

## 设计基础知识（详细看上课教材）

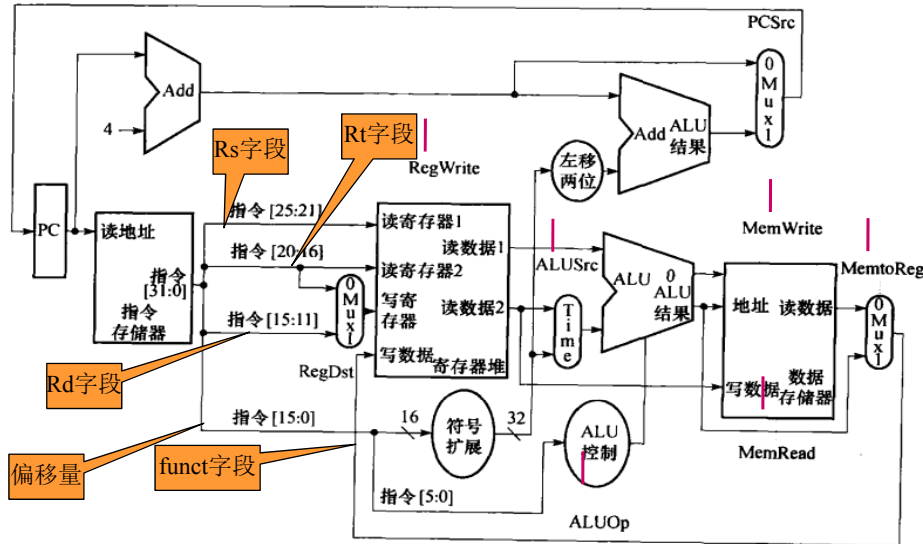


图 4-15 在图 4-12 的数据通路上增加了所有必需的多选器并标识出了所有的控制信号。控制信号以灰色线表示。还增加了 ALU 控制单元。PC 不需要写控制，因为它在每个时钟周期末都被写入一次。分支控制逻辑决定给 PC 自增还是写入分支目标地址。

### 4.2.1 MIPS 指令格式

MIPS 所有的指令均为 32 位。图 4.5 给出 MIPS 指令的 3 种格式。op 是指令码。

	31	26	25	21	20	16	15	11	10	6	5	0		
R 类型	op				rs		rt		rd		sa		func	
	6 位				5 位		5 位		5 位		5 位		6 位	
	31	26	25	21	20	16	15							0
I 类型	op				rs		rt		immediate					
	6 位				5 位		5 位		16 位					
	31	26	25											0
J 类型	op			address										
	6 位			26 位										

R 类型指令的 op 为 0，具体操作由 func 指定。rs 和 rt 是源寄存器号，rd 是目的寄存器号。

I 类型指令的低 16 位是立即数，计算时要把它扩展到 32 位。

J 类型的指令右边的 26 位是地址，用于产生跳转的目标地址。

### 4.2.2 MIPS 通用寄存器

MIPS 指令中的寄存器号 (rs、rt 和 rd) 有 5 位，因此它能访问  $2^5 = 32$  个寄存器。表 4.3 列出了这 32 个寄存器的名称和它们的用途。

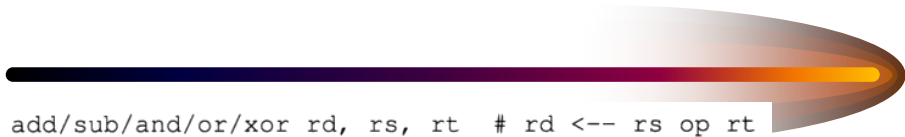
表 4.3 MIPS 通用寄存器

寄存器名	寄存器号	用 途
\$zero	0	常数 0
\$at	1	汇编器专用
\$v0 ~ \$v1	2 ~ 3	表达式计算或者函数调用的返回结果
\$a0 ~ \$a3	4 ~ 7	函数调用参数 1 ~ 3
\$t0 ~ \$t7	8 ~ 15	临时变量，函数调用时不需要保存和恢复
\$s0 ~ \$s7	16 ~ 23	函数调用时需要保存和恢复的寄存器变量
\$t8 ~ \$t9	24 ~ 25	临时变量，函数调用时不需要保存和恢复
\$k0 ~ \$k1	26 ~ 27	操作系统专用
\$gp	28	全局变量指针 (Global Pointer)
\$sp	29	堆栈指针 (Stack Pointer)
\$fp	30	帧指针 (Frame Pointer)
\$ra	31	返回地址 (Return Address)

### 4.3 MIPS 指令和 ALU 设计

表 4.4 20 条 MIPS 整数指令

指令	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	意 义
add	000000	rs	rt	rd	00000	100000	寄存器加
sub	000000	rs	rt	rd	00000	100010	寄存器减
and	000000	rs	rt	rd	00000	100100	寄存器与
or	000000	rs	rt	rd	00000	100101	寄存器或
xor	000000	rs	rt	rd	00000	100110	寄存器异或
sll	000000	00000	rt	rd	sa	000000	左移
srl	000000	00000	rt	rd	sa	000010	逻辑右移
sra	000000	00000	rt	rd	sa	000011	算术右移
jr	000000	rs	00000	00000	00000	001000	寄存器跳转
addi	001000	rs	rt	immediate			立即数加
andi	001100	rs	rt	immediate			立即数与
ori	001101	rs	rt	immediate			立即数或
xori	001110	rs	rt	immediate			立即数异或
lw	100011	rs	rt	offset			取整数数据字
sw	101011	rs	rt	offset			存整数数据字
beq	000100	rs	rt	offset			相等转移
bne	000101	rs	rt	offset			不等转移
lui	001111	00000	rt	immediate			设置高位
j	000010	address					跳转
jal	000011	address					调用



```
add/sub/and/or/xor rd, rs, rt # rd <-- rs op rt
```

```
sll/srl/sra rd, rt, sa # rd <-- rt shift sa
```

这是3条移位指令 (Shift Left/Right Logical/Arithmetic), 5位的 sa (Shift Amount) 指定移位的位数。

```
lui rt, imm # rt <-- imm << 16
```

lui (Load Upper Immediate) 指令把16位立即数 imm 左移16位, 存入 rt 寄存器。它与 ori 指令合作, 可以为一个32位的寄存器赋值: lui 赋高16位, ori 赋低16位。


```
addi rt, rs, imm # rt <-- rs + imm (符号扩展)
```

addi (Add Immediate) 是立即数的加法指令。注意目的寄存器号是 rt, 立即数要符号扩展到32位。



XU AI PING

计算机学院



```
lw rt, offset(rs) # rt <-- memory[rs + offset]
```

lw (Load Word) 是一条取存储器字的指令。寄存器 rs 的内容与符号扩展的 offset 相加, 得到存储器地址。从存储器取来的数据存入 rt 寄存器。注意, offset 就是前面讲的立即数。

```
sw rt, offset(rs) # memory[rs + offset] <-- rt
```

sw (Store Word) 是一条存字的指令, 与 lw 方向相反, 把 rt 寄存器的内容放入存储器。存储器地址的计算与 lw 相同。



XU AI PING

计算机学院

```
beq rs, rt, label # if (rs == rt) PC <-- label
```

beq (Branch on Equal) 是一条条件转移指令。当寄存器 rs 的内容与寄存器 rt 的内容相等时，转移到 label。如果程序计数器 PC 是 beq 指令的地址，则  $\text{label} = \text{PC} + 4 + \text{offset} \ll 2$ 。offset 左移两位导致 PC 的最低两位永远是 0，这是因为 PC 是字节地址而一条指令要占 4 个字节。offset 是要符号扩展的，因此 beq 能实现向前和向后两种转移。

```
bne rs, rt, label # if (rs != rt) PC <-- label
```

与 beq 类似，但 bne (Branch on Not Equal) 是在两个寄存器的内容不相等时转移。



XU AI PING

计算机学院

```
j target # PC <-- target
```

j (Jump) 是一条跳转指令。target 是跳转的目标地址，32 位，由 3 部分组成：最高 4 位来自于  $\text{PC} + 4$  的高 4 位，中间 26 位是指令中的 address，最低两位为 0。这条指令在生成目标地址时不需要任何电路进行计算，只需把 3 部分地址拼接起来就行。以下的两条指令也不需要计算。

```
jal target # r31 <-- PC + 8; PC <-- target
```

jal (Jump and Link) 指令与 j 类似，但要把返回地址保存在 r31 中。即 jal 是子程序调用指令。jal 下一条指令的地址是  $\text{PC} + 4$ ，为什么返回地址是  $\text{PC} + 8$ ？这是因为 MIPS 指令系统实现流水线的延迟转移功能，详见第 8 章。注意，寄存器号 31 是约定好的，该号码并不出现在指令中。因此在设计电路时，应当由硬件为 jal 指令产生这个号码。

```
jr rs # PC <-- rs
```

jr (Jump Register) 也是一条跳转指令，它把 rs 寄存器的内容写入 PC。如果指定 rs 为 31，则 jr 是从子程序返回的指令。



XU AI PING

计算机学院

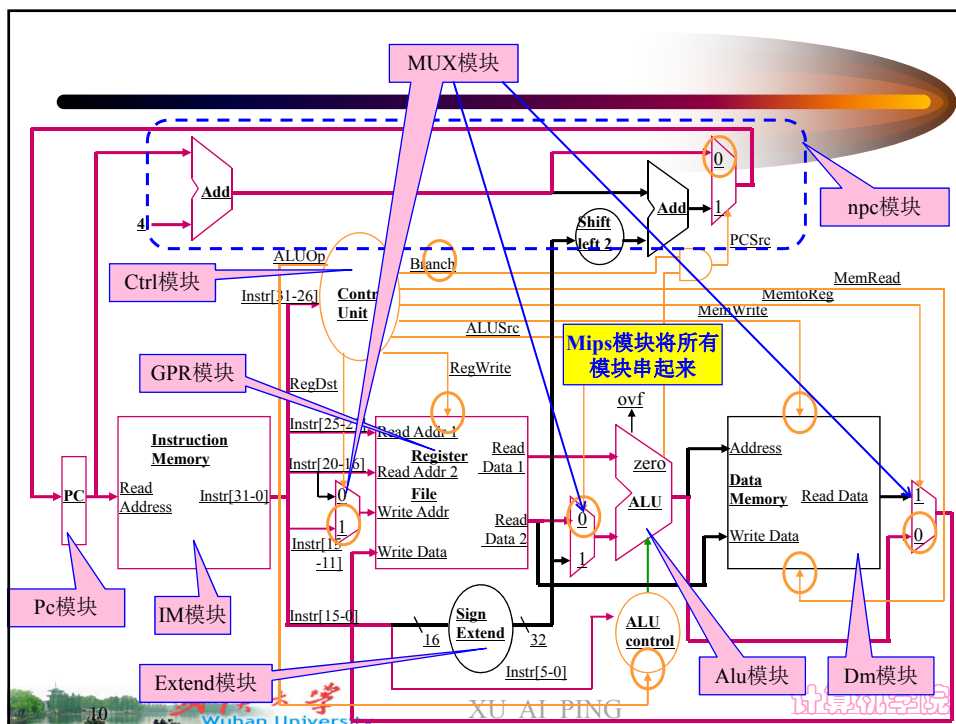
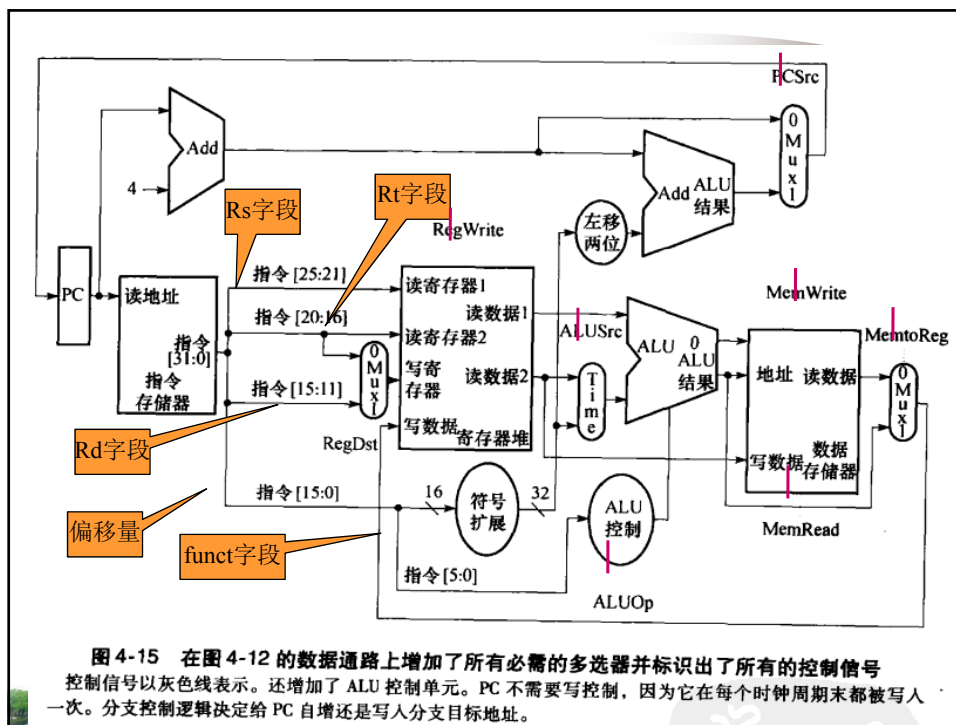


表 4.4 20 条 MIPS 整数指令

指令	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	意义
add	000000	rs	rt	rd	00000	100000	寄存器加
sub	000000	rs	rt	rd	00000	100010	寄存器减
and	000000	rs	rt	rd	00000	100100	寄存器与
or	000000	rs	rt	rd	00000	100101	寄存器或
xor	000000	rs	rt	rd	00000	100110	寄存器异或
sll	000000	00000	rt	rd	sa	000000	左移
srl	000000	00000	rt	rd	sa	000010	逻辑右移
sra	000000	00000	rt	rd	sa	000011	算术右移
jr	000000	rs	00000	00000	00000	001000	寄存器跳转



XU AI PING

计算机学院

addi	001000	rs	rt	immediate	立即数加
andi	001100	rs	rt	immediate	立即数与
ori	001101	rs	rt	immediate	立即数或
xori	001110	rs	rt	immediate	立即数异或
lw	100011	rs	rt	offset	取整数数据字
sw	101011	rs	rt	offset	存整数数据字
beq	000100	rs	rt	offset	相等转移
bne	000101	rs	rt	offset	不等转移
lui	001111	00000	rt	immediate	设置高位
j	000010	address			跳转
jal	000011	address			调用



XU AI PING

计算机学院

- MIPS-Lite1 = {addu, subu, ori, lw, sw, beq, jal}

addu	000000	rs	rt	rd	00000	100001	寄存器加
subu	000000	rs	rt	rd	00000	100010	寄存器减
ori	001101	rs	rt	immediate			立即数或
lw	100011	rs	rt	offset			取整数数据字
sw	101011	rs	rt	offset			存整数数据字
beq	000100	rs	rt	offset			相等转移
jal	000011	address					调用



XU AI PING

计算机学院

#### 4.4.4 控制的结束

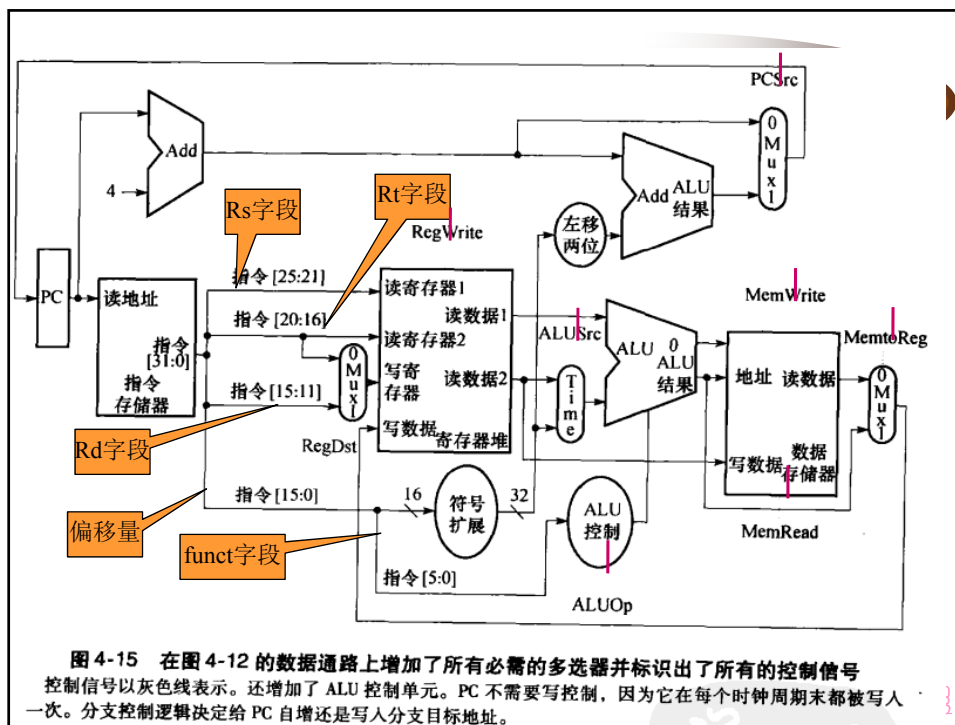
讨论过指令的操作之后，现在继续讨论控制单元的实现。控制单元的功能可由图 4-18 精确定义，其输入为 6 位操作码 Op [5:0]，输出为控制信号。这样，可以为每个输出建立一张真值表。

输入或输出	信号名	R 型	lw	sw	beq
输入	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
输出	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

图 4-22 简单的单周期实现的控制功能真值表

单周期实现 (single-cycle implementation): 也被称为单时钟周期实现 (single clock cycle implementation), 即一个时钟周期执行一条指令的实现机制。



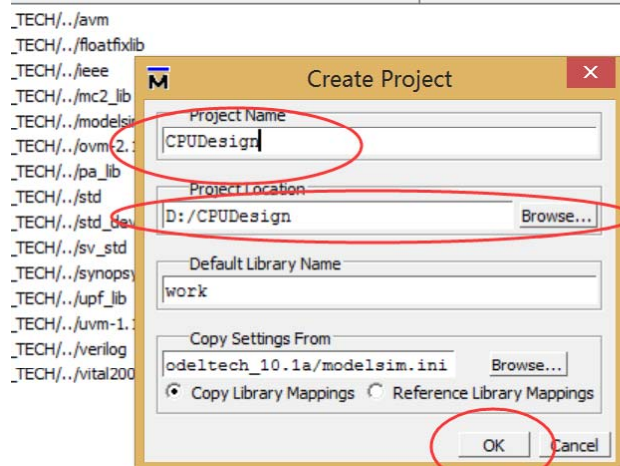
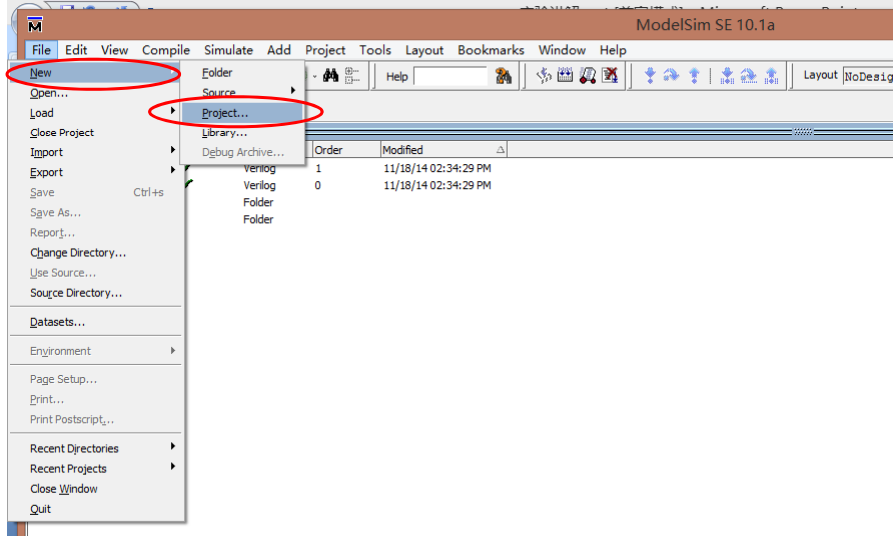


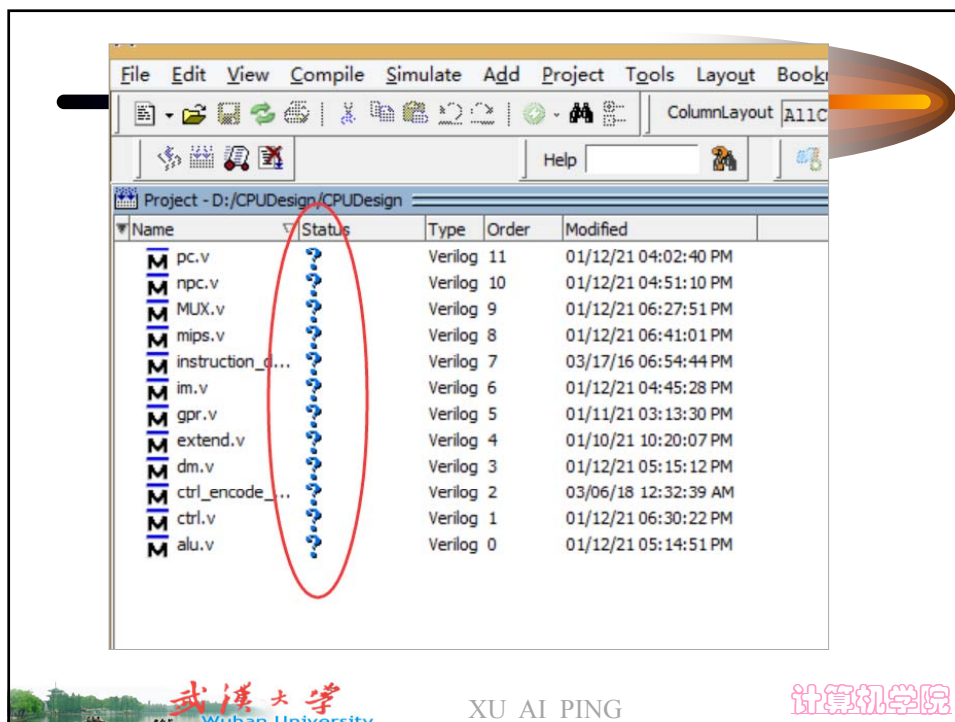
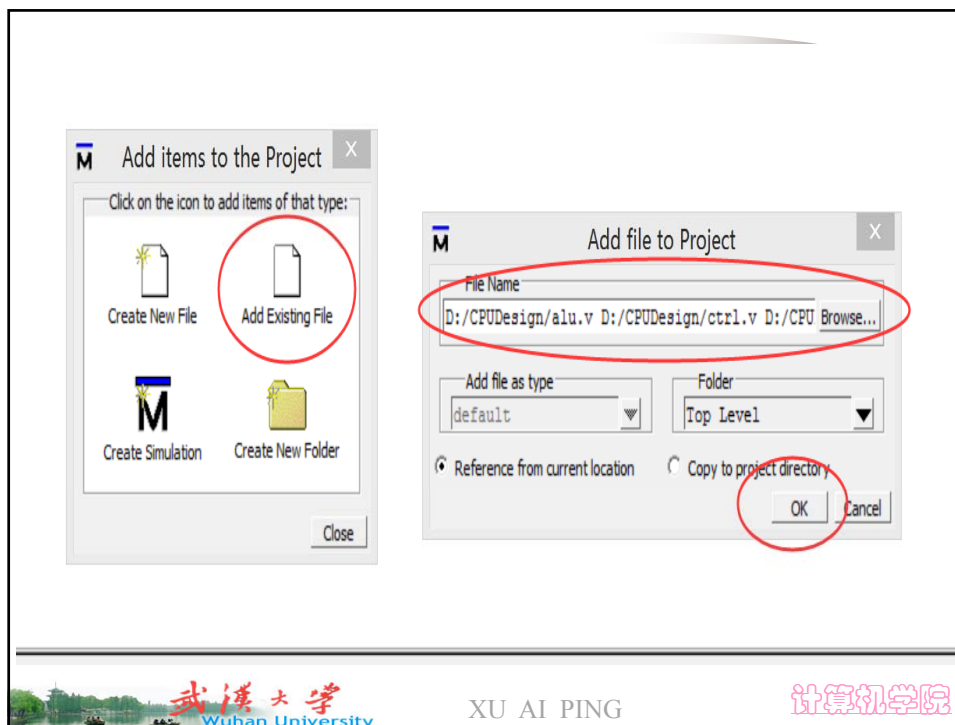
## ModelSim (看教程第5章)

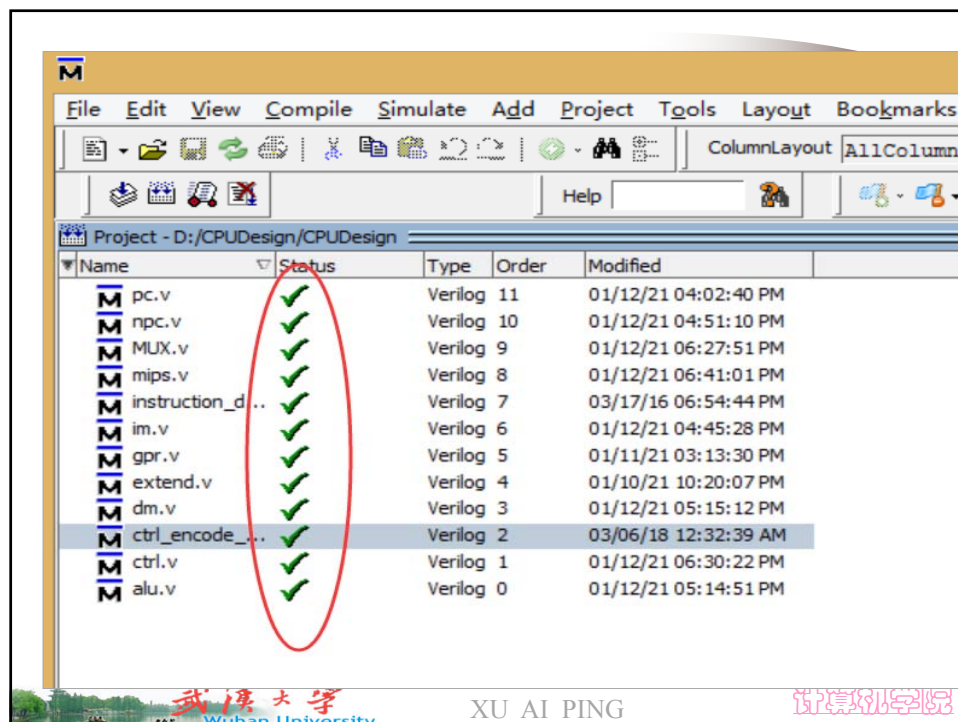
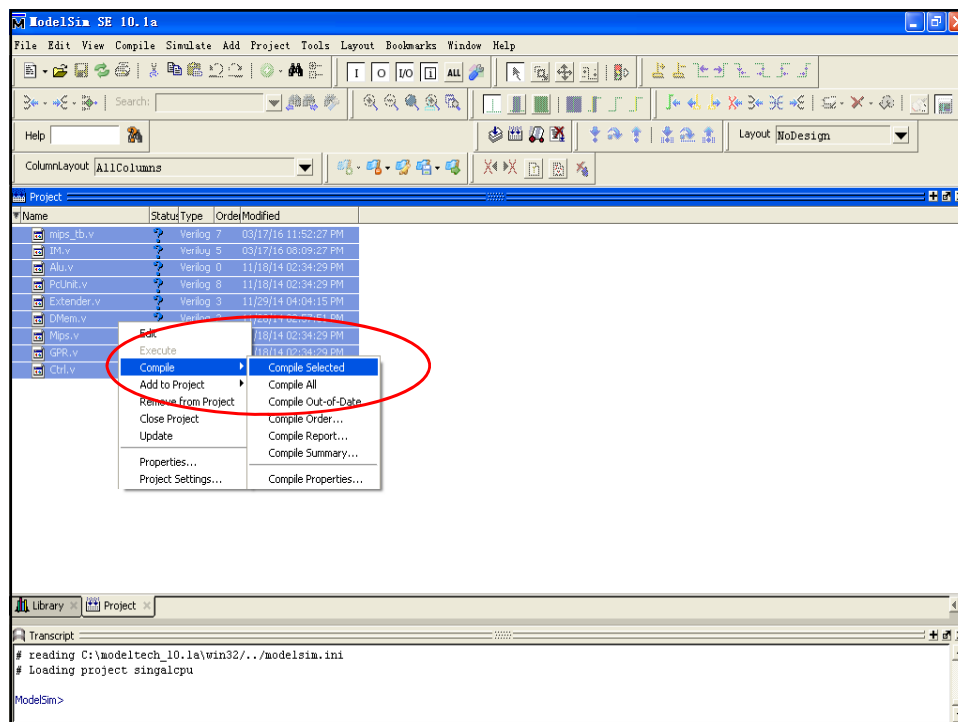
- Modelsim 仿真工具是 Model 公司开发的。它支持 Verilog、VHDL 以及他们的混合仿真，它可以将整个程序分步执行，使设计者直接看到他的程序下一步要执行的语句，而且在程序执行的任何步骤任何时刻都可以查看任意变量的当前值，通过仿真结果来验证结果是否正确。

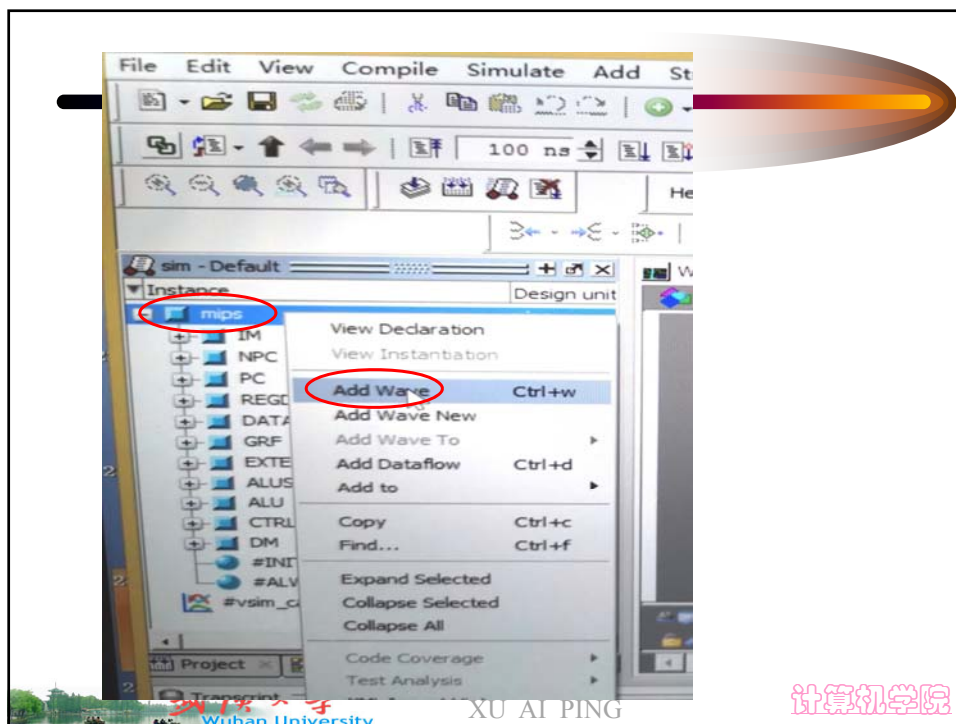
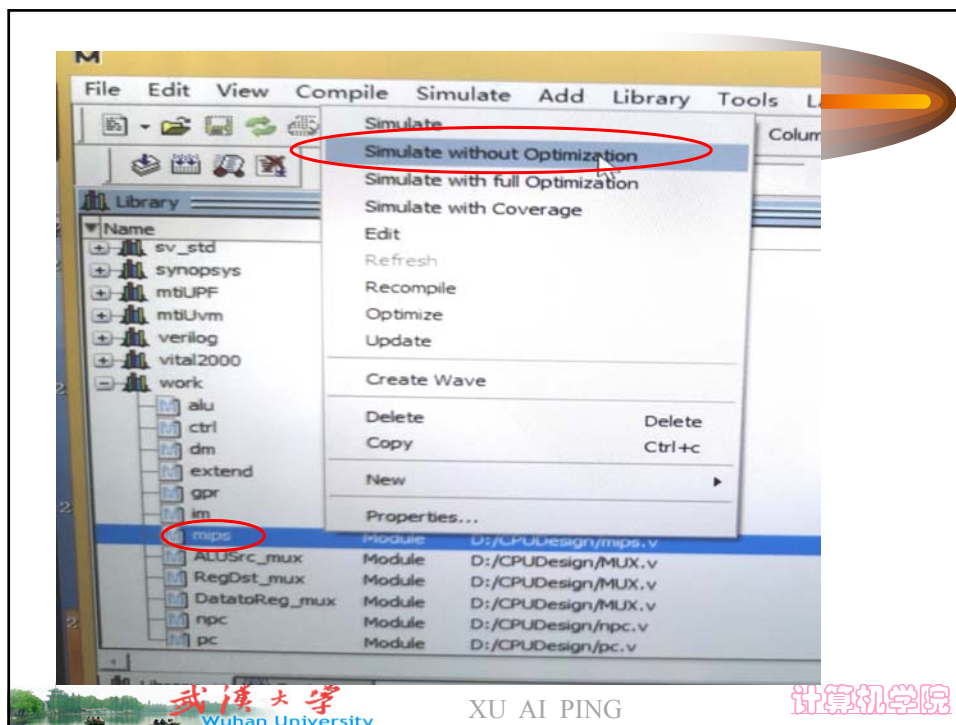


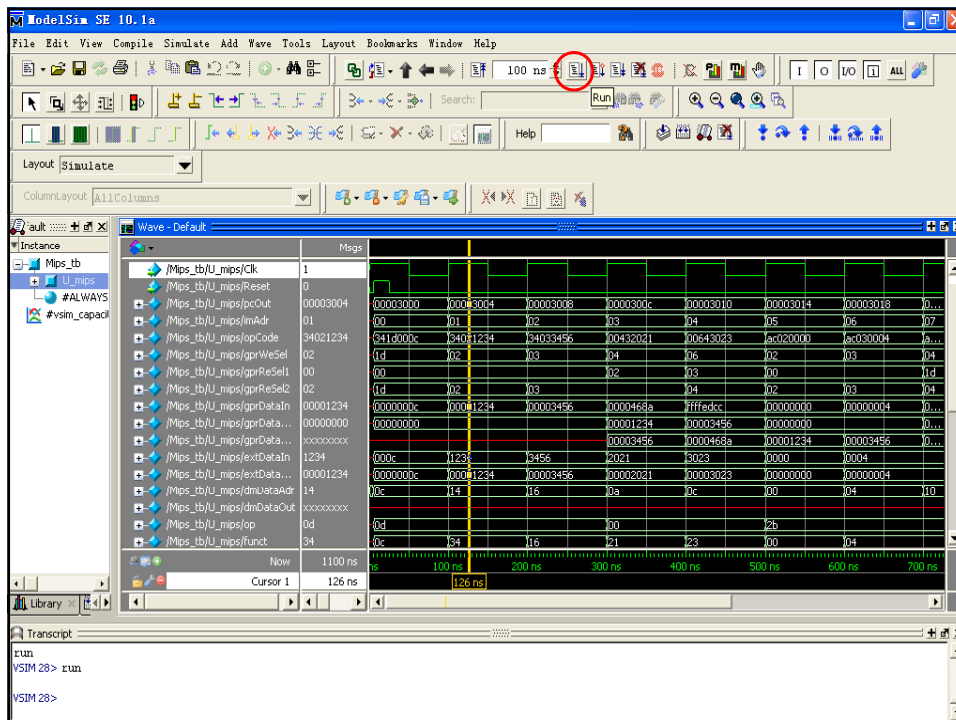
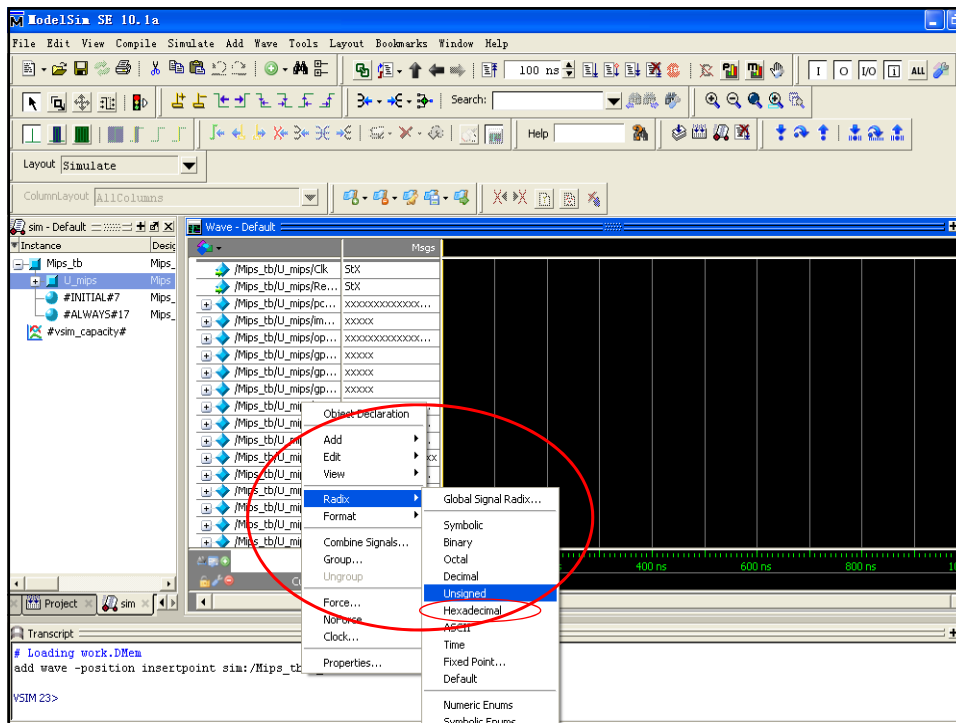
## cpudesign 演示

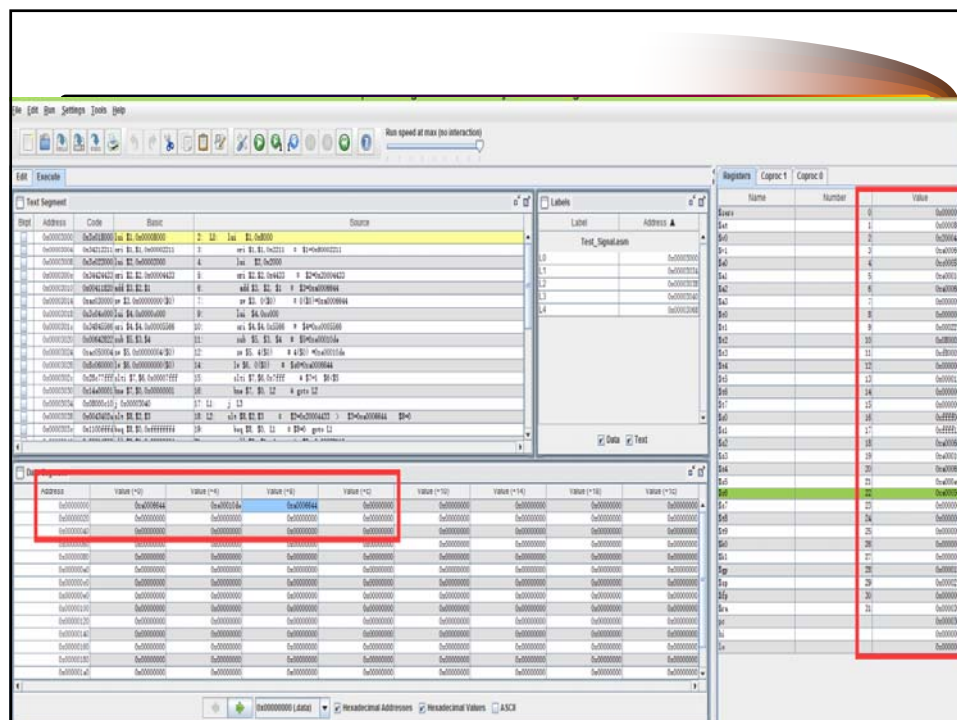
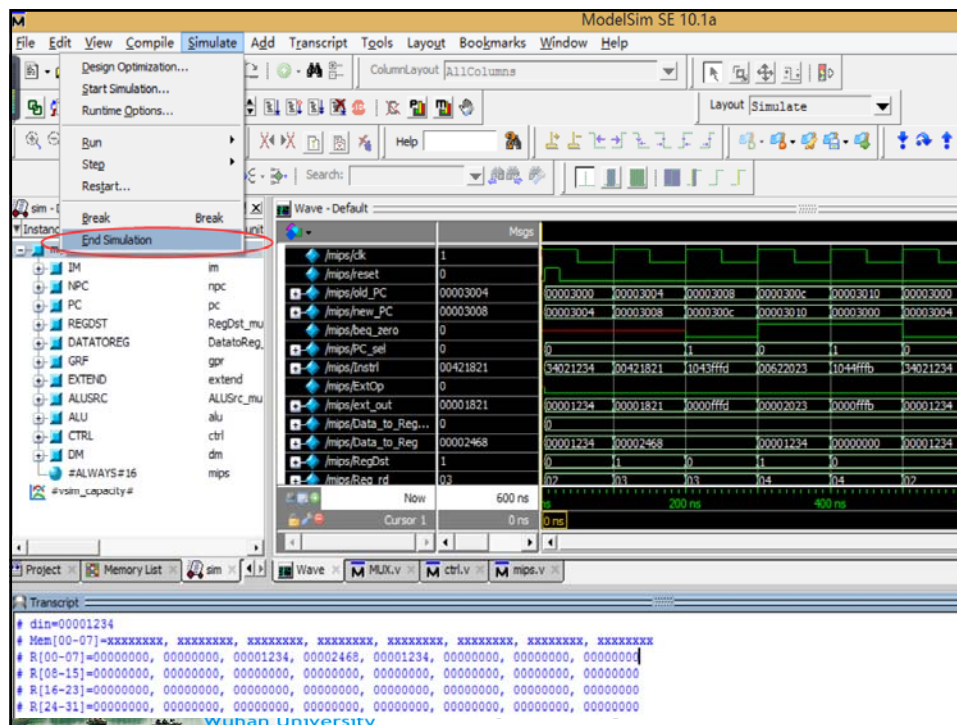




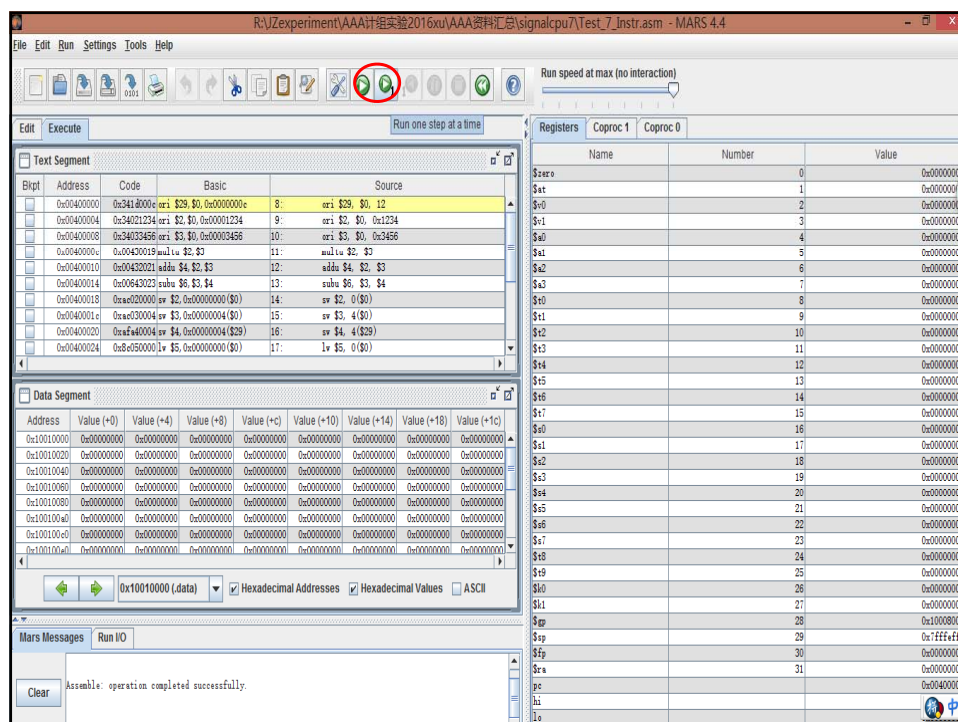
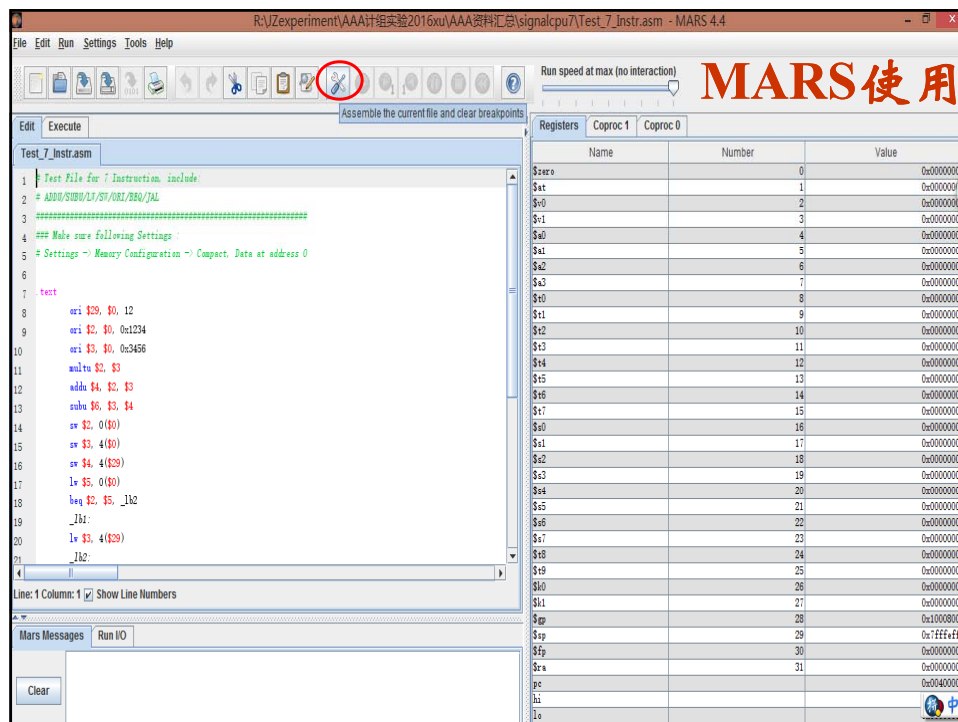




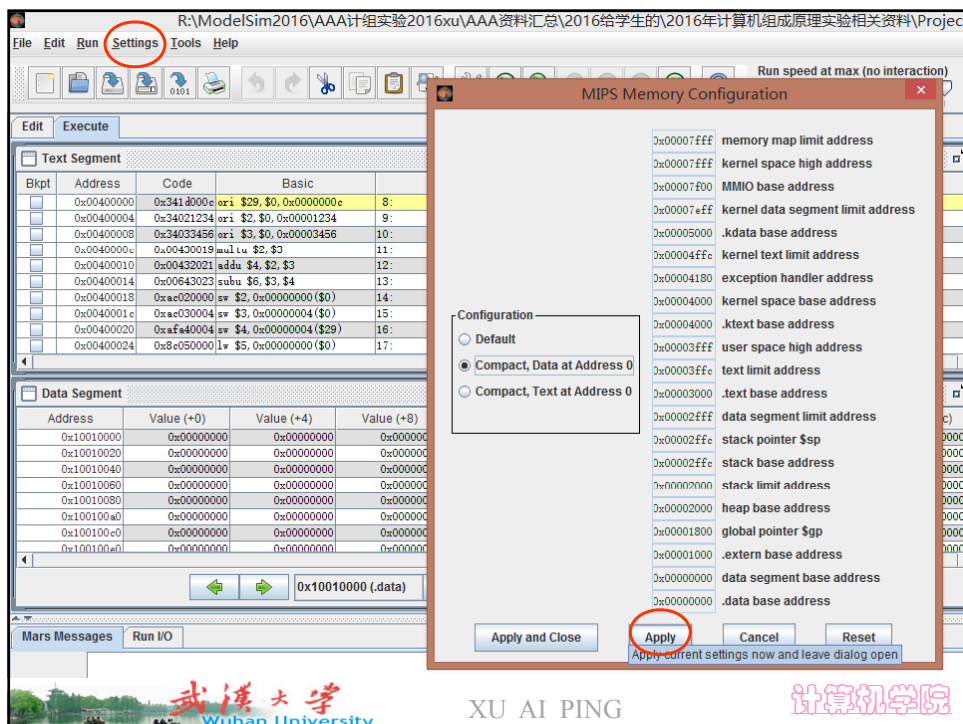
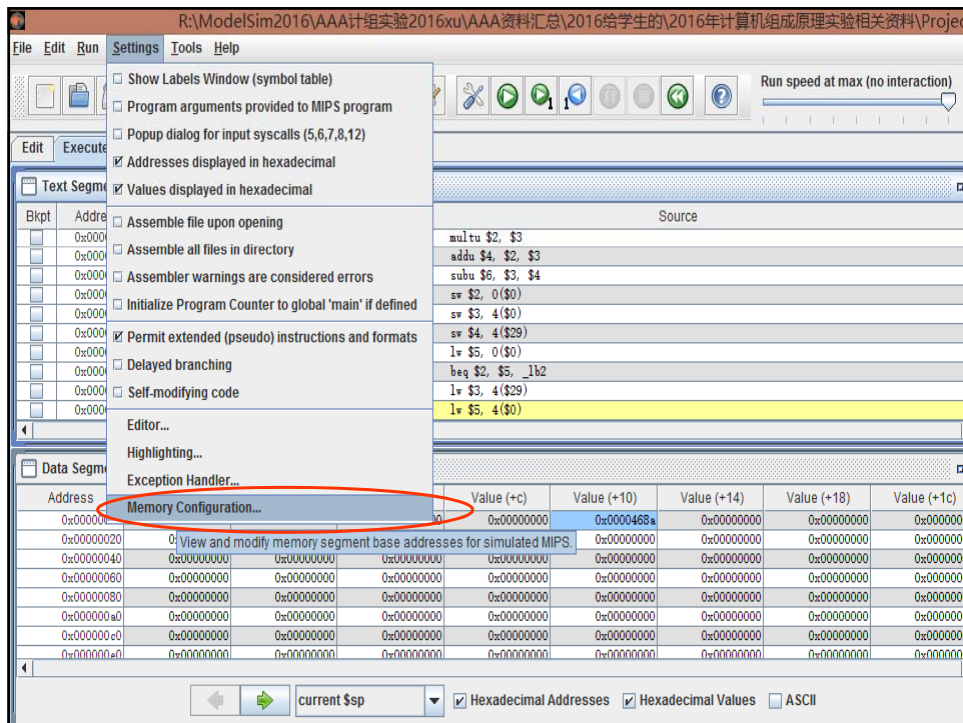


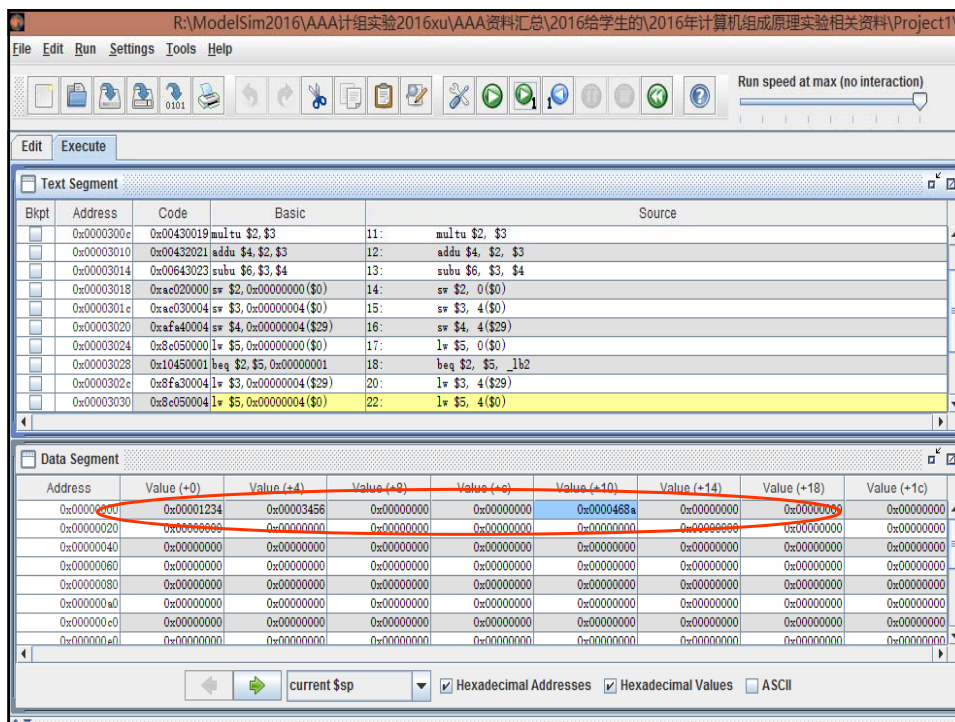












## 实验任务

- 设计单周期 CPU，支持如下指令集：
- { addu, subu, ori, beq, lw, sw, add, sub, or, and, addi, lui, sll, srl, sra, bne, j, jal, jr, slt, slti }
- 源代码：CPU\_Design; addu, subu, ori, beq 调试成功
- (1) 增加 lw, sw, add, sub, or, and, addi, lui, sll, srl, sra, bne, j, jal, jr, slt, slti，用 Test\_Instr.asm 测试，结果与 Mars 下执行的结果对比。
- (2) 用调试成功的指令编写一排序程序将如下的十六进制有符号数据在内存中按从小到大的顺序排序（要求用到子过程）并在实验板子上显示：  
10001008, 10001002, 80001001, 10001005, 80001000, FFF8000;
- (3) 设计流水线 CPU（完成数据相关控制和控制相关），仍然用 Test\_Instr.asm 测试



XU AI PING

计算机学院

## 成果提交形式

在5月1日之前将代码和实验报告电子版本发给老师，代码压缩打包，实验报告单独发：

实验报告文件名：班号\_学号姓名.docx

压缩文件名：班号\_学号姓名.后缀