计算机组成原理实验报告

16061007 王文珺

一、 CPU 设计文档

（一） 数据通路设计

数据通路的设计和 p6 的区别主要在于添加了与外界进行数据交流的 Bridge，协处理器 CP0，两个模拟外部中断的 Timer，并且在原本的数据通路中添加相应的端口以满足数据通路要求。Bridge 采用划分地址空间的方式来与外部通信，计时器主要功能是根据设定的时间来定时产生中断信号。

对于转发和暂停机制，由于我们之前使用的需求者供给者模型同样适用，而且新加入的指令也没有产生新类型的转发，所以所有的指令都可以对应到已有的各种转发和暂停中。新增指令的具体的实现转发和暂停控制信号我们在控制器设计模块进行详细阐述，接下来给出数据通路中所有部件的端口说明。

IF：

序号	端口名称	I\0	功能名称	功能描述
1	Clk	I	时钟	提供时钟信号
2	Reset	I	复位	当复位信号有效时，PC 被设置为 0x00003000
3	Branch	I	跳转	当 Branch 信号有效时，将跳转地址赋给 PC
4	Stall	I	暂停	当 Stall 信号有效时，冻结 PC
5	nextPC	I	跳转地址	提供跳转地址
6	Instr	O	指令	输出正在执行的指令
7	ADD4	O	下条地址	非跳转情况下的下条指令地址
8	Bfail	I	B 类指令失败	当 bfail 有效时，pc 下条地址为正常+4
9	EXLSet	I	异常	提供异常信号，PC 跳转到 4180
10	IntReq	I	中断	提供中断信号，PC 跳转到 4180
11	Eret_E	I	Eret 指令	当执行 Eret 指令，将 EPC 的值赋给 PC
12	PC_Exception	O	取指异常	输出取指异常信号

D_PIPE:

序号	端口名称	I\0	功能名称	功能描述
1	Clk	I	时钟	提供时钟信号
2	Reset	I	复位	当复位信号有效时，复位清零 D 级流水寄存器
3	Instr	I	指令	将上级指令存入 D 级流水寄存器
4	Stall	I	暂停	当 Stall 信号有效时，冻结 D 级流水寄存器
5	ADD4	I	下条地址	将上级指令的下条地址存入 D 级流水寄存器
6	Instr_D	O	D 级指令	输出 D 级流水寄存器的指令

7	PC4_D	O	D 级下条地址	输出 D 级流水寄存器的指令的下条地址
8	ExcCode	I	上级异常编码	输入上级异常编码
9	ExcCode_D	O	本级异常编码	输出本级异常编码

NPC:

序号	端口名称	I\0	功能名称	功能描述
1	Imm16	I	16 位立即数	提供 16 位立即数
2	Imm26	I	26 位立即数	提供 26 位立即数
3	PC4	I	当前 PC+4	提供当前的 PC+4
4	jrpc	I	跳转地址	提供从寄存器读出的地址
5	NPC_Sel	I	选择信号	0000/beq 跳转，0001/j，jal 跳转，0010/jr 跳转 0011/bne 跳转，0100/blez 跳转，0101/bltz 跳转 0110/bgez 跳转，0111/bgtz 跳转
6	CMPResult	I	比较结果	0/两数据不相等，1/两数据相等
7	OVERZERO	I	Rs 大于 0	用于判断 b 类指令跳转成功与否
8	BELOWZERO	I	Rs 小于 0	用于判断 b 类指令跳转成功与否
9	Bfail	O	B 类指令失败	指令为 b 类但是跳转条件无效时为 1
10	nPC	O	下条地址	输出下条地址

GRF:

序号	端口名称	I\0	功能名称	功能描述
1	Rreg1	I	寄存器 1 选择	通过多路选择器选择寄存器 1
2	Rreg2	I	寄存器 2 选择	通过多路选择器选择寄存器 2
3	Wreg	I	选择写入寄存器	当控制信号 RegWrite 有效时， 在时钟上升沿将 Wdata 中的数据写入 Wreg 通过 Decoder 选择的寄存器中，
4	RegWrite	I	写入控制	
5	Wdata	I	写入的数据	
6	Rdata1	O	数据 1 输出	输出寄存器 1 的数据
7	Rdata2	O	数据 2 输出	输出寄存器 2 的数据
8	clk	I	时钟	时钟信号
9	reset	I	复位	当 reset 信号有效时，将所有寄存器值清零
10	PrePC	I	当前地址	用于输出当前指令地址

CMP:

序号	端口名称	I\0	功能名称	功能描述
1	CMPD1	I	比较数 1	提供第一个比较数
2	CMPD2	I	比较数 2	提供第二个比较数
3	CMPResult	O	比较结果	比较结果：0/不相等 1/相等
4	OVERZERO	O	Rs 大于 0	Rs 大于 0 时此信号为 1
5	BELOWZERO	O	Rs 小于 0	Rs 小于 0 的此信号为 1

EXT：

序号	端口名称	I\0	功能名称	功能描述
1	Imm16	I	16 位立即数	提供被扩展的立即数

2	ExtOp	I	扩展信号	0/无符号扩展 1/符号扩展
3	Imm32	O	32 位立即数	输出扩展后的立即数

E_PIPE：

序号	端口名称	I\0	功能名称	功能描述
1	Clk	I	时钟	提供时钟信号
2	Reset	I	复位	当复位信号有效时，复位清零 E 级流水寄存器
3	Instr_D	I	D 级指令	将 D 级指令存入 E 级流水寄存器
4	Stall	I	暂停	当 stall 信号有效，复位清零 E 级流水寄存器
5	PC4_D	I	D 级下条地址	将 D 级下条地址存入 E 级流水寄存器
6	Resin	I	写入 res_e	将 res_e 的值存入 E 级流水寄存器
7	RF_RD_1	I	数据 1	将数据 1 存入 E 级流水寄存器
8	RF_RD_2	I	数据 2	将数据 2 存入 E 级流水寄存器
9	E32	I	32 位立即数	将 32 位立即数存入 E 级流水寄存器
10	MUX_A3	I	写入寄存器	将写入寄存器存入 E 级流水寄存器
11	Res_E	O	E 级 res_e	输出当前 E 级 res_e
12	Instr_E	O	E 级指令	输出当前 E 级指令
13	PC4_E	O	E 级下条地址	输出当前 E 级下条地址
14	V1_E	O	E 级数据 1	输出当前 E 级数据 1
15	V2_E	O	E 级数据 2	输出当前 E 级数据 2
16	A1_E	O	E 级寄存器 1	输出当前 E 级寄存器 1
17	A2_E	O	E 级寄存器 2	输出当前 E 级寄存器 2
18	A3_E	O	E 级写入寄存器	输出当前 E 级写入寄存器
19	E32_E	O	E 级 32 位立即数	输出当前 E 级 32 位立即数
20	Shamt	O	移位	输出用于移位的指令段
21	ExcCode	I	上级异常编码	输入上级异常编码
22	ExcCode_E	O	本级异常编码	输出本级异常编码
23	Eret_E	O	Eret 指令	识别到 Eret 指令

ALU:

序号	端口名称	I\0	功能名称	功能描述
1	A	I	数据 1	提供运算的数据 1
2	B	I	数据 2	提供运算的数据 2
3	Shamt	I	移位	提供移位的数据
4	ALUOp	I	计算信号	0000/加法，0001/减法，0010/或，0011/加载至高 16 位，0100/逻辑左移，0101/逻辑可变左移，0110/算术右移，0111/算术可变右移，1000/逻辑右移，1001/逻辑可变右移，1010/与，1011/异或，1100/或非，1101/有符号小于置 1，1110/无符号小于置 1
5	C	O	计算结果	输出计算结果
6	OverFlow	O	溢出	算术溢出信号

MultDiv:

序号	端口名称	I\0	功能名称	功能描述

1	Clk	I	时钟	提供时钟信号
2	Reset	I	复位	当复位信号有效时，复位清零 HI,LO 寄存器
3	Instr	I	指令	提供操作信号
4	Data1	O	数据 1	提供运算的数据 1
5	Data2	O	数据 2	提供运算的数据 2
6	HI	O	HI 寄存器	HI 寄存器的值
7	LO	O	LO 寄存器	LO 寄存器的值
8	Busy	O	正在计算	乘法持续 5 个周期，除法持续 10 个周期
9	Start	O	计算启动	识别到正在处理乘除法指令时有效 1 个周期
10	MD_EN	I	写入使能	当写入使能信号有效时才能进行写入

M_PIPE：

序号	端口名称	I\0	功能名称	功能描述
1	Clk	I	时钟	提供时钟信号
2	Reset	I	复位	当复位信号有效时，复位清零 M 级流水寄存器
3	Instr_E	I	E 级指令	将 E 级指令存入 M 级流水寄存器
4	PC4_E	I	E 级下条地址	将 E 级下条地址存入 M 级流水寄存器
5	res_E	I	E 级 res_e	将 res_e 的值存入 M 级流水寄存器
6	AO	I	计算结果	将 ALU 输出结果存入 M 级流水寄存器
7	V2_E	I	写入数据	将写入数据存入 M 级流水寄存器
8	A2_E	I	数据对应寄存器	将数据对应寄存器存入 M 级流水寄存器
9	A3_E	I	写入寄存器	将写入寄存器存入 M 级流水寄存器
10	res_M	O	M 级 res_m	输出当前 M 级 res_m
11	Instr_M	O	M 级指令	输出当前 M 级指令
12	PC4_M	O	M 级下条地址	输出当前 M 级下条地址
13	V2_M	O	M 级写入数据	输出当前 M 级写入数据
14	A2_M	O	M 级数据寄存器	输出当前 M 级数据对应寄存器
15	A3_M	O	M 级写入寄存器	输出当前 M 级写入寄存器
16	AO_M	O	M 级计算结果	输出当前 M 级计算结果
17	ExcCode	I	上级异常编码	输入上级异常编码
18	ExcCode_M	O	本级异常编码	输出本级异常编码

BE_EXT：

序号	端口名称	I\0	功能名称	功能描述
1	Instr	I	指令	通过该指令判断是否是 store 型指令
2	ADDR	I	地址低两位	提供地址低两位判断哪些字节需要写入
3	BE	O	写入信号	1111/sw 四个字节全部写入 1100/sh 写入到前两个字节 0011/sh 写入到后两个字节 1000/sb 写入到第一个字节 0100/sb 写入到第二个字节

				0010/sb 写入到第三个字节 0001/sb 写入到第四个字节

DM：

序号	端口名称	I\0	功能名称	功能描述
1	reset	I	复位	当复位信号有效时，DM 中的数据被全部清零
2	Address	I	32 位地址	利用 Splitter 选择 DM 中存储的某条数据
3	WriteData	I	写入的数据	当 MemWrite 信号有效时，将 WriteData 写入 Address 所选择的地址在 DM 中的位置
4	MemWrite	I	数据写入	
5	ReadData	O	读取的数据	输出 DM 被选择的位置所保存的数据
6	clk	I	时钟	时钟信号
7	PC	I	当前地址	用于输出当前指令地址
8	BE	I	字节使能	指明需要写入的字节
9	Instr	I	当前指令	用于判断异常
10	A	O	地址低两位	输出地址低两位
11	Addr_Exception0	O	取数异常	输出取数异常
12	Addr_Exception1	O	存数异常	输出存数异常

Data_EXT：

序号	端口名称	I\0	功能名称	功能描述
1	Instr	I	指令	通过该指令判断是否是 load 指令
2	Data	I	数据	提供 DM 读出的数据
3	A	I	地址低两位	通过该地址判断读出哪些字节
4	Dout	O	处理数据	经过处理的字节

CP0：

序号	端口名称	I\0	功能名称	功能描述
1	Clk	I	时钟	时钟信号
2	Reset	I	复位	复位 CP0 的寄存器值
3	A1	I	寄存器编号	提供读写的寄存器编号
4	DataIn	I	输入数据	写入的数据
5	PC4	I	当前地址+4	用于存储至 EPC
6	HWInt	I	外部中断	用于响应外部中断
7	We	I	写使能	用于写入 CP0 的寄存器
8	BD	I	延迟槽指令	判断当前异常是否为延迟槽指令
9	EXLClr	I	清除置位	清除 EXL 置位
10	EXCCODE	I	异常编码	输入最终的异常编码
11	EXLSet	O	异常	异常信号
12	IntReq	O	中断	中断信号
13	EPC	O	EPC 寄存器值	输出 EPC 寄存器值
14	DataOut	O	输出数据	输出读取 CP0 的寄存器值

W_PIPE:

序号	端口名称	I\0	功能名称	功能描述
1	Clk	I	时钟	提供时钟信号
2	Reset	I	复位	当复位信号有效时，复位清零 W 级流水寄存器
3	Instr_M	I	M 级指令	将 E 级指令存入 W 级流水寄存器
4	PC4_M	I	M 级下条地址	将 E 级下条地址存入 W 级流水寄存器
5	res_M	I	M 级 res_m	将 res_m 的值存入 W 级流水寄存器
6	AO_M	I	计算结果	将 ALU 输出结果存入 W 级流水寄存器
7	DM	I	读出数据	将 DM 读出数据存入 W 级流水寄存器
8	A3_M	I	写入寄存器	将写入寄存器存入 W 级流水寄存器
9	res_W	O	W 级 res_w	输出当前 W 级 res_w
10	Instr_W	O	W 级指令	输出当前 W 级指令
11	PC4_W	O	W 级下条地址	输出当前 W 级下条地址
12	A3_W	O	W 级写入寄存器	输出当前 W 级写入寄存器
13	AO_W	O	W 级计算结果	输出当前 W 级计算结果
14	DM_W	O	W 级读出数据	输出当前 W 级读出数据
15	EXLClr	O	清除 EXL 置位	识别到 Eret 指令时有效

（二） 控制器设计

控制器设计方面，主要由两种控制器组成，一种生成是正常的控制信号，一种生成转发和暂停信号。正常的控制信号通过和之前 project 一样的方法生成。

冲突控制器方面，继续采用高小鹏老师讲义中的方法，设置 E 级 Res 寄存器的初值，并进行简单传递，把每条新加的指令都通过其 rs 和 rt 的 Tuse 和 Tnew 把其判定条件并入原本已有的暂停和转发中即可。

在并入所有新增指令时都需要考虑该指令用到了 rs 还是 rt，他的 Tnew 在哪里产生，并且并入相应的 Tuse 行列。

```
assign Tuse_rt2=(op==`SW)||(op==`SB)||(op==`SH)||(op==`COP0&&Instr_D[25:21]==`MTC0);
```

至于暂停，此处增加有关 eret 指令的暂停，用于避免产生新的转发

```
assign Stall_ERET=(Instr_D==`ERET&&Instr_E[31:26]==`COP0&&Instr_E[25:21]==`MTC0)||(Instr_D==`ERET&&Instr_M[31:26]==`COP0&&Instr_M[25:21]==`MTC0);
```

由于不同的控制信号是在不同的级别使用的，所以这里使用了分布式译码，仅在需要此控制信号的级别传入该信号。即实例化四个 control 单元。

（三） 测试程序

```
ori $29,0x2ffc
ori $28,0x1800
ori $2,0xfc01
mtc0 $2,$12
```

```
        mfc0 $3,$12
        li $7,-2147483648
        subi $7,$7,1
        lui $4,0x7fff
        lui $5,0x7fff
        add $6,$4,$5
        nop
        lw $9,2($0)
        lui $8,1
        lui $10,0x6666
        ori $10,0x6666
        lw $9,0($10)
        lui $8,2
        lh $9,0($10)
        lui $8,3
        lh $9,3($0)
        lui $8,4
        lb $9,0($10)
        sw $8,0($10)
        lui $8,5
        sh $8,0($10)
        lui $8,6
        sb $8,0($10)
        lui $8,7
        sw $8,2($0)
        lui $8,8
        sh $8,3($0)
        lui $8,9


.ktext 0x4180
_entry:
        ori    $k0, $0, 0x1000
        sw    $sp, -4($k0)
        mfc0        $k1, $12
        sw    $k1, -8($k0)

        addiu       $k0, $k0, -256
        move        $sp, $k0

        j       _save_context
        nop

_main_handler:
```

```
        mfc0    $k0, $14
        addu    $k0, $k0, 4
        mtc0    $k0, $14
        j       _restore_context
        nop

_restore:
        eret

_save_context:
        sw      $1, 4($sp)
                sw      $2, 8($sp)
                sw      $3, 12($sp)
                sw      $4, 16($sp)
                sw      $5, 20($sp)
                sw      $6, 24($sp)
                sw      $7, 28($sp)
                sw      $8, 32($sp)
                sw      $9, 36($sp)
                sw      $10, 40($sp)
                sw      $11, 44($sp)
                sw      $12, 48($sp)
                sw      $13, 52($sp)
                sw      $14, 56($sp)
                sw      $15, 60($sp)
                sw      $16, 64($sp)
                sw      $17, 68($sp)
                sw      $18, 72($sp)
                sw      $19, 76($sp)
                sw      $20, 80($sp)
                sw      $21, 84($sp)
                sw      $22, 88($sp)
                sw      $23, 92($sp)
                sw      $24, 96($sp)
                sw      $25, 100($sp)
                sw      $26, 104($sp)
                sw      $27, 108($sp)
                sw      $28, 112($sp)
                sw      $29, 116($sp)
                sw      $30, 120($sp)
                sw      $31, 124($sp)
        mfhi    $k0
        sw  $k0, 128($sp)
        mflo    $k0
```

```
        sw    $k0, 132($sp)
        j     _main_handler
        nop



_restore_context:
        lw    $1, 4($sp)
              lw    $2, 8($sp)
              lw    $3, 12($sp)
              lw    $4, 16($sp)
              lw    $5, 20($sp)
              lw    $6, 24($sp)
              lw    $7, 28($sp)
              lw    $8, 32($sp)
              lw    $9, 36($sp)
              lw    $10, 40($sp)
              lw    $11, 44($sp)
              lw    $12, 48($sp)
              lw    $13, 52($sp)
              lw    $14, 56($sp)
              lw    $15, 60($sp)
              lw    $16, 64($sp)
              lw    $17, 68($sp)
              lw    $18, 72($sp)
              lw    $19, 76($sp)
              lw    $20, 80($sp)
              lw    $21, 84($sp)
              lw    $22, 88($sp)
              lw    $23, 92($sp)
              lw    $24, 96($sp)
              lw    $25, 100($sp)
              lw    $26, 104($sp)
              lw    $27, 108($sp)
              lw    $28, 112($sp)
              lw    $29, 116($sp)
              lw    $30, 120($sp)
              lw    $31, 124($sp)
        lw    $k0, 128($sp)
        mthi        $k0
        lw    $k0, 132($sp)
        mtlo        $k0
              j     _restore
        nop
```

二、 思考题

1、我们计组课程一本参考书目标题中有“硬件/软件接口”字样，那么到底什么是“硬件/软件接口”

硬件/软件接口在我看来就是为了达到硬件与软件协同一致，需要利用接口互通数据，信号，异常，中断等来达到通过软件支配硬件行为，而硬件行为也对应着不同的软件行为，从而达到互相配合和协同统一，就像大脑和躯干的关系，需要血液和神经信号的交流来达到高度协调统一，而这个互通有无的通路，就是硬件/软件接口。

2、在我们设计的流水线中，DM 处于 CPU 内部，请你考虑现代计算机中它的位置应该在何处。

现代计算机中，无论是 IM 指令存储器还是 DM 数据存储器，以及所有设备，都应该位于 CPU 外部，因为 DM 相当于现代计算机中的外存，容量大可以保存大量数据但是访问慢。现代计算机通过总线来传输所有的数据和信号，达到沟通的目的。各种外设使用不同的操作方法，由 CPU 来直接控制不同的外设不切实际。同时，外设的数据传送速度比处理器速度慢得多，使用高速系统与慢速的外设直接连接是愚蠢的

3、BE 部件对所有的外设都是必要的吗

我认为不是，BE 部件是否必要要依据外设的功能决定，在我们的 CPU 中，要依据外部设备支持的指令来决定，倘若 DM 为外部设备，那么就需要 BE 部件来支持 LH,LHU,SB,SH,LB,LBU 等指令，但是在我们 p7 的设计中，外部设备 timer 被要求不支持除 LW,SW 之外的指令，那么此时的 BE 部件相对来说就不是必要的。一切接口都是为了更好地沟通硬件和软件，都是为了满足其需求所设定的。

4、请开发一个主程序以及定时器的 exception handler。整个系统完成如下功能：

定时器在主程序中被初始化为模式 0；

定时器倒计数至 0 产生中断；

handler 设置使能 Enable 为 1 从而再次启动定时器的计数器。2 及 3 被无限重复。

主程序在初始化时将定时器初始化为模式 0，设定初值寄存器的初值为某个值，如 100 或 1000。（注意，主程序可能需要涉及对 CP0.SR 的编程，推荐阅读过后文后再进行。）

```
ori $t4, $0, 64513
mtc0 $t4, $12
ori $t1, $0, 0
```

```
ori $t1, $t1, 0x7f00
ori $t2, $0, 0
ori $t2, $t2, 1000 #preset
sw $t2, 4($t1)  #preset 存入相关寄存器
ori $t2, $0, 0
ori $t2, $t2, 9
sw $t2, ($t1) #启动倒数
ori $t3, $0, 1
ori $t3, $0, 2
ori $t3, $0, 3
ori $t3, $0, 4
ori $t3, $0, 5
ori $t3, $0, 6
ori $t3, $0, 7
ori $t3, $0, 8
ori $t3, $0, 9
ori $t3, $0, 10
ori $t3, $0, 1
ori $t3, $0, 2
ori $t3, $0, 3
ori $t3, $0, 4
ori $t3, $0, 5
ori $t3, $0, 6
ori $t3, $0, 7
ori $t3, $0, 8
ori $t3, $0, 9
ori $t3, $0, 10
```

handler 中只需再次设置 timer 的使能便可以再次启动。

5、请查阅相关资料，说明鼠标和键盘的输入信号是如何被 CPU 知晓的？

键盘、鼠标这类的低速设备是通过中断请求的方式进行 IO 操作的。即当键盘上按下一个按键的时候，键盘会发出一个中断信号，中断信号经过中断控制器传到 CPU，然后 CPU 根据不同的中断号执行不同的中断响应程序，然后进行相应的 IO 操作，把按下的按键编码读到寄存器（或者鼠标的操作），最后放入内存中。