实验三 CPU测试及汇编程序设计

实验目标

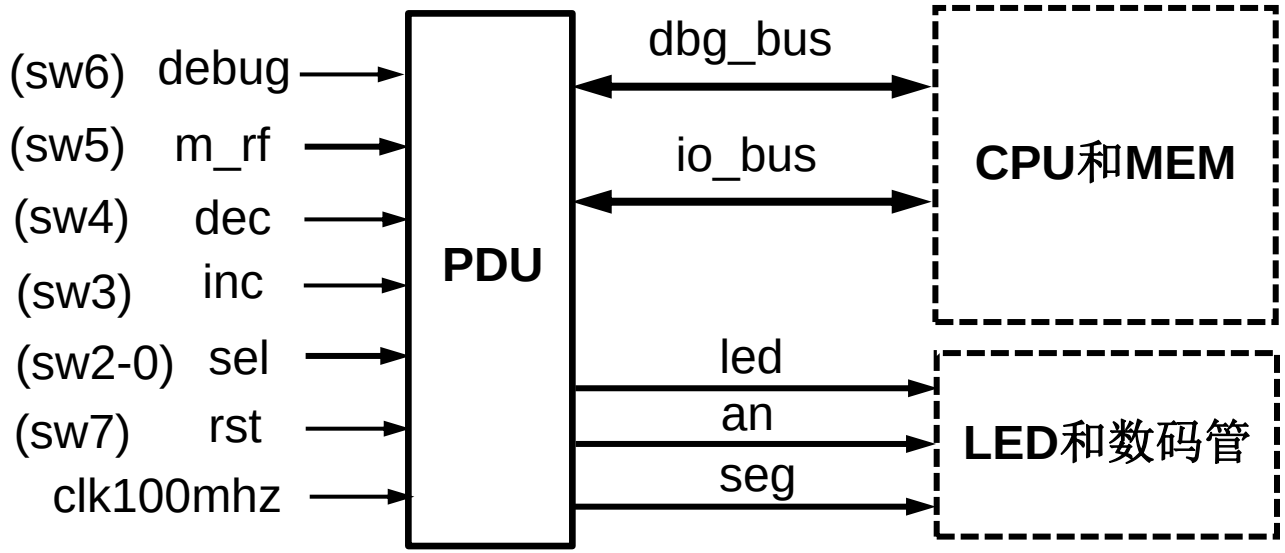
- **掌握CPU下载调试方法，以及测试数据 (COE文件) 的生成方法**
- **熟悉汇编程序的基本结构、仿真和调试的基本方法**
- **理解机器指令实现的基本原理（数据通路和控制器的协调工作过程）**

实验内容

1. 阅读RIPES示例汇编程序 (Console Printing)，单步执行程序，同时观察单周期CPU数据通路控制信号和寄存器内容的变化
2. 设计汇编程序：测试下列指令功能
   - sw, lw
   - add, addi
   - beq, jal
3. 设计汇编程序：计算斐波那契—卢卡斯数列
   - 依次输入数列开始两项（设置sw后按动button确认）
   - 按动button依次计算后续项，并输出至数码管上显示

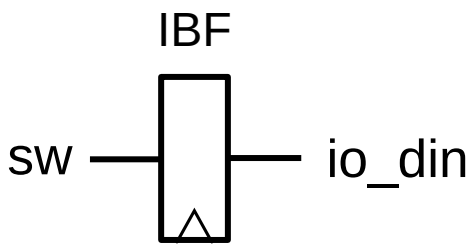
CPU调试单元

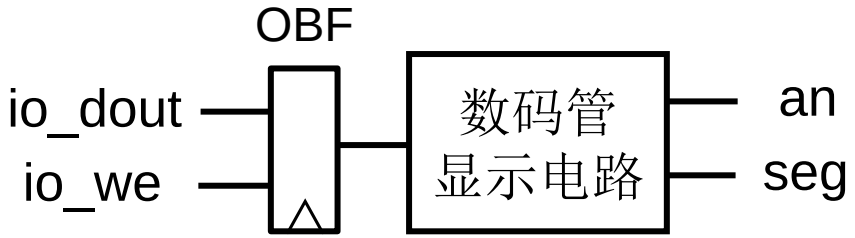
- **PDU：Processor Debug Unit，处理器调试单元**
  - 下载测试时，用于控制运行方式，显示运行状态和运行结果



IO_BUS信号

- **CPU运行时通过I/O信号访问开关(sw)和数码管(seg）**
  - io_din：CPU接收来自输入缓冲寄存器（IBF）的sw输入数据
  - io_dout：CPU向seg输出的数据
  - io_we：CPU向seg输出时的使能信号，利用该信号将io_dout存入输出缓冲寄存器（OBR），再经数码管显示电路将其显示在数码管（an，seg）
- **debug = 0：查看CPU的I/O输出结果**
  - 数码管显示io_dout，led可显示程序计数器(PC)





DBG_BUS信号

- **调试时将存储器和寄存器堆内容，以及CPU数据通路状态信息导出显示**
  - m_rf_addr：存储器(MEM)或寄存器堆(RF)的调试读口地址
  - rf_data：从RF读取的数据
  - m_data：从MEM读取的数据
  - pc_in：PC的输入数据
  - pc_out：PC的输出数据
  - instr：指令存储器的输出数据
  - rf_rd0：寄存器堆读口0的输出数据
  - rf_rd1：寄存器堆读口1的输出数据
  - rf_wd；寄存器堆写口的输入数据
  - alu_y：ALU的运算结果
  - ctrl:：控制器的控制信号

查看存储器和寄存器堆内容

- **debug = 1, sel = 0**
  - m_rf： 1，查看存储器(MEM)；0，查看寄存器堆(RF)
  - inc：m_rf_addr加1，查看下一个存储器或寄存器堆单元
  - dec：m_rf_addr减1，查看前一个存储器或寄存器堆单元
  - 8个LED指示灯显示m_rf_addr
  - 8个数码管显示rf_data/m_data

查看数据通路状态

- **debug = 1, sel = 1 ~ 7**
  - 8个数码管显示由sel选择的一个32位数据

    sel = 1：pc_in, PC的输入数据

    sel = 2：pc_out, PC的输出数据

    sel = 3：instr, 指令存储器的输出数据

    sel = 4：rf_rd0, 寄存器堆读口0的输出数据

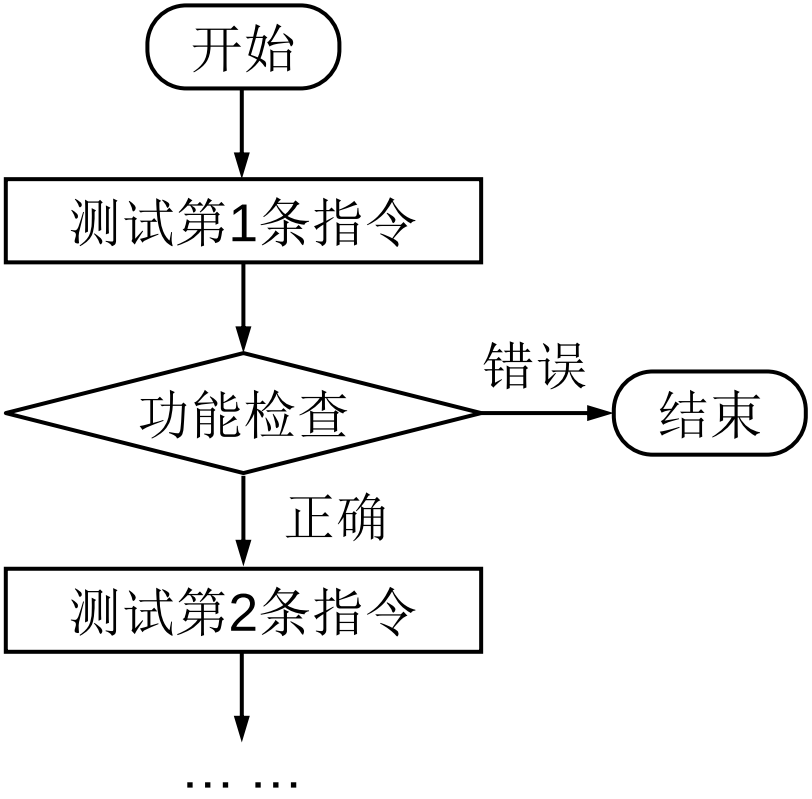
    sel = 5：rf_rd1, 寄存器堆读口1的输出数据

    sel = 6：rf_wd, 寄存器堆写口的输入数据

    sel = 7：alu_y, ALU的运算结果
  - 8个LED指示灯可显示控制器的控制信号

测试程序流程图

- **下载测试CPU功能**
- **功能检查**
  - 人工检查：将测试结果输出显示，正确则继续下条指令测试
  - 自动检查：测试结果自动判断
- **指令测试顺序**
  - 根据待测试指令与其他指令的依赖关系
  - 可以首先测试sw



示例：测试程序

```
.data
out: .word 0xff          #led, 初始全亮
in: .word 0              #switch

.text
la a0, out               #仿真需要
sw x0, 0(a0)             #test sw: 全灭led
addi t0, x0, 0xff        #test addi: 全亮led
sw t0, 0(a0)
lw t0, 4(a0)             #test lw: 由switch设置led
sw t0, 0(a0)
… …
```

RISC-V寄存器使用约定

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero 硬编码 0	—
x1	ra	Return address 返回地址	Caller
x2	sp	Stack pointer 栈指针	Callee
x3	gp	Global pointer 全局指针	—
x4	tp	Thread pointer 线程指针	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries 临时寄存器	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register 保存寄存器	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments 函数参数	Caller
x18–27	s2–11	Saved registers 保存寄存器	Callee
x28–31	t3–6	Temporaries 临时寄存器	Caller

RV32I指令类型

- 运算类
  - 算术：add, addi, sub, lui, auipc
  - 逻辑：and, or, xor
  - 移位(shift)：sll, srl, sra
  - 比较(set if less than)：slt, sltu

Category	Name	Fmt	RV32I Base	
Shifts	Shift Left Logical	R	SLL	rd,rs1,rs2
	Shift Left Log. Imm.	I	SLLI	rd,rs1,shamt
	Shift Right Logical	R	SRL	rd,rs1,rs2
	Shift Right Log. Imm.	I	SRLI	rd,rs1,shamt
	Shift Right Arithmetic	R	SRA	rd,rs1,rs2
	Shift Right Arith. Imm.	I	SRAI	rd,rs1,shamt
Arithmetic	ADD	R	ADD	rd,rs1,rs2
	ADD Immediate	I	ADDI	rd,rs1,imm
	SUBtract	R	SUB	rd,rs1,rs2
	Load Upper Imm	U	LUI	rd,imm
	Add Upper Imm to PC	U	AUIPC	rd,imm
Logical	XOR	R	XOR	rd,rs1,rs2
	XOR Immediate	I	XORI	rd,rs1,imm
	OR	R	OR	rd,rs1,rs2
	OR Immediate	I	ORI	rd,rs1,imm
	AND	R	AND	rd,rs1,rs2
	AND Immediate	I	ANDI	rd,rs1,imm
Compare	Set <	R	SLT	rd,rs1,rs2
	Set < Immediate	I	SLTI	rd,rs1,imm
	Set < Unsigned	R	SLTU	rd,rs1,rs2
	Set < Imm Unsigned	I	SLTIU	rd,rs1,imm

RV32I指令类型 (续1)

- 访存类
  - 加载(load)：lw, lb, lh, lbu, lhu
  - 存储(store)：sw, sb, sh
- 转移类
  - 分支(branch)：beq, bne, blt, bge, bltu, bgeu
  - 跳转(jump)：jal, jalr

Category	Name	Fmt	RV32I Base	
Branches	Branch =	B	BEQ	rs1,rs2,imm
	Branch ≠	B	BNE	rs1,rs2,imm
	Branch <	B	BLT	rs1,rs2,imm
	Branch ≥	B	BGE	rs1,rs2,imm
	Branch < Unsigned	B	BLTU	rs1,rs2,imm
	Branch ≥ Unsigned	B	BGEU	rs1,rs2,imm
Jump & Link	J&L	J	JAL	rd,imm
	Jump & Link Register	I	JALR	rd,rs1,imm
Loads	Load Byte	I	LB	rd,rs1,imm
	Load Halfword	I	LH	rd,rs1,imm
	Load Byte Unsigned	I	LBU	rd,rs1,imm
	Load Half Unsigned	I	LHU	rd,rs1,imm
	Load Word	I	LW	rd,rs1,imm
Stores	Store Byte	S	SB	rs1,rs2,imm
	Store Halfword	S	SH	rs1,rs2,imm
	Store Word	S	SW	rs1,rs2,imm

运算指令

- add rd, rs1, rs2 # x[rd] = x[rs1] + x[rs2]

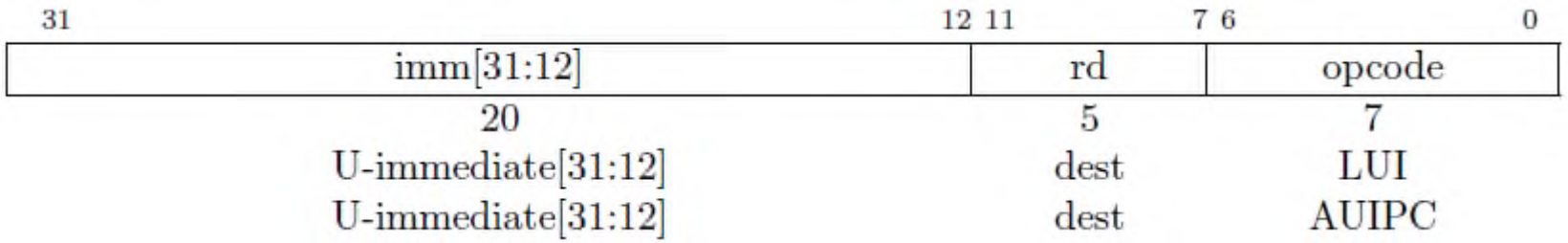
31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	src2	src1	ADD/SLT/SLTU	dest	OP
0000000	src2	src1	AND/OR/XOR	dest	OP
0000000	src2	src1	SLL/SRL	dest	OP
0100000	src2	src1	SUB/SRA	dest	OP

- addi rd, rs1, imm # x[rd] = x[rs1] + sext(imm)

31	20 19	15 14	12 11	7 6	0
imm[11:0]	rs1	funct3	rd	opcode	
12	5	3	5	7	
I-immediate[11:0]	src	ADDI/SLTI[U]	dest	OP-IMM	
I-immediate[11:0]	src	ANDI/ORI/XORI	dest	OP-IMM	

运算指令 (续1)

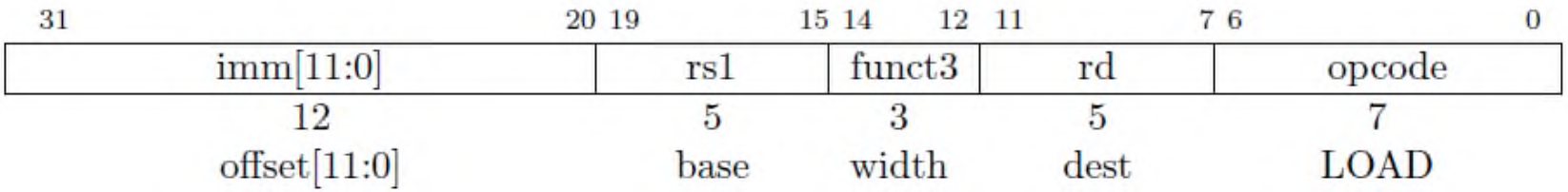
- lui rd, imm　　# x[rd] = sext(imm[31:12] << 12)
- auipc rd, imm　# x[rd] = pc + sext(imm[31:12] << 12)



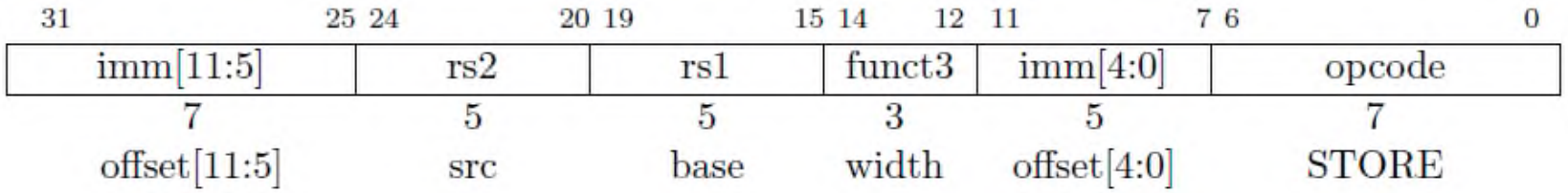
31 — 12	11 — 7	6 — 0
imm[31:12]	rd	opcode
20	5	7
U-immediate[31:12]	dest	LUI
U-immediate[31:12]	dest	AUIPC

访存指令

- lw rd, offset(rs1)　　# x[rd] = M[x[rs1] + sext(offset)]

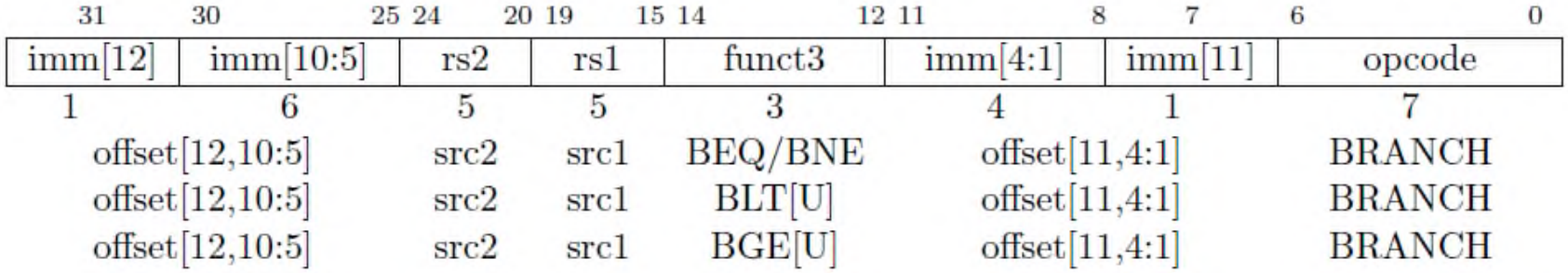


- sw rs2, offset(rs1)　　# M[x[rs1]+sext(offset)=x[rs2]



分支指令

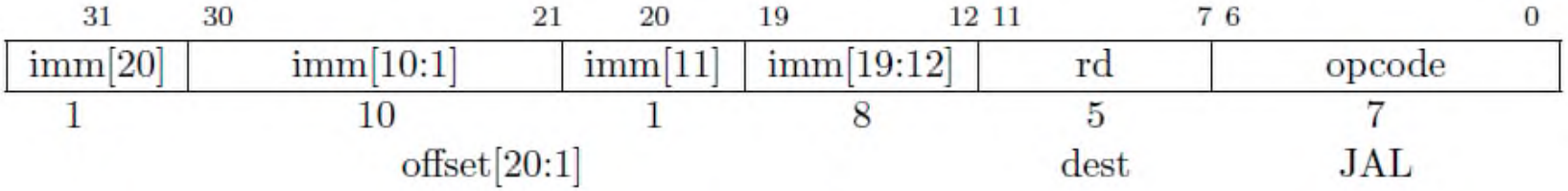
- beq rs1, rs2, offset # if (rs1 == rs2) pc += sext(offset)
- blt rs1, rs2, offset # if (rs1 $<_s$ rs2) pc += sext(offset)



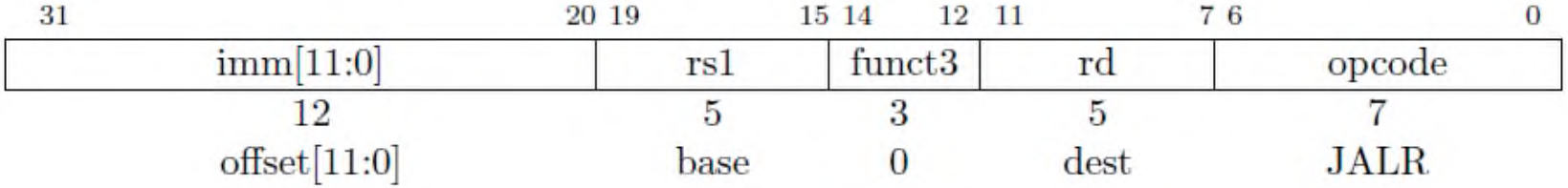
31	30 25	24 20	19 15	14 12	11 8	7	6 0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
1	6	5	5	3	4	1	7
offset[12,10:5]		src2	src1	BEQ/BNE	offset[11,4:1]		BRANCH
offset[12,10:5]		src2	src1	BLT[U]	offset[11,4:1]		BRANCH
offset[12,10:5]		src2	src1	BGE[U]	offset[11,4:1]		BRANCH

跳转指令

- jal rd, offset　　# x[rd] = pc+4; pc += sext(offset)



- jalr rd, offset(rs1)　# t =pc+4; pc=(x[rs1]+sext(offset))&~1; x[rd]=t



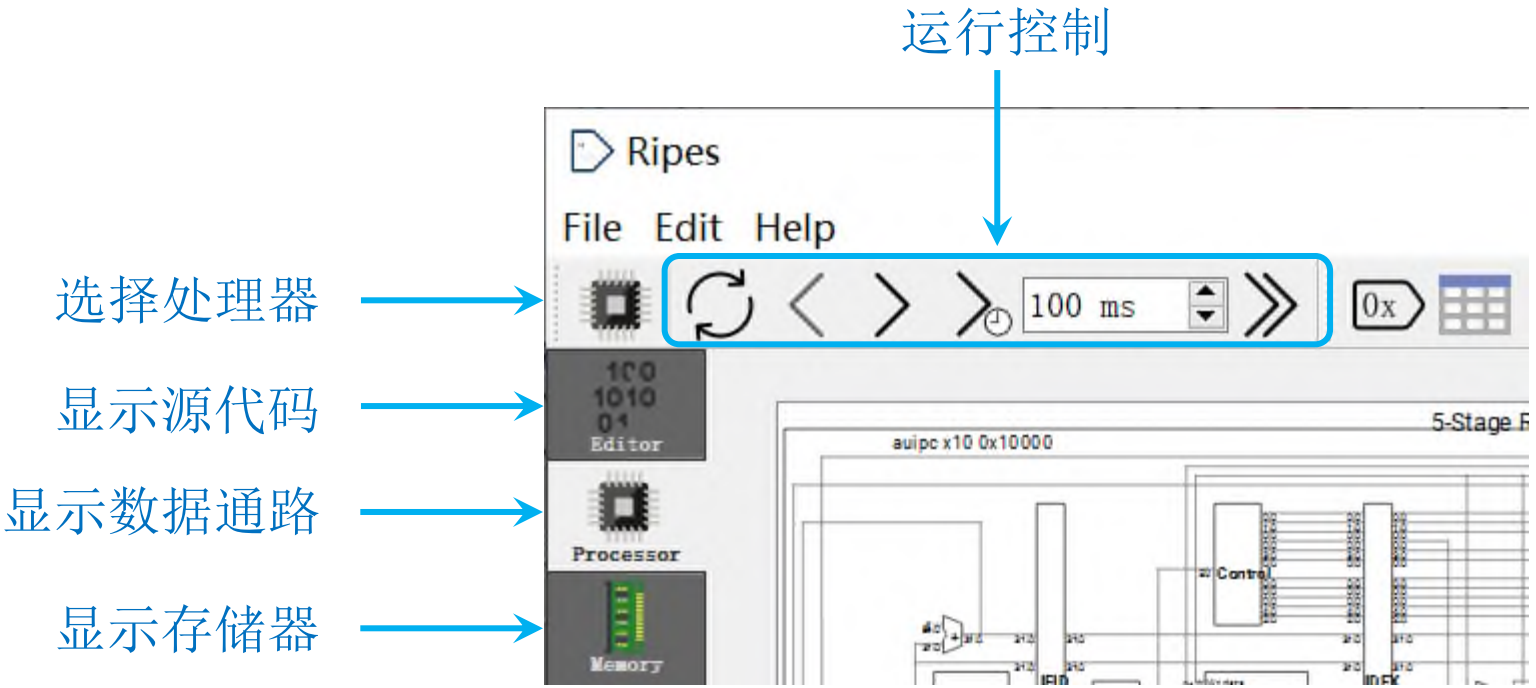
汇编指示和伪指令

- 汇编指示符（Assembler directives）
  - .data, .text
  - .word, .half, .byte, .string
  - .equ
  - .align
  - ……
- 伪指令
  - li, la, mv
  - nop, not, neg
  - j, jr, call, ret
  - ……

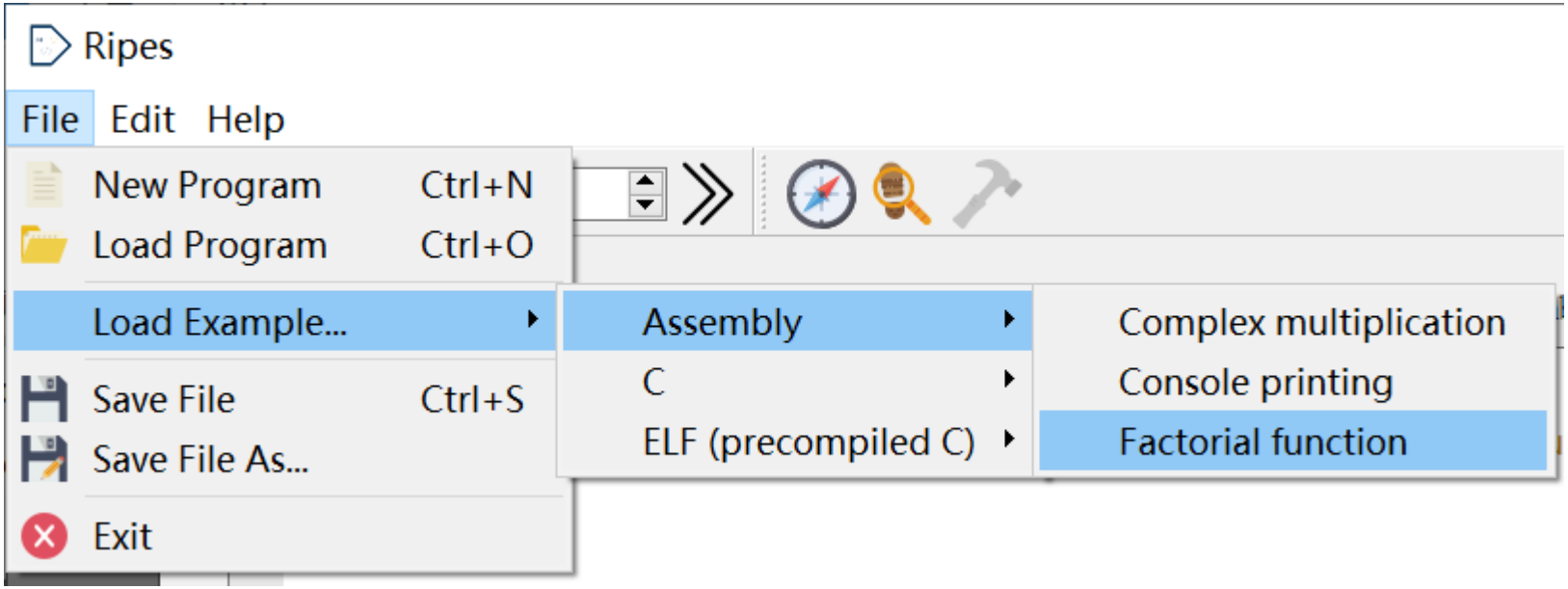
```
.equ CONSTANT, 0xdeadbeef
li a0, CONSTANT

# lui a0,0xdeadc
# addi a0,a0,-273
```

RIPES

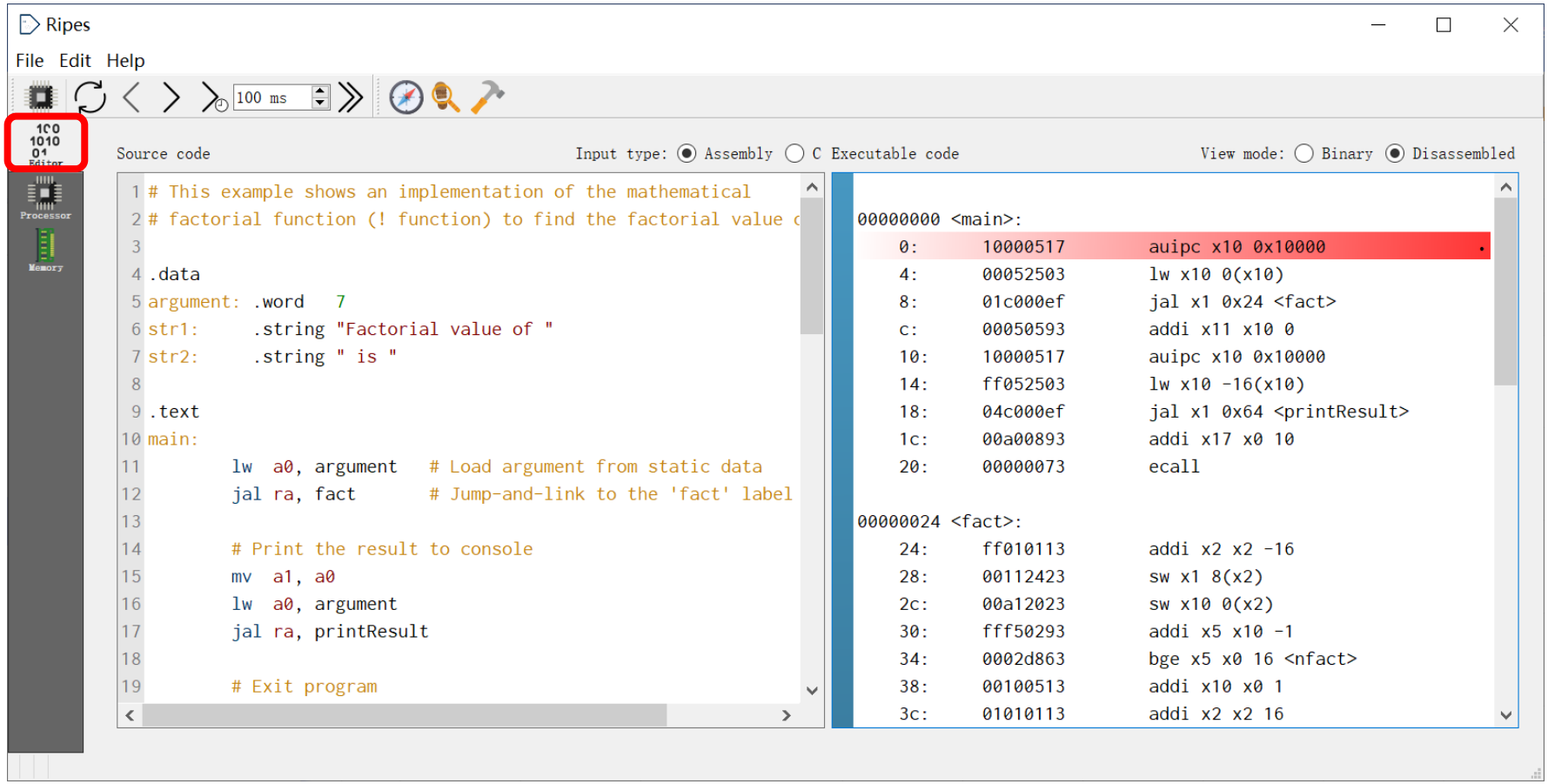
- 图形化的RISC-V模拟器



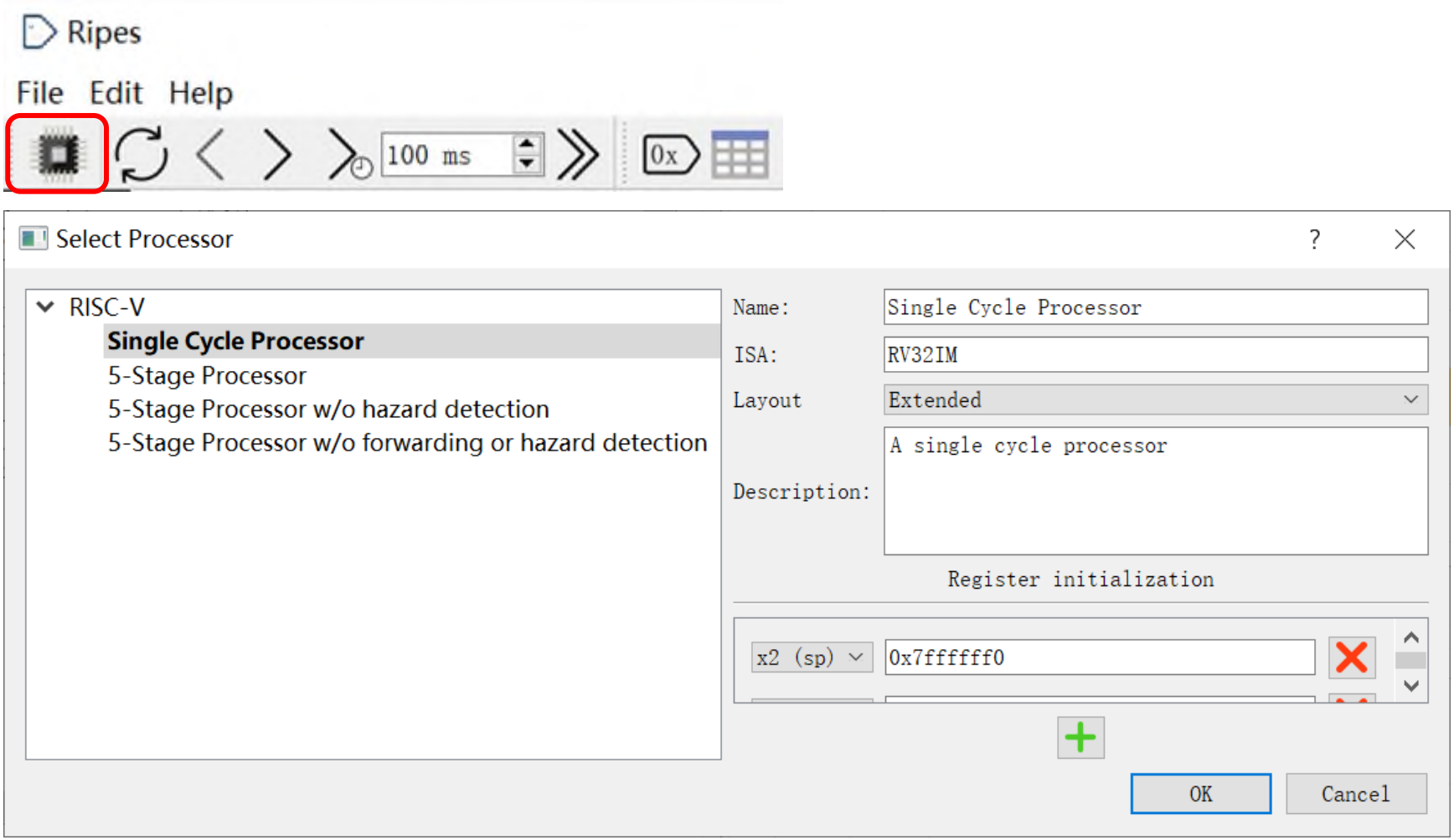
加载/编辑源程序


Ripes
File
Edit
Help
New Program Ctrl+N
Load Program Ctrl+O
Load Example...
Save File Ctrl+S
Save File As...
Exit
Assembly
C
ELF (precompiled C)
Complex multiplication
Console printing
Factorial function

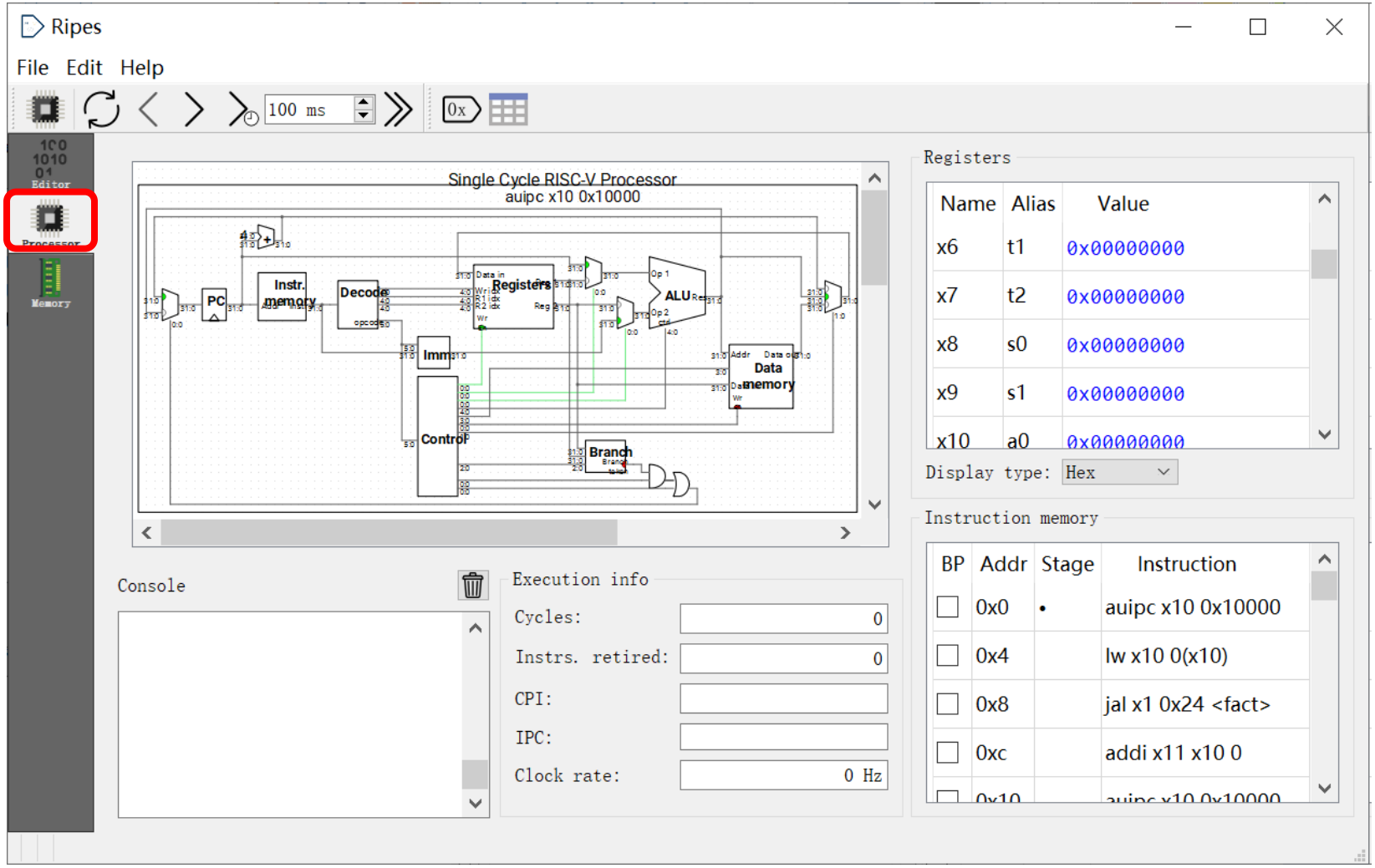
源程序界面


Ripes
File Edit Help
100 ms
Editor
Processor
Memory
Source code
Input type: Assembly C Executable code
View mode: Binary Disassembled
1 # This example shows an implementation of the mathematical
2 # factorial function (! function) to find the factorial value
3
4 .data
5 argument: .word 7
6 str1: .string "Factorial value of "
7 str2: .string " is "
8
9 .text
10 main:
11 lw a0, argument # Load argument from static data
12 jal ra, fact # Jump-and-link to the 'fact' label
13
14 # Print the result to console
15 mv a1, a0
16 lw a0, argument
17 jal ra, printResult
18
19 # Exit program
00000000 <main>:
0: 10000517 auipc x10 0x10000
4: 00052503 lw x10 0(x10)
8: 01c000ef jal x1 0x24 <fact>
c: 00050593 addi x11 x10 0
10: 10000517 auipc x10 0x10000
14: ff052503 lw x10 -16(x10)
18: 04c000ef jal x1 0x64 <printResult>
1c: 00a00893 addi x17 x0 10
20: 00000073 ecall
00000024 <fact>:
24: ff010113 addi x2 x2 -16
28: 00112423 sw x1 8(x2)
2c: 00a12023 sw x10 0(x2)
30: fff50293 addi x5 x10 -1
34: 0002d863 bge x5 x0 16 <nfact>
38: 00100513 addi x10 x0 1
3c: 01010113 addi x2 x2 16

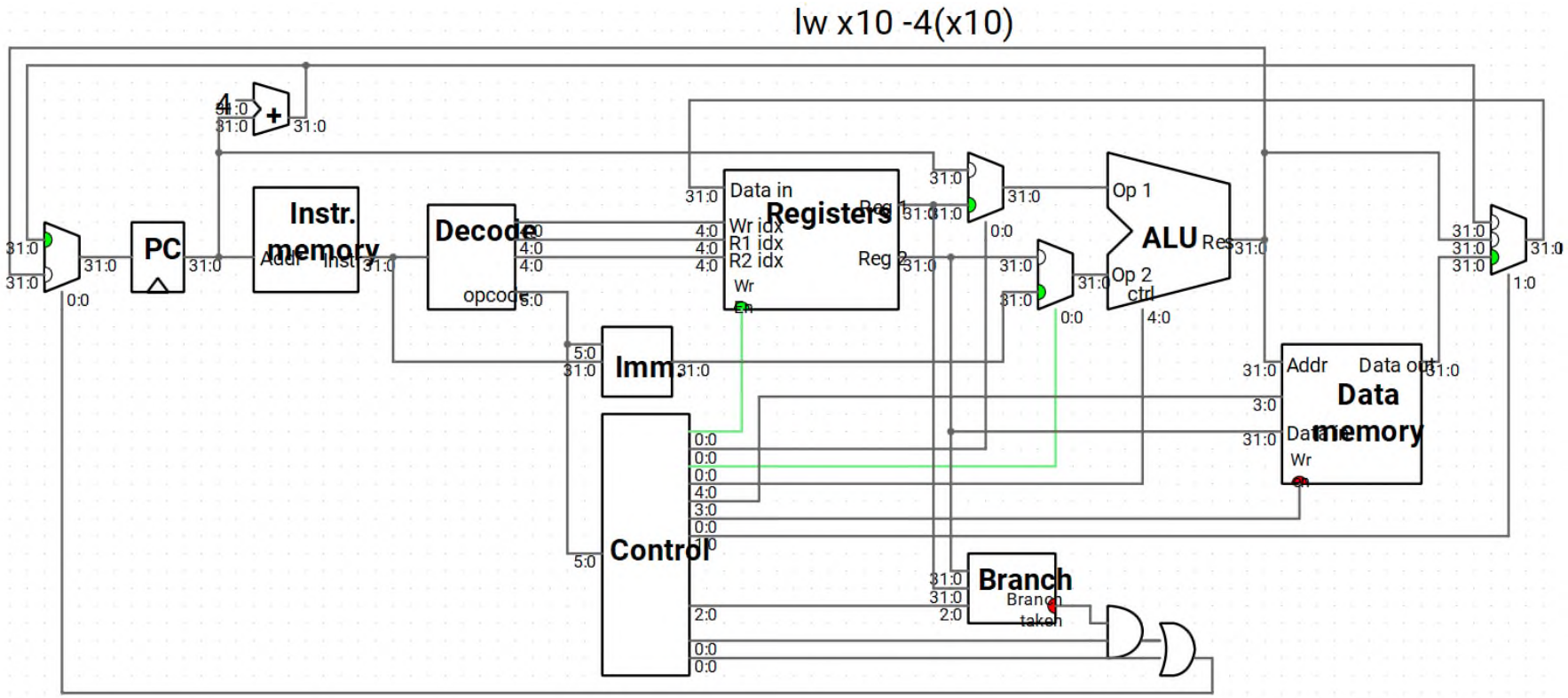
选择CPU结构


Ripes
File Edit Help
100 ms
0x
Select Processor
RISC-V
Single Cycle Processor
5-Stage Processor
5-Stage Processor w/o hazard detection
5-Stage Processor w/o forwarding or hazard detection
Name:
Single Cycle Processor
ISA:
RV32IM
Layout
Extended
Description:
A single cycle processor
Register initialization
x2 (sp)
0x7ffffff0
OK
Cancel

单周期CPU

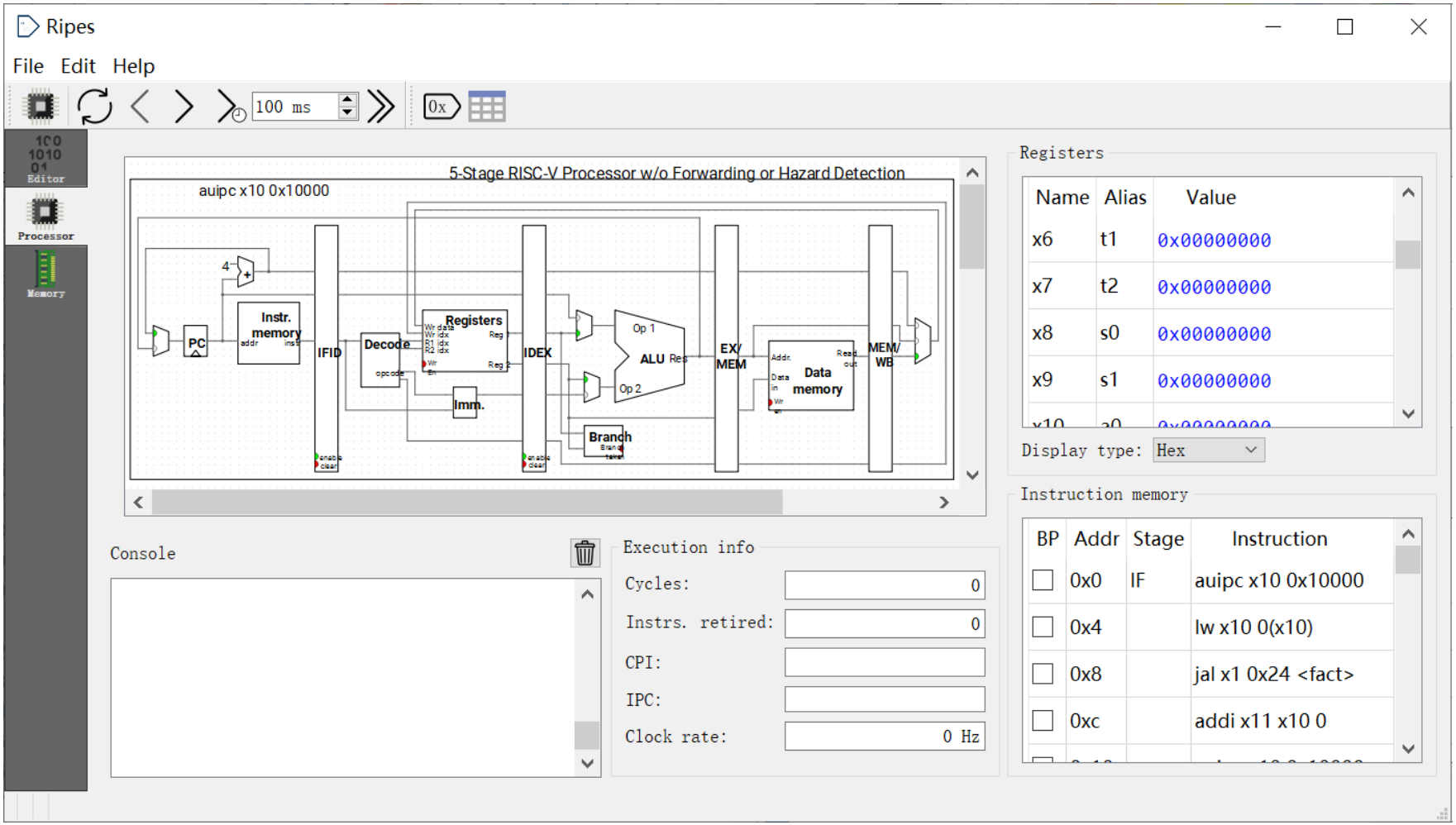

Ripes
File Edit Help
100 ms
0x
Editor
Processor
Memory
Single Cycle RISC-V Processor
auipc x10 0x10000
PC
Instr. memory
Decoder
Registers
ALU
Imm
Control
Branch
Data memory
Registers
Name Alias Value
x6 t1 0x00000000
x7 t2 0x00000000
x8 s0 0x00000000
x9 s1 0x00000000
x10 a0 0x00000000
Display type: Hex
Instruction memory
BP Addr Stage Instruction
0x0 • auipc x10 0x10000
0x4 lw x10 0(x10)
0x8 jal x1 0x24 <fact>
0xc addi x11 x10 0
Console
Execution info
Cycles: 0
Instrs. retired: 0
CPI:
IPC:
Clock rate: 0 Hz

单周期CPU数据通路

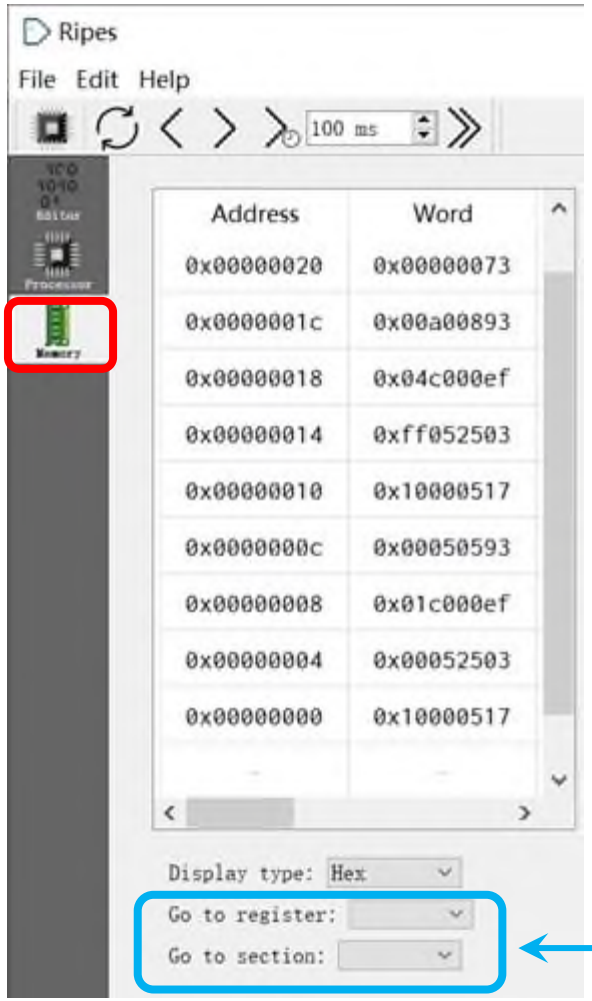


注：绿色表示有效信号；“ctrl + 滚轮”可缩放数据通路图

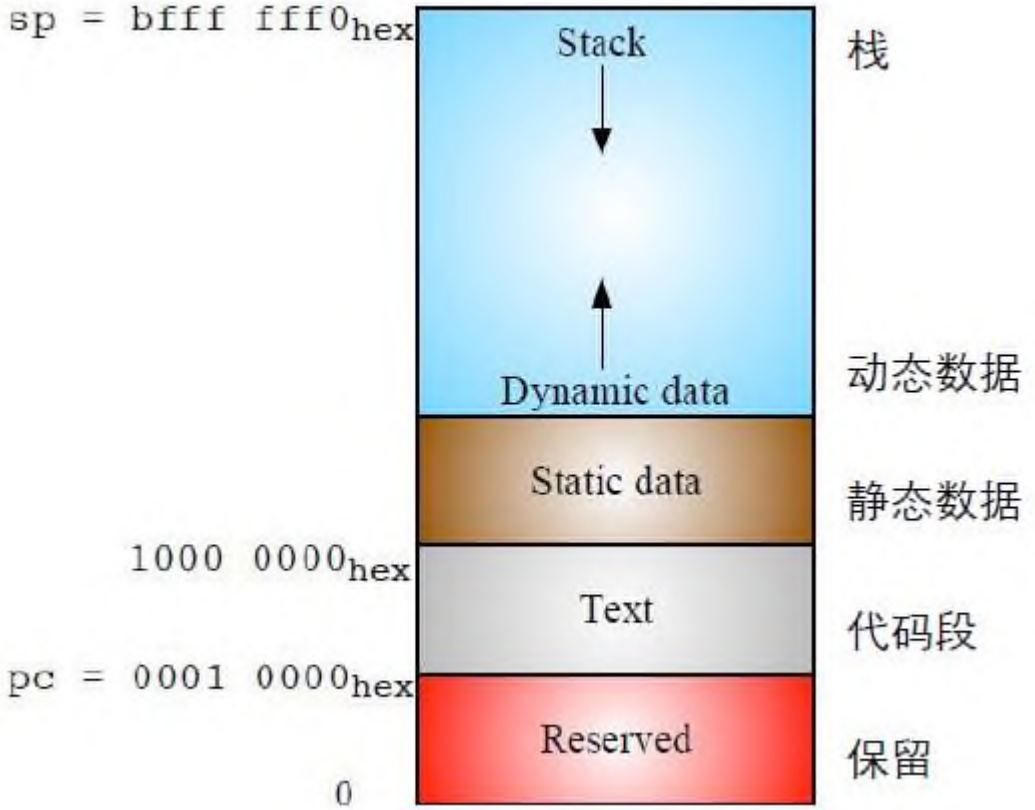
流水线CPU


Ripes
File Edit Help
100 ms
0x
Editor
Processor
Memory
5-Stage RISC-V Processor w/o Forwarding or Hazard Detection
auipc x10 0x10000
PC
Instr. memory
IFID
Decode
Registers
Imm.
IDEX
Op 1
ALU
Op 2
Branch
EX/ MEM
Data memory
MEM/ WB
Registers
Name Alias Value
x6 t1 0x00000000
x7 t2 0x00000000
x8 s0 0x00000000
x9 s1 0x00000000
Display type: Hex
Instruction memory
BP Addr Stage Instruction
0x0 IF auipc x10 0x10000
0x4 lw x10 0(x10)
0x8 jal x1 0x24 <fact>
0xc addi x11 x10 0
Console
Execution info
Cycles: 0
Instrs. retired: 0
CPI:
IPC:
Clock rate: 0 Hz

存储器空间分配



存储器定位



RARS 软件汇编程序转coe文件

1.运行汇编程序

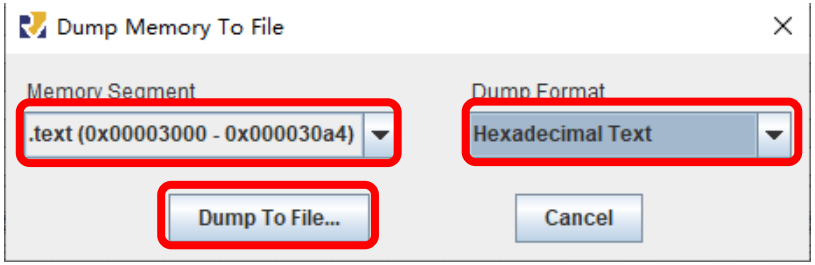
Run>>Assemble

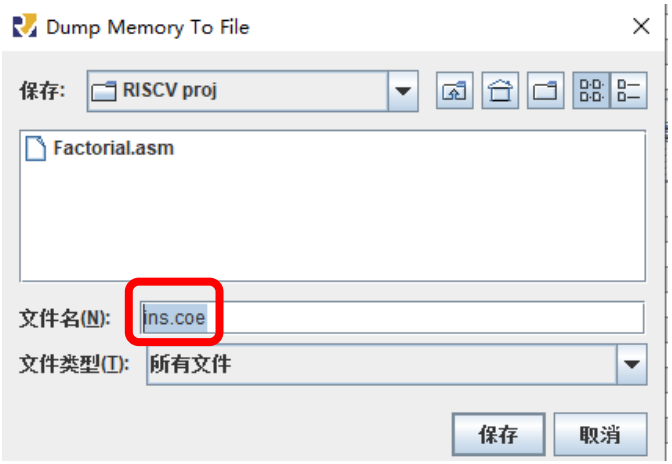
2.存储器设置

Settings>>Memory Configuration>>compact, data at Address0>>Apply and Close

3.代码段机器码(16进制)导出

File>>Dump Memory To File，按下图设置完毕后，选择文件保存路径，命名为ins.coe。

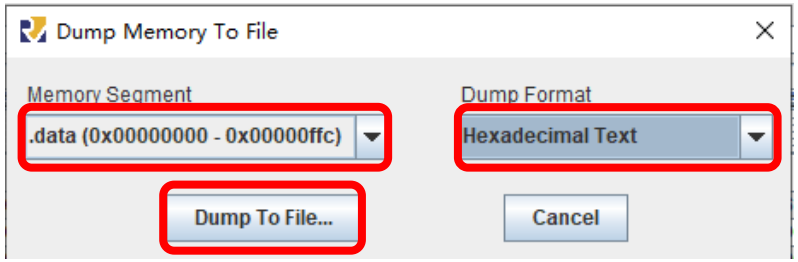


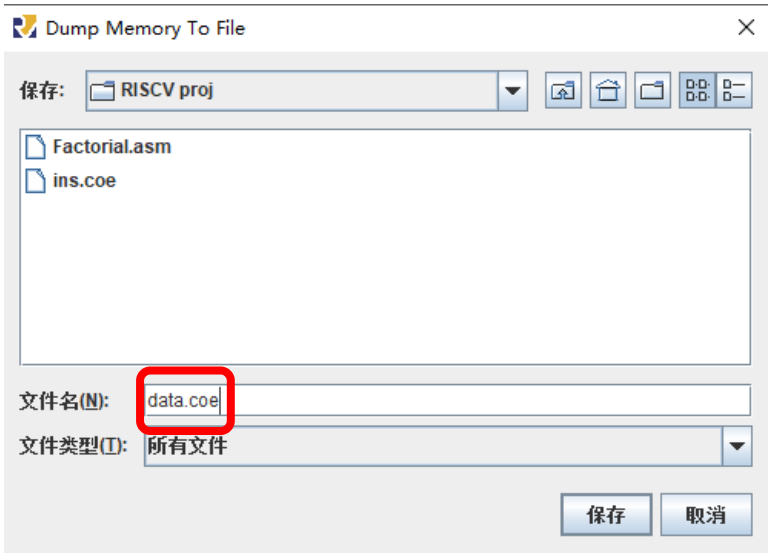


RARS 软件汇编程序转coe文件

4.数据段机器码(16进制)导出

File>>Dump Memory To File，按下图设置完毕后，选择文件保存路径，命名为data.coe。





5. 采用记事本分别打开生成的ins.coe和data.coe，在文档的最开始加上以下语句后保存：memory_initialization_radix = 16;

memory_initialization_vector =

实验步骤

1. **仿真RIPES示例汇编程序 (Console Printing)**
2. **设计汇编程序，实现人工检查6条指令功能，并生成COE文件**
3. **设计汇编程序，实现计算斐波那契—卢卡斯数列，并生成COE文件**

The End