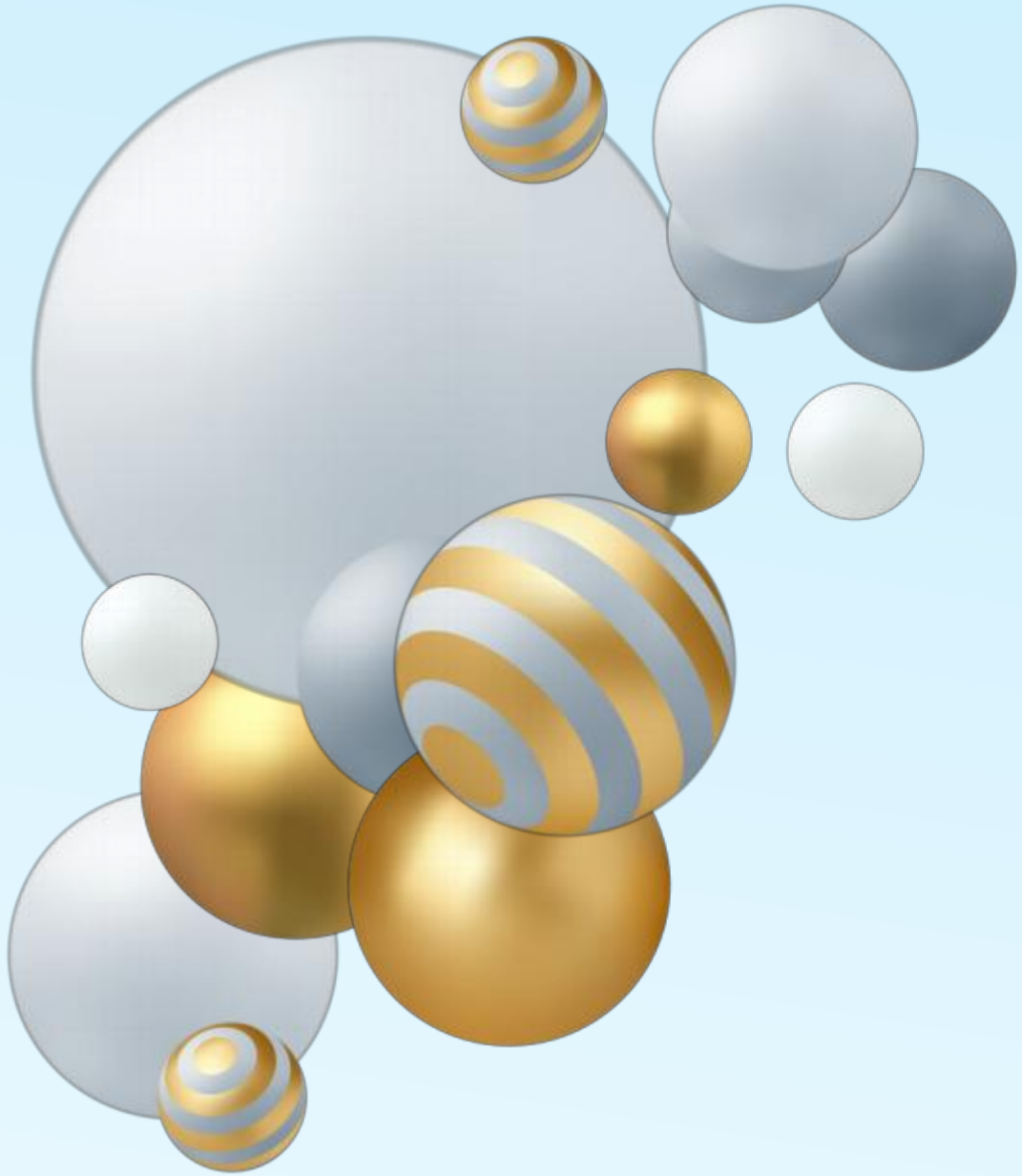


CS202

Lab1:Introduction

Special purpose circuit and General purpose processor

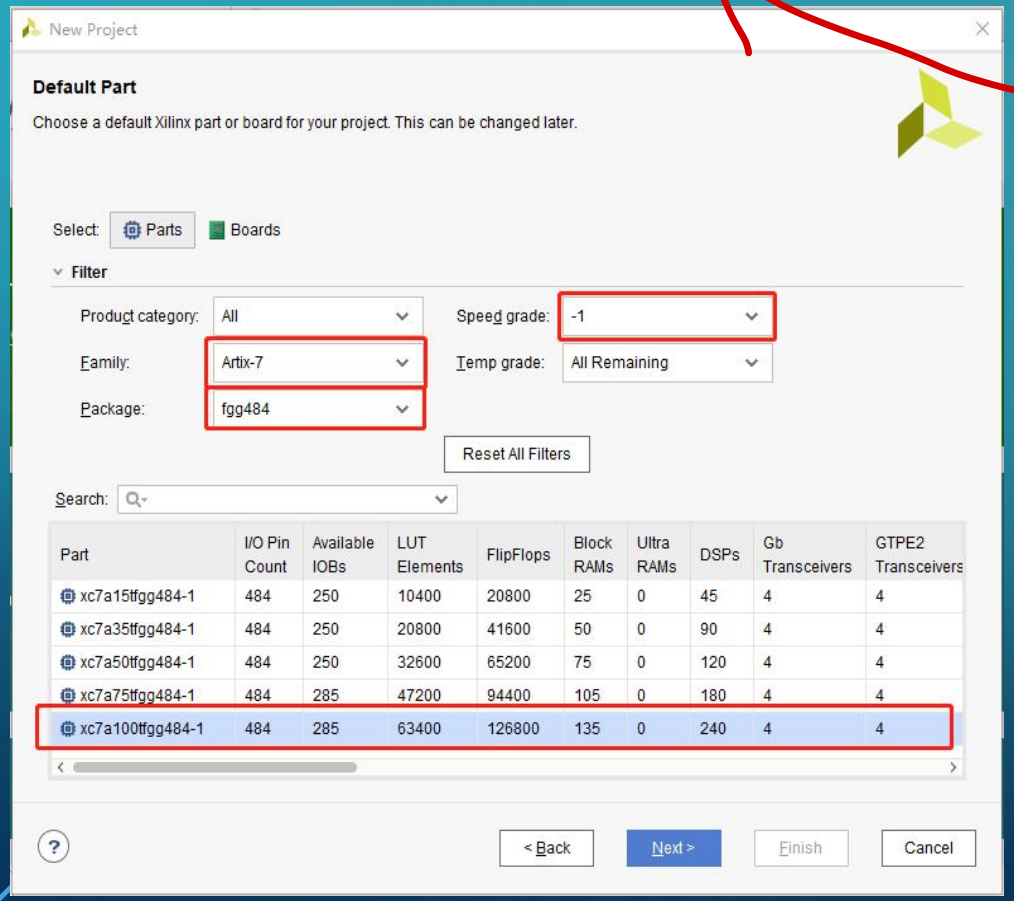
Tool kits

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

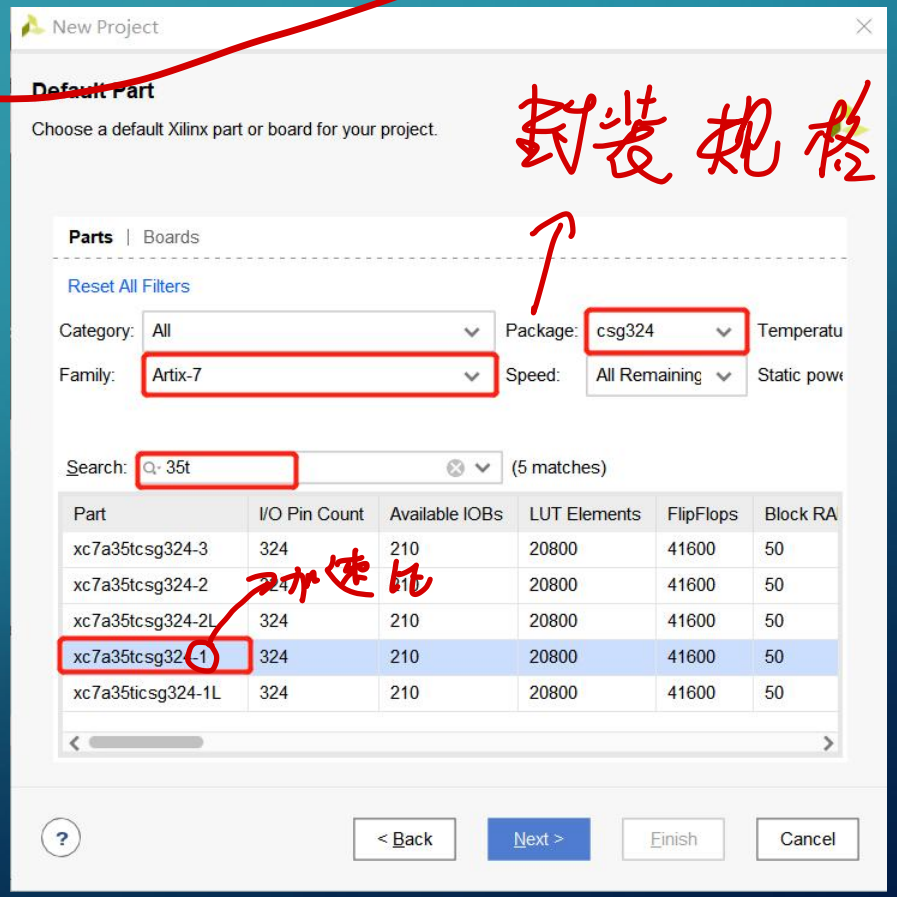
USING VIVADO + FPGA DEVELOPMENT BOARD(2)

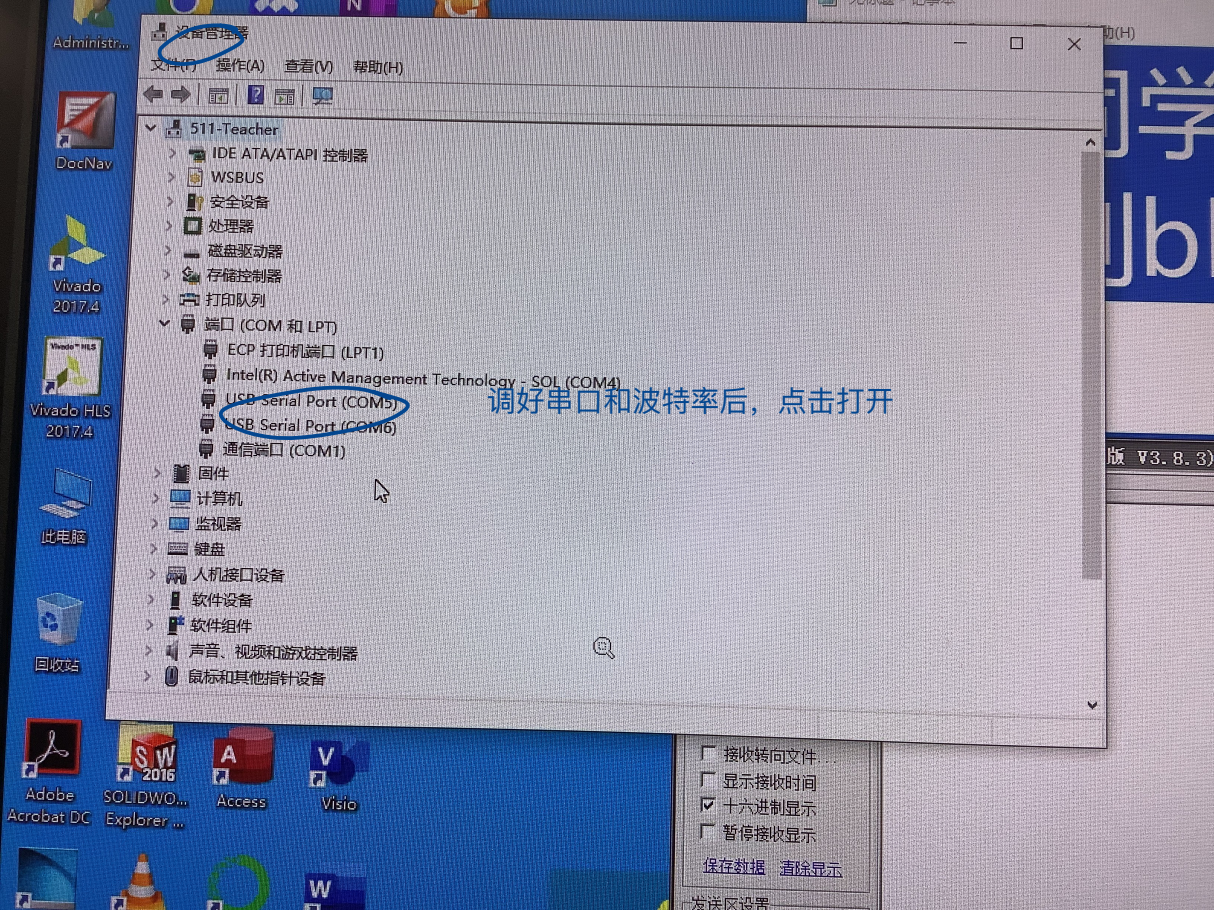
STEP1-2. select the corresponding FPGA chip. The version of FPGA chip in EGO1 is different from which in Minisys board.

The version of FPGA Chip in Minisys board:
Artix 7 xc7a100t fgg484-1



The version of FPGA Chip in EGO1 board:
Artix 7 xc7a35t CSG324-1




设备管理器
文件(F) 操作(A) 查看(V) 帮助(H)
511-Teacher
IDE ATA/ATAPI 控制器
WSBUS
安全设备
处理器
磁盘驱动器
存储控制器
打印队列
端口 (COM 和 LPT)
ECP 打印机端口 (LPT1)
Intel(R) Active Management Technology - SOL (COM4)
USB Serial Port (COM5)
USB Serial Port (COM6)
通信端口 (COM1)
固件
计算机
监视器
键盘
人机接口设备
软件设备
软件组件
声音、视频和游戏控制器
鼠标和其他指针设备
调好串口和波特率后，点击打开
接收转向文件
显示接收时间
十六进制显示
暂停接收显示
保存数据
清除显示
DocNav
Vivado 2017.4
Vivado HLS 2017.4
此电脑
回收站
Adobe Acrobat DC
SOLIDWO... Explorer ...
Access
Visio

Contact

- BlackBoard site:
  - Computer Organization Spring 2023
- QQ group:
  - 476808329

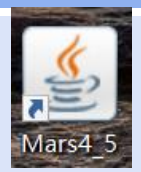
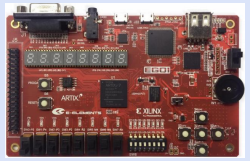
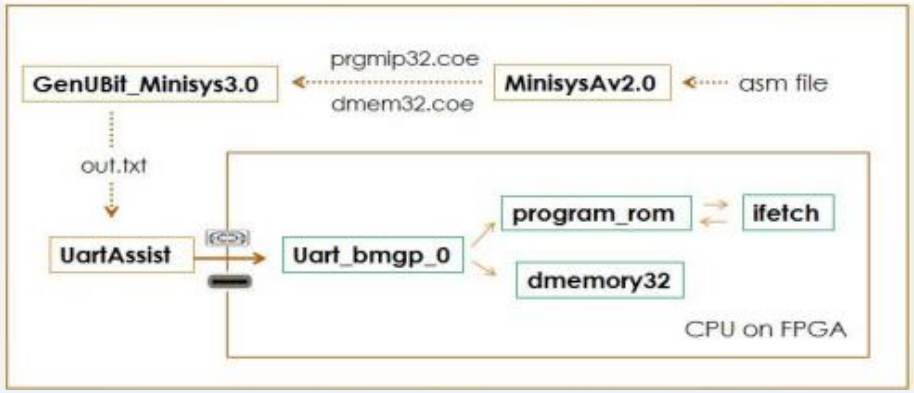
Assignments Submission Rules

- 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

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** 

以纯硬件电路进行实现

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

Design file + Constraint file

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

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

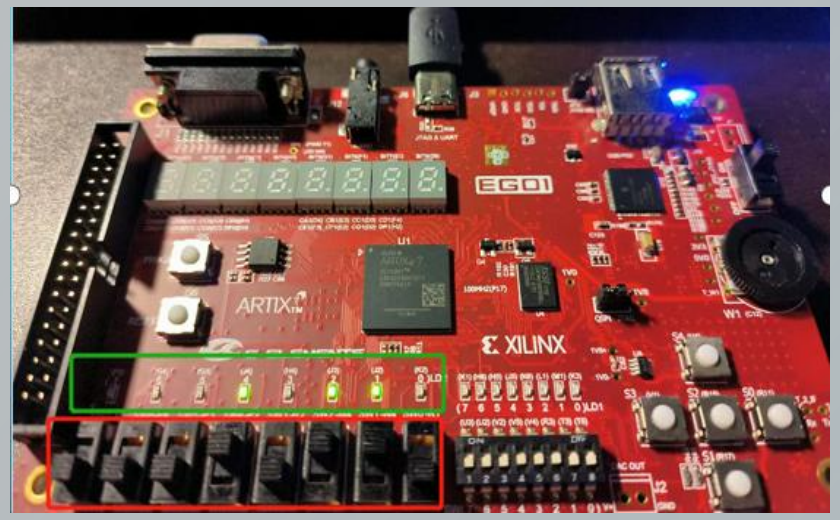
---> the circuit is implemented on the **FPGA chip**

---> **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
```

不仅需要设计，也需要约束文件

检查setting general project device
是否为对应开发板



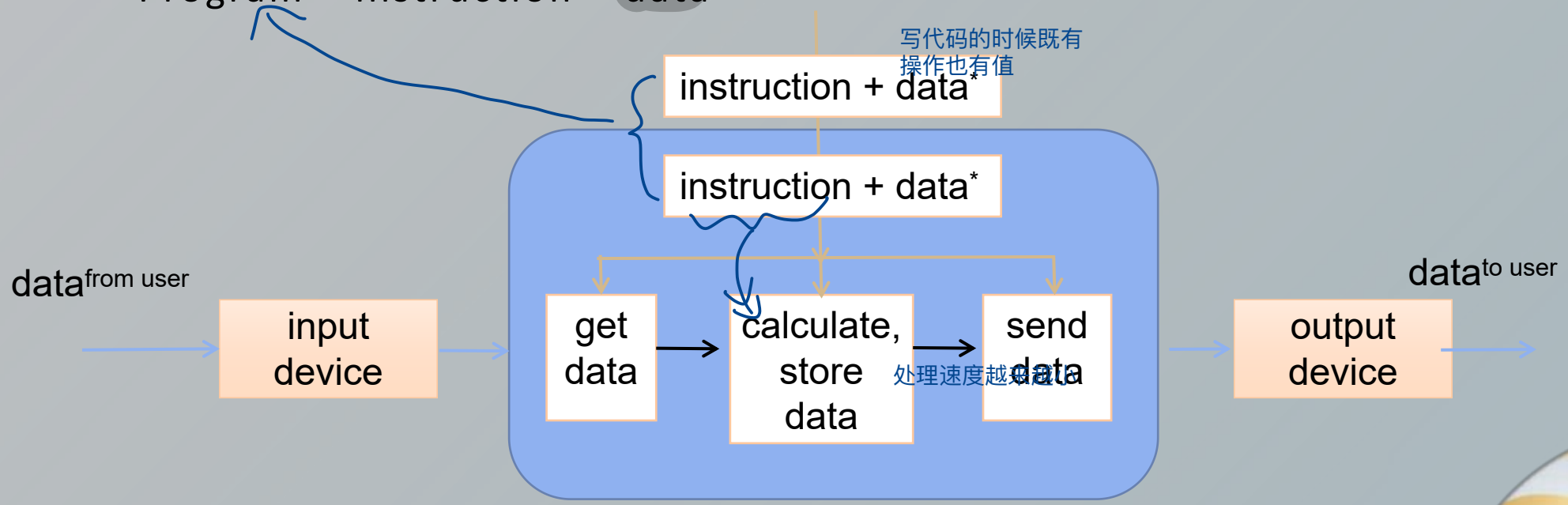

```
#Constraint file, example: sw2led.xdc
set_property IOSTANDARD LVCMOS33 [get_ports {led[ ]}]
...
set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[ ]}]
...
set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN F6 [get_ports {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]}]
```

电气特性? 3.3v

The 'General purpose processor'

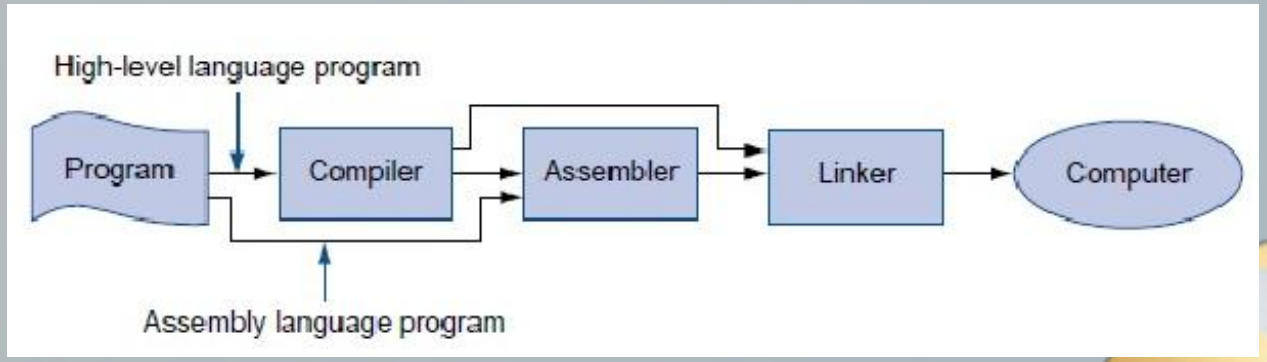
冯-诺-依体系结构

- 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

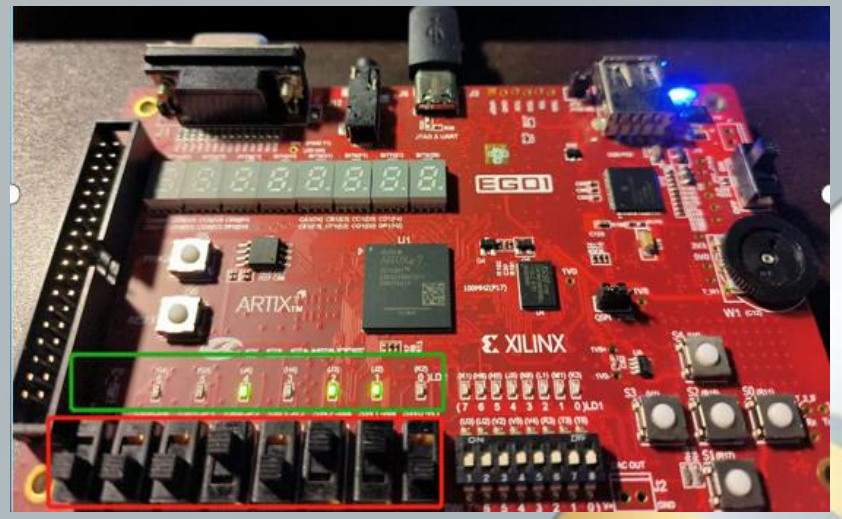
```
# Comment giving name of program and description of function
# Template.s
# Bare-bones outline of MIPS assembly language program

            .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
  press s3
- Send the instructions and data* (‘out.txt’) to the ‘tailored General purpose processor’, the ‘tailored General purpose processor’ receive the instructions and data*
  press s2
  using UartAssist
  see p15
- Run the ‘instructions and data*’ on the ‘tailored General purpose processor’
- Test

两种工作模式：通过按键切换



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.

s3 control it

s2 control it

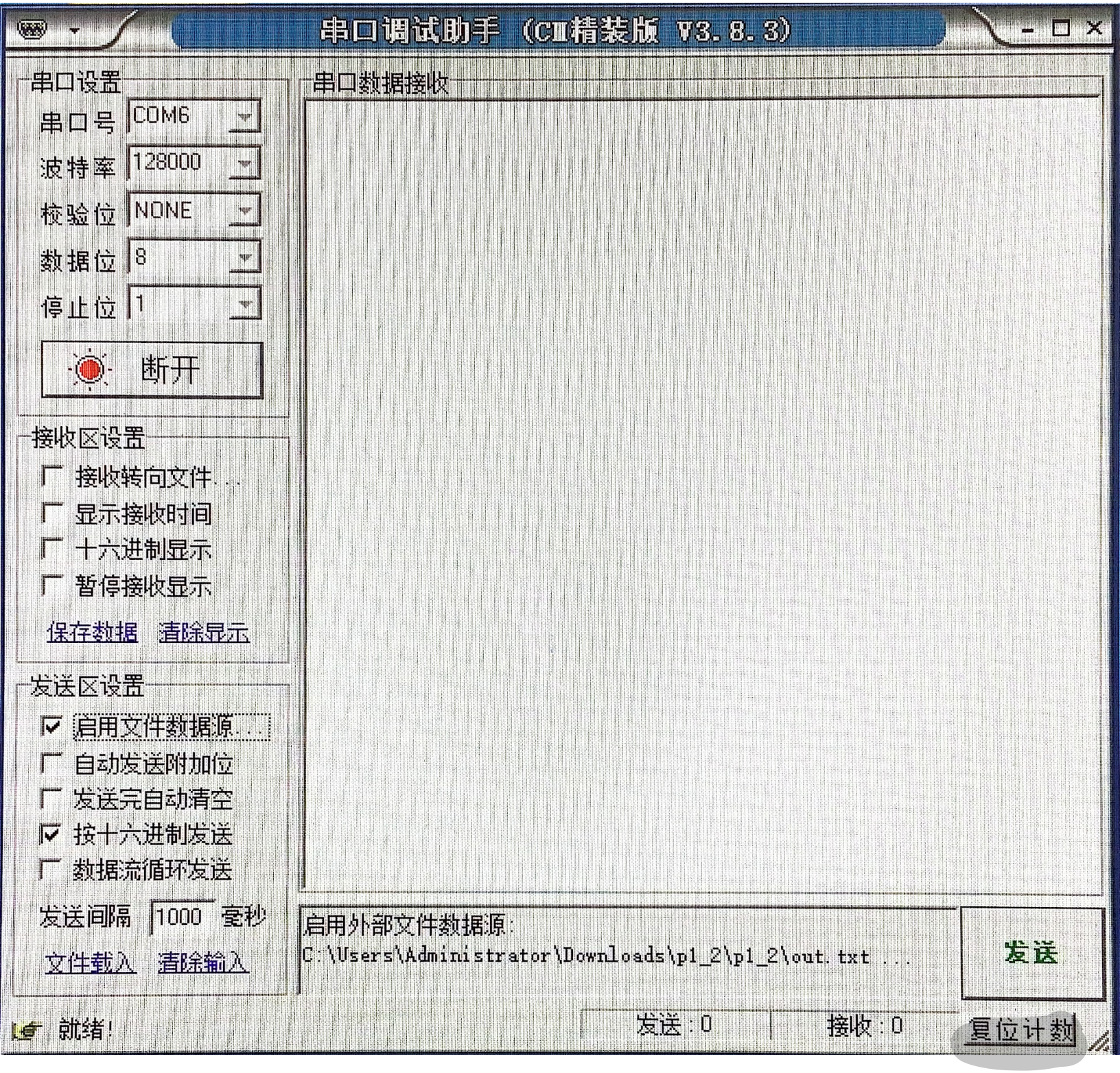
- ✓ **Data flow**
  - ✓ the data **MUST be stored into the register(s)** of 'tailored General purpose processor' before be calculated or be sent to the output device.

- ✓ **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.

other registers can't store data

Special one

- ✓ **I/O and address**
  - ✓ get data from input
    - ✓ 0x**FFFF_FC70**
      - ✓ **read from 0XFFFF_FC70 is to get the data from 16 switches**
    - ✓ 0x**FFFF_FC72**
      - ✓ **read from 0xFFFF_FC72 is to get 16bits which value is 16'H00FF**
  - ✓ send data to output
    - ✓ 0x**FFFF_FC60**
      - ✓ **write to 0xFFFF_FC60 is to write the data to 16 leds**
    - ✓ 0x**FFFF_FC62**
      - ✓ 0x**FFFF_FC60 is NOT used yet**


串口调试助手（CⅢ精装版 V3.8.3）
串口设置
串口号
COM6
波特率
128000
校验位
NONE
数据位
8
停止位
1
断开
接收区设置
接收转向文件...
显示接收时间
十六进制显示
暂停接收显示
保存数据
清除显示
发送区设置
启用文件数据源...
自动发送附加位
发送完自动清空
按十六进制发送
数据流循环发送
发送间隔
1000
毫秒
文件载入
清除输入
串口数据接收
启用外部文件数据源:
C:\Users\Administrator\Downloads\p1_2\p1_2\out.txt ...
发送
就绪!
发送：0
接收：0
复位计数

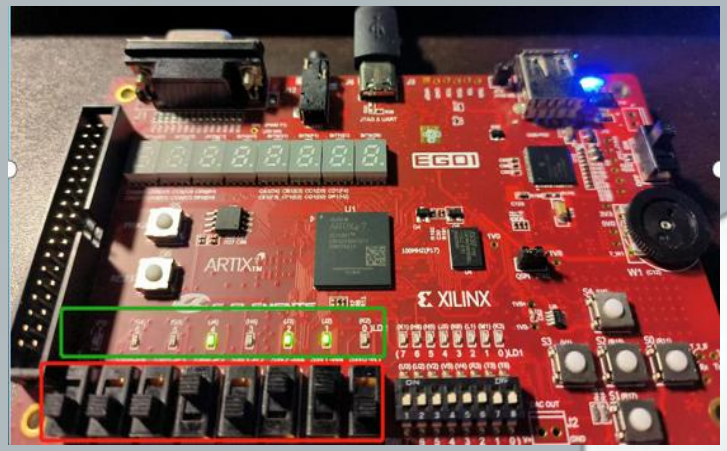

串口调试助手 (C■精装版 V3.8.3)
串口设置
串口号 COM6
波特率 128000
校验位 NONE
数据位 8
停止位 1
断开
接收区设置
接收转向文件...
显示接收时间
十六进制显示
暂停接收显示
保存数据 清除显示
发送区设置
启用文件数据源...
自动发送附加位
发送完自动清空
按十六进制发送
数据流循环发送
发送间隔 1000 毫秒
文件载入 清除输入
串口数据接收
0x00020000 bytes read. Program done!
Indicating the sent has been finished
启用外部文件数据源:
C:\Users\Administrator\Downloads\p1_2\p1_2\out.txt ...
发送
就绪!
发送:131076
接收:38
复位计数

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

```
#assmbly source file
.data 0x0000                    # data* is 4bytes, 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
```

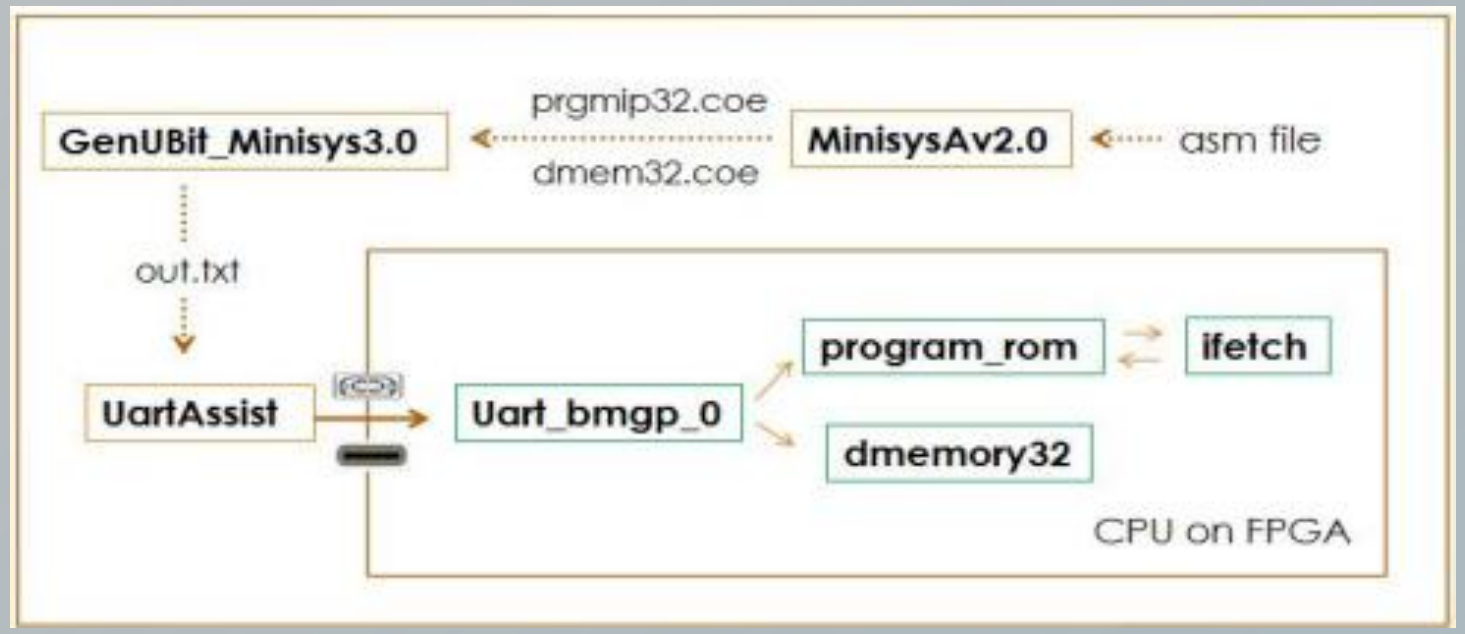


Practice1-2(3) - Test:lighting the led by 'General purpose processor'

Preparation:

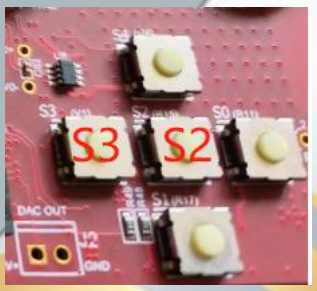
1. **Build** the assmbly source file, **assembler** it with '**MinisysAv2.0**' to get the coe file(s), Using '**GenUbit_Minisys3.0**' to **merger** two coe files into 'out.txt'.

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



Test on the board:

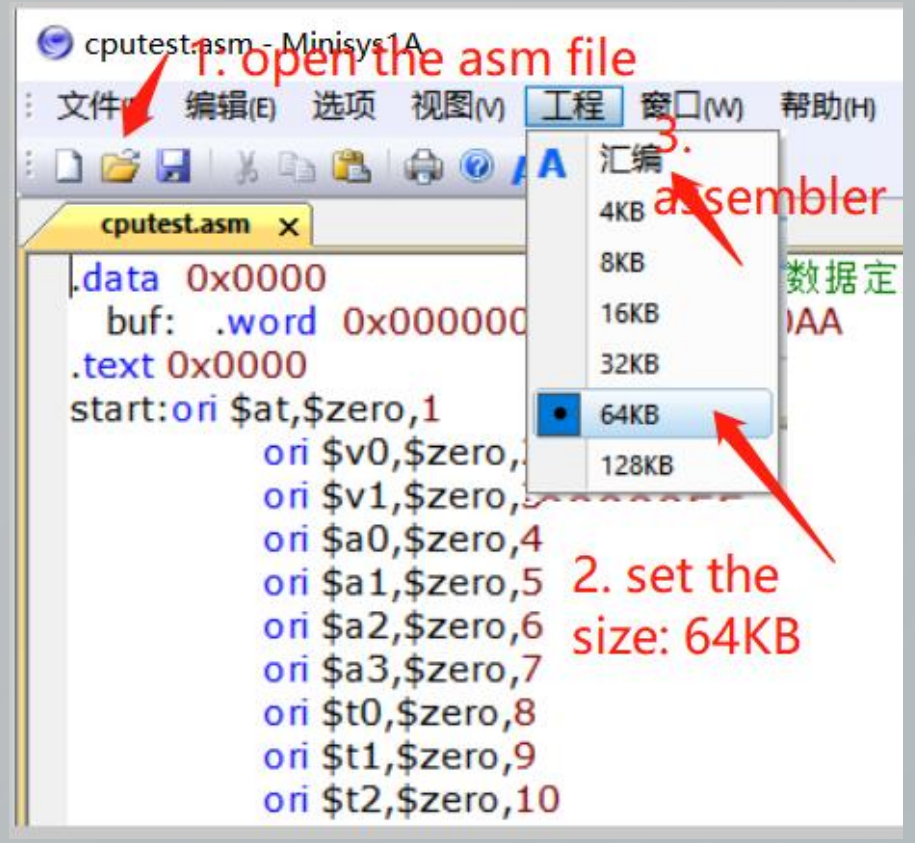
- step1: Bounce after pressing the button **S3**(on EGO1) to prepare for receiving the instructions from uart.
- step2: Send the instrusctions(in file 'out.txt') to the 'tailored General purpose processor' by 'UartAssist'
- step3: Bounce after pressing the button **S2**(on EGO1) to make the 'tailored 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:

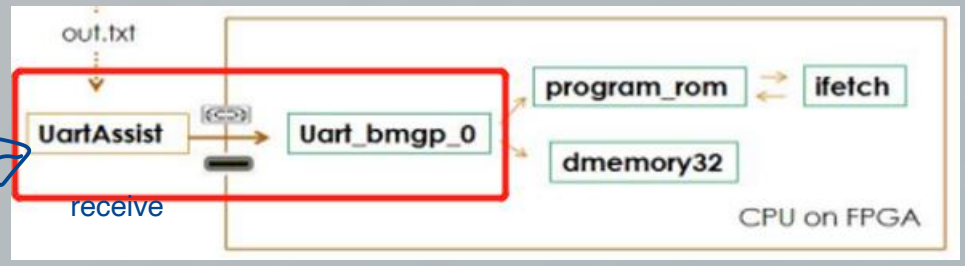
To generate the **out.txt**, the two coe files(prgmips32.coe and dmem32.coe) **MUST** be with the same direcotry as ‘GenUBit_Minisys3.0’ and ‘UARTCoe_v3.0’.

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:

2 modes are controlled by button



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)**

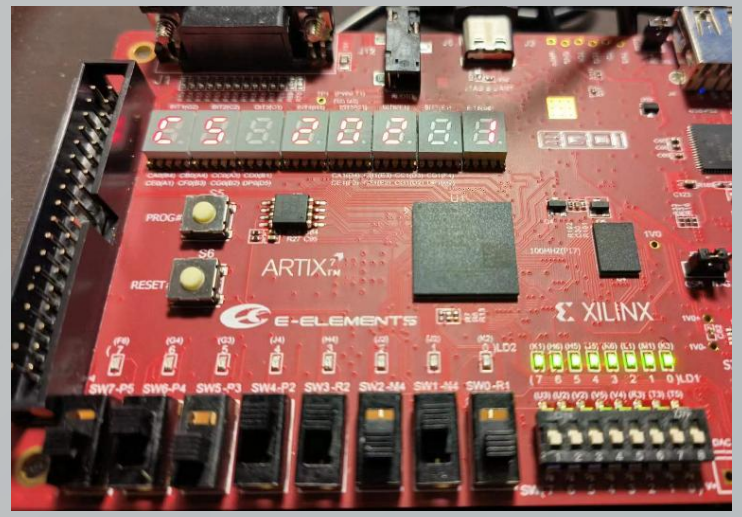
NOTE:

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’



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 16 'h00ff no matter how the dial switch changes.

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

64K. 汇编. coe. 0 warning 0 error 成功

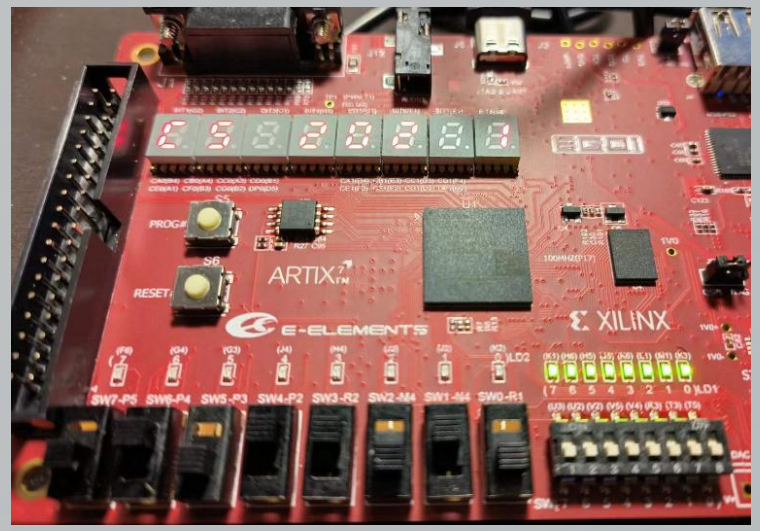
```
.data 0x0000                #A
        buf: .word 0x0000

.text 0x0000
start:
        lw   $1,0xC70($31)
        sw   $1,0xC62($31)

        j start
```

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

.text 0x0000
start:
        lw    $1,0xC72($31)
        sw    $1,0xC60($31)
        j start
```

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

.text 0x0000
start:
        lw   $1,0xC70($31)
        sw   $1,0xC60($31)

        j start
```

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

.text 0x0000
start:
        lw   $1,0xC72($31)
        sw   $1,0xC62($31)

        j start
```

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



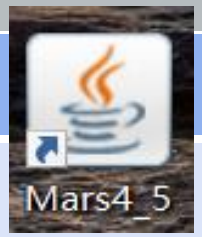
- To implement the same logical relationship between inputs and outputs in this test(keep the state of the led remains at 16 'h00ff no matter how the dial switch changes), which of the following processes are needed on practice 1-1(Lighting the led by ‘Special purpose circuit’ on page 6)?

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

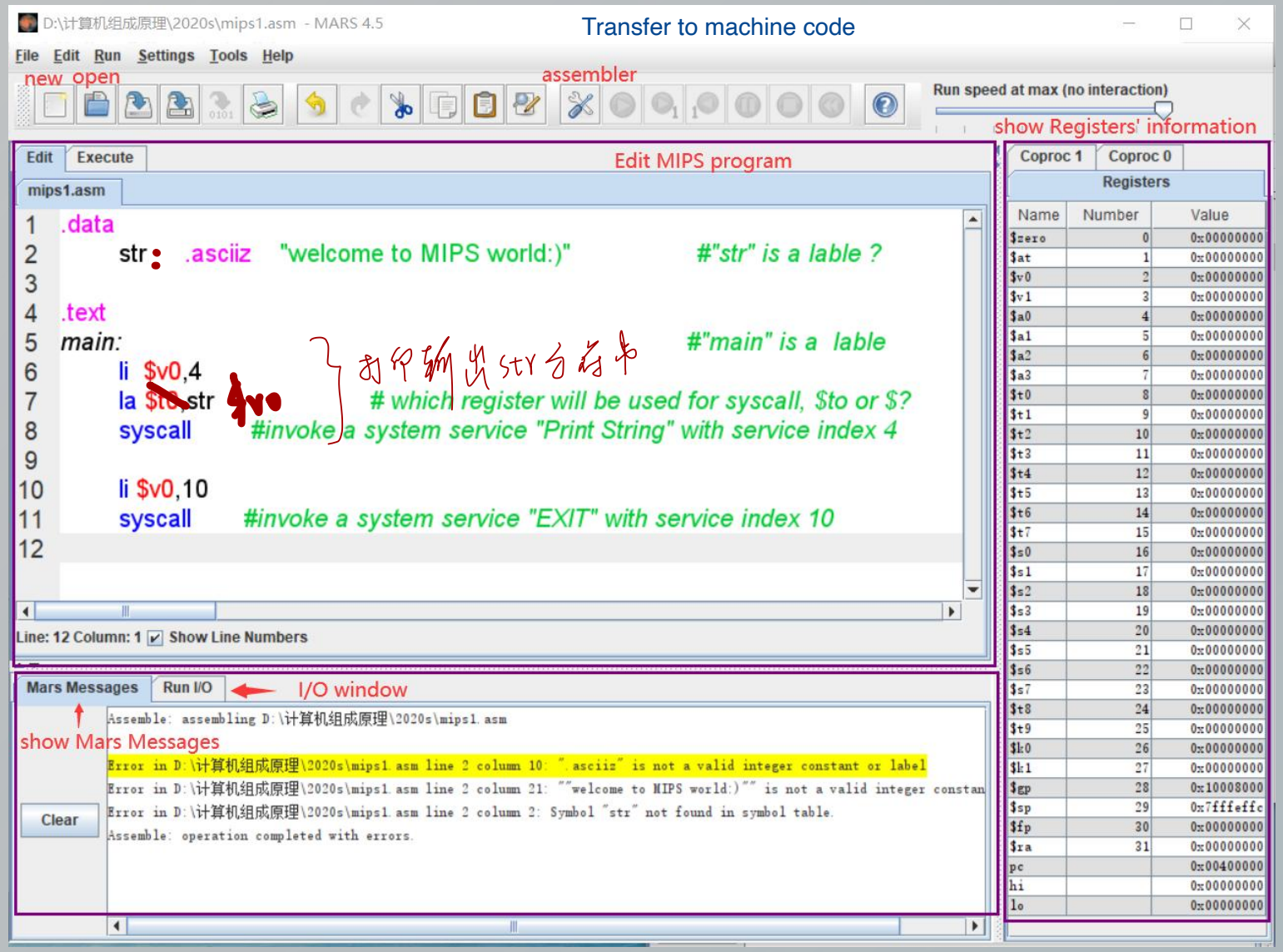
Experimental Tool Kits （Simulator： Mars）

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

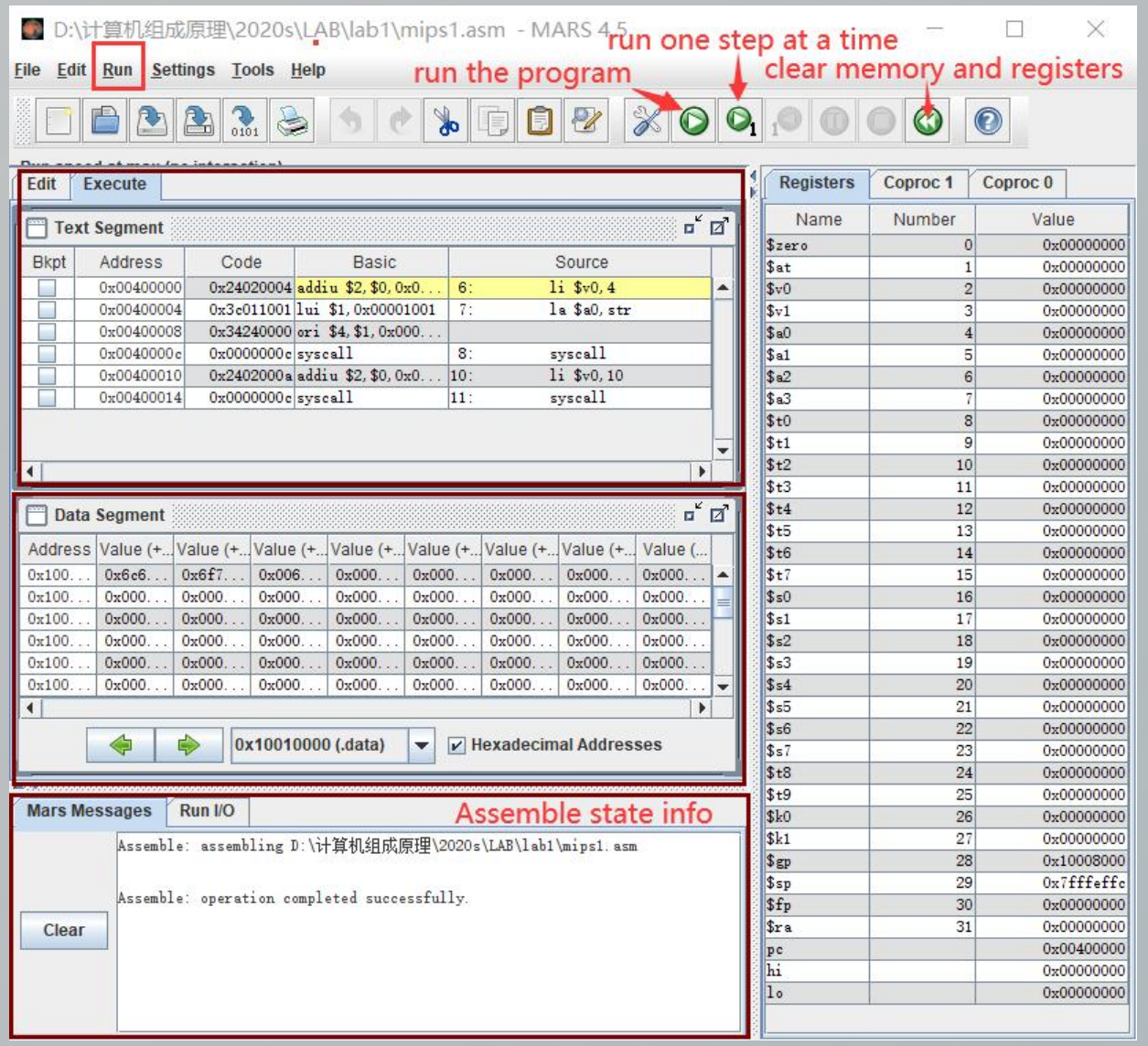


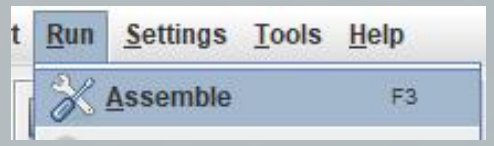
MARS is a **lightweight** interactive development environment (**IDE**) for programming in **MIPS** assembly language, intended for educational-level use with Patterson and Hennessy's Computer Organization and Design.

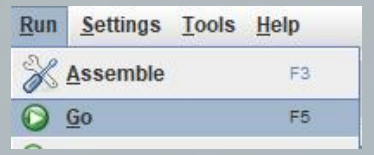
MARS are based on MIPS

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

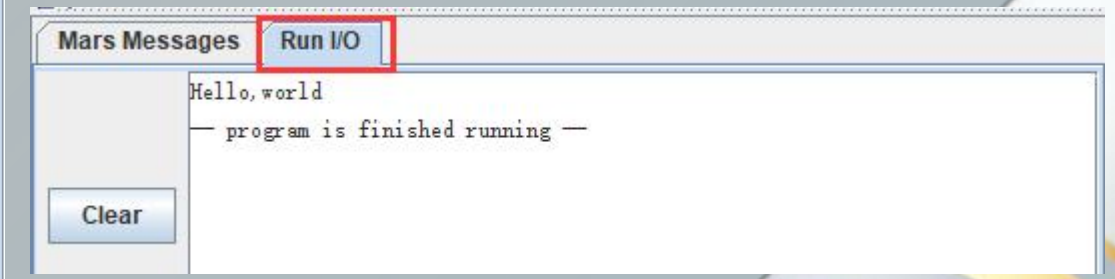
Mars(2)


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

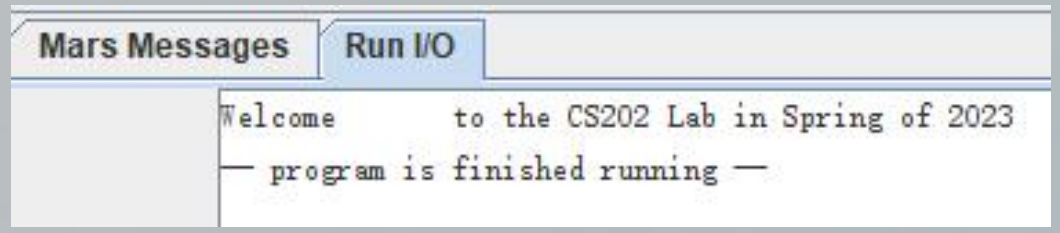

Run Settings Tools Help
Assemble F3
Go F5


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. Get the student ID from the keyboard, print the string "Welcome xxx to the CS202 Lab in Spring of 2023" on the "Run I/O" of Mars (xxx is your student ID).

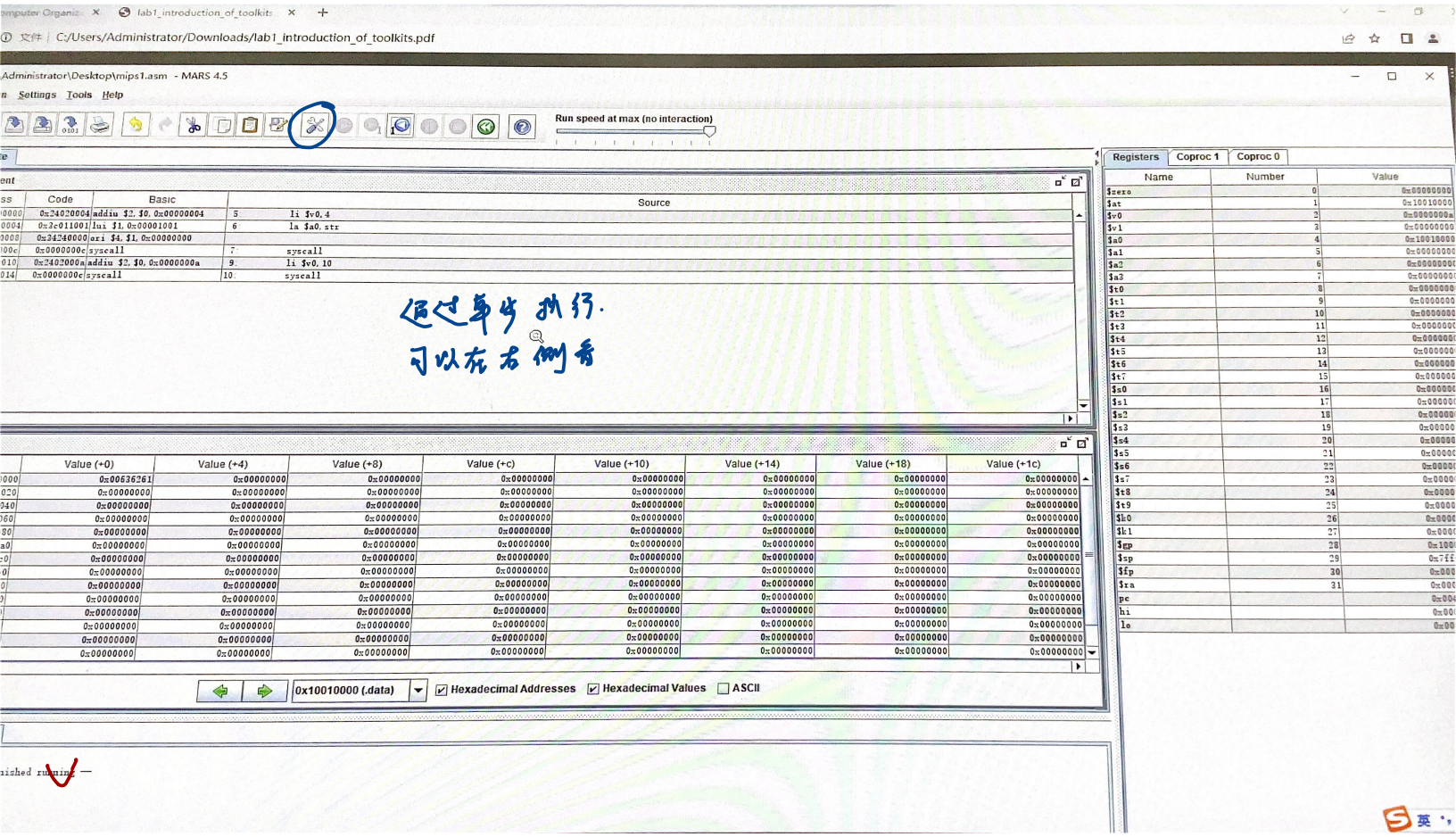


- 2-2. Practice and answer the following questions:
  - 1. Could the code works on the Mars be successfully assembled on '**MinisysAv2.0**' ?
  - 2. While running the code of practice 1-2 and 1-3 on Mars, would them work as same on the 'tailored General purpose processor' as on Mars?

Optional:System Calls

- **System Calls** help- syscalls
  - **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	


lab1_introduction_of_toolkits
C:/Users/Administrator/Downloads/lab1_introduction_of_toolkits.pdf
Administrator\Desktop\mips1.asm - MARS 4.5
Settings Tools Help
Run speed at max (no interaction)
Code Basic Source
0x24020004 addiu $2, $0, 0x00000004 5 li $v0, 4
0x3c011001 lui $1, 0x00001001 6 la $a0, str
0x34240000 ori $4, $1, 0x00000000
0x0000000c syscall 7 syscall
0x2402000a addiu $2, $0, 0x0000000a 9 li $v0, 10
0x0000000c syscall 10 syscall
逐过单步执行.
可以在右侧看
Registers Coproc 1 Coproc 0
Name Number Value
$zero 0 0x00000000
$at 1 0x10010000
$v0 2 0x0000000a
$v1 3 0x00000000
$a0 4 0x10010000
$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 0x00400018
hi 0x00000000
lo 0x00000000
Value (+0) Value (+4) Value (+8) Value (+c) Value (+10) Value (+14) Value (+18) Value (+1c)
0x00636261 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x10010000 (.data)
Hexadecimal Addresses
Hexadecimal Values
ASCII
nished running --

7 JRE 1.8