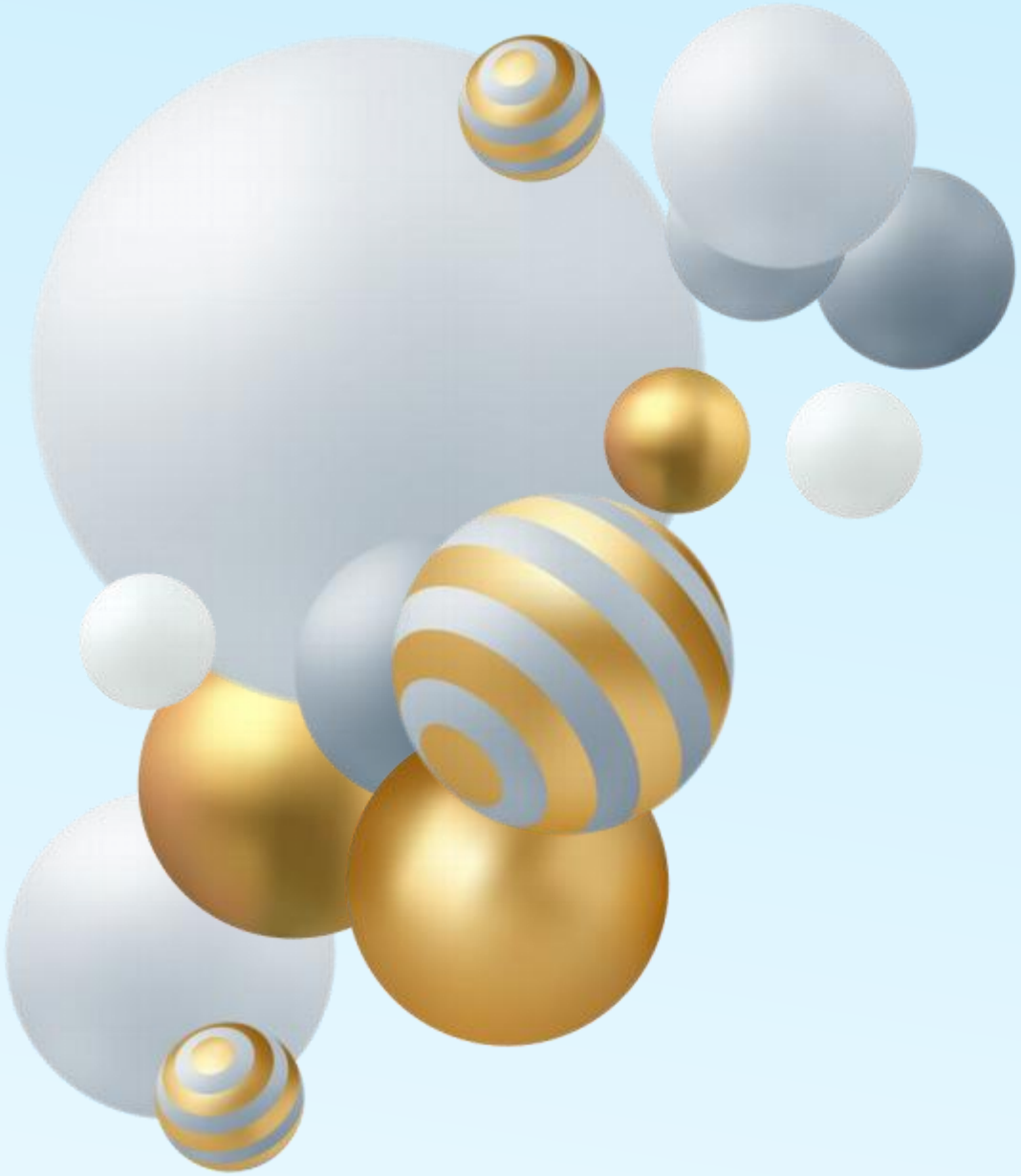
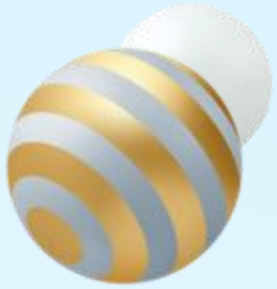


CS214

Lab1:Introduction

Special purpose circuit and General purpose processor

Tool kits



Computer Organization 2023s lab,
wangw6@sustech.edu.cn

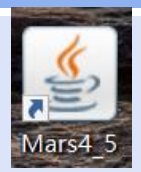
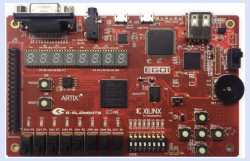
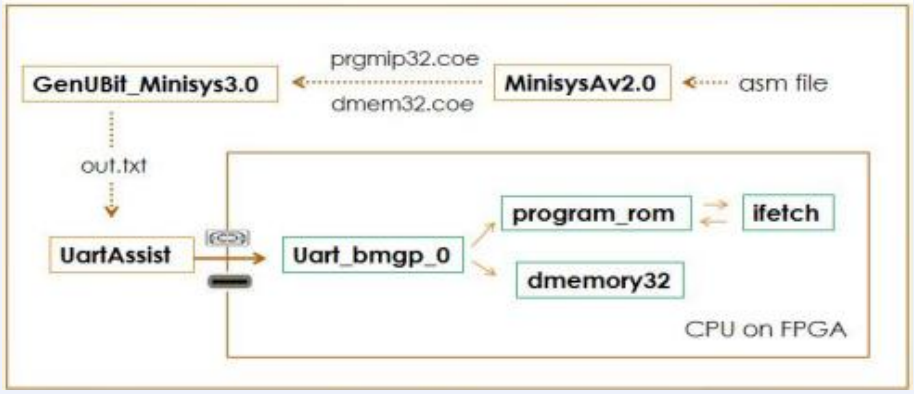
Assignments Submission Rules

- All assignments **MUST be submitted to BlackBoard site or OJ**, any other forms of submission are NOT accepted.
- If the submission is delayed **for one day, 20% discount** on the total score. If it is **delayed for more than a week, any submission is NOT ACCEPTABLE**! This assignment is 0 point.
- In the case of plagiarism: at the 1st time, the assignment was 0 for all concerned students and at the 2nd time, the grade of the experimental course is 0 for all concerned students.
- Reminder:
  - Assignment scoring and the score publication would be completed within two weeks after the publication of assignment. **If you have any question about the score, please email the relevant reviewer in one week after the score publication.**

Contents of Lab1

- **Experimental Tool kits**
- **Special purpose circuit** vs **General purpose processor**
  - practice 1-1,1-2,1-3
- **IDE on MIPS : Mars**
  - practice 2-1 ,2-2
- **Optional : Qtspim** installation and usage

Experimental Tool Kits

Task	Tool kits
Learn and practice **MIPS** (a type of Assemblly language)	➢ **Mars** / QtSpim  
Compare the **Risc-V** and MIPS	➢ **rars_27a7c1f** , Mars 
Design and implement an **CPU**	➢ **Vivado**  ➢ **FPGA based Development Board** **EGO1**  **or Minisys** 
Test the CPU with MIPS	➢ **Assembler** ➢ **Uart Tools** ➢ **Vivado** ➢ **FPGA based Development Board** 

Practice1-1 : Lighting the led by 'Special purpose circuit'

Design file + Constraint file

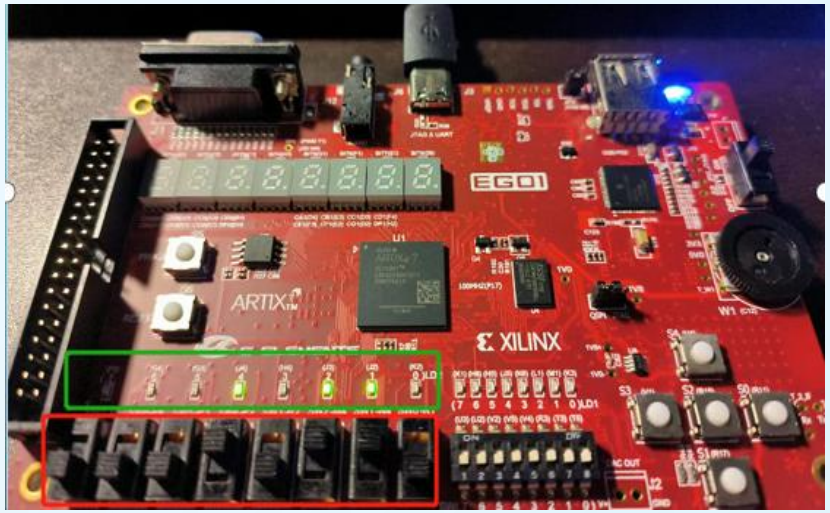
----(Vivado generate bitstream)---> **bitstream** file

---(Vivado hw manager, connect, program device)

---> the circuit is implemented on the **FPGA chip**

---> **Test** the circuit on the **FPGA based Development Board**

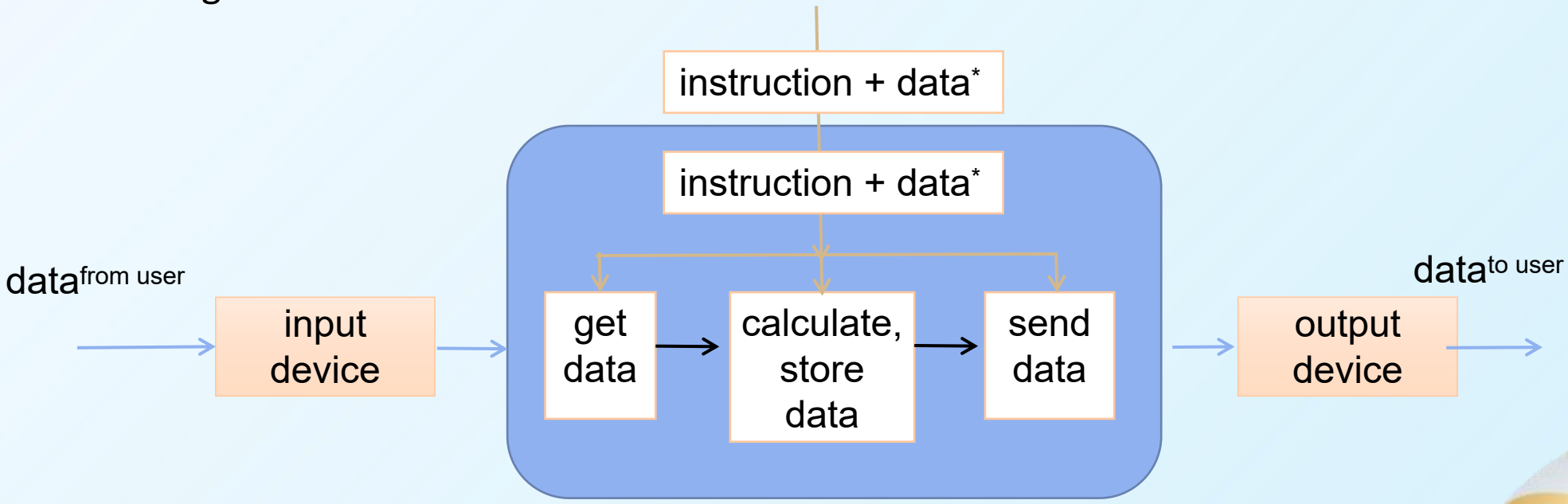
```
//Design file by verilog, save as sw2led.v
module sw2led(sw,led);
        input [ :0] sw;
        output [ :0] led;

        assign led = sw;
endmodule
```




```
#Constraint file, save as sw2led.xci
set_property IOSTANDARD LVCMOS33 [get_ports {led[ ]}]
...
set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[ ]}]
...
set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN F6 [get_ports {led[ ]}]
...
set_property PACKAGE_PIN K2 [get_ports {led[0]}]
set_property PACKAGE_PIN P5 [get_ports {sw[ ]}]
...
set_property PACKAGE_PIN R1 [get_ports {sw[0]}]
```

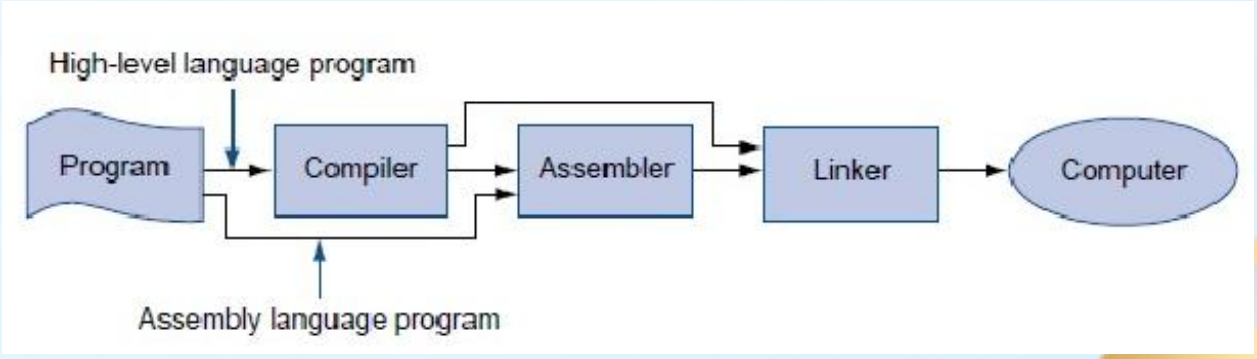
The ‘General purpose processor’

- General purpose processor
  - Process (get, calculation, storage, send) specified data according to instructions
  - Program storage and execution: store the program 1st, execution 2nd
  - Program = instruction + data*



Programming

- Programming： Analysis + Design + **Decription** + Debug and Test
- **Decription**： High level language vs Low level language
  - High level language：
    - Don’t consider the hardware(**Python**, **Java**)
    - Focus on hardware but not much(**C**)
  - **Low level** language:
    - Closely related to hardware (**MIPS**, **RISC-V**)



Assembly Program Structure

data declarations + program code

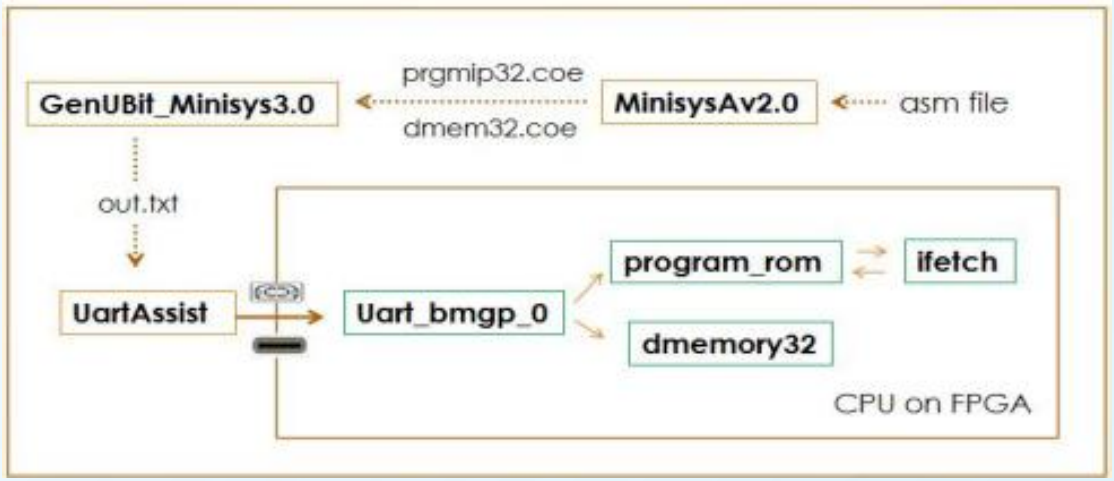
- **part1: Data Declarations**
  - placed in section of program identified with assembler directive(汇编说明/汇编器指示符): **.data**
  - **declare** variable names used in program; storage allocated in main **memory** (RAM)
- **part2: Program Code**
  - placed in section of text identified with assembler directive: **.text**
  - contains program code (instructions)
  - starting point of code e.g. ecution given label main:
  - ending point of code should use exit system call (see below in “System Calls” part)
- **part3: Comments (suggested)**
  - anything following # on a line
    # This stuff would be consideredd as comment

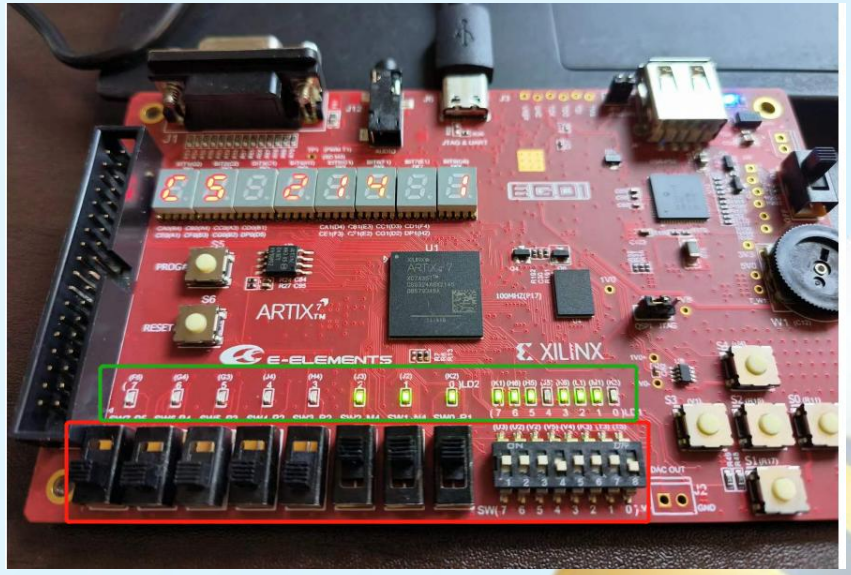
```
# Comment giving name of program and description of function
# Template.s
# Bare-bones outline of MIPS assembly language program

            .data       # variable declarations follow this line
                        # ...

            .text       # instructions follow this line

main:                   # indicates start of code (first instruction to execute)
                        # ...

# End of program, leave a blank line afterwards to make SPIM happy
```

Practice1-2: lighting the led by ‘General purpose processor’

A ‘tailored General purpose processor’ (a bitstream file), the instructions and data* (in hexadecimal, a file named ‘out.txt’) would be given, you are supposed to ligting the led on this system by following steps:

- Let the given ‘tailored General purpose processor’ work on the FPGA
- Send the instructions and data* (‘out.txt’) to the ‘tailored General purpose processor’, the ‘tailored General purpose processor’ receive the instructions and data*
- Run the ‘instructions and data*’ on the ‘tailored General purpose processor’
- Test





Practice1-2(1) - The 'tailored General purpose processor'

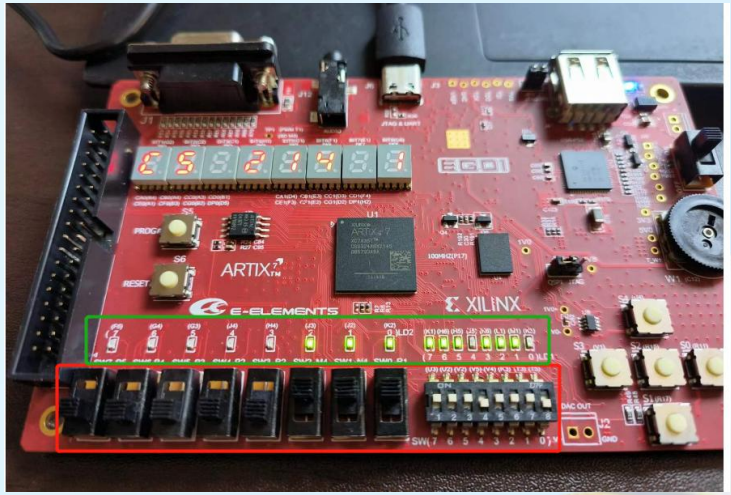
- ✓ **Two mode**
  - ✓ **Uart communication mode**: **get** the instructions and the data* from **uart** port.
  - ✓ **CPU work mode**: **process** data according instructions.
- ✓ **Data flow**
  - ✓ the data **MUST be stored into the register(s)** of 'tailored General purpose processor' before be calculated or be sent to the output device.
- ✓ **Register(s)**
  - ✓ There are total 32 registers, the width of each register is 32bits.
  - ✓ Only **$1** is writable
  - ✓ The value in **$31** is 0xFFFF_F0000, the initial value in other registers is 0.
- ✓ **I/O and address**
  - ✓ The unique input device with address: dial switch
    - ✓ **each 8 bit share an address**
    - ✓ 2 addresses for input: 0x**FFFF_FC70**, 0x**FFFF_FC72**
  - ✓ The unique output device with address: led
    - ✓ **each 8 bit share an address**
    - ✓ 2 addresses for output: 0x**FFFF_FC60**, 0x**FFFF_FC62**

Practice1-2(2) - The the instructions and data*

✓ **The instruction(s)** which could be understood by the 'tailored General purpose processor'

```
lw a,b          #move data from b to a, a MUST be register
sw a,b          #move data from a to b, a MUST be register
j  lablex       #jump to the instruction labled by lablex
```

```
#assmbly source file
.data 0x0000                    # the bitwidth of data* is 32, its initial value is 0
      buf :  .word  0x0000

.text 0x0000                    # instructions
start:
      lw   $1,0xC70($31)        #move the data from 0xFFFF_FC70 to register $1
      sw   $1,0xC60($31)        #move the data from register $1 to 0xFFFF_FC60

      lw   $1,0xC72($31)
      sw   $1,0xC62($31)
      j start                   # jump to the instructions labled by start
```

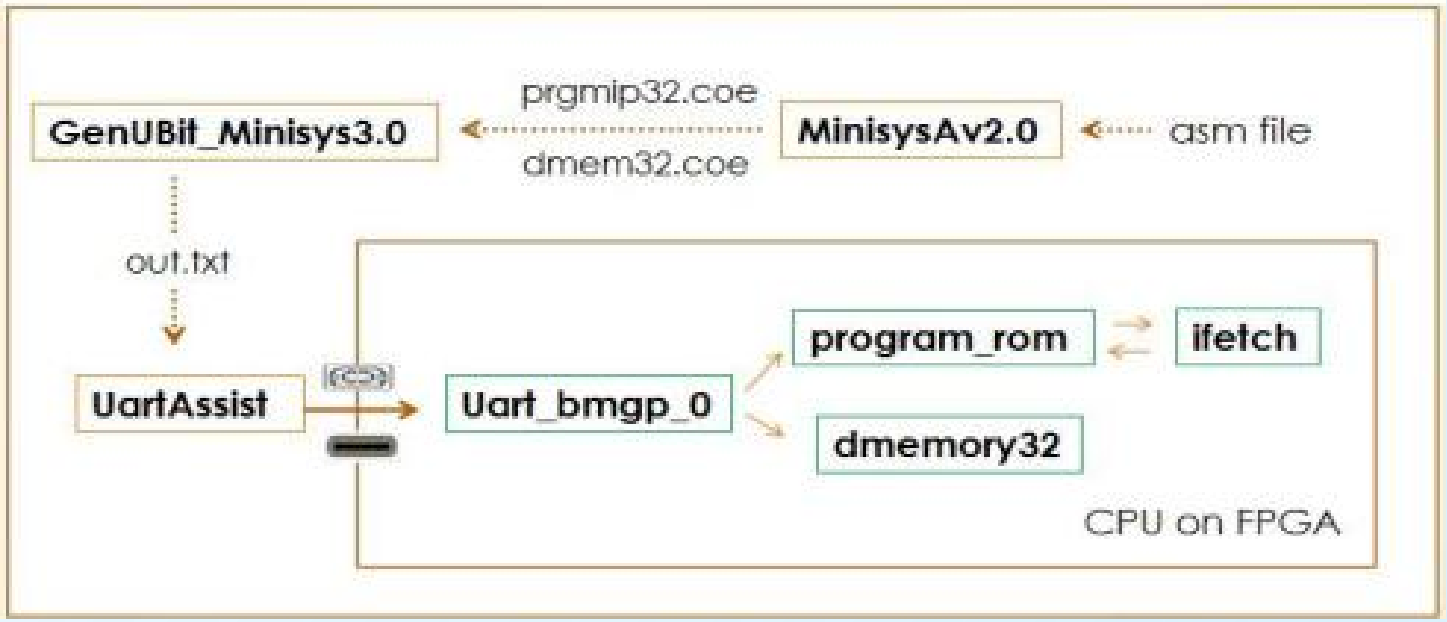


Practice1-2(3) - Test:lighting the led by ‘General purpose processor’

Preparation:

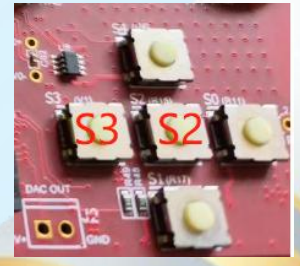
1. **Build** the assmbly source file, **assembler** it with ‘**MinisysAv2.0**’ to get the coe file(s), Using ‘**GenUbit_Minisys3.0**’ to **merger** two coe files into ‘out.txt’.

2. **Write the FPGA chip with the bitstream** file of the ‘tailored General purpose processor’.



Test on the board:

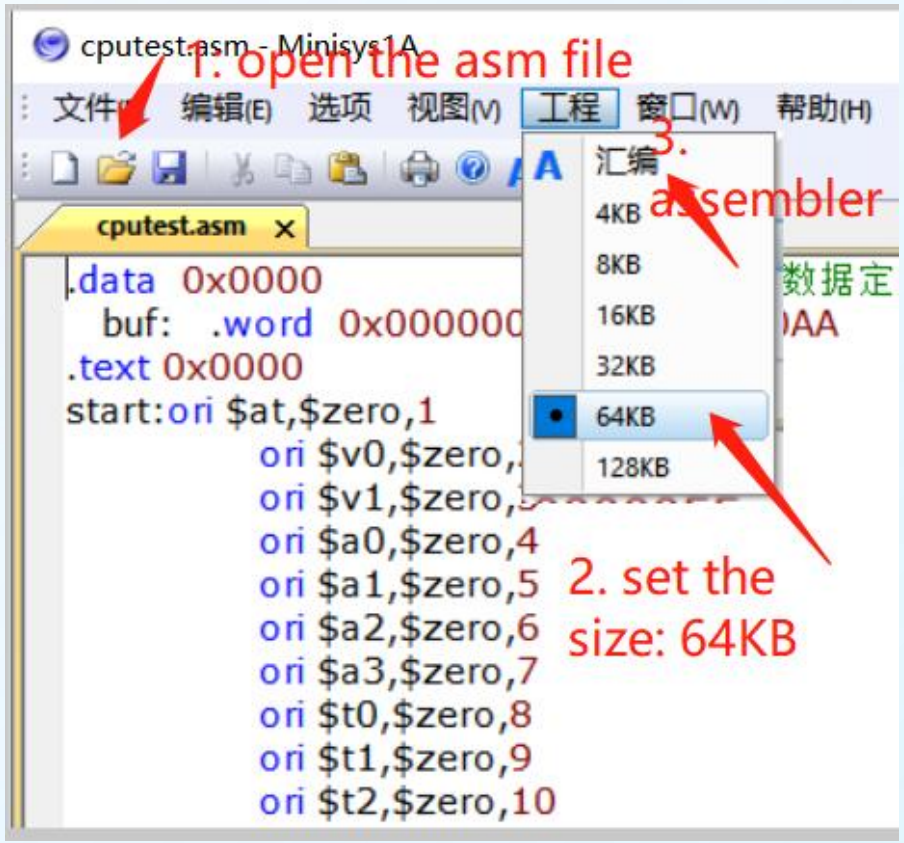
- step1: Bounce after pressing the button **S3**(on EGO1) to prepare for receiving the instructions from uart.
- step2: Send the instrusctions(in file ‘out.txt’) to the ‘tailored General purpose processor’ by ‘UartAssist’
- step3: Bounce after pressing the button **S2**(on EGO1) to make the ‘tailored General purpose processor’ work follwing the instructions which is get from the step2.
- step4: turn on/ off the Dial switchs, what’s the state of the leds?



TIPS (1) generate the 'out.txt'

Step1:

While using **MinisysAv2.0** to assembler the **asm** file to generate the **coe** files, follw the follwing steps:



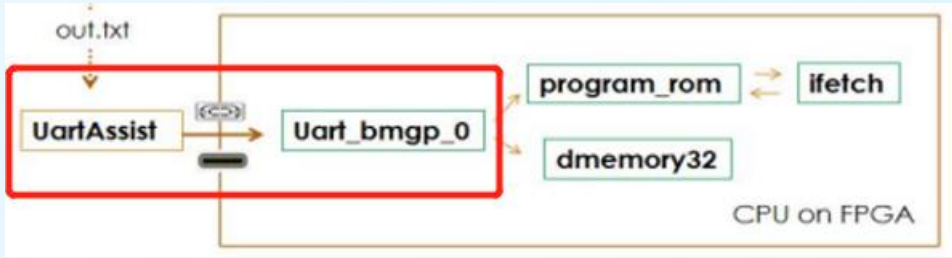
Step2:

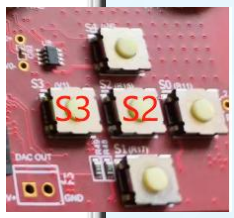
To generate the **out.txt**, the two coe files(prgmips32.coe and dmem32.coe) **MUST** be with the same direcotry as 'GenUBit_Minisys3.0' and 'UARTCoe_v3.0'.

Double click '**GenUBit_Minisys3.0**', or run it in the command line, the 'out.txt' would be generated in the same directory. The instructions and data are merged into the 'out.txt'.

TIPS (2) 'out.txt' and two modes

While using **UartAssit** to send the file to the FPGA embeded board, follow the following settings and steps:





Test on the board:

- step1: Bounce after pressing the button **S3**(on EGO1) to let the 'tailored General purpose processor' work as **Uart communication mode**.
- **step2: send the instrusctions(in file 'out.txt') to the 'tailored General purpose processor' by 'UartAssist' (串口调试助手)**

NOTE:

1. Before using 'UartAssit'(串口调试助手), the 'tailored General purpose processor' work as Uart communication mode.

2. Only 'UartAssit'(串口调试助手) receive the feedback from the 'tailored General purpose processor' and show Program done! means the 'tailored General purpose processor' get the file, if not, it's suggested to send the file again.

- step3: Bounce after pressing the button **S2**(on EGO1) to make the 'tailored General purpose processor' work on **CPU work mode** so as to execute the instructions and process data.
- ...

Practice1-3(1) ‘special purpose circuit’ vs ‘General purpose processor’

There are two group dial switchs for input(**iG1**,**iG2**) and two group led for output(**oG1**, **oG2**).

Do NOT re-program the device(write the bitstream file to the FPGA chip)

Do the following test on ‘tailored General purpose processor’ :

1. Creat a new asm file, to get the data from **iG1** and show the data on **oG2**, to get the data from **iG2** and show the data on **oG1**.

2. Assembler the new asm file with ‘MinisysAv2.0’ to get the coe file(s), Using ‘GenUbit_Minisys3.0’ to merg two coe files into ‘out.txt’.

3. Repeate the 4 steps on the last page(update the instructions in the ‘tailored General purpose processor’ by ‘UartAssist’ and make it work, do the test with different input data)

4. What’s the address of iG1, iG2, oG1 and oG2?

Practice1-3(2) ‘special purpose circuit’ vs ‘General purpose processor’

- To implement the same logical relationship between inputs and outputs in this test(get the data from **iG1** and show the data on **oG2**, to get the data from **iG2** and show the data on **oG1)**, which of the following processes are needed on practice 1-1(Lighting the led by ‘Special purpose circuit’ on page 6)?

  1) update the design source file 2)update the constraint source file 3) update the testbench file

  4) regenerate the bitstream file 5) re-program the device with the new bitstream file

  6) using a new FPGA chip to be programmed 7) reset the chip type in the vivado project

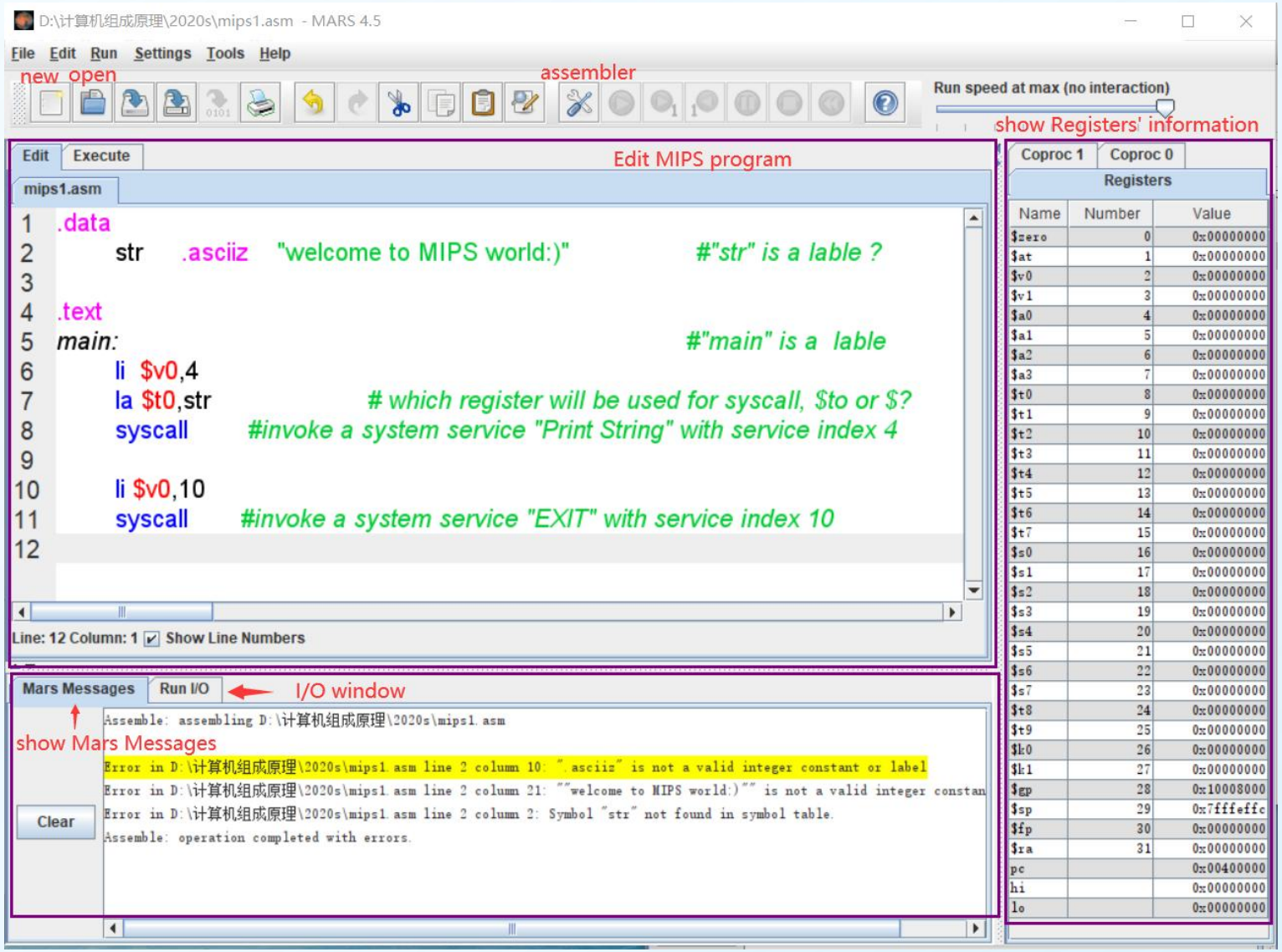
A. 1,2,3,4,5,6,7 **B**. 6 **C**. 7 **D**. 1,2,3,4,5 **E**. 1,2,4,5 **F**. 3,4,5 **G**. 1,4,5 **H**. 2,4,5 **I**. 3,4,5 **J**. 1,4,5,6,7

Experimental Tool Kits （Simulator： Mars）

Task	Tool kits
Learn and practice **MIPS**	➢ **Mars** / QtSpim
Compare the **Risc-V** and MIPS	➢ **rars_27a7c1f** , Mars
Design and implement an **CPU**	➢ **Vivado** ➢ **FPGA based Development Board: EGO1 , Minisys**
Test the CPU with MIPS	➢ **Assembler** ➢ **Uart Tools** ➢ **Vivado** ➢ **FPGA based Development Board**

Mars(1)

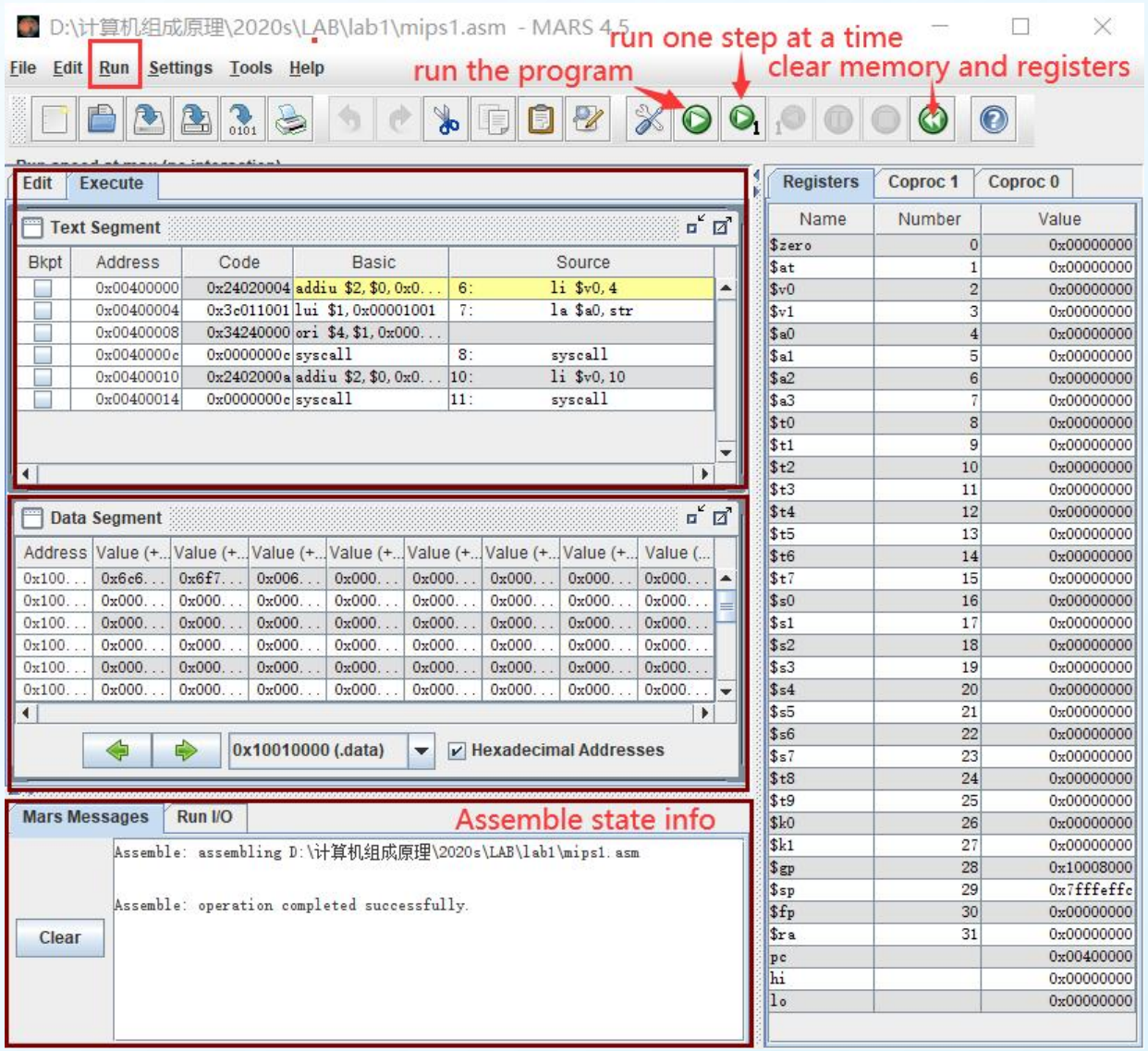
http://courses.missouristate.edu/kenvollmar/mars/download.htm

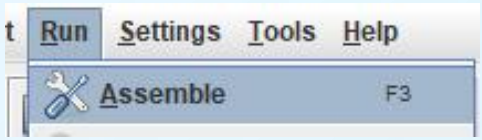


MARS is a **lightweight** interactive development environment (**IDE**) for programming in **MIPS** assembly language, intended for educational-level use with Patterson and Hennessy's Computer Organization and Design.

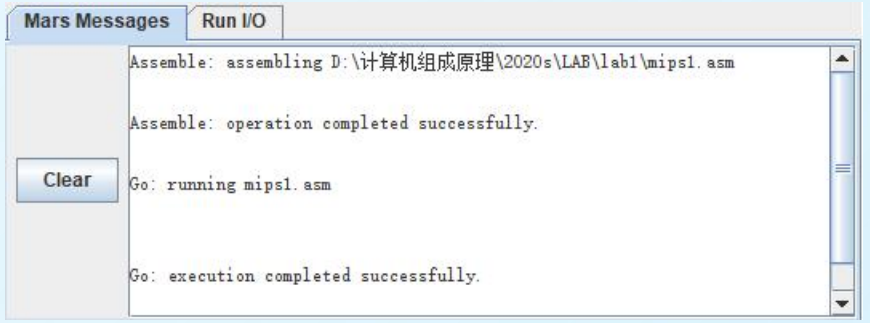
MARS requires Java J2SE 1.5 (or later) SDK installed on your computer.

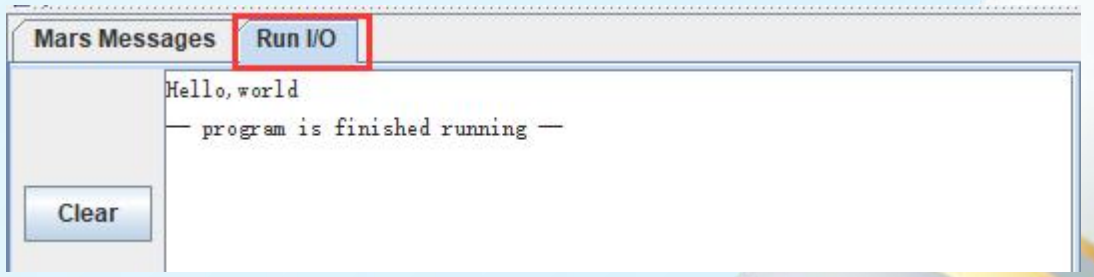
Mars(2)


D:\计算机组成原理\2020s\LAB\lab1\mips1.asm - MARS 4.5
File Edit Run Settings Tools Help
run the program
run one step at a time
clear memory and registers
Edit Execute
Text Segment
Bkpt Address Code Basic Source
0x00400000 0x24020004 addiu $2, $0, 0x0... 6: li $v0, 4
0x00400004 0x3c011001 lui $1, 0x00001001 7: la $a0, str
0x00400008 0x34240000 ori $4, $1, 0x000...
0x0040000c 0x0000000c syscall 8: syscall
0x00400010 0x2402000a addiu $2, $0, 0x0... 10: li $v0, 10
0x00400014 0x0000000c syscall 11: syscall
Data Segment
Address Value (+...
0x10010000 (.data)
Hexadecimal Addresses
Mars Messages
Run I/O
Assemble state info
Assemble: assembling D:\计算机组成原理\2020s\LAB\lab1\mips1.asm
Assemble: operation completed successfully.
Clear
Registers
Coproc 1
Coproc 0
Name Number Value
$zero 0 0x00000000
$at 1 0x00000000
$v0 2 0x00000000
$v1 3 0x00000000
$a0 4 0x00000000
$a1 5 0x00000000
$a2 6 0x00000000
$a3 7 0x00000000
$t0 8 0x00000000
$t1 9 0x00000000
$t2 10 0x00000000
$t3 11 0x00000000
$t4 12 0x00000000
$t5 13 0x00000000
$t6 14 0x00000000
$t7 15 0x00000000
$s0 16 0x00000000
$s1 17 0x00000000
$s2 18 0x00000000
$s3 19 0x00000000
$s4 20 0x00000000
$s5 21 0x00000000
$s6 22 0x00000000
$s7 23 0x00000000
$t8 24 0x00000000
$t9 25 0x00000000
$k0 26 0x00000000
$k1 27 0x00000000
$gp 28 0x10008000
$sp 29 0x7fffeffc
$fp 30 0x00000000
$ra 31 0x00000000
pc 0x00400000
hi 0x00000000
lo 0x00000000


Run Settings Tools Help
Assemble F3

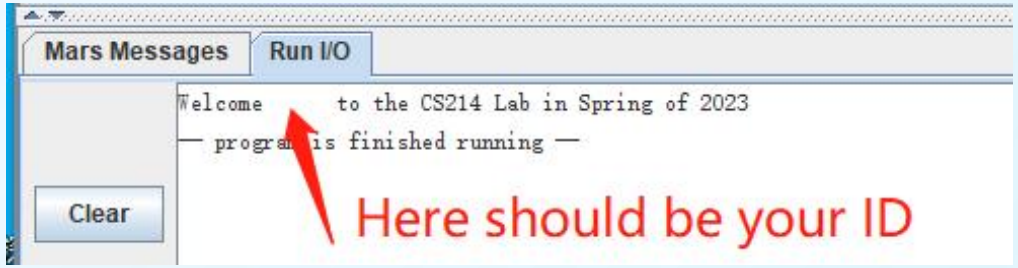

Run Settings Tools Help
Assemble F3
Go F5


Mars Messages
Run I/O
Assemble: assembling D:\计算机组成原理\2020s\LAB\lab1\mips1.asm
Assemble: operation completed successfully.
Go: running mips1.asm
Go: execution completed successfully.
Clear


Mars Messages
Run I/O
Hello, world
-- program is finished running --
Clear

Practice 2-1 & 2-2

- 2-1. **Install Mars on your PC** and **refer to the 'help' page** to implement the follwing task:
  - 1. What's the name and ID of the **registers** used while invoke the **"print string" syscall**?
  - 2. Creat a asm file and test it on the Mars: To get the student ID from the keyboard, print the string "Welcome xxx to the CS214 Lab in Spring of 2023" on the "Run I/O"of Mars (xxx is your student ID).



  - 2-2. Practice and answer the following questions:
    - 1. Could the code works on the Mars be successfully assembled on '**MinisysAv2.0**' ?
    - 2. While running the code of practice 1-2 and 1-3 on Mars, would them work as same on the 'tailored General purpose processor' as on Mars?

会报错，缺少符号

Optional:System Calls

- **System Calls**
  - **SPIM** provides a small set of **operating-system-like services** through the system call (**syscall**) instruction.
  - To request a service, a program **loads the system call code into register $v0** and **arguments into registers $a0~$a3** (or $f12 for floating-point values).
  - **System calls that return values put their results in register $v0** (or $f0 for floating-point results).

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $v0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $v0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $v0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $v0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	

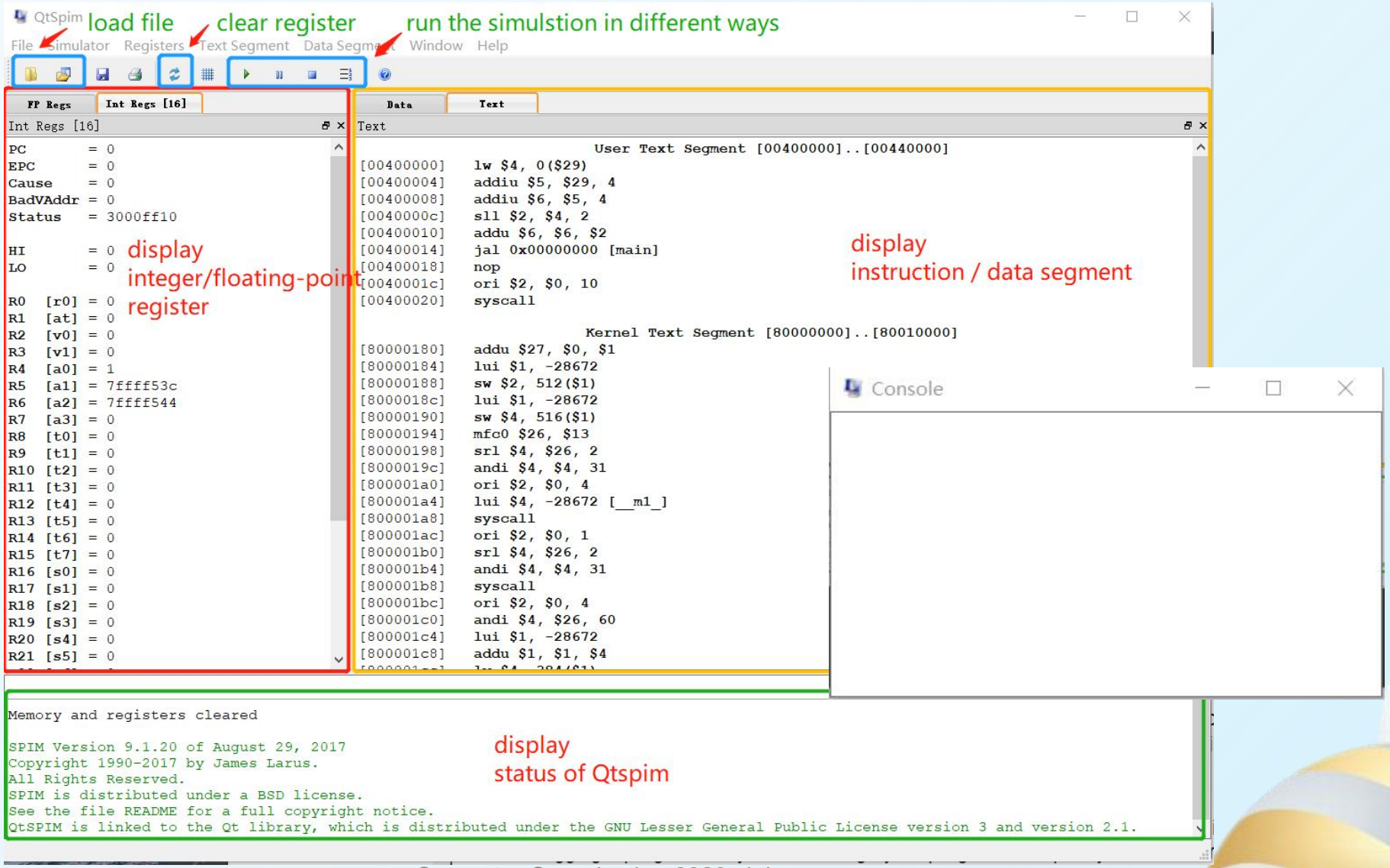
Optional: QtSpim（1）

http://spimsimulator.sourceforge.net/

- QtSpim
  - **QtSpim is the newest version of Spim that currently being actively maintained, runs on Microsoft Windows, Mac OS X, and Linux.**
    - Spim, A self-contained simulator that runs MIPS32 program developed by Prof. James Larus from University of Wisconsin-Madison
  - **QtSpim reads and executes programs written in assembly language（汇编语言） for a MIPS computer. QtSpim does not execute binary (compiled) programs.** To simplify programming, QtSpim provides a simple debugger and small set of operating system services
  - **QtSpim implements MOST of the MIPS32 assembler-extended instruction set. (**It omits the floating point comparisons and rounding modes and the memory system page tables.**)** which means that QtSpim will **NOT run programs for ALL MIPS processors.**

Name	Modified	Size
QtSpim_9.1.23_Windows.msi	2021-12-06	21.5 MB
qtspim_9.1.23_linux64.deb	2021-12-06	18.5 MB
QtSpim_9.1.23_mac.mpkg.zip	2021-12-06	22.2 MB

Optional: QtSpim (2)



Optional: QtSpim (3)

Load file and Do the **simulation**

Step1: loading a file (assembly code file usually have the extension "**.s**")

- Open file : **File->Load File** or **File ->Reinitialize and Load File**
- Tips:
  - **Reinitialize and Load File** : **Clears** all the changes made by a program, deleting all of its instructions, then **reload** the last file
  - **Before doing the simulation, make sure clear registers**, you can do it by :

    Simulator->Clear Registers or by related tools.

Step2:

- **Run** file : **Simulator->Run / Continue**
- The value of register in left window will be refreshed, the program's output will appear in the Console window.