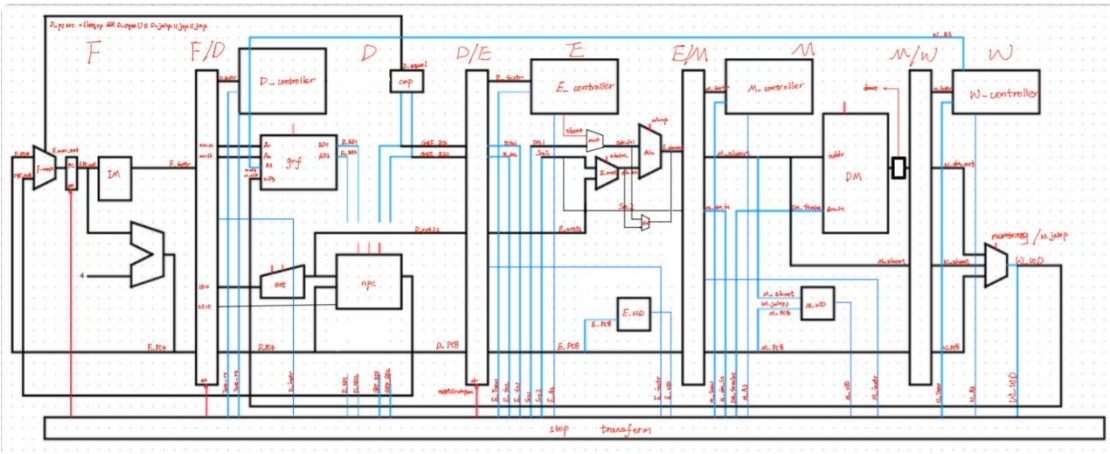
流水线 CPU 设计报告

一、数据通路设计

1、



指令			addu	subu	ori	lui	lw	sw	指令	beq	jal	jr	j	MUX	0	1	2
F级	pc		pc4	pc4	pc4	pc4	pc4	pc4	F级	pc4|npc	npc	npc	npc	pc_in	pc4@F	NPC	
	im		pc	pc	pc	pc	pc	pc		pc	pc	pc	pc	pc			
	pc4		pc	pc	pc	pc	pc	pc		pc	pc	pc	pc	pc			
F-D	D-Reg	IR@D	im	im	im	im	im	im	F-D	im	im	im	im	im			
		PC4@D	pc4	pc4	pc4	pc4	pc4	pc4		pc4	pc4	pc4	pc4	pc4			
D级	GPR	RA	IR@D[Rs]	IR@D[Rs]	IR@D[Rs]	IR@D[Rs]	IR@D[Rs]	IR@D[Rs]	D级	IR@D[Rs]		IR@D[Rs]		IR@D[Rs]			
		RB	IR@D[Rt]	IR@D[Rt]				IR@D[Rt]		IR@D[Rt]				IR@D[Rt]			
	EXT				IR@D[IMM]	IR@D[IMM]	IR@D[IMM]	IR@D[IMM]		IR@D[IMM]				IR@D[IMM]			
	NPC	pc4								PC4@D	PC4@D		PC4@D	PC4@D			
		jr_pc										GPR. RA		MFAD	GPR. RA	F1	F2
		imm								EXT				EXT			
		IR									IR@D		IR@D	IR@D			
	CMP	D1								IR@D[Rs]				MFAD			
		D2								IR@D[Rt]				MFBD			
	pc8										PC4@D			PC4@D			
D-E	E-Reg	IR@E	IR@D	IR@D	IR@D	IR@D	IR@D	IR@D	D-E	IR@D	IR@D	IR@D	IR@D	IR@D			
		PC8@E									pc8			pc8			
		RA@E	GPR. RA	GPR. RA	GPR. RA	GPR. RA	GPR. RA	GPR. RA						GPR. RA			
		RB@E	GPR. RB	GPR. RB				GPR. RB						GPR. RB			
		ext@E			EXT	EXT	EXT	EXT						EXT			
E级	ALU	A	RA@E	RA@E	RA@E	RA@E	RA@E	RA@E	E级					MFAE	RA@E	WD@W	AO@M
		B	RB@E	RB@E	ext@E	ext@E	ext@E	ext@E						ALUB	MFBE	ext@E	
			N/A							N/A				MFBE	RB@E	WD@W	AO@M
E-M	M-Reg	IR@M	IR@E	IR@E	IR@E	IR@E	IR@E	IR@E	E-M	IR@E	IR@E	IR@E	IR@E	IR@E			
		PC8@M									PC8@E			PC8@E			
		AO@M	ALU	ALU	ALU	ALU	ALU	ALU						ALU			
		WM@M						RB@E						RB@E			
M级	DM	add					AO@M	AO@M	M级					AO@M			
		WM						WM@M						MFBM	WM@M	WD@W	
M-W	W-Reg	IR@W	IR@E	IR@E	IR@E	IR@E	IR@E	IR@E	M-W	IR@E	IR@E	IR@E	IR@E	IR@E			
		PC8@W									PC8@M			PC8@M			
		AO@W	AO@E	AO@E	AO@E	AO@E								AO@E			
		DO@W					DM							DM			
W级	GPR	RW	IR@W[Rd]	IR@W[Rd]	IR@W[Rt]	IR@W[Rt]	IR@W[Rt]		W级		31			regwrite	IR@W[Rd]	IR@W[Rt]	31
		WD	AO@W	AO@W	AO@W	AO@W	DO@W				PC8@W			writeback	AO@W	DO@W	PC8@W

二、模块规格

1. IFU

（1）端口说明

端口名	方向	描述

CLK	I	时钟信号
RESET	I	异步复位信号
PC_in	I	原 PC 信号
PC_out	O	PC+4 后的信号
INSTRUCTION[31:0]	O	输出 32 位指令

（2）功能定义

序号	功能名称	功能描述
1	复位	当 RESET 信号为 1 时，对保存 PC 地址的寄存器进行异步清零操作。
2	取指	以 PC 的值为地址，从 IM 中取出对应指令

2. GRF

（1）端口说明

端口名	方向	描述
CLK	I	时钟信号
RESET	I	异步复位信号
RegWrite	I	写入使能
Read_Reg1	I	读取 Rs
Read_Reg2	I	读取 Rt
Write_Reg	I	读取存储地址
Write_Data	I	读取存储数据
Read_Data	O	Rs 输出
Read_Data2	O	Rt 输出

（2）功能定义

序号	功能名称	功能描述
1	复位	当 RESET 信号为 1 时，对 GRF 中所有寄存器进行

		异步清零操作。
2	写入	Regwrite=1 时，将数据写入
3	读取	将 Rs，Rt 输出

3、ALU

（1）端口说明

端口名	方向	描述
Input1[31:0]	I	输入 1
Input2[31:0]	I	输入 2
ALUOp[31:0]	I	选择计算功能
Equ	O	输入是否相等
Ans[31:0]	O	输出

（2）功能定义

序号	功能名称	功能描述
1	逻辑运算	ALUOp=00，C=A&B ALUOp=01，C=A|B ALUOp=10，C=A+B ALUOp=11，C=A-B
2	相等判断	若 A=B，则 Equ=1 若 A≠B，则 Equ=0

4、EXT

（1）端口说明

端口名	方向	描述
Imm16[15:0]	I	16 位立即数输入
ExtOp	I	Ext 功能选择
LuiOp	I	Lui 功能选择
Ext32	O	32 位数输出

（2）功能定义

序号	功能名称	功能描述
1	符号拓展	ExOp=0，无符号拓展 ExOp=1，有符号拓展
2	Lui 指令功能	LuiOp=1，将立即数添加至 32 位数高位，低位补零

5、DM

（1）端口说明

端口名	方向	描述
CLK	I	时钟信号
RESET	I	异步复位信号
Addr[5:0]	I	地址读入
Input[31:0]	I	数据读入
MemWrite	I	写入使能
MemRead	I	读取使能
Data[31:0]	O	32 位数据输出

（2）功能定义

序号	功能名称	功能描述
1	写入	当写入使能=1 时，将 Input 写入 Addr 对应的寄存器
2	读取	当读取使能=1 时，将 Addr 对应的寄存器的值输出

6、Controller

（1）端口说明

端口名	方向	描述
OP	I	Operation
FUN	I	Function
Regdst	O	GRF 写入地址 MUX 控制

		0：写入至 Rt 1：写入至 Rd
ALUsrc	O	ALU 输入 MUX 控制 0：input2 = Rt 1：input2 = ext32
MemtoReg	O	GRF 写入数据 MUX 控制 0：写入 DM 的输出 1：直接输出 ALU 结果
RegWrite	O	GRF 写入使能
MemWrite	O	DM 写入使能
MemRead	O	DM 输出使能
ExtOp	O	Ext 使能信号
LuiOp	O	Lui 使能信号
BeqOp	O	Beq 使能信号
ALUOp[1:0]	O	ALU 功能选择信号

（2）功能定义

序号	功能名称	功能描述
1	控制	根据 Op 和 Fun 输出相应指令与 MUX 的控制信号

7、Pipeline_Register

（1）端口说明

端口名	方向	描述
CLK	I	时钟信号
RESET	I	异步复位信号
En	I	写入使能
In	I	数据读入
Out	O	数据读出

8、Xlu

（1）端口说明

端口名	方向	描述
Xlu_in1	I	GRF[`rs] or [`shamt]
Xlu_in2	I	GRF[`rt]
Xlu_op	I	解析指令代码
Clk	I	时钟信号
Reset	I	异步复位信号
Start	O	开始信号
Busy	O	占用信号
Hi_out	O	Hi 寄存器输出
Lo_out	O	Lo 寄存器老师

（2）功能定义

序号	功能名称	功能描述
1	计算•存储	Mult 、 Multu 、 div 、 divu 、 mfhi 、mflo Mthi 、 Mtlo 指令实现

9、dmext

（1）端口说明

端口名	方向	描述
Dm_out	I	内存输出数据
Readdm_op	I	Readdm 控制
Addr	I	Dm 读入地址
M_dm_out	O	实际输出

（2）功能定义

序号	功能名称	功能描述
1	拓展 DM 数据	LW, LH , LHU, LB, LBU 指令实现

三、转发模块设计

1. 端口说明

端口名	方向
D_instr	I
E_instr	I
M_instr	I
E_Tnew	I
E_WD	I
M_WD	I
W_WD	I
E_A3	I
M_A3	I
W_A3	I
D_RD1	I
D_RD2	I
E_in1	I
E_in2	I
M_aluout	I
M_dm_in	I
GRF_RD1	O
GRF_RD2	O
Src1	O
Src2	O
DM_invalue	O

2. 算法逻辑

```
Assign GRF_RD1 =
(D_instr[`rs] == E_A3 && E_Tnew == 0 && D_instr[`rs] != 0) ? E_WD :
(D_instr[`rs] == M_A3 && M_Tnew == 0 && D_instr[`rs] != 0) ? M_WD :
(D_instr[`rs] == W_A3 && W_Tnew == 0 && D_instr[`rs] != 0) ? W_WD :
```

D_RD1;

如对于 GRF_RD1 的转发，通过判断 E\M\W 级指令对应的写入寄存器地址是否与 D 级需求的寄存器地址相同且不为 0，并判断 E\M\W 级的 Tnew 是否为 0，即结果是否已经传入流水线寄存器，来确定转发的数据与接口。

四、暂停模块设计

1. 端口说明

端口名	方向
Tuse_rs	I
Tuse_rt	I
E_Tnew	I
M_Tnew	I
W_Tnew	I
Stop_en	O

2. 算法逻辑

```
assign stop_en = ((Tuse_rs < E_Tnew)&&(D_instr[`rs] == E_A3)&&(D_instr[`rs] != 0)) ||
                 ((Tuse_rs < M_Tnew)&&(D_instr[`rs] == M_A3)&&(D_instr[`rs] != 0)) ||
                 ((Tuse_rs < W_Tnew)&&(D_instr[`rs] == W_A3)&&(D_instr[`rs] != 0)) ||
                  ((Tuse_rt < E_Tnew)&&(D_instr[`rt] == E_A3)&&(D_instr[`rt] != 0)) ||
                 ((Tuse_rt < M_Tnew)&&(D_instr[`rt] == M_A3)&&(D_instr[`rt] != 0)) ||
                 ((Tuse_rt < W_Tnew)&&(D_instr[`rt] == W_A3)&&(D_instr[`rt] != 0)) ||
                                  ((busy == 1 || start == 1) && (xluop == 1));
```

直接根据 E\M\W 级的 Tnew 与 D 级的 Tuse 进行大小比较，如果 Tuse<Tnew，则立刻暂停。

四、Tuse、Tnew 表格

	Tuse/D级开始		注：没有使用的直接记为3，初始化同 / 一段一段数 add：D到E ->1
	rs/offset	rt	指令描述
addu	1	1	GPR[rd] = GPR[rs] + GPR[rt]
subu	1	1	GPR[rd] = GPR[rs] - GPR[rt]
ori	1	3	GPR[rt] = GPR[rs] OR zero_extend(immediate)
lw	1	3	GPR[rt] = memory[GPR[rs]+offset]
sw	1	2	memory[Addr] = GPR[rs] + sign_ext(offset)
lui	3	3	GPR[rt] = immediate 16 || {16{1`b0}}
j	3	3	PC = PC31..28 || instr_in || 00
jal	3	3	GPR[31] = PC + 4
jr	0	3	PC = GPR[rs]
beq	0	0	(GPR[rs] == GPR[rt])
注：注意！只考虑使用端，不考虑生成段,eg lw->rt = 3 . Base -> 隐藏rs、rt			

Tnew/一级一级数	从X级前的流水线写入WD，则X=0,倒推	无用=0		
	部件	E	M	W
`addu`subu`add`sub`addi`subi	ALU	1	0	0
`ori`xor`andi`addiu`addi`nor	ALU	1	0	0
sll`srl`sra`sllv`srlv`srav`lui	ALU	1	0	0
`slt`slti`sltiu`sltu`mflo`mfhi	ALU`XLU	1	0	0
`mult`multu`div`divu	XLU	1	0	0
`sw`sb`sh`sw	DM	0	0	0
`lb`lbu`lh`lhu`lw	DM	2	1	0
`j`jal`jr`jalr	PC	0	0	0
`beq`bne`blez`bgtz`bltz`bgez	PC	0	0	0
`mtlo`mthi	XLU	0	0	0

五、CP0 设计，桥与 IO 设计

1. CP0

（1）端口说明

端口名	方向	描述
A1	I	CP0 读出地址
A2	I	CP0 写入地址
Din	I	读入数据
PC	I	PC 值
ExcCode	I	异常编码
HW	I	外部中断编码
We	I	使能信号
Exlset	I	Exl 置位信号
Exlclr	I	Exl 清零信号
clk	I	时钟信号
reset	I	清零信号

BD	I	延迟槽判断信号
Intreq	O	中断异常处理信号
EPC_out	O	返回地址输出
D_out	O	数据输出

2.Bridge

（1）端口说明

端口名	方向	描述
path_addr	I	外部设备写入地址
path_data	I	外部设备写入值
path_we	I	外部设备写入使能
timer0_dout	I	Timer0 输出
timer1_dout	I	Timer1 输出
IRQ0	I	Timer0 中断信号
IRQ1	I	Timer1 中断信号
Interrupt	I	外部中断信号
HW	O	中断信号编码
Pr_RD	O	数据写入
timer_addr	O	Timer 写入地址
timer_din	O	Timer 值读入
timer0_we	O	Timer0 写使能
timer1_we	O	Timer1 写使能

3.Timer

详见 Timer 使用说明文档。

六、测试程序

1.R-R

```
subu $1,$2,$3
addu $4,$1,$2


subu $1,$2,$3
addu $4,$2,$1


ori $1,5
addu $4,$2,$1


ori $2,5
addu $4,$2,$1



jal loop
ori $1,$0,1
loop:
ori $2,$0,10


ori $1,$0,122
jal loop1
addu $2,$31,$1
loop1:
ori $3,$0,239
jal loop2
addu $4,$3,$31
loop2:
jal loop3
ori $5,$0,23
loop3:
```

```
addu $6,$31,$5
jal loop4
ori $7,$0,55
loop4:
addu $8,$7,$31
```

2.R-LW,SW

```
ori $3,$0,4
ori $4,$0,8
sw $3,0($4)
lw $1,0($4)
subu $4,$2,$1

ori $8,$0,0x021a
ori $2,$0,0x1234
sw $8,0($0)
lw $1,0($0)
addu $3,$1,$2
ori $5,$0,0x9287
lw $4,0($0)
addu $6,$5,$4
lw $7,0($0)
addu $9,$7,$8
ori $11,$0,0x091a
```

3.R-Beq-Jal-Jr-LW-SW

```
beq  $a0, $t0, out0
beq  $a0, $t1, out1
subu $sp, $sp, $t2
sw   $a0, 0($sp)
sw   $ra, 4($sp)
```

```
subu $a0, $a0, $t1
jal  fiber
lw   $a0, 0($sp)
lw   $ra, 4($sp)
addu $sp, $sp, $t2
subu $sp, $sp, $t2
sw   $a0, 0($sp)
sw   $ra, 4($sp)
subu $a0, $a0, $t3
jal  fiber
lw   $a0, 0($sp)
lw   $ra, 4($sp)
addu $sp, $sp, $t2
jr   $ra
out1:
addu $s0, $s0, $t1
jr   $ra
out0:
jr   $ra
end:
addu $8, $s0, $0
```

4.Jr-jal,Jr-R

```
jal loop1
ori $2,$0,1
ori $3,$0,2
j exit1
ori $4,$0,3
loop1:
jr $31
```

```
ori $5,$0,2
exit1:
jal loop2
ori $6,$0,6
ori $7,$0,7
j exit2
ori $8,$0,8
loop2:
ori $9,$0,9
jr $31
ori $10,$0,10
exit2:
jal loop3
ori $11,$0,11
ori $12,$0,12
j exit3
ori $13,$0,13
loop3:
ori $14,$0,14
ori $15,$0,15
jr $31
ori $16,$0,16
exit3:
```

5.全部基础代码测试

```
xor $4, $1, $3
nor $3, $1, $2
sltu $3, $1, $2
sltu $3, $2, $1
lui $7, 312
```

```
ori $7, 123
sltu $3, $7, $2
slt $3, $7, $2
sll $3, $1, 2
srl $3, $1, 2
sra $3, $7, 2
sllv $3, $1, $2
srlv $3, $1, $2
srav $3, $7, $2
addi $3, $1, 1
addiu $3, $1, 1
andi $3, $1, 9
ori $3, $1, 9
xori $3, $1, 9
slti $3, $7, 1
sltiu $3, $7, 1
sb $7, 0($2)
sb $7, 1($2)
sb $7, 2($2)
sb $7, 3($2)
sh $7, 0($1)
sh $7, 2($1)
sw $7, 4($1)
lb $3, 0($1)
lbu $3, 0($1)
lh $3, 0($1)
lhu $3, 0($1)
lw $3, 0($1)
j jump1
nop
```

```
n3:
jalr $3, $31
nop
jump1:
lui $1, 0
lui $2, 0
ori $1, 123
ori $2, 234
mult $1, $7
mfhi $3
mflo $3
multu $1, $2
mfhi $3
mflo $3
div $1, $7
mfhi $3
mflo $3
divu $1, $2
mfhi $3
mflo $3
mthi $1
mfhi $3
mtlo $2
mflo $3
ori $1, 1
bne $1, $2,btype1
nop
ori $10, 123
btype1:
blez $2, btype2
```

```
nop
ori $10, 123
btype2:
bltz $7, btype3
nop
ori $10, 123
btype3:
bgez $2, btype4
nop
ori $10, 123
btype4:
bgtz $1, btype5
nop
ori $10, 123
btype5:
ori $11,123
ori $12,123
```

6. 异常中断测试 handler

```
.ktext 0x4180
_entry:
    mfc0    $k0, $14
    mfc0    $k1, $13
    ori $k0, $0, 0x1000
    sw  $sp, -4($k0)

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

    j   _save_context
    nop

_main_handler:
    mfc0    $k0, $13
    ori     $k1, $0, 0x007c
    and $k0, $k1, $k0
    beq     $0, $k0, _restore_context
    nop
    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
```

```
1   _main_handler:
2       mfc0    $k0, $13
3       ori     $k1, $0, 0x007c
4       and $k0, $k1, $k0
5       beq     $0, $k0, _restore_context
6       nop
7       mfc0    $k0, $14
8       addu    $k0, $k0, 4
9       mtc0    $k0, $14
10      j   _restore_context
11      nop
12  _restore_context:
13      lw  $1, 4($sp)
14          lw      $2, 8($sp)
15          lw      $3, 12($sp)
16          lw      $4, 16($sp)
17          lw      $5, 20($sp)
18          lw      $6, 24($sp)
19          lw      $7, 28($sp)
20          lw      $8, 32($sp)
21          lw      $9, 36($sp)
22          lw      $10, 40($sp)
23          lw      $11, 44($sp)
24          lw      $12, 48($sp)
25          lw      $13, 52($sp)
26          lw      $14, 56($sp)
27          lw      $15, 60($sp)
28          lw      $16, 64($sp)
29          lw      $17, 68($sp)
30          lw      $18, 72($sp)
31          lw      $19, 76($sp)
32          lw      $20, 80($sp)
33          lw      $21, 84($sp)
34          lw      $22, 88($sp)
35          lw      $23, 92($sp)
36          lw      $24, 96($sp)
37          lw      $25, 100($sp)
38          lw      $26, 104($sp)
39          lw      $27, 108($sp)
40          lw      $28, 112($sp)
41          lw      $29, 116($sp)
42          lw      $30, 120($sp)
43          lw      $31, 124($sp)
44      lw  $k0, 128($sp)
45      mthi    $k0
46      lw  $k0, 132($sp)
47      mtlo    $k0
48          j   _restore
49      nop
50  .text
51      ori     $2, $0, 0x1001
52      mtc0    $2, $12
53      ori $28, $0, 0x0000
54      ori $29, $0, 0x0000
55      lui $8, 0x7fff
56      lui $9, 0x7fff
57      add $10, $8, $9
58      or  $10, $8, $9
59
60  end:
61      beq $0, $0, end
62      nop
```

异常中断测试代码

li $2, 0xffff1234

mfc0 $2, $14

sw $2, 0($0)

lw $3, 0($0)

mtc0 $3, $14

```
mfc0 $3, $14
addu $3, $3, $3
jal label1
mtc0 $31, $14
label1:
mfc0 $3, $14

.text
    ori $28, $0, 0x0000
    ori $29, $0, 0x0f00
    mtc0    $0, $12
main:
    lui      $s0,0x8000
    lui      $s1,0x7fff
    ori      $s1,$s1,0xffff
    add     $10,$s0,$s0
    sub     $10,$s0,$s1
    addi    $10,$s1,10
    sw      $10,0x1002($0)
    sh      $10,0x1001($0)
    mult    $10,$10
    lw      $10,0x1002($0)
    lh      $10,0x1001($0)
    mult    $10,$10
    lhu      $10,0x1001($0)
    sub     $10,$s0,$s1
    addi    $10,$s1,10
    sw      $10,0x1002($0)
    sh      $10,0x1001($0)
    mult    $10,$10
```

```
    sw      $10,0x1002($0)
    sh      $10,0x1001($0)
    lw      $10,0x1002($0)
    lh      $10,0x1001($0)
    mult    $10,$10
    sh      $10,0x1001($0)
    add     $10,$s0,$s0
    sub     $10,$s0,$s1
    mult    $10,$10
    add     $10,$s0,$s0
    sub     $10,$s0,$s1
    j label_1
    add     $10,$s0,$s0
    sub     $10,$s0,$s1
label_1:
    mult    $10,$10
    add     $10,$s0,$s0
    sub     $10,$s0,$s1
    mult    $10,$10
    sh      $10,0x1001($0)
    lw      $10,0x1002($0)
    add     $10,$s0,$s0
    sh      $10,0x1001($0)
label_2:
    sub     $10,$s0,$s1
    mult    $10,$10
    sh      $10,0x1001($0)
    lw      $10,0x1002($0)
    nop
```

dead_loop:

    j   dead_loop

    nop

8. IO 测试

```
.text
li $t1, 0xFC01 # enable interrupt, disable exl
mtc0 $t1, $12
nop
nop
nop
li $t1, 50
sw $t1, 0x7F04
li $t2, 11
sw $t2, 0x7F00
nop
nop
loop:
lw $t3, 0x7F08
j loop
nop


.ktext 0x00004180

sw $1, 0x1000
mfc0 $k0, $13
andi $k0, $k0, 124
beqz $k0, continue
mfc0 $k0, $14
li $k1, 4
addu $k0, $k1, $k0
li $k1, 0xFFFFFFFC
and $k0, $k0, $k1
continue:
mtc0 $k0, $14
lw $1, 0x1000
eret
li $t1, 0xDEAD
```

七、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”接口字样，那么到底什么是“硬件/软件接口”？

答：硬件接口（hardware interface）指的是两个硬件设备之间的连接方式。硬件接口既包括物理上的接口，还包括逻辑上的数据传送协议。

软件接口也被称作“用户界面”,也就是“UI”，作为软件不同部分之间的交互接口。通常就是所谓的 API——应用程序编程接口，其表现的形式是源代码。

而硬件/软件接口则是指通过我们设计的 CPU 实现硬件与软件相互传递信息，数据

的连接口。

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。

答：脱离于 CPU 之外的独立主板上。

3. BE 部件对所有的外设都是必要的吗?

答：否，例如 timer，均是按字读取，不需要解析地址。

4. 请阅读官方提供的定时器源代码，阐述两种中断模式的异同，并分别针对每一种模式绘制状态转移图

答：详细见 Timer 使用设计文档

5. 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:

① 定时器在主程序中被初始化为模式 0;

② 定时器倒计数至 0 产生中断;

③ 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。

④ 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。)

答:

```
.text
li $1,0x7f00
li $2,0x7f04
li $5,0x0013
li $6,0x0011
li $7,0xfff1
mtc0 $7,$12
sw $6,0($2)
sw $5,0($1)
```

```
ori $3,$0,0x7f10
ori $4,$0,0x7f14

.ktext 0x00004180
li $1,0x7f00
li $t1,1
sw  $t1,0($1)
nop
nop
eret
```

6.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?

答：键盘、鼠标通过请求中断的方式对 CPU 进行 IO 操作。

例如，当按下键盘或鼠标时，他们会向 CPU 发送中断请求信号，中断信号经过中断控制器传到 CPU，然后在 CPU 处进行解析，不同的中断编码对应不同的中断操作，即对应唯一且一一对应的 IO 操作，最终将按键的信息存入内存中。