]=1;
    if (n > 1)
        result[1]=1;
    for(int j=2; j<n; j++){
        result[j] = result[j-2] + result[j-1];
    }
}
int main(){
    int fibonacci[20];
    fib(fibonacci, 20);
    for(int i=0; i<20; i++){
        printf("%d\n", fibonacci[i]);
    }
}
```

Use the file `fib01.json` to verify your result using the `cesp_utest` command.

Task 4. Recursive implementation of Fibonacci Sequence

Implement a function in RISC-V Assembler that calculates the Fibonacci sequence in a recursive fashion and run it in the RARS simulator. Follow the steps in fib02.asm that help you to implement the functionality defined by the following C code.
Please use comments to describe the functionality of your Assembler code.
Hint: Use the Stack.

```c
int fib(int n) {
    if (n <= 1)
        return n;
    return fib(n-1) + fib(n-2);
}
int main() {
    printf(" %d ", fib(20)));
}
```

Use the file `fib02.json` to verify your result using the `cesp_utest` command.