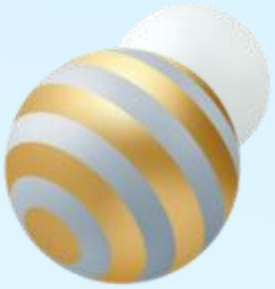
Computer Organization

Lab1:Introduction

Special purpose circuit VS General purpose processor

Tool kits



Assignments Submission Rules

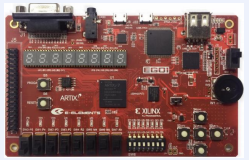
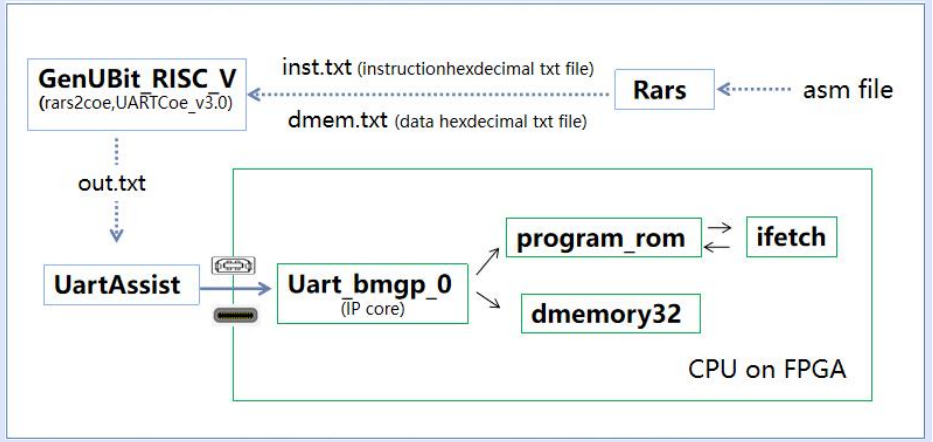
- All assignments **MUST be submitted to BlackBoard site or OJ before the DDL. Any form of delayed submission is not accepted, and the score for delayed submission assignments is 0**.
- In the case of plagiarism: at the 1st time, the assignment was 0 for all concerned students and at the 2nd time, the 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 DDL of assignment. **If you have any question about the score, please email the relevant reviewer in one week after the score publication.**

Topic

- **Experimental Objectives and Tool kits**

- **Special purpose circuit** vs **General purpose processor**
  - **Practice 1-1 (vivado, verilog and EGO1)**
    - ✓ related to Digital Design(fundamental of Computer Organization)
  - **Practice 1-2,1-3 (Rars, Uart-tools, vivado and EGO1)**
    - ✓ related to Computer Organization
    - ✓ practice **RISC-V**(a very basic part), **a tailed CPU**(provided) and **Tool kits**(easy to use) in Computer Organization

Experimental Objectives and Toolkits

Task	Tool kits
Learn and practice **RISC-V**(a type of Assemblly language)	➢ **Rars (rars_27a7c1f)** 
Design and implement an **CPU**	➢ **Vivado**  ➢ **FPGA based Development Board EGO1** 
Test the **CPU** with **program(s)**, both of which are based on **RISC-V**	➢ **Assembler (Rars)** ➢ **Uart Tools (GenUBit_RISC_V, UartAssist)** ➢ **Vivado** ➢ **FPGA based Development Board(EGO1)** 

Practice1-1 : Lighting led(s) by 'Special purpose circuit'

Design file(in verilog) + Constraint file

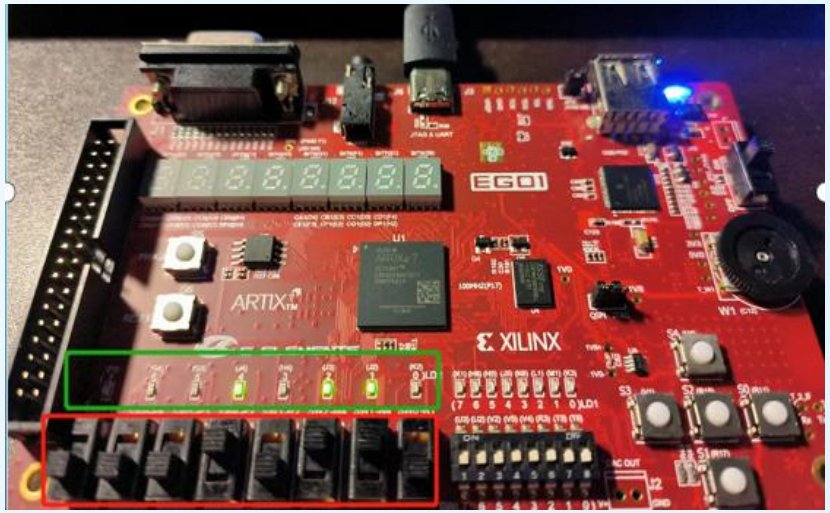
----(Vivado generate bitstream)---> **bitstream** file

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

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

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

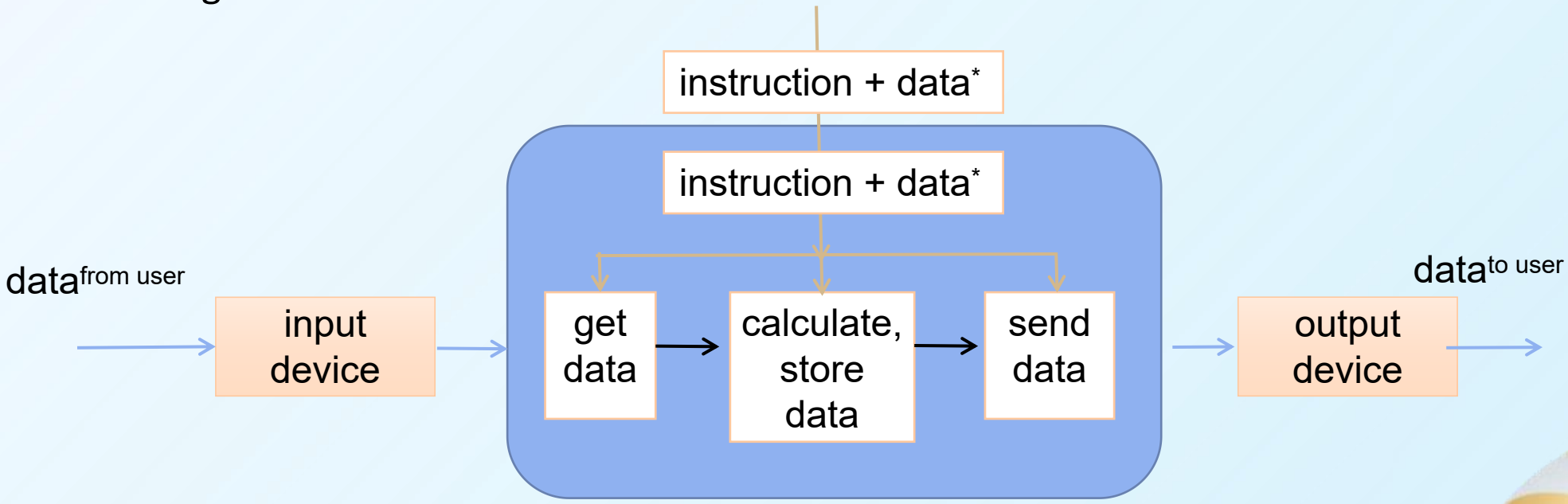
```
//Design file by verilog, save as sw2led.v
module sw2led(sw,led);
        input    [ TBD :0] sw;
        output [ TBD  :0] led;

        assign led = sw;
endmodule
```




```
#Constraint file, save as sw2led.xdc
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'

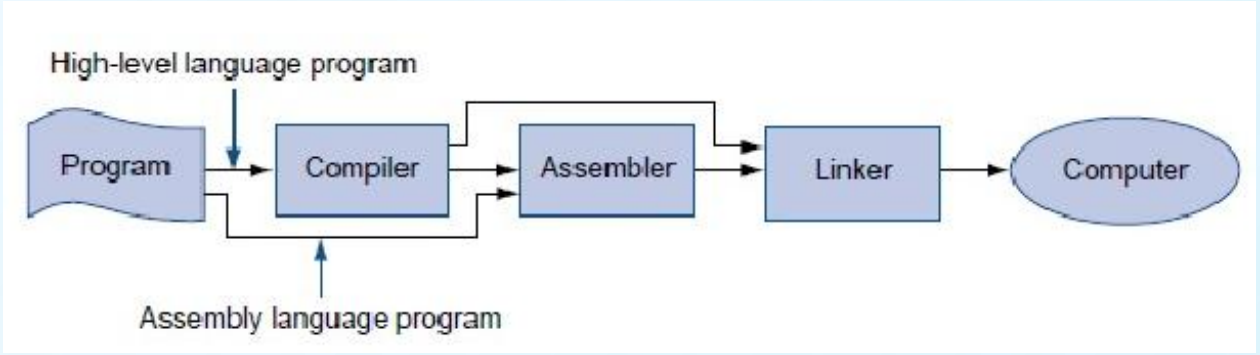
- General purpose processor
  - Process (get, calculation, storage, send) specified data according to instructions
  - Program storage and execution: store the program 1st, execution 2nd
  - Program = instruction + data*



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 (**Assembly language, such as RISC-V, MIPS**)

Assembly Program Structure

Data declarations + Program code + Comments(optional but suggested)

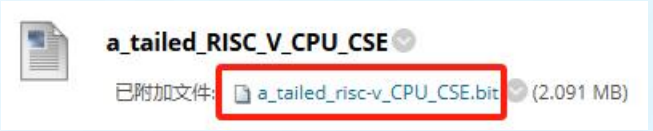
- **Part1: Data Declarations**
  - placed in section of program identified with assembler directive(汇编说明/汇编器指示符): **.data**
  - **declare** variable names used in program; storage allocated in main **memory** (RAM)
- **Part2: Program Code**
  - placed in section of text identified with assembler directive: **.text**
  - contains program code (instructions)
  - starting point of code, e.g. ecution given label main.
- **Part3: Comments (suggested)**
  - anything following # on a line
    # This stuff would be consideredd as comment

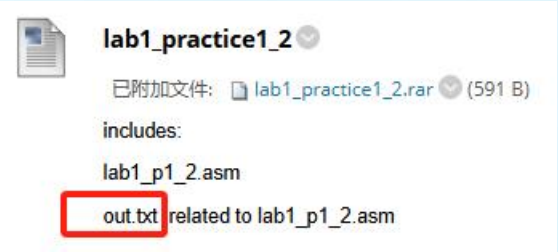
```
.data
        # here data* is 1byte, its initial value is 8'b0000_0001
        buf :  .byte  0x01

.text                   # instructions
main:
        lw  x1,  0(x0)
        sw x1,  4(x31)

        # jump to the instructions labled by main
        j main
```

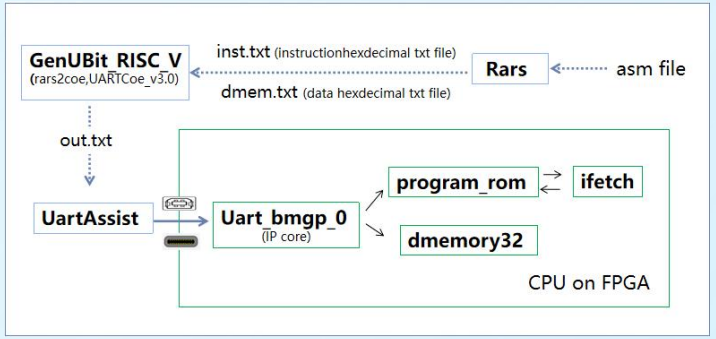
Practice1-2: lighting led(s) by ‘General purpose processor’

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:

- Step1. Let the given ‘tailored General purpose processor’ work on the FPGA chip.
- Step2. Make the ‘tailored General purpose processor’ ready to receive the program.
- Step3. Send the program(includs instructions and data* ,saved in file ‘out.txt’) to the ‘tailored General purpose processor’.
- Step4. Test the programe(‘instructions and data*’) on the ‘tailored General purpose processor’.



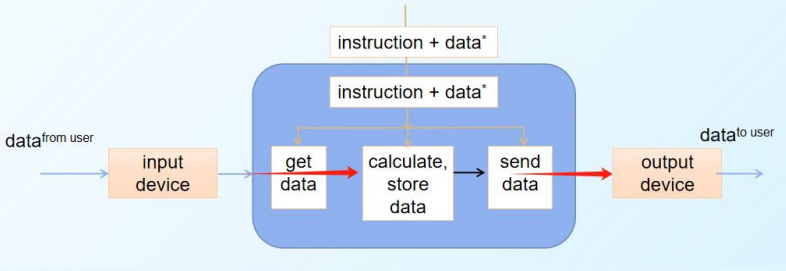




Practice1-2(1) - The 'tailored General purpose processor'

- ✓ **Two mode**
  - ✓ **Uart communication mode**: **get** the program(instructions and the data*) from **uart** port.
  - ✓ **CPU work mode**: **process data** according the program.

- ✓ **Register(s)**
  - ✓ There are total 32 registers, the width of each register is 32bits.
  - ✓ ONLY **x1** is writable.
  - ✓ The value in **x31** is **0x0000_FFC0**, the initial value of other registers is **0**.



- ✓ **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.

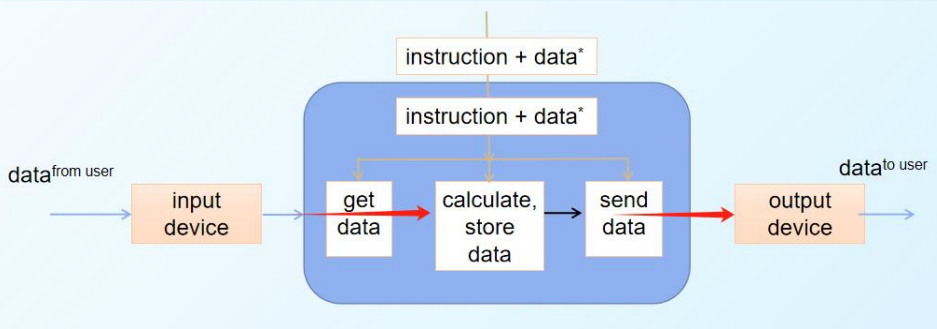
- ✓ **I/O and address**
  - ✓ get data from input(addressed by 0x**FFC8)**
    - ✓ **read from 0XFFC8 is to get the data from 8 switches**
  - ✓ send data to output(addressed by 0x**FFC4)**
    - ✓ **write to 0xFFC4 is to send the data to 8 leds**

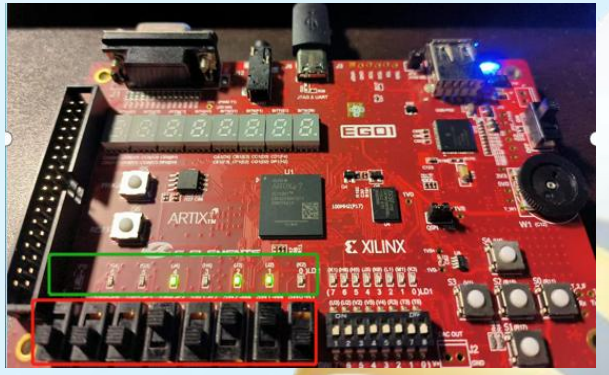
Practice1-2(2) - The the instructions and data*

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

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

```
#assmbly source file  in RISC-V
.data                # data* is 4bytes, its initial value is 0
        buf: .word 0x0000

.text                # instructions
main:
      #in the   'tailored General purpose processor', the value in register x31 is 0xFFC0.

        lw   x1, 8(x31)      #copy data from 0xFFC8(switches) to register x1
        sw  x1, 4(x31)       #copy data from register x1 to 0xFFC4(leds)

        j main               # jump to the instructions labled by main
```





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

Preparation-part1:

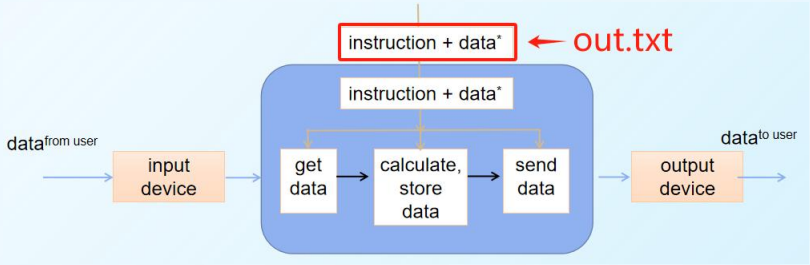
1. **Build** the assmbly source file. [Using **Rars**]

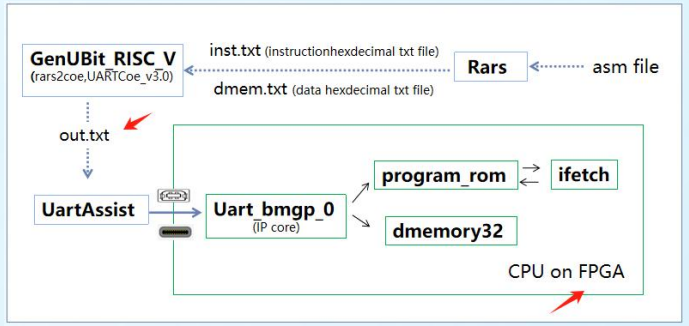
2. **Assembler** it to generater the machine code, then dump to file(s). [Using **Rars**]

3. **Generate** out.txt [Using **GenUbit_RISC_V]**

   3-1. **Change** the machine code file(s) to the coe file(s). [Using txt2coe]

   3-2. **Merge** two coe files into ‘out.txt’. [Using UARTCoe_v3.0]





Preparation-part2: Using vivado and EGO1(FPGA chip embeded)

Using vivado’s “**Program device” to write the bitstream** file of the ‘tailored General purpose processor’ **to the FPGA chip of EGO1**.

TIPS (1) generate the out.txt(1)

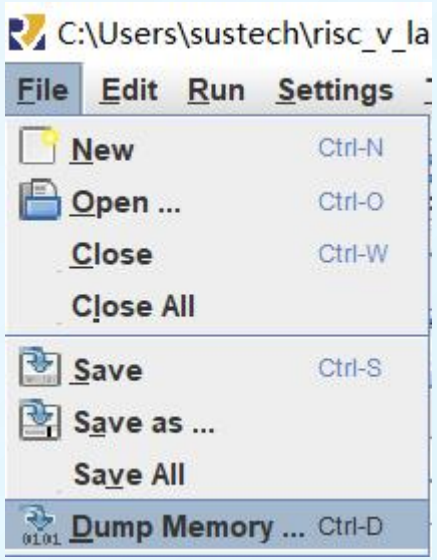
Step1: generate the machine code file(s) in **Rars**

1-1. Open the asm file

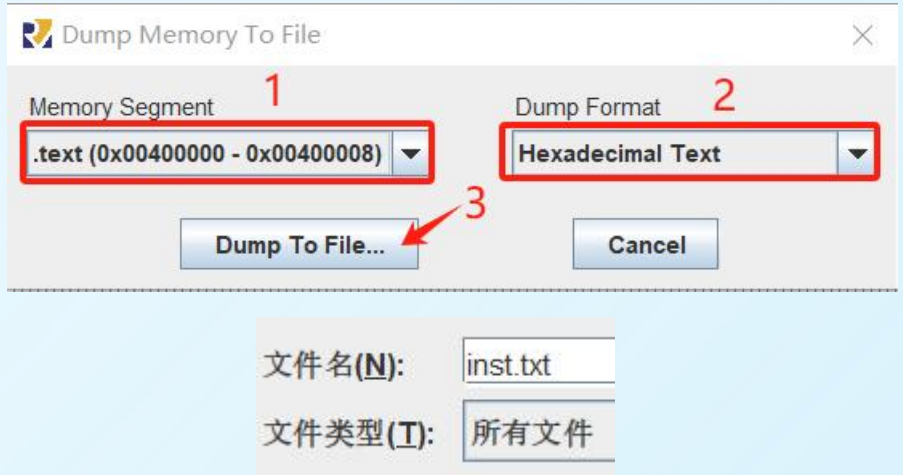
1-2. Assemble the asm file to the machine code

1-3. Dump machine code and data in avaliable format.

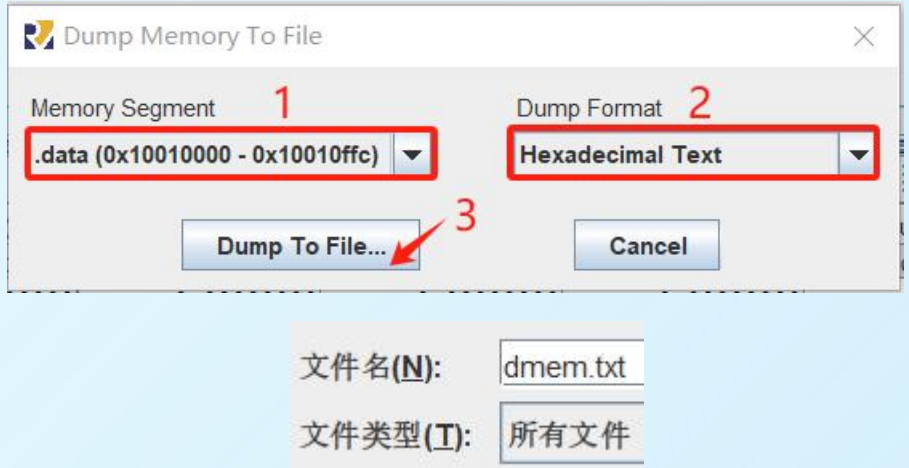
NOTE:
The instruction dump file is suggested to be named as "**inst.txt**", while the data dump file is suggested to be named as "**dmem.txt**"



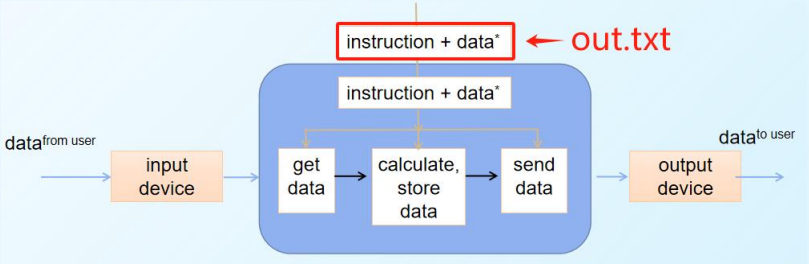
1-3-1. Dump **instructions** in **Hexadecimal Text**



1-3-2. Dump **data** in **Hexadecimal Text**



TIPS (2)　generate the out.txt(2)



Step2: generate the **out.txt** file.

NOTE:

2-1. To generate the **out.txt**, the two txt files(inst.txt and dmem.txt) **MUST** be with the same direcotry as '**GenUBit_RISC_V**' , '**rars2coe**' and '**UARTCoe_v3.0**'.

```
C:\Users\sustech\Downloads\uart_tools>ls
GenUBit_RISC_V.bat  UARTCoe_v3.0.exe  UartAssist.exe  dmem.txt  inst.txt  rars2coe.exe
```

2-2. Double click 'GenUBit_RISC_V', or run it in the command line, two coe files(prgmip32.coe and dmem32.coe) and the '**out.txt**' would be generated in the same directory.

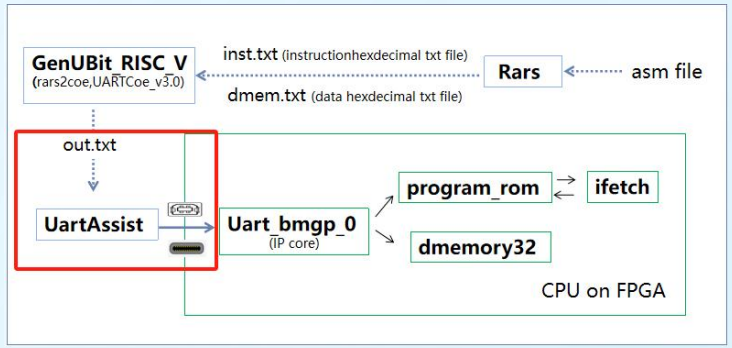
```
C:\Users\sustech\Downloads\uart_tools>ls
GenUBit_RISC_V.bat  UARTCoe_v3.0.exe  UartAssist.exe  dmem.txt  inst.txt  rars2coe.exe

C:\Users\sustech\Downloads\uart_tools>GenUBit_RISC_V.bat
2 files are read successfully
Hexadecimal file(s) detected.
Done.

C:\Users\sustech\Downloads\uart_tools>ls
GenUBit_RISC_V.bat  UartAssist.exe  dmem32.coe  out.txt      rars2coe.exe
UARTCoe_v3.0.exe    dmem.txt        inst.txt    prgmip32.coe
```

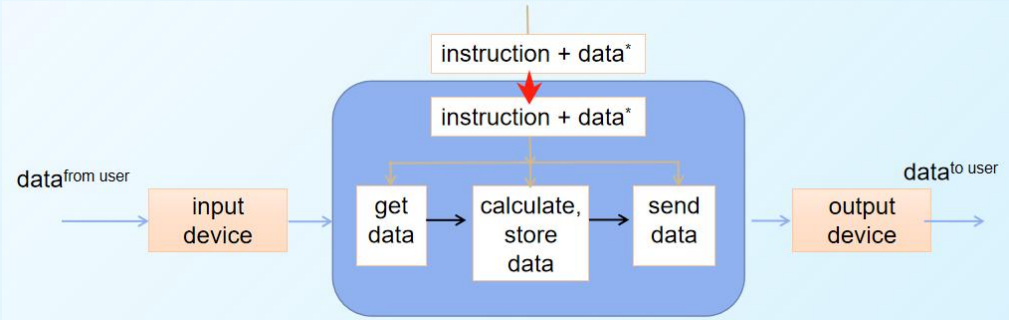
The instructions and data* are merged into the '**out.txt**'.

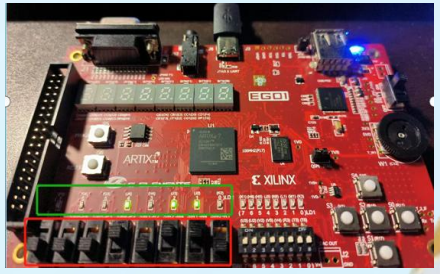
Practice1-2(4) - Test:lighting led(s) by ‘General purpose processor’

Test on the board:

NOTE: make sure the ‘tailde General purpose processor’ has been written to the FPGA chip of EGO1 before the following steps.







➢step1: Bounce after pressing the button **S6**(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: While the ‘UartAssist’ receive the data from the ‘tailored General purpose processor’, and show “0x00020000 bytes read. Program done! “, which indicates that the ‘tailored General purpose processor’ has successfully received the ‘instruction + data*’, and start the processing process based on them.

➢step4: turn on/ off the Dial switchs, what’s the state of the leds?

TIPS. Using ‘UartAssit’(串口调试助手) to send ‘out.txt’

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

Test on the board:

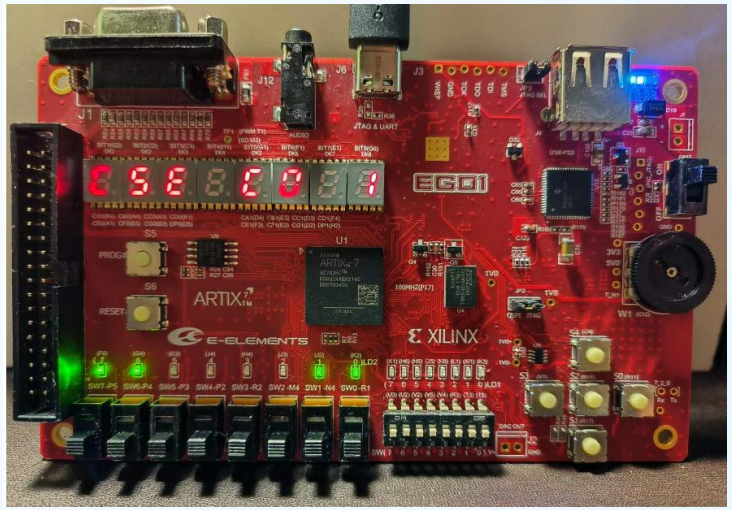
step-1. Before using ‘UartAssit’(串口调试助手), the ‘tailored General purpose processor’ has ‘worked’ on the FPGA chip of EGO1.

step-2. Press the button “S6” on the EGO1 board.

step-3. Open ‘UartAssit’(串口调试助手), do the setting firstly, then click on “打开” to connect with Uart port of EGO1, then choose and send the ‘out.txt’ file.

step-4. 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’ receive the file successfully, if not, it’s suggested to return to step-2 and try again.

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



Do NOT re-program the device(write the bitstream file to the FPGA chip), **just change the asm file**(there are 4 options for selection) to keep the state of the led remains at 8‘b1100_0011 no matter how the dial switch changes.

- Assembler the new asm file with ‘Rars’, then dump the ‘inst.txt’ and ‘dmem.txt’, Using ‘GenUbit_RISC_V’ to generate the ‘out.txt’.
- Repeate the steps on the last page(update the instructions and data in the ‘tailored General purpose processor’ by ‘UartAssist’ and make it work, do the test)

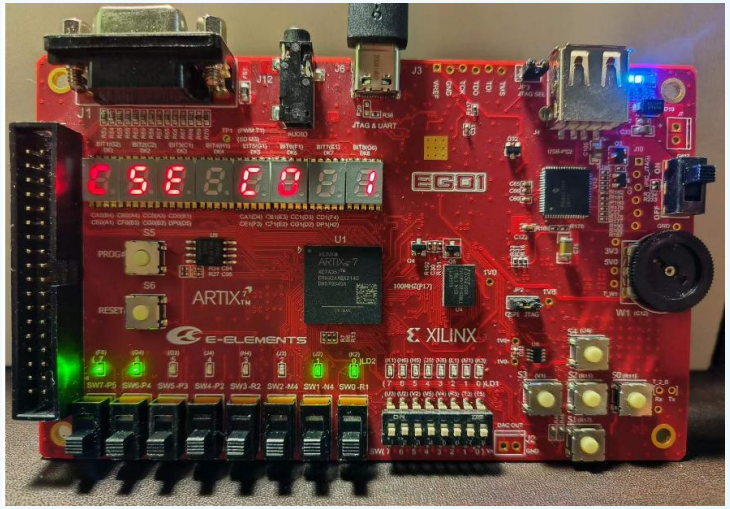
```
.data                    #A
        buf: .word 0x0000

.text
start:
        lw   x1,8(x31)
        sw  x1,4(x31)

        j start
```

```
.data                    #B
        buf: .word 0x0000

.text
start:
        sw   x31,4(x31)
        j start
```

```
.data                    #C
        buf: .byte 0xC3

.text
start:
        lw   x1, 0(x0)
        sw  x1, 4(x31)

        j start
```

```
.data                    #D
        buf: .byte 0xC3

.text
start:
        lw   x1,0(x31)
        sw  x1,4(x31)

        j start
```

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



- To implement the same logical relationship between inputs and outputs in this test(keep the state of the led remains at 8’b1100_0011 no matter how the dial switch changes), which of the following processes are needed on practice 1-1(Lighting the led by ‘Special purpose circuit’ on page 5)?

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 （Assembler/Simulator： Rars）

Task	Tool kits
Learn and practice **RISC-V**(a type of Assemblly language)	➢ **Rars (rars_27a7c1f)**
Design and implement an **CPU**	➢ **Vivado** ➢ **FPGA based Development Board EGO1**
Test the CPU with **RISC-V**	➢ **Assembler** ➢ **Uart Tools** ➢ **Vivado** ➢ **FPGA based Development Board**

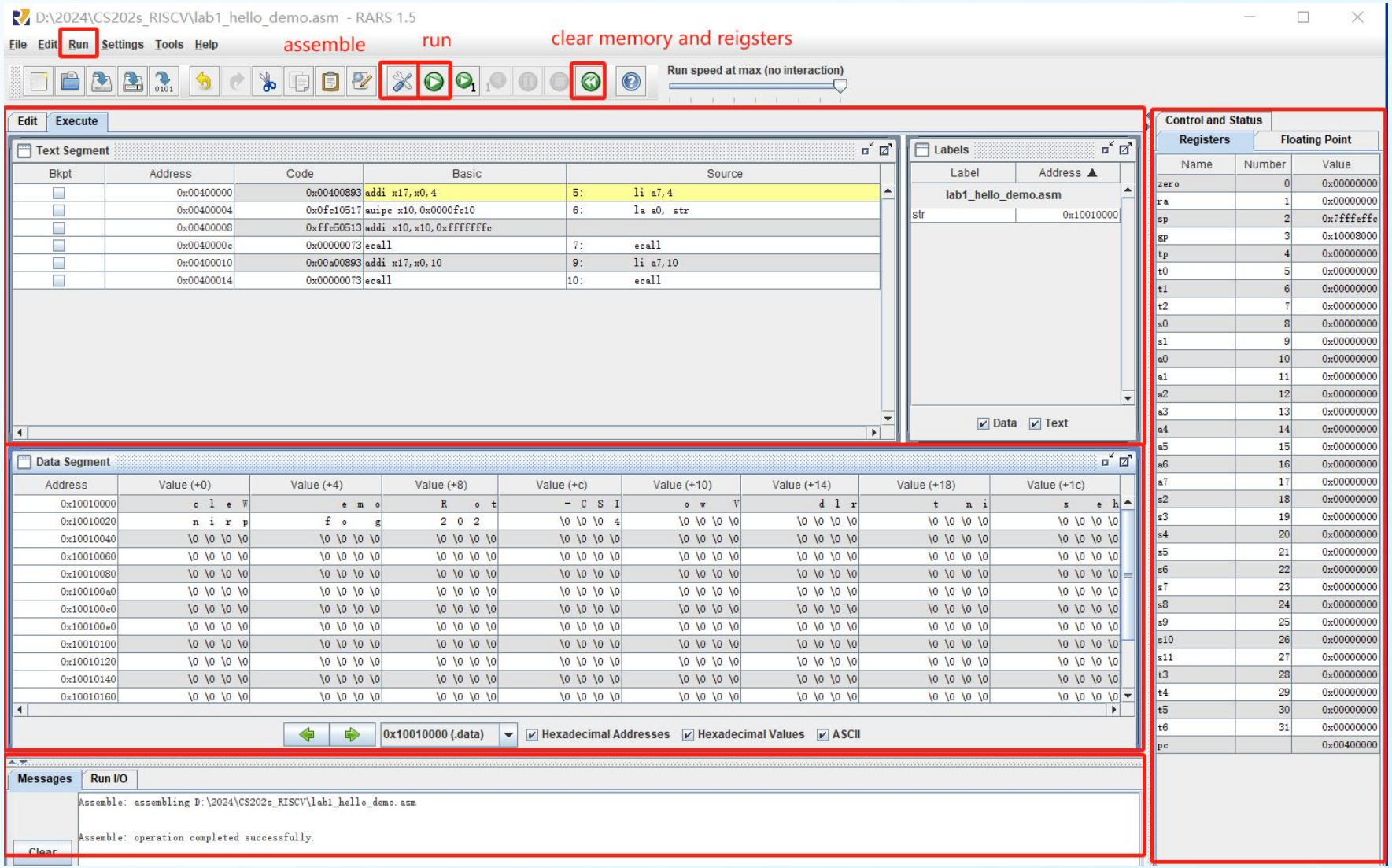
Rars(1)

- **RARS**, the RISC-V Assembler, Simulator, and Runtime, will assemble and simulate the execution of RISC-V assembly language programs.

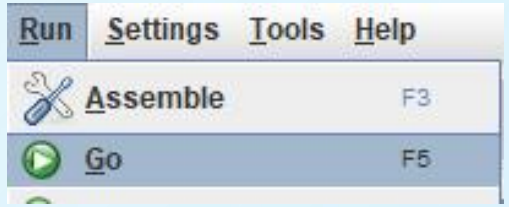
- RARS is written in Java and requires at least Release 1.8 of the Java SE Java Runtime Environment (JRE) to work. It is distributed as an executable JAR file.

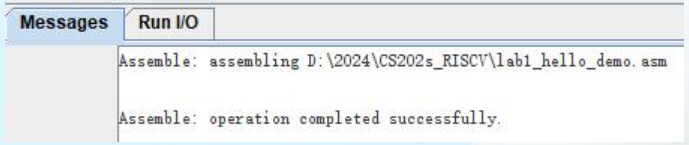
- Download https://github.com/TheThirdOne/rars/releases

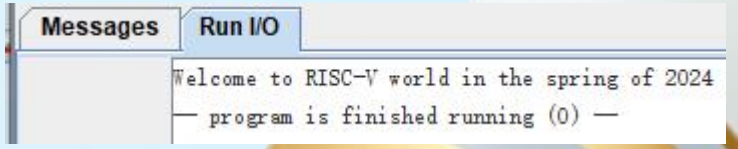
Rars(2)


D:\2024\CS202s_RISCV\lab1_hello_demo.asm - RARS 1.5
File Edit Run Settings Tools Help
assemble
run
clear memory and reigsters
Run speed at max (no interaction)
Edit Execute
Text Segment
Bkpt Address Code Basic Source
0x00400000 0x00400893 addi x17,x0,4 5: li a7,4
0x00400004 0x0fc10517 auipc x10,0x0000fc10 6: la a0, str
0x00400008 0xffc50513 addi x10,x10,0xfffffffc
0x0040000c 0x00000073 ecall 7: ecall
0x00400010 0x00a00893 addi x17,x0,10 9: li a7,10
0x00400014 0x00000073 ecall 10: ecall
Labels
Label Address
lab1_hello_demo.asm
str 0x10010000
Data Text
Control and Status
Registers Floating Point
Name Number Value
Data Segment
Address Value (+0) Value (+4) Value (+8) Value (+c) Value (+10) Value (+14) Value (+18) Value (+1c)
0x10010000 c l e W e m o R o t − C S I o w V d l r t n i s e h
0x10010020 n i r p f o g 2 0 2 \0 \0 \0 4 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII
Messages Run I/O
Assemble: assembling D:\2024\CS202s_RISCV\lab1_hello_demo.asm
Assemble: operation completed successfully.
Clear


Run Settings Tools Help
Assemble F3


Run Settings Tools Help
Assemble F3
Go F5


Messages Run I/O
Assemble: assembling D:\2024\CS202s_RISCV\lab1_hello_demo.asm
Assemble: operation completed successfully.


Messages Run I/O
Welcome to RISC-V world in the spring of 2024
— program is finished running (0) —

TIPS： Useful files and tools in lab1 on BB site

lab1_files

Build Content Assessments Tools

a_tailed_risc-v_CPU_CSE

Attached Files: a_tailed_risc-v_CPU_CSE.bit (2.091 MB)

lab1_practice1_2

Attached Files: lab1_practice1_2.rar (591 B)

includes:

lab1_p1_2.asm

out.txt related to lab1_p1_2.asm

tools

Build Content Assessments Tools

RARS

Attached Files: rars_27a7c1f.jar (1.769 MB)

RARS, the RISC-V Assembler, Simulator, and Runtime, will assemble and simulate the execution of RISC-V assembly language programs

Uart_tools

Attached Files: uart_tools.rar (797.158 KB)

1. rars2coe : change the Rars dump file to coe file
2. UARTCoe_v3.0: merge two coe files into one and adds some tag
3. GenUBit_RISC_V: a batch file to run rars2coe firstly, and run UARTCoe_v3.0 secondly
4. UartAssist: a software to commincate with UART port(usually to send/receive data)

cygwin_install_wizard

Attached Files: cygwin.7z (1.307 MB)

In Computer Orgnization Lab class, cygwin is only useful for the tool "rars2coe" which is used to change the dump file from Rars to coe file.

It's NOT needed in lab1 if you do the experiments on the students PC in lab classroom, If used on one's own computer, it needs to be installed.

NOTES: Cygwin would be avaliable in any path only if it has been installed and its path has been added to the system enviroment "PATH".