

《计算机组成原理实验》

2020年春季

武汉大学计算机学院 蔡朝晖

目录

- 实验目的&实验内容
- 设计方法
- Verilog语言
- ModelSim使用及示例分析
- Mars使用

实验目的

- 融会贯通计算机组成原理课程所教授的知识，通过对知识的综合应用，加深对CPU系统各模块的工作原理及相互联系的认识。
- 学习采用EDA (Electronic Design Automation) 技术设计MIPS单周期CPU/流水线CPU的技术与方法。
- 培养科学研究的独立工作能力，取得CPU设计与仿真的实践和经验。
- 了解SOC系统，并在FPGA开发板上实现简单的SOC系统。

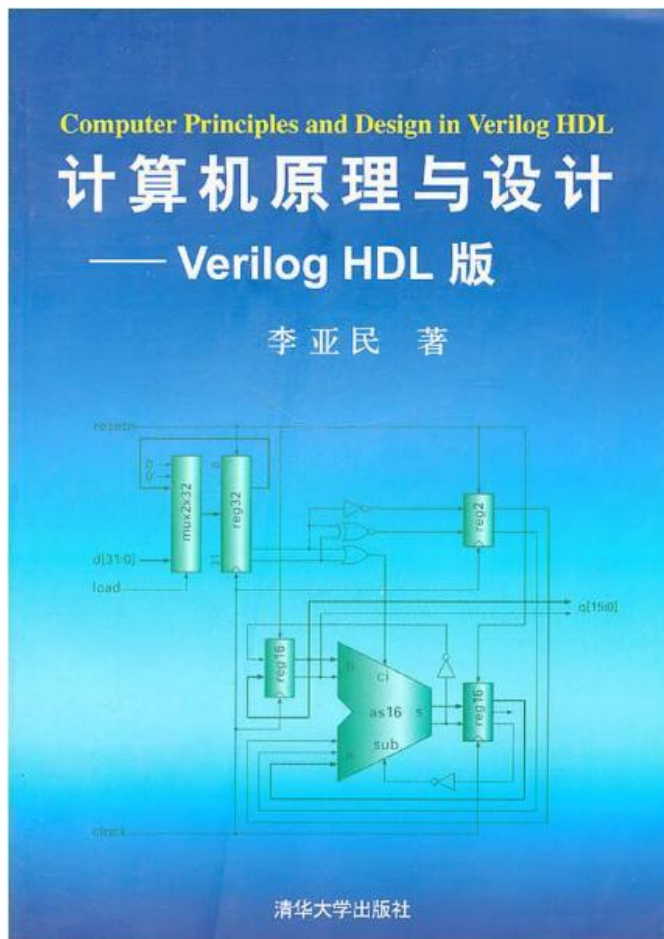
参考示例说明

- 单周期CPU，支持15条指令
 - add/sub/and/or/slt/sltu/addu/subu
 - addi/ori/lw/sw/beq
 - j/jal (将比较有无jal指令的两个scpu的区别)

实验内容一——单周期CPU

- 对示例单周期CPU进行指令扩展，至少支持以下指令（红色和蓝色为增加的指令）
 - add/sub/and/or/slt/sltu/addu/subu
 - addi/ori/lw/sw/beq
 - j/jal
 - xor/xori/addiu/andi/nor
 - sll/sra/srav/lui/slti/sltiu/srl/sllv/srlv/jr/jalr
 - lb/lh/lbu/lhu/sb/sh（数据在内存中以小端形式存储little endian）
 - bne/blez/bltz/bgtz/bgez

实验参考书



目录

- 实验目的&实验内容&考核方式
- 设计方法
- Verilog语言
- ModelSim使用及示例分析
- Mars使用
- Nexys DDR开发版介绍
- Vivado使用及示例分析
- 学号排序实验具体要求
- 一些提醒

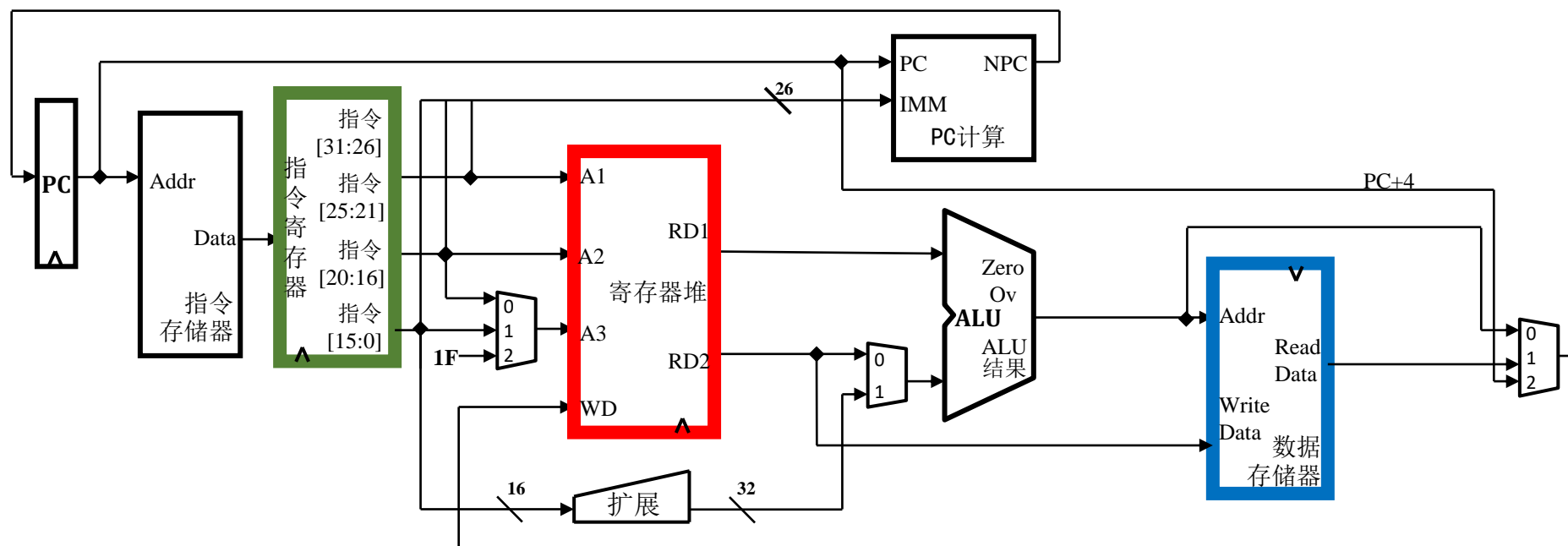
方法概述

- 数据通路设计：解决静态拓扑
 - 功能部件，即子系统的功能设计
 - 功能部件间的静态连接关系
- 控制器设计：解决动态执行
 - 选择信息流的动态路径，核心问题是解决MUX的选择
 - 在正确的时间给出正确的控制信号取值，决定功能部件的功能

设计方法

1. 阅读项目说明，了解项目目标
2. 阅读MIPS指令手册，了解待实现指令的RTL描述
3. 数据通路设计
 - 根据RTL，设计出每条指令的分数据通路
 - 将所有分数据通路合成完整的数据通路
 - 关键：综合出需要的MUX
4. 控制器设计
 - 根据RTL，给出每条指令控制信号真值表
 - 将所有指令的控制信号真指表合成完整真值表

单周期CPU数据通路

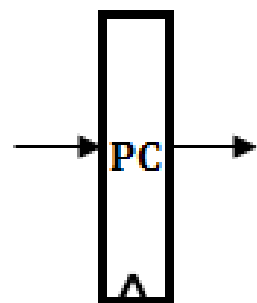


功能部件

- 数据通路是以功能部件为基础构造的
 - 数据通路：系统
 - 功能部件：子系统
- 构造数据通路
 - Step0：以功能部件为基础
 - Step1：给出执行每条指令所需部件的连接关系
 - Step2：综合出执行指令集所需部件的连接关系
- 功能部件建模
 - 外部特性：数据通路设计关心的
 - 内部实现：基本与数据通路设计无关

功能部件建模：PC

- 程序计数器
- 功能与控制

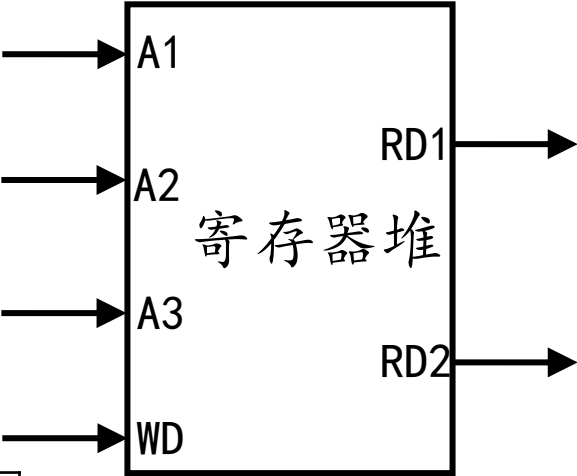


- PCWr 决定写入（单周期CPU每个周期都写入PC，该信号可省略或为1）

输入	NPC[31:0] PCWr, clk, rst	
输出	PC[31:0]	
数据结构	32位寄存器	
行为	功能	操作
	异步复位	rst上升沿时, if rst then PC← 32' h00000000
	同步加载	clk上升沿时: if PCWr then PC ← NPC

功能部件建模：寄存器堆

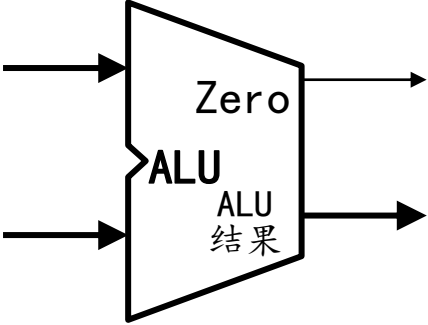
- 功能与控制
 - 读出：不需要控制
 - 写入：RFWr 为1



输入	A1 [4:0]、A2 [4:0]、A3 [4:0]、WD [31:0] RFWr、clk	
输出	RD1 [31:0]、RD2 [31:0]	
数据结构	rf [31:0]，32个32位寄存器	
行为	功能	操作
	读出寄存器值	RD1←rf [A1]；RD2←rf [A2]
	写入寄存器值	clk上升沿时 if (RFWr) then rf [A3]←WD

功能部件建模：ALU

- 功能与控制
 - ALUOp[2:0] 决定执行何种计算
- Zero: 判断A是否等于B



输入	A[31:0], B[31:0], ALUOp[2:0]		
输出	C[31:0], Zero		
行为	ALUOp	功能	操作
	000	NOP	$C \leftarrow A$
	001	ADD	$C \leftarrow A + B$
	010	SUB	$C \leftarrow A - B$
	011	AND	$C \leftarrow A \& B$
	100	OR	$C \leftarrow A \mid B$
	101	SLT	$C \leftarrow (A < B) ? 32'd1 : 32'd0$
	110	SLTU	$C \leftarrow (\{1'b0, A\} < \{1'b0, B\}) ? 32'd1 : 32'd0$
	others	SLTU	$C \leftarrow A$
	---	A等于B?	$Zero \leftarrow (C == 32'b0) ? 1 : 0$

功能部件及其控制信号使用约束

- RF、DM：需要写使能
 - RFWr/DMWr：只能在特定时间有效，其他时间则必须无效
- 注意时钟沿与状态间的关系
 - 寄存器的值：上升沿前准备，上升沿写入

推荐工作流程

- 安装软件
- 学习软件基本使用方法
- 根据给定的代码，学习Verilog
- 并行地设计你的系统
- 单指令调试，修改设计，调试……
- 根据测试用例调试，修改设计，调试……

资料说明

- \source\SCPU 是一个简单的单周期CPU实现（支持jal指令），
 \source\SIMTop\是ModelSim仿真需要的文件，
 \source\FPGATop\是Vivado运行需要的文件，
 \source 下还包含其他文件。
- \Verilog教程 目录下的“计算机原理与设计：Verilog HDL版.pdf”既是Verilog教程，也是CPU设计教程。
- \Nexys4DDR 目录下的“Nexys4-DDR_rm.pdf”是Nexys4DDR FPGA板的手册。
- \CoDWork\SCPUSIM\ 是 ModelSim示例工程（支持jal指令）
- \CoDWork\SCPUSIM_woJAL\ 是 ModelSim示例工程（不支持jal指令）
- \CoDWork\SCPU_SOC\ 是 Vivado示例工程（支持jal指令）

资料说明

- \source\SCPU 是一个简单的单周期CPU实现（支持jal指令），
 \source\SIMTop\是ModelSim仿真需要的文件，
 \source\FPGATop\是Vivado运行需要的文件，
 \source 下还包含其他文件。
- \Verilog教程 目录下的“计算机原理与设计：Verilog HDL版.pdf”既是Verilog教程，也是CPU设计教程。
- \Nexys4DDR 目录下的“Nexys4-DDR_rm.pdf”是Nexys4DDR FPGA板的手册。
- \CoDWork\SCPUSIM\ 是 ModelSim示例工程（支持jal指令）
- \CoDWork\SCPUSIM_woJAL\ 是 ModelSim示例工程（不支持jal指令）
- \CoDWork\SCPU_SOC\ 是 Vivado示例工程（支持jal指令）

目录

- 实验目的&实验内容&考核方式
- 设计方法
- Verilog语言
- ModelSim使用及示例分析
- Mars使用
- Nexys DDR开发版介绍
- Vivado使用及示例分析
- 学号排序实验具体要求
- 一些提醒

Verilog语言

- Verilog HDL是一种硬件描述语言，Verilog HDL语言不仅定义了语法，而且对每个语法结构都定义了清晰的模拟、仿真语义。
- 用这种语言编写的模型能够使用Verilog仿真器进行验证。
- Verilog HDL中有两类数据类型：网线wire数据类型和寄存器reg数据类型。网线类型表示构件间的物理连线，而寄存器类型表示抽象的数据存储元件。
- Verilog HDL 还具有内置逻辑函数，例如&（按位与）和|（按位或）。
- 对高级编程语言结构，例如条件语句、情况语句和循环语句，语言中都可以使用。

网线与寄存器

网线与寄存器可以是位、向量或数组。

- `wire a;` // Simple wire
- `reg[-1:4] vec;` // Six-bit register
- `integer imem[0:1023];` // Array of 1024 integers
- `reg[31:0] dcache[0:63];` // A 32-bit memory

模块

- 模块是Verilog 的基本描述单位，用于描述某个设计的功能或结构及其与其他模块通信的外部端口，一个模块可以在另一个模块中使用。
- 一个模块的基本语法如下：

```
module HalfAdder(A, B, Sum, Carry);  
    input A, B;  
    output Sum, Carry;  
    assign #2 Sum = A ^ B;  
    assign #5 Carry = A & B;  
endmodule
```

- 模块的名字是HalfAdder。模块有4个端口：两个输入端口A和B，两个输出端口Sum和Carry。由于没有定义端口的位数，所有端口大小都为1位；同时，由于没有各端口的数据类型说明，这四个端口都是网线wire数据类型。

模块实例化

- 声明模块

```
module mymod(y, a, b);
```

- 实例化

```
mymod mm1(y1, a1, b1);    //connect by position
```

```
mymod (y2, a2, b2),
```

```
      (y3, a3, b3);        //instance name omitted
```

```
mymod mm2(. a(a4), . b(b4), . y(c4)); //connect by name
```

时延

- Verilog HDL模型中的所有时延都根据时间单位定义。下面是带时延的连续赋值语句实例。

```
`timescale 1ns/100ps
```

- 此语句说明时延时间单位为1ns并且时间精度为100ps（时间精度是指所有的时延必须被限定在0.1ns内）。

```
assign #2 Sum = A ^ B;
```

- #2指2个时间单位。#2代表2ns。
- #5.22是指5.2ns，#5.27是指5.3ns。

行为描述方式

设计的行为功能使用下述过程语句结构描述：

- 1) `initial`语句：此语句只执行一次。
- 2) `always`语句：此语句总是循环执行，或者说此语句重复执行；用于描述时序逻辑

`always@`(敏感事件列表) 敏感事件上升沿`posedge`，下降沿`negedge`，或电平

- 敏感事件列表中可以包含多个敏感事件，但不可以同时包括电平敏感事件和边沿敏感事件，也不可以同时包括同一个信号的上升沿和下降沿，这两个事件可以合并为一个电平敏感事件。
- 在`verilog2001`中“，”和“`or`”都可以用来分割敏感事件了，可以用“*”代表所有输入信号，这可以防止遗漏。

always@ (敏感事件列表)

- 合法的写法:

always@ *

always@ (posedge clk1, negedge clk2)

always@ (a or b)

`timescale 100ns/100ns //定义仿真基本周期为100ns

always #1 clk=~clk // #1代表一个仿真周期即100ns

连续赋值

- 便于描述逻辑或数据通路
- 连续赋值语句的语法为：

```
assign [delay] LHS_net = RHS_expression ;
```

右边表达式使用的操作数无论何时发生变化，右边表达式都重新计算，并且在指定的时延后变化值被赋予左边表达式的线网变量。时延定义了右边表达式操作数变化与赋值给左边表达式之间的持续时间。如果没有定义时延值，缺省时延为0。

```
wire [8:0] sum;
```

```
wire [7:0] a, b;
```

```
wire carryin;
```

```
assign sum = a + b + carryin;
```

//连续赋值，当a，b或carryin改变时自动重新计算

系统任务和函数

- 使用方式：\$<identifier>
- \$符号指示这是系统任务和函数
- 系统函数有很多，如：
 - 返回当前仿真时间\$time
 - 显示/监视信号值 (\$display, \$monitor)
 - 停止仿真\$stop
 - 结束仿真\$finish
 - 例如：\$monitor(\$time, “a=%b, b=%h”, a, b);
当信号a或b的值发生变化时，系统任务\$monitor显示当前仿真时间，信号a的值（二进制格式），信号b的值（16进制格式）。

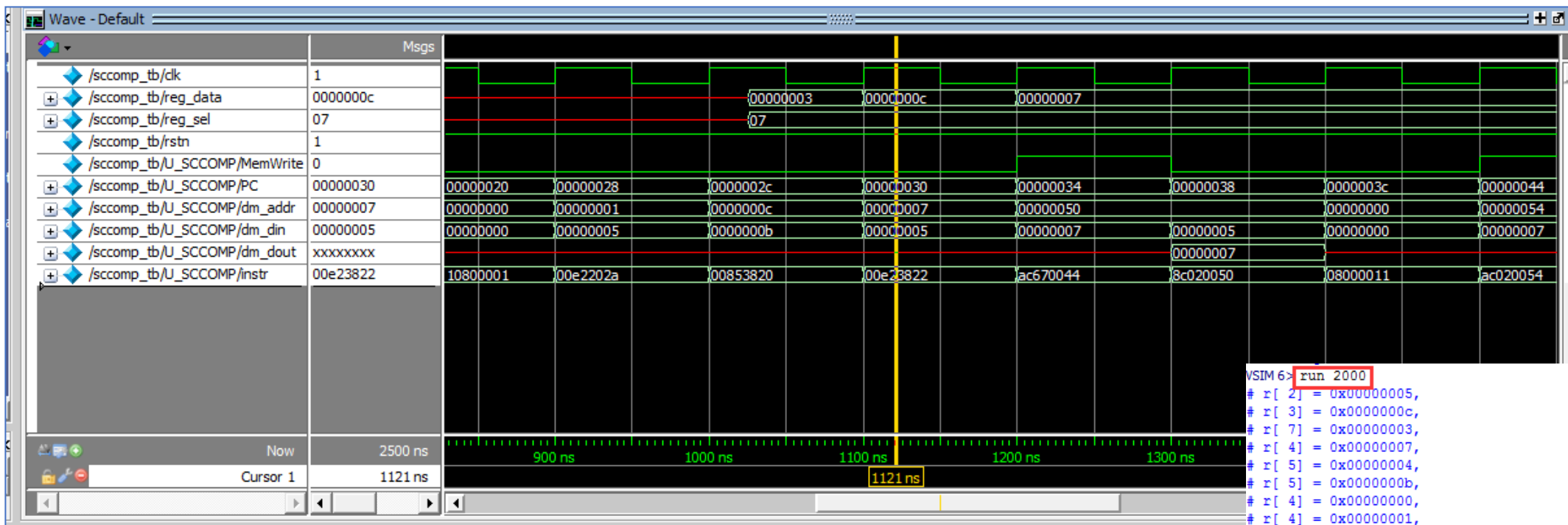
目录

- 实验目的&实验内容&考核方式
- 设计方法
- Verilog语言
- ModelSim使用及示例分析
- Mars使用

ModelSim使用

- Mentor公司的ModelSim是业界最优秀的HDL语言仿真软件，它能提供友好的仿真环境，是业界唯一的单内核支持VHDL和Verilog混合仿真的仿真器。它采用直接优化的编译技术、Tcl/Tk技术、和单一内核仿真技术，编译仿真速度快，编译的代码与平台无关，便于保护IP核，个性化的图形界面和用户接口，为用户加快调错提供强有力的手段，是FPGA/ASIC设计的首选仿真软件。
- 注意ModelSim的安装路径不能含有中文。
- ModelSim教程
 - Modelsim详细使用教程（一看就会）.pdf
 - Modelsim精选入门教程.pdf

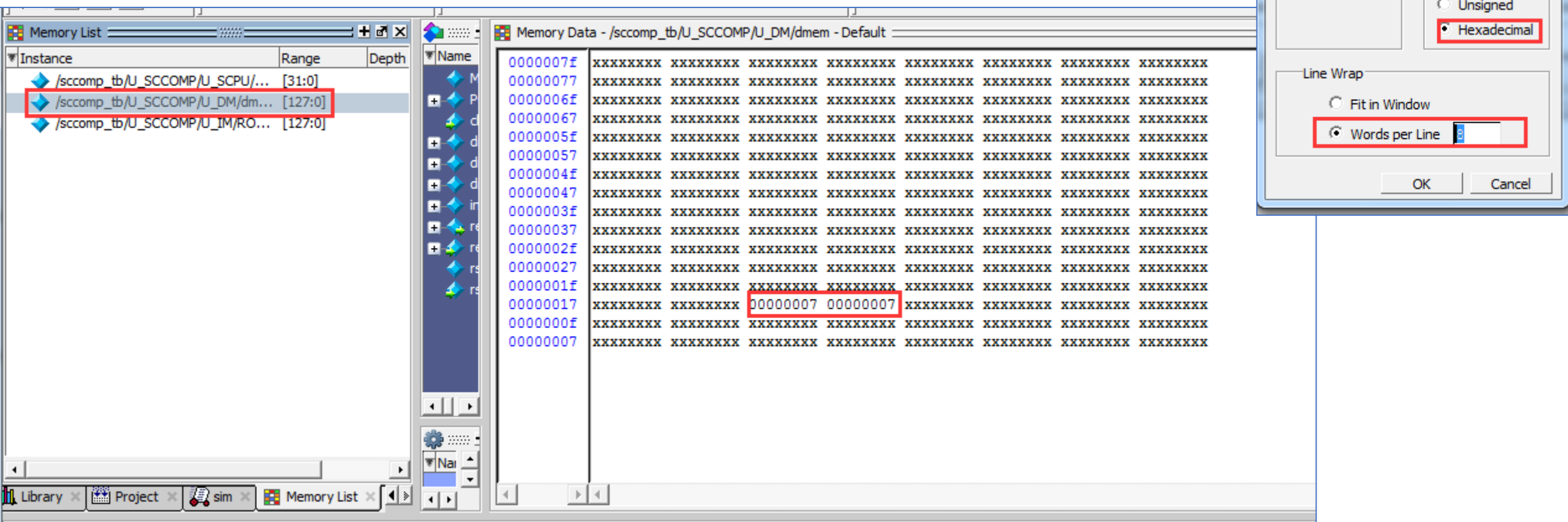
运行仿真



```
VSIM 6> run 2000
# r[ 2] = 0x00000005,
# r[ 3] = 0x0000000c,
# r[ 7] = 0x00000003,
# r[ 4] = 0x00000007,
# r[ 5] = 0x00000004,
# r[ 5] = 0x0000000b,
# r[ 4] = 0x00000000,
# r[ 4] = 0x00000001,
# r[ 7] = 0x0000000c,
# r[ 7] = 0x00000007,
# dmem[0x 50] = 0x00000007,
# r[ 2] = 0x00000007,
# r[31] = 0x00000040,
# dmem[0x 54] = 0x00000007,
# Break in Module scomp_tb at D:/CoDWork/SCPUSIM/scomp_tb.v line 54
VSIM 7>]
```

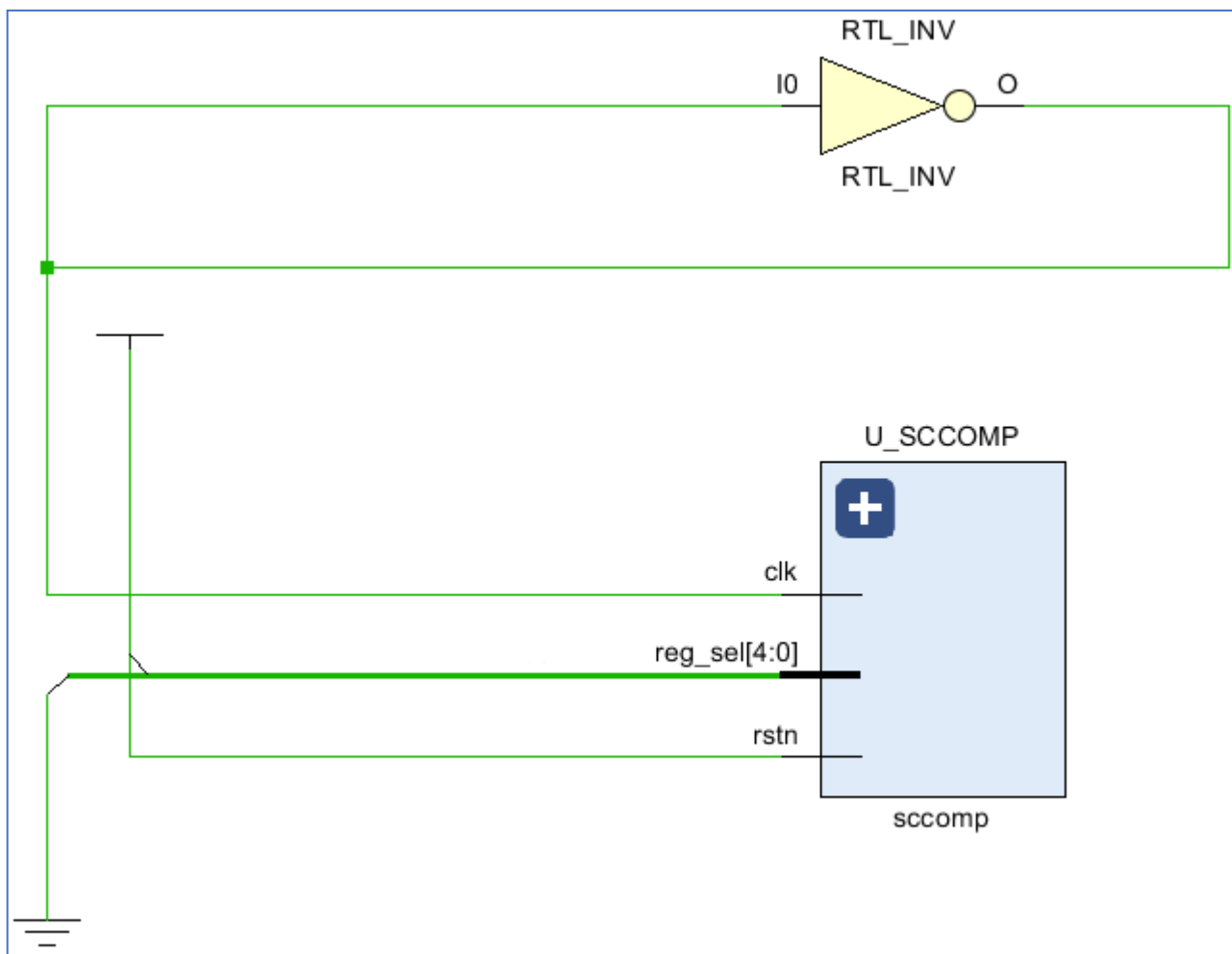
- 在Transcript窗口通过命令行输入 run 仿真时间长度
- 或者通过菜单Simulate → Run 运行仿真
- 观察信号波形图是否符合预期
- 检查Transcript 窗口中输出的调试信息是否正确（寄存器的值，内存单元的值）

查看Memory & Register Files



- 点击菜单View→Memory List
- 右键单击，修改显示数据的进制和宽度

示例ModelSim工程分析——顶层实体 (sccomp_tb)



- Testbench
- 提供激励信号

目录

- 实验目的&实验内容&考核方式
- 设计方法
- Verilog语言
- ModelSim使用及示例分析
- Mars使用

Mars使用

- Mars是MIPS汇编程序和运行时模拟器
- 用于MIPS汇编语言程序设计的IDE

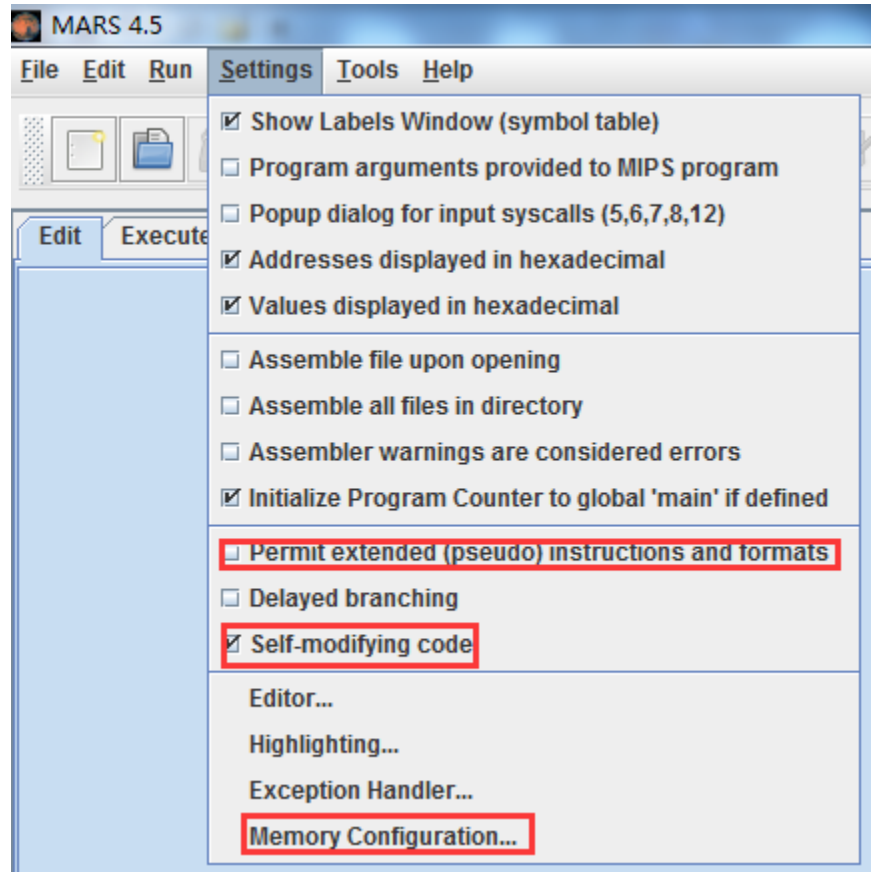
MIPS ASM代码

\source\mipstestloopjal_sim.
asm是仿真示例中运行的程序

```
C:\Users\Q. Liu\Desktop\source\mipstestloopjal_sim.asm - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?

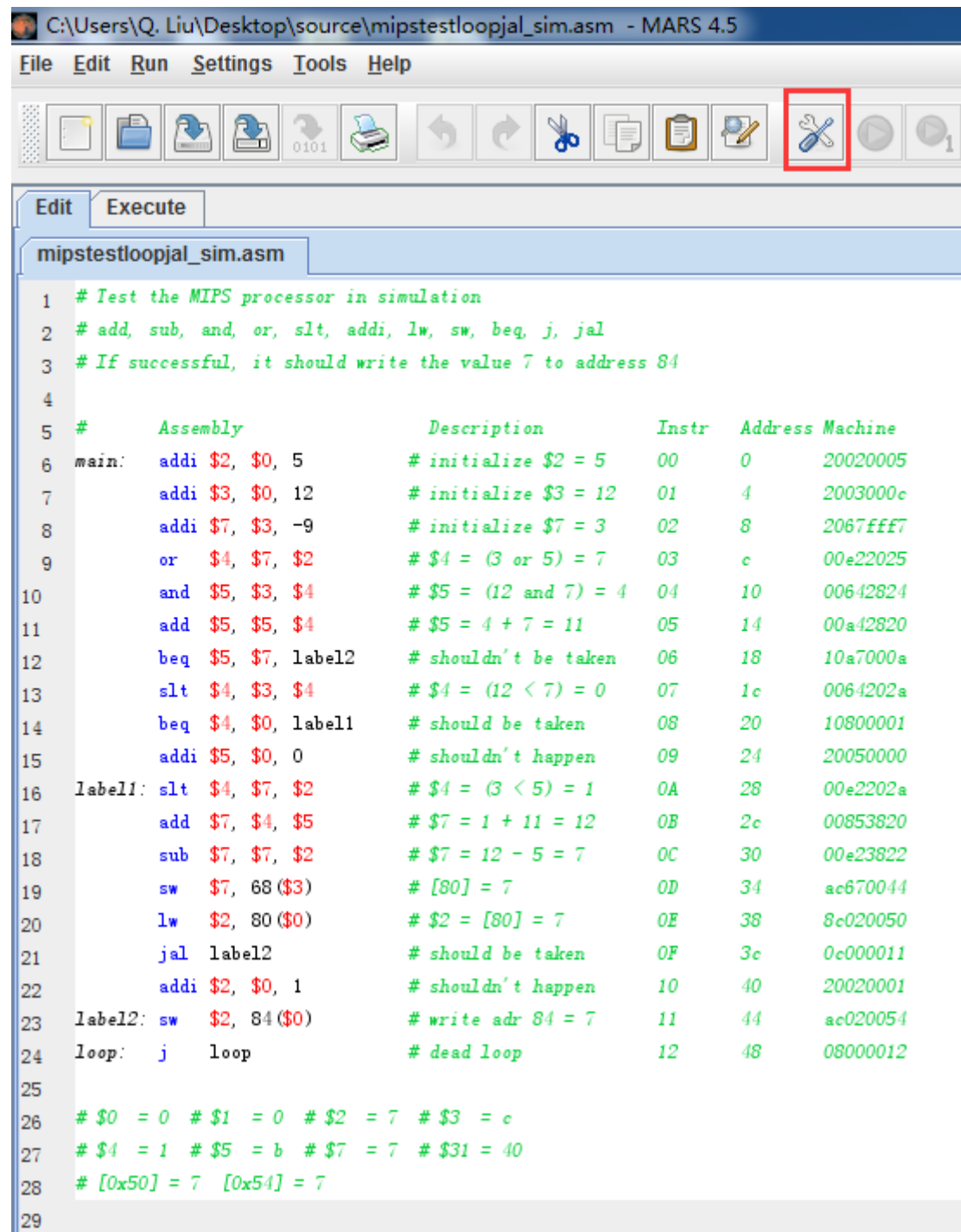
mipstestloopjal_sim.asm
1  # Test the MIPS processor in simulation
2  # add, sub, and, or, slt, addi, lw, sw, beq, j, jal
3  # If successful, it should write the value 7 to address 84
4
5  #      Assembly      Description      Instr  Address Machine
6  main:  addi $2, $0, 5      # initialize $2 = 5      00      0      20020005
7         addi $3, $0, 12     # initialize $3 = 12     01      4      2003000c
8         addi $7, $3, -9     # initialize $7 = 3      02      8      2067fff7
9         or    $4, $7, $2    # $4 = (3 or 5) = 7      03      c      00e22025
10        and   $5, $3, $4    # $5 = (12 and 7) = 4    04      10     00642824
11        add   $5, $5, $4    # $5 = 4 + 7 = 11       05      14     00a42820
12        beq   $5, $7, label2 # shouldn't be taken    06      18     10a7000a
13        slt   $4, $3, $4    # $4 = (12 < 7) = 0      07      1c     0064202a
14        beq   $4, $0, label1 # should be taken       08      20     10800001
15        addi  $5, $0, 0     # shouldn't happen      09      24     20050000
16  label1: slt   $4, $7, $2    # $4 = (3 < 5) = 1      0A      28     00e2202a
17         add   $7, $4, $5    # $7 = 1 + 11 = 12      0B      2c     00853820
18         sub   $7, $7, $2    # $7 = 12 - 5 = 7       0C      30     00e23822
19         sw    $7, 68($3)    # [80] = 7              0D      34     ac670044
20         lw    $2, 80($0)    # $2 = [80] = 7         0E      38     8c020050
21         jal   label2       # should be taken       0F      3c     0c000011
22         addi  $2, $0, 1     # shouldn't happen      10      40     20020001
23  label2: sw    $2, 84($0)    # write adr 84 = 7      11      44     ac020054
24  loop:   j     loop        # dead loop             12      48     08000012
25
26 # $0 = 0 # $1 = 0 # $2 = 7 # $3 = c
27 # $4 = 1 # $5 = b # $7 = 7 # $31 = 40
28 # [0x50] = 7 [0x54] = 7
```

MIPS Memory Configuration



打开文件&汇编

- File→Open...
- /source/mipstestloop.asm
- Assemble




```
1 # Test the MIPS processor in simulation
2 # add, sub, and, or, slt, addi, lw, sw, beq, j, jal
3 # If successful, it should write the value 7 to address 84
4
5 #      Assembly      Description      Instr  Address Machine
6 main:  addi $2, $0, 5      # initialize $2 = 5      00      0      20020005
7        addi $3, $0, 12     # initialize $3 = 12     01      4      2003000c
8        addi $7, $3, -9     # initialize $7 = 3      02      8      2067fff7
9        or $4, $7, $2       # $4 = (3 or 5) = 7      03      c      00e22025
10       and $5, $3, $4      # $5 = (12 and 7) = 4    04      10     00642824
11       add $5, $5, $4      # $5 = 4 + 7 = 11       05      14     00a42820
12       beq $5, $7, label2  # shouldn't be taken    06      18     10a7000a
13       slt $4, $3, $4      # $4 = (12 < 7) = 0      07      1c     0064202a
14       beq $4, $0, label1  # should be taken       08      20     10800001
15       addi $5, $0, 0      # shouldn't happen      09      24     20050000
16 label1: slt $4, $7, $2     # $4 = (3 < 5) = 1       0A      28     00e2202a
17        add $7, $4, $5      # $7 = 1 + 11 = 12      0B      2c     00853820
18        sub $7, $7, $2      # $7 = 12 - 5 = 7       0C      30     00e23822
19        sw $7, 68($3)      # [80] = 7              0D      34     ac670044
20        lw $2, 80($0)      # $2 = [80] = 7         0E      38     8c020050
21        jal label2        # should be taken       0F      3c     0c000011
22        addi $2, $0, 1     # shouldn't happen      10      40     20020001
23 label2: sw $2, 84($0)     # write adr 84 = 7      11      44     ac020054
24 loop:  j  loop           # dead loop             12      48     08000012
25
26 # $0 = 0 # $1 = 0 # $2 = 7 # $3 = c
27 # $4 = 1 # $5 = b # $7 = 7 # $31 = 40
28 # [0x50] = 7 [0x54] = 7
29
```

设置断点

Edit		Execute				
Text Segment						
Bkpt	Address	Code	Basic	Source		
<input type="checkbox"/>	0x00000000	0x20020005	addi \$2,\$0,0x00000005	6: main:	addi \$2, \$0, 5	# initialize \$2 = 5 00...
<input type="checkbox"/>	0x00000004	0x2003000c	addi \$3,\$0,0x0000000c	7:	addi \$3, \$0, 12	# initialize \$3 = 12 01...
<input type="checkbox"/>	0x00000008	0x2067fff7	addi \$7,\$3,0xffffffff7	8:	addi \$7, \$3, -9	# initialize \$7 = 3 02...
<input type="checkbox"/>	0x0000000c	0x00e22025	or \$4,\$7,\$2	9:	or \$4, \$7, \$2	# \$4 = (3 or 5) = 7 03...
<input type="checkbox"/>	0x00000010	0x00642824	and \$5,\$3,\$4	10:	and \$5, \$3, \$4	# \$5 = (12 and 7) = 4 04...
<input type="checkbox"/>	0x00000014	0x00a42820	add \$5,\$5,\$4	11:	add \$5, \$5, \$4	# \$5 = 4 + 7 = 11 05...
<input type="checkbox"/>	0x00000018	0x10a7000a	beq \$5,\$7,0x0000000a	12:	beq \$5, \$7, label2	# shouldn't be taken 06...
<input type="checkbox"/>	0x0000001c	0x0064202a	slt \$4,\$3,\$4	13:	slt \$4, \$3, \$4	# \$4 = (12 < 7) = 0 07...
<input type="checkbox"/>	0x00000020	0x10800001	beq \$4,\$0,0x00000001	14:	beq \$4, \$0, label1	# should be taken 08...
<input type="checkbox"/>	0x00000024	0x20050000	addi \$5,\$0,0x00000000	15:	addi \$5, \$0, 0	# shouldn't happen 09...
<input type="checkbox"/>	0x00000028	0x00e2202a	slt \$4,\$7,\$2	16: label1:	slt \$4, \$7, \$2	# \$4 = (3 < 5) = 1 0A...
<input type="checkbox"/>	0x0000002c	0x00853820	add \$7,\$4,\$5	17:	add \$7, \$4, \$5	# \$7 = 1 + 11 = 12 0B...
<input type="checkbox"/>	0x00000030	0x00e23822	sub \$7,\$7,\$2	18:	sub \$7, \$7, \$2	# \$7 = 12 - 5 = 7 0C...
<input type="checkbox"/>	0x00000034	0xac670044	sw \$7,0x00000044(\$3)	19:	sw \$7, 68(\$3)	# [80] = 7 0D...
<input type="checkbox"/>	0x00000038	0x8c020050	lw \$2,0x00000050(\$0)	20:	lw \$2, 80(\$0)	# \$2 = [80] = 7 0E...
<input type="checkbox"/>	0x0000003c	0x0c000011	jal 0x00000044	21:	jal label2	# should be taken 0F...
<input type="checkbox"/>	0x00000040	0x20020001	addi \$2,\$0,0x00000001	22:	addi \$2, \$0, 1	# shouldn't happen 10...
<input type="checkbox"/>	0x00000044	0xac020054	sw \$2,0x00000054(\$0)	23: label2:	sw \$2, 84(\$0)	# write adr 84 = 7 11...
<input checked="" type="checkbox"/>	0x00000048	0x08000012	j 0x00000048	24: loop:	j loop	# dead loop 12...

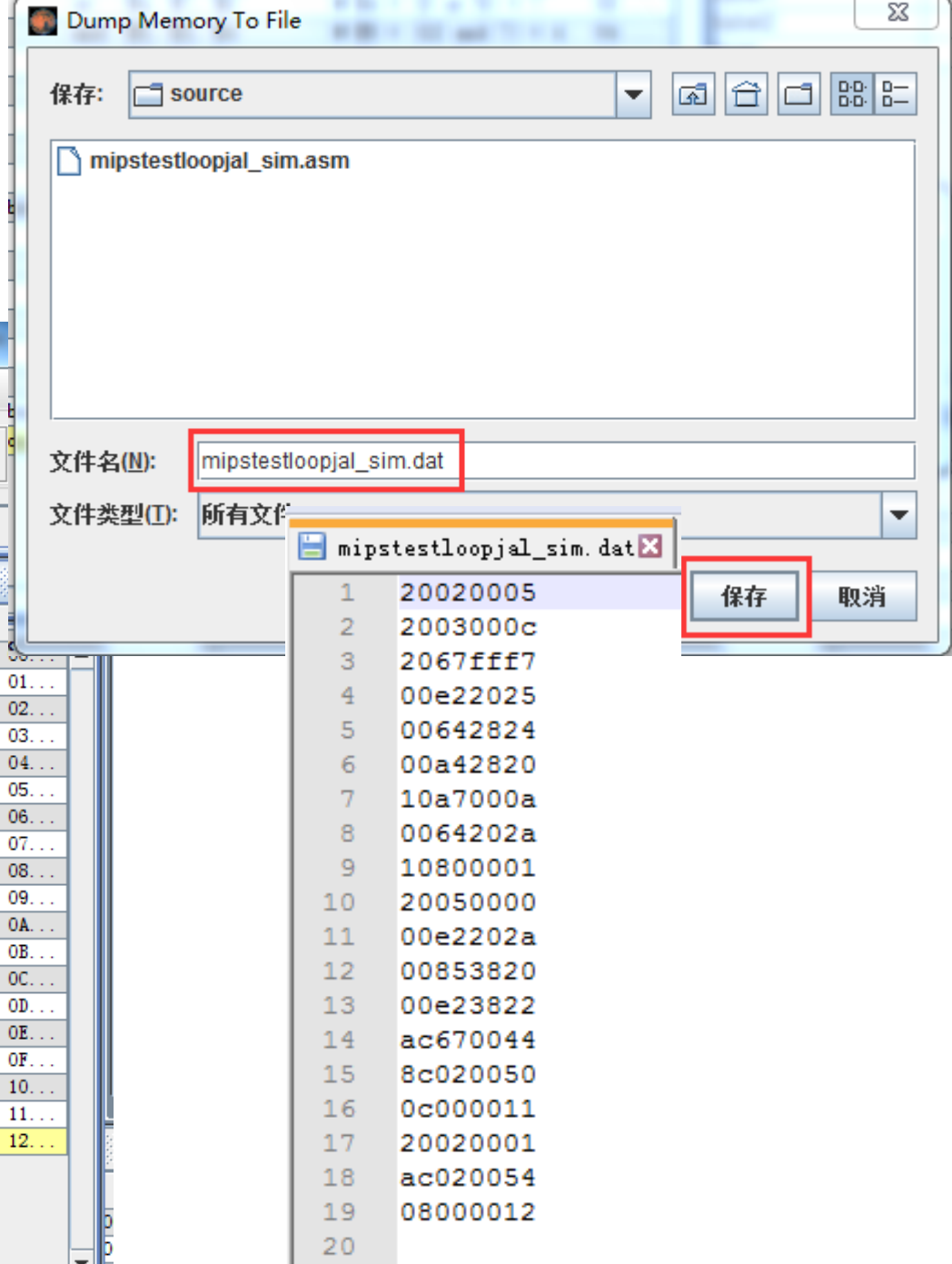
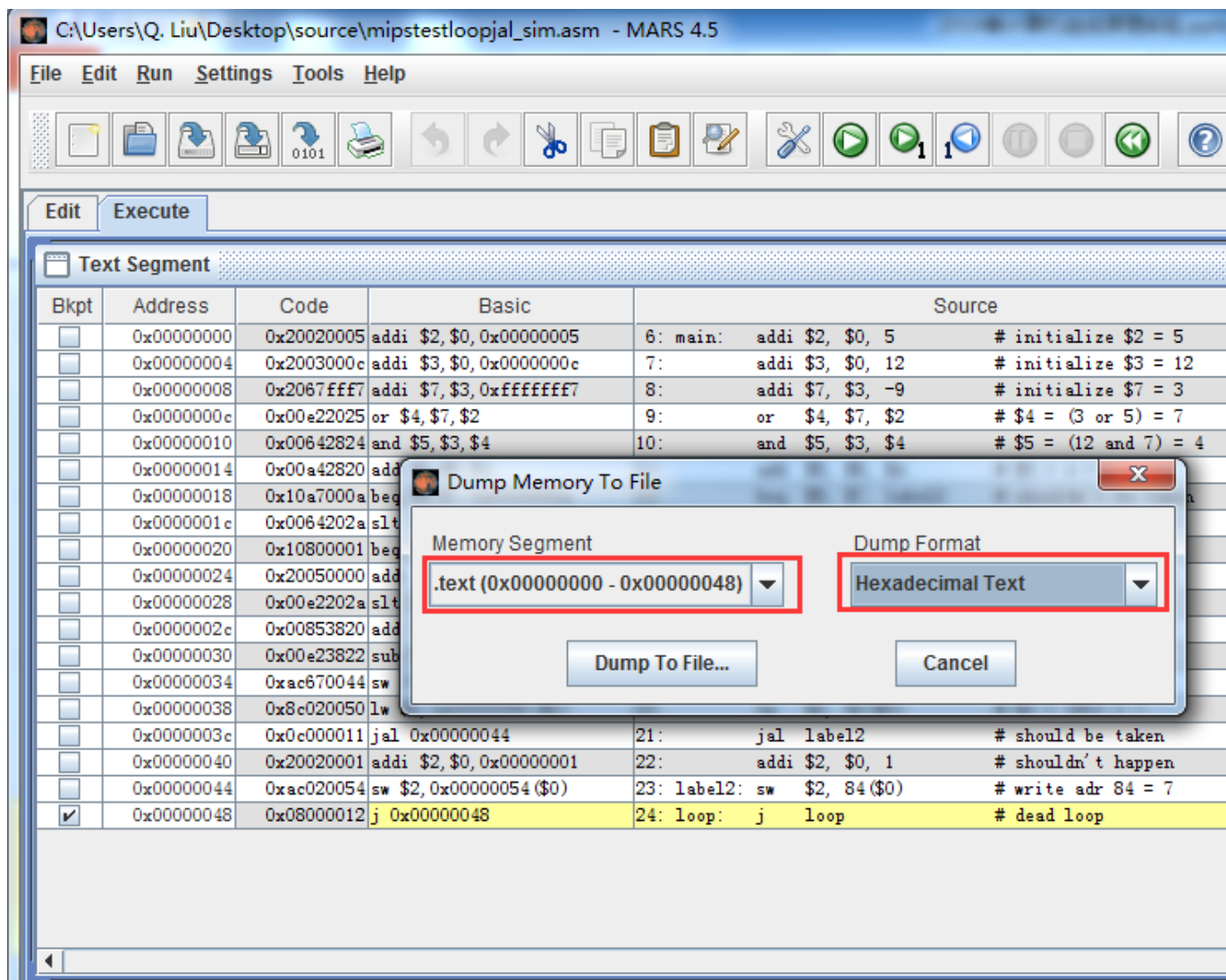
运行 & 寄存器内容查看

Run s					
					
Edit Execute					
Text Segment					
Bkpt	Address	Code	Basic	Source	
<input type="checkbox"/>	0x00000000	0x20020005	addi \$2,\$0,0x00000005	6: main:	addi \$2, \$0, 5 # initialize \$2 = 5 00...
<input type="checkbox"/>	0x00000004	0x2003000c	addi \$3,\$0,0x0000000c	7:	addi \$3, \$0, 12 # initialize \$3 = 12 01...
<input type="checkbox"/>	0x00000008	0x2067fff7	addi \$7,\$3,0xfffffffff7	8:	addi \$7, \$3, -9 # initialize \$7 = 3 02...
<input type="checkbox"/>	0x0000000c	0x00e22025	or \$4,\$7,\$2	9:	or \$4, \$7, \$2 # \$4 = (3 or 5) = 7 03...
<input type="checkbox"/>	0x00000010	0x00642824	and \$5,\$3,\$4	10:	and \$5, \$3, \$4 # \$5 = (12 and 7) = 4 04...
<input type="checkbox"/>	0x00000014	0x00a42820	add \$5,\$5,\$4	11:	add \$5, \$5, \$4 # \$5 = 4 + 7 = 11 05...
<input type="checkbox"/>	0x00000018	0x10a7000a	beq \$5,\$7,0x0000000a	12:	beq \$5, \$7, label2 # shouldn't be taken 06...
<input type="checkbox"/>	0x0000001c	0x0064202a	slt \$4,\$3,\$4	13:	slt \$4, \$3, \$4 # \$4 = (12 < 7) = 0 07...
<input type="checkbox"/>	0x00000020	0x10800001	beq \$4,\$0,0x00000001	14:	beq \$4, \$0, label1 # should be taken 08...
<input type="checkbox"/>	0x00000024	0x20050000	addi \$5,\$0,0x00000000	15:	addi \$5, \$0, 0 # shouldn't happen 09...
<input type="checkbox"/>	0x00000028	0x00e2202a	slt \$4,\$7,\$2	16: label1:	slt \$4, \$7, \$2 # \$4 = (3 < 5) = 1 0A...
<input type="checkbox"/>	0x0000002c	0x00853820	add \$7,\$4,\$5	17:	add \$7, \$4, \$5 # \$7 = 1 + 11 = 12 0B...
<input type="checkbox"/>	0x00000030	0x00e23822	sub \$7,\$7,\$2	18:	sub \$7, \$7, \$2 # \$7 = 12 - 5 = 7 0C...
<input type="checkbox"/>	0x00000034	0xac670044	sw \$7,0x00000044(\$3)	19:	sw \$7, 68(\$3) # [80] = 7 0D...
<input type="checkbox"/>	0x00000038	0x8c020050	lw \$2,0x00000050(\$0)	20:	lw \$2, 80(\$0) # \$2 = [80] = 7 0E...
<input type="checkbox"/>	0x0000003c	0x0c000011	jal 0x00000044	21:	jal label2 # should be taken 0F...
<input type="checkbox"/>	0x00000040	0x20020001	addi \$2,\$0,0x00000001	22:	addi \$2, \$0, 1 # shouldn't happen 10...
<input type="checkbox"/>	0x00000044	0xac020054	sw \$2,0x00000054(\$0)	23: label2:	sw \$2, 84(\$0) # write adr 84 = 7 11...
<input checked="" type="checkbox"/>	0x00000048	0x08000012	j 0x00000048	24: loop:	j loop # dead loop 12...

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000007
\$v1	3	0x0000000c
\$a0	4	0x00000001
\$a1	5	0x0000000b
\$a2	6	0x00000000
\$a3	7	0x00000007
\$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	0x00001800
\$sp	29	0x00003ffc
\$fp	30	0x00000000
\$ra	31	0x00000040
pc		0x00000048
hi		0x00000000
lo		0x00000000

Hex代码导出方法

- File->Dump Memory...



实验内容二——多周期CPU

- 实现流水线CPU，至少支持以下指令：
 - add/sub/and/or/slt/sltu/addu/subu
 - addi/ori/lw/sw/beq
 - j/jal
 - xor/xori/addiu/andi/nor
 - sll/sra/srav/lui/slti/sltiu/srl/sllv/srlv/jr/jalr
 - lb/lh/lbu/lhu/sb/sh (数据在内存中以小端形式存储 little endian)
 - bne/blez/bltz/bgtz/bgez
- 能够正确处理多周期CPU中状态转换。

实验内容三——流水线CPU

- 实现流水线CPU，至少支持以下指令：
 - add/sub/and/or/slt/sltu/addu/subu
 - addi/ori/lw/sw/beq
 - j/jal
 - xor/xori/addiu/andi/nor
 - sll/sra/srav/lui/slti/sltiu/srl/sllv/srlv/jr/jalr
 - lb/lh/lbu/lhu/sb/sh (数据在内存中以小端形式存储 little endian)
 - bne/blez/bltz/bgtz/bgez
- 能够正确处理流水线CPU中的数据冒险和控制冒险