计算机系统综合实验

计33　伍一鸣　2012011347

August 19, 2016

Contents

1 实验目标及完成情况

1.1 实验目标

实现32位mips指令系统的五级流水线CPU，在CPU上运行监控程序。

1.2 完成情况

完成了32位mips指令系统的五级流水线CPU，未能运行监控程序，在CPU上运行了一个fibonacci数列计算的程序。

2 项目分工

- 设计数据通路和指令集：杜华峰，郭栋，伍一鸣
- CPU代码实现与调试：杜华峰，郭栋
- 监控程序的修改：伍一鸣
- 所有文档撰写：伍一鸣

3 指令机器码

rd,rs,rt均为寄存器

3.1 逻辑操作

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	rs	rt	rd	00000	100100
指令格式	AND rd rs rt					
指令功能	R[d] ← R[s] & R[t]					
功能说明	将rs 与rt 的值相与后的结果保存至rd 中					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	rs	rt	rd	00000	100101
指令格式	OR rd rs rt					
指令功能	R[d] ← R[s] | R[t]					
功能说明	将rs 与rt 的值相或后的结果保存至rd 中					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	rs	rt	rd	00000	100110
指令格式	XOR rd rs rt					
指令功能	R[d] ← R[s] ∧ R[t]					
功能说明	将rs 与rt 的值相异或后的结果保存至rd 中					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	rs	rt	rd	00000	100111
指令格式	NOR rd rs rt					
指令功能	R[d] ←∼(R[s] | R[t])					
功能说明	将rs 与rt 的值或非后的结果保存至rd 中					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	001100	rs	rt	immediate		
指令格式	ANDI rt rs immediate					
指令功能	R[t] ← R[s] & Zero-extend(immediate)					
功能说明	将rs 的值与立即数零扩展后相与的结果保存至rt 中					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	001110	rs	rt	immediate		
指令格式	XORI rt rs immediate					
指令功能	R[t] ← R[s] ∧ Zero-extend(immediate)					
功能说明	将rs 的值与立即数零扩展后相异或的结果保存至rt 中					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	001111	00000	rt	immediate		
指令格式	LUI rt immediate					
指令功能	R[t] ← immediate * 65536					
功能说明	将16 位立即数放至rt 的高16 位中					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	001101	rs	rt	immediate		
指令格式	ORI rt rs immediate					
指令功能	R[t] ← R[s] | Zero-extend(immediate)					
功能说明	将rs 与立即数immediate 零扩展后相或的结果保存至rd 中					

3.2 移位操作

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	00000	rt	rd	immediate	000000
指令格式	SLL rd rt immediate					
指令功能	R[d] ← R[t] << immediate					
功能说明	将rt 中的值左移立即数immediate 位后的结果保存至rd 中					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	00000	rt	rd	immediate	000010
指令格式	SRL rd rt immediate					
指令功能	R[d] ← R[t] >> immediate(logical)					
功能说明	将rt 中的值逻辑右移立即数immediate 位后的结果保存至rd 中					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	00000	rt	rd	immediate	000011
指令格式	SRA rd rt immediate					
指令功能	R[d] ← R[t] >> immediate(arithmetic)					
功能说明	将rt 中的值算术右移立即数immediate 位后的结果保存至rd 中					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	rs	rt	rd	00000	000100
指令格式	SLLV rd rt rs					
指令功能	R[d] ← R[t] << R[s]					
功能说明	将rt 中的值左移rs 位后的结果保存至rd 中					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	rs	rt	rd	00000	000110
指令格式	SRLV rd rt rs					
指令功能	R[d] ← R[t] >> R[s](logical)					
功能说明	将rt 中的值逻辑右移rs 位后的结果保存至rd 中					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	rs	rt	rd	00000	000111
指令格式	SRAV rd rt rs					
指令功能	R[d] ← R[t] >> R[s](arithmetic)					
功能说明	将rt 中的值算术右移rs位后的结果保存至rd 中					

3.3 移动操作

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	rs	rt	rd	00000	001011
指令格式	MOVN rd rt rs					
指令功能	if rt $\neq$ 0 then rd $\leftarrow$ rs					
功能说明	若rt不为0，则将rs的值赋给rd					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	rs	rt	rd	00000	001010
指令格式	MOVZ rd rt rs					
指令功能	if rt = 0 then rd $\leftarrow$ rs					
功能说明	若rt为0，则将rs的值赋给rd					

3.4 算术操作

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	rs	rt	rd	00000	100001
指令格式	ADDU rd rs rt					
指令功能	R[d] ← R[s] + R[t]					
功能说明	将rs 与rt 的值相加后的结果保存至rd 中					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	rs	rt	rd	00000	100011
指令格式	SUBU rd rs rt					
指令功能	R[d] ← R[s] - R[t]					
功能说明	将rs 与rt 的值相减后的结果保存至rd 中					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	rs	rt	rd	00000	101010
指令格式	SLT rd rs rt					
指令功能	if(R[s] < R[t]) then R[d] = 1,else R[d] = 0					
功能说明	比较rs 与rt 的值并根据结果将rd 赋值					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	rs	rt	rd	00000	101011
指令格式	SLTU rd rs rt					
指令功能	if(R[s] < R[t]) then R[d] = 1,else R[d] = 0					
功能说明	比较rs 与rt 的无符号值并根据结果将rd 赋值					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	001001	rs	rt	immediate		
指令格式	ADDIU rt rs immediate					
指令功能	R[t] ← R[s] + (sign extended)immediate					
功能说明	对立即数immediate进行符号扩展后与rs的值求和，保存到rt中，不检查溢出					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	001010	rs	rt	immediate		
指令格式	SLTI rt rs immediate					
指令功能	if(R[s] < (sign extended)immediate)then R[t]=1,else R[t]=0					
功能说明	对立即数immediate进行符号扩展后与rs的值无符号比较并根据结果将rt 赋值					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	001011	rs	rt	immediate		
指令格式	SLTIU rt rs immediate					
指令功能	if(R[s] < (sign extended)immediate)then R[t]=1,else R[t]=0					
功能说明	对立即数immediate进行符号扩展后与rs的值有符号比较并根据结果将rt 赋值					

3.5 转移指令

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	rs	00000	00000	00000	001000
指令格式	JR rs					
指令功能	PC ← R[s]					
功能说明	无条件跳转至rs 中所存地址执行					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	rs	00000	rd	00000	001001
指令格式	JALR rd rs 或者JALR rs					
指令功能	PC ← R[s], R[d] ← RPC					
功能说明	无条件跳转至rs 中所存地址执行，将延时槽后一条指令的地址保存到rd中作为返回地址，rd默认为$31					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000010	instr index				
指令格式	J target					
指令功能	PC ← (PC+4)[31,28]||target*4					
功能说明	跳转至新地址执行，新地址低28位为target乘以4的值，新地址高4位为PC+4的高4位					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000011	instr index				
指令格式	JAL target					
指令功能	PC ← (PC+4)[31,28]||target*4，$31 ← RPC					
功能说明	跳转至新地址执行，新地址低28位为target乘以4的值，新地址高4位为PC+4的高4位,返回地址保存到$31中					

以上四条指令都要在转移之前先执行延迟槽指令

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000100	rs	rt	offset		
指令格式	BEQ rs rt offset					
指令功能	if (rs = rt) then PC = PC+4+(signed extend(offset * 4))					
功能说明	若rs等于rt则执行跳转操作					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000111	rs	00000	offset		
指令格式	BGTZ rs offset					
指令功能	if (rs > 0) then PC = PC+4+(signed extend(offset * 4))					
功能说明	若rs大于0则执行跳转操作					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000110	rs	00000	offset		
指令格式	BLEZ rs offset					
指令功能	if (rs $\leq$ 0) then PC = PC+4+(signed extend(offset * 4))					
功能说明	若rs不大于0则执行跳转操作					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000101	rs	rt	offset		
指令格式	BNE rs rt offset					
指令功能	if (rs $\neq$ rt) then PC = PC+4+(signed extend(offset * 4))					
功能说明	若rs不等于rt则执行跳转操作					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000001	rs	00000	offset		
指令格式	BLTZ rs offset					
指令功能	if (rs < 0) then PC = PC+4+(signed extend(offset * 4))					
功能说明	若rs小于0则执行跳转操作					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000001	rs	00001	offset		
指令格式	BLEZ rs offset					
指令功能	if (rs $\geq$ 0) then PC = PC+4+(signed extend(offset * 4))					
功能说明	若rs不小于0则执行跳转操作					

3.6 存储指令和空指令

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	100011	base	rt	offset		
指令格式	LW rt offset(base)					
指令功能	R[t] ← MEM[signed extended(offset)+GPR[base]]					
功能说明	从内存中指定的加载地址处，读取一个字，保存到rt中，要求地址对齐					

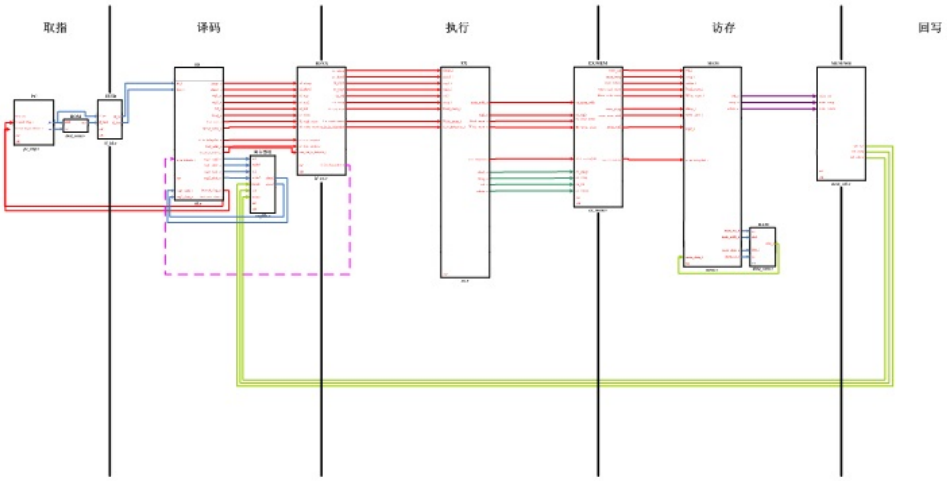
指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	101011	base	rt	offset		
指令格式	SW rt offset(base)					
指令功能	R[t] → MEM[signed extended(offset)+GPR[base]]					
功能说明	从rt处读取一个字，保存到内存中指定的加载地址中，要求地址对齐					

指令编码	31-26	25-21	20-16	15-11	10-6	5-0
	000000	00000	00000	00000	00000	000000
指令格式	NOP					
指令功能	无					
功能说明	空指令					

4 指令对比

16位指令	32位对应指令
ADDIU	ADDIU
ADDIU3	
ADDSP	
ADDU	ADDU
AND	AND
B	BEQ
BEQZ	XOR+BEQ
BNEZ	XOR+BNE
BTEQZ	XOR+BEQ
CMP	SLT+ADDU
JR	JR
LI	XOR+ADDIU
LW	LW
LW_SP	
MFIH	XOR+ADDU
MFPC	
MTIH	
MTSP	
NOP	NOP
OR	OR
SLL	SLL
SRA	SRA
SUBU	SUBU
SW	SW
SW_SP	

5 数据通路



6 流水线设计

6.1 取指阶段

- PC 模块：给出指令地址，其中实现指令寄存器PC，该寄存器的值就是指令地址。
- IF/ID模块：实现取指不译码阶段之间的寄存器，将取指阶段的结果在下一个时钟传递到译码阶段。

6.2 译码阶段

- ID 模块：对指令进行译码，译码结果包括运算类型、运算所需的源操作数、要写入的目的寄存器等。
- Regfile 模块：实现了32 个32 位通用寄存器，可以同时进行两个寄存器的读操作和一个寄存器的写操作。
- ID/EX 模块：实现译码不执行阶段之间的寄存器，将译码阶段的结果在下一个时钟周期传递到执行阶段。

6.3 执行阶段

- EX 模块：依据译码阶段的结果，进行指定的运算，给出运算结果。
- EX/MEM 模块：实现执行不访存阶段之间的寄存器，将执行阶段的结果在下一个时钟周期传递到访存阶段。

6.4 访存阶段

- MEM 模块：如果是加载、存储指令，那么会对数据存储器进行访问。
- MEM/WB 模块：实现访存不回写阶段之间的寄存器，将访存阶段的结果在下一个时钟周期传递到回写阶段。

6.5 回写阶段

- HILO 模块：实现寄存器HI、LO，在乘法指令的处理过程中会使用到这两个寄存器。

7 冲突问题

因为时间等因素，没有考虑冲突的问题，只是在fibonacci数列计算的程序中可能出现冲突的两条指令之间加上了4条NOP语句。

8 模块接口

8.1 PC模块

接口名	宽度	输入/输出	作用
rst	1	输入	复位信号
clk	1	输入	时钟信号
pc	32	输出	要读取的指令地址
ce	1	输出	指令存储器使能信号
branch_flag_i	1	输入	是否转移
branch_target_address_i	32	输入	转移地址
new_pc	32	输入	要读取的指令地址

8.2 Regfile模块

接口名	宽度	输入/输出	作用
rst	1	输入	复位信号
clk	1	输入	时钟信号
waddr	32	输出	要写入的寄存器地址
wdata	1	输出	要写入的数据
we	1	输入	写使能信号
raddr1	5	输入	第一个读端口地址
re1	1	输入	以一个读端口使能信号
rdata1	32	输出	第一个读端口的值
raddr2	5	输入	第二个读端口地址
re2	1	输入	以二个读端口使能信号
rdata2	32	输出	第二个读端口的值

8.3 ID模块

接口名	宽度	输入/输出	作用
rst	1	输入	复位信号
pc_i	32	输入	指令地址
inst_i	32	输入	译码阶段指令
reg1_data_i	32	输入	第一个读端口输入
reg2_data_i	32	输入	第二个读端口输入
reg1_read_o	1	输出	第一个读端口使能信号
reg2_read_o	1	输出	第二个读端口使能信号
reg1_addr_o	5	输出	第一个读端口地址
reg2_addr_o	5	输出	第二个读端口地址
aluop_o	8	输出	运算子类型
alusel_o	3	输出	运算类型
reg1_o	32	输出	源操作数1
reg2_o	32	输出	源操作数2
wd_o	5	输出	目的寄存器地址
wreg_o	1	输出	是否需要写入目的寄存器
ex_wreg_i	1	输入	处于执行阶段指令是否写
ex_wd_i	5	输入	处于执行阶段指令写地址
ex_wdata_i	32	输入	处于执行阶段指令写数据
mem_wreg_i	1	输入	处于访存阶段指令是否写
mem_wd_i	5	输入	处于访存阶段指令写地址
mem_wdata_i	32	输入	处于访存阶段指令写数据
branch_flag_o	1	输出	是否转移
branch_target_address_o	32	输出	转移目标地址
is_in_delayslot_o	1	输出	当前指令是否位于延迟槽
link_addr_o	32	输出	返回地址
next_inst_in_delayslot_o	1	输出	下一跳指令是否位于延迟槽
is_in_delayslot_i	1	输入	当前指令是否位于延迟槽

8.4 EX模块

接口名	宽度	输入/输出	作用
rst	1	输入	复位信号
aluop_i	8	输入	运算子类型
alusel_i	3	输入	运算类型
reg1_i	32	输入	源操作数1
reg2_i	32	输入	源操作数2
wd_i	5	输入	目的寄存器地址
wreg_i	1	输入	是否需要写入目的寄存器
wd_o	5	输出	目的寄存器地址
wreg_o	1	输出	是否需要写入目的寄存器
wdata_o	32	输出	写入目的寄存器的值
is_indelayslot_i	1	输出	是否位于延迟槽
link_address_i	32	输出	返回地址
mem_addr_o	32	输出	加载/存储地址
reg2_o	32	输出	要存的数据

8.5 MEM模块

接口名	宽度	输入/输出	作用
rst	1	输入	复位信号
wd_i	5	输入	目的寄存器地址
wreg_i	1	输入	是否需要写入目的寄存器
wdata_i	32	输入	目的寄存器的值
wd_o	5	输出	目的寄存器地址
wreg_o	1	输出	是否需要写入目的寄存器
wdata_o	32	输出	写入目的寄存器的值
reg2_i	32	输出	要存储的数据
mem_data_i	32	输入	读取的数据
mem_addr_i	32	输入	加载/存储地址
is_write	1	输出	是否写ram

9 监控程序

后来目标有所修改，所以监控程序并未完成，比较有用的代码是按照原本16位term的方式打的两张32位的表。

```
const AsmID RegsList[32]={
        {"zero",0},
        {"at",1},
        {"v0",2},
        {"v1",3},
        {"a0",4},
        {"a1",5},
        {"a2",6},
        {"a3",7},
        {"t0",8},
        {"t1",9},
        {"t2",10},
        {"t3",11},
        {"t4",12},
        {"t5",13},
        {"t6",14},
        {"t7",15},
        {"s0",16},
        {"s1",17},
        {"s2",18},
        {"s3",19},
        {"s4",20},
        {"s5",21},
        {"s6",22},
        {"s7",23},
        {"t8",24},
        {"t9",25},
        {"k0",26},
        {"k1",27},
        {"gp",28},
        {"sp",29},
        {"fp",30},
        {"ra",31},
};
```

```
const struct TASM const_asm[] = {
    {"AND",0X3FFF824,0XFC1FFFFF,0XFFE0FFFF,0XFFFF07FF},
    {"OR",0X3FFF825,0XFC1FFFFF,0XFFE0FFFF,0XFFFF07FF},
    {"XOR",0X3FFF826,0XFC1FFFFF,0XFFE0FFFF,0XFFFF07FF},
    {"NOR",0X3FFF827,0XFC1FFFFF,0XFFE0FFFF,0XFFFF07FF},

    {"ANDI",0X33FF0000,0XFC1FFFFF,0XFFE0FFFF,0XFFFF0000},
    {"XORI",0X3BFF0000,0XFC1FFFFF,0XFFE0FFFF,0XFFFF0000},
    {"LUI",0X3C1F0000,0XFFE0FFFF,0XFFFF0000,0XFFFFFFFF},
    {"ORI",0X37FF0000,0XFC1FFFFF,0XFFE0FFFF,0XFFFF0000},

    {"SLL",0X1FF800,0XFFE0FFFF,0XFFFF07FF,0XFFFFF83F},
    {"SRL",0X1FF802,0XFFE0FFFF,0XFFFF07FF,0XFFFFF83F},
    {"SRA",0X1FF802,0XFFE0FFFF,0XFFFF07FF,0XFFFFF83F},

    {"SLLV",0X3FFF804,0XFC1FFFFF,0XFFE0FFFF,0XFFFF07FF},
    {"SRLV",0X3FFF806,0XFC1FFFFF,0XFFE0FFFF,0XFFFF07FF},
    {"SRAV",0X3FFF807,0XFC1FFFFF,0XFFE0FFFF,0XFFFF07FF},

    {"MOVN",0X3FFF80B,0XFC1FFFFF,0XFFE0FFFF,0XFFFF07FF},
    {"MOVZ",0X3FFF80A,0XFC1FFFFF,0XFFE0FFFF,0XFFFF07FF},

    {"ADDU",0X3FFF821,0XFC1FFFFF,0XFFE0FFFF,0XFFFF07FF},
    {"SUBU",0X3FFF823,0XFC1FFFFF,0XFFE0FFFF,0XFFFF07FF},
    {"SLT",0X3FFF82A,0XFC1FFFFF,0XFFE0FFFF,0XFFFF07FF},
    {"SLTU",0X3FFF82B,0XFC1FFFFF,0XFFE0FFFF,0XFFFF07FF},

    {"SLTI",0X281F0000,0XFFE0FFFF,0XFFFF0000,0XFFFFFFFF},
    {"SLTIU",0X2CFF0000,0XFC1FFFFF,0XFFE0FFFF,0XFFFF0000},

    {"JR",0X3E00008,0XFC1FFFFF,0XFFFFFFFF,0XFFFFFFFF},
    {"JALR",0X3E00009,0XFC1FFFFF,0XFFFF07FF,0XFFFFFFFF},
    {"J",0X10000000,0XFC000000,0X0XFFFFFFFF,0XFFFFFFFF},
    {"JAL",0X18000000,0XFC000000,0X0XFFFFFFFF,0XFFFFFFFF},

    {"BEQ",0X13FF0000,0XFC1FFFFF,0XFFE0FFFF,0XFFFF0000},
    {"BNE",0X17FF0000,0XFC1FFFFF,0XFFE0FFFF,0XFFFF0000},

    {"BGTZ",0X1FE00000,0XFC1FFFFF,0XFFFF0000,0XFFFFFFFF},
    {"BLEZ",0X1BE00000,0XFC1FFFFF,0XFFFF0000,0XFFFFFFFF},

    {"BGEZ",0X7E10000,0XFC1FFFFF,0XFFFF0000,0XFFFFFFFF},
    {"BLTZ",0X7E00000,0XFC1FFFFF,0XFFFF0000,0XFFFFFFFF},

    {"LW",0X8FFF0000,0XFC1FFFFF,0XFFE0FFFF,0XFFFF0000},
    {"SW",0XAFFF0000,0XFC1FFFFF,0XFFE0FFFF,0XFFFF0000}
};
```

10 收获与总结

通过这次大作业的实践，让我们三个对于计算机组成原理的课程有了较为完整一次复习，对于计原的那些概念性的东西有了更深的理解。

在这次大作业的实践中让我们对于计划赶不上变化有了深刻的理解，以后在作计划的时候要一切往前并留出50%的缓冲时间才能保证计划的顺利完成。