



# Computer Organization

## Lab 1: Introduction

---

Special purpose circuit VS General purpose processor

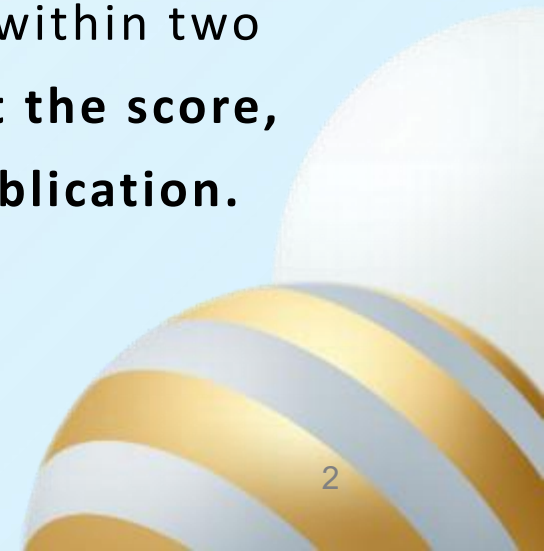


**Tool kits**



## 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

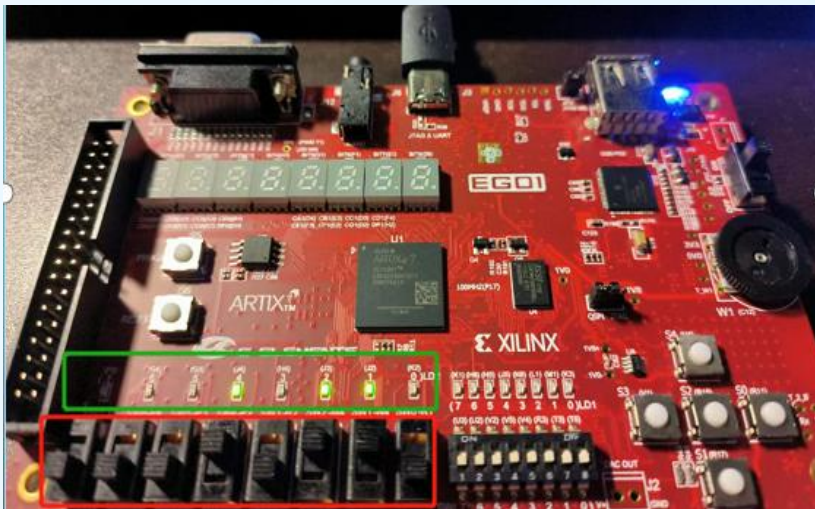
Task	Tool kits
Learn and practice <b>RISC-V</b> ( a type of Assembly language)	<ul style="list-style-type: none"><li>➤ <b>Rars (rars_27a7c1f)</b></li></ul> 
Design and implement an <b>CPU</b>	<ul style="list-style-type: none"><li>➤ <b>Vivado</b></li></ul>   <ul style="list-style-type: none"><li>➤ <b>FPGA based Development Board EGO1</b></li></ul>
Test the <b>CPU</b> with <b>program(s)</b> , both of which are based on <b>RISC-V</b>	<ul style="list-style-type: none"><li>➤ <b>Assembler (Rars)</b></li><li>➤ <b>Uart Tools (GenUBit_RISC_V, UartAssist)</b></li></ul>  <ul style="list-style-type: none"><li>➤ <b>Vivado</b></li><li>➤ <b>FPGA based Development Board(EGO1)</b></li></ul>



## 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
```

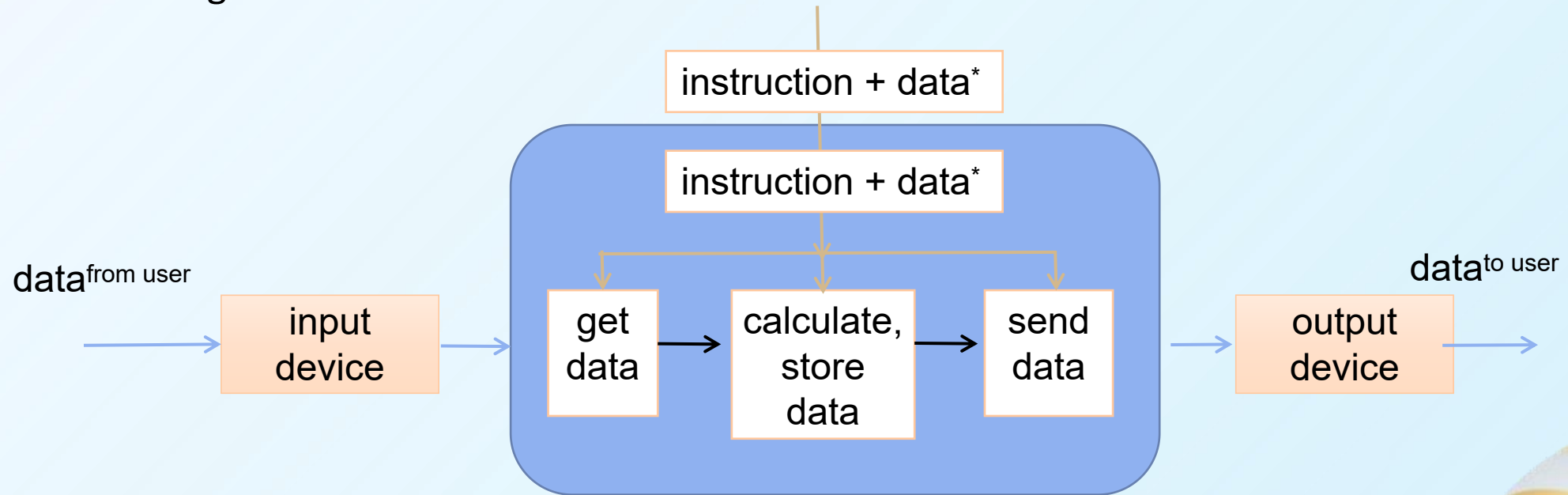
```
#Constraint file, save as sw2led.xci
set_property IOSTANDARD LVCMOS33 [get_ports {led[ ]}]
...
set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[ ]}]
...
set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN F6 [get_ports {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 1<sup>st</sup>, execution 2<sup>nd</sup>
  - Program = instruction + data<sup>\*</sup>





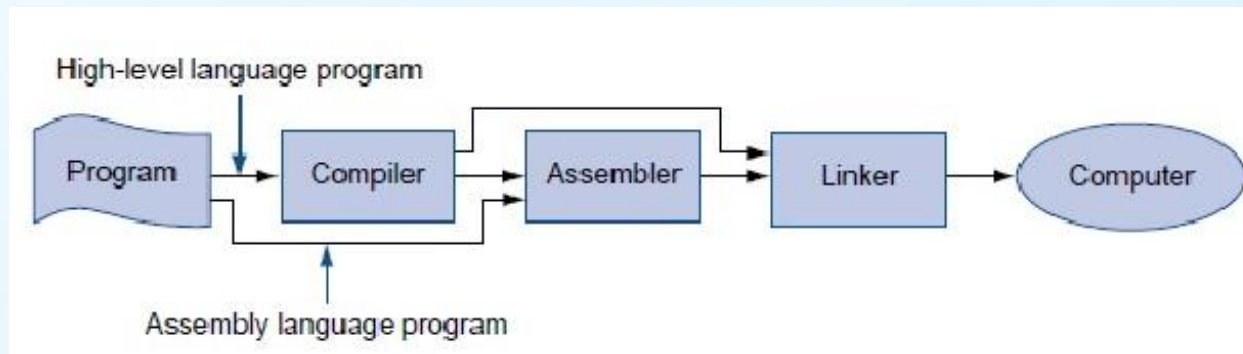
# Programming

**Programming: Analysis + Design + Description + Debug and Test**

- **Description:** 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. execution given label main.

- **Part3: Comments (suggested)**

- anything following # on a line  
# This stuff would be considered 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
```

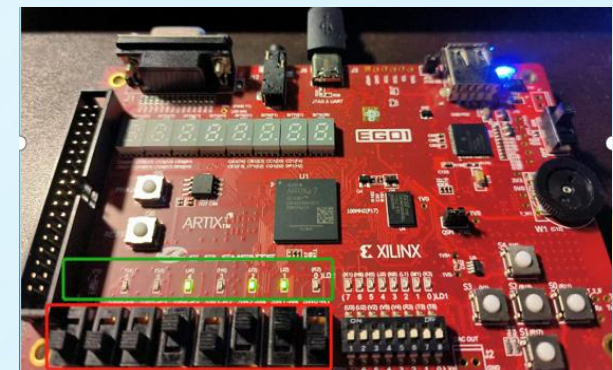
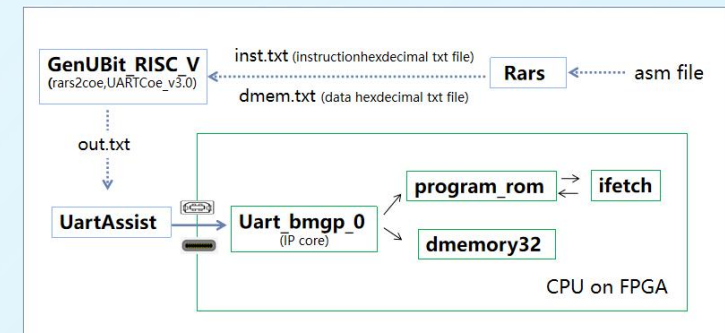
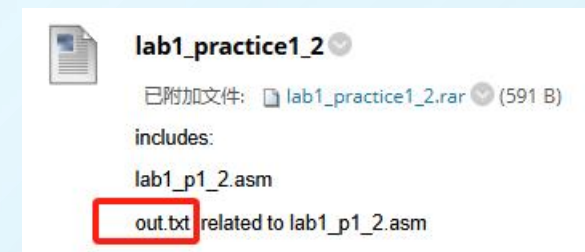
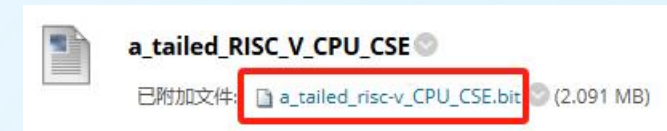




## Practice1-2: lighting led(s) by 'General purpose processor'

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 lighting 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'.





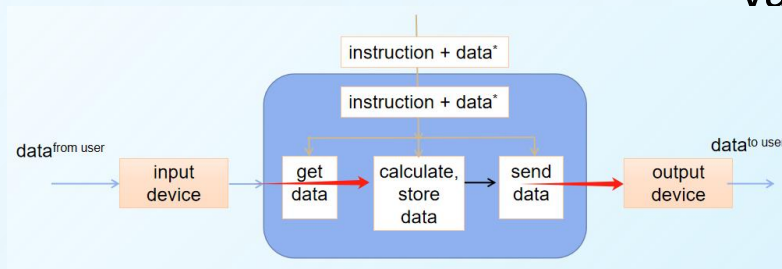
## Practice1-2(1) - The 'tailored General purpose processor'

### ✓ 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 **0xFFC8**)
  - ✓ **read from 0xFFC8 is to get the data from 8 switches**
- ✓ send data to output(addressed by **0xFFC4**)
  - ✓ **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

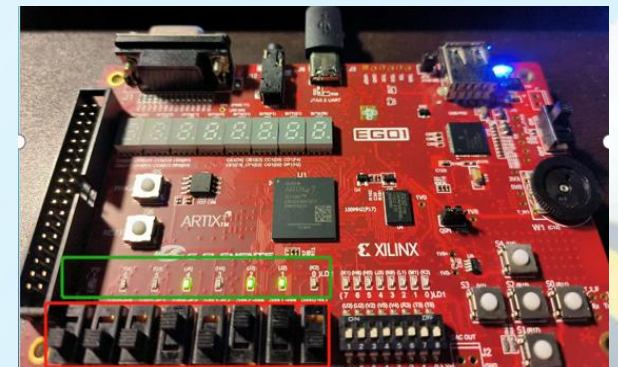
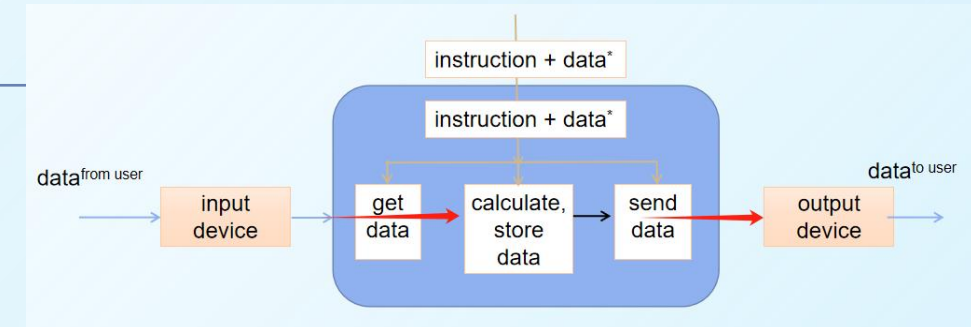
#assmbly source file in RISC-V

.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 `assembly` source file.

[Using **Rars**]

2. **Assembler** it to generate the machine code, then dump to file(s).

[Using **Rars**]

3. **Generate** `out.txt`

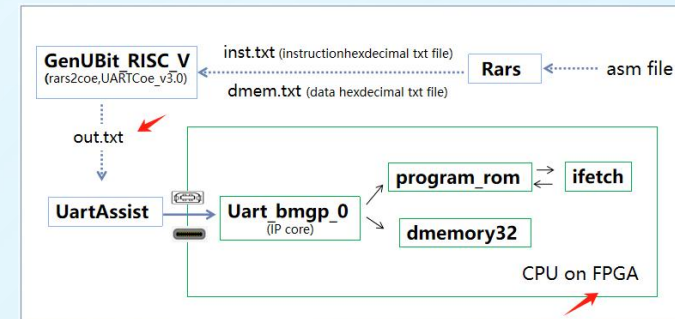
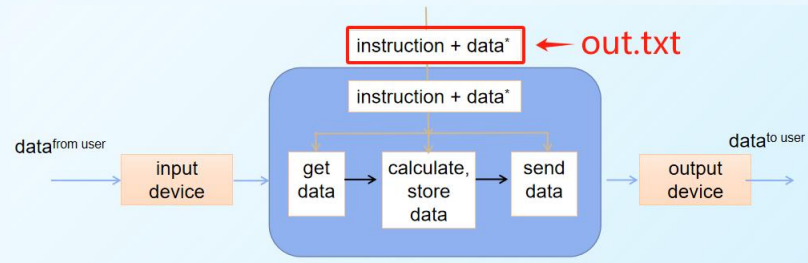
[Using **GenUbit\_RISC\_V**]

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)

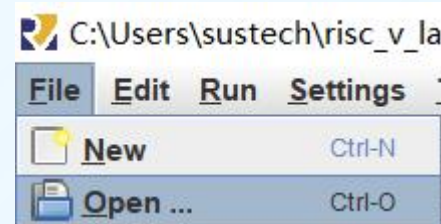
Using vivado's "Program device" to write the bitstream file of the 'tailored General purpose processor' to the FPGA chip of EGO1.



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

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

1-1. Open the asm file

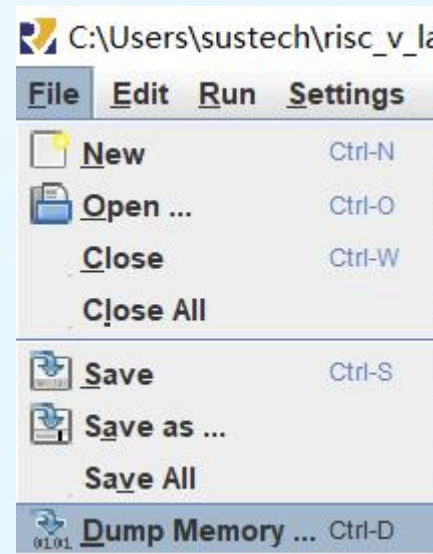


1-2. Assemble the asm file to the machine code

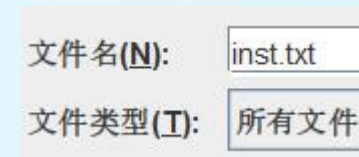
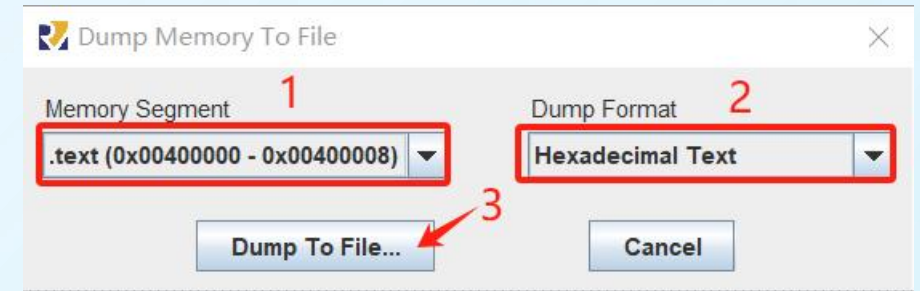


1-3. Dump machine code and data in available format.

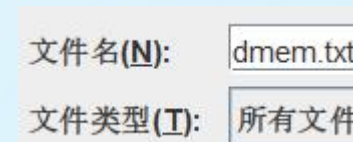
NOTE:  
The instruction dump file is suggested to be named as “**inst.txt**”, while the data dump file is suggested to be named as “**dmem.txt**”



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**'.

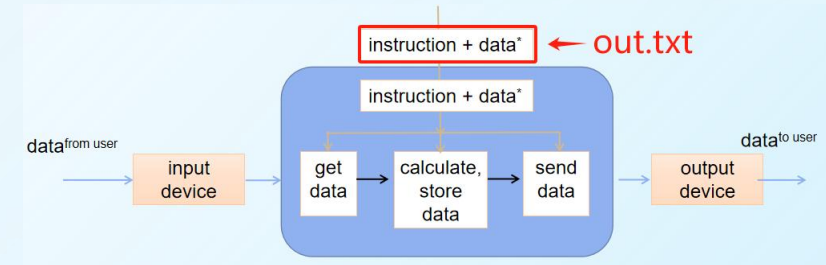
```
C:\Users\sustech\Downloads\uart_tools>ls
GenUBit_RISC_V.bat  UARTCoe_v3.0.exe  UartAssist.exe  dmem.txt  inst.txt  rars2coe.exe
```

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.

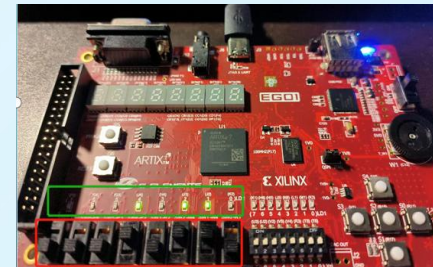
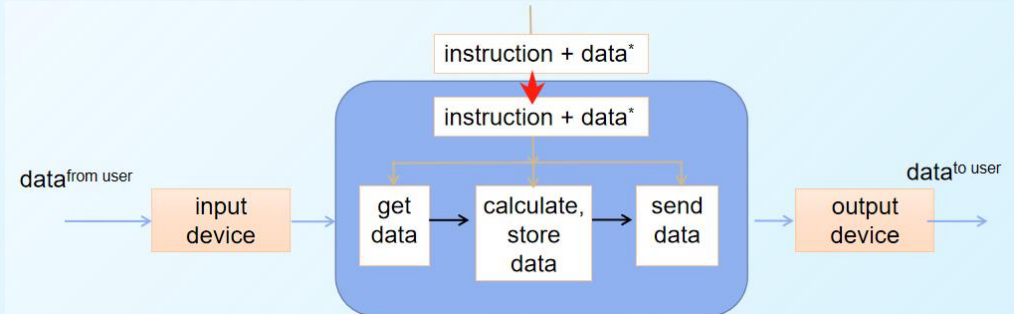
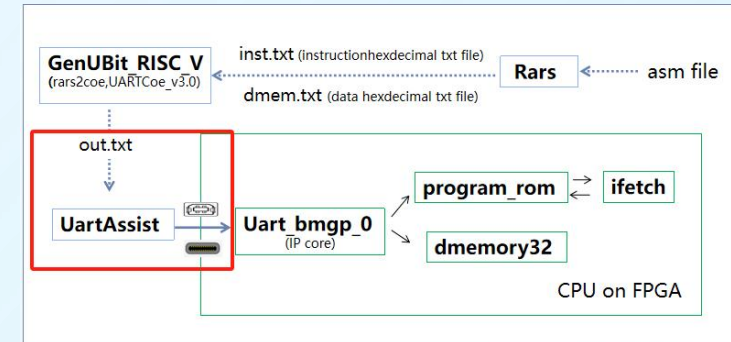
➤step1: Bounce after pressing the button **S6**(on EGO1) to prepare for receiving the instructions from uart.



➤step2: Send the instrusctions(in file '**out.txt**') to the 'tailored General purpose processor' by 'UartAssist'

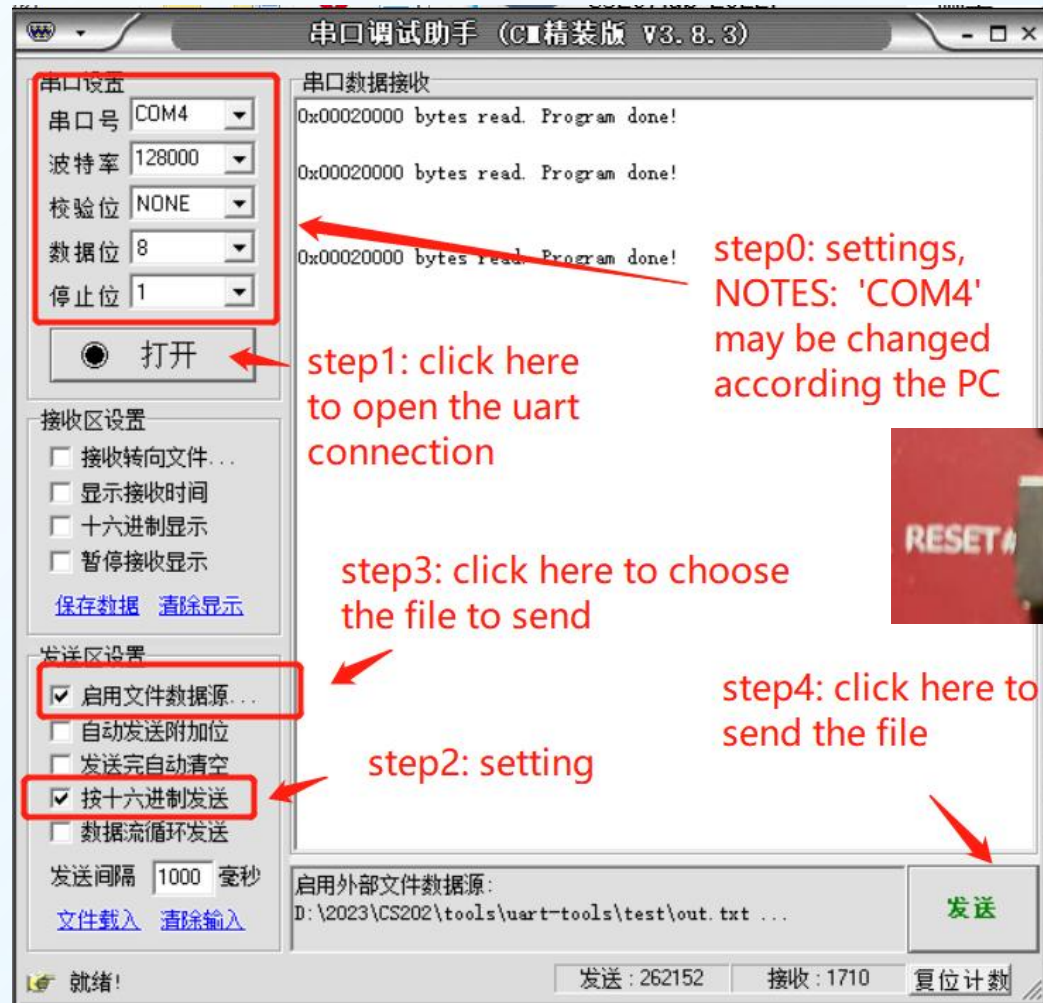
➤step3: While the 'UartAssist' receive the data from the 'tailored General purpose processor', and show "0x00020000 bytes read. **Program done!**", which indicates that the 'tailored General purpose processor' has successfully received the 'instruction + data\*', and start the processing process based on them.

➤step4: turn on/ off the Dial switches, 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.

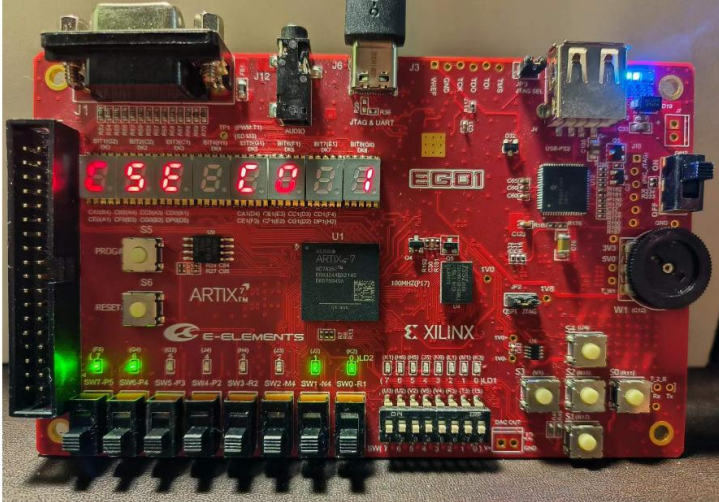
**step-2.** Press the button "S6" on the EGO1 board.

**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.

- Assemble the new asm file with '**Rars**', then dump the 'inst.txt' and 'dmem.txt', Using 'GenUbit\_RISC\_V' to generate the 'out.txt'.
- Repeat 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
```

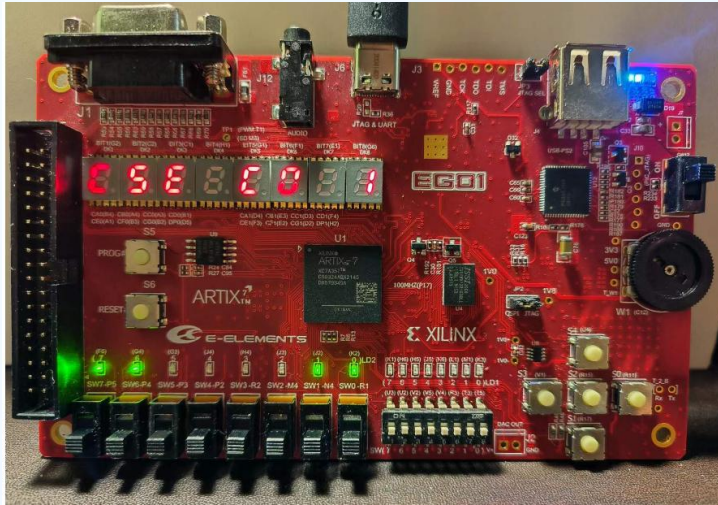
```
.data                                #D
    buf: .byte 0xC3
```

```
.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 <b>RISC-V</b> ( a type of Assembly language)	➤ <b>Rars (rars_27a7c1f)</b>
Design and implement an <b>CPU</b>	➤ <b>Vivado</b> ➤ <b>FPGA based Development Board EGO1</b>
Test the CPU with <b>RISC-V</b>	➤ <b>Assembler</b> ➤ <b>Uart Tools</b> ➤ <b>Vivado</b> ➤ <b>FPGA based Development Board</b>





## 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 registers

Run speed at max (no interaction)

Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x00400893	addi x17,x0,4	5: li a7,4
<input type="checkbox"/>	0x00400004	0x0fc10517	swipc x10,0x0000fc10	6: la a0, str
<input type="checkbox"/>	0x00400008	0xffe50513	addi x10,x10,0xfffffff3	
<input type="checkbox"/>	0x0040000c	0x00000073	ecall	7: ecall
<input type="checkbox"/>	0x00400010	0x00a00893	addi x17,x0,10	9: li a7,10
<input type="checkbox"/>	0x00400014	0x00000073	ecall	10: ecall

Labels

Label	Address
lab1_hello_demo.asm	
str	0x10010000

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	c l e f	e m o	R o t	- C S I	o w Y	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
0x10010040	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010060	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010080	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100a0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100c0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100e0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010100	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010120	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010140	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010160	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

Control and Status

Registers

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffeffe
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

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 communicate with UART port( usually to send/receive data)



#### cygwin\_install\_wizard

Attached Files: [cygwin.7z](#) (1.307 MB)

In Computer Organization 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 available in any path only if it has been installed and its path has been added to the system environment "PATH".