实验二 汇编程序设计

2024春季

zjx@ustc.edu.cn

实验目标

- **理解龙芯架构32位精简(LoongArch32 Reduced, LA32R)处理器基础整数指令的功能和编码格式**
- **掌握LA32R简单汇编程序设计、仿真和调试的基本方法，以及下载测试代码和数据(COE文件) 的生成方法**

实验内容

1. 理解LA32R基础整数指令功能和编码格式

- 算术/比较运算：add.w, sub.w, addi.w, lu12i.w, pcaddu12i, slt, sltu, slti, sltui
- 逻辑运算：and, or, nor, xor, andi, ori, xori
- 移位运算：sll.w, srl.w, sra.w, slli.w, srli.w, srai.w
- 访存：st.w, ld.w, st.h, st.b, ld.h, ld.b, ld.hu, ld.bu
- 转移：beq, bne, blt, bge, bltu, bgeu, b, bl, jirl

2. 设计汇编程序，实现数组排序

- 数组数据：1024个32位无符号数
- 排序算法：冒泡降序排序

LA32R寄存器

- **32个通用寄存器（寄存器堆）**
  - r0 ~ r31
  - 助记符与rn等价

    例如，ra = r1
  - 有无$前缀等价

    例如，$ra = ra,
  - 大小写等价

    例如，R1 = r1

寄存器编号	助记符	使用约定
0	zero	总是为0
1	ra	子程序返回地址
2	tp	Thread Pointer，指向线程私有存储区
3	sp	栈指针
4~11	a0~a7	子程序的前八个参数
4~5	v0~v1	v0/v1是a0/a1的别名，用于表示返回值
12~20	t0~t8	不需保存的暂存器
21	Reserved	暂时保留不用
22	fp	Frame Pointer，栈帧指针
23-31	s0~s8	寄存器变量，子程序使用需要保存和恢复

- **PC**

加/减/比较运算指令

- add.w rd, rj, rk # rd = rj + rk
- sub.w rd, rj, rk # rd = rj - rk
- addi.w rd, rj, si12 # rd = rj + SE(si12)
- lu12i.w rd, si20 # rd = {si20, 12’b0}
- pcaddu12i rd, si20 # rd = pc + {si20, 12’b0}

- slt rd, rj, rk # rd = rj $<_s$ rk
- sltu rd, rj, rk # rd = rj $<_u$ rk
- slti rd, rj, si12 # rd = rj $<_s$ SE(si12)
- sltui rd, rj, si12 # rd = rj $<_u$ SE(si12)

逻辑运算指令

- and rd, rj, rk # rd = rj & rk
- or rd, rj, rk # rd = rj | rk
- nor rd, rj, rk # rd = ~ (rj | rk)
- xor rd, rj, rk # rd = rj ^ rk

- andi rd, rj, ui12 # rd = rj & ZE(ui12)
- ori rd, rj, ui12 # rd = rj | ZE(ui12)
- xori rd, rj, ui12 # rd = rj ^ ZE(ui12)

移位运算指令

- sll.w rd, rj, rk # rd = rj << rk[4 : 0]
- srl.w rd, rj, rk # rd = rj >> rk[4 : 0]
- sra.w rd, rj, rk # rd = rj >>> rk[4 : 0]

- slli.w rd, rj, ui5 # rd = rj << ui5
- srli.w rd, rj, ui5 # rd = rj >> ui5
- srai.w rd, rj, ui5 # rd = rj >>> ui5

访存指令

- ld.w rd, rj, si12 # rd = MemW[rj + SE(si12)]
- ld.b rd, rj, si12 # rd = SE(MemB[rj + SE(si12)])
- ld.h rd, rj, si12 # rd = SE(MemH[rj + SE(si12)])
- ld.bu rd, rj, si12 # rd = ZE(MemB[rj + SE(si12)])
- ld.hu rd, rj, si12 # rd = ZE(MemH[rj + SE(si12)])

- st.w rd, rj, si12 # MemW[rj + SE(si12)] = rd
- st.b rd, rj, si12 # MemB[rj + SE(si12)] = rd[7:0]
- st.h rd, rj, si12 # MemH[rj + SE(si12)] = rd[15:0]

转移指令

- beq rj, rd, offs16 # if (ri == rd) pc += SE({offs16, 2'b0})
- bne rj, rd, offs16 # !=
- blt rj, rd, offs16 # $<_s$
- bge rj, rd, offs16 # $!<_s$
- bltu rj, rd, offs16 # $<_u$
- bgeu rj, rd, offs16 # $!<_u$

- b offs26 # pc += SE({offs26, 2'b0})
- bl offs26 # r1 = pc + 4, pc += SE({offs26, 2'b0})
- jirl rd, rj, offs16 # rd = pc + 4, pc = rj + SE({offs16, 2'b0})

指令编码格式

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
2R-type	opcode																						rj					rd				
3R-type	opcode																	rk					rj					rd				
4R-type	opcode												ra					rk					rj					rd				
2RI8-type	opcode														I8								rj					rd				
2RI12-type	opcode										I12												rj					rd				
2RI14-type	opcode								I14														rj					rd				
2RI16-type	opcode						I16																rj					rd				
1RI21-type	opcode						I21[15:0]																rj					I21[20:16]				
I26-type	opcode						I26[15:0]																I26[25:16]									

		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
LD.B	**rd, rj, si12**	0	0	1	0	1	0	0	0	0	0	si12												rj					rd				
LD.H	**rd, rj, si12**	0	0	1	0	1	0	0	0	0	1	si12												rj					rd				
LD.W	**rd, rj, si12**	0	0	1	0	1	0	0	0	1	0	si12												rj					rd				
ST.B	**rd, rj, si12**	0	0	1	0	1	0	0	1	0	0	si12												rj					rd				
ST.H	**rd, rj, si12**	0	0	1	0	1	0	0	1	0	1	si12												rj					rd				
ST.W	**rd, rj, si12**	0	0	1	0	1	0	0	1	1	0	si12												rj					rd				
LD.BU	**rd, rj, si12**	0	0	1	0	1	0	1	0	0	0	si12												rj					rd				
LD.HU	**rd, rj, si12**	0	0	1	0	1	0	1	0	0	1	si12												rj					rd				
JIRL	**rd, rj, offs**	0	1	0	0	1	1	offs[15:0]																rj					rd				
B	**offs**	0	1	0	1	0	0	offs[15:0]																offs[25:16]									
BL	**offs**	0	1	0	1	0	1	offs[15:0]																offs[25:16]									
BEQ	**rj, rd, offs**	0	1	0	1	1	0	offs[15:0]																rj					rd				
BNE	**rj, rd, offs**	0	1	0	1	1	1	offs[15:0]																rj					rd				
BLT	**rj, rd, offs**	0	1	1	0	0	0	offs[15:0]																rj					rd				
BGE	**rj, rd, offs**	0	1	1	0	0	1	offs[15:0]																rj					rd				
BLTU	**rj, rd, offs**	0	1	1	0	1	0	offs[15:0]																rj					rd				
BGEU	**rj, rd, offs**	0	1	1	0	1	1	offs[15:0]																rj					rd				

		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SLLI.W	**rd, rj, ui5**	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	ui5					rj					rd				
SRLI.W	**rd, rj, ui5**	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	ui5					rj					rd				
SRAI.W	**rd, rj, ui5**	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	ui5					rj					rd				
SLTI	**rd, rj, si12**	0	0	0	0	0	0	1	0	0	0	si12												rj					rd				
SLTUI	**rd, rj, si12**	0	0	0	0	0	0	1	0	0	1	si12												rj					rd				
ADDI.W	**rd, rj, si12**	0	0	0	0	0	0	1	0	1	0	si12												rj					rd				
ANDI	**rd, rj, ui12**	0	0	0	0	0	0	1	1	0	1	ui12												rj					rd				
ORI	**rd, rj, ui12**	0	0	0	0	0	0	1	1	1	0	ui12												rj					rd				
XORI	**rd, rj, ui12**	0	0	0	0	0	0	1	1	1	1	ui12												rj					rd				
LU12I.W	**rd, si20**	0	0	0	1	0	1	0	si20																				rd				
PCADDU12I	**rd, si20**	0	0	0	1	1	1	0	si20																				rd				

		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADD.W	**rd, rj, rk**	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	rk					rj					rd				
SUB.W	**rd, rj, rk**	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	rk					rj					rd				
SLT	**rd, rj, rk**	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	rk					rj					rd				
SLTU	**rd, rj, rk**	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	rk					rj					rd				
NOR	**rd, rj, rk**	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	rk					rj					rd				
AND	**rd, rj, rk**	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	rk					rj					rd				
OR	**rd, rj, rk**	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	rk					rj					rd				
XOR	**rd, rj, rk**	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	rk					rj					rd				
SLL.W	**rd, rj, rk**	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	rk					rj					rd				
SRL.W	**rd, rj, rk**	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	rk					rj					rd				
SRA.W	**rd, rj, rk**	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	rk					rj					rd				
MUL.W	**rd, rj, rk**	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	rk					rj					rd				
MULH.W	**rd, rj, rk**	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	1	rk					rj					rd				
MULH.WU	**rd, rj, rk**	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	rk					rj					rd				
DIV.W	**rd, rj, rk**	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	rk					rj					rd				
MOD.W	**rd, rj, rk**	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	rk					rj					rd				
DIV.WU	**rd, rj, rk**	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	rk					rj					rd				
MOD.WU	**rd, rj, rk**	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	rk					rj					rd				

汇编器指令和宏指令

- **汇编器指令**
  - .data, .text # 数据段，代码段
  - .word, .half, .byte, .string # 字，半字，字节，字符串
  - .align n # 2^n 字节对齐
  - … …
- **宏指令**
  - nop # 空操作，等价 andi/ori r0, r0, 0
  - li.w rd, imm # rd = imm, 最多转换为 lu12i.w 和 ori 两条指令
  - la.local rd, label # rd = label, 最多转换为 lu12i.w 和 ori 两条指令
  - … …

LA32R简单测试程序

```
# inst_ram.txt, 计算斐波拉契数列

1c000000:   addi.w   $t0, $zero, 0x0    # f0 = 0
1c000004:   addi.w   $t1, $zero, 0x1    # f1 = 1
1c000008:   addi.w   $s0, $zero, 0x0   # 循环计数i = 0
1c00000c:   addi.w   $s1, $zero, 0x1   # 计数步长 = 1
1c000010:   ld.w     $a0, $zero, 1024   # 输入循环终值n
      loop:
1c000014:   add.w    $t2, $t0, $t1       # fi = fi-2 + fi-1
1c000018:   addi.w   $t0, $t1, 0x0       # fi-2 = fi-1
1c00001c:   addi.w   $t1, $t2, 0x0       # fi-1 = fi
1c000020:   add.w    $s0, $s0, $s1      # i++
1c000024:   bne      $s0, $a0, loop      # if i! = n, 循环loop
1c000028:   st.w     $t2, $zero, 1028   # 结束，输出fn
        end:
1c00002c:   bne      $s1, $zero, end    # 进入死循环end
```

```
; inst_ram.coe

memory_initialization_radix = 16;
memory_initialization_vector =
0280000c
0280040d
02800017
02800418
28900004
0010358e
028001ac
028001cd
001062f7
5ffff2e4
2990100e
5c000300
```

LA32R指令功能测试程序

- 参考资料
  - CPU设计实战：LoongArch版
  - https://bookdown.org/loongson/_book3/
  - https://gitee.com/loongson-edu/cdp_ede_local
- **mycpu_env/func/inst/*.S**：针对每条指令或功能点的汇编测试程序
- **mycpu_env/func/start.S**：主函数，执行必要的启动初始化后调用**func/inst/**下的各汇编程序
- **mycpu_envfunc/include/*.h**：测试程序配置信息和宏定义

```
|--func/                    实验任务所用的功能验证测试程序。
|   |--include/             功能验证测试程序共享的头文件所在目录。
|   |   |--sysdep.h         一些GCC通用的宏定义的头文件。
|   |   |--asm.h            LoongArch汇编需用到的一些宏定义的头文件，比如LEAF(x)。
|   |   |--regdef.h         LoongArch32 ABI下，32个通用寄存器的汇编助记定义。
|   |   |--cpu_cde.h        SoC_Lite相关参数的宏定义，如访问数码管的confreg的基址。
|   |   |--inst_test.h      各功能测试点的验证程序使用的宏定义头文件
|   |--inst/                各功能测试点的汇编程序文件。
|   |   |--Makefile         子目录里的Makefile，会被上一级目录中的Makefile调用。
|   |   |--n*.S             各功能测试点的验证程序，汇编语言编写。
|   |--obj/                 功能验证测试程序编译结果存放目录
|   |   |--*                详见后面小节的说明。
|   |--start.S              功能验证测试的引导代码及主函数。
|   |--Makefile             编译功能验证测试程序的Makefile脚本
|   |--bin.lds              编译bin.lds.S得到的结果，可被make reset命令清除
|   |--convert.c            生成coe和mif文件的处理工具的C程序源码
```

实验要求

- **阅读理解用于LA32R指令测试的汇编程序**

- **设计汇编程序，实现数组排序，并生成COE文件**
  - 使用LARS（Loongarch32R Assembler and Runtime Simulator）
  - https://soc.ustc.edu.cn/COD/lab1/lars/

The End

RISC-V寄存器

- **PC和32个通用寄存器（合称寄存器堆）**

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

RV32I指令类型

- 运算类
  - 算术：add, sub, addi, auipc, lui
  - 逻辑：and, or, xor, andi, ori, xori
  - 移位(shift)：sll, srl, sra, slli, srli, srai
  - 比较(set if less than)：slt, sltu, slti, sltiu

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)

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

RV32I指令功能

- add rd, rs1, rs2 # x[rd] = x[rs1] + x[rs2]
- addi rd, rs1, imm # x[rd] = x[rs1] + sext(imm)
- sub rd, rs1, rs2 # x[rd] = x[rs1] - x[rs2]
- auipc rd, imm # x[rd] = pc + imm[31:12] << 12
- lui rd, imm # x[rd] = imm[31:12] << 12
- and rd, rs1, rs2 # x[rd] = x[rs1] & x[rs2]
- or rd, rs1, rs2 # x[rd] = x[rs1] | x[rs2]
- xor rd, rs1, rs2 # x[rd] = x[rs1] ^ x[rs2]
- slli rd, rs1, shamt # x[rd] = x[rs1] $\ll$ shamt
- srli rd, rs1, shamt # x[rd] = x[rs1] $\gg_u$ shamt
- srai rd, rs1, shamt # x[rd] = x[rs1] $\gg_s$ shamt

RV32I指令功能

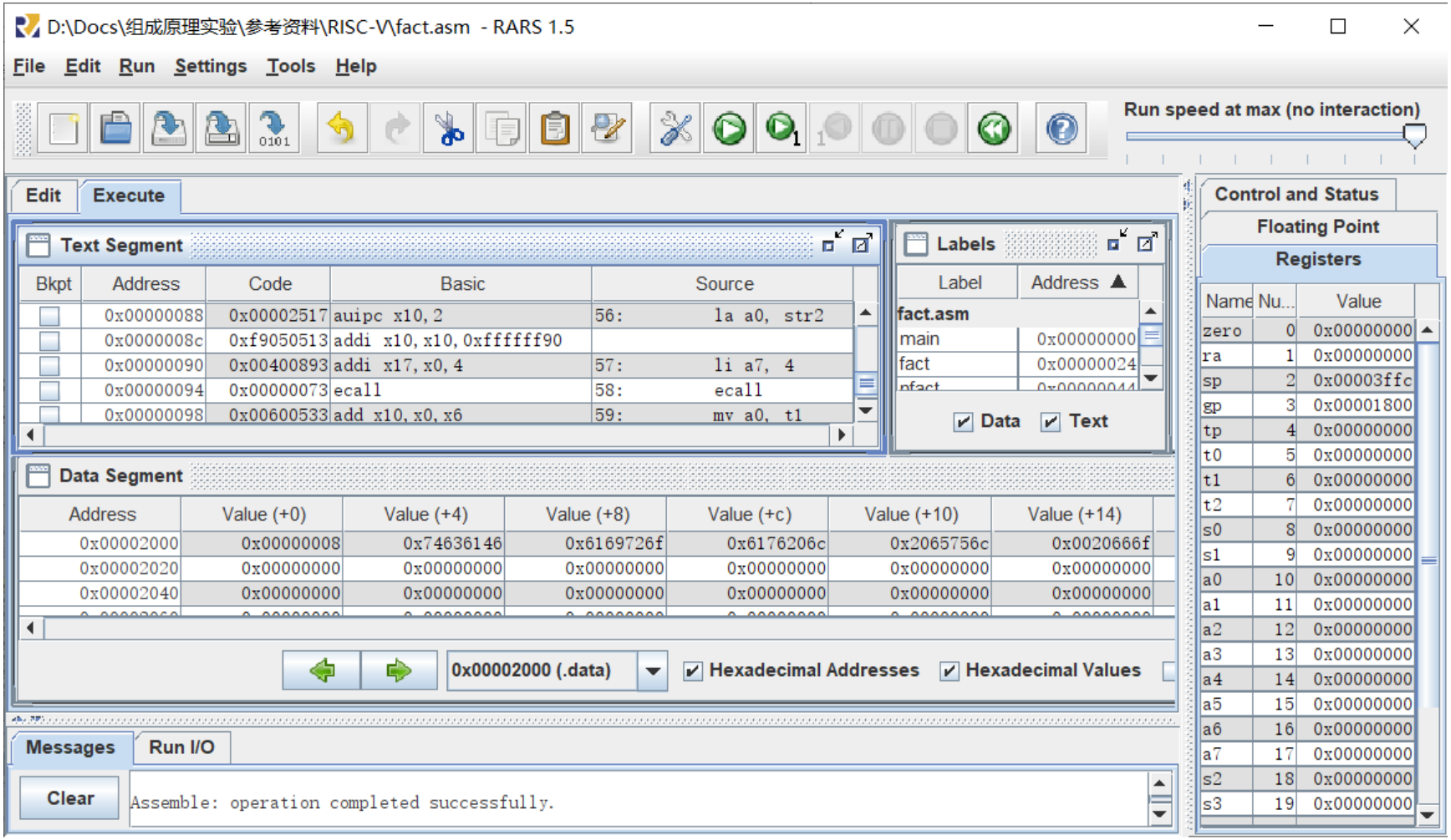
- lw rd, offset(rs1) # x[rd] = M[x[rs1] + sext(offset)]
- sw rs2, offset(rs1) # M[x[rs1] + sext(offset)] = x[rs2]
- beq rs1, rs2, offset # if (rs1 == rs2) pc += sext(offset)
- blt rs1, rs2, offset # if (rs1 $<_s$ rs2) pc += sext(offset)
- bltu rs1, rs2, offset # if (rs1 $<_u$ rs2) pc += sext(offset)
- 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

汇编指示符和伪指令

- 汇编指示符（**Assembly Directives**）
  - .data, .text
  - .word, .half, .byte, .string
  - .align
  - ……
- 伪指令（**Pseudo Instructions**）
  - li, la, mv
  - nop, not, neg
  - j, jr, call, ret
  - ……
- 参考资料：**RISC-V Assembly Programmer's Manual**
  - https://github.com/riscv-non-isa/riscv-asm-manual/blob/master/riscv-asm.md#risc-v-assembly-programmers-manual

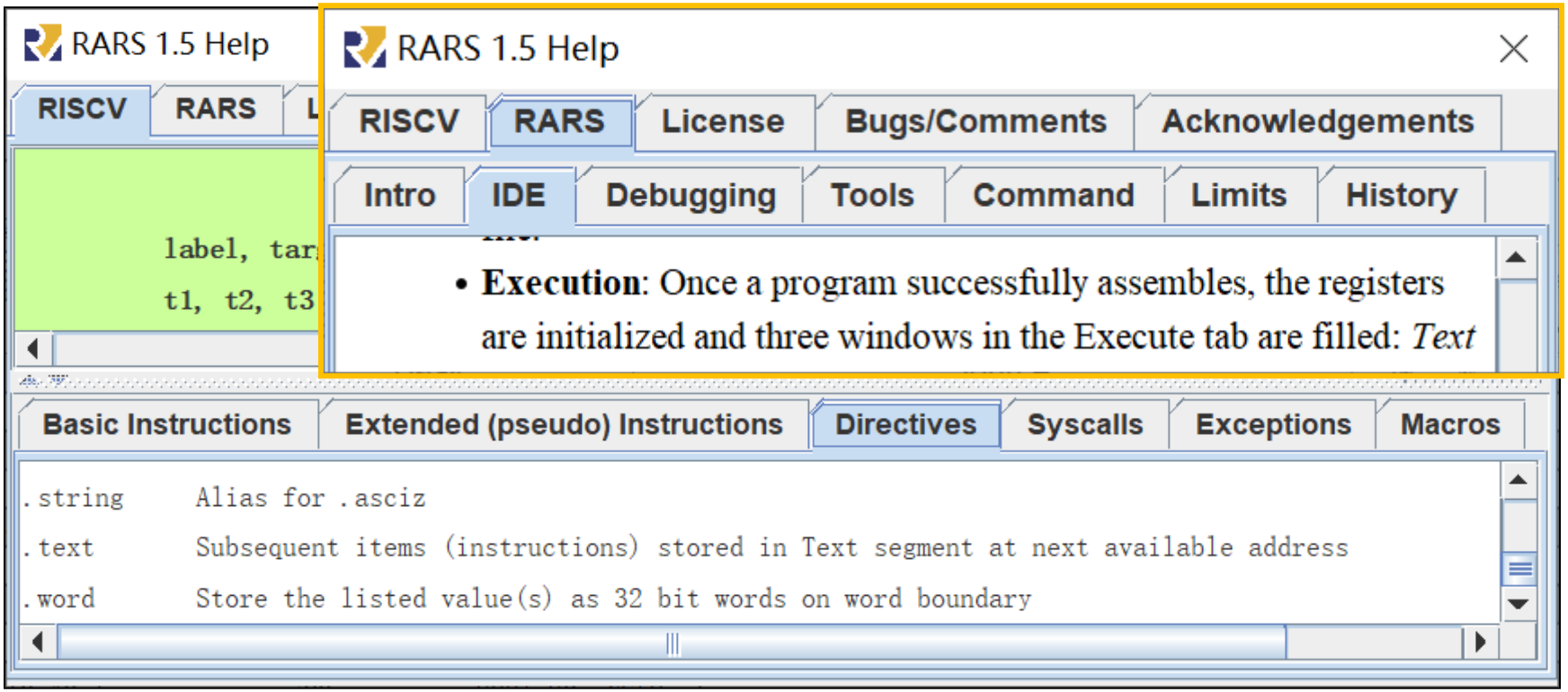
RARS

- **RISC-V Assembler & Runtime Simulator**



Help

- **RISCV：指令、伪指令、指示符、系统调用……**
- **RARS：IDE、调试、工具……**

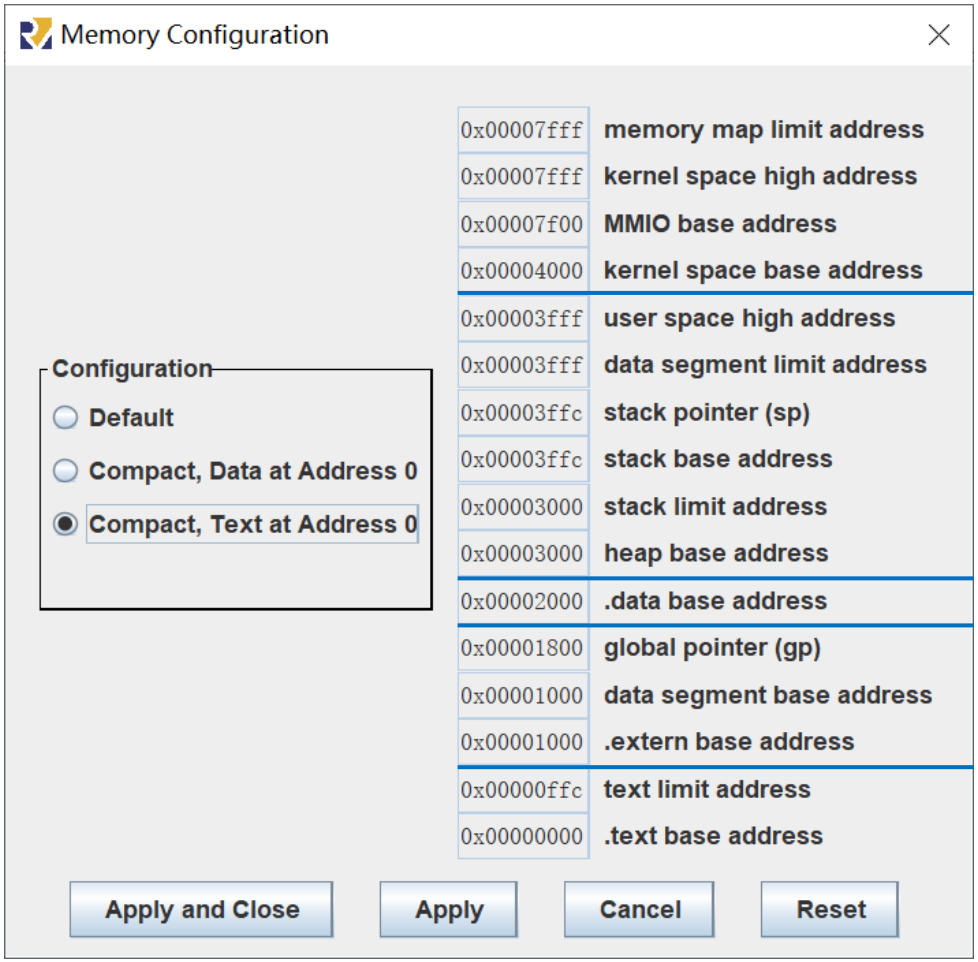


存储器配置

- **Setting >> Memory Configuration…**
- 假定配置为紧凑型

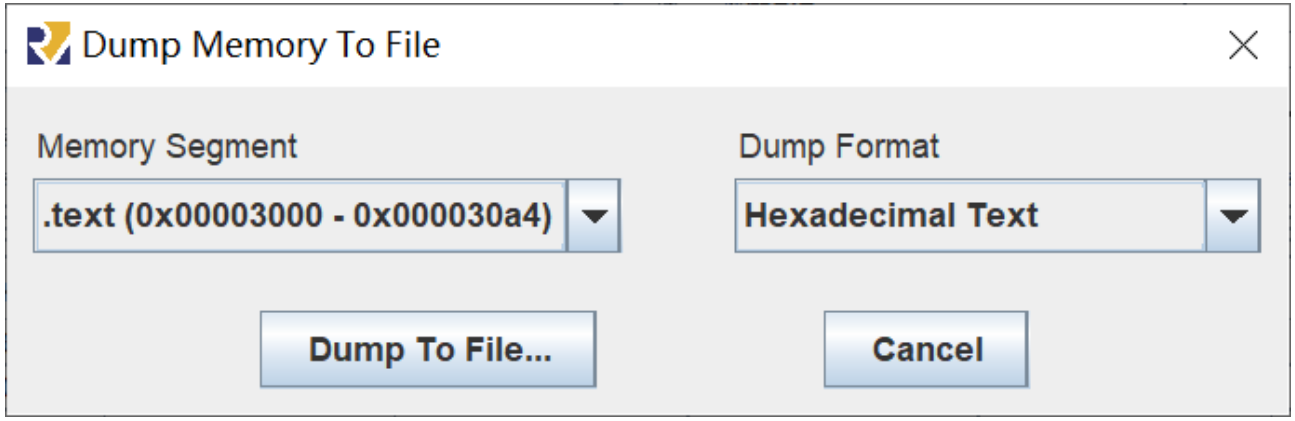
  代码地址：
  - 0x0000 ~ 0xffc

  数据地址：
  - 0x2000 ~ 0x2ffc



汇编程序转COE文件

- 配置存储器：**Setting >> Memory Configuration…**
- 汇编程序：**Run >> Assemble**
- 导出代码和数据：**File >> Dump Memory…**



- 生成COE文件：导出文本的开头添加以下两行

memory_initialization_radix = 16;
memory_initialization_vector =