实验三 汇编程序设计

2022春季

zjx@ustc.edu.cn

实验目标

- 了解汇编程序的基本结构，以及汇编程序仿真和调试的基本方法
- 熟悉RISC-V常用32位整数指令的功能，掌握简单汇编程序的设计，以及CPU下载测试方法和测试数据 (COE文件) 的生成方法

实验内容

1. **设计汇编程序：逐条简单测试以下10条指令功能**
   - add, addi, sub, auipc
   - lw, sw
   - beq, blt, jal, jalr
2. **设计汇编程序：排序256个无符号数据**
   - 排序算法不限
   - 仅使用上述10条指令实现

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

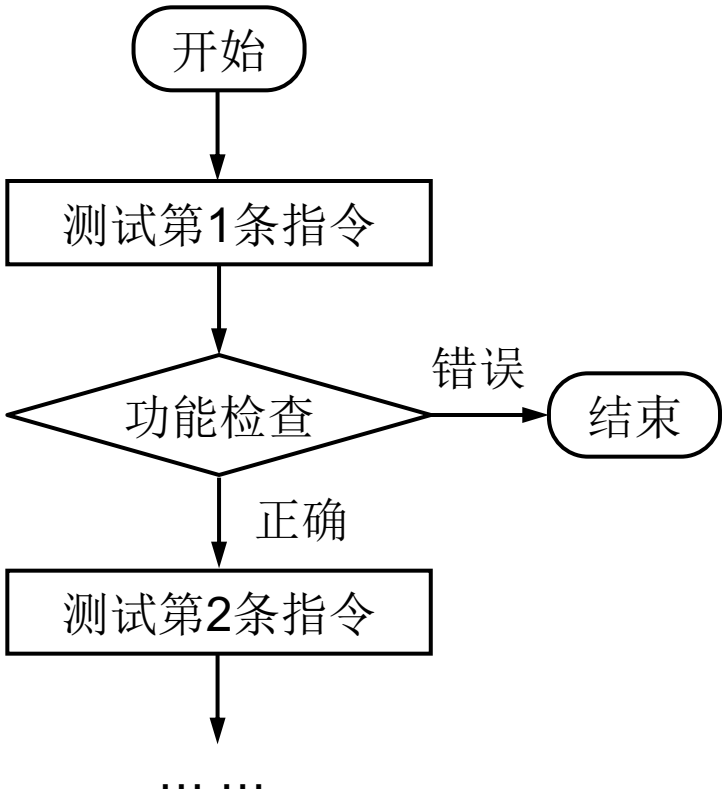
- add rd, rs1, rs2 # x[rd] = x[rs1] + x[rs2]
- addi rd, rs1, imm # x[rd] = x[rs1] + sext(imm)
- sub rd, rs1, rs2 # x[rd] = x[rs1] - x[rs2]
- auipc rd, imm # x[rd] = pc + sext(imm[31:12] << 12)
- lw rd, offset(rs1) # x[rd] = M[x[rs1] + sext(offset)]
- sw rs2, offset(rs1) # M[x[rs1]+sext(offset)=x[rs2]
- beq rs1, rs2, offset # if (rs1 == rs2) pc += sext(offset)
- blt rs1, rs2, offset # if (rs1 $<_s$ rs2) pc += sext(offset)
- 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

简单测试指令功能

- **指令功能检查**
  - 人工检查指令执行结果，正确则继续下条指令测试
  - 或者自动判断指令测试结果
- **指令测试顺序**
  - 根据待测试指令与已测试指令的依赖关系确定测试先后顺序



示例：测试程序

```
.data                                #假定起始地址为0
led_data:  .word  0xffff             #led指示灯状态，初始全亮
swt_data: .word  0xaa55              #开关(switch)状态

.text
sw      x0, 0(x0)                    #test sw: 全灭led
addi    t0, x0, 0xfff                #test addi: 全亮led
sw      t0, 0(x0)
lw      t0, 4(x0)                    #test lw: 由switch设置led
sw      t0, 0(x0)
… …
```

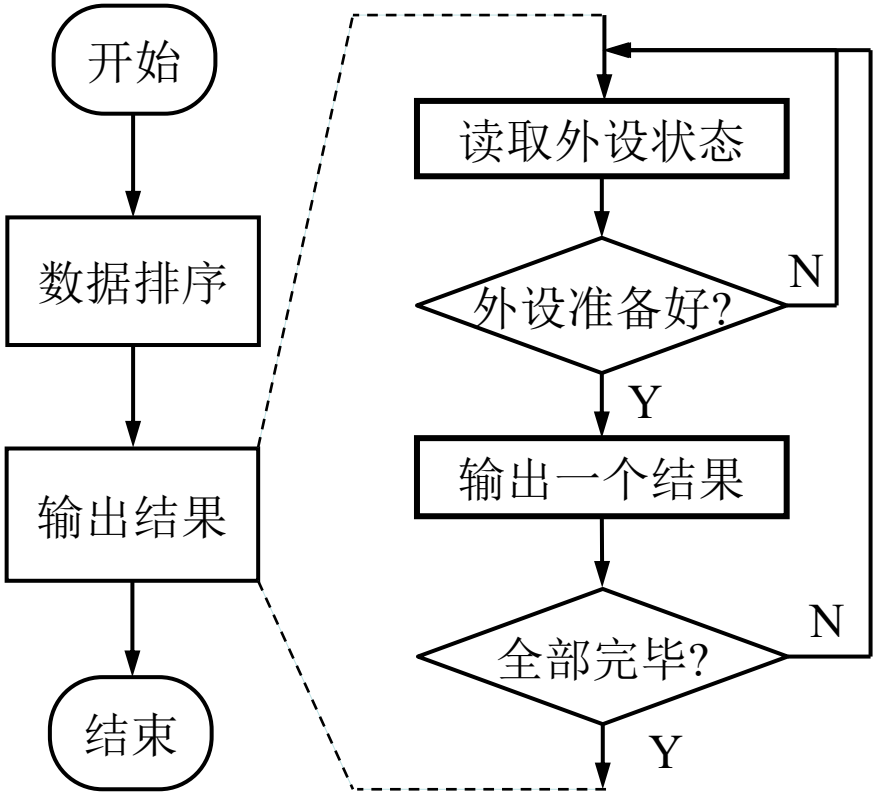
数据排序

- **实现类似Lab2功能**
  - 数据排序
  - 输出排序结果
- **仿真输入/输出**
  - MMIO：Memory-Mapped Input and Output

    例如，输出显示(Display) 外设

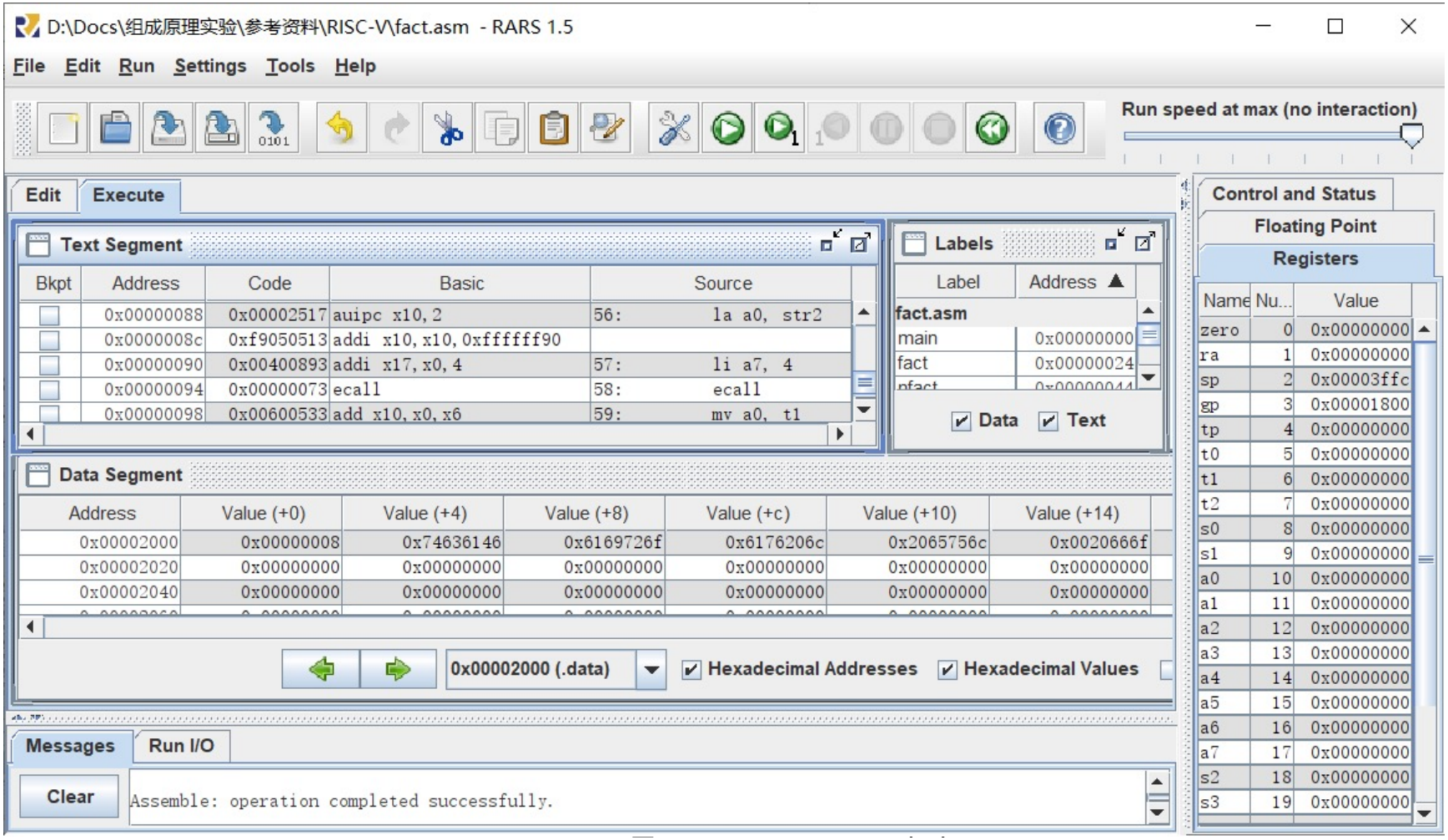
    0x7f08: 状态地址

    0x7f0c: 数据地址



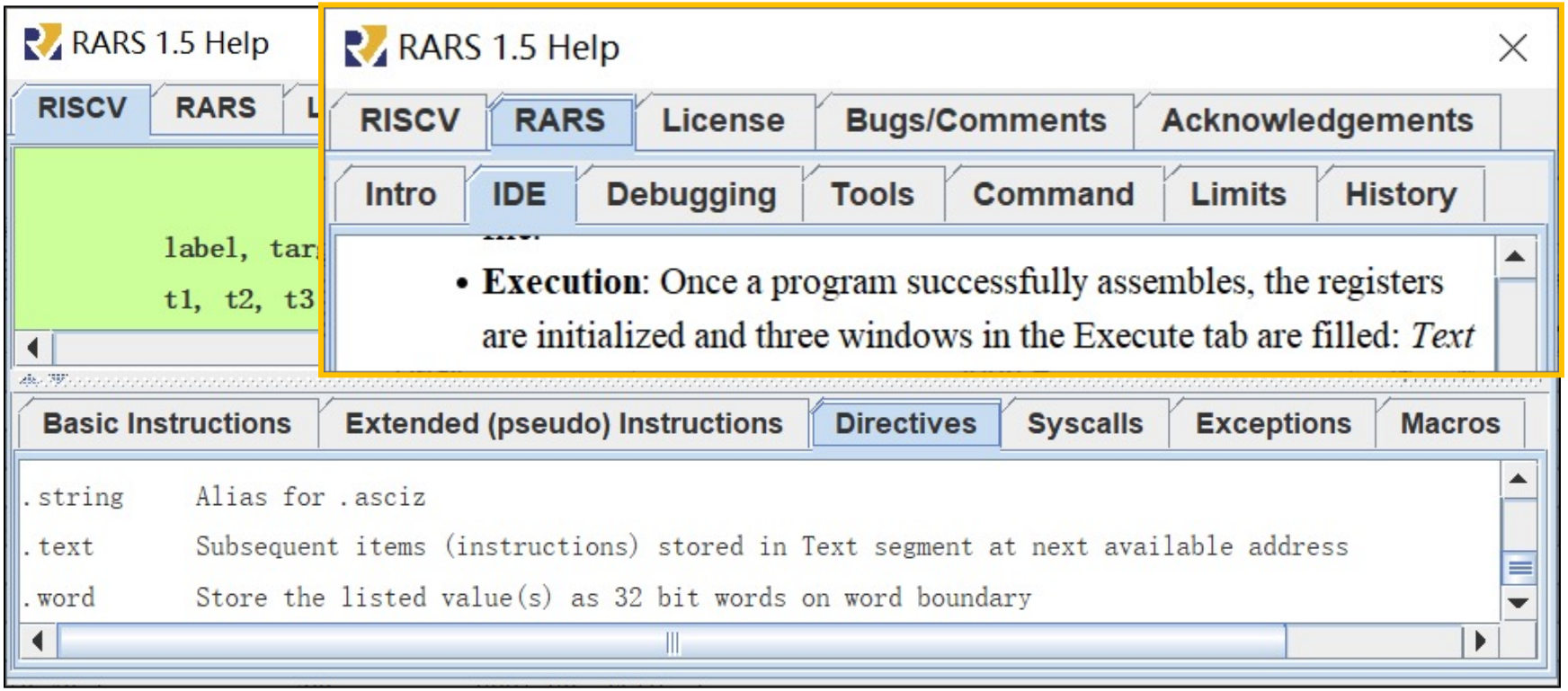
RARS

- **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……**



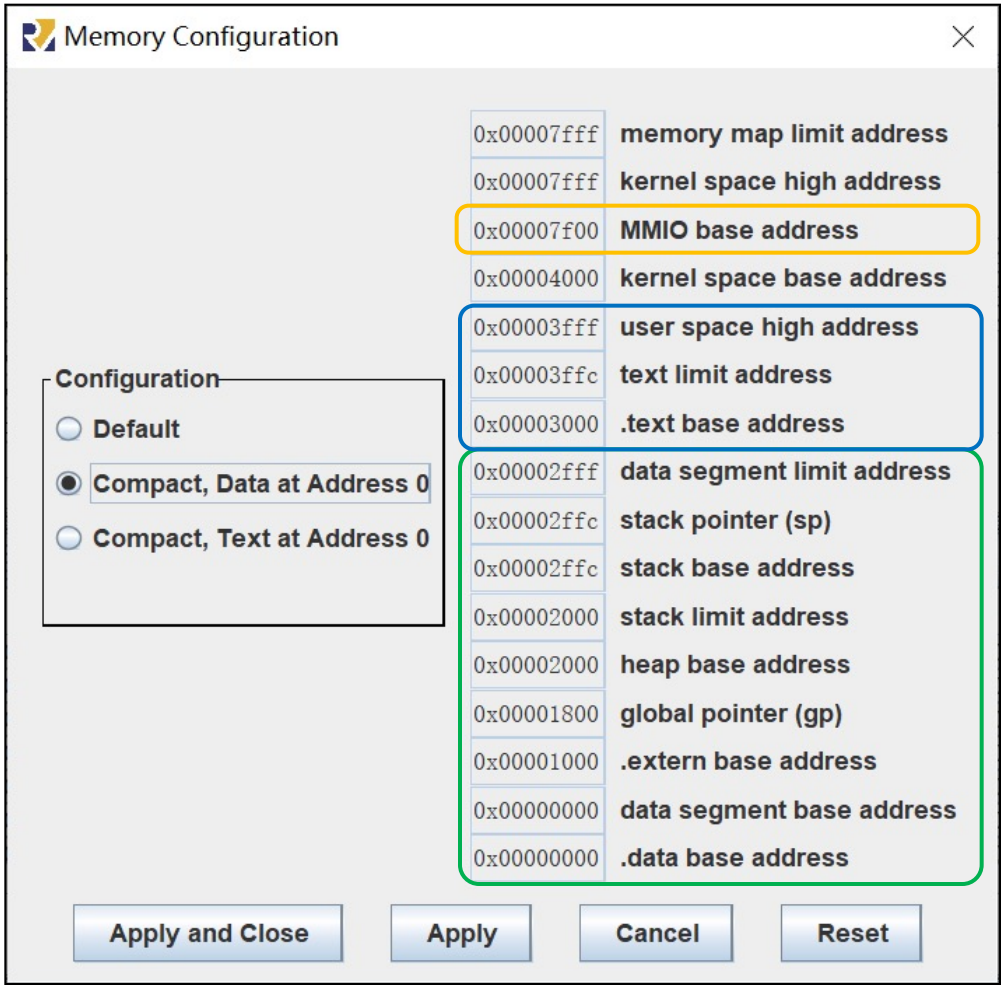
存储器配置

- **Setting >> Memory Configuration…**
- **假定设置为紧凑型**

  数据地址：
  - 0x0000_0000 ~ 0x0000_2ffff

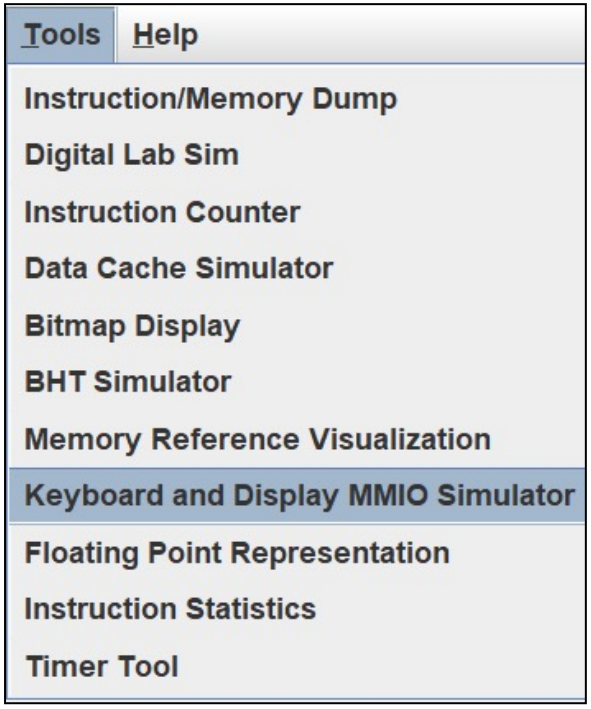
  代码地址：
  - 0x0000_3000 ~ 0x0000_3fff

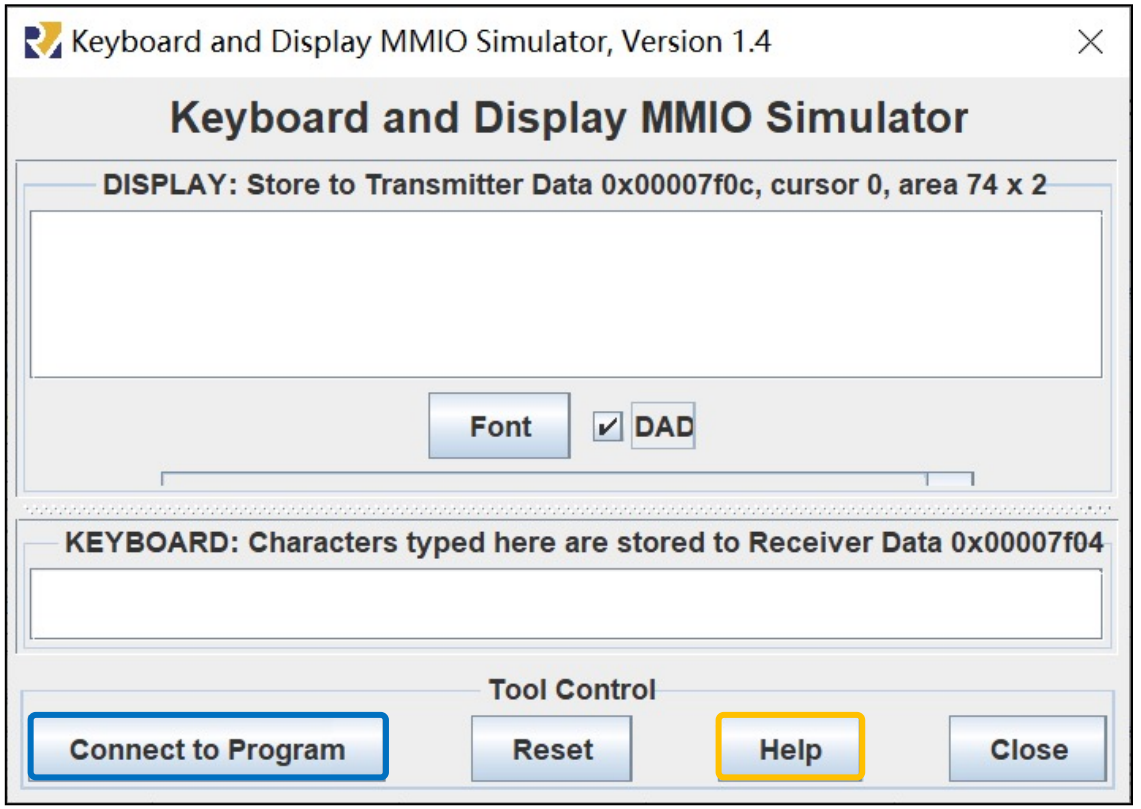
  MMIO地址：
  - 0x0000_7f00 ~



MMIO

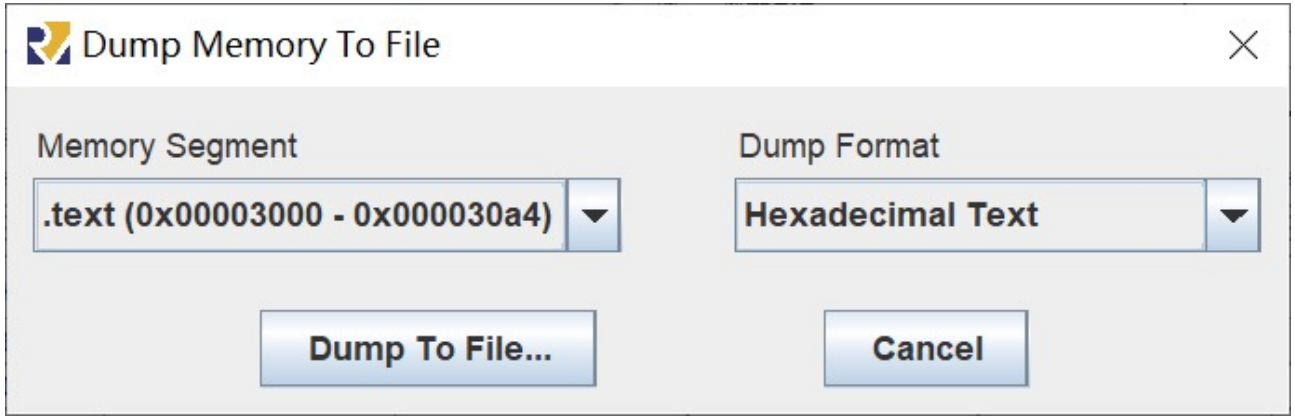
- **Memory-Mapped Input and Output**





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 =
```

实验步骤

1. **设计汇编程序，实现对10条指令功能的逐条简单测试和人工检查，并生成COE文件**
   - 选项：实现若干条指令功能的充分测试和自动检查
2. **设计汇编程序，实现数组排序，并生成COE文件**
   - 对存储器例化时初始化的数组排序，并将结果输出到显示器
   - 选项：随机生成或键盘输入数组（第一个数是数组的大小，随后为数组的数据）

The End